Pedro Kantek

Algoritmos

Unicenp
Jul/07, dez/07
Curitiba

Sumário

Capítulo 1

Contrato Pedagógico

Regras da disciplina

Uma aula dupla tem 100 minutos, que serão – em princípio – assim divididos: 10 minutos para os bons dias, 30 minutos de teoria, 30 de apresentação do exercício prático e mais 30 minutos para os alunos fazerem o seu exercício.

Cada aluno vai receber um exercício individual e único.

O exercício deverá ser devolvido feito até a data escrita na própria folha do exercício. Se entregue atrasado pagará 50% da nota de multa. Em outras palavras, a nota do exercício será dividida por 2.

Não haverá segunda chamada de exercícios perdidos.

Não serão aceitos xerox, fax ou e-mail de exercícios. Apenas o exercício original que contiver o nome do aluno será aceito. Se o aluno quiser (alguns querem), tire cópia xerox do exercício para guardar a coleção deles. Entretanto, todo exercício entregue ao professor será corrigido e devolvido ao aluno.

A nota bimestral será obtida fazendo a média aritmética dos exercícios do bimestre, com peso de 60% e a prova bimestral, esta com peso 40%.

As provas também são individuais e diferentes para cada aluno.

Exercícios Individuais

Cada exercício tem um identificador único, chamado seqüência com as seguintes informações: **aaLGM101**, que deve ser assim interpretada: [aa] Ano atual;[LG] disciplina de Algoritmos; [M ou N] Turno;[1] 1^o bimestre do ano letivo [01] Primeiro exercício deste bimestre;

Algoritmo da raiz quadrada

código: 3
finalizado em 12/11/07
Implementa o algoritmo (antigamente aprendido no inicio do ciclo fundamental, hoje não mais) que permite calcular à mão a raiz quadrada. É um processo razoavelmente complexo e ele permite vivenciar exatamente o que um algoritmo faz.

Introdução a algoritmos

código: 4
finalizado em 20/12/05

Apresenta os conceitos de divisão inteira e resto e a seguir pede para analisar o algoritmo de mdc (máximo divisor comum) devido a Euclides (aprox 400aC). Além disso, o exercício pede um algoritmo de uma tarefa trivial do dia a dia do aluno.

Jogo da vida

código: 6
finalizado em 23/10/07
Um exercicio simples, no qual o Jogo da Vida (Conway, 1970) é apresentado e são feitos comentários sobre a sua importância na Ciência da Computação. As 4 regras do jogo são apresentadas e depois é dado um tabuleiro 10×10. Pede-se ao aluno que simule 4 gerações e informe depois como ficaram 4 células do tabuleiro especialmente escolhidas. O objetivo do exercício é divertir e depois trabalhar ainda informalmente os conceitos de: matriz, algoritmo, iteração, entre outros.

GPS: Global Positioning System

código: 7
finalizado em 02/01/06
Este exercício começa descrevendo o acidente do avião VARIG 254 que se perdeu no Mato Grosso achando estar no Pará. A seguir o princípio de funcionamento dos sistemas GPSs é mostrado. Uma simulação em 2D é pedida ao aluno, usando-se características da física e operações da geometria analítica.

Problemas introdutórios

código: 8
finalizado em 29/08/06
Apresenta 2 questões comuns sobre regras de nomes de variáveis e tipos de variável a usar em determinados algoritmos. Depois gera uma coleção de 10 exercícios matemáticos individuais sobre raciocínio. A resposta é sempre um número.

Achar um número em uma sequencia

código: 9
finalizado em 12/12/07
São mostradas 30 sequencias de 11 números cada. A lei de formação não está explicitada e deve ser descoberta pelo aluno. Um dos 11 números está faltando e o aluno precisa descobrir qual é. Os números buscados são somados de 3 em 3 para facilitar a correção. Este exercício nasceu de uma inspiração obtida olhando os exercícios da Olimpíada Brasileira de Informática.

Matemática e lógica básicas

código: 11
finalizado em 01/09/06
60 exercícios (30 de matemática e 30 de lógica) são apresentados ao aluno. Ele deve achar 10 respostas (cada grupo de 6 exercícios tem suas respostas somadas. Isto facilita a correção).

SEs compostos e encadeados

código: 17
finalizado em 28/01/06
6 exercícios (funções) nos quais 6 valores numéricos aleatórios são fornecidos no início. A seguir uma série de comandos alternativos (SE) tanto compostos como encadeados, usando os conectivos $\wedge$ (e), $\vee$ (ou) e $\sim$ (não). Depois pede-se o resultado da soma de 5 variáveis.

Pseudocódigo X Fluxograma

código: 18
finalizado em 05/05/07
8 trechos de programa são apresentados através de um fluxograma e através do pseudocódigo equivalente. Entretanto, em alguns desses 8 trechos ambos são equivalentes e em alguns outros trechos houve uma sutil alteração ou no fluxograma ou no pseudocódigo de maneira que ambos deixaram de ser equivalentes. O aluno deve investigar quais dos 8 trechos são efetivamente equivalentes.

Pseudocódigo X Diagramas de Nassi-Schneidermann

código: 18
finalizado em 05/05/07
8 trechos de programa são apresentados através de um diagrama de Nassi-Schneidermann e através do pseudocódigo equivalente. Entretanto, em alguns desses 8 trechos ambos são equivalentes e em alguns outros trechos houve uma sutil alteração ou no diagrama ou no pseudocódigo de maneira que ambos deixaram de ser equivalentes. O aluno deve investigar quais dos 8 trechos são efetivamente equivalentes.

Chines de funções simples

código: 21
finalizado em 03/04/06
São apresentadas 5 funções de mesma estrutura com até 3 níveis de IF's encadeados. Não há ainda estruturas de repetição. As funções inicializam as variáveis A, B e C no começo, e ao final, o exercício pergunta qual o valor de uma delas (que seguramente mudou de valor durante o processamento).

Dígitos verificadores

código: 24
finalizado em 04/02/01
Mostra-se a importância dos DVs e depois mostra-se os algoritmos de cálculo do CPF e do CNPJ. O aluno deve calcular os DVs de 2 CPFs e de 2 CNPJs.

Aritmética não decimal

código: 27
finalizado em 28/11/01
É mostrada uma generalização da aritmética para bases não decimais, variando entre base = 3 e base = 33, excetuando-se as bases 2, 10 e 16. Conversões, somas e subtrações nestas bases são pedidas.

Cálculo de calendários

código: 31
finalizado em 25/02/02
Mostra o algoritmo de Aloysius Lilius e Cristopher Clavius do século XVI para calcular o Domingo da Páscoa, do qual decorrem todos os feriados móveis de nosso calendário: Carnaval, Sexta da Paixão e Corpus Christi. Além de mostrar a distribuição de bissextos, o exercício mostra o cálculo de qualquer dia da semana. Finalmente, o exercício sugere um ano hipotético (entre 1600 e 2400) e pede que o aluno calcule o dia da semana de um dia qualquer e os 3 feriados móveis deste ano

Localização de um oleoduto

código: 35
finalizado em 27/12/02
9 poços de petróleo são dispostos sobre uma grade de 100m × 100m. Pede-se que o aluno calcule de que maneira ligá-los através de tubos de maneira a minimizar a quantidade de tubos usados. Exercício extraído do livro do Cormen, pág 155 da edição brasileira.

Kumon de algoritmos

código: 54
finalizado em 23/06/06
Esta folha é para treinar os comandos básicos de algoritmos (se, enquanto, repita e para). São 17 exercícios simples, de apenas um único comando cada um deles.

Geração de seqüências

código: 57
finalizado em 30/12/05
Neste exercício são apresentadas 8 funções que contém laços (loops). Na parte final de cada laço há a impressão de uma variável. O exercício mostra os primeiros 4 valores impressos e pede que o aluno ache o oitavo elemento impresso. Ótimo exercício para ser implementado.

Engenharia reversa de algoritmos

código: 62
finalizado em 31/01/06
Este exercício mostra 4 funções. Em cada uma há 3 variáveis, cujos valores são inicializados no começo. Mas, apenas 2 destes valores são mostrados ao aluno, cabendo a ele, descobrir qual a inicialização da terceira variável a partir do comportamento da função e dos resultados que ela gera ao ser executada.

Funções envolvendo se, enquanto e repita

código: 65
finalizado em 20/04/06
Este exercício gera 4 funções razoavelmente complexas, nas quais se misturam comandos se, enquanto e repita, além de um conjunto de cerca de 15 variáveis em cada função. O aluno deve seguir o fluxo e informar ao final qual o valor que será impresso. É um convite à implementação.

Chines de algoritmos II

código: 69
finalizado em 29/02/00
4 funções geradas aleatoriamente, mas que garantidamente executam em tempo finito são mostradas. Cada função recebe 5 valores mostrados. Pede-se que o aluno calcule e informe, ao final da execução de cada função, o valor da soma das 5 variáveis originais.

Genérico de construção de algoritmos

código: 70
finalizado em 02/01/06
Este exercício comporta uma base de dados de exercícios para uso em muitas aulas de algoritmos. Para cada uma das aulas, a base contém cerca de 10 exercícios distintos. Ao gerar as folhas para os alunos, cada um deles receberá 4 (5 ou 6, depende da aula) de exercícios distintos. Todos eles exigem implementação nas aulas de laboratório.

Achar o 10^o termo de uma seqüência

código: 73
finalizado em 04/06/06
Este exercício mostra 20 seqüências de 7 números cada. Cada uma é resultado de um ciclo *para* no qual estão envolvidas 1, 2 ou 3 variáveis, cujo valor é desconhecido. Ao aluno cabe:

- sugerir valores para x, y e z
- verificar se os valores sugeridos geram a seqüência dada
- corrigir – se necessário – e voltar a testar
- após o acerto, gerar e responder o 10^o termo.

Chines de algoritmos

código: 77
finalizado em 21/02/00
4 funções usuais na informática são mostradas: busca em cadeias, médias em um vetor numérico, ordenação por inserção e soma de colunas em uma matriz. Dados de entrada são fornecidos, junto com o código de cada função pedindo-se ao aluno que infira os resultados que serão produzidos.

Balance Line

código: 84
finalizado em 02/03/00
O algoritmo de balance line é mostrado e é discutida a sua importância para atualização em lote de grandes volumes de dados. O exercício oferece dois arquivos de 30 registros cada (o arquivo de cadastro antigo e o arquivo de movimentações), pedindo-se ao aluno que gere o novo cadastro atualizado bem como o relatório de incompatibilidades da atualização. Perguntas são feitas sobre o conteúdo dessas duas saídas

Indexação e indireção

código: 110
finalizado em 18/02/01
Dados 3 vetores de 16 valores numéricos cada e dadas 5 variáveis numéricas globais, o exercício pede que o aluno ache o resultado de 48 operações de indexação e de indireção.

Busca simples em tabela

código: 116
finalizado em 07/08/06
Este exercício descreve o mecanismo das tabelas, suas operações de inclusão e exclusao. Na parte de busca, apresenta e compara 3 técnicas (busca linear, busca linear ordenada e busca linear com sentinela). O aluno deve implementar os 3 algoritmos e inferir o desempenho deles nas 3 técnicas.

Manuseio de tabelas II

código: 119
finalizado em 10/08/06
São mostrados 5 algoritmos: inclusão em tabelas desordenadas, exclusão puxando os sobrantes, exclusão preenchendo com um indicador de exclusão, busca binária e inclusão em tabela ordenada. Por enquanto, a folha pede apenas o preenchimento de algumas complexidades. As folhas são todas iguais entre si.

Certificação de algoritmos básicos

código: 125
finalizado em 11/07/07
Neste exercício são apresentados 5 algoritmos básicos (um número é primo ?, soma dos divisores de um número, achar a potência de um número, validar uma data e alocar bolas de brinquedo). Os algoritmos podem ou não conter um pequeno erro. O aluno deve descobrir quais estão errados e quais estão certos.

Certificação de algoritmos de vetor

código: 125
finalizado em 11/07/07
Neste exercício são apresentados 5 algoritmos de vetor (achar o maior elemento, onde está uma determinada chave, quantas vezes um elemento existe no vetor, calcular a média, moda e mediana e responder se o vetor está em ordem). Os algoritmos podem ou não conter um pequeno erro. O aluno deve descobrir quais estão errados e quais estão certos.

Certificação de algoritmos de vetor

código: 125
finalizado em 11/07/07
Neste exercício são apresentados 5 algoritmos de vetor (ver se uma frase é palíndromo, incluir uma chave em um vetor, achar o número da maior palavra em uma frase, exclusão de todos os valores k no vetor V e achar a amplitude de um vetor). Os algoritmos podem ou não conter um pequeno erro. O aluno deve descobrir quais estão errados e quais estão certos.

Certificação de algoritmos de matriz

código: 125
finalizado em 11/07/07
Neste exercício são apresentados 5 algoritmos de matriz (achar o vetor soma na vertical, calcular uma média ad-hoc na matriz, achar o vetor menor na horizontal, calcular o produto de duas matrizes e dada uma matriz com dados eleitorais, tesponder quem ganhou a eleição). Os algoritmos podem ou não conter um pequeno erro. O aluno deve descobrir quais estão errados e quais estão certos.

Matrizes

código: 128
finalizado em 22/08/06
Apresenta 5 algoritmos de matrizes. Uma totalização nos dois sentidos, o cálculo da matriz transposta, uma matriz com elementos abaixo da diagonal principal zerados (um grafo não dirigido ?), depois a multiplicação de 2 matrizes e finalmente a solução de um sistema de equações lineares. O algoritmo é mostrado e um exemplo completo está descrito na folha. Finalmente, o aluno é convidado a resolver (a mão ou usando o computador, a escolha é dele) um sistema com 8 equações e 8 incógnitas.

Manipulações no cubo Rubik

código: 135
finalizado em 21/09/06
O cubo Rubik é apresentado e é sugerida uma estrutura de dados que suporte manipulações sobre ele. Depois uma lista de 9 operadores é também mostrada, aplicada à estrutura de dados recém definida. O aluno é convidado a aplicar dois operadores sobre 2 cubos aleatórios e depois ïdeve resolver uma instância trivial do cubo (apenas 2 movimentos).

Fazenda de ostras de Zing Zhu

código: 139
finalizado em 24/09/06
Este problema foi retirado da Maratona de Programação da ACM do ano de 2004. Ele apresenta uma quase-solução, já que resolve apenas algumas instâncias (e não outras). O objetivo do exercício é exatamente discutir a freqüência com o profissional encontra esta situação no mundo real, quando nem sempre há um algoritmo conhecido e correto. Existe uma ilha rasa e o seu proprietário estabelece uma série de cercas. Dada uma certa altura da maré, pergunta-se quanto de área da ilha permanece seco.

Números romanos

código: 144
finalizado em 16/09/06
Apresenta o sistema romano de numeração. Os algoritmos de conversão de arábico para romano e vice-versa são apresentados. Há 6 conversões em cada sentido para o aluno fazer. Deve-se alertar o fato de que na conversão de arábico → romano há várias respostas possíveis (todas corretas) devendo o aluno fazer aquela que é dada pelo algoritmo estudado. Isto atrapalha o uso de conversores livremente disponíveis para resolver o problema proposto.

Universidade Pinguinhos

código: 148
finalizado em 07/10/06
Problema retirado da Maratona de Programação de 2005 da ACM. Dá-se um curso universitário composto de cerca de 15 matérias com diversas relações de pré-requisitos. Estabelecida a prioridade das matérias, e a quantidade máxima de matérias por semestre, pede-se que o aluno monte a grade proposta.

Piscina/Mágico

código: 151
finalizado em 28/10/06
Dois exercícios retirados da maratona de Programação da ACM dos anos de 2005 e 2004 respectivamente. O primeiro, dá um estoque de azulejos de tamanho pequeno médio e grande respectivamente e pede a disposição de custo mínimo para azulejar uma piscina de dimensões dadas. O segundo descreve um truque de adivinhação de cartas e pede que o aluno resolva 3 instâncias desse truque.

Regata e Luzes da festa

código: 152
finalizado em 03/08/07
O primeiro exercício é da olimpíada ACM de 2005 e o segundo da IOI de 1998. A regata apresenta as coordenadas de um ponto de origem e de um ponto de destino, além de diversos obstáculos (dados pelas coordenadas de suas extremidades). O aluno deve calcular qual o menor caminho da origem ao destino. O exercício das luzes, sugere N lâmpadas inicialmente acesas, que podem ser controladas através de 4 botões (1=chaveia todas, 2=só as impares, 3=só as pares 4=1,4,7...) O exercício informa algumas lâmpadas que devem ficar acesas e outras apagadas, e pergunta como ficaram todas as N lâmpadas, após a aplicação dos botões de controle.

Jogo do Retângulo

código: 153
finalizado em 09/11/07
Este exercício extraído da Olimpíada Internacional de Informática de 2005, ocorrido na Polônia, oferece um retângulo de dimensões inteiras. Cada jogador pode fazer 1 corte (vertical ou horizontal) sempre em unidades inteiras. Feito o corte, o menor pedaço é desprezado e segue o jogo. Quem ficar com o retângulo 1×1 perde. São dados os método de solução e os algoritmos. Dadas 3 instâncias (2 vencedoras e 1 perdedora) pede-se que o aluno identifique a perdedora e ofereça o lance ganhador nas outras 2.

Caminho e Vulcão

código: 155
finalizado em 11/08/07
Dois exercícios baseados em similares pedidos na Olimpíada Espanhola de Informática. No primeiro, uma matriz de letras é estabelecida e pede-se o caminho que estabelece uma palavra dada. Note-se que a matriz tem as extremidades ligadas, o que torna mais complexa a busca. No segundo, um vulcão descrito pelas altitudes de seus platôs, derrama lava e o problema pede que o aluno descubra para onde vai a lava nos diversos instantes discretos de tempo

Genoma e Palavras cruzadas

código: 156
finalizado em 31/12/07
2 problemas baseados na prova da OBI de 1999. No primeiro problema são dadas duas cadeias de DNA e o aluno deve localizar as correspondências da cadeia original e de sua complementar. No segundo problema, um tabuleiro de palavras cruzadas de dimensão conhecida é dado e são dadas as localizações das casas pretas. O aluno deve numerar o tabuleiro e informar os endereços das palavras na horizontal e na vertical.

Trem/caminhão, robo e restaurante

código: 157
finalizado em 03/01/08
3 problemas baseados na prova da OBI. No primeiro problema são dadas informações sobre frete e o aluno deve escolher o transporte por trem ou por caminhão. No segundo o caminho de um robot é dado por suas coordenadas e pergunta-se quantas voltas (giros) sobre si mesmo o robot dá. No terceiro, dados de entrada e saida de pessoas em um restaurante são dados e pergunta-se: qual o maior número de pessoas que estiveram em um dado momento dentro do recinto ?

Seq.H, carga pesada e rede ótica

código: 158
finalizado em 04/01/08
3 problemas baseados na prova da OBI. No primeiro problema é definido um critério definidor de uma seqüência binária. 4 seqüências dão dadas e pergunta-se se elas atendem ou não ao critério (recursivo). No segundo um mapa de estradas com alturas máximas são dadas e pede-se qual a carga mais alta que pode ser levada de uma cidade a outra. Finalmente, no terceiro problema, da-se uma rede ótica com seus custos por ramo e pede-se o menor custo de ligação (árvore de cobertura mínima).

Quermesse, troca de notas e saldo de gols

código: 159
finalizado em 04/01/08
3 problemas baseados na prova da OBI. No primeiro problema pede-se quem ganhou um concurso (fácil). No segundo problema, usando notas de 50, 10, 5 e 1, deve-se informar quantas e quais notas compõe um valor (fácil). No terceiro, deve-se escolher qual o período em que um certo grande time teve o maior saldo de gols (não tão fácil).

Macaco prego e visita ao MASP

código: 160
finalizado em 08/01/08
2 problemas baseados na prova da OBI. No primeiro problema pede-se qual a delimitação de uma reserva florestal que congregue o maior número possível de macacos prego a partir dos dados parciais de suas aparições. No segundo, o roteiro de uma visita às obras do MASPdeve ser estabelecido minimizando o esforço e priorizando as obras mais importantes. Trata-se de um problema de minimização.

Anéis quadrados, Balaios e Meteoros

código: 161
finalizado em 08/01/08
3 problemas baseados na prova da OBI. No primeiro problema pede-se qual a configuração quando vários anéis quadrados são vistos de cima. O segundo pede determinar qual a seqüência de produção de balaios que minimiza a multa por atraso e o terceiro determina uma fazenda e diversos locais de queda de meteoritos, perguntando quantos caíram dentro da fazenda.

Dominó, sorvete e pirâmide de caixas

código: 162
finalizado em 09/01/08
3 problemas baseados na prova da OBI. No primeiro problema pergunta-se se um dado conjunto de pedras de dominó podem ou não formar um jogo. No segundo, dão-se diversos trajetos de sorveteiros e pergunta-se qual a parte da praia que ficará atendida por pelo menos um sorveteiro, e no terceiro, dão-se diversas caixas de diversas dimensões e pergunta-se qual a altura da maior pilha possível, desde que uma caixa nao ultrapasse nenhuma dimensão da caixa que está abaixo

Tesouro, Aeroporto e Pedágio na Coréia

código: 163
finalizado em 17/01/08
3 problemas baseados na prova da OBI. No primeiro problema dão-se 2 valores (de João e de José) e o conteúdo de uma arca do tesouro. A pergunta é se é possível ambos terem o mesmo valor. O segundo pergunta qual aeroporto ficará mais congestionado, e o terceiro pergunta quais cidades uma pessoa pode alcançar gastando no máximo P pedágios.

Finlândia, Supermercado e Número de Erdos

código: 164
finalizado em 18/01/08
3 problemas baseados na prova da OBI. No primeiro problema pede-se quantas cidades um ciclista pode visitar na Finlândia, desde que não tenha que subir nada. No segundo, pede-se a melhor localização para um centro de distribuição de uma rede de supermercados. No terceiro, dão-se diversos artigos com seus autores, e pede-se o número de Erdos de 3 deles.

Furos, Senha bancária e Orkut

código: 165
finalizado em 22/01/08
3 problemas baseados na prova da OBI. No primeiro problema pede-se qual o menor diâmetro de uma chapa redonda que cobre todos os furos cujos centros são dados. No segundo são dadas 2, 3 ou 4 digitações de senhas em um banco eletrônico, pedindo-se ao aluno que deduza a senha original correta e no terceiro pede-se que o aluno indique se é ou não é possível criar uma comunidade Orkut com as regras que lá são dadas.

Colheita de minhocas, furos e frota de taxis

código: 166
finalizado em 22/01/08
3 problemas baseados na prova da OBI. No primeiro problema pde-se determinar o valor da colheita usando a trajetória otimizada de uma máquina de colheita de minhocas. No segundo, definem-se os buracos de minhoca como ligações entre partes do universo e a partir do mapa delas, pergunta-se se todos os planetas desse universo podem ser acessados. Finalmente, no terceiro o aluno deve decidir se abastece uma frota com álcool ou com gasolina.

10 problemas a base de papel e lápis

código: 170
finalizado em 17/12/07
10 problemas baseados na prova da OBI de 2003. São da fase introdutória (para alunos de 2.grau) e portanto não exigem nenhum conhecimento especializado de programação. Apenas de solução de problemas.

5 problemas a base de papel e lápis-a

código: 171
finalizado em 18/12/07
5 problemas baseados na prova da OBI de 2004. São da fase introdutória (para alunos de 2.grau) e portanto não exigem nenhum conhecimento especializado de programação. Apenas de solução de problemas. São eles: supermercado(2), maionese(1) e mapas da prefeitura(2)

5 problemas a base de papel e lápis-k

código: 171
finalizado em 18/12/07
10 problemas baseados na prova da OBI de 2004. São da fase introdutória (para alunos de 2.grau) e portanto não exigem nenhum conhecimento especializado de programação. Apenas de solução de problemas. São eles: Atendimento ao consumidor(2), Clara e Luiz (1) e Economia do transporte(2).

10 problemas a base de papel e lápis

código: 172
finalizado em 19/12/07
10 problemas baseados na prova da OBI de 2005. São da fase introdutória (para alunos de 2.grau) e portanto não exigem nenhum conhecimento especializado de programação. Apenas de solução de problemas. São eles: Equipe de software (2), Flores (2), Pane seca (1) Linguagem de programação (2), Semana de provas (1) e Futebol(2).

7 problemas a base de papel e lápis-a

código: 173
finalizado em 22/12/07
7 problemas baseados na prova da OBI de 2006. São da fase introdutória (para alunos de 2.grau) e portanto não exigem nenhum conhecimento especializado de programação.

Apenas de solução de problemas. São eles: Excursão dos alunos(2), Conjunto de rock (2), azulejos (1) e transporte de alunos (2).

6 problemas a base de papel e lápis-b

código: 173
finalizado em 22/12/07
6 problemas baseados na prova da OBI de 2006. São da fase introdutória (para alunos de 2.grau) e portanto não exigem nenhum conhecimento especializado de programação. Apenas de solução de problemas. São eles: Pantanal (2), Torneio de tenis (2), hidrologia (1) e Tornado (1).

6 problemas a base de papel e lápis-c

código: 173
finalizado em 22/12/07
6 problemas baseados na prova da OBI de 2006. São da fase introdutória (para alunos de 2.grau) e portanto não exigem nenhum conhecimento especializado de programação. Apenas de solução de problemas. São eles: Ciclo de palestras (2), Analista (2), Casa mal afamada (1) e Notação pós-fixa (1).

7 problemas a base de papel e lápis-d

código: 173
finalizado em 22/12/07
7 problemas baseados na prova da OBI de 2006. São da fase introdutória (para alunos de 2.grau) e portanto não exigem nenhum conhecimento especializado de programação. Apenas de solução de problemas. São eles: Mapas a colorir (2), MP3 (3), Horta da Maria (1) e Sanduiche do João (1). Não estão neste material e serão aplicados diretamente em sala de aula.

Bibliografia para Algoritmos

FOR05 FORBELLONE, A. L. V. e EBERSPÄCHER, H. F. - Lógica de Programação.

FAR85 FARRER, Harry et all. Algoritmos estruturados. Rio de Janeiro, Guanabara, 1985. 241p. (Programação estruturada de computadores).

FOR72 FORSYTHE, Alexandra I. et all. Ciência de Computadores - 1. curso. Volume 1. Rio de Janeiro, Ao Livro Técnico. 1972. 234p.

FOR72b FORSYTHE, Alexandra I. et all. Ciência de Computadores - 1. curso. Volume 2. Rio de Janeiro, Ao Livro Técnico. 1972. 560p.

GUI75 GUIMARÃES, Ângelo de Moura & LAGES, Newton Alberto de Castilho. Algoritmos e estruturas de dados. Rio de Janeiro, LTC, 1975. 216p.

KNU71 KNUTH, Donald E. The Art of Computer Programming. Volumes 1 e 2. Reading, Massachussets, Addisonb Wesley Publishing Company, 1971, 624p.

SIL90 SILVA PINTO, Wilson. Introdução ao desenvolvimento de algoritmos e estruturas de dados. São Paulo, ÉRICA, 1990, 201p.

WIR89 WIRTH, Niklaus. Algoritmos e estruturas de dados. Rio de Janeiro, Prentice-Hall do Brasil, 1989, 245p.

O autor, pelo autor

Meu nome é Pedro Luis Kantek Garcia Navarro, conhecido como Kantek, ou Pedro Kantek. Nasci em Curitiba há mais de 50 anos. Sou portanto brasileiro, curitibano e coxa-branca com muito orgulho, mas sendo filho de espanhóis (meus 7 irmãos nasceram lá), tenho também a nacionalidade espanhola. Aprendi a falar em *castellano*, o português é portanto meu segundo idioma. Estudei no Colégio Bom Jesus e quando chegou a hora de escolher a profissão, lá por 1972, fui para a engenharia civil, mas sem muita convicção. Durante a copa do mundo de futebol de 1974 na Alemanha, ao folhear a Gazeta do Povo, achei um pequeno anúncio sobre um estágio na área de processamento de dados (os nomes informática e computação ainda não existiam). Lá fui eu para um estágio na CELEPAR, que hoje, mais de 30 anos após, ainda não acabou. Na CELEPAR já fui de tudo: programador, analista, suporte a BD (banco de dados), suporte a TP (teleprocessamento), coordenador de auto-serviço, coordenador de atendimento, ... Atualmente estou na área de governo eletrônico. Desde cedo encasquetei que uma boa maneira de me obrigar a continuar estudando a vida toda era virar professor. Comecei essa desafiante carreira em 1976, dando aula num lugar chamado UUTT, que não existe mais. Passei por FAE, PUC e cheguei às Faculdades Positivo em 1988. Sou o decano do curso de informática e um dos mais antigos professores da casa. Na década de 80, virei instrutor itinerante de uma empresa chamada CTIS de Brasília, e dei um monte de cursos por este Brasil afora (Manaus, Recife, Brasília, Rio, São Paulo, Fortaleza, Floripa, ...). Em 90, resolvi voltar a estudar e fui fazer o mestrado em informática industrial no CEFET. Ainda peguei a última leva dos professores franceses que iniciaram o curso. Em 93 virei mestre, e a minha dissertação foi publicada em livro pela editora Campus (Downsizing de sistemas de Informação. Rio de Janeiro: Campus, 1994. 240p, ISBN:85-7001-926-2). O primeiro cheque dos direitos autorais me manteve um mês em Nova Iorque, estudando inglês. Aliás, foi o quarto livro de minha carreira de escritor, antes já havia 3 outros (MS WORD - Guia do Usuário Brasileiro. Rio de Janeiro: Campus, 1987. 250p, ISBN:85-7001-507-0, Centro de Informações. Rio de Janeiro: Campus, 1985. 103p, ISBN:85-7001-383-3 e APL - Uma linguagem de programação. Curitiba. CELEPAR, 1982. 222p). Depois vieram outros. Terminando o mestrado, rapidamente para não perder o fôlego, engatei o doutorado em engenharia elétrica. Ele se iniciou em 1994 na UFSC em Florianópolis. Só terminou em 2000, foram 6 anos inesquecíveis, até porque nesse meio tive que aprender o francês - mais um mês em Paris aprendendo-o. Finalmente virei engenheiro, 25 anos depois de ter iniciado a engenharia civil. Esqueci de dizer que no meio do curso de Civil desistí (cá pra nós o assunto era meio chato...) em favor de algo muito mais emocionante: matemática. Nessa época ainda não havia cursos superiores de informática. Formei-me em matemática na PUC/Pr em 1981. Em 2003, habilitei-me a avaliador de cursos para o MEC. Para minha surpresa, fui selecionado e virei delegado do INEP (Instituto Nacional de Pesquisas Educacionais) do Governo Brasileiro. De novo, visitei lugares deste Brasilzão que sequer imaginava existirem (por exemplo, Rondonópolis, Luiziânia, Rio Grande, entre outros), sempre avaliando os cursos na área de informática: sistemas de informação, engenharia e ciência da computação. Atualmente estou licenciado da PUC e no UNICENP respondo por 4 cadeiras: Algoritmos (1. ano de sistemas de informação), Estrutura de Dados (2.ano, idem) e Tópicos Avançados em Sistemas de Informação (4.ano, idem), além de Inteligência Artificial (último ano de Engenharia da Computação). Já fiz um bocado de coisas na vida, mas acho que um dos meus sonhos é um dia ser professor de matemática para crianças: tentar despertá-las para este mundo fantástico, do qual – lastimavelmente – boa parte delas nunca chega sequer perto ao longo de sua vida.

O autor, há muitos anos, quando ainda tinha abundante cabeleira

Capítulo 2

Ciência da Computação

> Programas de qualquer tamanho estão sujeitos a conterem "bugs"(erros). Alguns deles acabam custando caro.
>
> Em 22/jul/62 um foguete foi lançado com a missão de chegar a Venus. Esta foi a primeira tentativa de explorar qualquer planeta. O foguete desviou-se do curso e foi destruido.
>
> Investigações posteriores revelaram que o responsável pelo problema foi um erro de programação dos computadores. Apesar de testes e depurações extensas, a omissão de um simples hífen num dos comandos de um programa não foi detectada. Este erro causou a falha da missão inteira a um custo de US$ 18,5 milhões. Trecho do livro "Software Testing Techniques"de Boris Beizer.
>
> Se o construtores construíssem edifícios da mesma forma que os programadores escrevem programas, o primeiro pica-pau poderia destruir a civilização. Gerald Weinberg, citado por James Martin, em "O manifesto".
>
> Se alguma coisa pode dar errada, dará errada. Lei de Murphy

2.1 Representação do conhecimento

Se olharmos a história deste nosso pequeno planeta e abstrairmos a hipótese Teológica (já que cientificamente ela não tem comprovação) veremos uma história mais ou menos assim:

1. Há uns 3.000.000.000 de anos, surgem os primeiros seres vivos, capazes de – entre outras coisas – se reproduzirem por cópia de si mesmos. Aparece aqui o DNA (ácido desoxi-ribonucleico) o elemento comunicador da herança. Foi o primeiro e até hoje bem importante local onde se armazenou o conhecimento.

2. Mais tarde, surge o sexo. Ele tem enorme importância nesta história ao permitir que os filhos herdem um poco do DNA do pai e um pouco da mãe, acelerando a evolução.

3. Um bocado de tempo depois, surgem os animais superiores e nestes o cérebro passa a ser responsável pelo local onde o conhecimento é produzido e armazenado. Denomina-se esta fase à do conhecimento extragenético.

4. Enquanto limitado ao conhecimento localmente produzido, o cérebro pode armazenar pouco. O próximo estágio de ampliação é quando surge a fala. Agora o conteúdo de um cérebro pode ser repassado a outro, com resultados melhores e crescentes. O nome é conhecimento extrassomático

5. Com a fala, os dois sujeitos tem que estar tête-à-tête. A seguir, surge a escrita. Agora a comunicação é possível na modalidade um-muitos (Antes era um-um) e mesmo um cérebro morto pode se comunicar pela escrita.

6. Enquanto a escrita é manuscrita, está naturalmente limitada. A próxima novidade é a imprensa que ao automatizar a produção de textos, amplia significativamente o acesso e a disseminação de informações.

7. Finalmente, o último estágio da ampliação do conhecimento reside no software. Hoje, 23 milhões de brasileiros sabem declarar seu IR, sem precisar fazer um exaustivo aprendizado sobre as regras da declaração, graças a um... software. O nome é inteligência extrassomática.

Durante o século XX a maior indústria foi a do petróleo. Depois, a do automóvel e finalmente no final do século, a indústria de computadores passou a ser a maior. A tendência para o futuro é de que ela suplante todas as demais. Hoje o homem mais rico do mundo construiu a sua fortuna vendendo um software.

O computador nasce na cabeça de um cientista inglês (Alan Matheson Turing em 1937) e é construído para a guerra pelos ingleses em 1942. A seqüência de escopo do computador é

guerra a idéia aqui é usar o computador como otimizador dos recursos bélicos, criptografia, pesquisa operacional, tabelas balísticas...→

aritmética o computador se transforma em super-calculadora. A matemática passa a ser a linguagem da informática, por excelência. O FORTRAN é dessa época... →

negócios em plena sociedade capitalista, os negócios passam a ser otimizados pelo computador. Dessa época é o COBOL a linguagem representante... →

aldeia global Negócios, governos, universidades e organizações sociais passam a compartilhar o ciberespaço. Surge e se robustece a Internet... →

lazer o computador vira um eletrodoméstico. As casas de classe média passam a contar com um (ou mais de um) computador. Jogos, música, filmes, e-mule, mp3, ... →

... →

realidade virtual Não se sabe direito o que virá por aqui. As técnicas de simulação levadas ao paroxismo permitirão criar realidades virtuais, com toda a contradição exposta no título. Talvez retornemos à dúvida de Platão (*o universo existe ou é apenas uma sensação que eu tenho?*).

O computador se diferencia de qualquer outra máquina pela sua generalidade. No texto original de Turing isto fica claro pela conceituação de uma máquina universal. Assim, diferentemente de um liquidificador (que só serve para picar comida e tem apenas um botão de liga-desliga), um computador precisa – antes de qualquer coisa – ser programado para fazer algo. De fato, se comprar um computador vazio e ligá-lo, ele não vai fazer nada.

Tecnicamente, chama-se a parte física do computador de **hardware** significando ferragem, duro (hard), componente físico, etc. Já a programação recebe o nome de **software** significando mole (soft em oposição a hard), componente lógico, não físico, programa de computador.

A fabricação de hardware é engenharia. Sua produção pode ser planejada e otimizada e isso de fato é feito. A fabricação de software, a despeito de ser chamada de engenharia, cá entre nós, não o é muito. Está mais para arte do que para engenharia.

Talvez, como seres humanos, devamos dar graças aos céus por esta situação. Pois enquanto a produção de hardware é (ou pode ser) em grande parte robotizada, a produção de software ainda exige um bom cérebro cinzento por trás.

As restrições que ainda existem sobre o software e que talvez nunca deixem de existir, impedem a programação dos computadores em linguagem natural. Hoje ainda não é possível dialogar com um computador como se fala com uma pessoa medianamente inteligente. Os computadores ainda não conseguem tal proeza.

Para programar um computador necessitamos uma linguagem. O hardware só entende uma linguagem de pulsos elétricos (chamada de linguagem de máquina). Para facilitar a nossa vida (já que é muito dificil para o homem programar nela), criaram-se programas tradutores que recebem comandos e dados em uma linguagem artificial e a convertem em linguagem de máquina. A maioria desses tradutores recebeu o mesmo nome da linguagem que eles traduzem. Assim, temos o Java, o Cobol, Pascal, Apl, Clipper, Dbase, Basic, Natural, Fortran, C, Php, Lisp, Prolog, Modula, entre outras. Veja no sítio www.tiobe.com o índice TPC que mensalmente lista as 100 linguagens mais usadas no mundo.

Quando os primeiros computadores foram construidos, a única linguagem por eles entendida era a binária. Seqüências intermináveis de uns e zeros, algo assim como 0 0 1 0 1 0 0 0 1 0 1 0 1 0 1 1 1 0 1 0 1 1 0 0 1 0 1 0 1 0 0 1 0 0 0 1 0 1 0 1. Era um autêntico samba do crioulo doido. Chamou-se a isto, mais tarde, linguagem de primeira geração.

A seguir, o primeiro melhoramento: o assembler (ou montador), programa que era capaz de entender instruções escritas em uma linguagem um pouco mais humana, e traduzir cada uma destas instruções em sua expressão binária equivalente. Junto com este programa criou-se a primeira linguagem digna deste nome: o assembler, até hoje usado, se bem que muito raramente.

Programar em assembler ainda exige muito talento e paciência. A máquina precisa ser conhecida em inúmeros detalhes de construção, os programas são longos e difíceis (principalmente sua manutenção). Entretanto esta é a linguagem mais eficiente do ponto de vista dos consumo dos recursos da máquina, isto é, ela gera programas velozes e pequenos.

Depois, vieram as linguagens de terceira geração: a primeira foi o FORTRAN, depois COBOL, PL/I, PASCAL, C, C++, Java, LISP, Prolog e outras menos votadas: BASIC, ADA, APL. Há um distanciamento do programador em relação à máquina, e uma aproximação do mesmo em relação ao problema a resolver. Estas linguagens são mais fáceis de aprender e embora gerem programas maiores e mais lentos, aumentam em muito a produtividade humana de escrever código de programação.

Um exemplo diferente para mostrar todos estes conceitos de linguagens poderia ser o seguinte: Suponha um engenheiro brasileiro, que só conhece o idioma português. Ele precisa transmitir certas ordens de fabricação para um colega soviético, que só conhece o idioma russo, com escrita cirílica. Os dois podem tentar conversar quanto quiserem, mas provavelmente não haver nenhum resultado prático útil. É necessária a presença de um tradutor, alguém que ouvindo o português consiga transformar as ordens em suas equivalentes russas. Finalmente, o engenheiro brasileiro verá se as ordens foram corretamente traduzidas e executadas analisando o resultado final do trabalho.

Neste exemplo, o engenheiro brasileiro é o programador. O colega soviético é a máquina (que ao invés de russo, só entende a linguagem elétrica). O tradutor é a linguagem de programação.

Podemos ainda estabelecer dois níveis para linguagens de programação: baixo e alto. Linguagem de baixo nível é aquela que requer a especificação completa do problema nos seus mínimos detalhes. No nosso exemplo, equivaleria ao engenheiro brasileiro descrevendo tin-tin por tin-tin todos os passos para montar uma engenhoca qualquer. Já as linguagens de alto nível, pressupõe uma tradução "inteligente", o que libera o engen-

heiro brasileiro de informar todas as etapas. Ele diz o que deve ser feito, sem precisar estabelecer como isto vai ser realizado.

Todas as linguagens compartilham uma estrutura parecida, já que rodam todas no mesmo computador que é construído usando a chamada Arquitetura de Von Neumann. Bem resumido, essa arquitetura prevê a existência da memória, de uma unidade aritmético lógica, canais e o elemento ativo: a unidade de controle. Essa estrutura comporta os seguintes tipos de comandos (ordens): entrada-saída de dados, aritmética, movimentação em memória, condicional e desvio.

Com apenas essas 5 ordens, são construídos todos os programas que existem ou que existirão em nosso mundo usando este tipo de computador.

Uma pergunta que se pode fazer aqui, é se é melhor aprender tudo de uma linguagem e se tornar um especialista nela, ou se é melhor conhecer várias, ainda que não chegando a muita profundidade em nenhuma.

Não há resposta fácil, mas apostar todas as suas fichas em uma única pode ser perigoso. A tecnologia tem substituído a linguagem "da vez" mais ou menos a cada 8, 10 anos. Pior, ao se fixar em uma linguagem, corre-se o risco de misturar questões do problema (do algoritmo) com questões do programa (da linguagem).

Como na vida real, parece ser melhor buscar ser poliglota, ainda que obviamente haja uma linguagem preferida. Neste caso, todo o esforço deve ser no núcleo mais ou menos comum a todas as linguagens e deixar em segundo plano o aprendizado das idiossincrasias de cada uma.

EXERCÍCIO 1 Lembrando que existe muita semelhança entre seguir um programa de computador e uma receita culinária, prepare um fluxograma da seguinte receita:

Bolo de nozes da Dona Filustreca

Ingredientes: 250g de chocolate em pó, 250g de passas, 3 xícaras de açucar, meia xícara de leite condensado, meia xícara de óleo, 1 colher de chá de baunilha, 250g de manteiga e 1 colher de chá de sal.

Modo de fazer: ponha o leite, o óleo, o açucar, o chocolate e o sal em uma panela e leve ao fogo, mexendo até a fervura. Reduza o fogo e continue a fervura até o ponto de caramelo. Retire do fogo e deixe esfriar por 10 min. Em seguida, bata com a manteiga e a baunilha até a mistura ficar homogênea. Distribua as passas no fundo de uma forma grande untada de manteiga. Derrame a massa sobre as passas. Deixe esfriar por 10 minutos, corte em pedaços e o bolo está pronto para ser comido.

EXERCÍCIO 2 Seja agora um exemplo numérico. Eis aqui a seqüência de Fibonacci

```
1, 1, 2, 3, 5, 8, 13, 21, 34, 55, ...
```

Escreva um procedimento que informe a soma dos primeiros 1000 números da seqüência de Fibonacci.

Fica claro por estes dois exemplos, que programar um computador é "emprestar inteligência ao computador". Ou seja, para programar a solução de um problema, há que saber resolver esse problema. E, não apenas resolver em tese, mas descrever detalhadamente todas as etapas da solução.

De fato, a programação é detalhista ao extremo.

Como se disse acima, o computador surge em 1937, descrito em um artigo denominado "on a computable numbers". Este artigo era tão inédito que quase não é publicado: faltavam revisores que o atestassem. O conceito matemático de computador está lá, quase 10 anos de existir algo físico que lembrasse o conceito.

Ao mesmo tempo, o exército nazista alemão começa a usar um esquema criptográfico muito inteligente através de uma máquina denominada ENIGMA, a qual, tinham eles certeza, era inexpugnável. Mal sabiam que do outro lado do canal da Mancha estava Turing. Ele criou um computador, batizado COLOSSUS, que quebrava o código ENIGMA em poucas horas. Foi o primeiro computador construído pelo homem. O ENIAC de que os americanos se gabam tanto, veio depois.

No meio da guerra, Turing foi enviado aos Estados Unidos e lá teve contacto com Von Neumann. Este é o autor da assim chamada arquitetura de Von Neumann, que é a que usamos até hoje em qualquer computador.

2.2 Algoritmo

> *Algorithms are the most important, durable, and original part of computer science because they can be studied in a language – and machine – independent way. This means that we need techniques that enable us to compare algorithms without implementing them.*
>
> Skiena, Steven - The Algorithm Design Manual.

Segundo mestre Aurélio, algoritmo é "Processo de cálculo, ou de resolução de um grupo de problemas semelhantes, em que se estipula, com generalidade e sem restrições, regras formais para a obtenção do resultado ou da solução do problema".

Algoritmo é a idéia que está por trás de um programa de computador. É a "coisa" que permanece igual ainda que esteja em um programa Java para rodar num PC em Curitiba, ou esteja em um programa PASCAL rodando em um Cray em Valladolid na Espanha.

Do ponto de vista da informática, algoritmo é a regra de solução de um problema, isto é, surgida uma necessidade buscar-se-á uma solução, ou construir-se-á um algoritmo capaz de solucionar o problema.

Já um programa de computador, (segundo Wirth) "é uma formulação concreta de algoritmos abstratos, baseados em representações e estruturas específicas de dados"e devidamente traduzido em uma linguagem de programação.

Outra definição de algoritmo, esta dos autores Angelo Guimarães e Newton Lages:

Algoritmo é a descrição de um padrão de comportamento, expressado em termos de um repertório bem definido e finito de ações "primitivas", das quais damos por certo que elas podem ser executadas". Para encerrar, e tomando alguma liberdade, podemos considerar um algoritmo como sendo uma receita de bolo.

O conceito de algoritmo deve ser entendido, para sua correta compreensão, em seus dois aspectos, a quem chamaremos estático e temporal. Na sua visão estática, um algoritmo é um conjunto de ordens, condições, testes e verificações. No seu aspecto temporal, o algoritmo passa a ser algo vivo, pois atua sobre um conjunto de dados de entrada, para gerar os correspondentes dados de saída.

Tais características não podem ser separadas, elas estão instrinsicamente ligadas. A dificuldade em fazer bons algoritmos, é ter em mente, enquanto se escreve o algoritmo (aspecto estático) o que ele vai fazer com seus dados (aspecto temporal).

Vamos exemplificar esta discussão, comentando sobre o algoritmo de solução de uma equação do segundo grau.

Dada a equação $Ax^2 + Bx + C = 0$, para encontrar as duas raizes, segundo a inesquecível fórmula de Bhaskara, procedemos da seguinte forma:

- Calculamos DELTA, que é B ao quadrado, menos o produto de 4,A e C.
- Se DELTA < 0, as raizes não existem no campo dos números reais

- Se DELTA = 0, as raizes são iguais
- A primeira raiz é (-B) + $\sqrt{DELTA}$ dividido por 2A
- A segunda raiz é (-B) - $\sqrt{DELTA}$ dividido por 2A.

No seu aspecto dinâmico, teríamos que gerar uma série de dados de entrada (triplas A,B,C), seguir o algoritmo, e verificar se os resultados que ele gera são aqueles esperados de antemão.

Dado um problema, para o qual temos que escrever um algoritmo, usualmente não há solução única. Quanto mais complexo o problema a resolver, maior a quantidade de possíveis algoritmos corretos. Cada um certamente terá as suas qualidades e os seus defeitos. Alguns são muito bons, e outros muito ruins, mas entre um e outro existem inúmeras variações, que nem sempre são facilmente reconhecidas.

Pode-se dizer que um algoritmo é um programa de computador do qual se faz a abstração da linguagem de programação. Neste sentido o algoritmo independe de qual será sua implementação posterior. Quando pensamos em algoritmos, não nos preocupamos com a linguagem. Estamos querendo resolver o aspecto "conteúdo". Mais tarde, quando programarmos, a linguagem passa a ser importante, e agora a preocupação é com a "forma".

2.2.1 Qualidades de um bom algoritmo

> "Se um programa é fácil de ler, ele é provavelmente um bom programa; se ele é difícil de ler, provavelmente ele não é bom."(Kernighan e Plauger)

Clareza O algoritmo é uma ferramenta de entendimento e solução de um problema. Deve portanto, ser o mais claro possível. Ao fazer o algoritmo seu autor deve se preocupar constantemente em se o que está pensando está visível no que está escrevendo.

> "Nunca sacrificar clareza por eficiência; nunca sacrificar clareza pela oportunidade de revelar sua inteligência."Jean Paul Tremblay.

Impessoalidade Nem sempre quem vai examinar, depurar, corrigir, alterar etc , um algoritmo é seu autor. Portanto, nada de usar macetes, regras pessoais, nomes que só tem sentido para o autor etc. Neste ponto é de muita ajuda a existência de um bem organizado conjunto de normas de codificação e de elaboração de algoritmos para a organização. As melhores e mais bem organizadas empresas de informática dedicam parcela significativa de seus esforços a esta atividade.

Simplicidade Poucos programadores resistem à tentação de elaborar uma genial saída para um problema. Nada contra isso, desde que essa saída não fira os princípios de simplicidade que todo algoritmo deve procurar satisfazer. Nem sempre é fácil fazer algoritmos descomplicados, principalmente porque em geral os problemas que enfrentamos não são simples. Transformar algo complicado em soluções singelas, parece ser a diferença entre programadores comuns e grandes programadores.

Reaproveitamento Existe inúmera bibliografia sobre algoritmos. Grandes instalações de informática costumam criar o conceito de "biblioteca de algoritmos". Muitos softwares também estão partindo para esta saída. O programador não deve se esquecer nunca de que um algoritmo novo, custa muito caro para ser feito, e custa muito mais caro ainda para ser depurado. Se já existir um pronto e testado, tanto melhor.

Capricho Tenha sempre em mente que a principal função de um algoritmo é transmitir a solução de um problema a outrem. Desta maneira, um mínimo de capricho é indispensável a fim de não assustar o leitor. Letra clara, identação correta, nomes bem atribuídos, tudo isto em papel limpo e sem borrões. Como o leitor já pode estar imaginando, esta característica é fundamental em provas, avaliações e trabalhos. A primeira conseqüência prática desta regra é: faça seus algoritmos a lápis.

Documentação Nem sempre o texto que descreve o algoritmo é muito claro. Além do que, ele informa o que é feito, mas não porque é feito, ou quando é feito. Assim, o programador pode e deve lançar mão de comentários sempre que sentir necessidade de clarificar um ponto.

> "Bons comentários não podem melhorar uma má codificação, mas maus comentários podem comprometer seriamente uma boa codificação". Jean Paul Tremblay.

Correção (ou integridade ou ainda robustez) Um algorítmo íntegro é aquele onde os cálculos estão satisfatoriamente colocados, e atuam corretamente para todos os dados possíveis de entrada. Não adianta um algoritmo sofisticado e cheio de estruturas de controle, se suas operações aritméticas elementares não estiverem corretas. É mais comum (e perigoso) o algoritmo que funciona em 95% dos casos, mas falha em 5% deles.

Eficiência Esta característica tem a ver com economia de processamento, de memória, de comandos, de variáveis etc. Infelizmente, quase sempre é uma característica oposta à da clareza e da simplicidade, sendo estas mais importantes do que a eficiência. Entretanto, em alguns casos, (principalmente na programação real - dentro das empresas), a eficiência pode ser fundamental. Neste caso, perde-se facilidade, para se ter economia. O bom programador é aquele que alia a simplicidade ao máximo de eficiência possível.

Generalidade Dentro do possível, um algoritmo deve ser o mais genérico que puder. Isto significa que vale a pena um esforço no sentido de deixar nossos algoritmos capazes de resolver não apenas o problema proposto, mas toda a classe de problemas semelhantes que aparecerem. Usualmente o esforço gasto para deixar um algoritmo genérico é pequeno em relação aos benefícios que se obtém desta preocupação.

Modularidade Algoritmos grandes dificilmente ficam bons se usarmos uma abordagem linear. É praticamente impossível garantir clareza, impessoalidade, documentação etc, quando se resolve um algoritmo de cima a baixo. Uma conveniente divisão em módulos (isto é, em sub-algoritmos), para que cada um possa ser resolvido a seu tempo. Não podemos esquecer que a regra áurea da programação estruturada é "dividir para conquistar".

Comentários Permitem a agregação de informações destinadas ao leitor humano do algoritmo. Seu conteúdo não é levado em consideração na análise da integridade e correção do algoritmo. Tem finalidade apenas explicativa e informativa. Mas isso não significa que eles sejam desprezíveis. Pelo contrário, comentários são importantes em algoritmos de um modo geral.

Os comentários são delimitados por um "abre-chave"({) e um "fecha-chave"(}). Eles podem ser colocados em qualquer ponto do algoritmo e tudo o que estiver entre chaves é desconsiderado no momento de seguir o algoritmo.

Exemplo {Isto é um comentário}

Há dois tipos de comentários aqui: aqueles encerrados entre chaves e aqueles colocados depois de uma dupla barra.

```
{Isto é um comentário, que só será encerrado quando se encontrar
o fecha chave, ainda que ele ocupe mais de uma linha }
// isto também é um comentário, desde a dupla barra até o fim da linha
```

2.2.2 Como se escreve um algoritmo

A questão agora é: "como se escreve um algoritmo?" Usando uma ferramenta adequada para comunicar algum tipo de computação que o algoritmo execute. Qualquer linguagem poderia ser adequada, mas é melhor inventar uma sublinguagem derivada do português (ou do inglês em livros nesse idioma) que ajude a treinar o pensamento criativo em algoritmos.

A idéia é que a ferramenta ajude a fazer a tarefa. Por analogia, experimente aparafusar algo pequeno usando um pé de cabra.

O nome da sublinguagem que usaremos é "português estruturado", também conhecida como "portugol".

2.3 Portugol

Figura 2.1: Qual a melhor linguagem ?

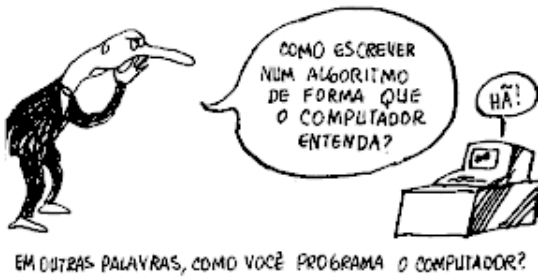


Figura 2.2: Como se entender com ele ?

Como vimos, português estruturado é uma linguagem inventada aqui para simplificar a tarefa de aprender a construir, depurar e documentar algoritmos estruturados. Existem inúmeras versões, cada professor de lógica tem a sua. É importante salientar que a

síntaxe e a construção de português estruturado são arbitrados por alguém, segundo seus critérios, mas uma vez estabelecido um padrão, pele precisa ser seguido.

Esta restrição tem duas vertentes. A primeira, é que há que haver um mínimo de concordância para que outras pessoas possam ler e entender o que escrevemos. A segunda, é que uma das principais características do bom programador é a disciplina intelectual. Nada mais apropriado que começarmos a exercitar esta difícil qualidade desde já.

Por que PORTUGOL e não PORTUGUÊS ? Português (como qualquer idioma natural) tem inúmeras desvantagens:

- Não é entendido pela máquina (exige tradução complexíssima)
- É ambíguo
- É muito extenso

Vejamos por exemplo, uma orientação extraída do manual de organização de uma Empresa.

> "O funcionário fará jus ao PRÊMIO PRODUTIVIDADE se estiver trabalhando há mais de 6 meses da época de distribuição (fevereiro de cada ano), e trabalhar na área de vendas, ou então se tiver sido especialmente citado como tendo tido alta produtividade. Perdem o direito, funcionários que tiverem mais de 2 faltas e mais de 5 atrasos, ou que tiverem fruido licença saúde do INSS no período aquisitivo".

Parece claro, não é ? Agora tentemos responder:

- João tem 3 meses de casa e foi citado como de "alta produtividade". Ganha ou não?
- Maria teve 3 faltas. Perde ou não ?

Outro exemplo, "minha tia tem uma gatinha e ela é muito bonita". Quem é bonita, a tia ou a gatinha ?

O linguajar acima, não é adequado para expressarmos algoritmos pois é inexato, dúbio, e deixa muita margem de interpretação, usa verbos e comandos proibidos, isto é, que o computador não saberá executar. Devemos lembrar sempre que a margem de interpretação do computador, qualquer computador, é sempre ZERO, e os computadores tem a péssima mania de fazer o que mandamos eles fazerem, e não o que queremos que eles façam.

Por que PORTUGOL e não uma linguagem qualquer como Java? ou qualquer outra linguagem de programação?

Uma linguagem de programação, ainda que de alto nível, caso do Java, exige um conhecimento da máquina que ainda não temos. Também é importante salientar que idealmente falando, não é boa política enfrentar dois problemas interligados ao mesmo tempo. Isto é, não convém misturar dificuldades de lógica, (advindas do problema que se quer resolver) com dificuldades da linguagem (advindas do ambiente tecnológico onde o programa vai rodar), pois ambas somadas podem ser demais para a cabeça do autor. Já dizia Dijkstra "deve-se reconhecer que tem-se uma cabeça pequena, e que é melhor tratar de conviver com ela e respeitar suas limitações". Aproveitando a citação, mais de uma do mesmo autor: "A regra áurea da programação estruturada é dividir para reinar".

Os programas em PORTUGOL vão deixar de lado inúmeras chateações, (que são imprescindíveis para o programa rodar em um computador real) tais como: instruções de ambientação na compilação de programas, descrições de operações de entrada/saída complexas, pouca ou nenhuma preparação de controles etc.

É claro, que anexo a esta disciplina, existe outra de laboratório de programação, onde vai se aprender o Java. Nessa hora, os algoritmos já devem estar consolidados, e a preocupação passa a ser a máquina e seu ambiente. Na nossa cadeira de algoritmos, devemos esquecer a máquina e nos concentramos no problema e na sua solução.

Português estruturado Cada autor tem a sua visão do que seja o ideal em termos de linguagem de especificação de algoritmos. Veja, por exemplo, o quadro

para i := 1,...., n **faça**
  para j := 1,...., n **faça**
  $c_{ij} := 0$
  para k := 1,...., n **faça**
  $c_{ij} := c_{ij} + a_{ik} \cdot b_{kj}$
 	
(a) Szwarcfiter e Markenzon ([SM94])
	para i ← 1 *até* n *faça*
	  para j ← 1 *até* n *faça*
	  $c_{ij} \leftarrow 0$
	  para k ← 1 *até* n *faça*
	  $c_{ij} \leftarrow c_{ij} + a_{ik} \cdot b_{kj}$
	
	(b) Terada ([Ter91])

for i ← 1 **to** n **by** 1 **do**
  for j ← 1 **to** n **by** 1 **do**
  c(i,j) ← 0
  for k ← 1 **to** n **by** 1 **do**
  c(i,j) ← c(i,j) + a(i,k) · b(k,j)
 	
(c) Horowitz e Sahni ([HS82])
	1 **set** i ← 1
	2 **set** j ← 1
	  set c[i,j] ← 0
	3 **set** k ← 1
	  c[i,j] ← c[i,j] + a[i,k] · b[k,j]
	  if k <= n **then go to** 3
	  if j <= n **then go to** 2
	  if i <= n **then go to** 1
	
	(d) Knuth ([Knu73])

Algoritmos para o produto de duas matrizes n x n em várias linguagens algorítmicas. (`M.C.Carrard`)

O que um computador entende

Baseado na arquitetura de Von Neumann, (um processador, uma unidade de controle, uma ULA, memória e registradores), todo computador construído neste paradigma é capaz de obedecer, lá no nível mais baixo, a um pequeno conjunto de ordens. Note-se como ele é restrito.

Entrada/Saída Este tipo de ordem, é para orientar o computador no sentido de adquirir um dado externo (entrada), ou para divulgar alguma coisa que ele calculou (saída).

Aritmética Estas ordens são similares aquelas que aprendemos no primeiro grau, e destinam-se a informar ao computador como ele deve efetuar algum cálculo. Estas ordens são: adição, subtração, multiplicação, divisão, potenciação, funções trigonométricas etc. Não se esqueça que a linguagem universal dos computadores

é a matemática. Escrever um programa de computador correto é como demonstrar um teorema.

Alternativa É talvez a ordem mais importante no universo da computação. É ela que dá ao computador condições de "ser inteligente" e decidir qual de dois caminhos deve se seguido. Sempre tem a forma de uma pergunta, seguida por duas ordens distintas. Se a pergunta for respondida afirmativamente, a primeira ordem será executada, e se a pergunta for negada, a segunda ordem será executada.

Desvio Esta ordem é para dar ao computador habilidade de saltar algumas ordens e executar outras. Um programa de computador nem sempre é obrigado a seguir as instruções seqüencialmente começando no começo e terminando no fim. O desvio é potencialmente perigoso quando usado sem critério, e por isso veremos em seguida que uma das regras da programação estruturada é o estabelecimento de rígidos controles sobre os desvios.

Movimentação na memória Embora pareça estranho, esta é ordem mais comum em um programa. Considerando o tamanho das memórias de computador e a quantidade de coisas que são lá colocadas, é imprescindível que os programas tenham a habilidade de manusear todas estas coisas. Fazendo uma figura de linguagem, suponha que a memória do computador é um grande supermercado. Se os artigos não puderem se mexer de lugar, bem poucas coisas acontecerão.

Outra observação, é que o verbo "mover", quando usado em manipulações dentro da memória é usado por razões históricas, e certamente contém um equívoco. O correto seria usar o verbo "copiar", já que quando um conteúdo é levado de um lado a outro, sua instância anterior não desaparece. Usa-se aqui uma regra jocosa, mas que sempre é verdadeira. **computador não usa apagador**.

ciclo de vida

Qual a história de vida de um programa de computador ? Isto é como nasce, cresce e morre um programa ? Grosso modo, pode se dizer que as principais etapas são:

- estudo do problema: Espera-se que haja algum problema ou dificuldade esperando ser resolvido pelo programa de computador. É pré-requisito antes de tentar a solução de qualquer questão, estudá-la, compreender o entorno, as limitações, os balizadores, os recursos disponíveis. Uma boa maneira de estudar e compreender um problema é fazer hipóteses, estudar comportamentos, projetar simulações. Só se pode passar adiante se boa parte do entorno do problema for entendida. Note-se que nem sempre é possível aguardar até ter 100% do conhecimento antes de seguir adiante. Às vezes não dá tempo, mas o percalço principal nem é esse: é que muitas vezes o conhecimento não existe e só vai aparecer quando se cutucar o problema. Como se diz em espanhol *los caminos se hacen al caminar*. Quando isto ocorrer, e ocorre muitas vezes, não tem jeito: anda-se para frente e para trás, e a cada passada o conhecimento é maior.

- bolar ou procurar o algoritmo. Agora é hora de converter a solução encontrada em algo próximo a um algoritmo. Isto é, algo formal, com condições iniciais, resultado final, e manipulações entre o que entra e o que sai. Usam-se os comandos permitidos em algoritmos e só estes. Usualmente, há aqui o que se chama de aprofundamento interativo. Começa-se com uma declaração genérica que vai sendo detalhada e especificada até chegar ao nível básico, no qual os comandos algorítmicos (entrada/saída, aritméticos, alternativos, movimentação e de repetição)

podem ser usados. Note que pode-se copiar ou aproveitar um algoritmo já existente. Entretanto este re-uso dificilmente é imediato. Sempre há que estudar e eventualmente modificar um algoritmo já existente, a menos que quando este foi feito seu autor já tinha a preocupação com o reuso.

- transcrever em LP. É hora de escolher um computador, um ambiente operacional e sobretudo uma linguagem de programação que esteja disponível e seja conhecida. Dessas 3 opções (computador, ambiente e linguagem) surge a plataforma definitiva: agora é possível escrever, testar, depurar e implementar o programa de computador.

- depurar. Usualmente há uma seqüência de testes:

  - compilação correta: nenhum erro apresentado pelo compilador
  - testes imediatos sem erro: testes *ad hoc* para este programa (e só para ele) dão os resultados esperados.
  - teste alfa: um esforço completo de testes, inclusive integrando este programa aos seus antecessores e sucessores. Usualmente o teste é feito pelo(s) autor(es) do programa.
  - teste beta: um teste completo feito por terceiros. Há uma regra aqui de que quem faz (programador) não deve ser quem testa (testador).

- implementar e rodar. Com o programa testado e liberado é hora de rodá-lo em produção. Nesta hora ocorre o que tem sido chamado de *deploy* (dizem as más línguas que sempre associado a seu irmão gêmeo, o *delay*). Dependendo do porte da instalação e do sistema, às vezes esta passagem é complexa e trabalhosa.

- manutenção. Seria muito bom se a história acabasse na etapa anterior. Não é o que ocorre, já que o mundo muda e os programas precisam mudar com ele. Se a manutenção é pequena, usualmente pode-se fazê-la sem muito transtorno, mas caso não seja, este ciclo recomeça. Note-se que após algumas manutenções grandes o programa começa a dar mostrar de instabilidade –a popular colcha de retalhos – e é hora de começar tudo de novo, na nova versão do programa.

Como se aprende algoritmos?

Não é fácil aprender algoritmos. No que têm de engenharia, é mais fácil. Há fórmulas, procedimentos, práticas consagradas prontas a serem usadas. Mas, no que são arte, tais receitas desaparecem. É mais ou menos como tentar ensinar alguém a desenhar (outra arte) ou a compor música (outra ainda).

Olhando como se aprende a desenhar, é possível ter alguns *insights*. Aprendizes de desenho copiam desenhos e partes de desenho extensivamente. Pode-se tentar algo parecido aqui.

Em resumo, estas seriam as indicações

- Estudando algoritmos prontos. Pegue bons livros de algoritmos, e estude as implementações que lá aparecem. Transcreva tais algoritmos em um simulador (tipo visualg) ou mesmo em uma linguagem de programação. Execute os algoritmos. Busque entender o que cada parte do mesmo faz. Sugira hipóteses sobre o funcionamento e provoque pequenas mudanças no código para confirmar (ou rejeitar) suas hipóteses.

- Tenha intimidade com algum ambiente de compilação e testes. Esta proposta é importante para poder testar algoritmos. Se ficar embatucado diante de uma simples implementação, você terá grandes dificuldades de escrever algoritmos.

- Escreva pequenos algoritmos. Tente fazer os exercícios (começando sempre pelos mais simples) de qualquer bom livro de algoritmos.

- Treine a interpretação dos enunciados. Enquanto não entender o que deve ser feito, não adianta olhar para o algoritmo, pois o motivador das ações que ele toma estará ausente. Nunca se deve começar a estudar ou a construir um algoritmo se não houver um claro entendimento do que o algoritmo deve fazer.

2.4 Programação Estruturada

O termo programação estruturada veio à luz pela primeira vez, nos fins da década de 60, quando Edsger Dijkstra (Lê-se como *Dikster*) escreveu um artigo, publicado em *Communications of the ACM*, cujo título é "O comando GO TO é prejudicial". Vivíamos a época de ouro do COBOL, e o comando GO TO é a instrução de desvio desta linguagem. Dijkstra observou que a qualidade dos programadores decai em função do número de GO TOs usados em seus programas. Segundo ele, "Comandos GO TO tendem a criar caminhos de lógica confusos e programas pouco claros". A sua recomendação foi de que o comando em questão fosse banido das linguagens de alto nível.

Nessa época também (1966) dois professores italianos, Giuseppi Jacopini e Corrado Bohm, provaram matematicamente que qualquer lógica de programação poderia ser derivada de três tipos básicos de procedimentos, a saber: seqüência, alternativa e repetição.

Chegou-se então ao âmago da programação estruturada, que é trocar a instrução de desvio pelas instruções de repetição. Ao fazer isto, todos os componentes de um programa passam a ter uma entrada e saída únicas. Atente-se que ao usar uma instrução de desvio, essa regra de uma entrada e uma saída é violada: o conjunto passa a ter no mínimo duas saídas.

Seqüência simples Trata-se de um conjunto de comandos simples, que seguem um ao outro e que são executados na ordem em que aparecem. Exemplo

```
A ← 10
B ← A + 5
X ← 1 + 2
```

Dentro de uma seqüência simples de comandos a entrada sempre é pelo primeiro comando e a saída sempre é pelo último. Graças ao princípio da programação estruturada (uma entrada e uma saída), uma seqüência simples pode ser substituída por um bloco com entrada e saída únicas.

Alternativa É um comando que permite dois caminhos alternativos, daí o nome, a depender de alguma condição. Na matemática convencional isto já existe, por exemplo na fórmula de Bhaskhara. No momento de calcular a raiz quadrada de ∇, há que se tomar uma decisão. Se $\nabla < 0$ os cálculos cessam e a resposta de raízes imaginárias deve ser dada. Se $\nabla \geq 0$, a raiz pode ser extraída e um ou dois valores de raízes emergem.

O ponto importante aqui é que em algum lugar os dois ramos se juntam novamente, e a regra de ouro da programação estruturada (entrada única e saída única também) continua verdadeira.

Tal como na seqüência, um comando alternativo, por mais complexo que seja, pode ser substituído por única caixa.

Repetição O terceiro bloco da programação estruturada é a resposta à necessidade de usar o comando de desvio. Perceba-se que aqui existe um desvio implícito, já que ao se terminar o bloco, há que se desviar para o início do mesmo, que é o que caracteriza uma repetição. A restrição a obedecer não é a ausência de desvio – de resto, impossível de obedecer – mas sim a regra da entrada e saída únicas. Em outras palavras não é proibido usar o desvio, desde que ele esteja contido em limites bem determinados.

Tal como na seqüência e na alternativa, um comando de repetição, por maior ou mais complexo que seja, pode ser substituído por única caixa.

A importância desta descoberta para o software teve tanto impacto quanto a de que qualquer forma lógica de hardware pode ser construida pelas combinações de portas AND, OR e NOT.

A grande vantagem dos três elementos da programação estruturada é que possuem a característica de pacote ou caixa preta, uma vez que todos tem uma única entrada e uma única saída. Isto significa que partes de programas construidos com estas estruturas podem ser também considerados pacotes, sempre com entradas e saidas únicas. Esta é a melhor qualidade da programação estruturada, e é por esta característica que o GOTO deve ser desprezado na construção de bons algoritmos. Tanto isto é verdadeiro, que as linguagens mais modernas, não tem mais a instrução de desvio, banida por perniciosa e desnecessária.

2.5 A máquina de Turing

Embora a humanidade use o conceito de algoritmo há milênios (O de MDC é devido a Euclides), e o próprio nome algoritmo derive do autor árabe de um tratado de álgebra escrito por volta de 800 dC, a definição precisa do que seja um algoritmo geral é apenas de 1930. Ela é devida a Turing, através do seu conceito de Máquina de Turing. Esta não é uma máquina real, sendo apenas uma abstração matemática, daí o seu poder de encanto.

Em 1900, o matemático alemão Hilbert apresentou em um congresso uma lista de problemas que segundo ele *estariam ocupando as mentes matemáticas no século que ora se iniciava*. O décimo problema de Hilbert, dizia: **...haverá algum procedimento mecânico geral que possa em princípio resolver todos os problemas da matemática ?** [1]

Em 1937, Turing respondeu a esta pergunta através do conceito de Máquina de Turing. Ela é composta por um número finito de estados, ainda que possa ser muito grande. Deve tratar um input infinito. Tem um espaço exterior de armazenagem também infinito. Turing imaginou o espaço exterior para dados e armazenamento como sendo uma fita, que pode mover-se para frente e para trás. Além disso a máquina pode ler e escrever nesta fita. A fita está composta por quadrados (ou seja, o nosso ambiente é discreto). Apenas 2 símbolos podem ser escritos nos quadrados da fita, digamos 0 significando quadrado em branco e 1 (quadrado preenchido), ou vice-versa.

Isto posto, lembremos que:

1. os estados internos do aparelho são finitos em número

2. o comportamento da máquina é totalmente determinado pelo seu estado interior e pelo input.

[1] Eis o problema original, como proposto por Hilbert: Determination of the solvability of a diophantine equation: Given a diophantine equation with any number of unkown quantities and with rational numerical coefficients: to devise a process according to which it can be determined by a finite number of operations whether the equation is solvable in rational integers

Ou seja, dado o estado inicial e um símbolo de input, o aparelho deve ser determinístico [2]. Logo ele,

- muda seu estado interno para outro, e/ou
- Muda o simbolo que foi lido pelo mesmo ou por outro, e/ou
- movimenta-se para a direita ou para a esquerda, 1 quadrado, e/ou
- Decide se continua ou interrompe e para.

Definição de uma máquina de Turing Para especificarmos nossa máquina de Turing, teríamos que escrever algo do tipo

Estado atual	Sinal lido	faz o que	Vai para estado	Escreve	Anda para onde
0	0	→	0	0	D
0	1	→	13	1	E
1	0	→	65	1	D
1	1	→	1	0	D
2	0	→	0	1	D.PARE
2	1	→	66	1	E

Esta definição acima, é adequada para a tarefa de numeração das Máquinas de Turing, fenômeno esse que é central na teoria. Entretanto, para operar com esta máquina, talvez seja mais adequado escrever uma tabela como

estado	chega	escreve?	anda	vai para
0	0		D	0
0	1		E	13
1	0	1	D	65
1	1	0	D	1
2	0	1	PARE	
2	1		E	66

As diversas máquinas de Turing Para ver isso funcionando, precisa-se numerar as máquinas de Turing. Necessita-se um pouco de esperteza aqui. Primeiro, converte-se D, E, PARE, → e vírgula como os números 2, 3, 4, 5 e 6. Ou usando um sistema unário, 110, 1110, 11110, 111110 e 1111110. Zero será 0, e 1 será 10, como já se viu. As colunas 4 e 5 nas tabelas não precisam separação, já que o conteúdo da coluna 5 é apenas 1 ou zero e sempre ocupa 1 caracter.

Pode-se dar ao luxo de não codificar nenhuma seta, nem nada do que as antecede (colunas 1, 2 e 3) desde que se organize os comandos em rigorosa ordem crescente (e tomando cuidado para preencher comandos que no modo tradicional não seriam escritas por nunca ocorrerem na máquina).

Fazendo tudo isso, e escrevendo toda a programação da máquina que soma 1 a um número unário, ela fica:

```
10101101101001011010100111010010110101111010000111010010101 1
10100010111010100011010010110110101010101101010101101010100.
```

Convertendo este número binário para decimal chegamos a 450813704461563958982113775643437908
O que significa que a máquina que soma 1 em unário é a 450.813.704.461.563.958.982.113.775.643.437.908[a]

[2] um fenômeno é deterministico quando gera sempre o mesmo resultado a partir das mesmas condições iniciais. Por exemplo, soltar um corpo em queda livre. Ao contrário um fenômeno probabilístico (ou randômico ou estocástico) é o fenômeno no qual dadas as mesmas condições iniciais o resultado é imprevisível. Por exemplo, lançar um dado

máquina de Turing. Naturalmente, mudando-se (otimizando-se) alguma coisa na programação, esta máquina muda de número.

A máquina que soma 1 em unário é a 177.642^a máquina. A que multiplica 2 em binário expandido é $10.389.728.107^a$ máquina e a que multiplica por 2 um número unário é a $1.492.923.420.919.872.026.917.547.669$ ^{a} máquina.

E assim vai. A T_7, é não corretamente especificada, já que existem uma seqüência de 5 uns, e ela emperra ao processar tal seqüência.

Máquina Universal de Turing Teremos uma máquina U, que antes de mais nada lerá as informações na fita de entrada para que ela passe a se comportar como a máquina T. Depois disso vem os dados de input que seriam processados pela máquina T, mas que serão pela U, operando como se fosse a T.

Dizendo que quando a n-ésima máquina de Turing atua sobre o número m produz o número p, podemos escrever $T_n(m) = p$. Podemos pensar nesta relação como sendo uma função de 2 parâmetros (n e m) que leva a p. Esta função, pelo que vimos é totalmente algoritmica. Logo, ela pode ser realizada por uma máquina de Turing, a quem chamaremos de U. Então, fornecendo o par (n, m) a U, obtemos como resposta p.

Podemos escrever $U(n, m) = T_n(m)$

A máquina U quando alimentada primeiro com o número n, passa a se comportar como a T_n

Como U é uma máquina de Turing, ela é uma $T_{alguma-coisa}$. Quanto é essa alguma coisa ?

É mais ou menos 7.24×10^{1688} , ou seja o número 724 seguido de 1686 zeros.

Todos os computadores modernos, desde um humilde Z80 até um Cray multi-vetorial são máquinas de Turing.

Acabamos de ser apresentados a um moderno computador: em $U(n, m) = p$, U é o hardware, n é o software (ou programa), m são os dados de entrada e finalmente p é o resultado esperado.

Para efeito de entender a MT, pode-se desenhar algo como segue: Usar como modelo a MT que multiplica por 2 em unário Construir o modelo acima no quadro e depois testar as sequencias 000010000, 00001100000 e 00001110000. Todas vão funcionar.

2.6 Linguagens de programação

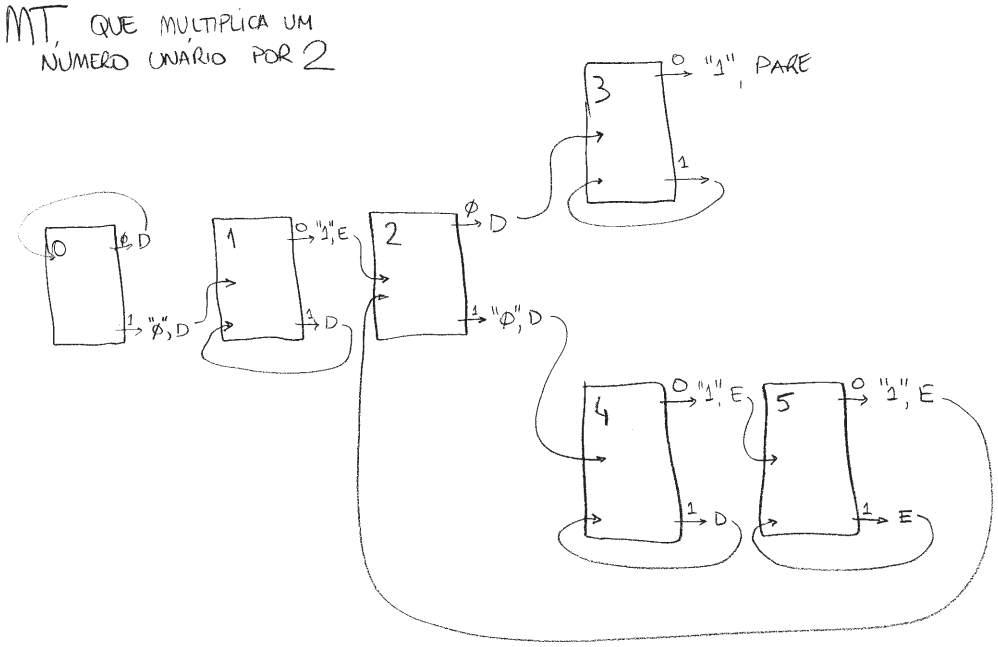
2.6.1 Assembler

O assembler não é uma linguagem de alto nivel. É apenas um tradutor de códigos e portanto está muito próximo da linguagem de máquina. Utilizá-lo é um autêntico calvário, já que todas as operações que o computador precisa realizar ficam por conta do programador. A produtividade é muito baixa, e a única vantagem que se consegue é a eficiência: os programas voam.

Por outro lado, tamanha aderência às entranhas da máquina cobra seu preço: na mudança de plataforma tecnológica um monte de coisas precisa ser reestudado e reciclado.

Não é uma boa idéia programar profissionalmente em assembler. Mas, quem domina esta técnica, é bastante valorizado no mercado. Mesmo que trabalhando em outras linguagens, quem conhece o assembler, conhece o interior da máquina. Sem dúvida alguma, isto tem o seu valor. A seguir um trecho pequeno de um programa em assembler para plataforma Intel.

```
L100: MOV AH,2AH ;pega a data
```




```
INT 21H
PUSH AX ;DEPOIS VOU DAR POR EXTENSO

MOV BX,10
MOV DI,OFFSET DIA

MOV AH,0
MOV AL,DL
DIV BL
OR AX,3030H
STOSW
MOV AL," "
STOSB


MOV AH,0
MOV AL,DH
PUSH AX ;DEPOIS VOU DAR POR EXTENSO
DIV BL
OR AX,3030H
STOSW
MOV AL," "
STOSB
```

2.6.2 Fortran

Fortran é uma linguagem muito antiga, originalmente sugerida pela IBM na década de 50. Seu texto se assemelha a uma descrição matemática e de fato, FORTRAN significa

FORmula TRANslation. É, ou pelo menos era, uma linguagem franciscana de tão pobre em recursos. Foi a única linguagem ensinada em escolas de engenharia até a década de 80. O primeiro compilador de FORTRAN foi desenvolvido para o IBM 704 em 1954-57 por uma equipe da IBM chefiada por John W. Backus.

A inclusão de um tipo de dados de número complexo na linguagem tornou a linguagem Fortran particularmente apta para a computação científica.

As principais revisão são a Fortran 66, 77, 90 e 95. A mais famosa e usada é a Fortran 90. Sua principal vantagem é a eficiência em calculo numérico pesado.

A seguir, um exemplo de FORTRAN

```
C        1         2         3         4         5         6
C234567890123456789012345678901234567890123456789012345678901234567890123456789012345
      PROGRAM BASKHARA
C
      REAL  A,B,C, DELTA, X1,X2, RE, IM
C
      PRINT *, "Este programa resolve uma equação de 2o.grau"
      PRINT *, "do tipo: a*x**2 + b*x + c = 0"
C
      PRINT  10, "Digite a,b,c: "
 10   FORMAT(A,1X,$)
 20   READ(*,*,ERR=20) A,B,C
C
      DELTA=B*B-4.*A*C
C
      IF (DELTA.GT.0) THEN      ! (DUAS RAIZES REAIS)
         X1=(-B-SQRT(DELTA))/(2.*A)
         X2=(-B+SQRT(DELTA))/(2.*A)
         PRINT *, "RAIZES:  X1=",X1
         PRINT *, "         X2=",X2
      ELSE IF (DELTA.EQ.0) THEN ! (DUAS RAIZES REAIS IGUAIS)
         X1=-B/(2.*A)
         X2=X1
         PRINT *, "RAIZES: X1=X2=",X1
      ELSE                      ! (DUAS RAIZES COMPLEXAS)
         RE=-B/(2.*A)
         IM=SQRT(-DELTA)/(2.*A)
         PRINT *, "RAIZES COMPLEXAS: X1=",RE," -",IM,"i"
         PRINT *, "                  X2=",RE," +",IM,"i"
      ENDIF
C
      END
```

2.6.3 Lisp

LISP é a segunda linguagem de alto nivel mais antiga.

O LISP nasce como filho da comunidade de Inteligência Artificial. Na origem, 4 grupos de pessoas se inseriram na comunidade de IA: os lingüistas (na busca de um tradutor universal), os psicólogos (na tentativa de entender e estudar processos cerebrais humanos), matemáticos (a mecanização de aspectos da matemática, por exemplo a demonstração de teoremas) e os informáticos (que buscavam expandir os limites da ciência).

O marco inicial é o encontro no Dartmouth College em 1956, que trouxe duas conseqüências importantes: a atribuição do nome IA e a possibilidade dos diferentes pesquisadores se conhecerem e terem suas pesquisas beneficiadas por uma interfecundação intelectual.

A primeira constatação foi a de que linguagens existentes privilegiavam dados numéricos na forma de arrays. Assim, a busca foi a criação de ambientes que processassem símbolos na forma de listas. A primeira proposta foi IPL (Information Processing Language I, por Newel e Simon em 1956). Era um assemblão com suporte a listas. A IBM engajou-se no esforço, mas tendo gasto muito no projeto FORTRAN decidiu agregar-lhe capacidade de listas, ao invés de criar nova linguagem. Nasceu a FLPL (Fortran List Processing Language).

Em 1958, McCarthy começou a pesquisar requisitos para uma linguagem simbólica. Foram eles: recursão, expressões condicionais, alocação dinâmica de listas, desalocação automática. O FORTRAN não tinha a menor possibilidade de atendê-las e assim McCarthy junto com M. Minsky desenvolveu o LISP. Buscava-se uma função Lisp Universal (como uma máquina de Turing Universal). A primeira versão (chamada LISP pura) era completamente funcional. Mais tarde, LISP foi sofrendo modificações e melhoramentos visando eliminar ineficiências e dificuldades de uso. Acabou por se tornar uma linguagem 100% adequada a qualquer tipo de resolução de problema. Por exemplo, o editor EMACS que é um padrão no mundo UNIX está feito em LISP.

Inúmeros dialetos LISP apareceram, cada um queria apresentar a sua versão como sendo a melhor. Por exemplo, FranzLisp, ZetaLisp, LeLisp, MacLisp, Interlisp, Scheme, T, Nil, Xlisp, Autolisp etc. Finalmente, como maneira de facilitar a comunicação entre todos estes (e outros) ambientes, trabalhou-se na criação de um padrão que foi denominado Common Lisp. Como resultado o Common Lisp ficou grande e um tanto quanto heterogêneo, mas ele herda as melhores práticas de cada um de seus antecessores.

Seja agora um exemplo de uma função lisp:

```
> (defun li (L1 L2)   ; pergunta se duas listas são iguais
    (cond
       ((null L1) (null L2))
       ((null L2) nil)
       ((not (eql (car L1) (car L2))) nil)
       (t (li (cdr L1) (cdr L2)))))
LI
```

2.6.4 Prolog

O nome Prolog para a linguagem concreta foi escolhido por Philippe Roussel como uma abreviação de *PROgrammation LOGique*. Foi criada em meados de 1972 por Alain Colmerauer e Philippe Roussel, baseados no conceito de Robert Kowalski da interpretação procedimental das cláusulas de Horn. A motivação para isso veio em parte da vontade de reconciliar o uso da lógica como uma linguagem declarativa de representação do conhecimento com a representação procedimental do conhecimento.

*Histórico

- Precursores: Newell, Shaw e Simon, com sua Logic Theory Machine, que buscava a prova automática de teoremas, em 1956.

- Robinson, 65, no artigo *A machine oriented logic based on the resolution principle*. Propunha o uso da fórmula clausal, do uso de resolução e principalmente do algoritmo de unificação.

- Green em 1969, escreveu o *Question Answer System*, que respondia perguntas sobre um dado domínio. Na mesma época, Winograd escreveu o software Planner.
- Todos estes tinham problemas de eficiência, por causa, entre outras coisas, da explosão combinatória.
- Em 1970, em Edinbourg, Kowalski bolou o seguinte esquema:
  - Limitou a expressividade às cláusulas de Horn
  - Criou uma estratégia de resolução, não completa, mas muito eficiente
- Em 72, em Edinbourg e Marselha, surgiu a linguagem PROLOG.
- Em 77, Waren escreve o primeiro compilador de PROLOG. Começa o sucesso de PROLOG na Europa. Nos EUA, sempre deu-se maior atenção ao LISP.
- Foi a base para o computador de 5^a geração do Japão.

A seguir, um pequeno trecho de uma sessão PROLOG:

```
parent(maria,jorge):-true.
parent(joao,jorge):-true.
parent(jorge,ana):-true.
...
parent(cris,paulo):-true.
```

Daí: podemos perguntar:

```
? parent(joao,jorge)
YES
? parent(celia,cris)
NO
? parent(cris,carlos)
NO
? parent(X,jorge)
X=maria
X=joao
no (significando acabou)
? parent(X,Y)
X=maria
Y=jorge
X=joao
Y=jorge
...
? parent(X,Y),parent(Y,paulo)  [ler esta vírgula como OU]
X=jorge
Y=cris
```

2.6.5 Cobol

O nome vem da sigla de COmmon Business Oriented Language (Linguagem Comum Orientada aos Negócios), que define seu objetivo principal em sistemas comerciais, financeiros e administrativos para empresas e governos.

O COBOL foi criado em 1959 pelo Comitê de Curto Prazo, um dos três comitês propostos numa reunião no Pentágono em Maio de 1959, organizado por Charles Phillips

do Departamento de Defesa dos Estados Unidos. O Comitê de Curto Prazo foi formado para recomendar as diretrizes de uma linguagem para negócios. Foi constituído por membros representantes de seis fabricantes de computadores e três órgãos governamentais, a saber: Burroughs Corporation, IBM, Minneapolis-Honeywell (Honeywell Labs), RCA, Sperry Rand, e Sylvania Electric Products, e a Força Aérea dos Estados Unidos, o David Taylor Model Basin e a Agência Nacional de Padrões (National Bureau of Standards ou NBS). Este comitê foi presidido por um membro do NBS. Um comitê de Médio Prazo e outro de Longo Prazo foram também propostos na reunião do Pentágono. Entretanto, embora tenha sido formado, o Comitê de Médio Prazo nunca chegou a funcionar; e o Comitê de Longo Prazo nem chegou a ser formado. Por fim, um subcomitê do Comitê de Curto Prazo desenvolveu as especificações da linguagem COBOL.

As especificações foram aprovadas pelo Comitê de Curto Prazo. A partir daí foram aprovadas pelo Comitê Executivo em Janeiro de 1960, e enviadas à gráfica do governo, que as editou e imprimiu com o nome de Cobol 60. O COBOL foi desenvolvido num período de seis meses, e continua ainda em uso depois de mais de 40 anos.

O COBOL foi definido na especificação original, possuia excelentes capacidades de autodocumentação, bons métodos de manuseio de arquivos, e excepcional modelagem de dados para a época, graças ao uso da cláusula PICTURE para especificações detalhadas de campos. Entretanto, segundo os padrões modernos de definição de linguagens de programação, tinha sérias deficiências, notadamente sintaxe prolixa e falta de suporte da variáveis locais, recorrência, alocação dinâmica de memória e programação estruturada. A falta de suporte à linguagem orientada a objeto é compreensível, já que o conceito era desconhecido naquela época. Segue o esqueleto de um programa fonte COBOL.

```
 IDENTIFICATION DIVISION.
         PROGRAM-ID. HELLO-WORLD.
      *
       ENVIRONMENT DIVISION.
      *
       DATA DIVISION.
      *
       PROCEDURE DIVISION.
       PARA-1.
           DISPLAY "Alô, Mundo.".
      *
           STOP RUN.
```

2.6.6 APL

Ela nasceu do trabalho de um professor de matemática canadense de nome Keneth Iverson. Sua proposta original era a de produzir uma nova notação matemática, menos sujeita às ambigüidades da notação convencional.

Na década de 60, trabalhando na IBM em conjunto com Adin Falcoff, ambos produziram a primeira versão de APL, quando um interpretador da linguagem ficou disponível.

A principal característica de APL é o uso de um conjunto especial de caracteres que incluem algumas letras gregas (rho, iota...), símbolos matemáticos convencionais (o sinal de vezes, o de dividido...) e alguns símbolos especialmente inventados.

Este fato sempre limitou a disseminação da linguagem. Até o advento das interfaces gráficas, como o windows, por exemplo, exigia-se um hardware especial para poder programar em APL.

Programas em APL em geral sempre são muito pequenos, embora poderosos. A linguagem está preparada para tratar arranjos de grandes dimensões. Por exemplo, quando em APL se escreve A+B, se A e B forem escalares (isto é um número único),

a resposta também o será. Se A e B são vetores de 100 números, a resposta também o será. Idem para matrizes e até arrays-nd. Em algumas versões de APL este n chega a 256 dimensões.

Este é um comando EDIT (troca algo, de... para...)

```
      ∇edit[□]∇
    ∇
[0]   r←depara edit x
[1]   r←,x
[2]   r[(r=1↑depara)/⍳⍴r]←1↓depara
[3]   r←(⍴x)⍴r
    ∇ 2007-07-22 09.33.44 (GMT-4)
```

2.6.7 Basic

A linguagem BASIC (acrônimo para Beginners All-purpose Symbolic Instruction Code), foi criada, com fins didáticos, pelos professores John G. Kemeny e T. Kurtz em 1963 no Dartmouth College.

BASIC também é o nome genérico dado a uma grande família de linguagens de programação derivadas do Basic original. Provavelmente existem mais variações de Basic do que de qualquer outra linguagem de programação.

Basic é uma linguagem imperativa de alto nível, pertencente à terceira geração, que é normalmente interpretada e, originalmente, não estruturada, por ter sido fortemente baseada em FORTRAN II.

Um programa em Basic tradicional tem suas linhas numeradas, sendo que é quase que padrão usar números de 10 em 10 (o que facilita a colocação de linhas intermediárias). Os comandos são poucos, simples e facilmente compreensíveis na língua inglesa (LET, IF, ...). Um programa em Basic, que calcula a fórmula de Bhaskara ficaria como:

```
10  REM RESOLVE EQUACAO DO SEGUNDO GRAU
20  READ A,B,C
25  IF A=0 THEN GOTO 410
30  LET  D=B*B-4*A*C
40  IF D<0 THEN GOTO 430
50  PRINT "SOLUCAO"
60  IF D=0 THEN GOTO 100
70  PRINT "PRIMEIRA SOLUCAO",(-B+SQR(D))/(2*A)
80  PRINT "SEGUNDA SOLUCAO",(-B-SQR(D))/(2*A)
90  GOTO 20
100 PRINT "SOLUCAO UNICA",(-B)/(2*A)
200 GOTO 20
410 PRINT "A DEVE SER DIFERENTE DE ZERO"
420 GOTO 20
430 PRINT "NAO HA SOLUCOES REAIS"
440 GOTO 20
490 DATA 10,20,1241,123,22,-1
500 END
```

O programa demonstra a falta de estruturação da linguagem original, pois o IF funciona como um GOTO condicional, o que favorece o código espaguete.

2.6.8 Clipper

Tudo começou com o dbase, que foi o precursor de bancos de dados para micros. Por volta de 83, 84 surgiu o dbase 2 (nunca houve o 1) para micros de 8 bits com memórias típicas de 64Kb e um ou dois disquetes de 8 polegadas e cerca de 512 KB de capacidade em cada um.

O dbase já era um gerenciador de dados, dentro da visão relacional, com mínima redundância de dados e com todas as garantias de acesso e uma linguagem estruturada, no melhor padrão, de uso geral, capaz de garantir um aumento na produtividade na programação.

A história do dBASE começa em 74, no Jet Propulsion Laboratory, conhecido como JPL, e instalado em Pasadena, Califórnia.

Nesta instalação da NASA trabalhava Jeb Long, pesquisador que lidava com dados obtidos nas pesquisas espaciais através de naves não tripuladas. Estes, por serem em grande número, começaram a trazer problemas e dissabores a Jeb Long, que desenvolveu, então, um programa de computador capaz de gerenciar tais informações. Este programa com nome de JPLDIS, foi concluído e alcançou razoável sucesso tendo sido bastante comentado pelo pessoal técnico. Foi o que bastou para que um analista de sistemas (Wayne Ratliff) começasse a desenvolver um programa gerenciador de dados, genérico, mas tendo como modelo o JPLDIS.

Rapidamente o novo trabalho tornou-se melhor do que o original, e Ratliff desenvolveu-o cada vez mais.

Em 79, Ratliff achou que o programa já estava maduro e pronto para enfrentar o mercado, e ele foi anunciado com o nome de VULCAN. Foi, entretanto, um fracasso: não se chegou a vender sequer 50 cópias.

Alguns meses depois, um grande distribuidor de programas de micros, George Tate, tomou conhecimento do dBASE. Testou-o e ficou entusiasmado.

Achou que o que faltava a Ratliff era suporte comercial, uma vez que seu trabalho era muito bom. Rapidamente providenciaram-se algumas mudanças cosméticas no programa, seu nome foi mudado para dBASE II (nunca houve o dBASE I, tratava-se de estratégia comercial, para apresentar o produto como um melhoramento), e uma nova empresa (ASHTON TATE) foi criada para vender este produto.

Em pouco tempo, dBASE tornou-se um marco na história de softwares para microcomputadores.

A grande vantagem de um banco de dados é retirar a amarração entre programas e dados.

A explicação desta vantagem é que dados são muito mais estáveis do que os procedimentos (=programas). Separando-os garante-se que longevidade aos dados, que em geral são os mais custosos para adquirir.

O dbase como linguagem era interpretado, o que trazia desempenho pífio nas aplicações, além da eventual falta de segurança (qualquer um podia alterar os dados). Para corrigir estas duas deficiências, logo surgiram inúmeros compiladores, dos quais o mais famoso foi o CLIPPER.

Eis um programa em CLIPPER, por acaso um trecho do programa que emite provas aleatórias e diferentes:

```
w_nomq = space(8)
w_file = "LIXO    "
@ 08,45 say "Nome Arquivo: " get w_file pict "@!"
@ 09,5 say "Qual a turma ? " get w_turm pict "@!"
```



```
@ 09,40 say "OU arquivo de nomes " get w_nomq pict "@!"
oba := 1
do while oba == 1
   oba := 0
   W_mess := "   "
   @ 11,15 say "Questao 1:"
   @ 12,20 say "Topico: " get w_top1
   @ 12,50 say "Grupo: " get w_gru1 pict "99"
```

2.6.9 Natural

NATURAL é uma linguagem de quarta geração que pode ser utilizada, indistintamente por usuários finais trabalhando em auto-serviço, ou por programadores profissionais em suas atividades normais no CPD.

Embora possa ler arquivos seqüenciais, o Natural é mais indicado para ler bancos de dados ADABAS, que a partir da década de 90 foi o padrão de bancos de dados em nosso país e em boa parte do mundo.

Eis um exemplo de programa em Natural:

```
0010 AT TOP OF PAGE
0020    DO
0030       WRITE 20T "INFORMACOES SALARIAIS PARA SANTA  CATARINA"
0040       SKIP 1
0050    DOEND
0060 FIND PESSOAL WITH ESTADO = "SC"
0070    SORTED BY CIDADE
0080 AT BREAK OF CIDADE
0090    DISPLAY  NOTITLE  "NOME DA/CIDADE" OLD(CIDADE)
0100             "TOTAL/SALARIOS" SUM(SALARIO) (EM=ZZZ,ZZZ,ZZZ)
0110             "SALARIO/MEDIO"  AVER(SALARIO)
0120             "SALARIO/MAXIMO" MAX(SALARIO)
0130             "NUMERO/PESSOAS" COUNT(CIDADE) (EM=ZZ99)
0140 AT END OF DATA
0150    DO
0160       SKIP 1
0170       WRITE "TOTAL DE TODOS OS SALARIOS" TOTAL(SALARIO)
0180    DOEND
0190 END
```

2.6.10 Pascal

É uma linguagem de programação estruturada que recebeu este nome em homenagem ao matemático Blaise Pascal. Foi criada em 1970 pelo suíço Niklaus Wirth, tendo em mente encorajar o uso de código estruturado.

Segundo o autor, Pascal foi criada simultaneamente para ensinar programação estruturada e para ser utilizada em sua fábrica de software. A linguagem reflete a liberação pessoal de Wirth após seu envolvimento com a especificação de ALGOL 68, e sua sugestão para essa especificação, o ALGOL W.

A linguagem é extremamente bem estruturada e muito adequada para ensino de linguagens de programação. É provavelmente uma das linguagens mais bem resolvidas entre as linguagens estruturadas, e certamente um dos exemplos de como uma linguagem especificada por uma pessoa pode ser bem melhor do que uma linguagem especificada por um comitê.

Pascal originou uma enorme gama de dialetos, podendo também ser considerada uma família de linguagens de programação. Grande parte de seu sucesso se deve a criação, na década de 80, da linguagem Turbo Pascal, inicialmente disponível para computadores baseados na na arquitetura 8086 (com versões para 8080 no seu início).

Pascal é normalmente uma das linguagens de escolha para ensinar programação, junto com Scheme, C e Fortran. Só mais recentemente o Java entrou nesta lista.

Comercialmente, a linguagem foi sucedida pela criação da linguagem Object Pascal, atualmente utilizada nas IDEs Borland Delphi, Kylix e Lazarus. Academicamente, seus sucessores são as linguagens subseqüentes de Niklaus Wirth: Modula-2 e Oberon.

A partir da versão 2005, o Delphi passou a se referir a sua linguagem de programação como Delphi Language.

Assim como a Linguagem C, que é padronizado pela ANSI (Ansi C), o Pascal possui padrões pela ISO, como o Pascal Standard e o Advanced Pascal. Um exemplo:

```
program Teste;
var a,b:integer;
uses crt;
begin
   writeln ('Digite um número para A');
   readln (a);
   writeln ('Digite o número para B');
   readln (b);
   if (a > b) then     { Se A é maior que B então }
       writeln ('A é maior que B')
   else                { Senão... }
       writeln ('B é maior que A');
end
```

2.6.11 C

C é uma linguagem de programação estruturada e padronizada criada na década de 1970 por Dennis Ritchie e Ken Thompson para ser usada no sistema operacional UNIX. Desde então espalhou-se por muitos outros sistemas operacionais, e tornou-se uma das linguagens de programação mais usadas.

C tem como ponto-forte a sua eficiência e é a linguagem de programação de preferência para o desenvolvimento de software básico, apesar de também ser usada para desenvolver aplicações. É também muito usado no ensino de ciências da computação, mesmo não tendo sido projetada para estudantes e apresentando algumas dificuldades no seu uso. Outra característica importante de C é sua proximidade com a linguagem de máquina, que permite que um projetista seja capaz de fazer algumas previsões de como o software irá se comportar ao ser executado.

C tem como ponto fraco a falta de proteção que dá ao programador. Praticamente tudo que se expressa em um programa em C pode ser executado, como por exemplo pedir o vigésimo membro de um vetor com apenas dez membros. Os resultados muitas vezes totalmente inesperados e os erros são difíceis de encontrar.

Muitas linguagens de programação foram influenciadas por C, sendo que a mais utilizada atualmente é C++, que por sua vez foi uma das inspirações para Java. O exemplo:

```
#include <stdio.h>
struct pessoa
{
        unsigned short idade;
```



```
        char nome[51];        /*vector de 51 chars para o nome*/
        unsigned long bi;
}; /*estrutura declarada*/
int main(void) {
        struct pessoa Sousa={16,"Diogo Sousa",123456789};
                                        /*declaracao de uma variavel tipo struct
                                                pessoa com o nome Sousa*/
        printf("Idade: %d\n",(int)Sousa.idade);  /* "%d" espera um int */
        printf("Nome: %s\n",Sousa.nome);
        printf("BI: %lu\n",Sousa.bi);
        return 0;
}
```

2.6.12 Java

Java é uma linguagem de programação orientada a objeto desenvolvida na década de 90 pelo programador James Gosling, na empresa Sun Microsystems. Diferentemente das linguagens convencionais, que são compiladas para código nativo, a linguagem Java é compilada para um ”bytecode” que é executado por uma máquina virtual.

Desde seu lançamento, em maio de 1995, a plataforma Java foi adotada mais rapidamente do que qualquer outra linguagem de programação na história da computação. Em 2003 Java atingiu a marca de 4 milhões de desenvolvedores em todo mundo. Java continuou crescendo e hoje é uma referência no mercado de desenvolvimento de software. Java tornou-se popular pelo seu uso na Internet e hoje possui seu ambiente de execução presente em web browsers, mainframes, SOs, celulares, palmtops e cartões inteligentes, entre outros.

Principais Características da Linguagem Java

A linguagem Java foi projetada tendo em vista os seguintes objetivos:
Orientação a objeto - Baseado no modelo de Smalltalk e Simula67
Portabilidade - Independência de plataforma - “write once run anywhere”
Recursos de Rede - Possui extensa biblioteca de rotinas que facilitam a cooperação com protocolos TCP/IP, como HTTP e FTP
Segurança - Pode executar programas via rede com restrições de execução
Sintaxe similar a Linguagem C/C++
Facilidades de Internacionalização - Suporta nativamente caracteres Unicode
Simplicidade na especificação, tanto da linguagem como do ”ambiente”de execução (JVM)
É distribuída com um vasto conjunto de bibliotecas (ou APIs)
Possui facilidades para criação de programas distribuídos e multitarefa (múltiplas linhas de execução num mesmo programa)
Desalocação de memória automática por processo de coletor de lixo (garbage collector)
Carga Dinâmica de Código - Programas em Java são formados por uma coleção de classes armazenadas independentemente e que podem ser carregadas no momento de utilização.

Eis um exemplo:

```
public abstract class Animal {
   public abstract void fazerBarulho();
}

public class Cachorro extends Animal {
   public void fazerBarulho() {
       System.out.println("AuAu!");
   }
```

```
}

public class Gato extends Animal {
   public void fazerBarulho() {
       System.out.println("Miau!");
   }
}
```

2.6.13 PHP

PHP (um acrônimo recursivo para "PHP: Hypertext Preprocessor") uma linguagem de programação de computadores interpretada, livre e muito utilizada para gerar conteúdo dinâmico na Web.

A linguagem surgiu por volta de 1994, como um subconjunto de scripts Perl criados por Rasmus Lerdof, com o nome Personal Home Page Tools. Mais tarde, em 1997, foi lançado o novo pacote da linguagem com o nome de PHP/FI, trazendo a ferramenta Forms Interpreter, que era na verdade um interpretador de comandos SQL.

Trata-se de uma linguagem modularizada, o que a torna ideal para instalação e uso em servidores web. Diversos módulos são criados no repositório de extensões PECL (PHP Extension Community Library) e alguns destes módulos são introduzidos como padrão em novas versões da linguagem. Muito parecida, em tipos de dados, sintaxe e mesmo funções, com a linguagem C e com a C++. Pode ser, dependendo da configuração do servidor, embutida no código HTML. Existem versões do PHP disponíveis para os seguintes sistemas operacionais: Windows, Linux, FreeBSD, Mac OS, OS/2, AS/400, Novell Netware, RISC OS, IRIX e Solaris

Construir uma página dinâmica baseada em bases de dados é simples, com PHP. Este provê suporte a um grande número de bases de dados: Oracle, Sybase, PostgreSQL, InterBase, MySQL, SQLite, MSSQL, Firebird etc, podendo abstrair o banco com a biblioteca ADOdb, entre outras.

PHP tem suporte aos protocolos: IMAP, SNMP, NNTP, POP3, HTTP, LDAP, XML-RPC, SOAP. É possível abrir sockets e interagir com outros protocolos. E as bibliotecas de terceiros expandem ainda mais estas funcionalidades.

Veja um exemplo de PHP

```
<? ...
      $tipss =mysql_result($result,0,'SSERVTIPSS');
      $porss =mysql_result($result,0,'SSERVPORSS');
      $daali =mysql_result($result,0,'SSERVDAALI');
      $anexo =mysql_result($result,0,'SSERVANEXO');
      $query = "select * from ANDAM where ANDAMNSERV ='$nserv1' "  ;
      $result = mysql_query($query);
      $quantos = mysql_num_rows($result);
      $quantos++;
      ?>
      <table > <tr>
      <td>Numero da solicitacao de servico<td><? echo $nserv ?><tr>
      <td>autor<td><? echo $nomeu ?><tr>
      <td>cliente<td><? echo $clien ?><tr>
      <td>CA responsavel<td><? echo $cares ?><tr>
      <td>interlocutor no cliente<td><? echo $intcl ?><tr>
      <td>fone<td><? echo $fonic ?><tr>
      <td>email<td><? echo $emaic ?><tr>
```


2.6.14 J

Para os que acharam APL uma linguagem meio sem pés nem cabeça, eis aqui J. Olhando para o J, o APL passa a ser tão comportada quanto um COBOL da década de 70.

Para entender o J, precisamos estudar a vida do cara que inventou o APL, o canadense Ken Iverson. Na minha opinião, o sujeito foi um gênio. Coloco-o sem nenhum medo de errar na galeria dos grandes matemáticos da humanidade, talvez o primeiro (junto com Mandelbroot) que tenha realmente conseguido casar com sucesso a matemática e a computação.

Depois de propor o APL como uma notação matemática (década de 50), de liderar o grupo que converteu o APL em linguagem de programação (década de 60 na IBM) e de liderar a popularização da linguagem (anos 70 e começos dos 80), em meados dos 80, ele chegou a algumas conclusões:

- o uso de um alfabeto não usual (para ser educado), restringia o uso do APL (lembremos que ainda não havia o windows e portanto para usar APL havia que comprar hardware especializado - e caro).
- o desenvolvimento continuado por 30 anos da linguagem apontou algumas inconsistências teóricas no modelo. Não esqueçamos que o Iverson era um matemático da pesada
- o custo que o APL sempre teve inviabilizava seu uso pelos menos aquinhoados

Dessas elocubrações nasceu a linguagem J. O nome não tem explicação, exceto a dada por um de seus autores ("Why 'J'? It is easy to type."). Há quem diga que J segue "I"no alfabeto, sendo este "I"a notação Iverson. Seja como for, J não tem nada a ver com Java. Para maiores detalhes veja www.jsoftware.com.

Eis a seguir um programa J:

```
gerasima =: 3 : 0
r=.(2$y)$(1+?200$2 2 2 2 2 3 3 3 4 4 4 5 6)*2+-<.1.5*?200$2 1 1 1 2 1 2
xx=.(y?(<.y*1.6))
z=.+/|:r*($r)$xx
r,.((y,1)$z),.(y,1)$xx
)
```

Equivale ao programa gerasima do workspace vivo128, que gera um sistema de n incógnitas e n equações, depois o resolve para obter os termos independentes e possibilitar propor ao aluno que descubra as incógnitas.

Capítulo 3

Escrevendo algoritmos

Para escrever algoritmos usando português estruturado (portugol), primeiro necessita-se estabelecer algumas regras de notação e de síntaxe. Começa-se descrevendo qual a forma de ler os quadros indicativos da síntaxe da linguagem.

Esta notação resume a linguagem de especificação de algoritmos. Ela é um apanhado de no mínimo 4 linguagens (Java, PASCAL, CLIPPER e APL), e tem a finalidade de expressar o funcionamento do algoritmo em bom português, da maneira mais clara possível e completamente descompromissado das idiossincrasias dessas e outras linguagens de programação.

Símbolo no formato	Significado
palavra	Na escrita desse comando, "palavra"deve ser escrita exatamente como aparece no formato.
<palavra>	<palavra> deve ser substituído por algum tipo de palavra aceitável neste contexto
[palavra] ou [< palavra >]	O fato de algo estar escrito entre colchetes significa que é OPCIONAL
... (reticências)	Repetição (de 0 a n vezes) de um determinado item. As reticências, às vezes também aparecem na omissão de uma parte opcional em um comando qualquer
{ opção 1 | opção 2 | ... }	A barra vertical indica que uma de várias opções deve ser a escolhida

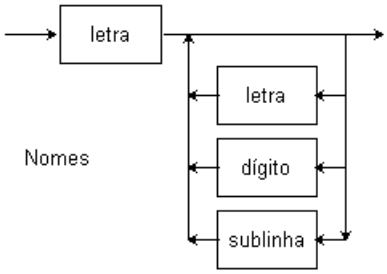
Eis como usar esta tabela. Seja por hipótese a seguinte especificação de comando:

<nome> gostaria [muito] de {comer | beber}

A partir da especificação acima, seriam válidos os comandos
João gostaria de beber
Maria gostaria muito de comer
Antônio gostaria de comer...

3.1 Nome

O elemento básico da linguagem é o nome, também conhecido como identificador. Sua construção é dada pelo diagrama a seguir



Dentro deste diagrama, (e nos próximos que tiverem este formato) qualquer caminho seguido, levar a identificadores válidos. Consideram-se "letra", as 26 letras do alfabeto latino maiúsculas, e "dígito", os 10 dígitos de 0 a 9. Deve-se atentar que o branco não faz parte do rol de caracteres válidos, o que faz com que o identificador não possa ser constituído de mais de uma palavra. Pode-se usar neste caso o separador (_), chamado "sublinha". Atente-se para a importância dos nomes criados pelo programador serem escritos em maiúscula.

Exemplos válidos

```
NOME
SOM_TERMOS
RAIZ1
SALDO08
SALDO_09
```

Exemplos não válidos

```
nome (usa-se letras minúsculas)
Saldo Devedor (usa-se duas palavras)
SaldoDevedor (mistura-se letras maiúsculas e minúsculas)
123SALDO (começa por um dígito numérico)
```

Embora a maioria dos livros de lógica não façam distinção entre o uso de maiúsculas e minúsculas para efeito de estabelecimento de identificadores, aqui vai-se padronizar nossos identificadores sempre EM LETRAS MAIÚSCULAS. Isto pode incomodar um pouco no início, mas se revelará uma grande vantagem na análise de algoritmos feitos por terceiros, ou mesmo fde autoria própria, mas feitos há tempos.

O autor do algoritmo tem total autoridade para nomear seus identificadores como queira. Entretanto, há uma regra de bom senso implícita. Cada identificador deve ter um nome o mais próximo possível de sua função. Exemplo: Se precisarmos um identificador para representar uma somatória, poderemos chamá-lo de SOMA, SOMAT, SM, etc, mas nunca de ABC, ou XYZ, ou ainda AKLJHJKH.

Quanto ao tamanho do identificador, também só há uma regra: BOM SENSO. Nem tão pequeno que fique quase impossível identificá-lo pelo nome, nem tão longo que seja cansativo escrevê-lo. Exemplo. Ao gerar um identificador para conter a quantidade total de horas trabalhadas deve-se chamá-lo como QTHT, ou QTDHORAS, ou HORTRAB, ou similar, mas seria um exagero chamar o identificador de QUANTIDADE_DE_HORAS_TRABALHADAS_NO_MES.

Finalmente, deve-se discutir se as palavras que compõe o algoritmo devem ou não ser acentuadas. Parece uma discussão bizantina, mas não o é necessariamente. Se o leitor do algoritmo for um ser humano, pode-se acentuá-lo sem nenhum problema. Entretanto, se o algoritmo for – no futuro – processado por algum programa, há que se ter em mente, que em geral, para qualquer programa "á" é diferente de "a" e esta regra vale para todas as letras. Também "ç" é diferente de "c".

3.2 Variáveis

Ao identificador atribui-se um valor, que será usado nas computações futuras. Quando este valor pode variar ao longo do processamento, diz-se que o identificador representa uma variável. (No contrário, dir-se-ia que o identificador representa uma constante). Outra definição para variável é: "Um lugar onde se guarda uma valor".

Por norma do português estruturado, todas as variáveis devem ser declaradas antes de poderem ser utilizadas. Chama-se à atenção, pois esta não é uma regra obrigatória. Inúmeras linguagens de programação, permitem que se defina uma variável no instante em que ela passa a ser necessária e em qualquer ponto do programa. (Exemplo: BASIC, dbase, APL etc) Em portugol (e em Pascal, Cobol, Fortran, C, Java etc), isto não é possível. Assim, nossos programas terão dois blocos: O primeiro é o de definição de variáveis, e o segundo é o de procedimentos, sempre nesta ordem.

3.2.1 Tipos de variáveis

Existem cinco tipos de variáveis básicas em portugol: inteiro, real, caracter, lógico e cadeia.

Inteiro

É uma variável que pode conter números inteiros (positivos ou negativos, não importa). Exemplos: 5, 1009, -6730 etc. O conceito de inteiro é familiar a nós. Qualquer operação de contagem, pode ser estabelecida a partir do conjunto dos valores inteiros. Uma quantidade discreta [1](i.é. enumerável) sempre pode ser processada com base nos inteiros. Embora possam ter uma definição formal, para nós é suficiente a seguinte descrição: Uma variável inteira é a que pode armazenar qualquer um dos números do conjunto:

$-\infty, ... -(n+1), -n, ..., -2, -1, 0, 1, 2, ..., n, n+1, ..., +\infty$

Real

É outra variável numérica que pode conter qualquer número real, isto é, inteiro ou fracionário. Ex.: 1.5, -0.99, 1700,78 etc. Claramente salta aos olhos que armazenar uma variável real é mais "caro"(em tempo e em recursos) do que processar uma variável inteira. Aliás, esta é uma das razões porque existem variáveis inteiras. Outra é a característica da enumerabilidade que não está presente no conjunto dos números reais.

Caracter

Uma variável que pode conter um único caracter, como uma letra, ou um único dígito. O real conteúdo de uma variável caracter depende do código básico que está em uso. Mas, por enquanto, pode-se simplificar esta questão dizendo que qualquer caracter que esteja no teclado do computador pode ser colocado em uma variável caracter.

Cadeia ou string

É um tipo de variável formado por um ou mais de um caracteres. Por exemplo, um nome: "JOSE", ou uma cidade como "CURITIBA"etc. Ou seja, a diferença entre uma variável caracter e uma variável cadeia é que a primeira tem tamnho 1 (um único caracter) e a segunda tem tamanho maior que um, ou seja, vários caracteres.

[1] discreta = quantia que exprime valores inteiros, objetos, coisas, enfim grandezas não contínuas

Lógico ou booleano

O nome booleando vem de George Boole, um matemático. É uma variável que pode conter apenas 2 valores, conhecidos pelas palavras: VERDADEIRO e FALSO. Algumas linguagens não têm este tipo (C ou APL, por exemplo) usando artifícios para simular o conceito.

O tipo cadeia traz consigo uma pequena dificuldade, que é a necessidade de estabelecer - quando da definição da variável - qual o seu tamanho máximo, em número de caracteres. Esta atitude é necessária para informar a quem lê o algoritmo (seja um homem ou um computador) quanto de espaço reservar para conter esta variável.

Esta dificuldade não existe nas variáveis numéricas nem nas lógicas que tem tamanhos predeterminados e principalmente fixos.

Existe uma característica das linguagens de programação conhecida como "tipagem". Há linguagens de tipagem forte (por exemplo Pascal) e as há de tipagem fraca (por exemplo C). Este conceito tem a ver com restrições e verificações que a linguagem determina ou executa a cada comando. A tipagem forte torna as linguagens um pouco mais demoradas, mas evita erros cometidos pelo programador ao não verificar as operações que comandou. Este tipo de erro, de resto muitas vezes é difícil de localizar e corrigir, pois o sintoma que surge frequentemente é intermitente e nem sempre tem muito a ver com o real local onde o erro está acontecendo. Outra vertente de linguagens fracamente tipadas é quando a linguagem tem regras de conversão entre tipos diversos e elas são aplicadas antes de qualquer comando.

3.2.2 Código de caracteres

No momento de representar caracteres, aparece um problema: "Qual sua lei de formação e representação ?". Tal problema não aparece nos números: a matemática ajuda. Entretanto, não há uma aritmética para caracteres, e portanto precisa-se construir uma série de propriedades e relações entre os caracteres. Para não criar um novo conceito, vai-se aproveitar o código de representação padrão em uso nos computadores atuais. Trata-se do código ASCII (American standard Code for Information Interchange). A seguir, uma parte do próprio:

Código de caracteres Existem diversos códigos usados no mundo da computação. O mais usual é o código ASCII (American Standard Code for Intrechange of Informations), embora se usem também o BCD, EBCDIC e mais recentemente haja uma tendência no uso do UNICODE. Como exemplo, eis uma parte do código ASCII.

dec	carac	dec	carac	dec	carac	dec	carac	dec	carac	dec	carac
32	sp	48	0	64	@	80	P	96		112	p
33	!	49	1	65	A	81	Q	97	a	113	q
34	”	50	2	66	B	82	R	98	b	114	r
35	#	51	3	67	C	83	S	99	c	115	s
36	$	52	4	68	D	84	T	100	d	116	t
37	%	53	5	69	E	85	U	101	e	117	u
38	&	54	6	70	F	86	V	102	f	118	v
39	’	55	7	71	G	87	W	103	g	119	w
40	(	56	8	72	H	88	X	104	h	120	x
41	)	57	9	73	I	89	Y	105	i	121	y
42	*	58	:	74	J	90	Z	106	j	122	z
43	+	59	;	75	K	91	[	107	k	123	{
44	,	60	<	76	L	92	&	108	l	124	|
45	-	61	=	77	M	93	]	109	m	125	}
46	.	62	>	78	N	94	^	110	n	126	
47	/	63	?	79	O	95	_	111	o	127	

Esta seqüência ASCII, é muito importante pois permite estabelecer o atributo de ordem no universo dos caracteres. Olhando para ela, podemos afirmar, por exemplo, que o caractere ”a”é menor do que o caractere ”b”. Da mesma maneira, o ”a”minúsculo é maior do que o ”A”maiúsculo (aliás, isto vale para todas as letras), e qualquer dígito numérico é menor do que qualquer letra.

Um especial cuidado deve ser tomado, pois esta não é a única seqüência possível, e nem sequer a melhor. Cada máquina e cada implementação pode ter a sua. Alguns outros códigos: BRASCII (Código padrão BRASILEIRO para intercâmbio de informações), EBCDIC (Extended binary-coded decimal interchange coded), Z-Code etc.

Outra observação é que na tabela ASCII não existem caracteres acentuados. Duas conseqüências advém deste fato: nem sempre se podem usar tais caracteres e se eles forem usados, não se sabe qual a sua ordem no conjunto. Trocando em miúdos, se se solicitar a ordenação de uma lista de palavras contendo AMORA, ÂNCORA BROA, CAFE, ARROZ, CEREAL, não será nenhuma surpresa se a resposta for AMORA, ARROZ, BROA, CAFE, CEREAL e ÂNCORA.

EXERCÍCIO 3 Uma salada de frutas, produzida industrialmente leva maçã, banana, abacaxi e açúcar nas proporções de 1 Kg : 2 Kg : 0,5 Kg : 0,5 Kg respectivamente. Definir algoritmo que leia 4 quantidades representando os pesos disponíveis dos quatro ingredientes, calcule e escreva a quantidade de salada que poder ser construida, desprezando-se as quantidades que ultrapassam a proporção. Exemplos:

Se for lido	só serão usadas	logo total
10,10,10,10	5,10,2.5,2.5	20Kg
10,20,10,10	10,20,5,5	40Kg
10,10,1,1	2,4,1,1	8Kg

EXERCÍCIO 4 Imagine que você tem uma entrada com um par de números reais. Eles representam as dimensões x e y medidas em metros, de um retângulo. Escreva o algoritmo que leia esse par de números e escreva x, y e a área do retângulo de lados x e y, apenas se o perímetro deste mesmo retângulo for superior a 12.0 m.

EXERCÍCIO 5 Defina e escreva uma função que receba três números reais, encontre e devolva o maior deles.

EXERCÍCIO 6 Defina um algoritmo capaz de comparar dois números inteiros e positivos e imprimir a mensagem IGUAIS ou DIFERENTES em função dos valores de tais números. Eles serão considerados IGUAIS quando sua diferença for menor a 10% do maior número. Serão considerados diferentes quando sua diferença for maior ou igual a 10% do valor do maior. O algoritmo termina quando for lido um número negativo.

EXERCÍCIO 7 Defina e escreva um algoritmo que leia um número e determine se ele é divisível por 3. Se for o algoritmo deve imprimir a mensagem "E DIVISÍVEL", e se não for deve imprimir "NÃO É DIVISÍVEL".

EXERCÍCIO 8 Escrever um algoritmo que leia um conjunto de três variáveis que indicam respectivamente: Nome do funcionário (cadeia-30), número de horas (inteiro) e valor da hora trabalhada (real).

Se o número de horas é maior que 40, calcule o valor a pagar, sabendo que as horas extras são remuneradas a base de 1,5 x a hora normal. Se o salário for maior que R$ 650,00 retire 10% do salário para o governo (imposto). escreva nome, horas, valor da hora, valor a receber e valor do imposto. Dados terminam quando for lido um número de horas igual a zero.

Por exemplo,

Lido	Impresso
João,50,100	João,50,100,4950,550
Maria,30,10	Maria,30,10,300,0
XXX,0,0	fim.

EXERCÍCIO 9 O departamento de pessoal de uma empresa solicitou o desenvolvimento de um programa de computador capaz de calcular o desconto devido para a previdência. Assim, escreva um algoritmo capaz de calcular o desconto para o INSS, sujeito às seguintes regras:

- Inicialmente o programa deve ler o valor do salário de referência
- Depois deve ler uma série de dados compostos por RG e salário bruto.
- A série termina quando for lido um RG $= 0$.
- Para um salário bruto até 10 S.R. o desconto é 8% do salário bruto
- Para um salário entre 10,01 S.R. e 20 S.R. desconto é 9% do salário bruto
- Acima de 20 S.R. o desconto é de 10% do salário bruto.
- Ao final, imprimir o RG, o salário bruto, o desconto e o salário líquido.

EXERCÍCIO 10 Pretende-se informatizar um estacionamento. Nele existem 3 tipos de preços: 2 R $ por hora ou fração para carros pequenos, 2,50 R$/hora para carros médios e 3 R$/hora para carros grandes. Definir algoritmo que leia uma serie de informações formadas por: Tipo do carro: caracter; hora-entrada: inteiro, minuto-entrada: inteiro; hora-saida: inteiro, minuto-saida: inteiro. Para cada conjunto de informações lidas, o algoritmo deve imprimir a quantidade de dinheiro devida.

3.3 Comando de atribuição

Comando de atribuição é aquele que permite colocar/alterar valores de conteúdo em variáveis. Para tanto usaremos o símbolo ←. [2]

Quando dizemos expressão, podemos estar querendo dizer "valor" (que é a forma mais simples de expressão), ou simplesmente um conjunto de valores e operadores (que é a definição de expressão).

Exemplos:

A ← 1 (aqui, a expressão é apenas um valor)

B ← 1 + 2 (neste caso, a expressão é um conjunto de 2 valores e uma operação, no caso, a adição)

É importante notar, que a expressão que se encontra do lado direito da flecha de atribuição (←) deve ser compatível com o tipo de variável que foi definida. Em outras palavras, no comando A ← 1+1, a expressão 1+1 tem que ser compatível com a variável A, ou seja A deverá ter sido definida como inteira ou real, mas não como caracter, cadeia ou lógica.

Na atribuição de variáveis do tipo inteira, a expressão deverá ser uma expressão inteira, isto é, sem parte fracionária. Para variáveis do tipo real, a expressão deverá ser numérica sem nenhuma outra restrição. Esta é a única exceção à regra acima, pois podemos colocar um valor inteiro em uma variável real.

Na atribuição de variáveis cadeia, o valor deverá estar envolvido entre aspas simples ('). Ele não deverá ter um tamanho maior do que aquele estabelecido para a variável. Se isto acontecer, o valor será truncado. Entretanto, não é boa norma de programação usar esta facilidade (ou característica) pois ela dificulta a compreensão do algoritmo.

Finalmente, nas variáveis lógicas, deveremos usar as palavras VERDADEIRO e FALSO, ou quando não houver risco de confusão, podemos usar as abreviaturas F ou V.

Exemplos de atribuições:

variáveis inteiras

X ← 1
XAC ← 1 + 1 + 1
DEV ← 2000 × 23
N ← -1

variáveis reais

B ← 1.89
XAD ← 1 ÷ 3
N ← -1.0009
ALF ← 1000900
X ← 1

variáveis cadeia

NOME ← 'ALFREDO'
COD ← 'ABCDEabcde'
ALFA ← 'abcde12345'

variáveis lógicas

SIM ← VERDADEIRO
NÃO ← FALSO
VARLOG ← V
VARLOG2 ← F

[2] A questão do símbolo da atribuição é bem complicada. Poucos ambientes (como o APL) dispõe do símbolo nativo ←. Os outros improvisam, com os símbolos := do Pascal, ou simplesmente = do C e Java. No primeiro caso, são 2 símbolos, ao invés de um único, e no segundo surge a confusão entre A=1 (significando A recebe 1) e A=1 (significando a pergunta: A é igual a 1 ?). Daí o C e o Java fazerem a pergunta com ==. Não há uma saída fácil.

variáveis caracter
NOME ← 'A'
XUNXO ← 'X'
NUM ← '1'
AST ← '*'

Duas perguntas a responder:

- A ← '1'
  A é variável numérica ou caracter ?
- B ← 'V'
  B é variável lógica ou cadeia ?

Como regra geral, sempre que houver mistura entre elementos inteiros e reais, nas operações +, − e × o resultado será real.

Exemplos de uso de operadores em expressões numéricas
A ← 1 × 1,78 ÷ 3
B ← trunc(3.1416)
C ← abs (-6 + trunc (3.4))
D ← 1 + 10 + 100 - 1000
etc

Uma observação final e bem importante é que dentro de um mesmo algoritmo uma variável pode ter diversos (isto é, mais de um) comandos de assinalamento. Isto não é erro, e ao contrário é bem freqüente. Nestes casos, a variável terá o valor que foi nela colocado por último. Por exemplo, em

```
1: A ← 1
2: ...
3: A ← 10
4: ...
5: escreva (A)
6: ...
```

Se nos comandos representados com ... não houver nenhuma alteração na variável A, o resultado impresso ao final será 10 (e não 1).

3.4 Expressões

Expressões são conjuntos de variáveis, valores, operações e eventualmente parênteses, que demandam algum tipo de computação determinada na expressão e produzem um resultado. As expressões se classificam em geral devido ao tipo de operações que existem dentro da expressão, podendo ser aritméticas, condicionais, lógicas e de caracteres.

3.4.1 Aritméticas

As expressões aritméticas são aquelas que envolvem variáveis numéricas, valores idem e as operações aritméticas, dando como resultado um número (que obviamente pode ser real ou inteiro).

Notação Exponencial Para os casos em que seja necessário representar números muito grandes ou muito pequenos, pode-se lançar mão da notação exponencial, que será escrita de acordo com a seguinte regra: A primeira parte do número será escrita como um número convencional, podendo ter o ponto decimal (constante ponto flutuante) ou não (constante inteira). Depois, vem a letra maiúscula "E", indicando "elevado a". Um segundo número inteiro, positivo ou negativo indicando a potência de 10 à qual a primeira parte do número está elevada.

Por exemplo:

o número	deve ser entendido como
3E4	3×10^4 ou 3×10000 ou 30000
6.3E-2	6.3×10^{-2} ou 6.3×0.01 ou 0.063
-4.22E6	-4.22×10^6 ou -4.22×1000000 ou -4220000

Operações usuais

Função	Formato	Objetivo	Tipos - operandos	Tipo do resultado	Exemplo
Adição	A + B	Adicionar A e B	inteiro ou real	Se A e B são inteiros, A+B é inteiro. Se pelo menos um dos dois é real, o resultado é real	3 + 7 é 10
Subtração	A - B	Subtrair B de A	inteiro ou real	idem ao anterior	5 - 2 é 3
Multiplicação	A × B	O produto de A e B	inteiro ou real	idem ao anterior	3×12 é 36
Divisão real	A ÷ B	O quociente de A por B	inteiro ou real	real	10÷2 é 5
Potência	A^B	A elevado à B	inteiro ou real	real	2^5 é 32
Absoluto	abs(A)	O valor absoluto de A	inteiro ou real	O mesmo tipo de A	abs(4) é 4
Seno	Sin(A)	seno de A radianos	inteiro ou real	real	sin(3.14...) é 1
Cosseno	cos(A)	cosseno de A radianos	inteiro ou real	real	cos(3.14...) é 0
Inteiro	trunc(A)	A parte inteira de A	real	inteiro	trunc(2.5)é 2
Fracionário	frac(A)	A parte fracionária de A	real	real	frac(2.5) é .5
Exponencial	exp(A)	E (2.718...) elevado a A	inteiro ou real	real	exp(1) é 2.718
Log Natural	log(A)	Logaritmo natural de A	inteiro ou real	real	log(1) é 0
Arredondamento	round(A)	Arredonda A para o inteiro mais próximo	real	inteiro	round(3.6) é 4
Raiz quadrada	sqrt(A)	Raiz quadrada de A	inteiro ou real, mas A (0	real	sqrt(4) é 2
Quadrado	sqr(A)	O quadrado de A	inteiro ou real	O mesmo de A	sqr(4) é 16
Somar 1	A++	A variável A é incrementada de 1 unidade	inteira ou real	O mesmo de A	Se A é 10, A++ coloca 11 em A.

Subtrair 1	A−−	A variável A é decrementada de 1 unidade	inteira ou real	O mesmo de A	Se A é 10, A−− coloca 9 em A.
Divisão inteira	A div B	o quociente inteiro da divisão de A por B	inteiros	inteiro	4 div 3 é 1
Resto	A mod B	O resto da divisão inteira de A por B	inteiros	inteiro	10 mod 7 é 3

Prioridades Na execução de um comando complexo, podemos encontrar duas ou mais operações numéricas uma ao lado da outra, e neste caso, pode surgir a pergunta: Qual realizar antes ?

Exemplo: $2 + 3 \times 4$ é igual a 14 ou 20 ? Afinal, $2 + (3 \times 4)$ é 14, e$(2 + 3) \times 4$ é 20.

Embora na matemática tradicional já se tenha uma lista de prioridades que deve ser obedecida, aqui no português estruturado também há uma lista, que se resume a:

1. Primeira Prioridade: Parênteses
2. Prioridade: Funções
3. Prioridade: Menos unário (oposto)
4. Prioridade: Potenciação
5. Prioridade: Multiplicação e divisão
6. Prioridade: Adição e subtração
7. Prioridade: Comparações $(>, <, =, \geq, \leq, \neq)$
8. Prioridade: Operadores lógicos, $\sim, \wedge, \vee$.

Se houverem duas operações de mesma prioridade, as mesmas serão realizadas em qualquer direção convencionada, por exemplo, da esquerda para a direita.

É difícil decorar esta tabela, além do que, algumas linguagens de computador podem implementar tabelas ligeiramente diferentes desta. Assim, resta como sugestão ao aluno:

- Definir linhas pequenas (clareza e auto-documentação). Se for necessário, quebrar um comando complexo em vários simples.
- Usar e abusar de parênteses, ainda que redundantes, pois eles reforçam a ordem de execução e desobrigam de conhecer e consultar a tabela de prioridades.

Round e Trunc

Arredondar um número é converter um real em inteiro aproximando-o do inteiro mais próximo. A regra básica é que números cuja parte fracionária é menor do que 0.5 são trazidos para o inteiro menor e números cuja parte fracionária é igual ou maior a 0.5 são arredondados para mais.

Já o truncamento simplesmente despreza a parte fracionário, trazendo o resultado sempre para o inteiro menor ou igual ao número dado.

As funções round e trunc operam sobre números positivos e negativos. Quando for este último caso, a definição continua prevalecendo, o resultado é encontrado como se o operando fosse positivo, e a seguir o sinal de menos é colocado sobre o resultado. Acompanhe nos exemplos:

trunc(1.8) é 1
trunc(-1.8) é -1
round(1.8) é 2
round(-1.8) é -2

Duas considerações finais:

- Podemos considerar a existência no português estruturado de uma constante chamada MAXINT, contendo sempre o MAIOR valor possível de ser representado em uma variável inteira. Em outras palavras, desde que A seja inteira, quando compararmos A com MAXINT, A será menor ou no máximo igual a MAXINT, nunca maior. Para todos os efeitos práticos esse é o nosso infinito (∞).

- Podem ser introduzidos pela pessoa que está escrevendo o algoritmo quaisquer novos operadores matemáticos, desde que estes não sejam ambíguos e sejam facilmente entendíveis, a fim de poderem ser seguidos mais tarde.

Div e Mod

Existem duas importantes funções matemáticas, a quem chamar-se-á div e mod e que são usadas em matemática inteira. Ambas atuam sobre números inteiros e dão como resultado também números inteiros. Div devolve o resultado inteiro de uma divisão. Já mod devolve o resto inteiro da divisão de dois números. Em ambos operadores, o segundo termo tem que ser diferente de zero.

Em resumo, A div B, é o resultado inteiro da divisão de A por B.
Exemplo:
10 div 3 é 3
5 div 3 é 1
890 div 100 é 8
1000 div 1 é 1000
A mod B é o resto inteiro da divisão de A por B.
Exemplo
10 mod 3 é 1
5 mod 3 é 2
890 mod 100 é 90
1000 mod 1 é 0
As duas funções se completam na expressão $I \; mod \; J = I - (I \; div \; J) \times J$.

```
Exemplo: 20 mod 6 = 20 - (20 div 6) * 6
                  2 = 20 - 3 * 6
                  2 = 20 - 18
                  2 = 2                                    c.q.d.
```

Um especial cuidado deve ser tomado quando o primeiro operando de div for menor que o segundo, por exemplo em 2 div 3. Neste caso, a resposta é 0. Esta regra segue válida se o primeiro número é 0. Por exemplo, em 0 div 5, a resposta segue sendo zero e não há erro nesta chamada.

A mesma consideração pode ser feita na função mod. 2 mod 3 é 2 e 0 mod 5 é zero, sem erro nos dois casos.

suc, pred, ord e chr

Para os tipos ordinais pré-definidos (caracter, inteiro e lógico), existem algumas funções que podem ser usadas. Para cada tipo de operandos os universos são:

inteiros neste caso, a seqüência de ordenação é aquela dos números inteiros da matemática:
... ∞, ..., -2, -1, 0, 1, 2, ... $+\infty$.

caracteres aqui a seqüência é dada pelo código nativo do ambiente, no caso o ASCII.

booleanos a seqüência é FALSO, VERDADEIRO.

Ord A Função ord devolve o ordinal do operando dentro do seu universo original. Assim, a ord de um operando caracter, devolve o ordinal dentro do código ASCII. A ord de um lógico, considera o universo de 2 valores (V e F). e a ord de um número inteiro, é o próprio número inteiro. Embora atue sobre os 3 tipos acima, na verdade ela se aplica verdadeiramente aos caracteres.

Por exemplo: ORD(-3) é -3.
ord('a') é 97.
ord(FALSO) é 0.
ord('7') é 55,
ord('W') é 87,
ord('z') é 122,

No caso dos caracteres, a resposta à função ord se encontra na tabela vista anteriormente. Consultamos o operando de ord na coluna referente a ASCII, e a resposta de ord é o número decimal que estiver na mesma linha.

Chr A função chr é a função inversa da função ord, porém só funciona para caracteres. Dado um número inteiro entre 0 e 255, para o código ASCII e 0 e 64536 para o código UNICODE, a função chr deste número devolve o caractere correspondente a ele. Para simular como o portugol faz isto, dado um número, procura-se na tabela em qual linha ele ocorre sob a coluna decimal, e a seguir responde-se com o caractere ASCII correspondente à mesma linha.

Exemplo, CHR(97) é "a", etc.

Suc Trata-se da função sucessora, que nos devolve o próximo valor ao do operando considerado o seu universo original. Esta função só se aplica a operandos ordinais pré-definidos (inteiro, caracter e lógico). Se o operando for inteiro, a resposta também será. Se o operando for lógico, a resposta também será. Da mesma maneira se o operando for caracter.

suc('a') é "b"
suc(FALSO) é VERDADEIRO
suc(23) é 24, e assim por diante
suc('1') é '2',
CUIDADO ====> succ('9') é ':' e succ(9) é 10.

Uma aplicação interessante para esta função é a substituição do incremento de variáveis.

Por exemplo, em vez de fazer:
I $\leftarrow$ I + 1;
podemos fazer
I $\leftarrow$ suc(I);

Pred Esta função devolve o predecessor, e também só se aplica a ordinais pré-definidos. Também (tal como no suc) o tipo da resposta é o mesmo tipo do operando. Exemplos:
pred('b') é "a"
pred(FALSO) é VERDADEIRO
pred(23) é 22, e assim por diante.

3.4.2 Relacionais

As expressões relacionais são as que envolvem os operadores $=$, $\neq$, $>$, $\geq$, $<$ e $\leq$. Este operadores visam a estabelecer se uma dada proposição é falsa ou verdadeira. É comum em qualquer linguagem de programação, comparar-se 2 valores, perguntando, por exemplo, se o primeiro é maior do que o segundo. A resposta, na forma lógica, dirá se a afirmação é ou não é verdade.

São eles: igual ($=$), diferente ($\neq$), maior ($>$), maior ou igual ($\geq$), menor ($<$) e menor ou igual ($\leq$).

Estes operadores sempre relacionam duas variáveis ou constantes de tipos compatíveis. Por exemplo ao perguntar $3 > 5$?, a resposta será "falso".

Note que não é permitido (ou seja gera-se um erro), misturar valores de tipos distintos. Então, a comparação 'A' $\neq$ 3, embora logicamente pudesse estar correta (ou seja a letra A não é igual ao número 3), dá erro em qualquer linguagem de programação e portanto está proibida de ser usada na construção de algoritmos.

A exceção à regra acima é quando se comparam dois números, sendo um deles do tipo inteiro e outro do tipo real. Embora de tipos diferentes (inteiro e real), a matemática permite fazer essa comparação, já que ambos são números.

Tem-se então que os operandos poderão ser de qualquer tipo, desde que compatíveis, mas a resposta sempre será do tipo lógico.

Função	Formato	Objetivo	Operando	Resultado	Exemplo
Igual	A = B	Comparar o conteúdo de A e de B	ambos inteiros, reals ou alfanuméricos	.V. se A = B e .F. se A $\neq$ B.	3 = 4 é .F., 66.0 = 66 é .V., "AB"= "ab"é .F.
Diferente	A $\neq$ B	igual ao anterior	igual ao anterior	.V. se A $\neq$ B e .F. se A = B.	3 $\neq$ 4 é .V., 66.0 $\neq$ 66 é .F. "AB"$\neq$ "AB"é .F.
Maior	A > B	igual ao anterior	ambos inteiros ou reais	.V. se A > B e .F. se A $\leq$ B.	5 > 2 é .V. 2 > 5 é .F.
Maior ou igual	A $\geq$ B	igual ao anterior	igual ao anterior	.V. se A $\geq$ B e .F. se A < B.	5 $\geq$ 2 é .V. 2 $\geq$ 2 é .V.
Menor	A < B	igual ao anterior	igual ao anterior	.V. se A < B e .F. se A $\geq$ B.	5 < 2 é .F. 2 < 5 é .V.
Menor ou igual	A $\leq$ B	igual ao anterior	igual ao anterior	.V. se A $\leq$ B e .F. se A > B.	2 $\leq$ 5 é .V. 2 $\leq$ 2 é .V.

3.4.3 Lógicas

As expressões lógicas envolvem valores lógicos (verdadeiro e falso) conectados por operadores lógicos, que são 3: "E" ($\wedge$), "OU" ($\vee$) e "NÃO" ($\neg$ ou $\sim$). Estes operadores

destinam-se as operações lógicas entre operandos. Eles atuam sobre os valores V (verdade) e F (falso). Acompanhe a seguir as tabelas verdade:

Em termos verbais a expressão A $\wedge$ B será verdadeira quando A e B forem verdadeiros e será falsa senão.

Vendo na tabela a seguir:

$\wedge$	verdadeiro	falso
verdadeiro	verdadeiro	falso
falso	falso	falso

A expressão A $\vee$ B será verdadeira quando A for verdadeiro ou B for verdadeiro ou ainda quando ambos forem verdadeiros. A expressão só será falsa quando A e B forem falsos. Veja:

$\vee$	verdadeiro	falso
verdadeiro	verdadeiro	verdadeiro
falso	verdadeiro	falso

Finalmente, a expressão $\sim$ A (lida como NÃO A), devolve o valor lógico oposto ao de A. Então $\sim$ verdadeiro é falso e $\sim$ falso é verdadeiro.

$\sim$ ou $\neg$	
verdadeiro	falso
falso	verdadeiro

Função	Formato	Objetivo	Operando	Resultado	Exemplo
E-lógico	A $\wedge$ B	Realizar a operação E-lógico	ambos lógicos	.V. se A e B são .V., .F. em caso contrário	.V. $\wedge$ .V. é .V. .V. $\wedge$ .F. é .F. .F. $\wedge$ .V. é .F. .F. $\wedge$ .F. é .F.
OU-lógico	A $\vee$ B	Realizar a operação OU-lógico	ambos lógicos	.V. se A ou B são .V., .F. em caso contrário	.V. $\vee$ .V. é .V. .V. $\vee$ .F. é .V. .F. $\vee$ .V. é .V. .F. $\vee$ .F. é .F.
NÃO-lógico	$\sim$ A ou $\neg$ A	Nega logicamente A	lógico	.V. se A é .F. .F. se A é .V.	$\sim$ .F. é .V. $\sim$ .V. é .F.

As palavras “verdadeiro” e “falso” bem como os símbolos ”V”e ”F” podem ser considerados constantes lógicas também chamadas booleanas e podem ser empregados livremente na construção dos algoritmos. As duas constantes, formam um conjunto ordenado, e podemos dizer que o F (falso) precede o V (verdadeiro). Como uma ajuda, podemos associar o Falso ao zero, e o Verdadeiro ao 1. Com isto todas as relações numéricas entre 0 e 1 continuam verdadeiras entre FALSO e VERDADEIRO.

A grande importância dos valores lógicos em portugol, decorre do fato de que qualquer comparação usando os operadores relacionais **sempre devolve um lógico**.

A importância das expressões lógicas é que elas permitem conectar duas ou mais expressões relacionais, formando uma nova e maior expressão relacional. Veja-se nos exemplos:

expressão lógica	será verdadeira quando...
$(A \vee B) \wedge C$	A e C forem verdadeira ou B e C forem verdadeiras
$\sim D$	D for falsa
$(E \wedge F) \vee G$	E e F forem verdadeiras ou então se G for verdadeiro

EXERCÍCIO 11 Suponha que você vai escrever um algoritmo em portugol para resolver as raizes de $ax^2+bx+c=0$. Quais variáveis seriam necessárias e como elas seriam definidas ?

EXERCÍCIO 12 Quando criança Gauss resolveu a somatória 1 + 2 + 3 + ... + 4998 + 4999 + 5000, de uma forma brilhante, intuindo toda a teoria de progressões. Se você tivesse que resolver o problema de Gauss, quais variáveis definiria, e como elas seriam definidas ?

EXERCÍCIO 13 Para cada uma das fórmulas a seguir, supor que será criado um algoritmo, no qual você deverá dizer quais variáveis serão necessárias e como serão definidas:

1. perímetro de um retângulo
2. cálculo do terceiro ângulo de um triângulo, dados 2 ângulos.
3. cálculo da quantidade de azulejos a serem colocados em um banheiro.
4. Volume de uma lata de leite
5. Preço a pagar pelo estacionamento de um veículo.

EXERCÍCIO 14 Informe qual o resultado esperado para a variável VAR1
VAR1 $\leftarrow$ (1 + (2 $\times$ (3 + 1) $\div$ 2)) - 2
VAR1 $\leftarrow$ sqr(3)
VAR1 $\leftarrow$ 2 $\times$ trunc(1.999)
VAR1 $\leftarrow$ abs(-(3 - 4))
VAR1 $\leftarrow$ trunc(1.5) - frac(1.5)
VAR1 $\leftarrow$ sen(3.1415$\div$4) $\div$ cos(3.1415$\div$4)
VAR1 $\leftarrow$ 2 $\div$ (2 $\times$ 3)

EXERCÍCIO 15 Nos exercícios a seguir, o aluno deve:

- Achar o resultado da expressão
- Descobrir-lhe o tipo
- Definir uma variável para conter este resultado.

Exemplo:
1 + 1 + 2, terá como resposta: 4, inteiro, SOM:inteiro; SOM $\leftarrow$ 4

1. 1 $\div$ 2
2. 1 + 3 $\div$ 1
3. trunc (3.5 + 2)
4. sen (1)
5. sen (0.33333) + cos (0.33333)
6. 1 $\times$ 2 $\times$ 3 $\times$ 4 $\times$ 5 $\times$ -6

7. $2 > 3$
8. $1 + 3 < 4$
9. $5 = 4 + 1$

EXERCÍCIO 16 Na série de exercícios a seguir, considere o seguinte conjunto de variáveis:
X ← 5 (inteiro)
Y ← 1 (inteiro)
AA ← 0.5 (real)
BB ← 4.9 (real)
CC ← 3 (real)
Para cada expressão pedida, calcule o resultado e determine o tipo (inteiro ou real) do resultado.

1. (AA - X) ÷ CC
2. CC + abs(trunc(BB))
3. frac(AA) - AA
4. X + Y + CC
5. sqrt(trunc(BB-AA))
6. (round(AA+BB)) mod CC
7. 1 + sqr(X - Y)
8. 1 + sqr(X) - Y
9. 5 × X × round(BB)
10. AA - abs(BB)
11. abs (AA - BB)
12. abs (AA) - abs (BB)
13. frac (int (BB))
14. (X + Y) ÷ AA
15. trunc (Y + AA)
16. X + Y + AA + CC
17. X + CC
18. X + Y
19. cos (int(AA))
20. round (AA)
21. round (BB + AA)
22. trunc (AA - BB)

EXERCÍCIO 17 Quanto é 400 mod 51
30 mod 7
(5 mod 4) + (22 div 10) + (3 mod 2)
4376 mod 10
4376 mod 100
4376 mod 1000
4376 mod 10000
10 mod 4
10 div 4
cos (9 div 11)
trunc (abs (round (6 div 6)))
(1 + 5) mod 3
1 + (5 mod 3)
(2 × 4) mod 2
2 × (4 mod 2)
10000 mod 1
5 div 0

EXERCÍCIO 18 Informe qual o valor final para VAR3
VAR3 ← (3 ≥ 2) ∨ (1 = 3)
VAR3 ← (1 + 1) = (3 - 1)
VAR3 ← (trunc(1.5 × 4) > 6) ∧ (1 ≠ 2)

EXERCÍCIO 19 Informe qual o valor da variável VAR2
VAR2 ← VERDADEIRO
VAR2 ← ∼ FALSO
VAR2 ← ∼ ∼ VERDADEIRO ("...Eu não disse que nunca viajaria...")
VAR2 ← FALSO ∨ FALSO
VAR2 ← VERDADEIRO ∧ (∼ VERDADEIRO)
VAR2 ← (∼ FALSO) ∨ (∼ VERDADEIRO)
VAR2 ← (FALSO OU VERDADEIRO) ∧ (∼ VERDADEIRO)
VAR2 ← FALSO ∧ FALSO
VAR2 ← FALSO ∨ FALSO
VAR2 ← VERDADEIRO ∧ (∼ VERDADEIRO)
VAR2 ← VERDADEIRO ∧ VERDADEIRO

EXERCÍCIO 20 Informe qual o valor para a variável VAR4.
VAR4 ← ((frac(0.999) - 1) > 0
VAR4 ← 1 = 2
VAR4 ← 1 + 2 = 2
VAR4 ← 0.5 ≥ 2.5 - trunc(2.9000)
VAR4 ← 3 > 1 + 1
VAR4 ← 1 + 3 × 4
VAR4 ← 2 × sqr(3 × 2 + 1)
VAR4 ← 1 × 2 × 3 × 4 × 0 + 1
VAR4 ← ((2 × 2) + 2) × 3
VAR4 ← 33 > (32 + trunc(2.45) - frac(0.5))
VAR4 ← 0.20 - 0.5
VAR4 ← 2 × 3 × 4 + 2 × 3 × 4
VAR4 ← (4 × (2 ÷ 2 × 2) ≥ 3.5) ∧ (1 = 3 × 0)
VAR4 ← (trunc (sen (3.14/2)) = 1) ∧ VERDADEIRO
VAR4 ← FALSO ∨ ((1 + 2) > (6 div 3))

VAR4 $\leftarrow$ (5 div 2) $>$ (trunc (3.5) + frac(3.5))
VAR4 $\leftarrow$ 7 $\div$ 2 + 2 $>$ 0
VAR4 $\leftarrow$ 5 + 5 - 5
VAR4 $\leftarrow$ (10 + frac(0) + trunc(0) - 10) $>$ 5
VAR4 $\leftarrow$ 5 $>$ 6 div 3 + 2 mod 1 + 0 mod 6
VAR4 $\leftarrow$ 2 $\times$ 8 $\div$ 4 $\div$ 2
VAR4 $\leftarrow$ (1 + trunc (13.5) $\times$ 2) $\times$ (1 $\div$ (2 + 1))

Capítulo 4

Comandos

4.1 Visão Top down e Bottom up

Estes termos, consagrados no jargão da informática, significam maneiras de atacar e resolver um problema em computador. A maneira *top down*, que pode ser traduzido como "de cima para baixo", pressupõe estudar o problema como um todo, e depois ir realizando a montagem do esquema completo, através da execução das partes, mas sem nunca esquecer o modelo completo.

A visão *bottom up*, parte da construção individual de todos os elementos, que posteriormente são juntados e testados. É possível fazer sistemas de computador usando as duas técnicas.

Fazendo uma analogia (meio mambembe, mas vá lá) com a montagem de um carro, na visão *top down* a primeira coisa a fazer seria juntar a carroceria e as rodas, de maneira que o carro conseguisse se mover. Depois, os diversos sistemas e componentes iriam sendo instalados peça a peça no carro e a cada etapa o carro continuaria a andar em suas próprias rodas. A montagem terminaria quando o carro andasse sozinho.

Já na visão *bottom up* cada um dos sistemas (motor, bancos, instrumentação, ...) seria montado isoladamente, testado e em caso de sucesso levado aonde o carro está sendo construído e aí posto em conexão com os outros sistemas do carro. O processo também termina quando o carro sair andando.

O exemplo não é dos melhores, porque um carro é diferente de um programa de computador. Lá se lida com coisas físicas que ocupam espaço e pesam para ser carregadas. Aqui se fala de entidades abstratas (dados e programas). Outro exemplo, este talvez algo melhor é a proposta de escrever um livro. Antes de tudo há que se ter o plano completo da obra, mas depois é possível visualizar a construção *top down* e a construção *bottom up*.

4.2 Seqüência de execução

Vale lembrar que a menos que o comando em questão determine outro caminho, os comandos dentro de um algoritmo vão sendo executado seqüencialmente, começando no primeiro e terminando no último, e sempre esperando terminar este para começar o próximo. Diz-se nestes casos que o fluxo cai por decantação. Assim, por exemplo, na seqüência

1: $A \leftarrow 1$
2: $B \leftarrow 2 + A$

Pode-se afirmar com certeza que primeiro será executado o comando 1. Depois dele é que será executado o comando 2, e aí o valor da variável A já estará estabelecido corretamente.

4.3 Entrada/Saída

A maioria dos algoritmos necessitará receber dados externos, e em algum momento do seu processamento precisará comunicar respostas. Para resolver estes dois problemas existem os chamados comandos de entrada/saída. É através deles que o algoritmo se comunica com o mundo externo.

Desprezando as complexidades inerentes a um processo de transferência de dados em um computador real – eis aí a principal vantagem do portugol –, os comandos que se usarão são dois: leia e escreva. O verbo ou comando "leia" significará uma entrada de dados externos para dentro do algoritmo. A forma desta entrada, isto é se os dados vão entrar pelo teclado, vão ser lidos do disco ou virão por uma linha telefônica é questão que não interessa agora. Nos próximos anos do curso, haverá muitas horas de aula sobre estas questões. Agora só interessa o fato de que o dado entra para o algoritmo e pronto.

Da mesma forma, para a saída dos dados, usar-se-á o comando escreva, que também não significa obrigatoriamente impressão em papel. Estes dados podem estar saindo no vídeo, em um plotter, em um disco, linha telefônica etc.

O formato destes comandos é:

```
leia (nome1, [nome2, ...])
```

Ao lado do verbo leia, e entre parênteses, deve-se escrever as variáveis que deverão ter seus valores fornecidos neste comando. Os identificadores citados já deverão ter sido definidos anteriormente no programa. Em geral (regra heurística) dados que vão ser lidos não precisam ser inicializados antes.

O comando de impressão é:

```
escreva ([mensagem] nome1)
```

Este comando externa um resultado disponível a quem está operando o algoritmo. É claro que os valores a imprimir terão que ter sido inicializados e/ou calculados antes da impressão. As variáveis a escrever tem seu nome citado entre parênteses.

Exemplos de leitura:
VALOR1, VALOR2: inteiro
leia (VALOR1, VALOR2)
De impressão
VALOR1: inteiro
VALOR1 ← ....
escreva ("O valor obtido é ", VALOR1)

Na impressão é comum se colocar constantes alfanuméricas, mensagens ou textos explicativos antes ou depois dos dados para clarear e facilitar a leitura humana posterior. Por exemplo, ao calcular uma raiz na variável RZ, é muito melhor escrever "escreva ('A raiz e',RZ)", do que simplesmente "escreva (RZ)". Neste último caso, o dado ficaria perdido (no vídeo, no papel, ...) sem que o operador soubesse o que era aquilo. Identicamente, para imprimir uma área, é melhor fazer "escreva (AR,'m2')", do que fazer só "escreva(AR)".

4.3.1 Comando de Entrada

Permite a introdução de valores para variáveis previamente definidas.

```
leia (<nome-1> [,<nome-2>,...])
```

Por exemplo,

```
1: A, B : inteiro
2: RAIZ : real
3: leia (A, B, RAIZ)
```

4.3.2 Comando de Saída

Permite a geração de resultados por parte do algoritmo. Permite mesclar variáveis e expressões. Em geral, as expressões alfanuméricas incluídas no comando servem para auxiliar na interpretação dos resultados gerados.

```
escreva (<nome-1> | <expressão-1> [,<nome> | <expressão> ...])
```

Por exemplo,

```
1: escreva ("a raiz procurada e ", RAIZ)
```

4.4 O comando alternativo

Como já se disse este é o comando que permite versatilidade e generalidade aos algoritmos. Ao permitir-se modificar caminhos tendo em vista certas condições é que o algoritmo habilita o computador a "pensar" (as aspas, por favor).

4.4.1 Alternativa simples

Existem oportunidades em que um determinado comando dentro de um algoritmo só deve ser executado se certas condições o permitirem. Por exemplo, ao olhar se uma pessoa está quites com as suas obrigações militares, isto deve ser feito apenas se o sexo da pessoa é masculino. Em nosso país, mulheres estão desobrigadas de tais tratativas.

O comando que identifica uma ação condicional é o comando "SE", que tem o seguinte formato:

```
se <condição> então
   ação
fimse
```

A condição é uma expressão cujo resultado final é do tipo lógico, isto é VERDADEIRO ou FALSO. A ação pode ser um comando simples ou uma seqüência de comandos.

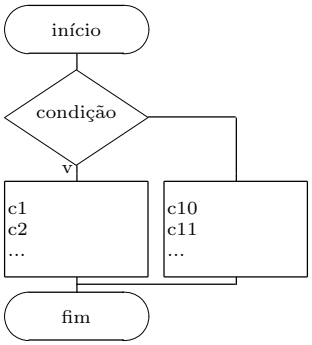
Exemplo:

```
1: se DELTA < 0 então
2:    escreva ("não é possível extrair esta raiz");
3: fimse
```

Para efeito de clareza na leitura de um algoritmo, costuma-se identar os comandos subordinados a uma condição. Esta é a razão pela qual se deixou 3 espaços em banco ao início do comando escreva no exemplo acima.

A margem deve ser trazida ao que era antes do comando "se", apenas quando se escrever o fimse correspondente.


```
1: se condição então
2:     c1
3:     c2
4:     ...
5: senão
6:     c10
7:     c11
8:     ...
9: fimse
```



4.4.2 Alternativa composta

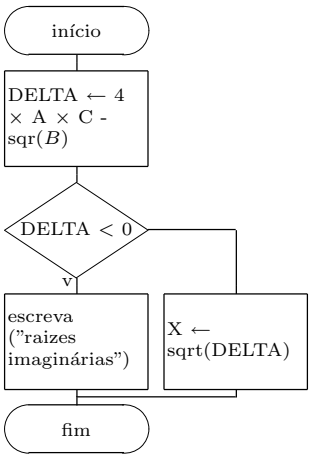
É uma extensão da alternativa simples. Neste caso podemos determinar o que fazer se a condição verdadeira, e o que fazer se a condição for falsa. O formato deste comando é (à esquerda em linguagem algorítmica, portugol, e à direita em fluxograma.

Se a condição estabelecida é verdadeira, são executados os comandos c1, c2, ... e não são executados os comandos c10, c11.... Se a condição é falsa, são executados os comandos c10, c11, ..., mas não os primeiros.

Neste caso a identação também é importante. Os comandos "se", "senão" e "fimse" começam na margem corrente. Todos os demais comandos internos a este deixam uma identação de 3 caracteres.

Exemplo:

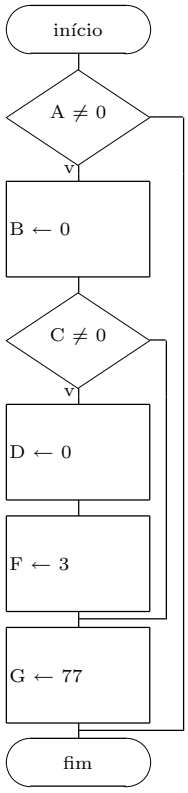
```
1: DELTA ← 4 × A × C - sqr(B)
2: se DELTA < 0 então
3:     escreva ("raizes imaginárias")
4: senão
5:     X ← sqrt(DELTA)
6: fimse
```



4.4.3 Alternativas aninhadas

Nada impede que exista uma condição dentro de outra, (regra da programação estruturada) e assim por diante. Nestes momentos a identação é mais importante ainda. Repare no exemplo a seguir:

Para ver a vantagem da identação analisar-se-á exatamente o mesmo código, porém escrito sem este recurso:




```
1: se A ≠ 0 então
2:    B ← 0
3:    se C ≠ 0 então
4:       D ← 0
5:       F ← 3
6:    fimse
7:    G ← 77
8: fimse
```

```
1: se A ≠ 0
2: B ← 0
3: C ≠ 0
4: D ← 0
5: F ← 3
6: fimse
7: G ← 77
8: fimse
```

EXERCÍCIO 21 Reescreva as condições acima usando a condição de igual.

Cada **se** tem que ter um **fimse** correspondente. Ao percorrer o fluxo, encontrando um **fimse** encerra-se o último **se** aberto. Esta regra é importante para determinar o fim de SE's aninhados.

Ao examinar um comando **se**, deve-se agir da seguinte forma:

Se a condição que acompanha o **se** for verdadeira, os comandos internos ao **se** devem ser executados. Se, ao contrário, a condição não for verdadeira, então, todos os comandos seguintes devem ser pulados até ser encontrado o comando **fimse** correspondente.

Agora é hora de voltar um pouco na teoria e relembrar o conceito das operações com operadores lógicos (VERDADEIRO e FALSO). Tais operações eram: $\wedge$, $\vee$ e $\sim$. Usando-as, em conjunto com o comando **se** simples ou composto, podemos criar trechos de algoritmo muito ricos. Veja-se alguns exemplos:

Definir se um valor esta compreendido entre 10 e 35, inclusive:

```
1: se VALOR > 9 ∧ VALOR < 36 então
```



```
2:    ... valor OK ...
3: senão
4:    ... valor ERRADO ...
5: fimse
```

Definir se um valor numérico representativo de um mês, está correto

```
1: se MES > 0 ∧ MES < 13 então
2:    ... mes OK ...
3: senão
4:    ... mes ERRADO ...
5: fimse
```

Um certo código pode assumir os seguintes valores: 10, 15, 17, 18 e 30. Testar se ele está ou não correto.

```
1: se COD = 10 ∨ COD = 15 ∨ COD = 17 ∨ COD = 18 ∨ COD = 30 então
2:    ... código OK ...
3: senão
4:    ... código ERRADO ...
5: fimse
```

As vezes é mais fácil organizar a saída correta através do SENÃO e não do ENTÃO, o que inverte o comando. Vejamos um exemplo.

Um indicador estar errado, se assumir os valores: 1, 4, 5, 6, 7, ou 9. Organizar o comando:

```
1: se IND = 1 ∨ IND = 4 ∨ IND = 5 ∨ IND = 6 ∨ IND = 7 ∨ IND = 9 então
2:    ... indicador ERRADO ...
3: senão
4:    ... indicador CERTO ...
5: fimse
```

Se entretanto, quiséssemos não inverter as saidas, precisaríamos negar as condições. Atente-se a que a negação de um conjunto de OUs é um conjunto de Es.

```
1: se IND ≠ 1 ∧ IND ≠ 4 ∧ IND ≠ 5 ∧ IND ≠ 6 ∧ IND ≠ 7 ∧ IND ≠ 9 então
2:    ... indicador CERTO ...
3: senão
4:    ... indicador ERRADO ...
5: fimse
```

Outra maneira de escrever o comando acima, seria:

```
1: se (IND = 1) ∨ (IND > 3 ∧ IND < 8) ∨ (IND = 9) então
2:    ... indicador ERRADO ...
3: senão
4:    ... indicador CERTO ...
5: fimse
```

Finalmente, se quiséssemos manter as saidas sem inversão:

```
1: se (IND ≠ 1) ∧ (IND ≤ 3 ou IND ≥ 8) ∧ (IND ≠ 9) então
2:    ... indicador CERTO ...
3: senão
4:    ... indicador ERRADO ...
5: fimse
```

Em resumo, a negação de

IGUAL	é DIFERENTE
DIFERENTE	IGUAL
MAIOR	MENOR OU IGUAL
MENOR	MAIOR OU IGUAL
MENOR OU IGUAL	MAIOR
MAIOR OU IGUAL	MENOR
E	OU
OU	E

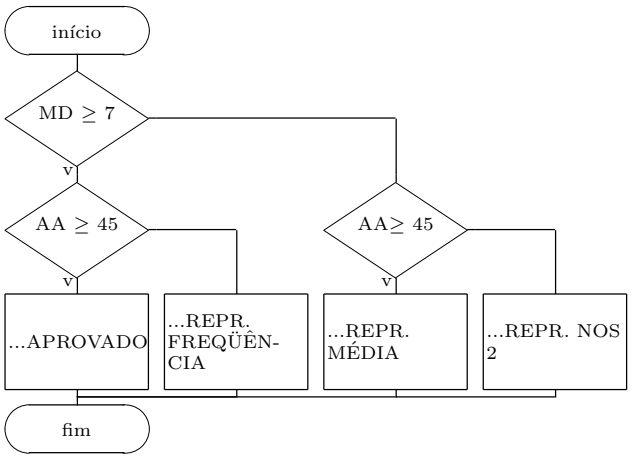
Outro Exemplo:
Suponhamos o seguinte trecho de lógica: Um aluno estará aprovado se:

- Tiver média maior ou igual a 7,00 E
- Tiver assistido a 45 ou mais aulas

Supondo que a média é a variável M, e que o número de aulas assistidas é a variável AA, poderia ficar

usando SEs aninhados Veja na figura 4.4.3

```
1: se MD ≥ 7 então
2:    se AA ≥ 45 então
3:       ...APROVADO
4:    senão
5:       ...REPR. FREQÜÊNCIA
6:    fimse
7: senão
8:    se AA≥ 45 então
9:       ...REPR. MÉDIA
10:   senão
11:      ...REPR. NOS 2
12:   fimse
13: fimse
```



usando SEs compostos

```
1: se MD ≥ 7 ∧ AA ≥ 45 então
2:    ...APROVADO
3: fimse
4: se MD ≥ 7 ∧ AA < 45 então
5:    ...REPROVADO POR FREQÜENCIA
6: fimse
7: se MD < 7 ∧ AA ≥ 45 então
8:    ...REPROVADO POR MÉDIA
9: fimse
10: se MD < 7 ∧ AA < 45 então
11:   ...REPROVADO NOS 2 CRITÉRIOS
12: fimse
```


misturarando tudo Funciona, embora não seja uma boa idéia.

```
1: se MD ≥ 7 ∧ AA ≥ 45 então
2:    ...APROVADO
3: senão
4:    se AA ≥ 45 então
5:       ...REPROVADO POR MÉDIA
6:    senão
7:       se MD ≥ 7 então
8:          ...REPROVADO POR FREQÜENCIA
9:       senão
10:         ...REPROVADO NOS 2 CRITÉRIOS
11:      fimse
12:   fimse
13: fimse
```

> Na programação, o demônio se esconde nos detalhes, Niklaus Wirth

EXERCÍCIO 22 Defina um algoritmo que receba séries de 3 valores reais indicativos de lados de um triângulo, medidos em centímetros. Para cada tripla, o programa deve responder as seguintes perguntas:

1. Tais lados podem formar um triângulo ?
2. Este triângulo é retângulo ?
3. Este triângulo é equilátero ?
4. Este triângulo é isósceles ?
5. Este triângulo é escaleno ?
6. Este triângulo é acutângulo ?
7. Este triângulo é obtusângulo ?

Condições do problema

é triangulo ?	maior $<$ A+B
retangulo	maior^2 $=$ A^2 $+$ B^2
equilatero	A $=$ B $=$ C
isósceles	(A $=$ B) ou (B $=$ C) ou (A $=$ C)
escaleno	(A $\neq$ B) e (A $\neq$ C)
acutângulo	maior^2 $<$ A^2 $+$ B^2
obtusângulo	maior^2 $>$ A^2 $+$ B^2

Por exemplo

se for lido	deverá ser impresso
3,4,5	sim, retângulo, escaleno
3,3,3	sim, equilátero, acutângulo
1,2,7	não
10,18,10	sim, isósceles, acutângulo

EXERCÍCIO 23 Um código está certo se for maior que 100, ou igual a 50. Escrever um algoritmo portugol que receba este valor e some 1 no código se ele estiver errado.

EXERCÍCIO 24 Uma data vem acompanhada de um campo chamado DIASEM, cujo valor é 1 para domingo, 2 para segunda, ... , 7 para sábado. Escrever um algoritmo portugol que receba DIASEM e verifique se o dia corresponde a um dia útil. (semana inglesa).

EXERCÍCIO 25 Escrever um algoritmo portugol que receba (não importa como) três valores numéricos (chamados A,B e C), e devolva a informação "OK"quando se satisfizerem as seguintes condições: (A deve ser maior que 10 e menor do que 100) OU (B deve ser diferente de C E C deve ser maior que 50). Se a condição não for satisfeita, o algoritmo deve devolver a mensagem "ERRO"

4.5 Estruturas de repetição

Para obedecer à terceira estrutura da programação estruturada e visando o reaproveitamento de código, ver-se-ão agora as possibilidades de repetir partes do algoritmo. Estas estruturas contém implícito um comando de desvio (**go to**) e nas linguagens mais modernas é o único desvio que é aceito.

Os trechos de programas que são repetidos ao se usar as estruturas de repetição são conhecidos genericamente com o nome de *loops* ou em português "laços". Assim, um erro de programação bastante comum é o chamado *loop infinito*, quando inadvertidamente a condição de saída é tal, que nunca é alcançada.

4.5.1 Repetição com condição no início: enquanto

Parece razoável que um algoritmo deve ser criado para a execução de um único conjunto de valores fornecidos como entrada. Por exemplo, ao escrever o algoritmo de aprovação de alunos na cadeira de algoritmos do Unicenp, o programador só precisa se preocupar com um único aluno, pois a regra de um vale para todos. Não teria sentido descrever os mesmos procedimentos para todos os alunos, isto seria interminável, além de deixar o algoritmo específico para um determinado número de alunos.

Falando em termos mais genéricos, ao escrever um programa de computador que calcule o salário de um empregado, deve-se imaginar apenas um empregado e não os milhares que o computador processará.

A chave para este problema está no reaproveitamento de instruções do algoritmo. Em outras palavras, uma vez escrito o caminho principal do algoritmo (o chamado caminho das pedras), nós vamos fazer todos os funcionários (ou alunos) passarem por este caminho. Uma das chaves para este procedimento é o comando chamado ENQUANTO. Seu formato:

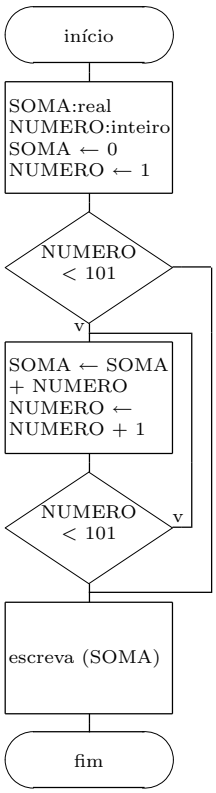
```
enquanto <condição> faça
   c1
   c2
   ...
fimenquanto
```

Este comando deve ser assim interpretado. A condição é avaliada. Se ela for falsa, o algoritmo deve saltar todos os comandos subordinados e continuar a execução após o comando **fimenquanto**. Entretanto, se a condição for verdadeira, os comandos subordinados são executados, até se encontrar o comando **fimenquanto**. Neste momento, há um desvio na seqüência de processamento e um retorno ao comando **enquanto**. A condição é novamente avaliada. Se falsa, pulam-se os comandos subordinados. Se verdadeira, os comandos são novamente executados, e assim por diante.

Se a condição for uma "verdade eterna", isto é, algo como $1 = 1$, tem-se um laço infinito, pois os comandos nunca deixarão de ser executados. Por outro lado se a condição for uma tautologia (sempre falsa), então os comandos subordinados nunca serão executados.

Exemplo

Calcular a soma dos números inteiros até 100.




```
1: SOMA:real
2: NUMERO:inteiro
3: SOMA ← 0
4: NUMERO ← 1
5: enquanto NUMERO < 101 faça
6:    SOMA ← SOMA + NUMERO
7:    NUMERO ← NUMERO + 1
8: fimenquanto
9: escreva (SOMA)
```

Comandos internos ao comando "enquanto" devem estar identados de 3 espaços, para clareza.

```
enquanto <condição>  faça
   <comando-1>
   <comando-2>
   ...
fimenquanto
```

Por exemplo:

```
1: leia A
2: enquanto A < 11 faça
3:    escreva A
4:    A ← A + 1
5: fimenquanto
```

4.5.2 Repetição com variável de controle: para

O segundo comando que se usa para controlar laços, atende pelo nome de **para**. Ele pressupõe a existência de uma variável de controle que irá (como o nome diz) controlar o início e o fim do laço.

O formato do comando **para** é

```
para variável DE valor-1 ATÉ valor-2 [PASSO valor-3] faça
   comando 1
   comando 2
```

```
   ...
   comando n
fimpara
```

A regra de funcionamento do para é:

- Antes de começar o trecho incluído no **para**, a variável mencionada no comando é inicializada com o valor-1.
- Se este valor for menor ou igual ao valor-2 o trecho subalterno é executado.
- Ao chegar ao final dos comandos, a variável é incrementada com o valor-3 (ou com 1 se nada for referenciado)
- Há um desvio incondicional, ao início do comando **para**, e o teste definido no passo <2> acima é refeito, com idênticas saídas.
- Dentro dos comandos subalternos ao **para**, a variável de controle não pode ser alterada pelos comandos escritos pelo usuário.
- Os valores 1, 2 e 3 podem ser valores auto-declarados (caso mais comum) ou podem ser quaisquer variáveis numéricas. Neste caso, elas também não podem ser alteradas dentro do **para**.
- Por convenção, quando o valor do **passo** for 1, toda a cláusula pode ser omitida. Exemplo: o comando `PARA K DE 1 ATE 10` equivale ao comando `PARA K DE 1 ATE 10 PASSO 1`.

Uma especial observação deve ser feita quando o valor-3 (o incremento) for negativo. Nestes casos há várias inversões no comando, a saber:

- O valor 1 deve ser maior do que o valor 2, já que a variável de controle vai diminuir ao invés de aumentar.
- O teste de saida é para maior ou igual, por idêntica razão.

Veja-se dois exemplos para ajudar a entender e a guardar estas questões:

```
1: para J de 1 até 9 passo 2 faça
2:    escreva J
3: fimpara
```

Aqui serão impressos os valores 1, 3, 5, 7 e 9. Já em

```
1: para J de 9 até 1 passo -2 faça
2:    escreva J
3: fimpara
```

Serão impressos os valores 9, 7, 5, 3 e 1.

Devemos lembrar que todo comando **para** pode ser convertido em seu equivalente **enquanto**. Dá mais trabalho (são 3 comandos) mas sempre é possível. Já a recíproca nem sempre é verdadeira. Só é possível transformar um **enquanto** em um **para** equivalente, quando se souber o número exato de iterações que o **enquanto** faria. Quando isto for desconhecido a conversão não é possível.

Exemplos:

usando para	equivalente usando enquanto

`para I de 1 até 10 passo 2`
`  escreva I`
`fimpara`
	`I ← 1`
	`enquanto (I ≤ 10)`
	`  escreva I`
	`I ← I + 2`
	`fimenquanto`

4.5.3 Repetição com condição no final: repita

Além da instrução **enquanto**, a maioria das linguagens de alto nível oferece outras formas de repetir uma parte do programa. A rigor, com uma única estrutura poder-se-ia resolver todos os problemas (como, demonstraram aqueles dois professores italianos). Entretanto, criar um programa não é um exercício acadêmico, ou não é só isso. Na universidade, não há pressa, e o rigor científico é o que importa. Na vida prática há pressa, e o rigor é importante na medida em que garante qualidade, ele não é um fim em sí mesmo.

Portanto, visando facilitar, acelerar e simplificar, tem-se aqui um outro comandos de repetição:

```
repita
   comando 1
   comando 2
   ...
   comando n
até <condição>
```

Os comandos internos ao **repita** são executados enquanto a condição contida na cláusula **até** não for satisfeita. Isto é, quando o fluxo chega ao **até**, a condição é avaliada. Se for verdadeira, o fluxo segue após o **até**. Se for falsa, o fluxo retorna ao comando imediatamente seguinte ao **repita**.

Embora parecido com o **enquanto**, o **repita** tem algumas diferenças, a saber

- Os comandos internos ao **repita** são executados sempre ao menos uma vez, independente da condição. Já no **enquanto**, tais comandos podem não ser executados nem uma vez.
- O laço do enquanto será realizado se a condição for VERDADEIRA. Já o laço do **repita** será realizado se a condição for FALSA.
- A estrutura **enquanto** testa e faz. A estrutura **repita** faz e testa.

Em geral tudo o que se faz com um deles se pode fazer com o outro. A vantagem é que para determinadas situações (que são específicas de cada algoritmo) um ou outro pode ser mais indicado, isto é pode resolver o problema com menos lógica adicional (o chamado "overhead") tais como contadores, chaves, testes etc.

Uma observação meio extemporânea, mas ainda assim importante aos java-parlantes ou aos C-parlantes: Nestas duas linguagens o comando de repetição com condição no início e no final é o mesmo. Trata-se do comando **while**. Muda apenas o local da condição. Por esta razão, não existe a inversão lógica de saída explicitada acima. Em Java e em C, a saída quando a condição aparece depois do bloco é quando a condição é FALSA. É o contrário do comando **repita** acima descrito.

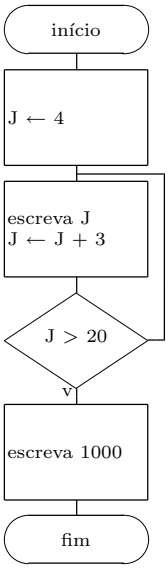
Veja-se em exemplos:

No portugol acima descrito tem-se

Neste trecho serão impressos os valores 4, 7, 10, 13, 16, 19 e 1000. Já em Java o trecho ficaria

```
1: J = 4
2: while {
3: system.out.println (J) // é assim mesmo ?
4: J = J + 3
5: J > 20
6: system.out.println (1000) }
```

Será impresso pelo Java: 4, 1000.




```
1: J ← 4
2: repita
3:     escreva J
4:     J ← J + 3
5: até J > 20
6: escreva 1000
```

tipos de algoritmos (algo jocoso)

Uma categorização com algo de chacota, mas que ajuda a entender os algoritmos:

Le e imprime o algoritmo recebe um dado e a partir dele gera resultados (90% dos algoritmos pertencem a esta classe)

Le e esquece o algoritmo recebe um dado, mas não gera resultado nenhum externo ao computador. (Por exemplo, um programa que atualize o relógio da máquina)

Inventa e imprima o algoritmo não recebe nada, mas ao ser acionado gera um resultado (Por exemplo, o algoritmo que estabelece "letra pequena" na impressora matricial.

Inventa e esquece ???

EXERCÍCIO 26 Definir algoritmo que escreva a soma dos primeiros 100 números inteiros (1, 2, 3, ... , 100)..

EXERCÍCIO 27 Definir algoritmo que escreva a soma dos primeiros 100 números pares inteiros (2, 4, 6, ... , 200).

EXERCÍCIO 28 Definir algoritmo que some todos os múltiplos de 7 compreendidos entre 100 e 10000, e informe ao final, quantos foram os múltiplos e quanto resultou sua soma.

EXERCÍCIO 29 Escreva um algoritmo que leia dois números inteiros n e m e:

- Teste se $m > n$, se não for dar uma mensagem de erro e terminar o processamento
- Somar todos os múltiplos de 13 que estiverem compreendidos entre n e m, inclusive
- Imprimir a soma ao final

EXERCÍCIO 30 Definir algoritmo que leia 17.000 números reais e informe, quantos:

1. foram maiores do que 100;

2. foram pares;

3. foram inteiros;

4. foram negativos;

EXERCÍCIO RESOLVIDO 1 Escreva um algoritmo que leia uma série de notas (que se encerram quando for lido um número negativo) e ao final escreva as duas maiores notas que apareceram.

Por exemplo, se as notas lidas forem 8, 3, 2, 1, 10, 7, 8, 9, 5, 4 e 3, o algoritmo deve imprimir 10, 9

Como o exercício é um pouco mais complexo, vamos ver diversas estratégias para a sua solução:

1. primeira estratégia

```
1: enquanto ... faça
2:    leia(N)
3:    se N > P então
4:       P ← N
5:    fimse
6:    se N > S então
7:       S ← N
8:    fimse
9: fimenquanto
```

   DEFEITO: ao final, P e S terão o mesmo valor (o maior)

2. segunda abordagem

```
1: enquanto ... faça
2:    leia(N)
3:    se N > P então
4:       P ← N
5:    fimse
6:    se N > S então
7:       se N ≠ P então
8:          S ← N
9:       fimse
10:   fimse
11: fimenquanto
```

   DEFEITO: Com a série 1, 10, 2, 8, 5 → funciona

   Com a série 1, 8, 7, 2, 9 → não funciona

3. terceira abordagem

```
1: inteiro NUM, NOT, PNU, PNO, SNU, SNO, X
2: X ← 1
3: PNO ← 0
4: SNO ← 0
5: enquanto X < 100 faça
6:    leia (NUM, NOT)
7:    se NOT > PNO então
8:       se PNO > SNO então
9:          SNO ← PNO
10:         SNU ← PNU
```

```
11:       fimse
12:       PNO ← NOT
13:       SNO ← NUM
14:    fimse
15:    se NOT > SNO então
16:       se NOT ≠ PNO então
17:          SNO ← NOT
18:          SNU ← NUM
19:       fimse
20:    fimse
21: fimenquanto
22: escreva (PNO,PNU)
23: escreva (SNO,SNU)
```

Eis a lista de variáveis usada acima
NUM = número do aluno atual
NOT = nota do aluno atual
PNU = número do aluno com a melhor nota até agora
PNO = melhor nota até agora
SNU = número do aluno com a segunda melhor nota até agora
SNO = segunda melhor nota até agora
X = contador até 100

EXERCÍCIO 31 Definir algoritmo capaz de jogar com o operador o "JOGO DO PALITO", e de vencer, sempre que possível. Este jogo tem a seguinte regra.

- São dois jogadores, a quem chamaremos: máquina (programa) e humano.
- O humano escolhe um número de palitos qualquer entre 20 e 30.
- A máquina retira 1, 2 ou 3 palitos.
- O humano também retira 1, 2 ou 3 palitos, e a seqüência prossegue, até que reste apenas um palito.
- Quem tirar o último palito perde.
- O programa deve atentar para impedir retiradas diferentes de 1 2 ou 3.

Dica: Estratégia vencedora: deixar o adversário sempre com 1, 5, 9, 13, 17, 21, 25 ou 29

EXERCÍCIO 32 Defina algoritmo que calcule e escreva o somatório expresso pela seguinte série:

$$S = \frac{500}{2} + \frac{480}{3} + \frac{460}{4} + ... + \frac{20}{26}$$

EXERCÍCIO 33 Escreva um algoritmo que leia uma seqüência de números positivos (a condição de fim é a leitura do número -1) e escreva ao final, qual o número mais próximo de 100 que foi lido.
DESAFIO: escreva o número par mais próximo a 100.

EXERCÍCIO 34 Escreva um algoritmo que leia uma seqüência de números positivos (a condição de fim é a leitura do número -1) e escreva ao final, qual o último número que foi lido.
DESAFIO: escreva o ante-penúltimo, ou -1 se não houver ante-penúltimo

EXERCÍCIO 35 Dados o seguinte trecho de lógica escritos usando o **enquanto**, escrever trecho equivalente usando o **repita**

```
1: Z ← 10
2: enquanto Z > 0 faça
3:     Z ← Z - 3
4:     escreva (Z)
5: fimenquanto
```

EXERCÍCIO 36 Dado o seguinte algoritmo que utiliza o comando **repita** escrever comandos equivalentes usando o comando **enquanto**.

```
1: GH ← 5
2: repita
3:     escreva (ABC)
4: até GH ≠ 5
```

EXERCÍCIO 37 Escreva o trecho a seguir, usando

1. o comando Enquanto e
2. o comando Repita

```
1: para J de 2 até -10 passo -3 faça
2:     K ← sqr(J)
3:     escreva K
4: fimpara
```

EXERCÍCIO 38 Escrever um algoritmo que leia um único número (que por definição é maior que 3), e escreva "PRIMO"se ele for primo, ou "NÃO PRIMO"se ele for divisível. Por exemplo, se for lido 10, o programa dever dizer "NÃO PRIMO", e se for lido 11, dever dizer "PRIMO".

EXERCÍCIO 39 Definir algoritmo que leia uma seqüência de valores numéricos inteiros e determine, ao final se eles estavam em ordem ascendente ou não. Se estiverem, deve imprimir "EM ORDEM", e se não estiverem, deve imprimir "FORA DE ORDEM".

EXERCÍCIO 40 Defina algoritmo que calcule e escreva o somatório expresso pela seguinte série. O número de termos deve ser lido a priori.

$$S = 1\frac{1}{2} + 2\frac{1}{3} + 3\frac{1}{4} + ... + n\frac{1}{n+1}$$

EXERCÍCIO 41 Escreva o algoritmo que leia N e escreva S, onde

$$S = \frac{1}{2} + \frac{2}{3} + \frac{3}{4} + ... + \frac{N}{N+1}$$

EXERCÍCIO 42 Dados o seguinte trecho de lógica escritos usando o **enquanto**, escrever trecho equivalente usando o **repita**.

```
1: A ← 10
2: B ← 20
3: enquanto (A + B) < 50 faça
4:     A ← A + 5
5:     B ← B + 10
6:     escreva (A+B)
```

```
7: fimenquanto
```

EXERCÍCIO 43 Dados o seguinte trecho de lógica escritos usando o **enquanto**, escrever trecho equivalente usando o **repita**.

```
1: K ← 5
2: enquanto K < 10 faça
3:    K ← K + 1.5
4:    escreva (K)
5: fimenquanto
```

EXERCÍCIO 44 Dado o seguinte algoritmo que utiliza o comando **repita** escrever comandos equivalentes usando o comando **enquanto**.

```
1: T ← 0
2: repita
3:    T ← T + 0.5
4:    escreva (T × 2)
5: até T > 10
```

EXERCÍCIO 45 Escreva o trecho a seguir, usando

1. o comando Enquanto e
2. o comando Repita

```
1: para T de 1 até 2 passo 2 faça
2:    escreva T
3: fimpara
```

EXERCÍCIO 46 Dado o seguinte algoritmo que utiliza o comando **repita** escrever comandos equivalentes usando o comando **enquanto**. a)

```
1: A ← 10
2: repita
3:    A ← A + 1
4: até (A > 10)
```

b)

```
1: GH ← 5
2: repita
3:    escreva (ABC)
4: até (GH ≠ 5)
```

c)

```
1: T ← 0
2: repita
3:    T ← T + 0.5
4:    escreva (T * 2)
5: até (T > 10)
```

EXERCÍCIO 47 Um comercial de rádio tem seu custo em função de 2 variáveis, a saber: duração e hora de irradiação, segundo a tabela:

Duração		Horário	
	de 0 a 5h59min	de 6 a 20h59min	de 21 a 24h
Até 10 segundos	5 US$	12 US$	8 US$
De 11 a 30 segs	12 US$	21 US$	17 US$
Acima de 30 segs	16 US$	26 US$	22 US$

Obs: o preço acima é para o comercial inteiro, ou seja, NÃO é por segundo de comercial.
Definir algoritmo que:

1. leia o valor da cotação do dólar naquele dia
2. leia série de duplas (duração,hora) escreva a dupla e o valor a ser cobrado em reais. A duração está dada em segundos.

Os dados terminam quando for lida uma dupla com duração igual a zero.
Valores válidos: $0 \leq$ Duração ≤ 60, $0 \leq Hora \leq 23$
Por exemplo, se o dólar estiver cotado a R$ 2,20 e

forem lidos	dever ser impresso
10,4	26,40 R$
20,23	37,40 R$
10,23	17,60 R$
0,0	fim...

EXERCÍCIO 48 Escreva um algoritmo capaz de determinar o valor da seguinte série:

$$S = \frac{2}{1} - \frac{3}{4} + \frac{4}{9} - \frac{5}{16} + \frac{6}{25} + ...$$

O número de termos da série deve ser lido, e nunca será maior que 25. Ao final do processamento, o algoritmo deve imprimir o valor da série.

EXERCÍCIO 49 Definir um algoritmo que leia uma série indeterminada de valores positivos, sendo que a marca de fim é a leitura do número -1. O algoritmo deve imprimir qual o último número lido, imediatamente antes do número -1. Por exemplo, se for lida a série 3, 7, 21, 4, -1, o algoritmo deve imprimir 4.

EXERCÍCIO 50 Defina e escreva um algoritmo para imprimir todos os números perfeitos entre 1 e 500. Um número perfeito é aquele que é igual a soma de todos os seus fatores. O primeiro número perfeito é o 6 (6=1+2+3), logo por definição o número 1 não é perfeito. Usar uma função para determinar se um número é ou não é perfeito. Outros números são: 28, 496. Depois disso, descobrir qual o próximo número perfeito, maior que 500.

EXERCÍCIO 51 Defina um algoritmo que calcule e escreva a soma da série a seguir, considerando 50 termos:

$$S = \frac{2}{3} + \frac{4}{5} + \frac{6}{7} + ... + \frac{100}{101}$$

EXERCÍCIO 52 Definir algoritmo que receba uma série de triplas de números inteiros. Para cada tripla deve ser gerada uma progressão aritmética supondo que:

- O primeiro elemento da tripla é o termo inicial
- O segundo elemento da tripla é a razão
- O terceiro elemento da tripla é o número de termos

Os elementos da PA deverão ser impressos. O processo termina quando for lido um número de termos igual a zero.

EXERCÍCIO 53 Escreva um algoritmo que leia uma série de números inteiros e positivos e ao final do processamento escreva qual o maior deles. O processamento se encerra quando for lido qualquer número negativo que não deve ser considerado para efeitos de comparação.

EXERCÍCIO 54 Escrever um algoritmo, que receba conjuntos de 3 notas de um aluno. O primeiro valor corresponde a média do primeiro semestre. O segundo valor é média do segundo semestre, e o terceiro valor correspondendo a nota final. Existe uma variável no algoritmo (pré-definida) chamada QTD-APROV. Para cada aluno aprovado, o algoritmo deve somar 1 em QTD-APROV. Regra de aprovação:

$$(N_1 \times 3 + N_2 \times 3 + N_3 \times 4) \div 10 \geq 7$$

Os dados terminam quando o primeiro valor da série for negativo. Por exemplo, se QTD-APROV tiver o valor 0, e forem lidas as triplas (5,7,2), (8,8,6), (1,10,5) e (-1,0,0) o resultado final de QTD-APROV será 1.

EXERCÍCIO 55 Definir um algoritmo que leia uma série de pares de valores inteiros que representam COMPRIMENTO e LARGURA de um retângulo. Calcular e imprimir a área do retângulo se o perímetro do mesmo for superior a 25. Os dados se encerram quando for lida a dupla zero,zero. Por exemplo, se forem lidas as duplas (2,2), (10,8), (10,1) e (0,0) só será impressa a área 80, equivalente à dupla (10,8).

EXERCÍCIO RESOLVIDO 2 Imagine um relógio analógico de ponteiros. Não existe o ponteiro de segundos, e os ponteiros realizam movimentos discretos, isto é, eles só se movem a cada 60 segundos. Escreva um algoritmo que leia diversos conjuntos de hora e minuto, e para cada conjunto lido, informe qual o menor ângulo que os dois ponteiros fazem entre si ao representar a hora informada. O algoritmo termina quando for lida a dupla 0,0. Os valores permitidos para hora estão entre 0 e 11 e para minuto os valores válidos estão entre 0 e 59. A saída do resultado pode ser em graus e décimos de grau, ou se o aluno preferir em graus e minutos.

```
 1: algoritmo ponteiro
 2: inteiro h m
 3: real ah am qtm angulo
 4: leia (h,m)
 5: enquanto h ≠ 99 faça
 6:    qtm ← (h * 60) + m
 7:    ah ← qtm * 0,5
 8:    am ← qtm * 6
 9:    enquanto ah ≥ 360 faça
10:       ah ← ah - 360
11:    fimenquanto
12:    enquanto am ≥ 360 faça
13:       am ← am - 360
14:    fimenquanto
15:    se se ah > am então
16:       angulo ← ah - am
17:    senão
18:       angulo ← am - ah
19:    fimse
20:    se se angulo > 180 então
21:       angulo ← 360 - angulo
22:    fimse
23:    escreva (angulo)
24:    leia(h,m)
25: fimenquanto
26: fim{algoritmo}
```


EXERCÍCIO 56 Interprete o seguinte trecho de algoritmo, informando (em português) o que faz ou para que serve o algoritmo analisado. Estude e informe a condição de fim. Se necessário, realize um "chinês"sobre os dados.

```
1: inteiro CA,CE
2: leia(CA,CE)
3: enquanto CA ≠ CE faça
4:    se CA < CE então
5:       escreva(CA)
6:    fimse
7:    escreva(CE)
8:    leia(CA,CE)
9: fimenquanto
```

EXERCÍCIO 57 Interprete o seguinte trecho de algoritmo, informando o que faz ou para que serve o algoritmo analisado. Estude e informe a condição de fim. Se necessário, realize um "chinês"sobre os dados.

```
1: inteiro A,B,C
2: real X
3: {A é _____}
4: {B é _____}
5: {C é _____}
6: {X é _____}
7: leia(A)
8: B ← 0
9: enquanto A ≠ 0 faça
10:    se A < B então
11:       B ← A
12:    fimse
13:    leia(A)
14: fimenquanto
15: escreva (B/2)
```

EXERCÍCIO 58 Interprete o seguinte trecho de algoritmo, informando o que faz ou para que serve o algoritmo analisado. Estude e informe a condição de fim. Se necessário, realize um "chinês"sobre os dados.

```
1: real PI,R,C
2: {R é _____}
3: {C é _____}
4: PI ← 3.141592
5: leia(R,C)
6: enquanto (R ≠ 0) ∨ (C ≠ 0) faça
7:    se R = 0 então
8:       C ← 2 × PI × R
9:    senão
10:      R ← C ÷ (2 × PI)
11:   fimse
12:   escreva(R,C)
13:   leia(R,C)
14: fimenquanto
```

EXERCÍCIO 59 Siga o seguinte algoritmo, e informe quais os valores de A, B, C e D que são impressos ao final.

```
 1: inteiro A, B, C, D
 2: A ← 0
 3: B ← 10
 4: C ← B × 3
 5: D ← B + C + A
 6: enquanto D < 0 faça
 7:    A ← A + 1
 8:    D ← D + 1
 9: fimenquanto
10: se A > 0 então
11:    D ← D × 2
12: senão
13:    B ← B × 2
14: fimse
15: escreva (A,B,C,D)
```

```
A:__________ B:__________ C:__________ D:__________
```

EXERCÍCIO 60 Brinquedos "PIRRALHOS ENDIABRADOS" é um grande distribuidor de presentes em todo o país. Recentemente, a empresa teve a oportunidade de comprar 15.000 pequenos brinquedos, todos embalados em caixas retangulares. O objetivo da compra, foi colocar cada brinquedo em uma esfera colorida, para revendê-los como surpresa, mais ou menos como o Kinder ovo. Existem esferas de raios 10, 20 e 30 cm. Cada brinquedo, tem um número de ordem, e as suas 3 dimensões A, B e C, medidas em centímetros. Definir um algoritmo que leia 15.000 quádruplas (ordem,A,B,C) e para cada uma delas escreva a ordem do brinquedo e o raio da esfera necessário. Todos os brinquedos caberão em uma das esferas.

EXERCÍCIO 61 Supondo um trecho de código escrito em pseudo-código:

```
1: para I de 1 até 100 faça
2:    para J de 3 até 11 passo 2 faça
3:       para K de 5 até 25 passo 5 faça
4:          escreva (I,J,K)
5:       fimpara
6:    fimpara
7: fimpara
```

Imagine que você precisa re-escrever este código e a sua nova linguagem não é estruturada, o que significa que não existem as estruturas para ... fimpara, enquanto ... fimenquanto e nem repita ... até. Você só conta com labels, testes e desvios.

Eis como ficaria:

```
 1: algoritmo qualquer
 2: I ← 1
 3: J ← 3
 4: K ← (*1)
 5: L1: se (I > 100)
 6: vápara FIM
 7: senão
 8: se (J > 11)
 9: vápara L2
10: senão
11: se (K > (*2))
12: vapara L3
```


13: senão
14: escreva (I,J,K)
15: K ← K + (*3)
16: vápara L1
17: fimse
18: fimse
19: fimse
20: L3: J ← J + (*4)
21: K ← 5
22: vápara L1
23: L2: I++
24: J ← (*5)
25: K ← 5
26: vápara L1
27: fimalgoritmo

Os valores propostos para (*1) (*2) (*3) (*4) e (*5) para que os dois trechos sejam equivalentes são

```
a)  5, 25,  5,  2,  2
b)  5,  5,  5,  2, 11
c)  5, 25,  5,  2, 11
d)  1, 25,  1,  1,  2
e)  1, 25,  1,  2,  1
```

Resposta certa: letra (a)

4.5.4 Comando de múltipla escolha: Escolha

Nos casos em que há múltiplas saídas para um determinado teste, nós podemos usar a estrutura de alternativas simples e compostas, devidamente encadeadas. Tal uso (de acordo com os dois mestres italianos) sempre é suficiente para a solução de qualquer problema.

Entretanto, para simplificar a possível solução, inúmeras linguagens tem o comando de teste múltiplo (COBOL tem o GOTO depending ON, PASCAL tem o comando CASE, dbase tem o CASE, APL tem o desvio para um vetor etc). Aqui, usar-se-á o seguinte formato:

```
escolha <expressão>
   caso valor-1 [: valor-2  ...] : comando-1
   caso valor-a [: valor-b  ...] : comando-a
   ...
   senão : comando-z
fimescolha
```

O funcionamento deste comando é simples: A expressão que aparece ao lado do comando **Escolha** é avaliada, e deve gerar um resultado determinado. Este resultado será comparado com os valores colocados ao lado direito das palavras **caso**. Quando um valor igual for encontrado, o comando colocado o seu lado será executado. Finalmente, se nenhum valor for igual, e existir a cláusula senão, esta será executada. Exemplo

1: ESCOLHA (A + 1)
2: CASO 1 : comando-1;
3: CASO 2 : 3 : 4 : comando-2;
4: SENÃO: omando-3;
5: FIMESCOLHA

Aqui, se A for zero, será executado o comando-1.
Se A for 1, 2 ou 3, será executado o comando-2.
Se A tiver qualquer valor diferente de 0, 1, 2 ou 3, será executado o comando-3.

4.5.5 Abandono de iteração: abandone

Este comando exige uma saída incondicional de cada uma das estruturas de repetição (enquanto, repita ou para), independente de condições. Geralmente, este comando é colocado dentro de um "se". O desvio sempre se dá para o comando seguinte ao fim da estrutura que está sendo usada. Eis o formato do comando

```
abandone
```

Este comando só tem sentido quando emitido dentro de um laço de processamento (dentro de um **enquanto**, **repita** ou **para**). Ele é utilizado quando se deseja - a partir de uma determinada condição - abandonar o processo iterativo em curso. Seu funcionamento é simples: ao ser executado, este comando provoca um desvio incondicional para o comando subseqüente ao fim do laço.

Exemplo

```
1: enquanto A > 10 faça
2:    comando 1...
3:    comando 2...
4:    se B ≠ 5 então
5:       abandone
6:    fimse
7:    comando 10...
8: fimenquanto
9: comando 20...
10: ...
```

Neste caso, quando abandone for executado, independente do estado da variável A (e do teste (A > 10), será executado o comando 20. Assim, quando abandone é executado dentro de um **para**, desvia-se para o comando seguinte ao **fimpara**. Da mesma forma, quando dentro de um **repita** executa-se o seguinte a **até** ... .

Se existirem dois ou mais laços (um dentro do outro), e for emitido um abandone, ele vale para o laço mais interno. Exemplo

```
1: enquanto (X = 10) faça
2:    comando 1 ...
3:    enquanto (Y > 20) faça
4:       comando 10 ...
5:       se (Z = 8) então
6:          abandone
7:       fimse
8:       comando 30 ...
9:    fimenquanto
10:   comando 40...
11: fimenquanto
12: comando 50...
```

Neste caso, ao ser executado o abandone, será executado o comando 40 e não o comando 50 como poderia parecer a primeira vista.

EXERCÍCIO 62 Dados o seguinte trecho de lógica escritos usando o **enquanto**, escrever trecho equivalente usando o **repita**.


```
1: A ← 0
2: enquanto A ≠ 3 faça
3:    A ← A + 1
4:    escreva (A)
5: fimenquanto
```

EXERCÍCIO 63 Dado o seguinte algoritmo que utiliza o comando **repita** escrever comandos equivalentes usando o comando **enquanto**.

```
1: A ← 10
2: repita
3:    A ← A + 1
4: até A > 10
```

EXERCÍCIO 64 Dado o trecho a seguir, escrito usando o comando **para**, reescreve-lo usando o comando **enquanto**

```
1: para J de 1 até 20 faça
2:    X ← J ÷ 3
3:    escreva (X)
4: fimpara
```

EXERCÍCIO 65 Dado o trecho a seguir, escrito usando o comando **para**, reescreve-lo usando o comando **enquanto**

```
1: para SEMENTE de 0 até 100 passo 2 faça
2:    SEM1 ← SEMENTE × 2
3:    SEM2 ← SEMENTE + 1.5 × ABC
4:    MEDIA ← (SEM1 + SEM2) ÷ 2
5:    escreva MEDIA
6: fimpara
```

EXERCÍCIO 66 Escreva o trecho a seguir, usando

1. o comando Enquanto e
2. o comando Repita

```
1: para K de 5 até 25 passo 3 faça
2:    escreva K+1
3: fimpara
```

EXERCÍCIO RESOLVIDO 3 Dois números são considerados "amigos"quando um número é igual a soma dos fatores do outro número. Por exemplo, 220 e 284 são "amigos", pois 220 é igual a soma dos fatores de 284 (142,71,4,2,1) e 284 é a soma dos fatores de 220 (110 55 44 22 20 11 10 5 4 2 1). Definir algoritmo que escreva os pares de números amigos existentes entre 1 e 500.

```
1: inteiro i,j,k,l,m,n
2: inteiro função somf (inteiro a)
3: inteiro var ind,som
4: som ← 0
5: para ind ← 1 to a-1 faça
6:    se a mod ind = 0 então
7:       som ← som + i
8:    fimse
9: fimpara
```

```
10: retorne som
11: fim{função}
12: para i ← 1 to 500 faça
13:    escreva i
14:    para j← i to 500 faça
15:       m ← somf(i)
16:       n ← somf(j)
17:       se (m = j) ∧ (n = i) então
18:          escreva (i,j)
19:       fimse
20:    fimpara
21: fimpara
```

EXERCÍCIO RESOLVIDO 4 Escreva uma função que receba um número inteiro positivo k, calcule e devolva o próximo número primo x, onde $k \leq x$.

Solução Para resolver este exercício, consideraremos definida e operante a função EPRIMO, com o seguinte cabeçalho `lógico função EPRIMO (inteiro x)`, que devolve VERDADEIRO quando x é primo e FALSO senão.

Eis a resposta

```
1: inteiro função PROXPRIM (inteiro K)
2: enquanto 1=1 faça
3:    se EPRIMO(K) então
4:       retorne K
5:       abandone
6:    fimse
7:    K++
8: fimenquanto
9: fimfunção
```

Atente para a condição terminadora do **enquanto**. Já que a saída se dará através do abandone, qualquer condição sempre verdadeira pode ser colocada aí.

EXERCÍCIO RESOLVIDO 5 Você tem uma coleção seqüencial de 10.000.000 de apostas na sena. Para cada apostador, existem os 6 números em que ele jogou e mais o número da aposta, que posteriormente identificará o apostador. Suponha uma função `lógico ganhou (inteiro a1, a2, a3, a4, a5, a6)` que devolve VERDADEIRO se esta aposta é vencedora e FALSO senão. Suponha também que a condição de fim de dados é a leitura de $a1$ negativo.

Solução Para resolver este exercício, duas hipóteses podem ser consideradas: a primeira faz uma dupla leitura (uma fora e outra dentro do loop).

```
1: algoritmo SENA
2: inteiro a1, a2, a3, a4, a5, a6, apostador
3: leia (a1, a2, a3, a4, a5, a6, apostador)
4: enquanto a1 > 0 faça
5:    se ganhou (a1, a2, a3, a4, a5, a6) então
6:       escreva apostador
7:    fimse
8:    leia (a1, a2, a3, a4, a5, a6, apostador)
9: fimenquanto
10: fim{algoritmo}
```


A segunda abordagem, faz apenas uma leitura, mas a função é nitidamente menos limpa e clara:

```
 1: algoritmo SENA2
 2: inteiro a1, a2, a3, a4, a5, a6, apostador
 3: a1 ← 1 {poderia ser qualquer valor maior que zero}
 4: enquanto a1 > 0 faça
 5:    leia (a1, a2, a3, a4, a5, a6, apostador)
 6:    se a1 > 0 então {pode ter chegado o negativo}
 7:       se ganhou (a1, a2, a3, a4, a5, a6) então
 8:          escreva apostador
 9:       fimse
10:    fimse
11: fimenquanto
12: fim{algoritmo}
```

EXERCÍCIO RESOLVIDO 6 Seja agora a tarefa, muito comum em programação, de obter números construídos segundo uma certa lei de formação, a princípio desconhecida. Só se conhecem os primeiros números de uma seqüência deles, e o que é pedido, é o resto dos números.

Por exemplo, se a seqüência for 1,2,3,4,5,6 e for pedido o próximo número, a resposta é 7, por que a lei de formação é a dos números naturais.

Entretanto, outras seqüências podem ser menos evidentes e conseqüentemente mais trabalhosas. Vejamos alguns exemplos:

```
  5    7    9   11   13   15   17  ...   19
  1    4    9   16   25   36   49  ...   64
  4    7   12   19   28   39   52  ...   67
  8   14   24   38   56   78  104  ...  134
  5    8   13   20   29   40   53  ...   68
  4   11   30   67  128  219  346  ...  515
 16   25   36   49   64   81  100  ...  121
512  729 1000 1331 1728 2197 2744  ... 3375
 36  144  324  576  900 1296 1764  ... 2304
  8   64  216  512 1000 1728 2744  ... 4096
```

Uma estratégia é pesquisar os possíveis valores de x, y e z, usando intuição, pistas, experiências passadas, ao estilo de Sherlock Holmes. Uma segunda estratégia é usar da informação de que $2 \geq x, y, z, \leq 7$ e adotar a força bruta, fazendo

```
 1: para X de 2 até 7 faça
 2:    para Y de 2 até 7 faça
 3:       para Z de 2 até 7 faça
 4:          para I de 1 até 10 faça
 5:             escreva ...
 6:          fimpara
 7:          escreva ...
 8:       fimpara
 9:    fimpara
10: fimpara
```

EXERCÍCIO 67 Escrever um algoritmo que leia uma série indeterminada de números positivos. A leitura deve prosseguir até ser lido o número 1203, quando o programa dar a mensagem "1203 ENCONTRADO" e terminar o processamento. Se este número não for encontrado, ao final dos dados (quando for lido um número negativo) o programa deve

imprimir o número lido mais próximo de 1203, com a mensagem "xxx MAIS PRÓXIMO", onde xxx é o número em questão.

EXERCÍCIO 68 Defina um algoritmo capaz de somar os primeiros 50 números múltiplos de 4, começando no 4 (isto é: 4, 8, 12, ...400). O algoritmo deve imprimir o resultado ao final.

EXERCÍCIO 69 Escreva um algoritmo capaz de calcular e imprimir o valor da somatória dada pela série a seguir. O número de termos é lido ao início do algoritmo.

$$S = \frac{12}{1} + \frac{22}{2} + \frac{32}{4} + ...$$

Para resolver este tipo de exercício, deve-se buscar a fórmula do termo genérico. Sem esta fórmula é difícil programar a solução. Fica:

$$T_n = \frac{12 + (n-1) \times 10}{2 \times (n-1)}$$

e sobretudo

$$S = \sum_{i=1}^{n} T_n$$

ou

$$S = \sum_{i=1}^{n} \frac{12 + (n-1) \times 10}{2 \times (n-1)}$$

EXERCÍCIO 70 Suponha

1: X $\leftarrow$ 2
2: Y $\leftarrow$ 5
3: **enquanto** $(X < 5) \wedge (Y < 3)$ **faça**
4: X++
5: Y- -
6: **fimenquanto**

- Ao se encerrar este algoritmo quais os valores de X e Y ?
- O que acontece se se tirar o "E"colocando em seu lugar "OU"

EXERCÍCIO 71 Seja

1: A $\leftarrow$ 56
2: B $\leftarrow$ 10
3: **enquanto** (A+B) mod 14 $\neq$ 0 **faça**
4: A $\leftarrow$ A + 10
5: B $\leftarrow$ A + 2
6: **fimenquanto**

- quais os valores de A e B ao final ?
- Se A começar com 52, o que acontece com o algoritmo ?

EXERCÍCIO 72 Qual é o 30º número primo ?

EXERCÍCIO 73 Escreva o algoritmo de uma função que receba um número inteiro positivo X e devolva o próximo primo maior ou igual a X.

EXERCÍCIO 74 Dois inteiros positivos a e b são relativamente primos se não tiverem fatores comuns exceto 1. Por exemplo, 4 e 25 são relativamente primos, apesar de nenhum deles o ser. Escreva uma função que receba X e Y e devolva .V. se eles forem primos relativos e .F. senão.

A propósito, os números 17.773 e 14.973 são relativamente primos ?

EXERCÍCIO 75 Diamantes são classificados através de 3 testes distintos, como segue:

- Dureza maior do que 12
- Percentagem de impurezas menor que 45
- Resistência à tração maior que 600 Kgf

Se o diamante passa nos três testes, ele ganha o grau 50. Se passa em dois, ganha o grau 40, se passa apenas em um teste, ganha 30, e se não passa em nenhum ganha o grau 15.

Escrever um algoritmo que leia um conjunto de variáveis formado por Número da partida (inteiro), dureza (real), percentagem de impurezas (real), resistência (real); e escreva o número da partida e a nota do diamante. Os dados terminam quando for lida a partida número zero. A nota do diamante deve ser calculada por uma função.

EXERCÍCIO 76 Escreva algoritmo capaz de criar e imprimir uma tabela de ângulos, senos e cossenos, para todos os ângulos de 0 a 90 graus.

Exemplo

ÂNGULO	SENO	COSSENO
0^o	0	1
1^o	0.01745240644	0.9998476952
2^o	0.0348994967	0.999390827
3^o	0.05233595624	0.9986295348
4^o	0.06975647374	0.9975640503
5^o	0.08715574275	0.9961946981
6^o	0.1045284633	0.9945218954
7^o	0.1218693434	0.9925461516
...		

EXERCÍCIO 77 Definir um algoritmo que escreva a diferença entre o 21. número primo e o 13. número primo.

EXERCÍCIO 78 Resolver o seguinte problema usando o comando ESCOLHA. Numa organização, o número de dependentes limita a quantidade de dias que um empregado pode faltar anualmente. Assim, empregados com menos de 3 dependentes podem faltar 5 dias. Quem tem 4, 5 ou 6 dependentes pode faltar 8 dias, e quem tem mais de 6 dependentes pode faltar 11 dias. Escrever algoritmo que leia o número de dependentes e escreva a quantidade de dias que podem ser fruidos anualmente para cada caso.

EXERCÍCIO 79 Um concurso de vendas é realizado dentro de uma empresa. Por razões territoriais, cada venda unitária, dá direito a um bônus cujo valor depende do estado em que foi feita a venda, segundo a tabela:

PR e SC dão direito a	5 bônus
RS, SP e RJ	4 bônus
MG, GO, MS e TO	3 bônus
BA e PE	2 bônus
demais estados não citados	1 bônus

Definir algoritmo que receba uma série de vendas no formato (valor, estado) e para cada uma informe a quantidade de bônus de direito. Os dados terminam quando for lido um estado igual a 'XX' ou um valor de vendas negativo.

EXERCÍCIO 80 Definir um algoritmo que leia uma série de números inteiros e positivos (condição de fim: leitura de um negativo). O algoritmo deve calcular e imprimir a média dos números lidos EXCLUÍDOS os zeros.

EXERCÍCIO 81 Existe um conjunto de muitos números inteiros e positivos, agrupados três a três. Deve-se escrever um algoritmo capaz de:

- Ler os três números
- Identificar o maior deles, e rejeitá-lo
- Calcular a média entre os dois números restantes
- Imprimir os dois números selecionados e a média.

A pesquisa termina quando o primeiro número dos três lidos, for negativo.

EXERCÍCIO 82 Dados conjunto de 20 pares de números, realizar sobre eles o seguinte processamento:

- Se ambos forem pares, eles devem ser somados
- Se ambos forem ímpares, devem ser multiplicados
- Se um for par e outro ímpar, deve-se imprimir zero.

Para cada conjunto lido, deve-se imprimir os dois números e o resultado obtido segundo as regras acima.

EXERCÍCIO 83 Uma eleição é anulada se a somatória de votos brancos e nulos é superior à metade dos votos totais. Suponha que numa eleição existem 3 candidatos e para apurá-la, deveremos ler um conjunto indeterminado de votos (números inteiros), usando a regra:

- 1 = voto no candidato número 1
- 2 = voto no candidato número 2
- 3 = voto no candidato número 3
- 8 = voto em branco
- 9 = voto nulo.

A entrada de dados termina quando for lido um voto igual a zero. Caso ocorra um empate, dever ser vencedor o candidato mais velho. As idades são: 1=55 anos, 2=58 anos e 3=47 anos. Escrever um algoritmo que:

1. Verifique se a eleição foi válida
2. Se foi, quem a venceu.

EXERCÍCIO 84 Escreva um algoritmo que leia uma sequëncia de números positivos (a leitura é um depois do outro, por restrição do problema não se pode usar um vetor) com condição de fim igual a leitura de um número negativo e escreva o último número lidos que tenha sido maior que 100.

EXERCÍCIO 85 Calcular a soma total e o produto total de uma série de números inteiros e positivos que serão fornecidos. A série termina quando for lido o número -1.

EXERCÍCIO 86 Escreva um algoritmo capaz de move atribuídas a atletas de ginástica olímpica. Tal como na prática, este algoritmo deve eliminar a maior e a menor notas, e a seguir calcular e imprimir a média entre as duas notas restantes.

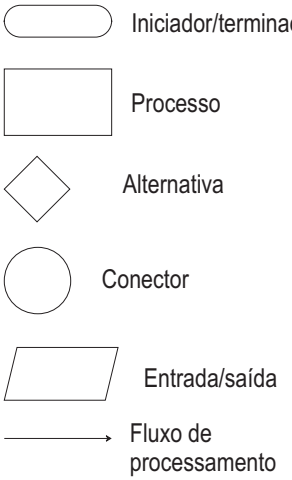
Capítulo 5

Nassi-Schneiderman

Fluxos e Nassi-Schneiderman

Além da maneira pela qual descreveremos nossos algoritmos (que é através do Portugol), existem outras formas de representar algoritmos. Falaremos de mais duas: a primeira é a linguagem dos fluxogramas. Tem importância histórica na informática, pois foi a primeira (e durante muito tempo única) representação de programas e similares.

A seguir uma lista dos principais símbolos usados em fluxogramas



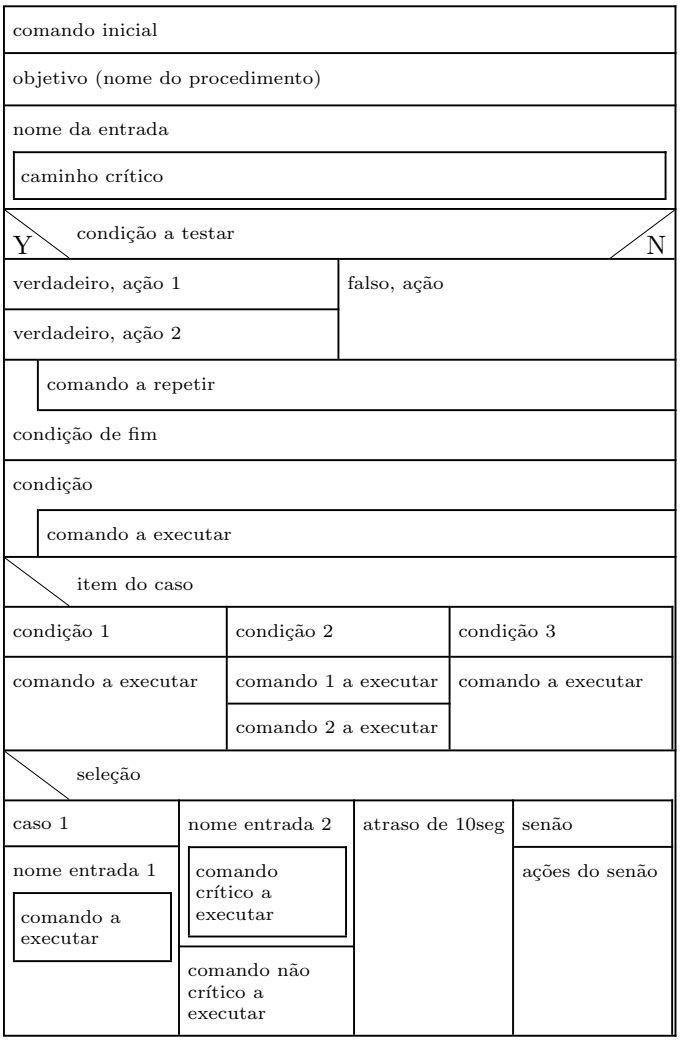
Fluxogramas perderam sua condição de únicas ferramentas, entre outras razões, pelas a seguir expostas:

- Prestam-se mal à programação estruturada, pois permitem a construção de algoritmos não estruturados.
- Exigem capricho, dando muito trabalho para serem feitos e mantidos.
- São necessárias quantidades enormes de papel, pois há um overhead muito grande, isto é, só se escreve dentro dos quadrinhos, e boa parte do papel se perde.
- Embora hoje já existam programas para fazer e guardar fluxos em computador (como por exemplo o FLOWCHART e o AUTOFLOW), eles não são muito práticos para este trabalho. Assim para seguir a tendência moderna (que é guardar matriz da documentação em computador e não em papel) o fluxograma resulta incômodo.

Outra possibilidade de representação é através dos diagramas de NASSI & SHNEIDERMAN, também conhecidos como mapas de CHAPIN ou estructogramas. Alguns autores negam importância a este trabalho, afirmando tratar-se de pseudo-código (no caso, PORTUGOL), apenas escrito de outra maneira. Outros, reconhecem força e inovação nesta idéia.

A idéia destes diagramas é formatar em um retângulo todas as informações do processamento usando marcas que identifiquem as diversas operações permitidas. Tem a vantagem sobre os fluxogramas de não permitirem o desvio. Os símbolos usados são:

nome da estrutura — objetivo



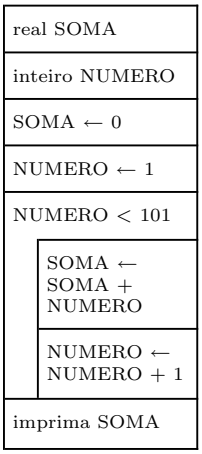
A seguir, alguns exemplos de algoritmos feitos em um e noutro esquema

Seja o seguinte algoritmo

```
Algoritmo Calcular {a soma dos números inteiros até 100}
real SOMA
inteiro NUMERO
SOMA ← 0
NUMERO ← 1
enquanto {NUMERO < 101}
```

```
SOMA ← SOMA + NUMERO
NUMERO ← NUMERO + 1
fim{enquanto}
imprima SOMA
fim{algoritmo}
```

calcular — soma os 100 primeiros números



5.1 Chines

Também conhecido como teste de mesa, o chines é a execução manual, passo a passo, de um algoritmo pelo programador. Em geral, é através do chines que o autor do algoritmo se certifica de que ele funciona.

Infelizmente, o chines costuma apontar onde e quando o algoritmo falha, mas não garante de que o algoritmo esteja 100% correto, pois o caminho eventualmente errado pode não ter sido seguido durante o chines.

As regras a seguir para um proveitoso chines, são:

Obter uma massa de dados significativa O conjunto a testar não deve ser extenso, mas precisa ser abrangente e incluir as condições que estão sendo testadas. Escolher bem os dados a testar é uma arte. Se mal escolhidos não vão certificar nada e só vão dar trabalho.

Obter o resultado correto Com os dados de entrada em mãos, o programador precisa obter o resultado esperado que estes dados vão gerar. Como fazer isto, varia de caso a caso, mas é imprescindível que antes de começar o chines, se saiba qual resultado esperar.

Listar os parâmetros e variáveis Deve-se providenciar no alto de uma folha de papel, o nome de todos os parâmetros e variáveis que o algoritmo manuseia. Deve haver bastante espaço abaixo dos nomes, a fim de que os valores que irão sendo lançados, caibam.

Seguir o algoritmo Em geral, com a ponta cega de uma lapiseira, o programador segue cada um dos comandos escritos. Se houver a necessidade de uma pausa para

raciocinar, a ponta da lapiseira fica sobre o ponto de interrupção. A tabela de valores das variáveis vai sendo lida (consultada) e escrita (alterada) durante o chines

Decisão Quando o algoritmo e o chines terminam, um resultado é obtido. Então, ele é comparado com o resultado esperado, acima calculado. Se for diferente, um erro aconteceu. Raras vezes é no processo manual de obtenção do resultado, mas quando isso ocorre o algoritmo pode estar certo e é o resultado obtido que está errado. Mas, na maioria das vezes, é o algoritmo que está errado. É hora de procurar o erro e refazer todo o ciclo.

Se ambos são iguais, o algoritmo funcionou para este caso de teste. Quanto deste funcionamento adequado pode ser generalizado para qualquer caso de testes fica em aberto.

EXERCÍCIO 87 Faça um chines com o algoritmo a seguir:

```
1: inteiro função CALCU (inteiro A, B)
2: se A ≥ B então
3:    devolva A
4: senão
5:    devolva B
6: fimse
```

Capítulo 6

Visualg

O VISUALG é um programa que auxilia o aprendizado de algoritmos na medida em que permite a entrada e execução de um algoritmo escrito em portugol, conforme aqui descrito. Escrito pela empresa Apoio, pode ser obtido em www.apoioinformatica.com.br.

6.1 Regras de Visualg

Em visualg, valem as regras:

- cada comando de visualg deve estar em uma linha
- não há separadores (;), nem blocos (como { e {) nem goto
- todas as palavras reservadas não tem acentos ou cedilha
- não há separação entre maiúsculas e minúsculas nos comandos ou variáveis
- o limite de variáveis é de 500 (cada item de vetor conta como 1 variável)
- o ponto decimal é o ponto
- valores caractere estão entre aspas duplas
- O símbolo de atribuição é `<-`. Na verdade são dois símbolos
- Existe recursividade

Algoritmo É a unidade básica de escrita de algoritmos.

formato

```
1: algoritmo <nome>
2: var
3: variáveis
4: — se houver, a função entra aqui
5: inicio
6: comandos
7: fimalgoritmo
```

exemplo

```
1: algoritmo "teste"
2: var
3: N : inteiro
4: inicio
5: leia (N)
6: escreva (N * 2)
7: fimalgoritmo
```

Comentários são caracterizados por começar por \\.

Função A função é uma unidade não autônoma de programação. Ela sempre precisa receber seus dados (através de uma lista de parâmetros) e devolver um resultado a quem a chamou. Em Visualg são colocadas imediatamente antes da palavra `inicio` do bloco a que se referem.

Formato

```
1: funcao <nome>( <parametros>) : <tipo-resultado>
2: var
3: variáveis locais
4: inicio
5: comandos
6: ... retorne ...
7: fimfuncao
```

Exemplo

```
1: funcao "teste"(N : inteiro; A, B:real) : inteiro
2: var
3: inicio
4: retorne (N * 2)
5: fimfuncao
```

O <nome-de-função> obedece as mesmas regras de nomenclatura das variáveis. Por outro lado, a <seqüência-de-declarações-de-parâmetros> é uma seqüência de
[var] <seqüência-de-parâmetros>: <tipo-de-dado>
separadas por ponto e vírgula. A presença (opcional) da palavra-chave var indica passagem de parâmetros por referência; caso contrário, a passagem será por valor.

Por exemplo, eis alguns cabeçalhos de funções

```
1: funcao CALC22 (A, B : inteiro; C : real) : real
```

Definiu-se uma função de nome CALC22, que recebe 3 parâmetros, sendo os dois primeiros inteiros e o terceiro real. A função devolve um resultado real.

```
1: funcao EQA6 (X : inteiro) : inteiro
```

Definiu-se uma função de nome EQA6, que recebe um único número inteiro e devolve outro número inteiro.

```
1: funcao EXA (var X : inteiro)
```

A função EXA recebe um parâmetro inteiro. A presença da palavra var implica em que a passagem é por referência, ou seja o que a função modificar na variável X permanecerá modificado após o retorno da função EXA.

Nomes Ao construir algoritmos é necessário dar nomes a muitas coisas. A regra de construção de nomes é

- Uma única palavra
- Composta de letras e números
- Começando com uma letra
- Escrita em maiúsculo

Tipos Os tipos possíveis são 4: inteiro, real, lógico, caracter.

Tipo	Conteúdo	Exemplo
inteiro	qualquer número inteiro, variando entre $-\infty$ e $+\infty$	inteiro A A,B,C : inteiro
real	qualquer número inteiro ou não inteiro	A : real X,Y,Z : real
caracter	uma cadeia	A : caracter // o tamanho não é explícito
logico	somente pode conter os valores VERDADEIRO e FALSO	A : logico

Vetor Um vetor (ou matriz) é definido, escrevendo-se o nome, dois pontos (:) a palavra `vetor` e as especificações de repetição. Veja nos exemplos:

Formato	Exemplo
<nome> : vetor [inicio..fim] [inicio..fim] ... de <tipo>	AAA : vetor [1..10] de real BBB : vetor [0..4] [1..8] de inteiro

Comandos Os comandos em pseudo-código são poucos < − (recebe), leia, escreva, se...fimse, enquanto...fimenquanto, para...fimpara, repita...até, retorne e abandone.

Leia Serve para introduzir um dado externo para dentro do algoritmo. As variáveis citadas no comando já terão que ter sido definidas e tipadas.

formato	**exemplo**
leia (<lista de variáveis >)	leia (A)

Escreva Serve para produzir um dado como resposta do algoritmo. As variáveis citadas no comando já terão que ter sido devidamente atribuídas.

formato	**exemplo**
escreva (<lista de variáveis >)	escreva (”o valor procurado e ”,x)

Para variáveis numéricas, pode-se especificar var:tamanho:decimais.

< − Este comando permite inicializar uma variável ou alterar-lhe o conteúdo.

formato	**exemplo**
<variável> < − <variável> OU	A < − B
<variável> < − <constante> OU	B < − ”CURITIBA”
<variável> < − <expressão>	C < − A + 23

Expressão Numérica Qualquer combinação compatível de funções que acabem gerando um único resultado numérico.

Formato	O que faz	Exemplo
adição (+), subtração (−), multiplicação (∗), divisão real (/)	o usual da aritmética	A+3 A/2.5
divisão inteira (barra ao contrário)	só usa inteiros	`5\2` é 2
resto (percentagem)	o resto da divisão inteira	`5%2` é 1
abs(a:real):real	o valor absoluto de a	abs(3) é 3; abs(-3) é 3
arcxxxx(a:real):real	arco xxx	xxx pode ser cos, sen e tan
asc(s:caracter):inteiro	converte o caracter s em inteiro	
carac(c:inteiro):caracter	converte o byte c em caracter	
caracpnum(c:caracter):inteiro ou real		
compr(c:caracter):inteiro	deve ser o comprimento do string	
copia(c:caracter; p,n:inteiro):caracter	copia um substring	
xxx(a:real):real	funcao trigonometrica	xxx pode ser sen,cos,tan,cotan
a div b	divisao inteira de a po b	
exp(b,e)	eleva a base ao expoente	
grauprad(a:real):real	converte graus em radianos	
int(a:real):inteiro	converte real em inteiro	
log(a:real):real	logaritmo base 10	
logn(a:real):real	logaritmo base e	
maiusc(c:caracter):caracter	converte para maiuscula	
minusc(c:caracter):caracter	converte para minuscula	
a mod b	resto da divisao inteira de a por b	
numpcarac(n:inteiro ou real):caracter	converte numerica a caracter	
pi:real	devolve o valor de pi	
pos(subc,c:caracter):inteiro	devolve a posicao de subc em c	
quad(a:real):real	devolve o quadrado de a	
radpgrau(a:real):real	converte radiano para grau	
raizq(a:real):real	devolve a raiz quadrada	

Expressão Relacional Qualquer combinação compatível de relações que acabem gerando um único resultado lógico.

Formato	O que faz	Exemplo
Igual (=)	devolve VERDADEIRO se seus operandos são iguais e FALSO senão	3=3 é VERDADEIRO
Diferente (<>)	devolve VERDADEIRO se seus operandos são diferentes e FALSO senão	3<>3 é FALSO
Maior (>)	Em $A > B$ devolve VERDADEIRO se A é maior do que B	3>4 é FALSO
Menor (>)	Em $A < B$ devolve VERDADEIRO se A é menor do que B	3<4 é VERDADEIRO
Maior ou igual (>=)	Em $A >= B$ devolve VERDADEIRO se A é maior ou igual do que B	3>=3 é VERDADEIRO
Menor ou igual (<=)	Em $A <= B$ devolve VERDADEIRO se A é menor ou igual do que B	3<=3 é VERDADEIRO

Expressão Lógica Qualquer combinação compatível de expressões relacionais e/ou lógicas que acabem gerando um único resultado lógico.

Formato	O que faz	Exemplo
E	Em A E B, devolve VERDADEIRO se A e B são verdadeiros e devolve FALSO senão	VERDADEIRO E VERDADEIRO é VERDADEIRO Todas as outras combinações dão FALSO
Ou	Em A OU B, devolve VERDADEIRO se A ou B ou ambos são VERDADEIRO e devolve FALSO senão	FALSO OU FALSO é FALSO Todas as outras combinações dão VERDADEIRO
Não (NAO)	Inverte o valor lógico do operando	NAO VERDADEIRO é FALSO e NAO FALSO é VERDADEIRO

Se Este comando é denominado alternativo, pois permite escolher caminhos da programação dependendo de uma condição (expressão lógica). Note que o trecho entre `senão` e o comando imediatamente anterior a `fim{se}` são opcionais, razão pela qual no formato eles aparecem entre colchetes. Os comandos entre `então` e `senão` ou `fim{se}` (se não houver senão) serão executados apenas se a condição do comando for verdadeira. Se houver comandos entre `senão` e `fim{se}` os mesmos serão executados apenas se a condição for falsa.

formato

```
se <condição> entao
   comando_1
   comando_2
...
senao
   comando_11
   comando_12
fimse
```

exemplo

```
se A >= 4 entao
   B <- B + 1
   C <- C + D + A
   X <- 0
senao
   escreva (N * 2)
   Y <- 0
fimse
```

Enquanto Este comando permite a realização de laços (loops) dentro de programas. Começando o comando, a condição é avaliada. Se ela for falsa, há um desvio para o comando seguinte ao `fim{enquanto}`. Se a condição for verdadeira os comando internos ao `enquanto` são executados. Ao se encontrar o `fim{enquanto}` há um desvio incondicional ao início do `enquanto` e a condição inicial é reavaliada.

formato

```
enquanto < condição > faca
   comando_1
   comando_2
   ...
fimenquanto
```

exemplo

```
A <- 5
enquanto A <= 9 faca
   escreva (A)
   A <- A + 3
fimenquanto
```

serão impressos os valores 5 e 8.

Repita Este comando também permite a realização de laços (loops) dentro de programas.

formato
repita
comando$_1$
comando$_2$
...
ate < condição >

exemplo
A < − 5
repita
escreva (A)
A < − A + 3
ate A > 9
serão impressos os valores 5 e 8.

Para Este comando também permite a realização de laços (loops) dentro de programas. No início a variável citada no comando é inicializada com <constante1>. Depois é feita a condição. Se o passo está ausente ou é positivo, a variavel é testada para ≤ <constante2>. (Se o passo é negativo, o teste é com ≥). Se o resultado é VERDADEIRO os comandos internos são executados. Ao final deles, a variável é incrementada (ou decrementada se o passo é negativo) e depois há um retorno ao teste inicial. Quando o passo não é explicito, ele vale 1.

formato
para <var> de <constante1> ate <constante2> [passo <constante3>]
faça
comando$_1$
comando$_2$
...
fimpara

exemplo
para K de 3 ate 8 passo 2 faca
escreva (A)
fimpara
serão impressos os valores 3, 5 e 7.

Retorne Usado exclusivamente dentro de funções, tem a finalidade de devolver um resultado a quem chamou esta função. Ao contrário do "C", não necessariamente encerra a execução da função.

formato
retorne < expressão compatível >

exemplo
se A > 5 entao
retorne A
fimpara

Outros Comandos aleatorio
arquivo <nome-de-arquivo>
algoritmo "lendo do arquivo"arquivo "teste.txt"
timer on / timer off
pausa
debug
eco
cronômetro

6.2 Exemplos de Visualg

Exemplo 1

```
algoritmo "primos"
var
J,K,R: inteiro
```

```
funcao QP(N: inteiro): inteiro
var
A,B:inteiro
inicio
A <- 2
B <- 1
enquanto B <= N faca
   se EPRIMO(A) entao
      B <- B + 1
   fimse
   A <- A + 1
fimenquanto
retorne A - 1
fimfuncao

funcao EPRIMO(M: inteiro): logico
var
QT,DI:inteiro
inicio
QT <- 0
DI <- 2
enquanto DI < M faca
   se M % DI = 0 entao
      QT<- QT + 1
   fimse
   DI <- DI + 1
fimenquanto
retorne QT = 0
fimfuncao

inicio
leia (J,K)
R <- QP(K)-QP(J)
escreva (R)
escreva (QP(K))
escreva (QP(J))
fimalgoritmo
```

Exemplo 2

```
algoritmo "palito"
var
N,SEQ,J,TV:inteiro
inicio
N<-0
enquanto ((N<20) ou (N>30)) faca
   escreval ("com quanto comecamos ?")
   leia (N)
fimenquanto
enquanto (N>0) faca
   SEQ <- 1
   enquanto ((N-SEQ)>=0) faca
```



```
      SEQ <- SEQ + 4
 //     escreval ("depuracao... N=",N,"  SEQ=",SEQ)
   fimenquanto
   J<-4+(N-SEQ)
   se (J=0) entao
      J<-1
   fimse
   escreval ("eu joguei ",J)
   N<-N-J
   escreval ("Existem ",N," palitos")
   se (N<=0) entao
      escreva ("por incrivel que pareca,... perdi,... ")
      interrompa
   fimse
   TV<-0
   enquanto ((TV<1) ou (TV>3)) faca
      escreval ("jogue")
      leia (TV)
   fimenquanto
   N <- N - TV
   escreval("existem ",N," palitos")
   se (N<=0) entao
      escreval("burrao, eu ganhei...")
      interrompa
   fimse
fimenquanto
fimalgoritmo
```

6.2.1 Criando uma raiz quadrada

Suponha que o VisualG não tem a função "raiz quadrada". Como poderíamos construir uma ? Eis uma possível formulação

> Dado um número real x, $\quad x > 0$, calcular a raiz quadrada dele.

Pensando sobre o que já aprendemos sobre a raiz quadrada de x, tem-se que y é a raiz quadrada de x se e somente se $y^2 = x$. Ou reescrevendo, $y = \frac{x}{y}$. Isto nos dá uma idéia de algoritmo:

- Chute um valor g para y e teste-o.
- Calcule $\frac{x}{g}$
- Se $\frac{x}{g}$ é suficientemente perto de g, retorne g, senão tente com um chute melhor.

Se o processo estiver funcionando adequadamente, o chute inicial poderá ser qualquer número, então vai-se padronizar o primeiro chute como sendo 1. Com isso a função **sqrt** em VisualG fica

```
funcao sqrt(N:real):real
   retorne teste(N,1)
fimfuncao
```

A seguir, escreve-se a função **teste** e fica

```
funcao teste(X,G:real):real
inicio
   escreval(X,G)
   se perto((X/G),G) entao
      retorne G
   senao
      retorne teste(X, melhor(X,G))
   fimse
fimfuncao
```

Sobram 2 funções, **melhor** e **perto**. Ei-las:

```
funcao melhor(X,G:real):real
inicio
   retorne (G+(X/G))/2  //a media entre G e X/G
fimfuncao

funcao perto(A,B:real):logico
inicio
   retorne (B*0.001) > abs(A-B)
fimfuncao
```

6.2.2 Achando raízes por Newton-Raphson

Agora o algoritmo é para achar raízes n-ésimas. Ou seja, dados w e n quaisquer, o que se pretende agora é achar $\sqrt[n]{w}$. Para isto, deve-se procurar x tal que $x^n = w$. Reescrevendo $x^n - w = 0$.

Portanto, em $f(x) = x^n - w$ procura-se encontrar o valor de x quando $f(x) = 0$, isto é, onde a curva corta o eixo dos x.

Em outras palavras e mais genericamente, dada $f(x)$ como descobrir onde a função corta o eixo x?. Para poder resolver este problema far-se-ão 3 hipóteses:

- a função cruza o eixo x apenas em um lugar
- a função é contínua, vale dizer é diferenciável
- a derivada no intervalo é sempre positiva ou sempre negativa

Supondo uma função qualquer (sujeita às restrições acima), imagina-se um chute original g (no eixo x) e $f(g)$ no eixo y. Calculando a derivada no ponto g e verificando onde esta derivada (que nada mais é do que a inclinação da curva no ponto g) cruza o eixo x obtem-se um novo e melhor chute para a raiz procurada. Este, em essência, é o método de Newton.

Eis o desenvolvimento, para achar a raiz n do valor w

```
funcao raiz(w:real;n:inteiro):real
inicio
   retorne acharn(w,1,n)
fimfuncao

funcao acharn(w,g:real;n:inteiro):real
var
novo:real
inicio
```



```
   novo <- g-(f(w,g,n) / fprime(g,n))
 //  escreval(w,g,n,novo)
   se perton(novo,g) entao
      retorne novo
   senao
      retorne acharn (w,novo,n)
   fimse
fimfuncao

funcao f(w,g:real;n:inteiro):real
inicio
   retorne (g^n)-w
fimfuncao



funcao fprime(g:real;n:inteiro):real
inicio
   retorne n * (g^(n-1))
fimfuncao

funcao perton(a,b:real):logico
inicio
   retorne abs(a-b) < abs(b * 0.01)
fimfuncao
```

6.2.3 Depuração de algoritmos

Na atividade profissional ou amadora da construção de programas eficientes e eficazes, a programação ocupa apenas uma parte. E, não é a maior nem a principal parte. O que toma tempo e preocupação é a atividade de depurar os programas. Entende-se por depuração **a arte da descoberta, isolamento e correção de erros**.

Aliás, o programador mais talentoso não é o que comete poucos ou nenhum erro (impossível), mas aquele que – em os cometendo – localiza-os e corrige-os com rapidez.

Programar é ir das causas para o resultado, e depurar é ao contrário, ir do resultado para as causas. Só que nesta segunda parte, conspiram contra a atividade:

- o ruído ambiente
- o fato do programa poder ter sido feito por outro
- a pressão do tempo e do $.

EXERCÍCIO 88 Na véspera desta aula, os alunos serão divididos em 4 grupos. A cada grupo será dado um conto de Sherlock Holmes, SEM A PARTE FINAL do mesmo. O grupo terá entre 5 e 10 minutos para contar a história e toda a turma será convidada a descobrir o que aonteceu.

Depois deste prazo, o professor contará o final da história.

Capítulo 7

Modularização: funções

7.1 Funções

Existem 2 maneiras de definir um algoritmo. A primeira, é definindo um código completo, auto-suficiente, capaz de ser facilmente traduzido em um programa completo e auto-suficiente capaz de ser rodado em computador gerando resultados ao instante. Este algoritmo é por um cabeçalho reconhecido pela palavra constante

```
algoritmo <nome>
```

Note que a especificação `<nome>` deve ser substituída por um nome que identifique este algoritmo. O bloco se encerra pela especificação

```
fimalgoritmo
```

Entre estas duas constantes, deve ser escrito o bloco do algoritmo que sempre será formado por 2 componentes: a descrição de variáveis e os procedimentos que o algoritmo fará. Eles devem aparecer nesta ordem: primeiro a definição de variáveis e depois os procedimentos.

Uma dificuldade aqui é a necessidade de obter os dados do algoritmo do meio ambiente (possivelmente fazendo uma operação de leitura) e gerar os dados de saída para serem aproveitados, usualmente através de uma operação de impressão.

Como uma função Em computação, o uso da palavra **função** é ligeiramente diferente do uso similar na matemática, embora os 2 conceitos tenham muitas similaridades. Aqui, o uso da palavra função indicará um sub-programa separado do programa principal que computa alguns valores. A especificação do cabeçalho de uma função será

1: função <nome> ([var] <lista-vars> : <tipo> [;...]) : <tipo-resultado>

Com a seguinte regra de leitura:

<nome>	tal como no algoritmo, identifica um nome único
<lista-vars>	uma seqüência de nomes de variáveis separadas por vírgulas. Se houver a palavra var no início a passagem será por referÊncia e senão por valor
<tipo>	tipo das variáveis da lista anterior
;	separador em as listas de variáveis (Se mais de uma houver)
<tipo-resultado>	especifica qual o tipo que será devolvido por esta função após os cálculos que realizará. Pode ser um dos 5 tipos conhecidos até agora

A função de encerra pela especificação

```
fimfunção
```

A parte de definição de variáveis locais em uma função nem sempre é necessária, logo é opcional, mas se presente deve aparecer logo após o cabeçalho da função e antes dos procedimentos.

Esta definição é mais simples do que a do algoritmo completo. Os dados não precisam ser obtidos do exterior, eles serão entregues à função simplesmente por referenciá-los no cabeçalho da função. Identicamente não é necessário imprimir resultados bastando retorná-los (a quem chamou a função).

Acompanhe um mesmo problema resolvido dos dois jeitos. Seja escrever um algoritmo que calcule a média entre dois números

como algoritmo	como função
algoritmo MEDIA real A B M leia A B M ← A + B / 2 escreva M fimalgoritmo	função MEDIA (A, B : real) : real retorne (A + B) / 2 fimfunção

Do ponto de vista da disciplina de algoritmos, ambos os resultados acima são equivalentes. Entretanto, se alguém quisesse testar a segunda implementação (a função), teria que escrever o programa chamador, que poderia ter o seguinte aspecto:

```
algoritmo T2
real A B
leia A B
escreva MEDIA (A, B)
fimalgoritmo
```

O conceito de função permite a modularização dos algoritmos. Deve-se definir e usar uma função sempre que:

- Um processamento específico tiver que ser feito em diversos locais do algoritmo
- Um processamento for complexo e auto-contido
- Outros algoritmos possam vir a usar este mesmo processamento
- etc

7.1.1 Variáveis globais e locais

Uma variável global existe desde que é criada e pode ser consultada e alterada por qualquer algoritmo ou função que rode enquanto ela existe. Dito de outra maneira, todas as funções subalternas que forem chamadas após a criação da variável global a enxergarão e poderão modificá-la.

O uso de variáveis globais deve ser fortemente reprimido, pois elas são responsáveis por um dos maiores eventos causadores de erros em programas, o chamado efeito colateral. Tanto isto é verdade, que a grande vantagem do paradigma funcional de programação é não ter variávis globais. E, com isto ele acaba gerando programas que têm uma quantidade inacreditavelmente menor de erros.

Quando um programa define variáveis e em seguida chama uma função, aquelas variáveis que o programa definiu são consideradas como globais pela função chamada. Se esta primeira função chamar uma segunda função, esta última também enxergará as variaveis definidas pelo programa, como sendo variáveis globais.

Já a variável local a uma função é definida dentro da função e quando esta função se encerrar o programa original não terá mais acesso a elas. Entretanto se esta primeira função chamar uma segunda, as variáveis definidas como locais dentro da primeira função passam a ser globais e são enxergadas pela segunda função.

A função pode ter suas próprias variáveis internas, também conhecidas como variáveis locais à função. Existe um comando, também opcional, chamado "retorne", que significa devolver para o algoritmo que chamou esta função, qual o valor calculado por ela. O tipo deste resultado deve ser igual aquele especificado depois da palavra "função".

Toda função usada dentro de um programa deve ter sido definida antes de ser chamada.

Em resumo, a regra de alcance das variáveis é: uma variável definida em um local é global para todas as funções chamadas a seguir, mas não é enxergada pelas funções chamadas previamente.

Por exemplo:

```
1: algoritmo ALFA
2: var
3: A, B : inteiro
4: funcao BETA (C, D : inteiro) : inteiro
5: var
6: E : inteiro
7: inicio
8: ...
9: fimfuncao {funcao BETA}
10: inicio
11: ...
12: fimalgoritmo
```

Aninhamento de Funções Quando existe uma função dentro de outra, as variáveis da função externa são locais para ela e globais para as funções internas. Já as variáveis da função interna são locais a ela e são desconhecidas pela função externa.

Quando existe conflito de nomes (duas variáveis com mesmo nome, sendo uma local e outra global), vale a definição local, e a global deixa de poder ser acessada (Embora continue existindo normalmente). Mas, este é um procedimento que deve ser evitado pelo programador, tanto quanto possível.

A especificação dos parâmetros nada mais é do que uma definição de todas as variáveis que existem na lista-de-parâmetros.

EXERCÍCIO RESOLVIDO 7 Um exemplo de algoritmo Escreva uma função que receba 2 números inteiros, X e Y, (X (Y) e devolva a diferença entre o X-ésimo e o Y-ésimo números primos. Por exemplo, se X = 10 e Y = 8 a função deverá devolver 6, pois o 10^{o} número primo é 23 e o 8^{o} é 17 e 23 - 17 = 6

Verificação se um número é primo:

Entrada: um inteiro X qualquer
Saída: .V. se X for primo e .F. senão

```
1: lógico função EPRIMO (inteiro X)
2: inteiro DV, QT
3: QT ← 0
4: DV ← 2
5: enquanto (DV ≤ (X div 2)) faça
6:    se X mod DV = 0 então
7:       QT++
```



```
 8:    fimse
 9:    DV++
10: fimenquanto
11: retorne QT = 0
12: fimfunção
```

Localização do n-ésimo primo:

Entrada: um inteiro K qualquer
Saída: o k-ésimo numero primo

```
 1: inteiro função PRIMO (inteiro K)
 2: inteiro Z, QTD
 3: QTD ← 1
 4: Z ← 1
 5: enquanto (Z ≤ K) faça
 6:    se EPRIMO(QTD) então
 7:       Z++
 8:    fimse
 9:    QTD++
10: fimenquanto
11: retorne QTD - 1
12: fimfunção
```

Diferença entre 2 números primos:

Entrada: dois inteiros M e N
Saída: a diferença entre o m-ésimo e o n-ésimo números primos

```
1: inteiro EXEMPLO1 (inteiro M, N)
2: retorne PRIMO(M) - PRIMO(N)
3: fimfunção
```

Capítulo 8

Vetores e Matrizes

Se partirmos da premissa que algoritmos sozinhos são incapazes de ajudar a resolver problemas, necessitando de completas estruturas de definição de dados, veremos que os quatro tipos de dados vistos até aqui (inteiro, real, alfanumérico e lógico) são insuficientes.

Por exemplo, vejamos o caso de um programa capaz de calcular a diferença entre o salário e a média salarial de um grupo de 50 pessoas. O algoritmo em si é fácil, entretanto se faz necessário:

- a criação de 50 variáveis para conter os salários, permitir o cálculo da média, e depois das diferenças individuais, OU
- A dupla leitura dos dados, a primeira para calcular a média, e a segunda para obter as diferenças individuais.

Nenhuma das duas alternativas é conceitualmente correta, pois ambas complicam desnecessariamente algo que é simples. A solução para resolver este e inúmeros problemas similares, e a introdução do conceito de vetor.

A característica principal do vetor , é a criação de muitas variáveis de mesmo tipo, que atenderão pelo mesmo nome, e que serão acessadas pelo seu deslocamento dentro do vetor, ou seja pelo seu número.

No caso acima (dos 50 salários), ao usar a solução de vetores, criaríamos uma única variável de nome SALAR, na forma de vetor, contendo 50 ocorrências de uma variável tipo real. O conjunto poderia ser referenciado na totalidade com o nome de SALAR, ou cada um dos valores poderia ser acessado individualmente através do seu índice.

A principal característica que determina o uso de vetores para resolver problemas de programação é o seu acesso direto ou via índice. Isto significa que, em um vetor, dado um valor de índice, é imediata o acesso ao elemento do vetor, seja para obtê-lo (leitura) seja para alterá-lo (escrita).

Um vetor é uma coleção HOMOGÊNEA de coisas. Essas coisas são referenciadas pela sua posição dentro do vetor, já que são todas iguais. A cadeia é um vetor de caracteres, mas é tão comum que acaba sendo um tipo fundamental. Não precisaria.

8.1 Definição de Vetor

Na criação de vetores , apenas é necessário indicar qual o fator de ocorrência daquela variável. Este valor numérico inteiro, será escrito entre colchetes logo após o nome do vetor. Cuidado, que aqui o colchete não significa ”opcional”. Azares de quem tem um universo limitado de caracteres a utilizar para escrever seus algoritmos.

Esta especificação indica múltiplas ocorrências de uma variável em uma única dimensão

<nome> : vetor [inicio..fim] de <tipo> // exemplo: A : vetor [1..10] de inteiro

Exemplos Vejamos alguns exemplos:

1. Quantidade de carros em cada dia da semana
   QTD-CARR : vetor [1..7] de inteiro
2. Valor recebido por dia ao longo do mes
   VALOR : vetor [1..31] de real

EXERCÍCIO RESOLVIDO 8 Seja definir um vetor para conter as taxas inflacionarias dos últimos 12 meses:

INFLACAO : vetor [1..12] de real

EXERCÍCIO RESOLVIDO 9 Imaginemos agora a utilização de um vetor para conter as notas de uma turma de 60 alunos:

NOTA : vetor [1..60] de real

EXERCÍCIO RESOLVIDO 10 Podemos ter um vetor para guardar as populações brasileiras, ano a ano, nos últimos 20 anos:

POPAA : vetor [1..20] de inteiro

Exemplo:

- V : vetor [1..5] de inteiro {definição}
  V ← 20 10 35 65 47. O vetor V tem 5 inteiros. V[1] é 20, V[4] é 65. V[6] é erro.
- X : vetor [1..9] de caracter ou cadeia X {no segundo caso o tamanho é implícito}
  X ← "ABCDEFGHI". O vetor X tem 9 caracteres (é portanto, uma cadeia). X[1] é "A". X[5] é "E". X[32] é erro.
- A : vetor [1..3] de logico
  A ← verdadeiro, falso, falso. A é um vetor de lógicos. A[2] é falso.
- F : vetor [1..6] de real
  F ← 10.9 10 11 11.2 13 19. F é um vetor de reais. (Lembrar que um inteiro é um real também. V[1] é 10.9, V[2] é 10.0, e assim por diante.

O índice em um vetor tem que obrigatoriamente ser:

1. inteiro e positivo
2. menor ou igual ao tamanho do vetor

8.1.1 Origem dos índices

A origem da contagem varia entre 0 e 1. Nós sempre que NADA for dito em contrário, usaremos 1. Quando a origem é zero, conta-se: 0, 1, 2, ... n-1 (para contar n elementos). Quando a origem é 1, a contagem é a trivial: 1, 2, ... , n. Repare que a notação [inicio..final] a rigor permitiria quaisquer valor inicial e final, mas o mundo da programação já é suficientemente complexo. Assim, vai-se combinar que o início semore será 1 (como no COBOL, PASCAL, BASIC, APL, ...), ou mais raramente 0 (como no C, C++, Java, ...), mas nunca algo diferente de 0 ou 1.

8.2 Operações Básicas

Vetores podem ser processados coletivamente ou elemento a elemento. Por exemplo:

V : vetor[1..6] de inteiro
V ← 0 {coletivamente}
V ← 10,20,30,40,50,60 {idem}
**** OU ****
I ← 1
enquanto I ≤ 6 **faça**
  V[I] ← I × 10 {elemento a elemento}
  I ← I + 1
fimenquanto

Na leitura é a mesma coisa

Y : vetor[1..5] de real
leia Y {5 pfs são lidos e colocados em Y}
**** OU ****
I ← 1
enquanto I ≤ 5 **faça**
  leia Y[I] {leitura individual de cada elemento}
  I ← I + 1
fimenquanto

Atribuição

Trata-se de modificar o conteúdo dos elementos do vetor.

```
1: A[i] ← A[i] + 1
```

Inicialização

É a colocação do valor inicial dos elementos de um vetor , usualmente antes de começar o processamento.

```
1: B[j] ← 0
```

Enumeração

Percorre-se todos os elementos de um vetor, executando sobre eles uma mesma operação. Por exemplo, seja contar quantos elementos são negativos.

```
1: inteiro VALS[200]
2: inteiro I, QTOS
3: leia VALS
4: QTOS ← 0
5: para para I de 1 a 200 faça
6:    se VALS[I] ≥ 0 então
7:       QTOS++
8:    fimse
9: fimpara
```

Acumulação

Parecido com a enumeração, mas com o caráter de acumulação sobre os resultados da operação. Exemplo: seja somar todos os elementos de um vetor


```
1: inteiro VALS[200]
2: inteiro I, SOMA
3: leia VALS
4: SOMA ← 0
5: para para I de 1 a 200 faça
6:    SOMA ← SOMA + VALS[I]
7: fimpara
```

Inserção

Colocação de um novo elemento dentro do vetor. Usualmente, este novo elemento é encaixado em alguma posição do vetor, e todos os elementos anteriores que tinham índice igual ou maior a esta posição são deslocados uma posição. Exemplo: Seja manter um vetor de 10 notas, com as melhores notas até então aparecidas.

```
1: NOTAS : vetor[1..10] de real
2: NOTAS ← ... as notas até aqui
3: NOVANOTA
4: I ← 1
5: enquanto (I < 11) ∧ (NOVANOTA < NOTAS[I]) faça
6:    I ← I + 1
7: fimenquanto
8: se I = 11 então
9:    escreva ("a nota nova e menor do que as 10 já existentes)
10: senão
11:    J ← 9
12:    enquanto (J > 0) ∧ (NOTAS[J] < NOVANOTA) faça
13:       NOTAS[J+1] ← NOTAS[J]
14:       J ← J - 1
15:    fimenquanto
16:    NOTAS[I] ← NOVANOTA
17: fimse
```

Supressão

Operação inversa à da inserção. Retira-se um elemento, trazendo todos os que estavam uma posição abaixo dele uma posição acima.

Busca

No acesso direto, dado um índice, obtemos o valor do elemento no vetor. Aqui ocorre o contrário. Dado um valor do vetor, queremos saber qual seu índice (isto é, onde ele está?).

Ordenação

Reordenar os elementos de um vetor com vistas a deixá-los obedecendo alguma relação de ordem.

Acesso aos dados dentro de um vetor

Dado um vetor, seus valores podem ser acessados das seguintes maneiras:

Genericamente

Neste caso, considera-se um único valor para todas as ocorrências do vetor. Por exemplo, para zerar um vetor pode-se fazer:

```
1: inteiro VETVEN [10]
2: VETVEN ← 0
```

O comando VETVEN ← 0 é permitido e faz com que TODOS os 10 elementos de VETVEN sejam zerados.

Conjuntamente

Neste caso, um conjunto da de mesmo número de elementos é usado na referência ao vetor. Exemplo:

```
1: real VETVEN [10]
2: VETVEN ← (1 2 3 4 5 6 7 8 9 0)
```

Neste caso o conjunto (1 2 3 4 5 6 7 8 9 0) é usado para inicializar o vetor de uma só vez com conteúdos diferentes. Os valores devem aparecer entre parênteses, e devem ser em mesmo número do que a definição original do vetor

Individualmente

É o caso mais comum, e pressupõe a existência de um comando para cada elemento do vetor.

```
1: real VETVEN [10]
2: VETVEN [1] ← 0.1
3: VETVEN [2] ← 0.01
4: VETVEN [5] ← 1
5: ...
6: VETVEN [8] ← 100
7: VETVEN [9] ← 10
8: VETVEN [10] ← 1000
```

Estas 3 maneiras de referenciar, não se aplicam apenas à inicialização, mas também as demais ações possíveis em . Assim, podemos ter testes: "se"VETVEN = 0 - significando "se"todos os elementos de VETVEN, leituras: leia (VETVEN) - significando "leia"todos os elementos, gravação: escreva (VETVEN) - significando "escreva"todos..., etc.

Todos os comandos vistos podem e devem ser usados junto com o conceito de vetores. Entretanto, existe um que é muito orientado para estas estruturas. Trata-se do comando **para**. Vejamos um exemplo, em que o **para** é usado para calcular a soma dos elementos de um vetor.

```
1: algoritmo exemplo
2: inteiro I,SOMA
3: inteiro VET [120]
4: leia(VET)
5: SOMA ← 0
6: para I de 1 ate 120 faça
7:    SOMA ← SOMA + VET[I]
8: fimpara
9: escreva(SOMA)
10: fim algoritmo
```


Leitura e Impressão de Vetores

Em algoritmos , podemos ter a necessidade de ler ou imprimir um vetor de uma só vez. Então faremos:

```
1: inteiro VET1 [10]
2: ...
3: leia(VET1)
4: ...
```

O caso acima, é o mais comum, e nos assegura que o vetor inteiro foi lido sem complicação ou dificuldade. Entretanto, algumas vezes necessitaremos ler ou imprimir um vetor, elemento a elemento. Ficará então:

```
1: inteiro VET1[10]
2: inteiro I
3: ...
4: para I de 1 até 10 faça
5:    leia(VET1[I])
6: fimpara
```

Nem sempre é fácil acostumar-se a trabalhar com vetor. Os conceitos de índice e conteúdo costumam atrapalhar quem nunca lidou com isto. Mas com um pouco de prática verifica-se que a coisa não é muito complicada. Na tentativa de ajudar, vão aqui alguns exemplos. Sugiro que você os leia e os estude atentamente, até entender cada um e todos eles. Se tiver alguma dificuldade insanável, ou não conseguir entender o que está acontecendo, procure ajuda junto ao professor.

Exemplos:

Para executar estes exemplos valem as seguintes definições:

```
1: inteiro VET1,VET2,VETX [10]
2: inteiro I,J,K,L
3: VET1 ← (10 12 15 18 2 24 40 2 4 6)
4: VET2 ← (20 18 16 14 12 10 8 6 4 2)
5: VET3 ← (1 1 1 2 2 2 5 5 5 9)
6: I ← 3
7: J ← I+1
8: I ← 5
9: K ← 3
10: L ← 6
```

então

```
VET1[2]                          --->        12
VET2[J]                          --->        14
VET1[K+1]                        --->        18
VET1[K]+1                        --->        16
VET2[VET3[4]]                    --->        18
VET1[1+2+L]                      --->         4
VET2[22-VET2[2]]                 --->        14
VET3[VET1[10]-2]+1               --->         3
VET2[(3*VET1[5])-5]              --->        20
VET1[4+VET1[2]]                  --->        ERRO!
```

Um dos erros mais comuns e graves que acontece quando se utilizam vetores é o estabelecimento de um valor para o índice maior do que o comprimento total do vetor, ou em termos mais rigorosos, o erro acontece quando se usa para índice de um valor não compreendido no intervalo da definição do vetor. Este erro ocorre quase sempre em tempo de execução (o que torna mais difícil sua correção).

Exemplo

```
1: real W [20]
2: inteiro A
3: A ← 15
4: escreva(W[A+10])
```

Indireção

A indireção é uma dupla (ou tripla ou n-upla) indexação. Usualmente é representada por diversos colchetes. Veja no exemplo

```
1: escreva A[A[x]]
```

O que está se pedindo acima é a dupla indexação de A. A maneira de resolver o pedido (e descobrir o valor de A[A[x]]) e usando o valor de x, recuperar o conteúdo da posição A[x]. De posse deste valor usa-se-o como índice final para saber o valor de A[A[x]].

Se houver muita dificuldade de entender este conceito o mesmo pode ser desdobrado em

```
1: escreva A[A[x]]
```

é equivalente a

```
1: B ← A[x]
2: escreva A[B]
```

Tipos de erros em vetores

a) indice caindo fora do vetor. b) tipo do indice definido inadequadamente (ex: inteiro X[30]; caracter Y; Y <- '2'; X[Y] dá erro) Note que a maioria das linguagens exige inteiro (ou real mas sem decimais).

8.3 Ordenação

Importância do estudo da ordenação

- Na década de 70, James Martin sugeriu que 40% do tempo de TODOS os computadores é gasto neste algoritmo.
- O conceito de ordem é fundamental em programação
- Excelente tópico para estudar algoritmos e estudar comportamentos e complexidades

Antes de começarmos a estudar a classificação, vamos definir o problema:

Seja X um conjunto composto por i elementos entre os quais se pode estabecer uma relação de ordem. Dados X_i e X_j com $i \neq j$, sempre pode-se estabelecer $X_i > X_j$ ou $X_i = X_j$ ou $X_i < X_j$.

- X estará em ordem crescente se e somente se $X_i \leq X_j, \forall i < j$.
- X estará em ordem estritamente crescente se e somente se $X_i < X_j \forall i < j$.
- X estará em ordem decrescente se e somente se $X_i \geq X_j, \forall i < j$.
- X estará em ordem estritamente decrescente se e somente se $X_i > X_j \forall i < j$.

Um algoritmo de ORDENAÇÃO é aquele que recebe um X qualquer (possivelmente desordenado), permuta seus elementos e devolve X em ordem.

Tipicamente fazem parte de X_i um conjunto de informações. Neste caso, haverá uma parte de X_i denominada CHAVE e identificada por k (k=key) pela qual se fara a ordenação. Embora devamos ter em mente a existência dos outros campos, apenas a chave será tratada nos algoritmos.

Existem 2 classes de algoritmos de ordenação: os que trazem todos os dados para a memória (sort interno) e os que ordenam dados em mídia magnética seqüencial (sort externo). Note que neste caso o algoritmo apenas acessa uma parte do subconjunto total.

Um algoritmo de ordenação é estável se no caso particular em que se k[i] = k[j], a ordem original dos dados é preservada. É sempre bom que um método de ordenação seja estável.

Antes de prosseguir, vale uma lembrança. Para conjunto de dados pequenos (quanto?) não há necessidade de grandes pesquisas e deve-se usar o algoritmo mais simples possível.

O conjunto completo (completo?) de algoritmos de ordenação será visto e estudado no segundo ano (na disciplina de estrutura de dados), mas por enquanto vamos usar um algoritmo simples e fácil (mas ineficiente) para ordenar vetores:

Algoritmo de movimentação

```
1: inteiro[1000] {função} MOVIMENTA (inteiro V[1000])
2: inteiro Y[1000]
3: inteiro MAIOR, QUAL
4: inteiro I,J
5: para I de 1 até 1000 faça
6:    MAIOR ← −∞
7:    para J de 1 até 1000 faça
8:       se V[J] > MAIOR então
9:          MAIOR ← X[J]
10:         QUAL ← J
11:      fimse
12:   fimpara
13:   Y[I] ← X[QUAL]
14:   V[QUAL] ← −∞
15: fimpara
16: retorne Y
17: fim{função}
```

Este algoritmo é estável.

EXERCÍCIO 89 Exercício: faça um chinês com 4, 12, 8, 3, 9, 1, 5

8.4 Operações fundamentais na Informática

Dado um conjunto S de chaves, as 3 operações fundamentais em informática são as seguintes:

Pesquisa Dada uma chave desconhecida x, pesquisá-la contra S significa verificar se x pertence ou não ao conjunto S. Em termos matemáticos, $x \in S$? Esta função retorna um valor booleano.

Inserção Dada uma chave x que reconhecidamente não pertence ainda a S (e isto pode ser verificado pela operação acima), a operação de inserção significa inserir x em S. Ou $S \leftarrow S \cup \{x\}$. A função pode retornar apenas um indicativo de sucesso.

Exclusão Dada uma chave x que sabidamente pertence a S, a operação de exclusão significa retirar x do conjunto S. Ou $S \leftarrow S - \{x\}$. Pode-se retornar apenas um indicativo de sucesso.

Máximo Dado um conjunto S, descobrir-lhe seu máximo é obter o valor x, tal que $\forall y \neq x \in S, x \geq y$. Esta função retorna um ponteiro ou um cursor para o elemento x.

Mínimo Muito semelhante ao anterior, só que busca-se x tal que $\forall y \neq x \in S, x \leq y$.

Sucessor Esta operação busca descobrir qual o "seguinte"elemento do conjunto segundo um critério qualquer. A função recebe como entrada um valor x que está presente em S e devolve um apontador ou cursor para o número imediatamente maior a x em S. Outra hipótese de retorno é um terminador, caso x já seja o maior elemento em S.

Antecessor Esta operação busca descobrir qual o "anterior"elemento do conjunto segundo um critério qualquer. A função recebe como entrada um valor x que está presente em S e devolve um apontador ou cursor para o número imediatamente menor a x em S. Outra hipótese de retorno é um terminador, caso x já seja o menor elemento em S.

EXERCÍCIO 90 Palíndromo (adj.) Diz-se da frase que quer se leia da direita para a esquerda, quer da esquerda para a direita, tem o mesmo sentido (Aurélio). Exemplos de palíndromos:

sairam o tio e oito marias
oto come mocoto
socorram me subi no onibus em marrocos
roma me tem amor
em inglês: a man, a plan, a cat, a canal, panama

murder for a jar of red rum

rotor, radar,

me vê se a panela da moça é de aço madalena paes e vem

a diva da vida

luz azul

ato idiota

o treco certo

a base do teto desaba

atila toledo mata modelo talita

anotaram a data da maratona

a droga da gorda

o romano acata amores a damas amadas e roma ataca o namoro

só dá tapas e sapatadas

Definir algoritmo capaz de ler uma frase de até 80 caracteres e decidir se a mesma é ou não é um palíndromo. O algoritmo deve imprimir a letra "S"se for, e "N"se não for. Desconsiderar os brancos que estão à direita da frase e também aqueles que estão entre as palavras.

Para executar este exercício valem as seguintes definições:

1. inteiro Vet1[10] Vet2[10] Vetx[10]
2. inteiro I J K L
3. Vet1←(10 12 15 18 2 24 40 2 4 6)
4. Vet2←(20 18 16 14 12 10 8 6 4 2)
5. Vet3←(1 1 1 2 2 2 5 5 5 9)
6. I←3
7. J←I+1
8. I←5
9. K←3
10. L←6

EXERCÍCIO 91 Quanto é:

1. Vet1[2]
2. Vet2[J]
3. Vet1[K+1]
4. Vet1[K]+1
5. Vet2[vet3[4]]
6. Vet1[1+2+L]
7. Vet2[22-Vet2[2]]
8. Vet3[Vet1[10]-2]+1
9. Vet2[(3*Vet1[5])-5]
10. Vet1[4+Vet1[2]]

EXERCÍCIO 92 Escrever um trecho de algoritmo que defina um vetor de 7 elementos inteiros, e inicialize todos os elementos com o valor 5.

EXERCÍCIO 93 Escrever um trecho de algoritmo que defina um vetor de 365 elementos reais, e inicialíze todos os elementos com o valor 0.

EXERCÍCIO 94 Escrever um trecho de algoritmo que defina um vetor de 12 elementos lógicos, e inicialíze os primeiros 6 com VERDADEIRO e os outros 6 com FALSO.

EXERCÍCIO 95 Escreva um algoritmo capaz de definir e ler um vetor de 28 reais e a seguir calcular e imprimir a somatória dos elementos do vetor. Cada elemento deste vetor representa o valor da dívida de cada estado brasileiro para com a união, e o que se deseja obter impresso é o valor total desta dívida.

8.5 Tabelas

Pesquisa seqüencial

Dada um vetor, uma das ações mais comuns sobre ela, é a pesquisa para saber se determinado elemento se encontra ou não contido nela. A maneira mais simples de fazer esta pesquisa é percorrer a tabela em ordem seqüencial, até que:

- O elemento (denominado CHAVE) seja encontrado, ou
- Chegue-se ao fim da tabela, quando então concluiremos que o elemento CHAVE não existe na tabela.

Embora seja um algoritmo simples de se imaginar e de implementar, ele não é dos mais eficientes, principalmente para tabelas muito grandes e com muitas pesquisas por unidade de processamento.

Um primeiro melhoramento que podemos fazer, é classificar a tabela, para diminuir a quantidade de acessos até chegar a uma conclusão sobre a existência ou não da chave. Com a tabela classificada em ordem (digamos ascendente), a pesquisa da chave deve ser feita até que:

- A chave seja encontrada, OU
- Um elemento maior do que a chave seja encontrado, quando então concluiremos pela não existência da chave na tabela.

A seguir o algoritmo para fazer esta pesquisa:

```
 1: inteiro V[1000]
 2: inteiro I, CHAVE
 3: leia V
 4: leia CHAVE
 5: I ← 1
 6: enquanto I < 100 faça
 7:    se VET[I] = CHAVE então
 8:       escreva "encontrou"
 9:       abandone
10:    fimse
11:    I ← I + 1
12: fimenquanto
13: se I = 101 então
14:    escreva "Não encontrou"
15: fimse
```


16: fim?

EXERCÍCIO 96 Como exercício, indique qual a modificação que se deveria fazer no algoritmo acima, se o vetor lido estiver em ordem ascendente, a fim de não ter que percorrê-lo até o fim para certificar-se da não existência da chave.

EXERCÍCIO RESOLVIDO 11 Imagine um possível refinamento para tornar mais rápido o acesso seqüencial a uma tabela (de tamanho grande) classificada.

R: Basta aumentar o tamanho do salto, de 1 linha para 10, 20 ou qualquer outro valor.

Pesquisa binária

Trata-se da um significativo melhoramento em termos de pesquisa a tabela. Agora a tabela é percorrida aos saltos, e não elemento a elemento. A redução de acessos é grande e garante muita eficiência ao processo.

Para poder usar esta pesquisa a tabela PRECISA estar ordenada. Se não estiver os resultados da técnica são imprevisíveis.

Em resumo é o seguinte:

- A tabela é dividida ao meio (daí o nome bin ria)
- A chave é comparada com o elemento que dividiu a tabela. Se este for menor, a primeira parte da tabela é abandonada. Se for maior abandona-se a segunda parte.
- A pesquisa prossegue pela divisão ao meio da parte que ficou.
- Rapidamente as sucessivas divisões ao meio esgotam a tabela. O processo termina quando a chave for encontrada, ou quando a parte que restou da divisão for nula.

Vejamos no algoritmo abaixo

```
 1: inteiro V[100]
 2: inteiro INIC,METADE,FIM,CHAVE
 3: leia VET,CHAVE
 4: INIC ← 1 {campo delimita o limite inferior da pesquisa}
 5: FIM ← 100 {delimita o limite superior da pesquisa}
 6: repita
 7:    METADE ← int(INIC + FIM)/2
 8:    se CHAVE < VET[METADE] então
 9:       FIM ← METADE - 1
10:    senão
11:       INIC ← METADE + 1
12:    fimse
13: até VET[METADE] = CHAVE ∨ INIC > FIM
14: se VET[METADE] = CHAVE então
15:    escreva 'achou na posição ',metade
16: senão
17:    escreva 'não achou'
18: fimse
```

Vamos acompanhar um exemplo, (chinesinho) neste algoritmo. Seja o vetor: 1 7 12 20 40 41 47 49 60 88

Vamos fazer uma pesquisa binária para encontrar (ou não) a chave 77 nele.

primeira iteração: INIC ← 1, FIM ← 10, METADE ← 5. Como VET[5] é 40, e 40 < 77, temos: INIC ← METADE + 1 ou INIC ← 6.

segunda iteração: INIC ← 6, FIM ← 10, METADE ← 8. Como VET[8] é 49, e 49 < 77, temos: INIC ← METADE + 1 ou INIC ← 9.

terceira iteração: INIC ← 9, FIM ← 10, METADE ← 9. Como VET[9] é 60 e 60 < 77, temos: INIC ← METADE + 1 ou INIC ← 10.

quarta iteração: INIC ← 10, FIM ← 10, METADE ← 10. Como VET10] é 88 e 88 > 77, temos FIM ← METADE - 1 ou FIM ← 9.

Neste ponto o ciclo se encerra, pois INÍCIO > FIM. Finalmente descobrimos que a chave não está presente pois, VET [10] é 88 que é diferente de 77.

Note-se que o número médio de acessos em uma pesquisa binária é da ordem de $log_2 N$, onde N é o número de elementos da tabela.

EXERCÍCIO 97 Mediante a regra acima, determine qual o número médio de acessos em uma pesquisa binária para uma tabela de:

1. 5 elementos:
2. 10 elementos:
3. 20 elementos:
4. 50 elementos:
5. 100 elementos:
6. 1.200 elementos:
7. 15.000 elementos:
8. 1.000.000 elementso:

EXERCÍCIO 98 Dado o vetor: 1 4 5 6 8 10 13 18 21 60, realizar o chinez para descobrir a presença ou não das chaves, através do algoritmo de pesquisa binária:

1. 8
2. 9
3. 50
4. 200

8.6 Merge (intercalação) de dois vetores

Este é um problema tradicional em PD: Suponhamos que existam dois vetores, com as características:

- Idênticos lay-outs
- Uma chave de controle (em geral não repetida>
- Ambos classificados pela chave de controle.

O objetivo é intercalar os dois vetores formando um só, e que continue classificado. Por exemplo:


```
vetor 1                      vetor2
-----------------------      --------------------------
Chave 3                      Chave 1
Chave 8                      Chave 9
Chave 36                     Chave 10
Chave 87                     Chave 11
Chave 500                    Chave 450
                             Chave 490

Resultado
----------------------
Chave 1
Chave 3
Chave 8
Chave 9
Chave 10
Chave 11
Chave 36
Chave 87
Chave 450
Chave 490
Chave 500
```

EXERCÍCIO 99 Dadas duas datas no ano de 89, no formato dia,mês, escrever um algoritmo que as leia, escreva-as e calcule e escreva a diferença em dias entre ambas.

EXERCÍCIO RESOLVIDO 12 Dadas duas datas na década de 80, no formato dia,mês,ano, escrever um algoritmo que as leia, escreva-as e calcule e escreva a diferença em dias entre ambas.

```
 1: d1,m1,a1,d2,m2,a2,qt1,qt2 : inteiro
 2: V1, V1B [1..12] vetor de inteiro
 3: V2 [80..89] vetor de inteiro
 4: V1 ← (0,31,59,90,120,151,181,212,243,273,304,334)
 5: V1B ← (0,31,60,91,121,152,182,213,244,274,305,335)
 6: V2 ← (0,366,731,1096,1461,1827,2192,2557,2922,3288)
 7: qt1 ← qt2 ← 0
 8: leia (d1,m1,a1,d2,m2,a2)
 9: escreva (d1,m1,a1,d2,m2,a2)
10: se a2 = 80 ∨ a2 = 84 ∨ a2 = 88 então
11:    qt2 ← d2 + v1b[m2] + v2[a2]
12: senão
13:    qt2 ← d2 + v1[m2] + v2[a2]
14: fimse
15: se a1 = 80 ∨ a1 = 84 ∨ a1 = 88 então
16:    qt1 ← d1 + v1b[m1] + v2[a1]
17: senão
18:    qt1 ← d1 + v1[m1] + v2[a1]
19: fimse
20: escreva (qt2-qt1)
```

EXERCÍCIO 100 Definir algoritmo que leia um vetor de 28 reais onde cada elemento do vetor representa o valor da dívida de cada estado brasileiro para com a união. Os elementos

estão colocados em ordem decrescente (isto é as maiores dívidas na frente) e escreva a quantidade de estados que são necessários para apresentar uma dívida de no mínimo 50% do total ?

EXERCÍCIO 101 Definir um algoritmo equivalente para a leitura de um vetor de 200 elementos, seguido do cálculo e impressão da soma dos elementos que são maiores que 1000. Deve ser impressa também a quantidade de maiores de 1000.

EXERCÍCIO 102 Escrever um algoritmo que seja capaz de ler um conjunto de 10 valores, calcular a média, e imprimir todos os valores do conjunto, a média, e a diferença entre cada valor individual e a média calculada.

EXERCÍCIO 103 Escreva um algoritmo que leia um vetor de 36 elementos e determine qual maior, e qual seu índice, imprimindo ambas as informações, mesmo que haja mais de um.

EXERCÍCIO 104 Definir algoritmo que leia um vetor de 278 inteiros e escreva-o de trás para a frente.

EXERCÍCIO 105 Escreva um algoritmo, que leia de uma só vez um vetor de 100 elementos numéricos reais, e escreva aqueles que forem maiores do que o valor 150, e também a quantidade deles.

EXERCÍCIO 106 Escreva um algoritmo que leia 200 valores (um a um), e crie um vetor com tais valores, informando a seguir qual a média dos valores positivos e qual a média dos valores negativos.

EXERCÍCIO 107 Definir algoritmo que leia uma frase de 80 caracteres e escreva a quantidade de vogais minúsculas que apareceram na frase.

EXERCÍCIO 108 Definir algoritmo que leia uma frase de 90 caracteres e escreva a quantidade de consoantes maiúsculas que apareceram na frase.

EXERCÍCIO 109 Definir algoritmo que leia uma frase de 100 caracteres e escreva a quantidade de palavras que apareceram na frase.

8.7 Pesquisa seqüencial

Dada um vetor, uma das ações mais comuns sobre ela, é a pesquisa para saber se determinado elemento se encontra ou não contido nela. A maneira mais simples de fazer esta pesquisa é percorrer a tabela em ordem seqüencial, até que:

- O elemento (denominado CHAVE) seja encontrado, ou
- Chegue-se ao fim da tabela, quando então concluiremos que o elemento CHAVE não existe na tabela.

Embora seja um algoritmo simples de se imaginar e de implementar, ele não é dos mais eficientes, principalmente para tabelas muito grandes e com muitas pesquisas por unidade de processamento.

Um primeiro melhoramento que podemos fazer, é classificar a tabela, para diminuir a quantidade de acessos até chegar a uma conclusão sobre a existência ou não da chave. Com a tabela classificada em ordem (digamos ascendente), a pesquisa da chave deve ser feita até que:

- A chave seja encontrada, OU
- Um elemento maior do que a chave seja encontrado, quando então concluiremos pela não existência da chave na tabela.

A seguir o algoritmo para fazer esta pesquisa:

```
1: inteiro V[1000]
2: inteiro I, CHAVE
3: leia V
4: leia CHAVE
5: I ← 1
6: enquanto I < 100 faça
7:    se VET[I] = CHAVE então
8:       escreva "encontrou"
9:       abandone
10:   fimse
11:   I ← I + 1
12: fimenquanto
13: se I = 101 então
14:    escreva "Não encontrou"
15: fimse
16: fim?
```

EXERCÍCIO 110 Como exercício, indique qual a modificação que se deveria fazer no algoritmo acima, se o vetor lido estiver em ordem ascendente, a fim de não ter que percorrê-lo até o fim para certificar-se da não existência da chave.

EXERCÍCIO RESOLVIDO 13 Imagine um possível refinamento para tornar mais rápido o acesso seqüencial a uma tabela (de tamanho grande) classificada.

R: Basta aumentar o tamanho do salto, de 1 linha para 10, 20 ou qualquer outro valor.

EXERCÍCIO 111 Mediante a regra acima, determine qual o número médio de acessos em uma pesquisa binária para uma tabela de:

1. 5 elementos:
2. 10 elementos:
3. 20 elementos:
4. 50 elementos:
5. 100 elementos:
6. 1.200 elementos:
7. 15.000 elementos:
8. 1.000.000 elementso:

EXERCÍCIO 112 Dado o vetor: 1 4 5 6 8 10 13 18 21 60, realizar o chinez para descobrir a presença ou não das chaves, através do algoritmo de pesquisa binária:

1. 8
2. 9

3. 50
4. 200

EXERCÍCIO 113 Definir algoritmo que

- Leia um vetor de 6000 salários
- Leia o valor do salário mínimo vigente
- Calcule e escreva quantas pessoas (=salários) ganham até 3,5 S.M.
- Idem para quem ganha mais de 3,5 e menos de 20 S.M.
- Idem para quem ganha 20 ou mais S.M.

EXERCÍCIO 114 Definir algoritmo que leia um vetor de 200 inteiros e escreva os elementos que são maiores do que seus medonhos de 10 chifres. Define-se o "medonho de n chifres" do elemento índice k de um vetor, ao elemento do mesmo vetor, com o menor índice j possível (onde $j \geq k$) tal que j é divisível por n, se este existir.

Exemplos:

Dado o vetor `1 2 3 4 6 7 8 9 0 0 0 2 3`, pergunta-se qual o medonho de 3 chifres do elemento 6 (quinto elemento do vetor) ? k é igual a 5. O próximo j maior que cinco e divisível por 3 (3 chifres) é 6. Logo o medonho é o sexto elemento, que é o 7.

Se pedirmos o medonho do último elemento de um vetor, ele só existirá se o medonho for o próprio elemento (no caso de j ser igual a k). Isto só ocorrerá se a ordem do último elemento for divisível pelo número de chifres.

EXERCÍCIO 115 Definir algoritmo que leia um vetor de 200 elementos inteiros, e a seguir calcule e escreva a somatória dos elementos do vetor DE ORDEM PAR.

EXERCÍCIO 116 Definir algoritmo que leia um vetor de 200 elementos inteiros, e a seguir calcule e escreva a somatória dos elementos do vetor DE VALOR PAR.

EXERCÍCIO 117 Definir algoritmo que leia um vetor de 300 inteiros, e escreva-o sujeito à ordem de entrada, mas imprimindo primeiro os pares e depois os ímpares.

EXERCÍCIO 118 Dado um vetor VIN de 10 elementos inteiros numéricos, criar o vetor VAI contendo 30 elementos. Onde VAI[1], VAI[2] e VAI[3] recebem VIN[1], e assim por diante. Escrever algoritmo que leia VIN e escreva VAI.

EXERCÍCIO 119 Uma grande empresa tem seus preços armazenados na forma de dois vetores de 650 ocorrências cada. No primeiro, a quem chamaremos COD estão os códigos dos produtos [inteiros]. No outro vetor, chamado PRE, estão os preços de cada produto, NA MESMA ORDEM DE APARECIMENTO do que COD.

Escrever um algoritmo que leia os vetores contendo CÓDIGO e QUANTIDADE e escreva o valor a pagar, pela fórmula VALOR = PRE [I] * QUANTIDADE [I].

EXERCÍCIO 120 Dado um vetor ALFA contendo 50 números inteiros e fracionários misturados, criar um vetor BETA de mesmo tamanho, onde só aparecem os números inteiros no início, e com zeros ao final. [Tantos zeros quantos eram os fracionários em BETA]. Escrever um algoritmo que leia o vetor ALFA e escreva BETA.

```
Exemplo: ALFA:   3     1.5       4       2.2     10       11
         BETA:   3       4       10       11      0        0
```


E se ao final do vetor, se colocar 99, em vez de zero, que mudança precisaria ser feita ?

EXERCÍCIO 121 Definir algoritmo que leia um vetor de 1500 números reais positivos sem qualquer tipo de ordenação. O algoritmo deve gerar outro vetor também com 1500 números onde os elementos que forem menores do que 10000 serão colocados na frente, e onde os maiores ficarão no fim. Exceto esta mudança, a ordem original do vetor deve ser preservada.

EXERCÍCIO 122 Definir algoritmo que leia uma série indeterminada de pares de valores formados por CÓDIGO,VALOR referentes a arrecadações do imposto de renda. CÓDIGO é um entre 137 códigos possíveis, um para cada tipo de imposto. VALOR é o valor em reais efetivamente cobrado. Os dados terminam quando o código 000 for lido. Ao final do processamento, o algoritmo deve imprimir quantos reais foram arrecadados em cada um dos códigos que foram lidos. Os dados não tem nenhum tipo de ordenação.

Exemplos de códigos possíveis (fictícios):

```
0246 - Taxas alfandegárias
0211 - Mensalão
0192 - Tri-leão
0432 - Multas por recolhimento fora de prazo,
...
```

EXERCÍCIO 123 Desejamos analisar o mapa de uso de um disquete. Como sabemos um disquete dupla-dupla de 5 1/4, tem 2 faces, 40 trilhas e 9 setores por trilha, o que dá um total de 720 setores. Definir um algoritmo que leia 20 vetores (um de cada vez, é claro), que correspondem cada vetor a um disco diferente, e determine qual o percentual de setores livre em cada um dos discos, imprimindo tal resultado. O vetor é de inteiros, e se existe um valor zero, isto significa que o setor está livre. Qualquer valor diferente de zero, implica em que o setor não está livre.

EXERCÍCIO 124 Definir um algoritmo que leia um vetor numérico real de 100 elementos, onde estão misturados: números positivos e negativos e números inteiros e fracionários. O algoritmo deve criar um vetor de saída também de 100 elementos, onde a ordem é:

- Primeiro os números negativos fracionários
- Depois os positivos fracionários
- Depois os inteiros

Em cada uma das subclasses, a ordem original do vetor de entrada deve ser preservada. Exemplo:

```
Vetor de entrada: 1.2  3.5  -8  -8.9  0  0  1.2  -3.1  -4  -0.9
Saída: -8.9  -3.1  -0.9  1.2  1.2  3.5  1.2  0  0  -4
```

Após calcular o vetor de saída, este deve ser impresso e o algoritmo encerrado.

EXERCÍCIO RESOLVIDO 14 Uma empresa de departamentos, quer saber quais os produtos que ela vende e que rendem mais faturamento. As opções de vendas são: brinquedos (cod. 11), armarinhos (cod. 21), roupas (cod. 33) e produtos eletrônicos (cod. 40). O cadastro de vendas da empresa tem o seguinte formato:

```
TIPO,inteiro
VALOR, real
```

E nele consta 1 registro para cada venda efetuada, isto é, os códigos se repetem. Escrever um algoritmo que leia o cadastro de vendas e totalize os quatro tipos ao final.

Solução usando vetores:

```
 1: tipo R = registro;
 2: inteiro: ti;
 3: real:va;
 4: fimregistro;
 5: tipo V = vetor [1:4] R
 6: V: tab;
 7: R: reg;
 8: inteiro: I;
 9: tab[1].ti ← 11;
10: tab[2].ti ← 21;
11: tab[3].ti ← 33;
12: tab[4].ti ← 40;
13: leia (reg);
14: enquanto reg.ti ≠ 0 faça
15:    I ← 1;
16:    enquanto  tab[I].ti ≠ reg.ti faça
17:       I ← I + 1;
18:    fimenquanto
19:    tab[I].va ← tab[I].va + reg.va
20:    leia(reg);
21: fimenquanto
22: escreva(tab)
```

EXERCÍCIO 125 Suponha uma lista sequencial (um vetor numérico) de nome VET, definido globalmente, formada por números inteiros positivos e contendo espaço para 20 números. O preenchimento sempre se dá a partir do começo do vetor sem lacunas. Os dados válidos se encerram quando é encontrado um número negativo, que faz o papel de sentinela e que sinaliza "FIM DE DADOS". Obviamente cabem apenas 19 números válidos pois sempre há que se reservar espaço para o sentinela. Notar que logo após a sentinela os números que porventura aparecerem em VET são não confiáveis e não devem ser considerados, isto é, o vetor não foi inicializado de alguma maneira antes dele começar a ser usado.

A seguir está o algoritmo de uma função que recebe como parâmetro um determinado número inteiro positivo X, e devolve a posição dele dentro de VET (caso ele de fato existisse lá dentro) ou devolve o valor 20 o que deve ser entendido como "X não se encontra em VET".

```
 1: Inteiro função ACHANUMERO(inteiro X)
 2: Inteiro I, J, K
 3: I ← 1
 4: J ← 0
 5: enquanto I < 20  faça
 6:    se VET[I] = X então
 7:       J ← I
 8:       I ← 21
 9:    fimse
10:    I++
11: fimenquanto
12: se J = 0 então
13:    Retorne (20)
```



```
14: senão
15:    Retorne J
16: fimse
17: Fim função
```

Sobre o algoritmo acima, serão feitas 4 afirmações

1. É desnecessário inicializar J com o valor 0, pois J tem o valor imediatamente alterado, logo abaixo.
2. Se em vez de fazer I ← 21 se fizesse I ← 20 a função deixaria de funcionar
3. Esta função pode não funcionar no caso em que X não pertence a VET
4. Esta função pode não funcionar no caso em que X pertence a VET

```
Estão erradas as afirmações:
a) 1, 2, 3 e 4
b) 1, 2 e 3
c) 1, 2 e 4
d) 2 e 3
e) 1 e 4
```

EXERCÍCIO 126 Defina um algoritmo que leia e escreva um conjunto de 200 valores numéricos armazenando-os em uma vetor numérico (real). Feito isto, o algoritmo deve:

- calcular e imprimir a somatória dos termos: $\frac{j^2}{a_j}$, onde j é um índice que varia entre 1 e 200, e a_j é a variável que foi lida e armazenada no vetor.
- Calcular e imprimir a quantidade de elementos que são menores que seu índice

EXERCÍCIO 127 Definir algoritmo capaz de receber uma série da datas, sempre referentes ao ano de 1988, e no formato dd, mm. Para cada uma delas, o algoritmo deve calcular uma nova data, no mesmo formato, e que corresponda ao período de 180 dias posterior a data original. A série de datas se encerra quando for lida um dia igual a zero.

EXERCÍCIO RESOLVIDO 15 Criar um algoritmo que leia (de uma vez) um vetor de 500 elementos inteiros. A seguir o programa deve determinar qual o valor que mais aparece neste vetor e quantas vezes ele aparece. Estas duas informações (qual é o mais repetido, e quantas vezes ele aparece) devem ser impressas.

```
 1: V [1..500] de inteiro
 2: MAT [1..500] [1..2] de inteiro
 3: I,J,ix : inteiro
 4: MAT[;1] ← -1 1.coluna inicializada com números negativos
 5: leia(V)
 6: para I de 1 até 500 faça
 7:    J ← 1
 8:    enquanto (J < 500) faça
 9:       se MAT[J,1] = V[I] então
10:          MAT[J,2] ← MAT[J,2] + 1
11:          abandone
12:       fimse
13:       se MAT[J,1] < 0 então
14:          MAT[J,1] ← V[I]
15:          MAT[J,2] ← 1
```

```
16:         abandone
17:       fimse
18:       J ← J + 1
19:     fimenquanto
20: fimpara
21: maior ← 0
22: I ← 1
23: enquanto (I< 500) faça
24:     se MAT[I,2] > maior então
25:       maior ← MAT[I,2]
26:       ix ← I
27:     fimse
28:     I ← I + 1
29: fimenquanto
30: escreva ("O maior e ",mat[ix,1])
31: escreva ("E a quantidade e ",mat[ix,2])
```

EXERCÍCIO 128 Uma fazenda resolveu automatizar o processo de tomada de decisão para escolha de sementes de trigo. Assim, a rea de plantio foi dividida em 8600 lotes distintos, que devidamente semeados, tiveram a cultura de 88/89 observada. A partir dos dados levantados no campo, foi criado um arquivo contendo o seguinte lay-out:

```
Identificação do lote, inteiro
Tipo de semente, inteiro
Área do lote plantado, em m2, real
Quantidade de trigo colhido no lote, em quilogramas, real
Qualidade do produto, inteiro
```

Existem 22 tipos de sementes, numerados de 1 a 22. A qualidade do produto é um número que significa: 1=excelente, 2=razoável, 3=inferior. Supõe-se que o custo de todos os tipos de semente é o mesmo.

Deve-se definir um algoritmo que leia o arquivo e determine e escreva qual o tipo de semente mais produtiva, seguindo o critério:

Se o trigo produzido é excelente, a produtividade do lote deve ser aumentada em 20%. Se o trigo é inferior, a produtividade do lote deve ser diminuída em 15%.

Para cada tipo de semente, devem ser totalizadas as informações de área e peso do produto produzido. A semente mais produtiva é aquela que tiver maior coeficiente peso / área.

EXERCÍCIO 129 Uma partida de rolamentos é composta de quantidades vari veis de rolamentos tipos 1, 2 e especial. Ela é aceita no TODO quando:

- pelo menos 97 % de rolamentos tipo 1 não tem defeito E
- pelo menos 95 % de rolamentos tipo 2 não tem defeito E
- Menos de 1000 rolamentos tipo especial tem defeito.

Os dados referentes a cada partida estão no computador na forma

```
NUMPAR, inteiro, TIP, caracter, QTDROL, inteiro, QTDDEF, inteiro
```

NUMPAR é um número que identifica a partida. No arquivo existem diversas partidas, porém todos os dados de uma partida estão agrupados. TIP, é apenas um caracter cujo conteudo pode ser: "1", "2"ou "E". QTDROL é a quantidade de rolamentos deste tipo que existem

na partilha. QTDDEF é a quantidade de rolamentos deste tipo defeituosos. Deve-se definir um algoritmo que leia todos os dados do arquivo, e para cada partida, deve-se imprimir seu número e a identificação "ACEITA"ou "REJEITADA"Um possível exemplo de arquivo poderia ser:

```
001,1,1200,10
001,E,3000,100
002,1,100,4
002,2,100,6
003,E,10000,800
004...
Neste caso, o programa deveria emitir o relatório:
001 ACEITA
002 REJEITADA
003 ACEITA
004 ...
```

EXERCÍCIO 130 Uma distribuidora de bebidas abastece o mercado com COCA-COLA. A empresa, no momento, está preparando o iminente lançamento da coca dietética. Assim, resolveu fazer uma pesquisa de opinião na cidade para determinar qual a melhor estratégia a seguir. Os dados da pesquisa estão em um arquivo cujo lay-out é:

```
Identificação do quarteirão da cidade, inteiro
Quantidade de crianças no quarteirão, inteiro
Quantidade de adolescentes no quarteirão, inteiro
Quantidade de pessoas obesas, no quarteirão, inteiro
Quantidade de diabéticos no quarteirão, inteiro.
Quantidade total de pessoas no quarteirão, inteiro
```

Deve-se criar um algoritmo que leia e processe os dados informando:

1. Quantos quarteirões tem mais de 100 obesos
2. Qual a percentagem média (=média das percentagens) de crianças por quarteirão
3. Quantos quarteirões tem mais adolescentes do que crianças
4. Quantos quarteirões tem mais de 5 % de diabéticos.

EXERCÍCIO 131 Uma revenda de automóveis quer fazer uma pesquisa para determinar se há correlação entre renda familiar e cor do automóvel adquirido. Para tanto, cada venda realizada no mês passado, gerou um registro magnético contendo:

```
Renda família, real (em R\$)
Cor do carro adquirido, cadeia
```

As cores possíveis são: amarelo, azul, branco, cinza, marrom, preto e verde. Já as classes de renda são 3: Classe 1, vai de 0 até 50 Salários mínimos. A classe 2, vai de 50,01 SM até 150 SM. A classe 3 é de quem ganha mais que 150 SM.

Para cada registro lido, deve-se ver em qual categoria ele cai (para tanto, o programa lerá no início, o valor de um salário mínimo) e incrementar 1 em uma variável que relacione a cor e a classe.

Ao final, o programa deve imprimir:

CLAS	AMARELO	AZUL	BRANCO	CINZA	MARROM	PRETO	VERDE
1	xxx	xx	xx	xx	xx	xx	xx
2	xxx	xx	xx	xx	xx	xx	xx
3	xxx	xx	xx	xx	xx	xx	xx

EXERCÍCIO 132 Um fabricante de margarina planeja homenagear a sua mãe no lançamento de um novo produto no ano que vem. A nova margarina deverá ter um nome formado por alguma combinação das letras M,E,R,T,A, que é o nome da mãe do fabricante. O exercício deve criar o algoritmo que faça e escreva as 120 combinações possíveis. Analisando o relatório, o fabricante escolherá o nome que quiser.

EXERCÍCIO 133 O instituto Nacional de Pesquisas Ufológicas estuda com interesse a ocorrência de contactos com seres extraterrestres em 5 países: Brasil, Argentina, Canadá, Estados Unidos e Portugal. Cada contacto é minuciosamente analisado e um resumo é cadastrado no computador. Os dados são:

```
Mês da ocorrência, inteiro
Ano da ocorrência, inteiro
País, cadeia
Grau do contacto, inteiro (1, 2 ou 3)
Confiabilidade, (1=provável embuste, 2=razoável,
                 3=confiável, 4=certo!)
```

A partir deste arquivo, deseja-se saber:

1. Qual o país com mais relatos ?
2. Qual o pais que proporcionalmente ao número de contactos apresenta mais embustes ?
3. Qual o Pais com mais contactos do terceiro grau, confi veis ou certos?
4. Qual o mês mais propício a ocorrências ?

EXERCÍCIO 134 Você tem a seguir 3 funções que buscam um determinado valor que pode ou não estar contido em um vetor, por hipótese, GLOBAL. Considere o nome do vetor como VET, e considere também que ele tem espaço de sobra no seu final. Assim, por exemplo, ele pode ter espaço para 1.000.000 de valores, mas atualmente usa bem menos do que isso.

Considere também a existência de uma variável GLOBAL chamada ULTIMOUSADO que contém o valor do último valor válido dentro de VET.

O algoritmo 1 é o seguinte:

```
1: inteiro função BUSCALIN (inteiro CHAVE)
2: inteiro I ← 1
3: inteiro RESPOSTA ← -1
4: enquanto I ≤ ULTIMOUSADO faça
5:    se CHAVE = VET[I] então
6:       RESPOSTA ← I
7:       I ← ULTIMOUSADO+1
8:    fimse
9:    I++
10: fimenquanto
11: retorne RESPOSTA
```

Para o algoritmo 2:

```
1: inteiro função BUSCALINSEN (inteiro CHAVE)
2: inteiro I ← 1
3: VET[ULTIMOUSADO+1] ← CHAVE {note que ULTIMO não é alterado}
4: enquanto CHAVE ≠ VET[I] faça
5:    I++
```



```
 6: fimenquanto
 7: se I ≠ ULTIMOUSALDO+1 então
 8:    retorne I
 9: senão
10:    retorne -1
11: fimse
```

Olhando os 2 algoritmos (que estão corretos) podem-se fazer algumas afirmações. Supondo que existem 800.000 valores em VET, analise as afirmações a seguir e informe ao final a soma das afirmações verdadeiras:

1 O algoritmo 1 é o mais rápido

2 O segundo nunca detecta a ausência da chave, pois logo de cara inclui o elemento a buscar

4 No segundo algoritmo, como a variável ULTIMOUSADO não é alterada a inclusão da chave buscada não ocorre

8 O segundo algoritmo é mais rápido

16 A eventual ordenação em ambos os casos deixa os algoritmos mais rápidos

Pede-se a soma das afirmações verdadeiras

EXERCÍCIO 135 No algoritmo a seguir, de uma função que detecta se o número X é ou não é primo, há quatro lacunas. Informe qual comando deveria ser colocado em cada uma das lacunas:

```
 1: lógico função PRIMO(inteiro X)
 2: inteiro QTD ← 0
 3: inteiro QUO ← 2
 4: enquanto QUO ≤ teto(raiz-quadrada(X)) ∧ QTD ≠ 0 faça
 5:    // teto(y) é o próximo inteiro maior ou igual a y
 6:    se X mod QUO = 0 então
 7:       _____.
 8:    fimse
 9:    _____.
10: fimenquanto
11: se QTD ≠ 0 então
12:    retorne _____.
13: senão
14:    retorne _____.
15: fimse
```

```
a) QTD++; QUO++; verdadeiro; falso
b) QUO++; QTD++; falso; verdadeiro
c) QTD++; QUO--; falso; verdadeiro
d) QTD++; QUO--; verdadeiro; falso
e) QTD++; QUO++; falso; verdadeiro
```

EXERCÍCIO 136 Suponha 2 pilhas, devidamente definidas para armazenarem números inteiros, de nomes A e B. Considere que elas tem capacidades grandes (infinitas, neste caso específico). Na situação inicial A pilha A contém os seguintes elementos: 1 (o primeiro a entrar), 5, 8, 14, 3, 2, 9, 6, 7 e 5 (o último a entrar) e B está vazia.

Após a execução do trecho correto de algoritmo a seguir, qual o conteúdo de B (lido do primeiro a entrar para o ultimo a sair)

```
 1: inteiro S ← 10
 2: enquanto (S ≤ 50) faça
 3:    Y ← desempilha (A)
 4:    S ← S + Y
 5:    empilha (B, Y)
 6: fimenquanto
 7: enquanto não vazia (B) faça
 8:    Y ← desempilha (B)
 9:    se Y mod 2 = 0 então
10:       empilha (A, Y)
11:    fimse
12: fimenquanto
```

```
a) 1, 5, 8, 14, 2, 6
b) 14, 2, 6
c) 5, 7, 6, 9, 2, 3, 14
d) 1, 5, 14, 2, 6
e) 1, 5, 8, 14, 3, 2, 9, 6, 7, 5
```

EXERCÍCIO 137 Definir algoritmo que leia uma cadeia de 200 caracteres e escreva a quantidade de letras maiúsculas que são seguidas por uma letra minúscula.

EXERCÍCIO 138 Definir algoritmo que leia um vetor de 208 inteiros e escreva os elementos que estão entre números maiores ou iguais.

Exemplo: Se o vetor lido foi `1 16 4 24 23 22 10 8 8 8 7 1 2 0 3` devem ser impressos os números `16, 24, 8, 8` e `2`.

EXERCÍCIO 139 Escreva algoritmo de uma função que leia um vetor de números de matrícula dos alunos aprovados em estrutura de dados (95 alunos) e que receba a matrícula de um aluno qualquer. A função deve devolver .V. se o aluno foi aprovado e .F. se foi reprovado.

EXERCÍCIO 140 Definir algoritmo que implemente uma variante da linguagem do "P", a seguir descrita. Toda sílaba formada só por uma consoante e uma vogal é considerada processável. A esta, na saida, deve ser agregada outra sílaba com a consoante "P"e a mesma vogal. O resto do texto não deve ser alterado. Definir algoritmo que leia uma frase de 150 caracteres, toda em maiúscula e escreva o resultado com os "P"s incluidos.

Exemplo:

```
O RATO VAI AO BURACO
O RAPATOPO VAPAI AO BUPURAPACOPO
```

EXERCÍCIO 141 Definir algoritmo que leia um vetor de 400 elementos inteiros e gere um segundo vetor de 200 elementos, onde cada elemento é a soma de dois elementos consecutivos do vetor de entrada.

EXERCÍCIO 142 Dado um vetor DIST com 15 distribuições de freqüencia estatísticas, criar um vetor ACUM, também de 15 elementos, contendo as distribuições acumuladas. O algoritmo deve ler DIST e imprimir DIST e ACUM.

EXERCÍCIO 143 Imagine 2 vetores de nomes A (com 50 elementos inteiros) e B (com 100 elementos inteiros). Defina um algoritmo que crie um terceiro vetor de nome C (com

150 elementos), sujeito a seguinte lei de formação:

```
C [1 3 5 ... 97 99]           ← A
C [2 4 6 ... 98 100]          ← B [1 2 3 ... 49 50]
C [101 102 103 ... 149 150] ← B [51 52 53 ... 99 100]
```

EXERCÍCIO 144 Definir algoritmo que leia um conjunto de 20 valores reais, armazene-os em um vetor e a seguir calcule a somatória expressa pela expressão:

$$S = (A_1 - A_{20})^2 + (A_2 - A_{19})^2 + ... + (A_{10} + A_{11})^2$$

EXERCÍCIO 145 Definir algoritmo que leia as notas de 120 alunos. Durante o processo de leitura devem ser calculadas a média e o desvio padrão. Posteriormente, devem-se imprimir as notas que estiverem fora do intervalo:

- limite inferior: Media - 3 $\times$ desvio padrão
- limite superior: Media + 3 $\times$ desvio padrão.

$$DP = \sqrt{\frac{\sum(X - X_m)^2}{n}}$$

Exemplo: Dadas 5 notas (8,9,2,5,6), calcular seu desvio padrão:

- Cálculo da média: 8 + 9 + 2 + 5 + 6 / 5 $\rightarrow$ 6 (média)
- pares: (8 - 6), (9 - 6), (2 - 6), (5 - 6), (6 - 6)
- Resultados: 2, 3, 4, 1, 0
- Ao quadrado: 4, 9, 16, 1, 0
- Somatório: 30
- Divido por 6: 5
- Raiz de 5: $\pm$2,2. Logo o D.P. é 2,2.

EXERCÍCIO 146 Existe um livro de 370 páginas que foi recentemente impresso. A editora deseja fazer um estudo sobre possíveis erros de impressão. Desta forma, um especialista foi convidado a levantar quantos erros existem em cada página. Cada valor foi digitado na forma de um vetor, formando um conjunto de 370 valores. Definir um algoritmo que leia este conjunto de dados, armazene-o na forma de vetor, e responda:

1. Qual a página que tem mais erros ? (Suponha que só existe uma, isto é ela é única)
2. Quantas páginas tem zero erros ?
3. Quantas páginas tem mais de 10 erros ?

EXERCÍCIO 147 Definir algoritmo que leia datas (sempre referentes a 1990), no formato DD,MM. Os dados devem terminar quando for lido um dia = 0. Ao lado de cada data, o algoritmo deve imprimir o número de dias transcorridos desde o dia 1 de janeiro de 1990.

EXERCÍCIO 148 Definir um algoritmo que leia valores sempre inferiores a 731. Para cada valor dia, o programa deve informar a qual dia de 89 ou de 90 ele se refere, supondo que o dia 1 corresponde a 1 jan 89 e o dia 730 a 31 dez 90. Dados terminam quando for lido um valor maior do que 730.

EXERCÍCIO 149 Definir algoritmo que leia 2 horas referentes sempre ao mesmo dia. A primeira hora indica quando um determinado processo produtivo começou, e a segunda hora indica quando ele terminou. Para cada par de horas, o programa deve calcular e imprimir qual a duração do processo em horas, minutos e segundos. Os dados de entrada estão no formato: HH,MM,SS. O dado de saída também está no mesmo formato.

EXERCÍCIO 150 Você precisa resolver um problema relativo a astronomia. Neste caso, pela magnitude dos números, o tipo real não é suficiente. O Objetivo do exercício é escrever um algoritmo capaz de implementar números inteiros com 40 casas de precisão, e com eles realizar operações de adição. Exemplo: Se o algoritmo ler os números:

```
 0000000200003000040000320000010001000000 e
 0000001010000600007000010010001000800002 a resposta ser :
0000000121000090001100004201000200090000 2
```

Deve-se atentar que a resposta pode ter 41 casas.

EXERCÍCIO 151 O ano letivo em uma pré-escola tem 125 dias de aula. Em cada dia, é escolhido o aluno de melhor comportamento e seu número de matrícula é colocado em um vetor de 125 elementos na posição correspondente ao dia específico (1. elemento do vetor corresponde ao 1.dia de aula, e assim por diante). Definir algoritmo que ao final do ano, quando o vetor estiver todo preenchido informe (escreva) qual o número do aluno que mais apareceu no vetor. O número de matrícula varia entre 1 e 2000 inclusive.

EXERCÍCIO 152 A Polícia Rodoviária vai usar uma nova abordagem para a repressão ao excesso de velocidade na estrada. Na saída de Curitiba, em direção a Paranaguá, um posto vai anotar e digitar no computador as placas dos carros que passarem. Na chegada às praias, outro posto vai fazer o mesmo trabalho. Desde que o computador tem relógio e sabe a que horas aconteceram os dois fatos, e conhecendo a distância entre os dois postos, que é de 92,6 Km, o programa irá multar os mais afobados.

O exercício pede que seja feita a rotina do programa que receberá os dois horários (no formato hh,mm ambos inteiros) e determine se o carro deve ou não ser multado. Lembrar que a velocidade máxima permitida é de 100 Km/h, com uma tolerância de 10 %.

8.8 Matriz

Pelas mesmas razões que nos levaram a criar o conceito de vetor, precisaremos agora tratar de um outro tipo de organização de dados: as matrizes. Trata-se de arranjos bi-dimensionais; de dados. Apenas para comparação, os vetores são arranjos unidimensionais de dados. Excepcionalmente, podemos ter mais de duas dimensões, permitindo algumas linguagens três ou mais. Algumas são mais privilegiadas neste aspecto: APL por exemplo, permite até 256 dimensões em algumas implementações e até 64 em outras.

Nos vetores, nós precisávamos um índice para referenciar um determinado elemento do vetor. No caso da matriz bi-dimensional, precisaremos 2 índices: o índice da linha e o da coluna. No caso de matrizes tridimensionais, são três os índices: plano, linha e coluna.

O formato é similar ao da definição de vetor, agora acrescentando-se apenas a quantidade de elementos na segunda dimensão. O formato é

<nome> vetor de [in..fi] [in..fi] de <tipo>

Por exemplo, a definição de uma matriz para conter os preços mensais (12 meses) de um produto ao longo dos últimos 5 anos, seria

```
1: PREÇO vetor de [1..5] [1..12] de real
```


Note-se que a rigor, qual é a primeira dimensão e qual a segunda é um critério do programador. No caso acima, a matriz terá 5 linhas por 12 colunas, mas nada impediria que ela fosse declarada como 12 linhas por 5 colunas.

Exemplos a) A criação de uma matriz 8 x 12 devidamente zerada, poderia ser assim:

```
1: M vetor de [1..8] [1..12] de real
2: inteiro I,J
3: para I de 1 até 8 faça
4:    para J de 1 até 12 faça
5:       M [I] [J] ← 0
6:    fimpara
7: fimpara
```

b) Dada uma matriz 7 x 13 devidamente preenchida, o algoritmo que encontra a transposta desta matriz é

```
1: M1 vetor de [1..7] [1..13] de real
2: M2 vetor de [1..13] [1..7] de real
3: inteiro I,J
4: leia(M1)
5: I ← 1
6: enquanto (I < 8) faça
7:    J ← 1
8:    enquanto (J < 14) faça
9:       M2 [J] [I] ← M1 [I] [J]
10:      J ← J + 1
11:   fimenquanto
12:   I ← I + 1
13: fimenquanto
```

EXERCÍCIO 153 Definir algoritmo que some duas matrizes A e B, de 7 linhas por 8 colunas cada uma, gerando a matriz C, também 7 x 8, onde cada elemento de C é a soma dos dois elementos correspondentes em A e B.

EXERCÍCIO 154 Definir algoritmo que some duas matrizes A e B, de 7 linhas por 8 colunas cada uma, gerando a matriz C, também 7 x 8, onde cada elemento de C é a soma dos dois elementos correspondentes em A e B, com uma restrição: Só se pode usar um índice. Dica: $L = f'(x)$; $C = f''(x)$. Como são f' e f'' ?

EXERCÍCIO 155 Definir algoritmo onde dada uma matriz M_1 de 6 x 9, devemos totalizá-la na vertical gerando um vetor de 9 elementos. Imprimir este vetor. A seguir deve-se encontrar e imprimir a média destes 9 valores.

EXERCÍCIO 156 Defina algoritmo capaz de ler uma matriz quadrada de ordem 23 e totalizar os elementos colocados abaixo da matriz principal (exclusive esta), imprimindo o total ao final.

EXERCÍCIO 157 Defina um algoritmo capaz de ler duas matrizes de dimensões 4 e 9 (a primeira) e 9 e 7 (a segunda). Após lidas as matrizes devem ser multiplicadas matricialmente gerando uma saída de dimensões 4 x 7. A matriz resultado deve ser impressa ao final.

EXERCÍCIO 158 Uma matriz esparsa é aquela que conta com grande número de elementos nulos dentro dela. Podemos arbitrar que uma matriz é esparsa quando apenas 5%

dos seus elementos não são nulos. Uma maneira razoável de armazenar matrizes esparsas é criar "matrizes comprimidas"que guardam a linha e a coluna dos valores que não são zero (guardam o valor também).

Isto posto:

- Escreva algoritmo que receba matriz esparsa e devolva matriz comprimida.
- O inverso
- Supondo matrizes de inteiros, indique quais tamanhos e percentuais de valores diferentes de zero recomendam o uso de matrizes comprimidas (despreze o tempo de processamento para conversão.)
- Proponha pelo menos outros 2 esquemas distintos deste, de economia de espaço para matrizes esparsas

EXERCÍCIO RESOLVIDO 16 Sejam duas matrizes a quem chamaremos A e B. A tem i linhas e j colunas, e B tem j linhas e k colunas. Deve-se escrever um algoritmo que tenha como resultado a variável C, de i linhas por k colunas, assim constituídos:

O elemento (i,k) de C será o menor valor das frações obtidas dividindo-se todos os pares onde o primeiro elemento está na linha i de A e o segundo elemento está na coluna k de B. Exemplo:

```
c[i,k] é o menor valor: a[i,1] / b[1,k] ou
                        a[i,2] / b[2,k] ou
                        a[i,3] / b[3,k] ou
                        ...
                        a[i,n] / b[n,k]
```

Escrever um algoritmo para o caso particular em que A tem 3 linhas e 4 colunas e B tem 4 linhas e 5 colunas. Com isto C terá 3 linhas e 5 colunas. O algoritmo deve ler A e B e imprimir C.

```
 1: algoritmo {exemplo7}
 2: MA vetor [1..3] [1..4] de real
 3: MB vetor [1..1] [1..5] de real
 4: MC vetor [1..3] [1..5] de real
 5: inteiro I J K
 6: real V1 [4]
 7: real menor
 8: I ← 1
 9: leia (MA,MB)
10: enquanto I < 3 faça
11:    J ← 1
12:    enquanto J < 5 faça
13:       K ← 1
14:       enquanto K < 4 faça
15:          V1[K] ← MA[I] [K] / MB[K] [J]
16:          K ← K + 1
17:       fimenquanto
18:       menor ← V1 [1]
19:       K ← 2
20:       enquanto K < 4 faça
21:          se V1 [K] < menor então
22:             menor ← V1 [K]
```



```
23:         fimse
24:         K ← K + 1
25:       fimenquanto
26:       C [I] [J] ← menor
27:       J ← J + 1
28:     fimenquanto
29:     I ← I + 1
30: fimenquanto
31: escreva(C)
32: fim{algoritmo}
```

EXERCÍCIO RESOLVIDO 17 Dada uma matriz numérica contendo 30 linhas (uma para cada aluno) e 5 colunas (1-matricula, 2-nota prova 1, 3-nota prova 2, 4-nota trabalho, 5-media), deve-se definir um programa que:

- leia a matriz (ela virá classificada em ordem ascendente de matrícula)
- Reclassifíque-a em ordem descendente de média
- escreva o resultado

```
 1: algoritmo {exemplo8}
 2: MAT vetor [1..30] [1..5] de real
 3: MATS vetor [1..30] [1..5] de real
 4: V vetor [1..30] de real
 5: V ← (1,2,3,4,5,6,... 28,29,30)
 6: inteiro I J K
 7: leia(mat)
 8: i ← 1
 9: limite ← 30
10: enquanto limite > 1 faça
11:   I ← 1
12:   enquanto I < limite faça
13:     se MAT[I] [5] < MAT[I+1] [5] então
14:       AUX ← MAT[I] [5]
15:       MAT[I] [5] ← MAT[I+1] [5]
16:       MAT[I+1] [5] ← AUX
17:       AUX ← V[I]
18:       V[I] ← V[I+1]
19:       V[I+1] ← AUX
20:     fimse
21:     I ← I + 1
22:   fimenquanto
23:   LIMITE ← LIMITE - 1
24: fimenquanto
25: para I de 1 ate 30 faça
26:   MATS [I][1] ← MAT[V[I]][1]
27:   MATS [I][2] ← MAT[V[I]][2]
28:   MATS [I][3] ← MAT[V[I]][3]
29:   MATS [I][4] ← MAT[V[I]][4]
30:   MATS [I][5] ← MAT[V[I]][5]
31: fimpara
32: escreva(mats)
33: fim{algoritmo}
```

EXERCÍCIO 159 Definir um algoritmo que leia uma matriz 9 x 9, some-a com a sua transposta e escreva o resultado.

EXERCÍCIO 160 Definir algoritmo que leia duas matrizes (A e B) de dimensões 7 x 13 e gere uma terceira matriz, onde cada elemento é o maior entre os elementos correspondentes em A e B.

EXERCÍCIO 161 Definir algoritmo que leia duas matrizes A e B, de 9 x 16 elementos numéricos, devendo gerar uma terceira matriz sujeita as seguintes regras:

- Se $A[L,C] + B[L,C] > 100$ então $C[L,C] \leftarrow A[L,C]$ e
- se $A[L,C] + B[L,C] \leq 100$ então $C[L,C] \leftarrow B[L,C]$

EXERCÍCIO 162 Definir um algoritmo que leia uma matriz 5 x 7, inverta suas linhas e a seguir, escreva-a. Exemplo: se a matriz lida for (3 x 4):

```
1  4  3  2
1  1  2  1
0  0  3  3

a resposta dever  ser:
0  0  3  3
1  1  2  1
1  4  3  2
```

EXERCÍCIO 163 Definir um algoritmo que leia uma matriz 5 x 8, inverta suas colunas de ordem par e a seguir, escreva-a. Exemplo: se a matriz lida for (3 x 8):

```
1  4  3  2  1  2  3  4
1  1  2  1  0  0  7  8
0  0  3  3  2  3  7  9

a resposta dever  ser:
1  4  3  2  1  2  3  4
1  8  2  0  0  1  7  1
0  9  3  3  2  3  7  0
```

EXERCÍCIO 164 Definir algoritmo que efetue a multiplicação matricial de duas matrizes, que deverão ser lidas de uma só vez: A primeira matriz tem forma 8 linhas por 9 colunas, e a segunda tem 9 linhas por 3 colunas. O resultado, depois de calculado, deve ser impresso.

EXERCÍCIO 165 Escrever um algoritmo que leia (de uma só vez) uma matriz de 23 linhas por 10 colunas. Esta matriz representa a população dos 10 maiores municípios de cada estado Brasileiro. Em cada linhas, as colunas estão em ordem decrescente, com exceção da primeira coluna, que sempre representa a capital. O algoritmo deve imprimir:

1. Qual o número do estado de capital mais populosa ?
2. Qual a média das populações das capitais do Brasil ?
3. Quais os números (de estado e de município) dos municípios que tem população maior que a capital ?

EXERCÍCIO 166 Uma matriz representa os valores da cotação da soja no mercado de Chicago em dólares por tonelada, de hora em hora (das 09:00 as 18:00 = 10 linhas), e nos 22 dias úteis de julho/89 (22 colunas). Deseja-se um algoritmo que informe:

1. Quais as maiores cotações dia a dia (22 valores)
2. Qual a hora em que foi mais freqüente aparecer a maior cotação (Neste caso não haverá empate, por definição).
3. Em quais dias a média da manhã (primeiros 5 valores) foi maior do que a média da tarde (5 últimos valores).

EXERCÍCIO 167 Dada uma matriz de dimensões 8 x 15, obter uma outra também de dimensões 8 x 15, onde cada linha foi dividida pelo menor elemento da linha correspondente na matriz original.

EXERCÍCIO 168 Dada uma matriz de dimensões 9 x 12, obter uma outra também de dimensões 9 x 12, onde cada linha foi dividida pela somatória dos elementos da linha correspondente na matriz original.

EXERCÍCIO 169 Dada uma matriz de 7 linhas por 12 colunas, definir algoritmo que a leia, e a seguir classifique as linhas da matriz pelos valores indicados na coluna 4. A matriz resultado deve ser impressão ao final.

EXERCÍCIO 170 Dada uma matriz de 8 linhas por 6 colunas, deve-se criar um algoritmo capaz de lê-la de uma só vez, e a seguir gerar uma matriz de idêntica forma (8x6) onde cada linha é o produto acumulado da linha da matriz original. Exemplo:

```
Se a matriz lida for:   1  3  5  0 o resultado será  1  3 15  0
                        4  2  0  2                  4  8  0  0
                        1  2  3  4                  1  2  6 24
                        2  2  2  2                  2  4  8 16
```

Porquê tudo começa com 1 ?

A palavra digital vem do latim digitus que significa dedo, e por que temos 10 dedos, toda nossa lógica aritmética se criou sobre um sistema decimal (de 10).

Já que são 10 dígitos e qualquer número é composto por uma combinação desses dígitos seria de se esperar que cada um dos 10 dígitos tivesse uma distribuição proporcional quando vai se representar um número qualquer. Assim, teoricamente, se analisarmos um grande conjunto de números, seria de se esperar que os dígitos 1, 2, 3, ... aparecessem em 10% das vezes, cada um iniciando os números.

Afinal, não existem dígitos mais bonitos ou mais simpáticos para que apareçam no começo dos números mais do que os outros. Ou será que existem ?

Em 1938, um matemático chamado Benford, acabou descobrindo que sim, existem dígitos iniciais mais frequentes do que outros. Ele estudou um monte de distribuições e chegou à conclusão que o dígito 1 ocorre no começo em cerca de 30% das vezes, independente da fonte ou do fenômeno que é consultado.

Parece estranho, mas é verdade: em qualquer tabela, uma grande quantidade de números começa com o dígito 1. Muito mais do que os demais dígitos.

Veja-se a seguinte conjunto de dados, extraído do livro de Benford. Veja na tabela 8.1.

Parece que a distribuição do primeiro dígito em números segue a seguinte distribuição logarítmica:

$$P(n) \approx \log(n+1) - \log n$$

para $n = 1, 2, ...9$. Este é a "Lei do Primeiro Dígito".

Tabela 8.1: Distribuição dos primeiros dígitos

Col.	Título do assunto	1	2	3	4	5	6	7	8	9	Números pesquisados
A	Populações	33.9	20.4	14.2	8.1	7.2	6.2	4.1	3.7	2.2	3259
B	Constantes	41.3	14.4	4.8	8.6	10.6	5.8	1.0	2.9	10.6	104
C	Exemplares aleatórios de jornais	30.0	18.0	12.0	10.0	8.0	6.0	6.0	5.0	5.0	100
D	Calores específicos de substâncias	24.0	18.4	16.2	14.6	10.6	4.1	3.2	4.8	4.1	1389
E	Peso molecular	26.7	25.2	15.4	10.8	6.7	5.1	4.1	2.8	3.2	1800
F	Drainage	27.1	23.9	13.8	12.6	8.2	5.0	5.0	2.5	1.9	159
G	Peso atômico	47.2	18.7	5.5	4.4	6.6	4.4	3.3	4.4	5.5	91
H	n^{-1} ou $\sqrt{n}$	25.7	20.3	9.7	6.8	6.6	6.8	7.2	8.0	8.9	5000
I	exemplares do Reader's Digest	33.4	18.5	12.4	7.5	7.1	6.5	5.5	4.9	4.2	308
J	Voltagem de raios X	27.9	17.5	14.4	9.0	8.1	7.4	5.1	5.8	4.8	707
K	Dados da liga americana de baseball	32.7	17.6	12.6	9.8	7.4	6.4	4.9	5.6	3.0	1458
L	Endereços aleatórios	28.9	19.2	12.6	8.8	8.5	6.4	5.6	5.0	5.0	342
M	$n^1, n^2, ...n!$	25.3	16.0	12.0	10.0	8.5	8.8	6.8	7.1	5.5	900
N	Taxas de Mortalidade	27.0	18.6	15.7	9.4	6.7	6.5	7.2	4.8	4.1	418
	Média	30.6	18.5	12.4	9.4	8.0	6.4	5.1	4.9	4.7	1011

Capítulo 9

Registros

Este tipo indica uma estrutura complexa, formada por vários tipos primitivos, e que passam a ser usados conjuntamente. Ela é criada em duas etapas. Na primeira, define-se do que é composta a estrutura, e na segunda, quais as variáveis que serão mapeadas por essa estrutura.

```
Definição da estrutura:
   estrutura <nome da estrutura>      // exemplo: estrutura ENDEREÇO
      <tipo> <nome campo 1>           //   alfanum RUA [30]
      <tipo> <nome campo 2>           //   inteiro NÚMERO
      ...                             //   inteiro CEP
      <tipo> <nome campo n>           //   alfanum CIDADE [20]
   fim {estrutura}                    // fim {estrutura}
```

Utilização da estrutura em variáveis:

```
<nome da estrutura> <nome da variável>   // exemplo: ENDEREÇO CLIENTE
```

Obviamente, depois que um tipo estrutura foi definido, ele pode ser usado em qualquer local da linguagem onde a indicação <tipo> apareça, por exemplo, em vetores.

9.1 Definição de registros

registro É uma coleção HETEROGÊNEA de coisas. As coisas são reconhecidas pelo seu nome.

Por exemplo, uma placa de automovel é um conjunto de 3 letras e 4 números. Não tem como definir estra estrutura usando vetores.

estrutura PLACA
caracter LETRA[3]
inteiro NUMERO
fim {estrutura}
PLACA VEICULO
VEICULO.LETRA ← “ABC”
VEICULO.NUMERO ← 1234
leia VEICULO
leia (VEICULO.LETRA, VEICULO.NUMERO)

Não confunda PLACA (é apenas uma estrutura, não ocupa lugar na memória e NÃO pode ser referenciada em comandos) com VEICULO (a variável cuja estrutura é PLACA, ocupoa lugar, pode ser referenciada...

Deve-se lembrar que uma matriz também pode ser definida com um vetor de vetores (ou um registro). Acompanhe:

estrutura LINHAS
inteiro COLUNAS[10]
fim { estrutura }
LINHAS MAT[20]

Note que neste caso, existe uma matriz MAT contendo 20 linhas por 10 colunas. O acesso agora se dá

MAT[3].COLUNAS[4] ← ...

ao invés de

MAT[3] [4] ← ...

que seria o normal de uma matriz.

vetor de registros Nada impede que um registro seja vetorizado. Acompanhe.

estrutura DEBITO
inteiro RG
pontoflutuante PRESTACAO
fim {estrutura}
DEBITO CLIENTES[100]

Neste caso, existe um vetor de 100 (registros) débitos. Para acessar a prestação do 10^o cliente, faríamos:

CLIENTES[10].PRESTACAO

registro de vetores Também nada impede que um registro tenha vetores

estrutura ECONOMETRIA
pontoflutuante INFLACAO[12]
inteiro POPULACAO
inteirolongo PIB
fim { estrutura }
ECONOMETRIA BRASIL, ARGENTINA, PARAGUAI

Neste caso, como acessar as 2 últimas taxas de inflação de BRASIL e de PARAGUAI ?

BRASIL.INFLACAO[12], BRASIL.INFLACAO[11] e
PARAGUAI.INFLACAO[12], PARAGUAI.INFLACAO[11]

EXERCÍCIO RESOLVIDO 18 Uma organização tem um conjunto de registros de vendas do mês de janeiro de 1988. Tais registros tem o formato:

```
REG: DIA,inteiro;
VALOR,real.
```

Criar um algoritmo capaz de ler os dados e imprimir os totais diários. O processamento termina quando for lido um dia igual a 00.

```
1: tipo R = registro
2: inteiro: dia
3: real:valor
4: fimregistro
5: R: reg
6: tipo V = vetor[1:31] real
7: V: acum
8: acum ← 0
9: leia(reg)
```

```
10: enquanto dia ≠ 0 faça
11:    acum [dia] ← acum [dia] + reg.valor
12:    leia(reg)
13: fimenquanto
14: escreva(acum)
```

EXERCÍCIO RESOLVIDO 19 Uma organização tem um conjunto de registros de vendas do mês de janeiro de 1988. Tais registros tem o formato:

```
REG: DIA,inteiro;
VALOR,real.
```

Criar um algoritmo capaz de ler os dados e imprimir os totais diários. O processamento termina quando for lido um dia igual a 00.

Resolver o problema, sujeito às restrições:

- Não se pode usar vetor
- Os dados já vêem classificados por ordem de dia.

```
 1: tipo R = registro
 2: inteiro: dia
 3: real:valor
 4: fimregistro
 5: R: reg
 6: inteiro: diaant
 7: real: total
 8: total ← 0
 9: leia(reg)
10: ant ← dia
11: enquanto dia ≠ 0 faça
12:    enquanto dia = diaant faça
13:       total ← total + valor
14:       leia(reg)
15:    fimenquanto
16:    escreva (total)
17:    total ← valor
18:    diaant ← dia
19:    se dia ≠ 0 então
20:       leia(reg)
21:    fimse
22: fimenquanto
```

EXERCÍCIO RESOLVIDO 20 Uma empresa tem registros de vendas de todo o ano de 1987, na forma (dia,mês,valor). Dia e mês são inteiros, e valor é real. Criar um algoritmo que leia todo o arquivo e escreva os totais diários. Não é necess rio definir registro.

```
1: inteiro: dia,mes
2: real: valor
3: tipo M = matriz [1:12, 1:31] real
4: M: cad
5: cad ← 0
6: leia (dia,mes,valor)
7: enquanto dia ≠ 0 faça
```



```
 8:    cad [mes,dia] ← cad [mes,dia] + valor
 9:    leia (dia,mes,valor)
10: fimenquanto
11: escreva(cad)
```

EXERCÍCIO 171 A Empresa DEUNOPÉ vai distribuir R$ 100.000 de prêmio de fim de ano a seus vendedores. Cada vendedor receber um fixo de 1.000. A diferença, isto é, a quantia que sobrar, será distribuída proporcionalmente ao total de vendas de cada um. A Empresa tem no máximo 50 vendedores , e seus dados estão em um arquivo que contém o lay-out

```
Nome do vendedor, cadeia [30]
total de vendas, real
```

Para resolver este algoritmo, sem usar vetores, deve-se ler o arquivo duas vezes.

EXERCÍCIO RESOLVIDO 21 Existe uma turma do curso de PD que teve os dados referentes ao bimestre transcritos em cartões na forma:

```
MATR,inteiro
NT1,NT2:real
NOTPROV,real
(NT=nota trab)
```

Os dados terminam quando for lida uma matrícula igual a zero. Escreva um algoritmo que defina o registro acima, leia o arquivo, calcule a média para cada aluno, usando a fórmula:

$$M = ((T_1 \times 2) + T_2 + (N_{prova} \times 7)) \div 10$$

Após cada cálculo, deve-se imprimir, o número da matrícula, a média final e a mensagem: "PARABÉNS", caso a média tenha sido maior ou igual a 7. Se a média foi menor que 7, a mensagem é "PRECISAMOS ESTUDAR MAIS...".

```
tipo C = registro
inteiro: matric
real: t1, t2, pr
fimregistro
C: cad
real: media
caracter: mensagem
leia (cad)
enquanto matric ≠ 0 faça
  media ← (t1 * 2) + t2 + (pr * 7)
  media ← media / 10
  se media ≥ 7 então
    mensagem ← "PARABÉNS"
  senão
    mensagem ← "PRECISAMOS ESTUDAR MAIS..."
  fimse
  escreva (matric, media, mensagem)
  leia (cad)
fimenquanto
```

EXERCÍCIO RESOLVIDO 22 Existe um arquivo que contém alunos que deverão se dirigir a unidades da universidade para fazer vestibular. Tais dados estão em um cadastro que tem o formato:

```
MATRE,inteiro
Dist-unid-1,real
DU2,real
DU3,real
```

Definir um algoritmo que leia tais registros e para cada um, gere um outro registro no formato

```
MATRS,inteiro
Num-uni-mais-prox,inteiro
distancia,real
```

Os dados acabam quando for lida uma matrícula zero.

```
1: tipo E = registro
2: inteiro: ME;
3: real: U1, U2, U3;
4: fimregistro
5: E: CADE;
6: tipo S = registro
7: inteiro: MS, UP;
8: real: DU;
9: fimregistro
10: S: CADS;
11: leia (CADE);
12: enquanto ME ≠ 0 faça
13:    se U1 ≤ U2 ∧ U1 ≤ U3 então
14:       UP ← 1;
15:       DU ← U1;
16:    fimse
17:    se U2 ≤ U1 ∧ U2 ≤ U3 então
18:       UP ← 2;
19:       DU ← U2;
20:    fimse
21:    se U3 ≤ U1 ∧ U3 ≤ U2 então
22:       UP ← 3;
23:       DU ← U3;
24:    fimse
25:    MS ← ME;
26:    grave (CADS);
27:    leia (CADE);
28: fimenquanto
```

EXERCÍCIO 172 Suponha as seguintes estruturas

```
1: estrutura X
2: inteiro A
3: real B
4: inteiro C[5]
5: fim estrutura
6: estrutura Y
7: inteiro D
8: X E[3]
9: fim estrutura
10: Y F
```


Pede-se o desenho da área F, com todos os seus componentes

EXERCÍCIO 173 Projete a estrutura necessária para comportar todas as informações dos alunos do UNICENP. Veja que cada aluno tem NOME (com tamanho máximo de 40 caracteres), idade e código de matrícula. Os alunos se agrupam em turmas de no máximo 70 alunos. Cada turno tem 4 turmas e cada curso tem 2 turnos. Finalmente, há no UNICENP 28 cursos. Para a estrutura acima, responda

- Quantos alunos cabem no total ?
- Supondo que alunos não preenchidos contém idade = 0, qual o algoritmo que conta quantos alunos há no curso 22 ?
- Qual o algoritmo que conta quantos alunos estudam pela manhã (1. turno) ?
- Qual o algoritmo que imprime o nome de todos os alunos do curso de informática (curso=8)

EXERCÍCIO 174 Suponha a estrutura necessária para conter um romance, sujeito as seguintes limites: linhas de 60 caracteres; páginas de 43 linhas; capítulos de 20 páginas; livros de 18 capítulos.

- Defina a estrutura em questão
- Escreva o algoritmo que imprime as capitulares (primeiro caractere de cada capítulo).
- Escreva o algoritmo que imprime o capítulo 2
- Escreva o algoritmo que imprime as páginas ímpares dos capítulos pares.

EXERCÍCIO 175 Suponha que para identificar uma origem ou um destino, há que se ter 3 informações: nome da cidade, sigla do estado a que pertence e país. Isto posto, escreva a estrutura necessária para identificar uma viagem. Uma viagem é composta pelas informações: origem, destino, data da viagem, duração em dias.

EXERCÍCIO RESOLVIDO 23 Na realização da última feira de moda no Parque Barigüi, estiveram perto de 100.000 visitantes. Cada família informou seu estado de origem e quantos componentes tinha. Tais dados se encontram em um arquivo com o formato:

```
ESTADO, cadeia[2];
NUMVIS, inteiro.  ESTADO é a sigla com 2 caracteres.
```

Criar um algoritmo que leia tal arquivo, exclua os visitantes do estado do Paraná, (sigla PR), e totalize por estado de origem, em ordem alfabética. Os dados terminam quando for lido um estado igual a "XX", que não deve ser considerado para efeito de cálculo.

```
1: tipo R = registro
2: estado : caracter
3: qtd : inteiro
4: fimregistro
5: TAB [1:24] vetor de R
6: reg : R
7: tab[1].estado ← "AC"
8: tab[2].estado ← "AL"
9: tab[3].estado ← "AM"
10: tab[4].estado ← "BA"
```

```
11: tab[5].estado ← "CE"
12: tab[6].estado ← "DF"
13: tab[7].estado ← "ES"
14: tab[8].estado ← "GO"
15: tab[9].estado ← "MA"
16: tab[10].estado ← "MG"
17: tab[11].estado ← "MS"
18: tab[12].estado ← "MT"
19: tab[13].estado ← "PA"
20: tab[14].estado ← "PB"
21: tab[15].estado ← "PE"
22: tab[16].estado ← "PI"
23: tab[17].estado ← "RJ"
24: tab[18].estado ← "RN"
25: tab[19].estado ← "RO"
26: tab[20].estado ← "RS"
27: tab[21].estado ← "SC"
28: tab[22].estado ← "SE"
29: tab[23].estado ← "SP"
30: leia (reg)
31: enquanto (reg.estado ≠ "XX" faça
32:    se reg.estado ≠ "PR" então
33:       I ← 1
34:       enquanto tab[I].estado ≠ reg.estado  faça
35:          I ← I + 1
36:       fimenquanto
37:       tab[i].qtd ← tab[i].qtd + reg.qtd
38:    fimse
39:    leia (reg)
40: fimenquanto
41: escreva (tab)
```

Outra possibilidade se solução seria não colocar os estados na tabela, e a medida que eles fossem chegando, a tabela fosse sendo incrementada. Neste caso, se a saída tivesse que ser ordenada este trabalho teria que ser feito a parte ao final dos dados. A solução ficaria:

```
 1: tipo R = registro
 2: estado : caracter
 3: qtd : inteiro
 4: fimregistro
 5: TAB [1:24] vetor de R
 6: reg : R
 7: tab ← ... {espaços e zeros}
 8: leia (reg);
 9: enquanto (reg.estado ≠ "XX" faça
10:    se reg.estado ≠ "PR" então
11:       I ← 1
12:       enquanto tab[I].estado ≠ reg.estado ∧ TAB[I].ESTADO ≠  faça
13:          I ← I + 1
14:       fimenquanto
15:       se  TAB[I].ESTADO =  então
16:          TAB[I].ESTADO ← REG.ESTADO
17:       fimse
```



```
18:       tab[i].qtd ← tab[i].qtd + reg.qtd
19:     fimse
20:     leia (reg)
21: fimenquanto
22: escreva (tab)
```

EXERCÍCIO RESOLVIDO 24 Uma companhia aérea tem 6 tipos de aviões: 737-200, 737-300, 727, 707, A300 e Bandeirantes. Do cadastro de viagens realizadas no último ano, obteve-se o seguinte arquivo:

```
TIPO,cadeia[6];
PASSAG,inteiro;
DIST,real,
```

onde TIPO é um campo de 6 posições com a identificação de tipo de avião, PASSAG é a quantidade de pessoas transportadas e DIST é a distância do trecho percorrido. Cada v"o que a Companhia realizou no ano tem um correspondente registro neste arquivo.

Deve-se escrever um algoritmo que leia o arquivo e crie uma tabela de 6 ocorrências contendo as quantidades totais de passageiros e distâncias por tipo de avião. O objetivo final é conhecer a produtividade de cada tipo de avião, dada pela razão: passageiros / distância. O programa deve imprimir o nome do tipo de todos os aviões, e ao lado sua produtividade. Os dados terminam quando for lido um tipo totalmente em branco.

```
 1: tipo R = registro
 2: aviao : caracter
 3: passag : inteiro
 4: distancia : real
 5: fimregistro
 6: TAB [1..6] vetor de R
 7: tipo R1 = registro
 8: av : caracter
 9: pa : inteiro
10: di : real
11: fimregistro
12: REG : R1
13: inteiro: I;
14: real: prod;
15: leia (reg)
16: enquanto (av ≠ ) faça
17:   I ← 1;
18:   enquanto  (I < 7) faça
19:     se aviao [I] = av então
20:       abandone
21:     fimse
22:     se aviao [I] =  então
23:       aviao [I] ← av;
24:       passag [I] ← pa;
25:       distancia [I] ← di;
26:       I ← 7;
27:       abandone
28:     fimse
29:   fimenquanto
30:   se I < 7 então
```

```
31:       passag [I] ← passag [I] + pa;
32:       distancia [I] ← distancia[I] + di;
33:     fimse
34:     leia(reg);
35: fimenquanto
36: I ← 1;
37: enquanto I < 7 faça
38:     prod ← passag [I] ÷ distancia [I]
39:     escreva ("Tipo ", aviao [I], "produtividade", prod);
40:     I ← I + 1;
41: fimenquanto
```

EXERCÍCIO RESOLVIDO 25 Exemplo: multiplicação matricial

```
 1: I ← 0
 2: J ← 0
 3: K ← 0
 4: A [1..3] [1..4] vetor de real
 5: B [1..4] [1..5] vetor de real
 6: R [1..3] [1..5] vetor de real
 7: A[1;1] ← 1
 8: A[1;2] ← 2
 9: A[1;3] ← 3
10: A[1;4] ← 4
11: A[2;1] ← 5
12: A[2;2] ← 6
13: A[2;3] ← 7
14: A[2;4] ← 8
15: A[3;1] ← 9
16: A[3;2] ← 10
17: A[3;3] ← 11
18: A[3;4] ← 12
19: B[1;1] ← 1
20: B[1;2] ← 2
21: B[1;3] ← 3
22: B[1;4] ← 4
23: B[1;5] ← 5
24: B[2;1] ← 6
25: B[2;2] ← 7
26: B[2;3] ← 8
27: B[2;4] ← 9
28: B[2;5] ← 10
29: B[3;1] ← 11
30: B[3;2] ← 12
31: B[3;3] ← 13
32: B[3;4] ← 14
33: B[3;5] ← 15
34: B[4;1] ← 16
35: B[4;2] ← 17
36: B[4;3] ← 18
37: B[4;4] ← 19
38: B[4;5] ← 20
39: para I de 1 ate 3  faça
```



```
40:    para J de 1 ate 5  faça
41:      S ← 0
42:      para K de 1 ate 4  faça
43:        S ← S + A[I;K] × B[K;J]
44:      fimpara
45:      R[I;J] ← S
46:    fimpara
47: fimpara
48: escreva R
```

EXERCÍCIO RESOLVIDO 26 Exemplo: Desvio Padrão Seja o desvio padrão

$$\sigma = \sqrt{\frac{\sum (X - \overline{X})^2}{n}}$$

que é calculado pelo seguinte algoritmo

```
 1: algoritmo dp
 2: I ← 0
 3: V ← [1..10] vetor de real ← 0
 4: M ← 0
 5: para I de 1 ate 10  faça
 6:   V[I] ← I vezes 2
 7: fimpara
 8: para I de 1 ate 10  faça
 9:   M ← M + V[I]
10: fimpara
11: M ← M ÷ 7
12: R ← 0
13: para I de 1 ate 10  faça
14:   R ← R + (V[I] − M)^2
15: fimpara
16: R ← R ÷ 7
17: R ← √R
18: escreva R
```

9.2 Processamento de Textos

Um capítulo importantíssimo do processamento de dados se refere ao processamento de texto. Trata-se de transformar o computador em uma super máquina de escrever capaz de realizar todas as tarefas triviais do manuseio de textos e também algumas outras facilidades desejadas. A importância deste assunto, em um curso de lógica de construção de algoritmos está em que para manusear texto, desenvolvem-se inúmeras técnicas muito úteis para a interpretação e manuseio de dados na forma alfabética, que não necessariamente precisam ser textos.

Um texto, nesta acepção é um conjunto indeterminado de caracteres. Pode ser uma variável cadeia (limitada a 255), ou um vetor de caracteres, neste caso, sem limite. Alguns caracteres incluídos no meio do texto podem formatá-lo de maneira a mudar seu "visual"quando o imprimirmos. Um exemplo deste tipo de caracter são o "RETORNO DE CARRO"e o "MUDANÇA DE LINHA". Quando colocados juntos no meio de um texto, eles causam a mudança de linha e o retorno ao início da próxima linha.

Os caracteres ASCII referentes: retorno de carro: CR = carriage return = Código 13 do ASCII LF = line feed = Código 10 do ASCII

Em outros textos, cada linha tem no seu início, uma vari vel numérica que indica qual o comprimento da linha, e neste caso não há necessidade de usarem-se delimitadores.

Cadeia e vetor de caracteres

Neste ponto, podemos ver com mais clareza, o conceito de cadeia de caracteres. Trata-se de um vetor de caracteres. Falando em termos rigorosos, a definição:

caracter VET[30] equivale a uma cadeia VET[30]

A explicação para a existência desta particularização do conceito de vetor, é que o tipo primitivo (no caso, o caracter, que tem sempre comprimento igual a 1), é muito pouco adequado, pois praticamente todas as informações alfanuméricas exigem mais de um caracter para serem úteis.

Entretanto, existem pelo menos três diferenças entre um vetor de caracteres e uma cadeia.

A cadeia está limitada a 255 caracteres de comprimento, e o vetor não tem esta limitação.

Em pascal, a cadeia pode ser lida e gravada de uma vez, ao contr rio dos vetores.

Entretanto, tanto a cadeia quanto o vetor podem ser percorridos através de índices.

OBSERVAÇÃO IMPORTANTE: Ao se fazer o ORD de um elemento indexado, a resposta será a ordem do elemento indexado no universo considerado e NÃO o índice do elemento dentro do vetor. Exemplo seja o vetor V ← 'ABCDEFGHI' Se fizermos ord(V[3]) a resposta é ord('C') que é 67, e não 3 como poderia parecer aos mais incautos.

EXERCÍCIO 176 Escreva um algoritmo que leia uma frase terminada por um ponto, com tamanho inferior a 80 caracteres, e um único espaço em branco separando cada palavra, e escreva o número de palavras da frase.
DESAFIO: a quantidade de palavras femininas (i. é: terminadas em "A").

EXERCÍCIO 177 Definir algoritmo capaz de receber uma frase e converter as letras minúsculas lidas em maiúsculas, imprimindo este resultado.

Exemplo: Se for lida a frase "Ivo viu a LARANJA", a resposta será "IVO VIU A LARANJA".

EXERCÍCIO 178 Definir um algoritmo que leia uma frase de até 80 caracteres, alinhada à esquerda e determine e escreva:

1. Qual o número da maior palavra da frase
2. Quantos caracteres ela tem de comprimento.

Exemplo: Se for lida a frase "Maria comeu a melancia", o algoritmo deverá imprimir 4,8.

EXERCÍCIO 179 Escrever um algoritmo para criptografar textos, usando o algoritmo da "Criptografia de Cezar", para k=3.

EXERCÍCIO 180 Justificação: Dada uma linha com um comprimento cl, e supondo uma frase com comprimento cf, e supondo mais, que cl seja maior do que cf, surge o problema de estabelecer a frase na linha ALINHADA pela esquerda e pela direita. Chama-se a isto de justificação, e os bons datilógrafos fazem isto quase instintivamente. Por exemplo,. se tivermos a frase "IVO.VIU.A.UVA"(. é branco), que tem um cf de 13, para ser impressa em uma linha de 20 (cl=20), alinhada a esquerda e a direita, como fazer isto ?

Um possível ataque para o problema é estabelecer:

nb = número de brancos (nb = cl - cf). No caso: 20 - 13 = 7

np = número de palavras. No caso np = 4

ne = numero de locais receptores de brancos. ne = pred (np). No caso ne=3
b1 = tamanho do primeiro preenchimento. b1 = nb div ne.
No caso b1 = 7 div 3 = 2
b2 = tamanho do segundo preenchimento. b2 = succ(b1).
No caso b2 = 3
n2 = quantidade de b2's. n2 = nb mod ne.
No caso: n2 = 7 mod 3 = 1
n1 = quantidade de b1's. n1 = ne - n2.
No caso: n1 = 3 - 1 = 2
Em resumo, para justificar "IVO.VIU.A.UVA"em 20 posições, devemos inserir 1 (n2) bloco de 3 (b2) espaços, e 2 (n1) blocos de 2 (b1) espaços no texto. O local de inserção fica a critério do usuário, mas pode-se sugerir a seguinte colocação: (. significa espaço em branco)
IVO...VIU....A...UVA.

EXERCÍCIO 181 Imagine um algoritmo capaz de ler uma entrada formada por até 60 caracteres, que correspondem a uma frase. Esta frase termina quando for encontrado um ponto. O algoritmo deve ser capaz de responder:

- Qual o tamanho da frase
- Quantas palavras existem na frase
- Quantas vezes a dupla "ma" apareceu na frase.

EXERCÍCIO 182 Escrever um algoritmo que leia uma linha de caracteres (máximo de 60) contendo uma frase (palavras separadas por branco). O algoritmo deve imprimir as palavras cujo comprimento for maior que 6 caracteres.

EXERCÍCIO 183 Dado um texto, sem nenhum caracter de controle, na forma de um vetor, imprimi-lo com tamanho de linha = 60, alinhado a esquerda. Não quebrar palavras.

EXERCÍCIO 184 Dado um texto, na forma de um vetor e formado por minúsculas e maiúsculas indistintamente, imprimi-lo todo em maiúsculas.

EXERCÍCIO 185 Dado um texto, na forma de um vetor e formado só por minúsculas, imprimi-lo colocando cada primeira letra de todas as palavras em maiúsculos.

EXERCÍCIO 186 Dado um texto, na forma de um vetor, calcular a quantidade de letras, brancos e palavras, imprimindo estes resultados ao final.

EXERCÍCIO 187 Escrever um algoritmo para criptografar textos, usando o algoritmo da "Criptografia de Cezar", para k=n, onde n deve ser lido e é variável embora fixo no texto a criptografar.

EXERCÍCIO 188 Dado um texto, sem nenhum caracter de controle, na forma de um vetor, imprimi-lo com tamanho de linha variável, informado no início do programa, alinhado a esquerda. Não quebrar palavras.

EXERCÍCIO 189 Dado um texto, sem nenhum caracter de controle, na forma de um vetor, imprimi-lo com tamanho de linha variável, informado no início do programa, alinhado à direita. Não quebrar palavras.

EXERCÍCIO 190 Suponha que todas as palavras terminadas em "a"são femininas e todas as terminadas em "o"são masculinas. Defina um algoritmo que leia uma frase de no máximo 80 caracteres contendo palavras separadas por um branco, e com a frase terminando por um ".". O algoritmo deve:

1. Contar quantas palavras existem
2. Quantas são masculinas
3. Quantas são femininas

Imprimir estes resultados ao final

EXERCÍCIO 191 Imagine um algoritmo capaz de ler uma entrada formada por até 60 caracteres, que correspondem a uma frase. Esta frase termina quando for encontrado um ponto. O algoritmo deve ser capaz de responder:
a) Qual o tamanho da frase
b) Quantas palavras existem na frase
c) Quantas vezes a dupla "ma"apareceu na frase.

Correspondência de cadeias

Este problema surge com alguma freqüência em diversos lugares na ciência da computação. O caso mais comum é o de pesquisar uma palavra dentro de um texto que está sendo editado. Qualquer processador de texto faz isso. Em outra aplicação busca-se comparar 2 cadeias de ácidos nucleicos dentro de uma molécula de DNA. Aqui são apenas 4 letras (ACTG) e largas seqüências são buscadas.

Eis a formalização do problema: Há uma cadeia C composta por 1..n caracteres. Há uma palavra a buscar, P composta por 1..m caracteres. Naturalmente $n \geq m$. Os caracteres de C e de P foram tirados de algum alfabeto finito Σ. Assim, podemos ter $\Sigma = \{0, 1\}$, ou $\Sigma = \{A, C, T, G\}$ ou o alfabeto ASCII ou similar.

Diz-se que P ocorre em T com deslocamento s, se a partir da posição $s + 1$ de T e pelos próximos m caracteres, há uma corresponência completa entre P e T.

O problema da correspondência de caracteres busca encontrar todos os delocamentos válidos em T que correspondem a P.

Há duas classes de algoritmos de busca aqui: o primeiro, é o mais simples, chamado de *força bruta*, não realiza nenhum pré-processamento nas cadeias

Força Bruta

```
1: Algoritmo Força Bruta para busca em cadeias (padrão P, texto T)
2: i ← 1
3: CONTADOR ← 0
4: enquanto (i < tamanho(T)-tamanho(P)) faça
5:    aux ← 0; j ← 1
6:    enquanto (j ≤ tamanho(P)) ∧ (T[aux + i] = P[j]) faça
7:       CONTADOR++
8:       aux++
9:       j++
10:   fimenquanto
11:   se j = tamanho(P) + 1 então
12:      escreva "ocorrência em ",i
```



```
13:       i ← i + tamanho(P) {pode ser +1. Depende da definição}
14:     senão
15:       i++
16:       CONTADOR++
17:     fimse
18: fimenquanto
19: fim algoritmo
```

Este algoritmo é claramente ruim, tem complexidade no pior caso de $O(tamanho(P) \times tamanho(T))$. Acompanhe no exemplo, a quantidade de testes que é feita:
T='o galo o gato e a gata gaguejaram' e P='gato'
QUANTIDADE DE TESTES = 38

Algoritmos eficientes

Há toda uma família de algoritmos que diminuem o tempo de processamento nesta tarefa. Todos realizam algum tipo de pré-processamento sobre as cadeias T e/ou P.

Rabin-Karp

A idéia deste algoritmo é converter a cadeia em números (ou melhor dizendo, é usar a interpretação numérica das cadeias) para efeito da busca. Dado que, caracteres em computador são representados usando bits, sempre será possível este procedimento sem perda de generalidade.

O algoritmo começa calculando a representação decimal de P, o que ocupa p dígitos. Depois, um vetor de mesmo comprimento de T é calculado. A cada p dígitos de P a sua representação decimal é calculada e este valor é guardado.

O processo é rápido. A cada deslocamento em T, basta excluir o dígito de mais alta ordem e acrescentar um novo dígito à direita.

A dificuldade neste caso é quanto ao tamanho dos números envolvidos, sobretudo se P é grande. A maneira de contornar esta dificuldade é usar a aritmética dos ponteiros do relógio, com o módulo q. Este valor usualmente é escolhido de forma que $10q$ caiba em uma palavra do computador de modo a poder fazer toda a aritmética inteira.

Entretanto, esta escolha introduz a possibilidade de *acertos espúrios*. Isto ocorre quando houver a coincidência de P mod q com T[i..j] mod q, sem que haja P = T[i..j].

Entretanto, pela escolha judiciosa de q, pode-se garantir que haja poucos acertos espúrios. Seja como for, a igualdade numérica não garante a localização de P em T. Há que se fazer o testye explícito a seguir. A vantagem do algoritmo de Rabin-Karp é excluir a grande maioria de candidatos em uma busca linear simples.

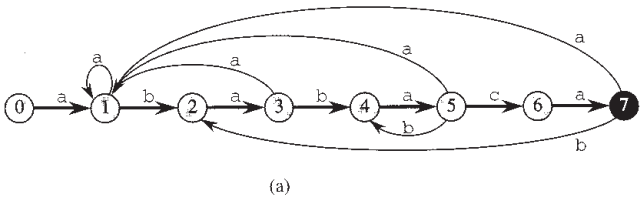
Autômatos de correspondência

O autômato finito é uma máquina de estados composto por um conjunto finito de estados, um estado inicial, um alfabeto finito de entrada e uma função $Q \times \Sigma$ em Q, chamada função de transição.

Veja-se um exemplo, extraído de CLR pág. 726 (versão brasileira)

Cada palavra P tem o seu próprio autômato de busca. Depois que o autômato foi construído ele é eficiente: examina-se cada caractere de T uma única vez. O problema é que a construção do autômato pode ser demorada principalmente se Σ é grande.

Veja-se a seguir um autômato para pesquisar P="ababaca", extraída da mesma obra, pág. 727.



estado	a	b	c	P
0	1	0	0	a
1	1	2	0	b
2	3	0	0	a
3	1	4	0	b
4	5	0	0	a
5	1	4	6	c
6	7	0	0	a
7	1	2	0	

i	–	1	2	3	4	5	6	7	8	9	10	11
$T[i]$	–	a	b	a	b	a	b	a	c	a	b	a
estado $\phi(T_i)$	0	1	2	3	4	5	4	5	6	7	2	3

Algoritmo de Boyer-Moore

Este algoritmo, descrito na década de 70, usa 3 artifícios aceleradores que permitem aumentar o tamanho do salto

- a busca é feita da esquerda para a direita
- Pré-compila uma tabela de ocorrências de caracteres
- Pré-compila uma tabela de ocorrências de prefixos no padrão.

Os algoritmos são: (referência para todos CLR, inglês, pag 870)

```
1:  Algoritmo BOYER-MOORE (texto T, padrao P)
2:  tabela1 ← CARACTERRUIM (padrão P, alfabeto S)
3:  tabela2 ← SUFIXOBOM (padrão P)
4:  i ← 0
5:  enquanto (i ≤ tamanho(T) - tamanho(P)) faça
6:     j ← tamanho(P)
7:     enquanto (j > 0) ∧ P[j] = T[i + j] faça
8:        j–
9:     fimenquanto
10:    se (j = 0) então
11:       escreva "ocorrência em ",i
12:       i ← i + tamanho(P)
13:    senão
14:       i ← i + MAX ((j-tabela1[T[i+j]]), tabela2[j])
```



```
15:    fimse
16: fimenquanto
17: fim algoritmo
```

```
1: Algoritmo CARACTERRUIM (padrão P, alfabeto S) {alfabeto S contém todos os
   caracteres de T (não repetidos)}
2: para cada caracter do alfabeto faça
3:    tabela1[indice do caracter] ← 0
4: fimpara
5: para j de 1 até tamanho(P) faça
6:    tabela1 [indice do caracter P[j]] ← j
7: fimpara
8: retorne tabela1
9: fim algoritmo
```

Exemplo deste algoritmo: CARACTERRUIM ('gato', 'o/galteujrm') 4 0 1 2 0 3 0 0 0 0 0, significando salto de 4 para /leujrm, salto de 3 (4-1) para 'g', 2 (4-2) para 'a' e 1 para 't'. Nenhum salto para 'o'.

A segunda tabela é construida pela função SUFIXOBOM, cujo algoritmo é:

```
1: Algoritmo SUFIXOBOM (padrão P)
2: tabelaPI ← PREFIXO (P)
3: tabelaPILINHA ← PREFIXO (reverso P)
4: tabela2 ← tamanho(P) - tabelaPI [tamanho (P)] {repetir tam(P) vezes}
5: para LM de 1 até tamanho(P) faça
6:    j ← tamanho(P) - tabelaPILINHA [LM]
7:    se (tabela2 [j] > LM - tabelaPILINHA [LM]) então
8:       tabela2[j] ← LM - tabelaPILINHA [LM]
9:    fimse
10: fimpara
11: retorne tabela2
12: fim algoritmo
```

```
1: Função PREFIXO (padrão P)
2: tabelaPREFIXOS ← 0..0 (o mesmo tamanho de P)
3: k ← 0
4: para q de 2 até tamanho(P) faça
5:    enquanto (k > 0) ∧ (P[k+1] ≠ P[q]) faça
6:       k ← tabelaPREFIXOS [k]
7:    fimenquanto
8:    se P[k+1] = P[q] então
9:       k++
10:   fimse
11:   tabelaPREFIXOS [q] ← k
12: fimpara
13: retorne tabelaPREFIXOS
14: fim algoritmo
```

No exemplo do teste 'o galo o gato e a gata gaguejaram' para o padrão P='gato' aqui, o número de testes é de apenas 10. Compare com os 38 da FORÇA BRUTA.

Vejamos um exemplo:

```
T=FOLGA MORRENDO PORQUE ALEM PORVUE DOS ANDES
P=PORQUE
     F O L G A   M R E N D P Q U V S              P O R Q U E
     0 2 0 0 0 0 0 3 6 0 0 1 4 5 0 0              6 6 6 6 6 1
```

Resposta 7 (contra 41 da força bruta)

```
Tabela 1: (Caracter ruim)
Ao se encontrar um T[x] diferente de um
P[y], em T1 se indica de quantas
posicoes pode-se saltar em x (no T), ou
seja, cada letra em T tem o
indice da ultima ocorrencia desse
caracter de T em P.

Exemplo:
Se T='o gato caiu' e P='gato', temos a primeira
comparacao entre T[4]=a e P[4]=o, como sao
diferentes, a T1 manda pular
     o   g a t c i u
 T1= 4 0 1 2 3 0 0 0
2 posicoes, (ja que j=4 e T[a]=2 e 4-2=2)

Tabela 2: (Sufixo bom)
Realiza um processamento apenas em P, e busca
sufixos do padrao que tenham
sido repetidos anteriormente no padrao.

Por exemplo, se P='abeb', a T2 e: 4,4,2,1, ja
que o sufixo b (na p.4) ja ocorreu anteriormente
em P (na p.2). Note que isto so e verdade que ja
P[1]<> P[3]
FOLGA MORRENDO PORQUE ALEM PORVUE DOS ANDES
     F O L G A   M R E N D P Q U V S     P O R Q U E
 T1= 0 2 0 0 0 0 0 3 6 0 0 1 4 5 0 0 T2= 6 6 6 6 6 1
FOLGA MORRENDO PORQUE ALEM PORVUE DOS ANDES
     =
PORQUE
Salto: 6 Prop T1: 6 versus  Prop T2: 1
---------------------------------
FOLGA MORRENDO PORQUE ALEM PORVUE DOS ANDES
          =
      PORQUE
Salto: 6 Prop T1: 6 versus  Prop T2: 1
---------------------------------
FOLGA MORRENDO PORQUE ALEM PORVUE DOS ANDES
               =
            PORQUE
Salto: 3 Prop T1: 3 versus  Prop T2: 1
---------------------------------
FOLGA MORRENDO PORQUE ALEM PORVUE DOS ANDES
                  =
               PORQUE
     S= 6 *** ACHOU ***
---------------------------------
FOLGA MORRENDO PORQUE ALEM PORVUE DOS ANDES
                          =
```



```
                      PORQUE
Salto: 6 Prop T1: 6 versus  Prop T2: 1
---------------------------------
FOLGA MORRENDO PORQUE ALEM PORVUE DOS ANDES
                                =
                             PORQUE
Salto: 6 Prop T1: 4 versus  Prop T2: 6
---------------------------------
FOLGA MORRENDO PORQUE ALEM PORVUE DOS ANDES
                                      =
                                   PORQUE
Salto: 6 Prop T1: 6 versus  Prop T2: 1
---------------------------------
```

EXERCÍCIO 192 Segundo a física, quando temos diversas resistências ligadas em paralelo, podemos substituí-las por uma única resistência chamada EQUIVALENTE, que tem uma resistência calculada pela fórmula:

$$\frac{1}{R_{eq}} = \frac{1}{R_1} + \frac{1}{R_2} + ... + \frac{1}{R_n}$$

Definir um algoritmo que leia um conjunto indeterminado de resistências, suponha-as ligadas em série, calcule e escreva a resistência equivalente do conjunto. Dados terminam quando for lido um valor igual a zero. Por exemplo, se forem lidos os valores de resistências iguais a 5, 10, 1 e zero, a resposta da resistência equivalente será 10/13.

EXERCÍCIO 193 De acordo com o regulamento do serviço postal americano, não pode seguir pelo correio, nenhum pacote cujo comprimento (maior dimensão) somado à amarração seja superior a 72 polegadas. Amarração é o comprimento do menor barbante que possa toda a volta ao pacote. Construa um algoritmo portugol que leia 3 dimensões e escreva "PODE"se o pacote puder ser mandado via correio, ou NÃO PODE"se não puder. As dimensões estão em centímetros, e deve ser usada a conversão: 1 polegada = 2,5 cm.

EXERCÍCIO 194 Escreva uma função capaz de receber dois números (inteiros) e devolver a média geométrica entre eles (real).

EXERCÍCIO 195 Escreva uma função capaz de receber um valor representativo de uma distância medida em polegadas, e retorne a mesma distância medida em metros (1pol = 2,54cm)

EXERCÍCIO 196 Suponha um algoritmo para cálculo de um D.V. para códigos de 4 dígitos assim formulado:

código	XYZT-d
cálculo de Q	Q = X + Y + Z + T
cálculo de d	d = Q mod 10

Escreva um algoritmo para calcular o DV de uma série de códigos lidos, na forma de 4 inteiros positivos. Para cada código lido, o algoritmo deve imprimi-lo junto com o DV. Os dados terminam quando for lido um número negativo. O cálculo do DV deve ser feito por uma função.

EXERCÍCIO 197 Suponha um algoritmo para cálculo de um D.V. para códigos de 4 dígitos assim formulado:

código	XYZT-d
cálculo de Q	Q = 2X + 3Y + 4Z + 5T
cálculo de d	d = Q mod 7

Escreva um algoritmo para calcular o DV de uma série de códigos lidos, na forma de 4 inteiros positivos. Para cada código lido, o algoritmo deve imprimi-lo junto com o DV. Os dados terminam quando for lido um número negativo. O cálculo do DV deve ser feito por uma função.

EXERCÍCIO 198 Suponha um algoritmo para cálculo de um D.V. para códigos de 4 dígitos assim formulado:

código	XYZT-d
cálculo de Q	Q = 3X + 5Y + 7Z + 9T
cálculo de w	w = Q mod 11
cálculo de d	Se w = 10, então d = 0, senão d = w.

Escreva um algoritmo para calcular o DV de uma série de códigos lidos, na forma de 4 inteiros positivos. Para cada código lido, o algoritmo deve imprimi-lo junto com o DV. Os dados terminam quando for lido um número negativo. O cálculo do DV deve ser feito por uma função.

EXERCÍCIO 199 Suponha um algoritmo para cálculo de um D.V. para códigos de 4 dígitos assim formulado:

código	XYZT-d
cálculo de q1 q1 = 2X + 3Z	
cálculo de q2	q2 = 2Y + 3T
cálculo de Q	q = q1 + q2
cálculo de d	d = q mod 10

Escreva um algoritmo para calcular o DV de uma série de códigos lidos, na forma de 4 inteiros positivos. Para cada código lido, o algoritmo deve imprimi-lo junto com o DV. Os dados terminam quando for lido um número negativo. O cálculo do DV deve ser feito por uma função.

EXERCÍCIO 200 Definir algoritmo capaz de gerar um dígito verificador alfabético, para um código formado por 7 dígitos numéricos. A regra deve ser: $dv = ((d1 \times 8) + (d2 \times 7) + (d3 \times 6) + (d4 \times 5) + (d5 \times 4) + (d6 \times 3) + (d7 \times 2))\ mod\ 26$

dv deve ser transformado de letra em número e a seguir impresso. O algoritmo deve ler inúmeros códigos (na forma cadeia[7]) até ler a cadeia '0000000'.

EXERCÍCIO 201 Definir algoritmo que leia um conjunto indeterminado, mas menor do que 129, de triplas formadas por:

- número do bimestre: inteiro de 1 a 4
- número do aluno: inteiro de 1 a 32
- Nota do aluno: real de 0,0 a 10,0.

Os dados terminam quando for lido um número de bimestre diferente daqueles valores válidos. Após ler o conjunto, o algoritmo deve imprimir:

1. Qual a média anual de cada aluno (32 valores)
2. Qual a média da turma em cada bimestre (4 valores)
3. Qual o número do aluno e o número do bimestre em que ocorreu a MAIOR NOTA absoluta do ano.

4. Qual o bimestre em que ocorreram mais zeros

5. Qual o bimestre com menor amplitude de notas

EXERCÍCIO 202 Definir um algoritmo que calcule o dígito verificador do número de matrícula dos alunos das FACULDADES NEUTRAS. Tal número é formado por 4 dígitos e o verificador é assim calculado:
número: d1 d2 d3 d4
res = (d1 * 2) + (d2 * 3) + (d3 * 5) + (d4 * 7)
dígito = resto da divisão de res por 9.
O algoritmo deve ler uma série de números e para cada um, deve imprimir o número e o correspondente dígito. A série termina quando for lido o número 0.

9.2.1 Calendários

Em 1347 a peste negra devastou a Europa. Esta doença é na verdade uma pneumonia que causa bubões (inchaços nas axilas e virilhas), sendo também chamada de peste bubônica. Os bubões eram caldos de cultura da bactéria causadora, que também era transmitida por pulgas dos ratos. O nome negra, vem do fato de que (supostamente) a carne das vítimas enegrecia pouco antes da morte. A história começa quando uma tribo Mogol de nome Kipchak resolve atacar um posto comercial genovês no Mar Negro. Usaram para isso uma das primeiras armas biológicas que a história registra: cadáveres humanos contaminados com a doença eram atirados através de catapultas para dentro da cidade. Esta embora armada e provisionada para resistir ao cerco, ao ver-se impotente para enfrentar esta arma resolveu fugir de navio de volta a Gênova, abandonando o posto comercial. Junto com eles, foi a doença. Durante os 4 anos seguintes a peste foi para a Sicília, África, Itália, Espanha, França, Inglaterra e depois toda a Europa. Vinte e cinco milhões de pessoas, um quarto da população européia, morreram. Dois séculos passariam antes que a população retornasse ao número de 100 milhões. A devastação da peste negra encerrou um ciclo na história da humanidade. Levantes sociais e políticos espoucaram, a mão de obra escasseou, o campo foi abandonado, os alimentos rarearam. Até aqui, o universo era governado por regras estabelecidas por duas autoridades: Aristóteles (384-322 aC) e o egípcio Cláudio Ptolomeu (100-170 dC). No milênio anterior, a Igreja havia mesclado essas regras à sua visão de mundo, resultando um bloco homogêneo: Deus havia criado a Terra no centro do Universo. Estrelas e planetas giravam em órbitas circulares em volta da terra. Oito esferas concêntricas feitas de material imutável e eterno continham a Lua, o Sol, Marte, Mercúrio, Júpiter, Vênus e Saturno. A última esfera continha as estrelas. Apenas na terra a matéria se decompunha e morria. Nas esferas tudo era eterno. Veja-se o reflexo disso nos nomes dos dias de semana, nos principais idiomas ocidentais:

Corpo Celeste	Inglês	Francês	Italiano	Espanhol
Sol	Sunday	dimanche	domenica	domingo
Lua	Monday	lundi	lunedi	lunes
Marte	Tuesday	mardi	martedi	martes
Mercúrio	Wednesday	mercredi	mercoledi	miércoles
Júpiter	Thursday	jeudi	giovedi	jueves
Vênus	Friday	vendredi	venerdi	viernes
Saturno	Saturday	samedi	sabato	sábado

A reação à peste foi um novo cenário que buscou enterrar e esquecer aquele o mais rápido possível: a renascença. A Itália, pela localização central iniciou o movimento do comércio entre oriente e ocidente. Nasceram aqui os primeiros sistemas administrativos, o conhecimento financeiro e os bancos. A ciência (a matemática) começou a ser usada:

na arquitetura, no comércio, na cartografia. Sabemos hoje que a Terra demora 365 dias, 5 horas, 48 minutos e 46 segundos para uma volta completa ao redor do sol. Egípcios haviam estimado este valor em 365,25 dias (ou seja 365 dias e 6 horas. Ficou uma diferença de 11 minutos e 14 segundos). Com a adoção do calendário egípcio por Júlio Cézar no século I dC, esta disparidade foi se acumulando ano após ano, afastando as datas do calendário das estações. Em meados do século XV ja havia dez dias de atraso. Em 1475 o papa Sisto IV pedira um estudo para determinar a causa do erro. Não houve quem conseguisse compatibilizar Aristóteles e Ptolomeu com o calendário. Podiam garantir estabilidade política, mas estavam cada vez mais incapazes de calcular a data da Páscoa corretamente.

Copérnico (1473-1543) começou a desenvolver em 1506 um sistema astronômico baseado em suas próprias observações e cálculos. Na tentativa de tirar a Terra do centro do universo e colocar aí o Sol, a dificuldade era explicar como as coisas não "caíam"em direção ao sol. Achava-se na época, que a matéria era naturalmente atraída para o centro do universo (embaixo da terra). Quando recrutado pelo secretário do papa, em 1514 para resolver o problema do calendário, Copérnico viu-se num dilema: ou reafirmava a teoria que a vaidade, o medo e a Bíblia haviam montado (estamos no centro do universo) e não resolvia o problema, ou chutava o pau da barraca para propor um novo calendário.

Ele, que bobo não era, recusou o convite, embora continuando a estudar o problema em segredo. Convencido pela correção de seus cálculos e encorajado por amigos, esboçou um rascunho de suas idéias em 1530. Este escrito provocou reações mistas. Finalmente, em 1543, autorizou a publicação de *De Revolutionibus Orbium Coelestium*, comumentemente conhecido como As Revoluções. Copérnico recebeu o primeiro exemplar em 24 de maio de 1543 e morreu poucas horas depois.

O próximo personagem desta história é Galileu Galilei. Pulamos Giordano Bruno, queimado na fogueira no Campo di Fiori em 17 de fevereiro de 1600. Galileu teria deixado cair duas bolas de pesos diferentes da Torre de Pisa para provar que ambas chegariam juntas, contrariando Aristóteles que afirmara chegar a mais pesada antes. Há dúvidas se isso de fato ocorreu. Mais modernamente supõe-se que tenha sido uma experiência intelectual apenas. O argumento intelectual é notável[1]Este é o melhor jeito de fazer física, à la Einstein...

Galileu freqüentou diversas universidades na Itália, sempre conflitando com os demais professores. Escreveu inúmeros livros, "inventou" o telescópio [2] Ao usar os telescópios, viu quatro satélites orbitando Júpiter e os anéis de Saturno e relatou isso no livro *Sidereus Nuncius*(O Mensageiro Celeste). Escreveu de leve, ainda sem adotar completamente o modelo copernicano. A Igreja já começou a enviar mensagens de que não concordava com as idéias desse livro. Em 21 de dezembro de 1614, o padre Tomás Caccini, em Florença criticou Galileu aformando que se Deus parou o sol a pedido de Josué para que os israelitas derrotassem os amoritas, como o sol poderia ser o centro do universo ? O padre foi mais longe: acusou Galileu, a matemática e todos os matemáticos de hereges políticos e religiosos. A defesa de Galileu é perfeita:

> Não me sinto na obrigação de acreditar que o mesmo Deus que nos dotou de sentidos, razão e intelecto, tencionava descartar o uso destes e por algum outro meio nos dar o conhecimento que com eles podemos obter [...] A

[1]Suponha jogar as 2 bolas e suponha que Aristóteles estava certo: a mais pesada chega antes. Agora suponha as mesmas duas bolas porém ligadas por fio. Por um lado pode-se argumentar que a mais leve "segura" a mais pesada e esta demora mais a chegar. Por outro lado, pode-se supor que o fio transforma as duas massas em uma só, que passa a ser mais pesada que a bola anterior. O resultado é que agora as bolas chegam antes do que chegavam. Para que ambos os raciocínios estejam certos e estão, a única possibilidade é elas chegarem juntas.

[2]Ele é assim considerado pois construiu os maiores telescópios em uso na Europa, ultrapassando seus modelos anteriores por diversas vezes.

> intenção do Espírito Santo é ensinar-nos como se vai para o céu e não como o céu funciona.

Em 5 de março de 1616 o cardeal Belarmino da Santa Inquisição decretou que o sistema copernicano era "falso e errôneo" afirmando que Deus fixou a Terra em seus alicerces para jamais ser movida. Galileu nunca se conformou com este decreto, buscando incessantemente a sua revogação. Em 1624, obteve do papa uma autorização para escrever seu livro mais famoso *Diálogo sobre os dois máximos sistemas do mundo*, que foi publicado em 1632 e recebido com louvores por acadêmicos de toda a Europa. Pouco depois, foi acusado pela inquisição de herege. Não pode ver as acusações ou as provas. Ficou no dilema: ou se retratava ou morria como Giordano Bruno. Em decisão que alguns criticaram como prejudicial a ciência, ele resolveu pela vida. Em 22 de junho de 1633, fez uma longa retratação. Diz a lenda que depois de terminada a leitura, ao se erguer da posição de joelhos onde estava,Galileu teria dito *"E pur, si muove"* . Só em 1757 a Igreja retirou a proibição sobre a obra. Em 1992, o papa João Paulo II reconheceu formalmente o erro da Igreja.

Depois vem Isaac Newton, um dos 5 maiores cientistas que a Humanidade produziu (decomposição da luz, cálculo diferencial e integral, gravitação entre outros). Ao escrever sobre a lei universal da gravitação, ele acabou de consolidar o modelo copernicano, que já havia sido engordado com as leis de Kepler sobre o deslocamento dos planetas.

Calendários Juliano e Ptolomeico

A encrenca do calendário está em que ele lida com 3 ciclos distintos: solar: Alterna as estações e depende do sol. Mede os anos; lunar: Alterna as luas (cheia, nova ...) e depende da lua. Mede os meses; semanal: Mede os dias da semana e tem origem religiosa. Indica o sabath.

A dificuldade e que estes 3 ciclos não são múltiplos entre si.

Em 45 aC, Júlio Cezar criou o calendário juliano, baseado no ptolomeico. Neste o ano tinha 365d e um quarto de dia. Os meses foram batizados de Janius (portas e janelas), Februas (festa da purificação), Mars (guerra), Apripe (abertura das flores), Maiores, Juniores, Quintílio, Sextílio, Septílio, Octílio, Novitílio e Decitílio. Logo depois, o próprio JC mudou o nome do Quintílio para Julius. Seu sucessor, Cezar Augusto não deixou por menos, mudando o próximo para Augustus.

Em 1582, o papa Gergório mudou o calendário, que com a reforma passou a ser conhecido como calendário gregoriano. Sumiram 10 dias em março desse ano. Adotou-se a regra dos anos bissextos. A adoção deste calendário demorou. Na Inglaterra ele só foi aceito em 1752. Em 1793, a Revolução Francesa mudou novamente o calendário. Nele há 12 meses de 30 dias e 5 feriados nacionais. Os meses: germinal, floral, thermidor, fructidor, brumaire. Napoleão o aboliu em 1805.

Novas propostas

Em 1954, a ONU propôs um novo calendário. Ele tem 52 semanas de 7 dias = 364 dias. O dia 365 é no final do ano e não tem número nem nome. Os quadrimestres sempre tem 31 + 30 + 30 + 30 dias.

Outra proposta foi o do calendário fixo, com 13 meses de 28 dias cada um. Aqui, as segundas sempre serão dias 1, 8, 15 e 22. Falta 1 dia, que fica no final do ano. O dia do bissexto é após 28 de junho. O novo mês é o SOL que fica entre junho e julho.

Finalmente, há uma nova proposta de estabelecer um tempo mundial, (desvinculado do sol), orientado a negócios e com divisões decimais.

Regra do bissexto

Sejam R4 ← resto da divisão do ano por 4; R100 ← resto da divisão do ano por 100 e R400 ← resto da divisão do ano por 400.

SE R4=0 ∧ ((R100 ≠ 0) ∨ (R400 = 0)) o ano é bissexto senão não é.

EXERCÍCIO 203 Imagine um cilindro cujo diâmetro da base tem o mesmo comprimento de sua altura. Dentro dele está inscrita uma esfera, cujo raio é o mesmo raio da base do cilindro. Escrever algoritmo que leia uma série de raios, e para cada um deles, calcule e escreva a diferença de volume entre o cilindro e a esfera. Os raios são lidos em cm e a diferença deve ser expressa em cm3. O ultimo raio é zero, e indica fim. Definir duas funções: uma para o cálculo do volume da esfera, e outra para o cálculo do volume do cilindro.

$$V_{esfera} = \frac{4}{3} \times \pi R^3$$

$$V_{cilindro} = \pi \times R^2 \times h$$

EXERCÍCIO 204 Definir algoritmo que calcule e escreva a soma dos primeiros 50 números pares acima de 1000 (1000 incluído), com os 60 primeiros números ¡mpares acima de 500 (501 inclu¡do).

EXERCÍCIO 205 Definir algoritmo que determine se uma seqüência contendo 1000 elementos, ESTÁ ou NÃO ESTÁ classificado em ordem descendente. Ao final do algoritmo, este deve imprimir as mensagens "EM ORDEM"ou "FORA DE ORDEM"conforme o caso.

Algoritmo Balance Line

Este algoritmo teve importância muito maior há 30 anos, quando os recursos de hardware não eram nem de longe tão abundantes quanto hoje. Em 1974, existiam na cidade de Curitiba 7 computadores: o da Universidade Federal do Paraná (um IBM 1130 com 8Kb de memória), e os computadores da Celepar, Copel, Bamerindus, Banestado, Cr Almeida e URBS, sendo que cada uma destas empresas tinha 1 (um) computador.

Como fazer para atualizar 80.000 registros (caso da folha de pagamento do Estado do Paraná) todos os meses com apenas um computador que era usado para todos os sistemas do Goverbo do Estado ?

Vale lembrar também que não existiam sistemas on-line, nem terminais remotos, muito menos micros.

A solução para este problema está no algoritmo que será estudado nesta aula, chamado Balance Line. Ele permite que grandes volumes de dados para tratamento atuarial (inclusão, alteração e exclusão) sejam serializados e processados sem interferência do operador.

Importância atual Este algoritmo ainda é muito importante, mesmo com abundância de hardware. Ele, na verdade, permite economizar também tempo de operador, e este sempre deve ser economizado (principalmente quando o operador somos nós. ;-).

Imagine-se um site onde pessoas devem fazer atualizações. Uma coisa é entrar 1 vez por semana e fazer uma pequena atualização. Outra é precisar entrar de uma só vez e alterar digamos 400 registros.

Para o primeiro caso, a estratégia atualmente usada (uma transação atuarial que sob o comando do operador altera o dado) está perfeita. Para a segunda, não. Poucos tem paciência para refazer esta mesma transação 400 vezes.

Neste segundo caso, será muito bom se o dono do sistema disponibilizar um balance line. Neste caso, o operador preparará uma massa de atualização (possivelmente usando o notepad ou similar), fará as verificações e correções, e quando a massa estiver OK, submete-la-á, eventualmente via upload para atualização serializada. Ao invés de 400 transações, o operador terá que se preocupar com apenas 1. O resto fica por conta do algoritmo Balance Line.

Capítulo 10

Exercícios práticos: 003 -Raiz quadrada

10.1 Exemplo de um algoritmo: raiz quadrada

Suponha que sua vida dependa de obter a raiz quadrada de um determinado número, por exemplo 54.768,87. O que você faria ?

1. A solução mais óbvia seria buscar uma calculadora capaz de extrair uma raiz quadrada. Por azar de nossa simulação, não há calculadoras, computadores, celular, etc. Nenhuma traquitana disponível.

2. Considerando que a operação de multiplicação é razoavelmente conhecida, outra possibilidade, seria avançar na base da tentativa-e-erro, usando o seguinte algoritmo.

   (a) Chute um número

   (b) Multiplique-o por ele mesmo

   (c) Se o resultado for igual ao número procurado, parabéns!

   (d) Se for menor, aumente o chute

   (e) Se for maior, diminua o chute

   (f) Volte à etapa da multiplicação

3. Outra possibilidade, esta trazida pela teoria dos logaritmos (não confunda logaritmo com algoritmo) é achar o logaritmo do número dado, dividir este valor por 2 e depois achar o antilogaritmo. Por exemplo, se não soubermos que $\sqrt{64} = 8$, podemos fazer o $\log_2 64 = 6$. Dividindo $6 \div 2 = 3$. E $2^3 = 8$. O problema está que poucas pessoas tem uma táboa de logaritmos na cabeça, ou mesmo conhecem as propriedades dos logaritmos.

4. A resposta exata vem através do algoritmo de extração de raiz quadrada, posto à nossa disposição pela matemática.

Algoritmo para raiz quadrada

Por definição, dado $y = \sqrt{x}$, onde x é não negativo, deve-se achar o real y, tal que $y^2 = x$. Embora o número localizado seja positivo, deve-se lembrar que – em tese – ele

também pode ser negativo, mas com o mesmo valor absoluto. Se não for possível obter a igualdade ($y^2 = x$), espera-se que y^2 seja menor que x e tão próximo quanto possível.

As etapas do algoritmo são:

1. Pegue uma folha de papel e escreva uma cruz grande, divindo a área de trabalho em 4 quadrantes.

2. Escreva o número do qual se quer achar a raiz quadrada (a quem chamaremos α), no quadrante superior esquerdo. Ao final do processo, a raiz vai aparecer no quadrante superior direito.

3. Separe o número α em grupos de 2 dígitos, a partir de sua vírgula decimal, à direita e à esquerda, completando com zeros não significativos quando necessário. Feito isso, nomearemos os grupos (de 2 dígitos) da esquerda para a direita como G_1, G_2, ... G_n.

4. Relembremos a tabela de quadrados perfeitos de 1 a 100.

1	2	3	4	5	6	7	8	9
1	4	9	16	25	36	49	64	81

   Esta tabela é importante pois todos os grupos acima formados são menores do que 100.

5. Busque o número que elevado ao quadrado gere um número menor ou igual do que G1. Coloque este número no quadrante superior direito. Chame-se este número como A_1. Fica

G1.G2.G3. ...	A1

6. Eleve A_1 ao quadrado e escreva o resultado no quadrante inferior esquerdo, embaixo de G_1. Chamemos este número A_1^2 de P_1.

7. Faça G_1 menos P_1 e escreva o resultado da subtração embaixo de P_1 (no quadrante inferior esquerdo). Chamemos este número de T_1. Fica

G1.G2.G3. ...	A
P1	
−	
T1	

8. (aaa) Baixe o próximo grupo (neste caso o G_2, colocando-o ao lado de T_1, junte-os e obtenha um novo número (T_1G_2).

9. Dobre o número A_1 (multiplicando-o por 2) e escreva-o no quadrante inferior direito, embaixo de A_1. Chame-se este número de D_1.

10. (***) Divida (divisão inteira) T_1G_2 por $D_1 \times 10$. Este resultado deve ser escrito em 3 lugares:

    - Ao lado de A_1, já que ele é o próximo número da raiz
    - Ao lado de D_1. Imediatamente depois, escreva um sinal de $\times$ (vezes)
    - Ao lado do sinal de $\times$ (vezes) escrito acima

11. Calcule a multiplicação posta acima. Se o resultado for menor ou igual a T_1G_2, prossiga. Se for maior, OPS!, Volte à etapa (***) e repita o processo com um resultado uma unidade menor.

12. Escreve-se o resultado da multiplicação embaixo de T_1G_2 e efetua-se a subtração.

13. Volte a (aaa)

Exemplo

Seja calcular $\sqrt{14796,5}$

```
01 47 96 50 00 00 00      121.640
============              ===========
-1                        rq(1)=1
-----------               -----------
 0 47                     2 x 1 = 2
  -44                     47 / 20 = 2
-------------             22 x 2 = 44
    3                     --------------
    396                   2 x 12 = 24
   -241                   396 / 240 = 1
-----------------         241 x 1 = 241
    155                   --------------
    15550                 2 x 121 = 242
   -14556                 15550 / 2420 = 6
-------------------       2426 x 6 = 14556
      994                 --------------
      99400               2 x 1216 = 2432
     -97296               99400 / 24320 = 4
----------------------    24324 x 4 = 97296
       2104               --------------
       210400             2 x 12164 = 24328
      -0                  210400 / 243280=0
------------------------- 243280 x 0 = 0
       210400             --------------
       21040000
```

Treino

Façamos a raiz quadrada de 1267.074.

```
      raizq  1267.074
 12 67 07 40 00 00
============                          ===========
                                      rq(  ) v =
-----------                           -----------
                                       x =
                                                    ops
-------------                            /  =
                                        x =
                                      --------------
                                       x  =
----------------                                    ops
                                          /   =
                                        x =
                                      --------------
--------------------                   x   =
                                           /    =
                                        x =
                                      --------------
----------------------                 x    =
```



```
                  /     =
                 x =
              --------------
```

10.2 Exercício 1

Simule a execução do algoritmo acima da raiz quadrada para os valores abaixo e em cada um, descubra o quinto número que é subtraído no algoritmo. Por exemplo, no exemplo dado acima, o quinto número seria 97296 e no que fizemos junto seria 355925.

```
24681
52543
26008
```

Nos 3 casos, ache o QUINTO NÚMERO QUE É SUBTRAÍDO na parte esquerda da tabela acima.

1	2	3

10.3 Exercício 2

Simule a execução do algoritmo acima da raiz quadrada para os valores abaixo e em cada um, descubra o quinto número que é subtraído no algoritmo. Por exemplo, no exemplo dado acima, o quinto número seria 97296 e no que fizemos junto seria 355925.

```
44431
28819
37226
```

Nos 3 casos, ache o QUINTO NÚMERO QUE É SUBTRAÍDO na parte esquerda da tabela acima.

1	2	3

10.4 Exercício 3

Simule a execução do algoritmo acima da raiz quadrada para os valores abaixo e em cada um, descubra o quinto número que é subtraído no algoritmo. Por exemplo, no exemplo dado acima, o quinto número seria 97296 e no que fizemos junto seria 355925.

```
38692
26660
55802
```

Nos 3 casos, ache o QUINTO NÚMERO QUE É SUBTRAÍDO na parte esquerda da tabela acima.

1	2	3

10.5 Exercício 4

Simule a execução do algoritmo acima da raiz quadrada para os valores abaixo e em cada um, descubra o quinto número que é subtraído no algoritmo. Por exemplo, no exemplo dado acima, o quinto número seria 97296 e no que fizemos junto seria 355925.

```
46951
41068
14107
```

Nos 3 casos, ache o QUINTO NÚMERO QUE É SUBTRAÍDO na parte esquerda da tabela acima.

1	2	3

10.6 Respostas

```
1         0   91684  193476
2    337184  203676  154336
3         0  228529   94484
4    346624  202625  166229
```


Capítulo 11

Exercícios Práticos: 004-Introdução

11.1 Algoritmos

> Um algoritmo é um método finito, escrito em um vocabulário simbólico fixo, regido por instruções precisas, que se movem em passos discretos, cuja execução não requer *insight*, esperteza, intuição, inteligência, ou clareza e lucidez e que mais cedo ou mais tarde chega a um fim. (David Berlinski)

O algoritmo é o que está por trás de todos os programas de computadores. Fazendo

uma analogia com a Rose (a empregada robot da família Jetson). Quando ela apresenta uma bela torta de maçãs, a torta é o produto final, a Rose é o computador e a receita da torta é o algoritmo.

Algoritmos são bem antigos na história do mundo. Provavelmente o mais antigo que é conhecido é devido a Euclides. Para poder observá-lo, vamos relembrar uma operação da aritmética básica. Trata-se da divisão inteira. Quando os números inteiros A e B são divididos em divisão inteira, dois resultados aparecem: o quociente e o resto.

Por exemplo, na divisão inteira $10 \div 7$, o quociente (resultado) é 1, enquanto o resto é 3.

Nos nossos algoritmos, o quociente da divisão inteira vai ser representado pela palavra *div*, já que o símbolo $\div$ e mesmo a barra simples / ficarão reservados para a divisão real. Portanto poderemos afirmar que

$$10 \; div \; 7 = 1$$

Da mesma maneira, o resto da divisão inteira, será representado pela palavra *mod*, e aqui podemos afirmar que

$$10 \; mod \; 7 = 3$$

.

Para treinar Procure calcular e responder:

1. $10 \; div \; 2 =$_____.

2. $10\ mod\ 2 =$_____.
3. $100\ div\ 33 =$_____.
4. $100\ mod\ 33 =$_____.
5. $8\ div\ 2 =$_____.
6. $8\ mod\ 2 =$_____.
7. $45\ div\ 11 =$_____.
8. $45\ mod\ 11 =$_____.
9. $2\ div\ 13 =$_____.
10. $2\ mod\ 13 =$_____.

11.1.1 Máximo Divisor Comum

Define-se o máximo divisor comum entre dois números inteiros A e B como sendo o maior número inteiro que divide simultaneamente tanto A como B sem deixar resto.

Por exemplo, se quisermos achar o MDC entre 10 e 14, teremos que achar os divisores de 10 (que são: 1, 2, 5 e 10) e de 14 (que são 1, 2, 7, 14) e localizar o maior número que é comum às duas listas, no caso 2. Assim, 2 é o MDC entre 10 e 14.

Para treinar Informe qual o MDC entre

1. $MDC(15, 18) =$_____.
2. $MDC(20, 40) =$_____.
3. $MDC(17, 31) =$_____.
4. $MDC(2, 11) =$_____.

Do exercício anterior percebeu-se que quando um dos números é primo, o MDC é sempre 1. Isto decorre da definição dos números primos: Um número inteiro é primo, quando seus únicos divisores inteiros são a unidade e ele próprio.

11.1.2 Algoritmo do Máximo Divisor Comum

Este algoritmo sempre é chamado com dois inteiros. Note que o primeiro número deve ser maior ou igual do que o segundo. Se esta condição não estiver satisfeita, os números devem ser INVERTIDOS antes de serem entregues ao algoritmo. Este algoritmo recursivo é devido a Euclides

```
1: inteiro função MDC(inteiro a, b) {obrigatoriamente: a ≥ b}
2: se b = 0 então
3:     devolva a
4:     \\ esta é outra maneira de identificar comentários
5: senão
6:     devolva MDC(b, (a mod b))
7: fimse
8: fim {função}
```

Suponha que far-se-á a chamada ao algoritmo acima, como segue:

```
MDC(1200, 1119)
O que aconteceria ?
Chamando com 1200 1119, Chamando com 1119 81, Chamando com 81 66,
Chamando com 66 15, Chamando com 15 6, Chamando com 6 3 e
Chamando com 3 0
Resposta é 3
```

11.1.3 Algoritmos no dia a dia

Na nossa vida, também usamos muitos algoritmos. Aqui eles têm uma característica tão restrita como os algoritmos computacionais, porque supostamente eles serão seguidos por seres inteligentes. Mas, vale a experiência: acompanhe o algoritmo que um de nós poderia usar para ir do Unicenp até a Praça Tiradentes

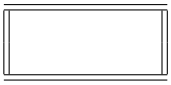
```
 1: Algoritmo para ir do Unicenp até a Praça Tiradentes
 2: se tenho dinheiro ? então
 3:    se tenho pressa ? então
 4:       chamo um taxi
 5:       mando ele ir para a Praça
 6:    senão
 7:       procuro uma carona OU
 8:       vou para o ponto do ônibus
 9:       ao desembarcar, pergunto qual a direção da Praça
10:       sigo andando até chegar nela
11:    senão
12:       saio do Unicenp pela JK, pegando à direita
13:       pego a BR277 à direita, sigo até a Pe Agostinho
14:       pego a Pe Agostinho à esquerda e sigo até a Al. Cabral
15:       viro à direita até a Carlos de Carvalho
16:       viro à esquerda e sigo até a Praça Tirandentes
17:    fimse
18: fimse
```

11.1.4 Exercício 1

Chame o algoritmo mdc com os valores

```
1987      1669
```

Responda no quadro próprio, qual o MDC pedido

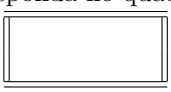


11.1.5 Exercício 2

Chame o algoritmo mdc com os valores

```
1858      1376
```

Responda no quadro próprio, qual o MDC pedido

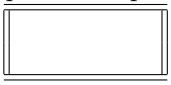


11.1.6 Exercício 3

Chame o algoritmo mdc com os valores

```
1370        1224
```

Responda no quadro próprio, qual o MDC pedido

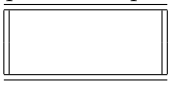


11.1.7 Exercício 4

Chame o algoritmo mdc com os valores

```
1939        1037
```

Responda no quadro próprio, qual o MDC pedido

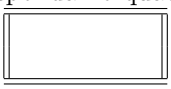


11.1.8 Exercício 5

Chame o algoritmo mdc com os valores

```
1659        1434
```

Responda no quadro próprio, qual o MDC pedido

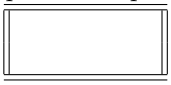


11.1.9 Exercício 6

Chame o algoritmo mdc com os valores

```
1777        1471
```

Responda no quadro próprio, qual o MDC pedido

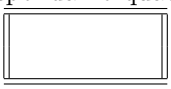


11.1.10 Exercício 7

Chame o algoritmo mdc com os valores

```
1808        1104
```

Responda no quadro próprio, qual o MDC pedido

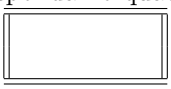


11.1.11 Exercício 8

Chame o algoritmo mdc com os valores

```
1380        1234
```

Responda no quadro próprio, qual o MDC pedido

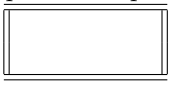


11.1.12 Exercício 9

Chame o algoritmo mdc com os valores

```
1876        1721
```

Responda no quadro próprio, qual o MDC pedido



11.1.13 Exercício 10

Chame o algoritmo mdc com os valores

```
1724        1707
```

Responda no quadro próprio, qual o MDC pedido

11.1.14 Respostas

```
 1    1
 2    2
 3    2
 4    1
 5    3
 6    1
 7   16
 8    2
 9    1
10    1
```


Capítulo 12

Exercício prático: 006-Jogo da Vida

12.1 O jogo da vida

O Jogo da vida foi desenvolvido pelo matemático britânico John Horton Conway em 1970. Começou com uma brincadeira mas depois virou coisa séria.

O jogo foi criado de modo a reproduzir, através de regras simples, as alterações e mudanças em grupos de seres vivos, tendo aplicações em diversas áreas da ciência.

As regras definidas são aplicadas a cada nova "geração"; assim, a partir de uma imagem em um tabuleiro bi-dimensional definida pelo jogador, percebem-se mudanças muitas vezes inesperadas e belas a cada nova geração, variando de padrões fixos a caóticos.

Origem Um dos problemas matemáticos mais famosos dos anos 40 era o de achar uma máquina que fosse capaz de construir cópias de si mesma, que teve uma solução baseada em um autômato celular extremamente engenhoso e complicado inventado pelo renomado matemático John von Neumann. Conway inventou o Jogo da Vida (ou Game of Life) ao utilizar suas descobertas anteriores relacionadas com o problema de encontrar um grupo simétrico de esferas em 24 dimensões, proposto por John Leech para simplificar a solução de von Neumann.

O jogo fez sua primeira aparição na edição de Outubro de 1970 da Scientific American, na coluna de jogos matemáticos de Martin Gardner. De um ponto de vista teórico, ele é interessante, pois tem o poder de uma máquina de Turing universal: tudo pode ser computado através de algoritmos no Jogo da Vida de Conway. Também era dito desde 1970 que foi destinado mais tempo de computação ao Jogo da Vida do que a qualquer outra atividade.

Desde sua publicação, ele tem atraído muito interesse devido aos caminhos surpreendentes que pode tomar. Life é um exemplo de auto-organização. Ele é interessante para biólogos, matemáticos, economistas, filósofos e outros a observar o modo como imagens complexas podem surgir de implementações de regras muito simples.

O Jogo da Vida tem um número de imagens reconhecidas que emergem de posições iniciais particulares. Antes da publicação, muitas imagens interessantes já haviam sido descobertas, incluindo o sempre envolvente R-pentominó, o auto propulsionado "glider", e várias "guns"(armas) que geravam um fluxo sem fim de novas imagens, todas aumentando o interesse no jogo. Sua popularidade foi ajudada pelo fato de ele ter vindo justo no momento em que uma nova geração de minicomputadores de baixo custo estavam

sendo disponibilizados no mercado, significando que o jogo poderia ser rodado por horas nessa máquinas que não eram utilizadas de noite. Isto escondeu a popularidade posterior dos fractais gerados por computador. Para muitos aficcionados, Life era simplesmente um desafio de programação, um método divertido de gastar os ciclos da CPU. Para alguns, no entanto, Life tinha conotações mais filosóficas. Ele desenvolveu um culto através dos anos 70 e no meio dos 80; os desenvolvimentos atuais foram tão longe a ponto de criar emulações teóricas de sistemas de computadores com as regras de um tabuleiro de Life.

Regras do Jogo da Vida As regras são simples e elegantes:

- Qualquer célula viva com menos de dois vizinhos vivos morre de solidão.
- Qualquer célula viva com mais de três vizinhos vivos morre de superpopulação.
- Qualquer célula com exatamente três vizinhos vivos se torna uma célula viva.
- Qualquer célula com dois vizinhos vivos continua no mesmo estado para a próxima geração.

É importante entender que todos os nascimentos e mortes ocorrem simultaneamente. Juntos eles constituem uma geração ou, como podemos chamá-los, um "instante"na história da vida completa da configuração inicial.

Descrição Este "jogo"é na realidade um jogo sem jogador, o que quer dizer que sua evolução é determinada pelo seu estado inicial, não necessitando de nenhuma entrada de jogadores humanos. Ele é jogado em um conjunto de células quadradas que seguem ao infinito em todas as direções. Cada célula tem oito "vizinhos", que são as células adjacentes, incluindo as diagonais. Cada célula pode estar em dois estados: "viva"ou "morta". (Também são usados os termos "ligado"e "desligado".) O estado do tabuleiro evolui e se modifica em pequenas passagens de tempo. Os estado de todas as células em um instante são considerados para calcular o estado de todas as células no instante seguinte. Todas as células são atualizadas simultaneamente. As transições dependem apenas do número de vizinhos vivos (ver as regras acima).

A idéia básica do "jogo"é começar com uma configuração simples de células vivas (organismos) que são colocadas em um tabuleiro 2D de vários métodos. Isto constitui a primeira geração. As "leis genéticas"de Conway para nascimentos, mortes e sobrevivência (as quatro regras acima) são então aplicadas e a nova geração é então colocada de acordo. Geração a geração os "jogador(es)"observam as várias imagens que surgem.

A vida sempre segue De uma imagem inicial de células vivas no tabuleiro, percebe-se que, conforme passam as gerações, a população anda constantemente de modo não usual, algumas vezes belo e quase sempre inesperado, porém muda. Em muitos poucos casos a sociedade eventualmente morre (todas as células se tornam células mortas), apesar de que isso pode não acontecer até que ocorram uma série de novas gerações. A maioria das imagens iniciais chegam a figuras estáveis - Conway chama-as de "vida parada- que não podem mudar ou imagens que oscilam para sempre. Imagens sem simetria inicial tendem a se tornar simétricas. Uma vez que isto ocorre, a simetria não pode mais ser perdida, apesar de poder crescer em riqueza.

Conway originalmente supôs que nenhuma imagem poderia crescer ilimitadamente. Em outras palavras, qualquer configuração com um número finito de células não pode crescer além de um limite finito superior ao número de células do campo. Esta foi provavelmente a mais profunda e mais difícil questão colocada pelo jogo atualmente. Conway ofereceu um prêmio de $50 à primeira pessoa que pudesse provar a afirmação

ou seu contrário antes do fim de 1970. Um caminho para provar seu contrário era ser capaz de descobrir imagens que continuassem adicionando contadores ao campo: Uma "gun"(uma configuração que constantemente atira objetos que se movimentam, como o "glider") ou um "puffer train"(uma configuração que se move mas deixa atrás uma trilha de "fumaça"). O prêmio foi ganho em novembro do mesmo ano por um time do MIT, a configuração inicial (mostrada no topo da página) cresce como um gun, emitindo seu primeiro glider na 40a geração. A gun emite então um novo glider a cada 30 gerações.

Imagens Existe uma série de diferentes imagens que podem ocorrer no Jogo da Vida, incluindo imagens paradas ("vidas paradas"), imagens repetitivas ("osciladores- o conjunto de vidas paradas), e imagens que se movimentam através do tabuleiro ("naves espaciais").

Imagens chamadas "Matusalens"podem evoluir por longos períodos de tempo antes de se repetir. "Diehard"é uma imagem que eventualmente desaparece após 130 gerações, ou passos. "Acorn"leva 5206 gerações para criar 13 gliders depois se estabiliza como vários osciladores.

Desde então, uma série de construções complicadas têm sido feitas, incluindo portas lógicas de gliders, um somador, um gerador de números primos e uma unidade de célula que emula o Jogo da Vida em uma escala muito maior e a passos lentos.

É possível construir portas lógicas AND, OR e NOT usando gliders. É possível construir uma imagem que aja como uma máquina de estado finito conectada a dois contadores. Isto possui o mesmo poder computacional de uma máquina de Turing universal (veja contador para a prova), Jogo da Vida é tão poderoso quanto qualquer computador com memória ilimitada: é o Turing completo. Além disso, uma imagem pode conter um conjunto de guns que se combina para construir novos objetos, incluindo cópias da imagem original. Um "construtor universal"pode ser construído que contenha um computador Turing completo, e que pode construir uma série de objetos complexos, incluindo mais cópias de si mesma.

Para este exercício considere que o tabuleiro está rodeado por células mortas.

Filosofia O Jogo da Vida de Conway cria um universo complicado a partir de poucas regras. É imaginável que todo o universo real seja um jogo similar, jogado em alguma dimensão mais alta. Esta possibilidade pode influenciar o debate de vivermos em um universo perfeitamente ajustado que requer um projeto inteligente. (texto fortemente baseado em pr.wikipedia.org)

12.2 Exercício 1

Seja o seguinte tabuleiro inicial, onde '.' significa célula morta e 'X' significa célula viva.

```
.  .  X  .  X  .  .  .  .  X
.  .  .  .  X  X  .  .  .  .
.  .  X  .  X  .  .  .  X  .
X  X  .  .  X  .  .  X  .  X
.  .  .  .  X  .  .  .  .  .
X  .  X  .  .  .  .  X  X  .
.  X  .  .  .  .  .  .  .  .
.  X  .  .  X  X  .  .  .  .
.  .  .  X  .  X  X  .  .  .
.  .  .  .  .  .  .  .  X  .
```


Informe a seguir se a posição pedida, após **4** gerações, está viva ou morta. Indique que ela está viva escrevendo **1** e que está morta escrevendo **0**.

L= 7 , C= 1	L= 7 , C= 2	L= 7 , C= 3	L= 7 , C= 6

12.3 Exercício 2

Seja o seguinte tabuleiro inicial, onde '.' significa célula morta e 'X' significa célula viva.

```
.  .  .  .  X  .  X  X  X  .
.  X  X  .  X  .  X  X  .  .
.  X  .  .  .  X  .  .  X  .
.  X  X  .  .  .  .  .  .  .
.  .  .  .  .  .  .  X  .  X
.  .  .  .  .  .  .  .  .  X
X  X  .  .  .  X  .  .  .  .
X  .  .  .  .  .  .  .  .  .
.  X  .  .  .  .  .  .  X  X
X  .  .  .  .  .  .  .  .  .
```

Informe a seguir se a posição pedida, após **4** gerações, está viva ou morta. Indique que ela está viva escrevendo **1** e que está morta escrevendo **0**.

L= 9 , C= 2	L= 2 , C= 2	L= 2 , C= 5	L= 3 , C= 1

12.4 Exercício 3

Seja o seguinte tabuleiro inicial, onde '.' significa célula morta e 'X' significa célula viva.

```
.  .  .  .  X  X  .  .  X  .
X  .  .  .  .  .  .  .  .  .
X  X  .  .  .  .  X  .  .  .
.  .  .  X  .  .  .  .  X  X
.  .  X  .  .  .  X  .  X  X
.  .  .  .  X  .  .  X  .  X
.  X  .  .  .  .  .  .  X  X
.  X  .  .  .  .  .  X  .  .
.  .  X  X  .  .  X  .  .  .
X  X  X  .  .  .  .  .  .  .
```

Informe a seguir se a posição pedida, após **4** gerações, está viva ou morta. Indique que ela está viva escrevendo **1** e que está morta escrevendo **0**.

L= 8 , C= 8	L= 8 , C= 9	L= 9 , C= 4	L= 10 , C= 2

12.5 Exercício 4

Seja o seguinte tabuleiro inicial, onde '.' significa célula morta e 'X' significa célula viva.

```
.  X  .  .  .  .  X  .  .  .
.  .  .  .  X  X  X  .  X  .
.  .  .  .  .  .  .  X  X  X
.  X  .  X  .  .  .  X  X  .
.  .  .  .  .  .  .  .  X  X
.  .  .  X  .  .  .  .  .  .
.  .  .  X  .  X  .  .  .  .
.  .  .  X  .  X  X  .  X  .
.  .  .  .  X  X  .  .  .  .
.  X  .  .  .  .  .  X  .  .
```

Informe a seguir se a posição pedida, após **4** gerações, está viva ou morta. Indique que ela está viva escrevendo **1** e que está morta escrevendo **0**.

L= 9 , C= 5	L= 9 , C= 6	L= 9 , C= 7	L= 1 , C= 6

12.6 Respostas

```
1      1 0 1 1
2      0 0 1 0
3      1 0 1 1
4      1 1 1 1
```


Capítulo 13

Exercício Prático:007-GPS

- 1 Sistema de Posicionamento Global, vulgarmente conhecido por GPS (do acrónimo do inglês Global Positioning System), é um sistema de posicionamento por satélite, por vezes incorrectamente designado de sistema de navegação, utilizado para determinação da posição de um receptor na superfície da Terra ou em órbita.

O sistema GPS foi criado e é controlado pelo Departamento de Defesa dos Estados Unidos da América, DoD, e pode ser utilizado por qualquer pessoa, gratuitamente, necessitando apenas de um receptor que capte o sinal emitido pelos satélites. O DoD fornece dois tipos de serviços GPS: Standard e Precision. Contrariamente ao que inicialmente acontecia, atualmente os dois serviços estão disponíveis em regime aberto em qualquer parte do mundo. O sistema está dividido em três partes: espacial, de controle e usuário. O segmento espacial é composto pela constelação de satélites. O segmento de controle é formado pelas estações terrestres dispersas pelo mundo ao longo da Zona Equatorial, responsáveis pela monitorização das órbitas dos satélites, sincronização dos relógios atômicos de bordo dos satélites e atualização dos dados de almanaque que os satélites transmitem. O segmento do usuário consiste num receptor que capta os sinais emitidos pelos satélites. Um receptor GPS (GPSR) decodifica as transmissões do sinal de código e fase de múltiplos satélites e calcula a sua posição com base nas distâncias a estes. A posição é dada por latitude, longitude e altitude.

Descrição técnica Receptores GPS vêm numa variedade de formatos, de dispositivos integrados dentro de carros, telefones, e relógios, a dispositivos dedicados somente ao GPS como estes das marcas Trimble, Garmin e Leica.

O sistema foi declarado totalmente operacional apenas em 1995. Seu desenvolvimento custou 10 bilhões de dólares. Consiste numa "constelação"de 28 satélites sendo 4 sobressalentes em 6 planos orbitais. Os satélites GPS, construídos pela empresa Rockwell, foram lançados entre Fevereiro de 1978 (Bloco I), e 6 de Novembro de 2004 (o 29º). Cada um circunda a Terra duas vezes por dia a uma altitude de 20200 quilômetros (12600 milhas) e a uma velocidade de 11265 quilômetros por hora (7000 milhas por hora). Os satélites têm a bordo relógios atómicos e constantemente difundem o tempo preciso de acordo com o seu próprio relógio, junto com informação adicional como os elementos orbitais de movimento, tal como determinado por um conjunto de estações de observação terrestres.

Medição com um GPS O receptor não necessita ter um relógio de tão grande precisão, mas sim de um suficientemente estável. O receptor capta os sinais de quatro satélites para determinar as suas próprias coordenadas, e ainda o tempo. Então, o receptor calcula a distância a cada um dos quatro satélites pelo intervalo de tempo entre o instante local e o instante em que os sinais foram enviados (esta distância é chamada pseudodistância).

Decodificando as localizações dos satélites a partir dos sinais de microondas (tipo de onda electromagnética) e de uma base de dados interna, e sabendo a velocidade de propagação do sinal, o receptor, pode situar-se na intersecção de quatro esferas, uma para cada satélite.

Coordenadas com um GPS com Bússula e Altímetro integrado Além de sua aplicação óbvia na aviação geral e comercial e na navegação marítima, qualquer pessoa que queira saber a sua posição, encontrar o seu caminho para determinado local (ou de volta ao ponto de partida), conhecer a velocidade e direcção do seu deslocamento pode-se beneficiar com o sistema. Atualmente o sistema está sendo muito difundido em automóveis com sistema de navegação de mapas, que possibilita uma visão geral da área que você está percorrendo.

A comunidade científica utiliza-o pelo seu relógio altamente preciso. Durante experiências científicas de captura de dados, pode-se registrar com precisão de micro-segundos (0,000001 segundo) quando a amostra foi obtida. Naturalmente a localização do ponto onde a amostra foi recolhida também pode ser importante. Agrimensores diminuem custos e obtêm levantamentos precisos mais rapidamente com o GPS. Unidades específicas têm custo aproximado de 3.000 dólares e precisão de 1 metro, mas existem receptores mais caros com precisão de 1 centímetro. A captura de dados por estes receptores é mais lenta.

Guardas florestais, trabalhos de prospecção e exploração de recursos naturais, geólogos, arqueólogos, bombeiros, são enormemente beneficiados pela tecnologia do sistema. O GPS tem-se tornado cada vez mais popular entre ciclistas, balonistas, pescadores, ecoturistas, geocachers (uma diversão bem moderna: uma caixa lacrada cheia de pinduricalhos e de um livro é escondida em algum lugar no planeta e suas coordenadas publicadas. Veja mais em http://www.geocaching.com), ou por leigos que queiram apenas orientação durante as suas viagens. Com a popularização do GPS, um novo conceito surgiu na agricultura: a agricultura de precisão. Uma máquina agrícola dotada de receptor GPS armazena dados relativos à produtividade em um cartão magnético que, tratados por programa específico, produz um mapa de produtividade da lavoura. As informações permitem também otimizar a aplicação de corretivos e fertilizantes.

Tipos de receptores Existem diferentes receptores GPS, desde diversas marcas que comercializam soluções "tudo-em-um", até os externos que são ligados por cabo ou ainda por bluetooth. Geralmente categorizados em termos de demandas de uso em Geodésicos, Topográficos e de Navegação. A diferenciação entre essas categorias, que a princípio pode parecer meramente de preço de aquisição é principalmente devido à precisão alcançada, ou seja a razão da igualdade entre o dado real do posicionamento, e o oferecido pelo equipamento. Sendo os mais acurados, com valores na casa dos milímetros, os receptores Geodésicos são capazes de captar as duas freqüências emitidas pelos satélites (L1 e L2), possibilitando assim a eliminação dos efeitos da refração ionosférica. Os topográficos, que tem características de trabalho semelhantes à categoria anterior, porém somente captam a portadora L1, também possuem elevada precisão, geralmente na casa dos centímetros. Ambas as categorias tem aplicações técnicas, e características próprias como o pós-processamento, o que significa que geralmente não informam o posicionamento instantaneamente.

No caso da categoria de maior uso, a de navegação, embora possua menor precisão de posicionamento, tem inúmeras vantagens como o baixo preço de aquisição e inúmeras aplicações, onde vê-se uma infinidade de modelos, tanto aqueles que integram diversos equipamentos como computadores de mão, celulares, relógios, etc., como aqueles dedicados exclusivamente ao posicionamento GPS, onde também encontramos aplicações para uso do dado de posicionamento em outros equipamentos como notebooks, rastreadores de veículos, etc.

Dia 3 de setembro de 1989 Maracanã lotado para assistir ao Brasil X Chile, pelas eliminatórias da Copa do Mundo de 90. Longe dali, em algum ponto a princípio entre Marabá e Belém, Cezar Augusto Garcez comanda um vôo cego. Perdido em pleno ar, tenta se posicionar. Localiza, em vão, uma rádio que transmitia a partida que entraria para a história como "o jogo da fogueteira". Aquele vôo também estaria nos jornais no dia seguinte: "Avião desaparece na Amazônia", publicou O Globo.

Herói e vilão, ao mesmo tempo, o comandante Garcez é a principal personagem do RG-254 que caiu na selva amazônica em 1989. O que deveria ser um vôo rotineiro se transformou numa tragédia. Desorientado, Garcez permaneceu durante três horas em vôo cego. Temendo que o erro fosse descoberto, passou diversas informações truncadas para a base, afirmou estar onde não estava. Sem combustível, arriscou o aparentemente impossível: um pouso na copa das árvores, em plena noite, com visibilidade praticamente nula. Garcez foi acusado de negligenciar rotinas básicas da aviação. Por outro lado, salvou a vida de muitos passageiros ao conseguir aterrissar a aeronave e cuidar dos feridos. Ainda hoje, aguarda julgamento.

13.1 GPS

Na década de 70, os EUA começaram a projetar e implementar o sistema conhecido como GPD (Global Positioning System). O primeiro satélite *Navstar* foi lançado em 1978. O objetivo principal era permitir que os estimados 40.000 usuários militares americanos se orientassem sobre a terra. Os civis começaram a usar o sistema a partir da década de 80. Hoje, estima-se em mais de 50.000.000 de dispositivos capazes de receber os sinais de GPS e apresentar uma estimativa de localização.

Além do sistema americano, existe o russo *Glonass* de uso exclusivo pelos militares e a partir da última semana de 2005, o sistema *Galileu* construído e operado pela Agência Espacial Européia.

A grande explosão de uso, deverá vir quando celulares e automóveis começarem a vir de fábrica com a possibilidade de informar sua localização. Associados a sistemas de mapas embarcados, poderão informar ao motorista qual caminho tomar para ir virtualmente a qualquer lugar sobre a superfície terrestre. Associados a telefones celulares, poderão informar a localização do telefone. Associados a sistemas de proteção contra roubo de automóveis, informarão a localização em tempo real e assim por diante. A lista de aplicações possíveis parece inesgotável.

Todos os sistemas acima se baseiam na triangulação (ou quadrangulação) do receptor em relação a 3 ou 4 (ou mais) satélites em órbita da terra.

O sistema começa pela identificação de diversos pontos fixos sobre a superfície terrestre. Tais pontos contam com estações transmissoras e permitem a cada satélite a recepção de diversos pontos. Medindo o tempo gasto por cada sinal desde a estação terrena até o satélite, e fazendo isso com diversas estações, o satélite conhece sua real posição com precisão de centímetros.

Cada satélite possui um relógio atômico, integrado à rede GPS e que garante precisão de tempo de 1 bilionésimo de segundo.

Todos os satélites emitem 2 mensagens: A primeira, envia a identificação do satélite, sua posição no espaço e sua hora interna. Uma segunda mensagem é uma seqüência de impulsos digitais em um padrão inconfundível.

Tudo funciona como se o receptor também gerasse no mesmo instante de tempo a mesma seqüência de impulsos digitais. Da análise dos dois sinais (o gerado pelo satélite e o gerado internamente no receptor) é possível medir o retardo do sinal do satélite. Conhecendo a velocidade da luz (fixa e igual a 299.729.458 m/seg) e conhecendo o tempo gasto pelo sinal do satélite ao receptor, é possível estabelecer a distância entre satélite e receptor.

Daqui, pode-se inferir que o receptor está sobre a esfera de centro x, y, z (as coordenadas no espaço do satélite) e de raio R, que é a distância entre o receptor e o satélite.

Ao fazer o mesmo cálculo com um segundo satélite, obtem-se uma segunda esfera, e o receptor está sobre o círculo de intersecção das duas esferas.

Finalmente ao fazer o cálculo com um terceiro satélite, obtêm-se uma terceira esfera. O receptor ainda pertence às 3 esferas, e a intersecção das 3 apresenta apenas 2 pontos no espaço. Um deles apenas sobre a superfície terrestre, e logo esta é a posição do receptor, com precisão de metros.

13.1.1 Problemas

Se os receptores pudessem ter um relógio atômico 3 satélites bastariam. Como um relógio atômico é muito caro e pesado, os receptores usam relógios de quartzo comum (sujeitos a erros de 1 seg/dia).

Então usa-se um quarto satélite para refazer os cálculos e permitir acertar o atraso/adiantamento do relógio local ao relógio da rede.

Se o usuário estiver em movimento, os sinais ainda vêm afetados pelo efeito Doppler, que precisa ser analiticamente corrigido antes de estabelecer a posição do receptor. Idem para os efeitos relativísticos associados.

O sistema americano usa duas radiofrequencias: L1 e L2. A L1 é chamada Sinal Civil, embora os militares também a usem. A L2 é de uso exclusivo militar, já que seus códigos (a segunda parte das mensagens) não são públicos e são protegidos por criptografia. Ao usar apenas o sinal L1, receptores conseguem determinar sua posição com erros de 5 a 10 m, causados pela ação da ionosfera. A camada de ar sobre a terra aumenta a densidade da atmosfera, causando refração nos sinais. É como um lápis colocado dentro de um copo de água.

A potência de um satélite é de 500W, o equivalente a 5 lâmpadas de 100W. Apenas para efeito de comparação, o sinal de TV de um satélite é 1 bilhão de vezes mais forte. Este fato permite que interferências (às vezes propositais) obscureçam ou até invalidem o sinal dos satélites. Este fato é usado em teatros de guerra, quando as forças americanas introduzem geradores locais de ruído centrados na banda usada pelo sistema. Essa é a razão pela qual embora o sistema seja público, russos e europeus tratam de ter cada um o seu.

13.2 Dois amigos

Vamos simular o cálculo de uma posição desconhecida usando triangulação, mas vamos trabalhar apenas em 2D, que ninguém aqui está a fim de enlouquecer por causa da geometria analítica. ;-)

Suponhamos uma região plana, na qual dois pontos tem coordenadas conhecidas. Você vai colocar 2 amigos, um em cada ponto, todos munidos de relógios absolutamente acertados até o milésimo do segundo, e completamente precisos.

Você combinou que às 14h00, um dos seus amigos dará um tiro de espoleta. Nessa mesma hora, você ligará um cronômetro. Alguns décimos de segundos depois, você ouvirá o tiro e desligará o cronômetro, obtendo um tempo gasto pelo barulho para percorrer a distância entre o seu primeiro amigo e você. Vamos chamar esta distância de D_1.

Lembrando que o som se propaga a 340m/seg, você conseguirá uma medida confiável entre você e seu amigo 1.

O procedimento será repetido às 14h15 com seu amigo número 2. Pelo mesmo mecanismo você obterá a distância entre você e seu amigo 2. Será a distância D_2.

Sabendo que seu amigo 1, se encontra no centro de um círculo de raio D_1 e coordenadas do centro iguais a x_1, y_1, e seu amigo 2 se encontra em x_2, y_2, também no centro de um círculo de raio D_2, e que você se encontra sobre os dois círculos, pede-se que você calcule a sua posição.

Obs: se você não tiver muita intimidade com a geometria análitica, um bom método é o gráfico. Use um papel milimetrado, disponha os dois pontos, trace duas circunferências de raios D_1 e D_2 e nos encontros das duas circunferências você estará.

Obs:

- Este procedimento também pode ser usado para verificar o acerto/erro das contas e a verificação final do ponto.
- Como só usaremos 2 amigos (ao invés de 3), haverá 2 pontos candidatos, ambos na superfície da terra. Responda os 2.

13.2.1 Exemplo

Seja um caso em que $x_1 = 264m$, $y_1 = 141m$. $x_2 = 520m$, $y_2 = 291m$. Os tempos são: $t_1 = 0,4749seg$ e $t_2 = 0,4017seg$.

Obtem-se $d_1 = 161,48m$ e $d_2 = 136,6m$. A fórmula do circulo 1 é $x^2 + y^2 - 528x - 282y + 63501.21 = 0$

A fórmula do circulo 2 é $x^2 + y^2 - 1040x - 582y + 336421 = 0$

Subtraindo uma equação da outra, fica-se com $512x + 300y - 272920 = 0$ e

$$y = \frac{272920 - 512x}{300} = 909.7 - 1.7x$$

Substituindo o y obtido em qualquer uma das equações, chega-se ao valor correto do observador que é $x_o = 410m$, $y_o = 210m$.

13.2.2 Como fazer

Para entender o funcionamento do sistema, é necessário relembrar um pouco de física e de geometria análitica.

Da física: distância = velocidade $\times$ tempo. Sabendo que a velocidade do som é 340 m/seg, obtêm-se as duas distâncias.

Da geometria analítica, um círculo no plano é perfeitamente determinada se soubermos as coordenadas α, β do seu centro, bem como o raio dele, usualmente representada por R. Da geometria analítica, a fórmula é

$$(x - \alpha)^2 + (y - \beta)^2 = R^2$$

Do problema, são conhecidos $\alpha_1, \alpha_2, \beta_1, \beta_2, D_1$ e D_2. Obtem-se agora duas fórmulas em x^2, x, y^2 e y. Operando com elas obtem-se dois pares de x, y que são os pontos candidatos.

13.2.3 Problema 1

Dados:

- Seu amigo 1 está em $x_1 = 253$ e $y_1 = 169$.
- Seu amigo 2 está em $x_2 = 671$ e $y_2 = 225$.
- O tiro do amigo 1 demorou $t_1 = .675453$ segundos. O tiro do amigo 2 demorou $t_2 = .568409$ segundos.

Os dois pontos candidatos a serem sua localização são:

x,y	x,y

13.2.4 Problema 2

Dados:

- Seu amigo 1 está em $x_1 = 392$ e $y_1 = 173$.
- Seu amigo 2 está em $x_2 = 696$ e $y_2 = 210$.
- O tiro do amigo 1 demorou $t_1 = .395137$ segundos. O tiro do amigo 2 demorou $t_2 = .507112$ segundos.

Os dois pontos candidatos a serem sua localização são:

x,y	x,y

13.2.5 Problema 3

Dados:

- Seu amigo 1 está em $x_1 = 283$ e $y_1 = 130$.
- Seu amigo 2 está em $x_2 = 507$ e $y_2 = 275$.
- O tiro do amigo 1 demorou $t_1 = .449615$ segundos. O tiro do amigo 2 demorou $t_2 = .335990$ segundos.

Os dois pontos candidatos a serem sua localização são:

x,y	x,y

13.2.6 Problema 4

Dados:

- Seu amigo 1 está em $x_1 = 310$ e $y_1 = 170$.
- Seu amigo 2 está em $x_2 = 679$ e $y_2 = 256$.
- O tiro do amigo 1 demorou $t_1 = .614903$ segundos. O tiro do amigo 2 demorou $t_2 = .499585$ segundos.

Os dois pontos candidatos a serem sua localização são:

x,y	x,y

13.2.7 Problema 5

Dados:

- Seu amigo 1 está em $x_1 = 201$ e $y_1 = 163$.
- Seu amigo 2 está em $x_2 = 627$ e $y_2 = 251$.
- O tiro do amigo 1 demorou $t_1 = .593287$ segundos. O tiro do amigo 2 demorou $t_2 = .686965$ segundos.

Os dois pontos candidatos a serem sua localização são:

x,y	x,y

13.2.8 Problema 6

Dados:

- Seu amigo 1 está em $x_1 = 288$ e $y_1 = 109$.
- Seu amigo 2 está em $x_2 = 619$ e $y_2 = 205$.
- O tiro do amigo 1 demorou $t_1 = .443191$ segundos. O tiro do amigo 2 demorou $t_2 = .570475$ segundos.

Os dois pontos candidatos a serem sua localização são:

x,y	x,y

13.2.9 Problema 7

Dados:

- Seu amigo 1 está em $x_1 = 308$ e $y_1 = 157$.
- Seu amigo 2 está em $x_2 = 561$ e $y_2 = 204$.
- O tiro do amigo 1 demorou $t_1 = .330474$ segundos. O tiro do amigo 2 demorou $t_2 = .429059$ segundos.

Os dois pontos candidatos a serem sua localização são:

x,y	x,y

13.2.10 Problema 8

Dados:

- Seu amigo 1 está em $x_1 = 274$ e $y_1 = 105$.
- Seu amigo 2 está em $x_2 = 511$ e $y_2 = 264$.
- O tiro do amigo 1 demorou $t_1 = .473127$ segundos. O tiro do amigo 2 demorou $t_2 = .366280$ segundos.

Os dois pontos candidatos a serem sua localização são:

x,y	x,y

13.2.11 Problema 9

Dados:

- Seu amigo 1 está em $x_1 = 313$ e $y_1 = 195$.
- Seu amigo 2 está em $x_2 = 506$ e $y_2 = 279$.
- O tiro do amigo 1 demorou $t_1 = .367376$ segundos. O tiro do amigo 2 demorou $t_2 = .256153$ segundos.

Os dois pontos candidatos a serem sua localização são:

x,y	x,y

13.2.12 Problema 10

Dados:

- Seu amigo 1 está em $x_1 = 329$ e $y_1 = 145$.
- Seu amigo 2 está em $x_2 = 519$ e $y_2 = 219$.
- O tiro do amigo 1 demorou $t_1 = .231924$ segundos. O tiro do amigo 2 demorou $t_2 = .372242$ segundos.

Os dois pontos candidatos a serem sua localização são:

x,y	x,y

13.2.13 Respostas

```
 1      478  215
 2      524  198
 3      408  218
 4      513  220
 5      400  196
 6      433  150
 7      416  188
 8      408  194
 9      422  256
10      406  162
```

Capítulo 14

Exercício Prático: 008 - Problemas

Quais os nomes válidos? Na lista de nomes a seguir, informe se cada nome é válido ou não.

nome	válido	inválido
R33		
33R		
SALDOBANCARIO		
JKGHHGGJHG		
OBA AQUI		
A+B		
ZERO/DOIS		

Tipos de variáveis Para representar as variáveis a seguir, qual o(s) tipo(s) mais adequado ? (Se houver mais de um tipo, indique com "1" o melhor, com "2" o seguinte e assim por diante).

nome do campo	int	real	lógico	carac	cadeia
nome					
salário					
número de filhos					
o sujeito tem filhos ?					
idade					
ano de nascimento					
categoria de motorista					
nome de uma vitamina					

14.1 Resolva os exercícios a seguir

Cada um dos exercícios a seguir apresenta uma resposta numérica única. Descubra-a e coloque o seu valor no quadrado duplo que segue a definição.

14.1.1 Problema 1

Um certo espetáculo musical no Japão, tem um custo para entrar de 3500 ienes para adultos e 1200 ienes para estudantes. No sábado passado havia 608 pessoas, e a renda da noite foi de 1.707.100 ienes. Quantos estudantes havia ?

14.1.2 Problema 2

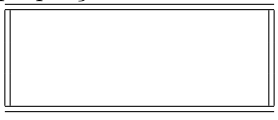
Em um tanque há 4000 bolinhas de pingue-pongue. Um menino começou a retirar as bolinhas, uma por uma, com velocidade constante quando eram 10h. Após 6 horas, há no tanque 3520 bolinhas. Se o menino continuasse no mesmo ritmo, a que horas do dia seguinte o tanque ficaria com 2000 bolinhas ? (OBM 2006)

14.1.3 Problema 3

Uma processadora de alimentos desenvolveu uma marca de salada de fruta enlatada de muito sucesso. A fórmula desta salada, envolve as seguintes frutas e ingredientes nas seguintes proporções: 1 Kg de banana; 0,5 Kg de abacaxi; 2 Kg de laranja e 250 gramas de açucar.

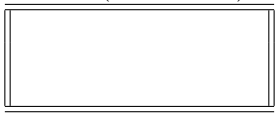
Suponha que em um determinado dia, existem disponíveis as seguintes quantidades no depósito: 100 Kg de açucar, 300 Kg de abacaxí, 800 Kg de bananas e 1 tonelada de laranja.

Pergunta-se quantos quilos de salada de fruta poderão ser produzidos preservando as proporções da fórmula original ?



14.1.4 Problema 4

A soma de dois números primos a e b é 34 e a soma dos primos a e c é 33. Quanto vale $a+b+c$? (OBM 2004)



14.1.5 Problema 5

Na multiplicação a seguir, a, b e c são algarismos:

```
    1 a b
      b 3  x
---------
    * * *
  * * *
---------
1 c c 0 1
```

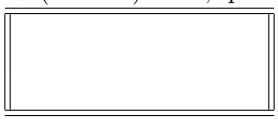
Qual o valor de $a+b+c$

14.1.6 Problema 6

O perímetro de determinado retângulo é 16 vezes maior do que o menor lado do retângulo. O comprimento do maior lado é 12 centímetros maior do que o menor lado. Qual o comprimento do maior lado, em centímetros?

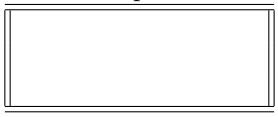
14.1.7 Problema 7

As letras O, B e M representam números inteiros. Se $O \times B \times M = 240$, $(O \times B) + M = 46$ e $O + (B \times M) = 64$, quanto vale $O + B + M$? (OBM 2005)



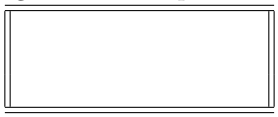
14.1.8 Problema 8

Na oficina mecânica do Zé Tião, nesta semana foram reparados 40 veículos, entre motos e carros. Como curiosidade, o auxiliar do Zé Tião contou que o número total de rodas dos veículos reparados foi 100 rodas. Pergunta-se quantas motos foram reparadas ?



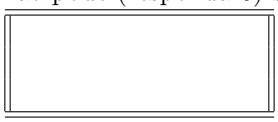
14.1.9 Problema 9

Um time de futebol ganhou 8 jogos a mais do que perdeu e empatou 3 jogos menos do que ganhou en 31 partidas jogadas. Quantas partidas o time ganhou ? (OBM 2006)



14.1.10 Problema 10

O correio de um determinado país só aceita transportar pacotes com peso inferior a 3 Kg e comprimento master menor que 250 centímetros. O comprimento master é o maior barbante que dá a volta completa ao pacote. Supondo um pacote de 2,85 Kg e cujas dimensões em metros são: 0,3 x 0,7 x 1,1 m, pergunta-se Este pacote pode (responda 1) ou não pode (responda 0) ser mandado pelo correio ?



14.1.11 Respostas

```
1  183.0
2  11.0
3  1500.0
4   36.0
5   10.0
```



```
6   14.0
7   20.0
8   30.0
9   14.0
10  0
```

Capítulo 15

Exercício prático: 009-Achar o número que falta

15.1 Qual o número que falta ?

Neste folha de exercícios, serão apresentadas diversas seqüências de 11 números. Os números iniciais (1, 2 ou 3) a depender da seqüência são aleatórios. Os próximos até o final seguem uma determinada lei de formação. Um dos números do final da seqüência é representado por um sinal de **?**. Você deve descobrir que número é esse

Exemplo Veja-se alguns casos

N.	1	2	3	4	5	6	7	8	9	10	11	resposta	some 3 em 3
1	8	12	21	37	62	98	147	211	292	?	513		
2	4	7	3	14	24	41	79	?	264	487	895	+	
3	8	2	15	22	42	69	116	?	311	506	822	+	=
4	5	7	12	19	31	50	81	131	212	?	555		
5	4	8	17	33	58	94	143	207	?	388	509	+	
6	9	13	22	38	63	99	148	212	293	?	514	+	=
7	1	4	5	9	14	23	37	60	?	157	254		
8	3	7	16	32	57	93	142	206	?	387	508	+	
9	6	9	24	57	138	333	804	1941	?	11313	27312	+	=
10	9	5	14	19	33	52	85	137	222	?	581		
11	8	1	7	16	24	47	87	158	292	?	987	+	
12	5	9	1	15	25	41	81	147	?	497	913	+	=
13	1	9	10	19	29	48	77	125	?	327	529		
14	5	4	9	13	22	35	57	?	149	241	390	+	
15	9	13	22	38	63	99	148	212	293	?	514	+	=

15.2 Exercício 1

N.	1	2	3	4	5	6	7	8	9	10	11	resposta	some 3 em 3
1	3	4	11	26	63	152	367	886	2139	?	12467		
2	6	10	19	35	60	96	145	?	290	390	511	+	
3	7	4	15	34	83	200	483	1166	2815	?	16407	+	=
4	3	7	16	32	57	93	142	206	?	387	508		
5	5	8	13	21	34	55	89	144	?	377	610	+	
6	9	4	4	17	25	46	88	159	293	?	992	+	=
7	8	8	24	56	136	328	792	?	4616	11144	26904		
8	7	5	14	21	37	60	99	?	262	425	689	+	
9	3	9	19	35	61	103	171	281	459	?	1213	+	=
10	6	4	16	26	48	80	134	?	360	586	952		
11	7	1	9	19	47	113	273	?	1591	3841	9273	+	
12	6	2	8	10	18	28	46	?	120	194	314	+	=
13	6	3	12	27	66	159	384	927	2238	?	13044		
14	1	9	19	47	113	273	659	1591	?	9273	22387	+	
15	4	5	18	32	59	100	168	277	?	740	1203	+	=
16	4	3	8	15	26	49	90	165	304	?	1028		
17	6	4	14	32	78	188	454	?	2646	6388	15422	+	
18	3	3	3	9	15	27	51	93	171	?	579	+	=
19	5	8	21	50	121	292	705	?	4109	9920	23949		
20	4	8	17	33	58	94	143	207	?	388	509	+	
21	5	2	7	9	16	25	41	66	?	173	280	+	=
22	8	12	21	37	62	98	147	211	292	?	513		
23	4	2	8	18	44	106	256	618	?	3602	8696	+	
24	4	7	18	43	104	251	606	?	3532	8527	20586	+	=
25	9	4	13	17	30	47	77	?	201	325	526		
26	4	5	9	18	32	59	109	200	368	?	1245	+	
27	1	5	14	30	55	91	140	?	285	385	506	+	=
28	7	1	15	17	47	81	175	?	687	1361	2735		
29	7	5	17	39	95	229	553	1335	3223	?	18785	+	
30	7	6	17	27	48	79	131	214	349	?	920	+	=

15.3 Exercício 2

N.	1	2	3	4	5	6	7	8	9	10	11	resposta	some 3 em 3
1	3	7	16	32	57	93	142	206	287	?	508		
2	5	9	18	34	59	95	144	?	289	389	510	+	
3	1	9	12	23	37	62	101	?	268	435	705	+	=
4	1	1	9	11	21	41	73	135	?	457	841		
5	7	11	20	36	61	97	146	?	291	391	512	+	
6	7	2	11	15	28	45	75	122	?	323	524	+	=
7	2	6	6	14	26	46	86	158	?	534	982		
8	7	11	20	36	61	97	146	210	291	?	512	+	
9	3	1	5	11	27	65	157	?	915	2209	5333	+	=
10	9	5	23	33	79	145	303	?	1199	2385	4783		
11	2	3	9	16	29	49	82	135	?	360	585	+	
12	8	9	17	26	43	69	112	?	293	474	767	+	=
13	9	13	22	38	63	99	148	212	293	?	514		
14	7	1	1	9	11	21	41	?	135	249	457	+	
15	9	5	19	43	105	253	611	1475	?	8597	20755	+	=
16	9	5	14	19	33	52	85	137	222	?	581		
17	4	7	15	29	59	117	235	?	939	1877	3755	+	
18	3	7	16	32	57	93	142	206	?	387	508	+	=
19	2	6	15	31	56	92	141	205	?	386	507		
20	7	5	8	20	33	61	114	208	383	?	1296	+	
21	4	3	13	22	41	69	116	191	313	?	829	+	=
22	5	2	9	20	49	118	285	?	1661	4010	9681		
23	4	8	19	34	60	101	168	?	451	734	1192	+	
24	7	11	20	36	61	97	146	210	?	391	512	+	=
25	7	11	20	36	61	97	146	210	291	?	512		
26	1	8	9	17	26	43	69	112	181	?	474	+	
27	7	2	11	24	59	142	343	828	1999	?	11651	+	=
28	3	2	7	11	20	33	55	?	147	239	388		
29	8	6	20	46	112	270	652	1574	3800	?	22148	+	
30	3	3	15	27	51	87	147	243	?	651	1059	+	=

15.4 Exercício 3

N.	1	2	3	4	5	6	7	8	9	10	11	resposta	some 3 em 3
1	6	2	10	22	54	130	314	758	1830	?	10666		
2	6	2	2	10	14	26	50	90	166	?	562	+	
3	7	7	21	35	77	147	301	?	1197	2387	4781	+	=
4	6	4	14	32	78	188	454	1096	2646	?	15422		
5	6	10	19	35	60	96	145	209	?	390	511	+	
6	2	8	1	11	20	32	63	115	210	?	713	+	=
7	3	8	12	21	34	56	91	148	240	?	630		
8	5	6	16	28	60	116	236	468	?	1876	3756	+	
9	8	6	20	46	112	270	652	1574	3800	?	22148	+	=
10	2	6	8	14	22	36	58	94	152	?	398		
11	1	5	14	27	49	84	141	233	?	623	1013	+	
12	3	1	4	8	13	25	46	84	155	?	524	+	=
13	8	5	17	26	47	77	128	209	?	554	899		
14	2	9	1	12	22	35	69	126	230	?	781	+	
15	5	2	12	19	36	60	101	?	272	443	720	+	=
16	4	6	1	11	18	30	59	107	196	?	665		
17	8	6	22	34	78	146	302	?	1198	2386	4782	+	
18	7	7	14	21	35	56	91	?	238	385	623	+	=
19	5	3	4	12	19	35	66	?	221	407	748		
20	7	1	12	17	33	54	91	149	?	397	645	+	
21	1	2	4	8	16	32	64	?	256	512	1024	+	=
22	5	7	9	21	37	67	125	229	421	?	1425		
23	2	4	14	26	48	82	138	228	?	610	992	+	
24	2	6	15	31	56	92	141	205	286	?	507	+	=
25	4	4	12	20	44	84	172	340	684	?	2732		
26	9	2	13	28	69	166	401	968	2337	?	13621	+	
27	5	4	10	15	26	42	69	112	?	295	478	+	=
28	2	7	9	16	25	41	66	?	173	280	453		
29	8	12	21	37	62	98	147	?	292	392	513	+	
30	5	9	18	34	59	95	144	?	289	389	510	+	=

15.5 Exercício 4

N.	1	2	3	4	5	6	7	8	9	10	11	resposta	some 3 em 3
1	8	7	18	28	49	80	132	215	?	568	921		
2	2	5	9	19	37	75	149	299	597	?	2389	+	
3	5	8	21	50	121	292	705	1702	?	9920	23949	+	=
4	7	11	20	36	61	97	146	210	?	391	512		
5	4	8	17	33	58	94	143	207	?	388	509	+	
6	2	9	11	20	31	51	82	133	?	348	563	+	=
7	9	9	21	33	57	93	153	249	405	?	1065		
8	6	2	10	14	26	42	70	114	?	302	490	+	
9	1	9	14	27	45	76	125	?	334	543	881	+	=
10	9	3	9	21	33	63	117	213	?	723	1329		
11	6	5	17	27	61	115	237	467	?	1875	3757	+	
12	4	8	17	33	58	94	143	207	?	388	509	+	=
13	7	11	20	36	61	97	146	210	291	?	512		
14	1	6	8	20	36	76	148	?	596	1196	2388	+	
15	2	6	15	31	56	92	141	205	?	386	507	+	=
16	9	7	4	20	31	55	106	?	353	651	1196		
17	5	2	12	16	40	72	152	296	?	1192	2392	+	
18	9	4	13	17	30	47	77	124	?	325	526	+	=
19	1	6	7	14	27	48	89	?	301	554	1019		
20	6	6	9	21	36	66	123	225	414	?	1401	+	
21	9	7	25	39	89	167	345	679	1369	?	5465	+	=
22	2	5	10	18	31	52	86	?	230	374	607		
23	4	7	15	29	59	117	235	469	?	1877	3755	+	
24	5	5	18	31	57	96	161	265	434	?	1149	+	=
25	7	2	9	11	20	31	51	?	133	215	348		
26	5	8	16	27	46	76	125	204	?	539	874	+	
27	7	11	20	36	61	97	146	?	291	391	512	+	=
28	7	11	20	36	61	97	146	210	291	?	512		
29	2	6	15	31	56	92	141	205	?	386	507	+	
30	4	6	16	38	92	222	536	1294	3124	?	18208	+	=

15.6 Respostas

```
 1 12169  1060  2820   953  9698  1970  2097  3347  1005  8685
 2   760   658  1060   995  4027  1115  1501  1255  5510  9663
 3  5319  7066 10503   913   932  1103   492  1535  7188   526
 4  5654   794  1048  1622   977   993  3653  1787   624  8219
```


Capítulo 16

Exercícios Práticos: 011 - Matemática e Lógica Básicas

16.1 Exercício 1

Note que a operação x mod y, quando $y = 0$, matematicamente não está definida. Entretanto, para esta folha, combinar-se-á que $x \; mod \; y = x$ quando $y = 0$. Da mesma maneira, $x \div 0$ não está definido. O caso particular, válido apenas aqui é $0 \div 0$ que aqui dará 1.

16.1.1 Exercício 1.1

Resolva as 6 linhas abaixo e some (aritméticamente) os resultados encontrados, lançando o total no quadro abaixo

- ((27 div 7) mod 2)+(⌊(((9 + 4) - (3 - 5)) ÷ (3 + 6)))
- ((26 div 5) mod 3)-(((8 + 3) × (⌊(9 ÷ 7))) + (9 + 3))
- (((4 + 7) - (5 × 5)) + (7 - 2))-(6 + 8)
- ((27 div 3) div 5)+(((8 + 3) + (3 × 3)) × (8 - 4))
- ((30 div 6) div 3)-(((3 × 4) + (9 + 2)) × (8 - 3))
- (2 - 3)+(((3 - 4) - (7 + 5)) + (7 × 3))

soma dos valores

16.1.2 Exercício 1.2

Resolva as 6 linhas abaixo e some (aritméticamente) os números das linhas que respondem VERDADEIRO, lançando o total no quadro abaixo

1 ((29 div 4) div 6)<(((7 + 7) + (3 + 9)) × (6 - 4))

2 (21 div 1)>(((6 - 5) × (5 + 2)) + (5 + 4))

4 (∼(.F. ∨ .V.))∧((∼(.F. ∧ .F.)) ∨ .F.)

8 (28 div 10)>(⌊(((8 + 7) × (9 × 8)) ÷ (6 + 6)))

16 (15 mod 10)>(((6 × 7) × (5 - 6)) - (5 + 3))

32 (∼(.V. ∨ .F.))∧((.V. ∨ .V.) ∨ .F.)

soma das linhas

16.1.3 Exercício 1.3

Resolva as 6 linhas abaixo e some (aritméticamente) os resultados encontrados, lançando o total no quadro abaixo

- (((14 mod 9) div 1) div (19 mod 8))-(((3 - 7) + (8 - 9)) + (2 - 8))
- (((21 mod 9) mod 5) div (16 mod 7))+(((4 - 7) + (4 + 4)) + (6 + 8))
- (6 × 9)+((⌊((9 + 7) ÷ (2 + 5))) + (8 + 4))
- (((30 div 10) mod 5) mod (16 div 6))-(((6 + 4) + (8 - 7)) - (8 - 3))
- (((24 div 7) mod 6) mod (22 div 3))+(((5 + 2) + (3 + 3)) × (3 + 5))
- (9 + 2)+(((8 + 8) × (3 - 2)) - (5 + 2))

soma dos valores

16.1.4 Exercício 1.4

Resolva as 6 linhas abaixo e some (aritméticamente) os números das linhas que respondem VERDADEIRO, lançando o total no quadro abaixo

1 ((26 div 5) mod 4)>(((5 - 3) + (4 + 5)) + (6 + 2))

2 (23 mod 10)>(((6 × 8) - (6 - 9)) - (5 - 4))

4 (.F. ∨ .V.)∨((.V. ∨ .V.) ∧ .F.)

8 (20 div 6)<(⌊(((3 + 2) + (2 × 5)) ÷ (4 + 8)))

16 (18 mod 4)<(((4 - 2) + (8 - 2)) × (2 - 7))

32 (.F. ∧ .V.)∨((.F. ∧ .F.) ∧ .F.)

soma das linhas

16.1.5 Exercício 1.5

Resolva as 6 linhas abaixo e some (aritméticamente) os resultados encontrados, lançando o total no quadro abaixo

- (((18 div 9) div 1) div (11 div 6))-(((3 + 7) × (2 + 3)) - (7 - 2))
- ((26 div 7) div 1)-(((4 × 9) - (9 - 5)) × (7 - 6))
- (((5 + 9) - (6 - 4)) - (6 - 9))-(6 × 9)
- (((11 mod 4) mod 5) div (25 mod 6))-(((3 - 7) + (8 × 6)) + (4 - 2))

- ((19 mod 3) mod 4)-(((8 - 3) × (4 × 2)) + (5 × 4))
- (($\lfloor$((7 × 3) ÷ (2 + 8))) × (8 + 9))-(2 + 3)

soma dos valores

16.1.6 Exercício 1.6

Resolva as 6 linhas abaixo e some (aritméticamente) os números das linhas que respondem VERDADEIRO, lançando o total no quadro abaixo

1 ((26 mod 7) mod 6)>(((7 - 6) + (7 × 9)) - (5 + 5))

2 (18 div 10)>($\lfloor$(((9 × 8) + (7 + 7)) ÷ (2 × 6)))

4 (.F. ∨ .F.)∨((.F. ∧ .V.) ∨ (∼.V.))

8 (27 div 8)>(((9 - 2) × ($\lfloor$(8 ÷ 7))) - (7 + 8))

16 (13 mod 7)>((($\lfloor$(7 ÷ 4)) + (7 + 8)) + (3 × 8))

32 (.V. ∧ .F.)∨((.F. ∧ .V.) ∧ .F.)

soma das linhas

16.1.7 Exercício 1.7

Resolva as 6 linhas abaixo e some (aritméticamente) os resultados encontrados, lançando o total no quadro abaixo

- (((14 mod 4) mod 6) mod (18 mod 10))-(((5 - 9) + (4 + 9)) + (2 + 4))
- (((18 div 1) div 4) div (29 div 10))+(((4 + 4) × (4 - 9)) + (9 - 3))
- (((8 + 7) + (6 + 4)) + (5 + 2))-($\lfloor$(8 ÷ 4))
- ((19 mod 3) mod 5)+(((7 + 7) + (3 - 4)) + (9 × 5))
- ((25 div 6) div 3)-(((5 + 8) × (9 + 5)) + (6 + 8))
- (((8 × 6) - (7 + 9)) × (3 + 7))-(3 - 2)

soma dos valores

16.1.8 Exercício 1.8

Resolva as 6 linhas abaixo e some (aritméticamente) os números das linhas que respondem VERDADEIRO, lançando o total no quadro abaixo

1 ((14 div 1) div 5)>($\lfloor$(((3 - 5) - (2 + 7)) ÷ (2 + 5)))

2 (22 div 4)>(((7 - 2) + (8 + 5)) × (8 + 7))

4 (.V. ∧ .V.)∧((.F. ∨ .V.) ∧ .F.)

8 (14 div 2)<(((6 × 4) × (6 - 5)) - ($\lfloor$(4 ÷ 3)))

16 (22 mod 6)<(((4 + 2) - (7 + 7)) + (⌊(8 ÷ 3)))

32 (.F. ∨ .V.)∨((.V. ∨ .F.) ∨ .F.)

soma das linhas

16.1.9 Exercício 1.9

Resolva as 6 linhas abaixo e some (aritméticamente) os resultados encontrados, lançando o total no quadro abaixo

- (((13 mod 9) mod 6) mod (19 mod 4))-(((7 - 2) + (5 + 2)) + (2 + 6))
- (((26 div 3) mod 5) mod (15 div 1))+(((6 + 8) + (8 + 2)) + (4 × 5))
- (3 + 7)+(((9 + 6) × (2 + 6)) × (6 - 3))
- ((11 mod 2) mod 3)+(((2 - 5) + (5 - 4)) - (3 + 8))
- (((12 mod 9) mod 6) mod (13 mod 7))+(((4 + 9) - (3 × 9)) - (3 - 6))
- (((6 - 7) + (⌊(8 ÷ 2))) - (4 - 6))-(2 + 2)

soma dos valores

16.1.10 Exercício 1.10

Resolva as 6 linhas abaixo e some (aritméticamente) os números das linhas que respondem VERDADEIRO, lançando o total no quadro abaixo

1 ((23 div 8) mod 6)>((⌊((3 + 7) ÷ (2 + 8))) + (4 - 7))

2 (24 div 10)<(((5 - 2) - (3 + 7)) - (6 + 8))

4 (.F. ∨ .V.)∧((.V. ∧ .F.) ∧ .V.)

8 (15 div 3)>(((4 + 4) + (9 - 2)) + (9 - 3))

16 (28 mod 6)<(((8 + 9) × (8 + 5)) + (9 - 3))

32 (.F. ∧ .F.)∧((.V. ∧ .F.) ∧ (∼.V.))

soma das linhas

16.2 Exercício 2

16.2.1 Exercício 2.1

Resolva as 6 linhas abaixo e some (aritméticamente) os resultados encontrados, lançando o total no quadro abaixo

- (((19 div 3) div 4) mod (21 div 2))-(((9 - 2) + (3 - 8)) + (9 + 6))
- ((20 div 6) div 3)-(((4 + 6) × (2 × 3)) + (4 + 3))

- (⌊(((7 + 6) + (4 + 4)) ÷ (9 - 4)))-(4 + 8)
- (((23 div 6) div 2) mod (18 div 3))-(((5 + 3) - (4 - 6)) - (8 + 6))
- ((24 div 9) div 2)-(((5 + 8) - (4 - 5)) + (8 - 5))
- (2 + 7)+((⌊((2 - 4) ÷ (5 - 3))) + (2 - 8))

soma dos valores

16.2.2 Exercício 2.2

Resolva as 6 linhas abaixo e some (aritméticamente) os números das linhas que respondem VERDADEIRO, lançando o total no quadro abaixo

1 ((23 div 1) div 4)<(((9 + 6) + (3 + 7)) + (4 - 2))

2 (19 div 1)>(⌊(((2 + 4) × (4 + 5)) ÷ (7 + 3)))

4 (.V. ∧ .F.)∧((.V. ∨ .F.) ∧ (∼.F.))

8 (26 div 4)<(((7 - 5) + (2 + 6)) + (5 - 2))

16 (14 div 1)<(⌊(((3 - 7) + (2 + 4)) ÷ (7 - 9)))

32 (.V. ∨ .V.)∨((.V. ∨ .V.) ∧ .F.)

soma das linhas

16.2.3 Exercício 2.3

Resolva as 6 linhas abaixo e some (aritméticamente) os resultados encontrados, lançando o total no quadro abaixo

- (((28 mod 5) mod 5) mod (27 mod 10))-(((9 + 7) + (7 - 4)) + (3 + 6))
- (((16 div 1) mod 3) mod (18 div 2))+(((5 + 7) × (3 + 4)) - (9 + 9))
- (2 - 9)+(((4 - 7) + (8 - 3)) + (4 - 5))
- ((15 div 4) div 2)+(((6 + 8) - (3 - 5)) + (3 + 2))
- (((20 mod 9) div 1) mod (29 mod 8))+(((3 + 7) × (2 + 8)) + (3 - 6))
- (6 + 7)+(((8 - 5) × (5 + 9)) + (8 - 7))

soma dos valores

16.2.4 Exercício 2.4

Resolva as 6 linhas abaixo e some (aritméticamente) os números das linhas que respondem VERDADEIRO, lançando o total no quadro abaixo

1 ((26 mod 4) mod 3)>(((8 + 2) + (6 + 2)) - (7 - 2))

2 (13 mod 8)<(((2 - 5) + (9 + 7)) - (3 - 2))

4 (.V. ∨ .F.)∧((.F. ∧ .F.) ∧ .V.)

8 (28 mod 6)<(((8 + 5) × (2 - 8)) - (⌊(7 ÷ 5)))

16 (30 div 7)<(((6 + 7) + (2 - 6)) - (⌊(6 ÷ 3)))

32 (.V. ∨ .F.)∨((.F. ∧ .V.) ∧ .V.)

soma das linhas

16.2.5 Exercício 2.5

Resolva as 6 linhas abaixo e some (aritméticamente) os resultados encontrados, lançando o total no quadro abaixo

- (((16 mod 10) mod 5) mod (29 mod 8))-(((5 + 7) - (6 + 2)) - (3 × 8))
- ((26 mod 5) mod 5)-(((7 + 6) + (5 × 8)) - (5 + 8))
- ((((⌊(5 ÷ 5)) - (5 + 5)) + (8 + 7))-(9 - 7)
- (((25 mod 6) div 1) mod (18 mod 7))-(((8 - 4) + (6 × 3)) + (9 + 5))
- (((19 mod 10) div 6) mod (12 mod 3))+(((8 + 2) - (9 - 3)) + (9 × 6))
- (5 - 3)+((⌊((3 - 8) ÷ (7 - 8))) - (6 × 3))

soma dos valores

16.2.6 Exercício 2.6

Resolva as 6 linhas abaixo e some (aritméticamente) os números das linhas que respondem VERDADEIRO, lançando o total no quadro abaixo

1 ((20 mod 8) mod 5)<(((9 + 9) + (2 - 6)) + (8 + 3))

2 (23 div 8)<(((4 - 6) - (4 + 8)) + (2 + 6))

4 (.V. ∨ .F.)∧((.F. ∧ .V.) ∧ .V.)

8 (23 div 6)>(⌊(((9 - 2) - (7 + 7)) ÷ (8 - 2)))

16 (25 mod 9)>((⌊((2 + 9) ÷ (2 + 8))) - (7 × 2))

32 (.V. ∧ .F.)∨((.F. ∧ .F.) ∧ (∼.V.))

soma das linhas

16.2.7 Exercício 2.7

Resolva as 6 linhas abaixo e some (aritméticamente) os resultados encontrados, lançando o total no quadro abaixo

- ((16 mod 5) mod 5)+(((2 × 3) × (2 + 9)) × (3 + 6))
- ((16 div 10) mod 3)-(((9 + 6) - (6 + 4)) - (6 + 2))
- (((5 + 3) - (⌊(9 ÷ 8))) - (6 + 2))-(5 × 7)
- (((27 mod 7) div 5) mod (16 mod 10))-(((3 × 8) + (8 + 2)) - (2 + 5))
- ((30 div 6) div 3)-(((6 - 7) - (2 + 4)) + (2 + 6))
- (4 - 8)+(((2 + 7) × (9 + 8)) - (⌊(8 ÷ 3)))

soma dos valores

16.2.8 Exercício 2.8

Resolva as 6 linhas abaixo e some (aritméticamente) os números das linhas que respondem VERDADEIRO, lançando o total no quadro abaixo

1 ((29 div 7) div 4)>((((⌊(9 ÷ 4)) - (8 + 6)) - (8 + 7))

2 (18 div 4)>(((6 - 9) × (6 + 6)) + (6 - 2))

4 (.F. ∨ .F.)∨((.V. ∧ .V.) ∨ .F.)

8 (17 div 3)<(((4 + 4) - (2 + 9)) + (2 + 8))

16 (22 mod 6)<((((⌊(9 ÷ 7)) × (4 + 2)) - (7 - 9))

32 (.F. ∧ .V.)∨((.F. ∨ .F.) ∨ .F.)

soma das linhas

16.2.9 Exercício 2.9

Resolva as 6 linhas abaixo e some (aritméticamente) os resultados encontrados, lançando o total no quadro abaixo

- (((18 div 6) mod 6) mod (27 div 5))-(((3 + 3) - (2 + 6)) × (2 + 5))
- (((19 div 2) div 1) div (23 div 7))+(((5 + 4) + (2 + 4)) + (2 + 4))
- (((6 - 2) + (5 + 6)) + (8 - 2))-(2 + 9)
- ((28 div 8) div 3)+(((3 - 4) + (6 + 2)) + (2 - 5))
- ((20 div 2) div 1)-(((3 + 3) - (4 + 9)) + (6 + 4))
- (((3 + 2) - (3 + 9)) + (4 - 2))-(6 - 2)

soma dos valores

16.2.10 Exercício 2.10

Resolva as 6 linhas abaixo e some (aritméticamente) os números das linhas que respondem VERDADEIRO, lançando o total no quadro abaixo

1 ((20 mod 8) mod 3)>((($\lfloor$(7 ÷ 3)) - (6 + 8)) + (6 + 7))

2 (26 div 5)>(((8 + 7) × (6 + 5)) - (7 + 9))

4 (∼(.V. ∧ .F.))∧((.V. ∧ .F.) ∧ .F.)

8 (16 div 1)<(((7 × 4) + (3 - 2)) - (9 - 5))

16 (17 div 5)<(((7 × 6) × (8 + 3)) - (9 × 9))

32 (.V. ∨ .F.)∧((∼(.V. ∨ .F.)) ∧ (∼.F.))

soma das linhas

16.2.11 Respostas

```
1.1   -68
1.2   19
1.3   222
1.4   4
1.5   -184
1.6   8
1.7   168
1.8   41
1.9   379
1.10  17

2.1   -99
2.2   43
2.3   213
2.4   50
2.5   -1
2.6   25
2.7   684
2.8   31
2.9   54
2.10  24
```

Capítulo 17

Exercícios Práticos: 017-SEs compostos e encadeados

17.1 SEs compostos e encadeados

Permite executar uma ou mais instruções caso uma condição seja verdadeira. Opcionalmente, pode ter a alternativa inversa, ou seja, executar outros comandos, se a condição for falsa. Comandos internos ao comando "se"devem estar identados de 3 espaços, para clareza.

```
se <condição>
         <comando-1>
         <comando-2>
         ...
   [senão <comando-11>
          <comando-12>
          ...]
fim {se}
```

Alternativa Simples Só possui a primeira parte do comando (comandos são executados se a condição for verdadeira)

```
se <condição>
   comando-1
   comando-2
   ...
fim {se}
```

Alternativa Composta Possui as duas partes. A primeira, se a condição for verdadeira e a segunda se for falsa.

```
se condição
   c1
   c2
   ...
senão
   c10
   c11
```

```
   ...
fim {se condição}
```

A seguir diversos algoritmos envolvendo apenas o comando de alternativa. Siga cada um dos algoritmos e ao final responda qual o valor impresso por cada um deles.

17.1.1 Exercício 1.1

Suponha a chamada EX1 (1, 6, 6, 6, 1, 6), com a seguinte definição

```
1: função EX1 (inteiro A, B, C, D, E, F)
2: se ((E ≠ 5) ∧ (B < 4)) então
3:    se (F ≠ 1) então
4:       D ← E × 5
5:    senão
6:       F ← E × 4
7:    fimse
8:    F ← A + D
9: senão
10:   A ← C × D
11:   se ((B ≥ 5) ∨ (A = 6)) então
12:      B ← C - D
13:   fimse
14: fimse
15: imprima A + B + C + D + E
16: fim {função}
```

17.1.2 Exercício 1.2

Suponha a chamada EX2 (4, 2, 3, 2, 1, 2), com a seguinte definição

```
1: função EX2 (inteiro A, B, C, D, E, F)
2: se ((F ≤ 1) ∨ (C ≤ 5)) então
3:    se (D > 5) então
4:       F ← A × C
5:    senão
6:       D ← C × F
7:    fimse
8:    D ← B + 5
9: senão
10:   E ← B + 5
11:   se ((∼ (B ≠ 2)) ∧ (∼ (A ≤ 1))) então
12:      C ← C + 2
13:   fimse
14: fimse
15: imprima A + B + C + D + E
16: fim {função}
```

17.1.3 Exercício 1.3

Suponha a chamada EX3 (3, 1, 2, 1, 3, 4), com a seguinte definição

```
1: função EX3 (inteiro A, B, C, D, E, F)
2: se (((D ≤ 5) ∨ (F ≥ 5)) ∧ (A ≠ 3)) então
3:    se ((F > 6) ∨ (∼ (D ≤ 2))) então
```

```
4:      B ← A - E
5:    senão
6:      C ← D - E
7:    fimse
8:    E ← C - 2
9:  senão
10:   D ← A × 4
11:   se ((A = 4) ∨ (A < 4)) então
12:     F ← D × B
13:   fimse
14: fimse
15: imprima A + B + C + D + E
16: fim {função}
```

17.1.4 Exercício 1.4

Suponha a chamada EX4 (2, 4, 5, 1, 2, 5), com a seguinte definição

```
1: função EX4 (inteiro A, B, C, D, E, F)
2: se (((E < 5) ∧ (∼ (C = 1))) ∨ (∼ (D < 3))) então
3:   se ((E = 2) ∨ (E ≠ 4)) então
4:     C ← C + B
5:   senão
6:     C ← B × E
7:   fimse
8:   C ← F - C
9: senão
10:  C ← A × E
11:  se (A ≤ 6) então
12:    C ← D - 4
13:  fimse
14: fimse
15: imprima A + B + C + D + E
16: fim {função}
```

17.1.5 Exercício 1.5

Suponha a chamada EX5 (5, 3, 3, 3, 3, 6), com a seguinte definição

```
1: função EX5 (inteiro A, B, C, D, E, F)
2: se ((∼ (D ≤ 3)) ∨ (∼ (A ≠ 4))) então
3:   se ((∼ (B < 6)) ∧ (E ≠ 4)) então
4:     D ← C - E
5:   senão
6:     F ← B × 2
7:   fimse
8:   A ← A + 5
9: senão
10:  F ← B - 4
11:  se (((D > 4) ∧ (C ≤ 2)) ∧ (C ≠ 5)) então
12:    F ← A × 3
13:  fimse
14: fimse
15: imprima A + B + C + D + E
```


16: fim {função}

17.1.6 Exercício 1.6

Suponha a chamada EX6 (1, 4, 3, 5, 2, 3), com a seguinte definição

1: função EX6 (inteiro A, B, C, D, E, F)
2: **se** ((∼ (F ≤ 2)) ∧ (∼ (B ≥ 6))) **então**
3: **se** ((D < 3) ∧ (∼ (B = 6))) **então**
4: C ← D - E
5: **senão**
6: B ← C × F
7: **fimse**
8: B ← A + 2
9: **senão**
10: B ← C - 2
11: **se** (F = 4) **então**
12: F ← B - F
13: **fimse**
14: **fimse**
15: imprima A + B + C + D + E
16: fim {função}

Respostas

1	2	3	4	5	6

17.2 Exercício 2

A seguir diversos algoritmos envolvendo apenas o comando de alternativa. Siga cada um dos algoritmos e ao final responda qual o valor impresso por cada um deles.

17.2.1 Exercício 2.1

Suponha a chamada EX1 (3, 2, 2, 3, 2, 1), com a seguinte definição

1: função EX1 (inteiro A, B, C, D, E, F)
2: **se** (((∼ (D ≤ 1)) ∧ (B < 1)) ∧ (A > 6)) **então**
3: **se** (D ≠ 2) **então**
4: D ← C × D
5: **senão**
6: A ← C + 4
7: **fimse**
8: A ← F + E
9: **senão**
10: B ← C × 3
11: **se** (((B = 5) ∧ (F = 1)) ∨ (B > 4)) **então**
12: C ← E × 5
13: **fimse**
14: **fimse**
15: imprima A + B + C + D + E
16: fim {função}

17.2.2 Exercício 2.2

Suponha a chamada EX2 (1, 4, 6, 3, 3, 2), com a seguinte definição

```
1: função EX2 (inteiro A, B, C, D, E, F)
2: se ((∼ (D ≥ 6)) ∧ (∼ (D > 5))) então
3:    se ((∼ (D > 1)) ∧ (∼ (C > 1))) então
4:       A ← D × 5
5:    senão
6:       D ← C - 3
7:    fimse
8:    A ← E - 4
9: senão
10:   C ← A - 3
11:   se ((C < 2) ∨ (F ≠ 5)) então
12:      D ← E - B
13:   fimse
14: fimse
15: imprima A + B + C + D + E
16: fim {função}
```

17.2.3 Exercício 2.3

Suponha a chamada EX3 (4, 1, 2, 3, 3, 1), com a seguinte definição

```
1: função EX3 (inteiro A, B, C, D, E, F)
2: se (∼ (F ≤ 1)) então
3:    se ((E ≠ 3) ∨ (E > 4)) então
4:       B ← F × 2
5:    senão
6:       F ← E + F
7:    fimse
8:    C ← B × 3
9: senão
10:   E ← C + E
11:   se ((D ≤ 1) ∧ (E ≥ 6)) então
12:      D ← F × E
13:   fimse
14: fimse
15: imprima A + B + C + D + E
16: fim {função}
```

17.2.4 Exercício 2.4

Suponha a chamada EX4 (6, 4, 6, 4, 2, 4), com a seguinte definição

```
1: função EX4 (inteiro A, B, C, D, E, F)
2: se ((∼ (D ≤ 2)) ∨ (E ≠ 6)) então
3:    se (((D < 6) ∧ (A = 6)) ∧ (∼ (F ≤ 6))) então
4:       A ← C - B
5:    senão
6:       B ← F × D
7:    fimse
8:    C ← A - 5
9: senão
```



```
10:    F ← F × 2
11:    se (((A < 6) ∨ (∼ (B = 6))) ∨ (∼ (A = 6))) então
12:       D ← B + 4
13:    fimse
14: fimse
15: imprima A + B + C + D + E
16: fim {função}
```

17.2.5 Exercício 2.5

Suponha a chamada EX5 (1, 2, 4, 2, 6, 6), com a seguinte definição

```
 1: função EX5 (inteiro A, B, C, D, E, F)
 2: se (((B ≠ 1) ∧ (B > 1)) ∧ (∼ (F > 4))) então
 3:    se (∼ (A < 4)) então
 4:       C ← A + E
 5:    senão
 6:       C ← E × 2
 7:    fimse
 8:    E ← B - 4
 9: senão
10:    F ← F × 2
11:    se (C > 2) então
12:       A ← F × D
13:    fimse
14: fimse
15: imprima A + B + C + D + E
16: fim {função}
```

17.2.6 Exercício 2.6

Suponha a chamada EX6 (6, 2, 6, 1, 5, 3), com a seguinte definição

```
 1: função EX6 (inteiro A, B, C, D, E, F)
 2: se (((∼ (A > 4)) ∨ (∼ (A ≤ 1))) ∨ (∼ (A > 6))) então
 3:    se (∼ (F ≤ 4)) então
 4:       B ← F + 2
 5:    senão
 6:       C ← D × 4
 7:    fimse
 8:    A ← C + 3
 9: senão
10:    D ← A - C
11:    se ((∼ (A ≠ 4)) ∨ (E ≤ 3)) então
12:       F ← C - 4
13:    fimse
14: fimse
15: imprima A + B + C + D + E
16: fim {função}
```

Respostas

1	2	3	4	5	6

17.2.7 Respostas

```
   1.1        49
1.2    17
1.3 21
1.4 5
1.5 17
1.6 14

   2.1        24
2.2    15
2.3 15
2.4 29
2.5 38
2.6 19
```

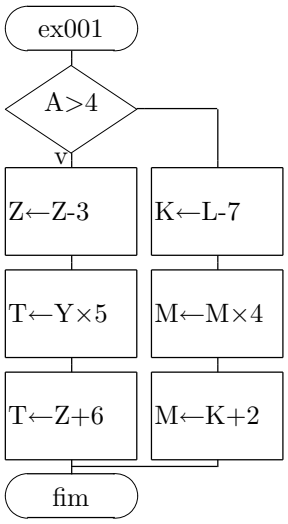

Capítulo 18

Exercícios Práticos: 018-FLuxogramas

18.1 Exercícios de Fluxos e Pseudocódigo

A seguir, são apresentados um fluxograma e a seguir um trecho em pseudo-código. Ambos devem ser analisados e deve-se chegar a uma conclusão: *são equivalentes ?* Se forem, os números que identificam cada exercício devem ser somados. Ao final da série, a soma deve ser respondida.

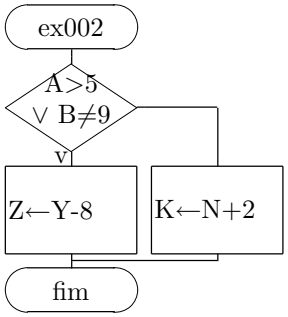
18.1.1 Exercício 1




```
 ex001
se (A>4) então
   Z←Z-3
   T←Y×5
   T←Z+6
 senão
   K←L-7
```

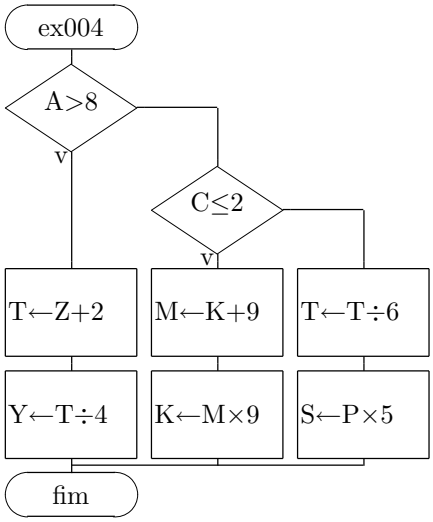
```
  M←M×4
  M←K+2
 fim{se}
```

Se os trechos acima forem equivalentes, some 1 à resposta.




```
ex002
 se (A≤5)∧(B=9) então
   Z←Y-8
 senão
   K←N+2
 fim{se}
```

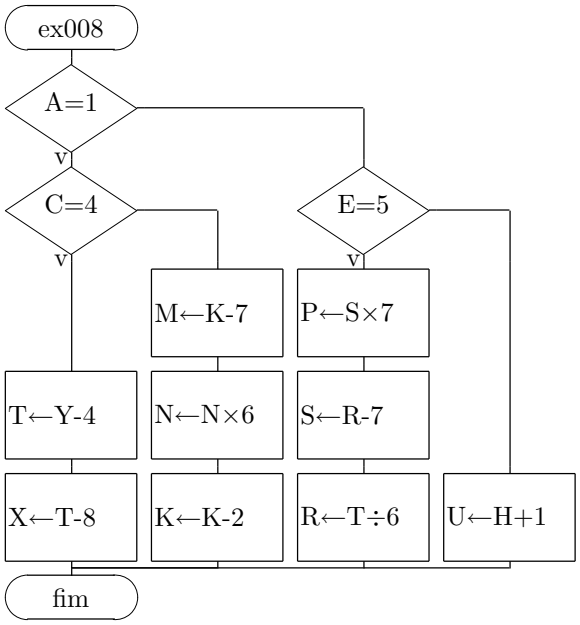
Se os trechos acima forem equivalentes, some 2 à resposta.




```
 ex004
se (A>8) então
   T←Z+2
   Y←T÷4
 senão
   se (C>2) então
     M←K+9
```

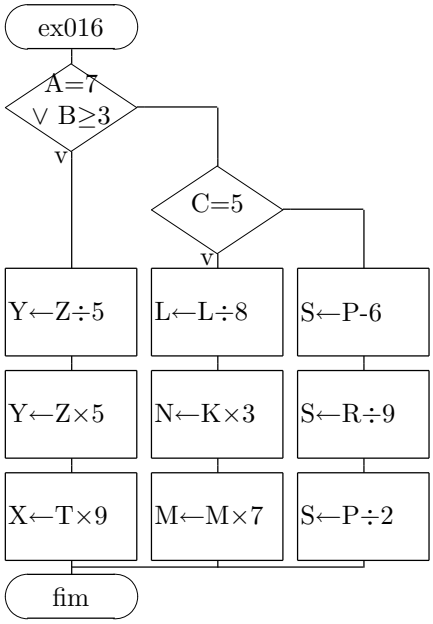
```
    K←M×9
  senão
    T←T÷6
    S←P×5
  fim{se}
 fim{se}
```

Se os trechos acima forem equivalentes, some 4 à resposta.



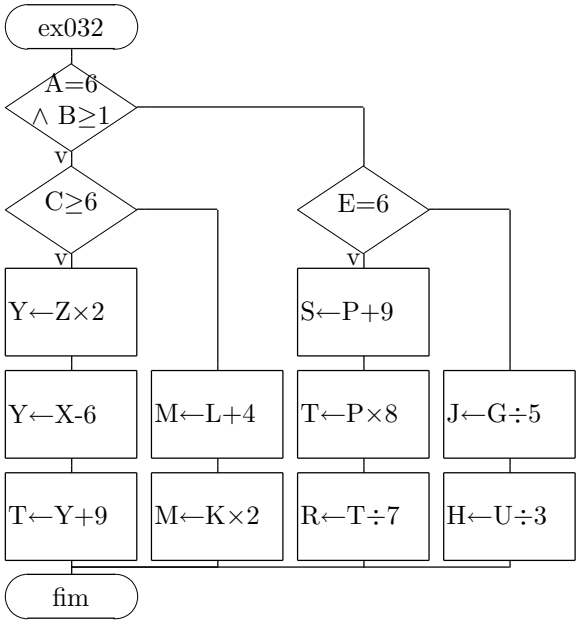

```
 ex008
se (A≠1) então
   se (E≠5) então
     P←S×7
     S←R-7
     R←T÷6
   senão
     U←H+1
   fim{se}
 senão
   se (C≠4) então
     M←K-7
     N←N×6
     K←K-2
   senão
     T←Y-4
     X←T-8
   fim{se}
 fim{se}
```

Se os trechos acima forem equivalentes, some 8 à resposta.



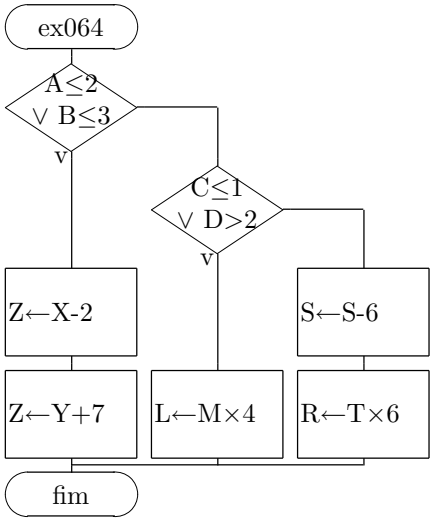

```
 ex016
se (A≠7)∧(B<3) então
   se (C≠5) então
     S←P-6
     S←R÷9
     S←P÷2
   senão
     L←L÷8
     N←K×3
     M←M×7
   fim{se}
 senão
   Y←Z÷5
   Y←Z×5
   X←T×9
 fim{se}
```

Se os trechos acima forem equivalentes, some 16 à resposta.



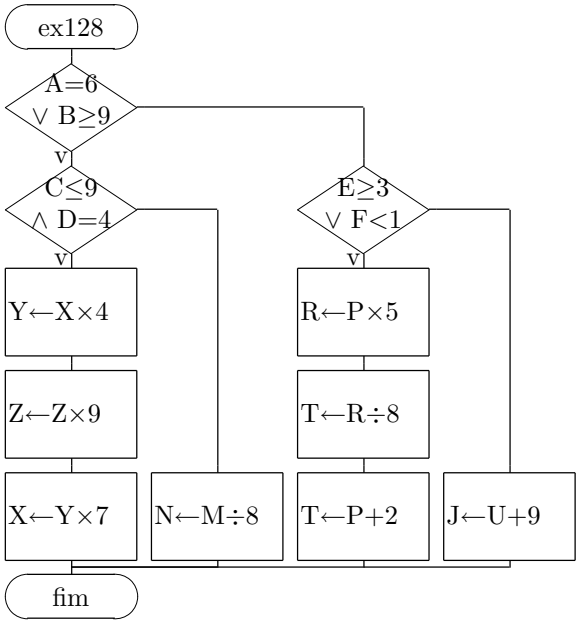

```
 ex032
se (A≠6)∨(B<1) então
   se (C≥6) então
     Y←Z×2
     Y←X-6
     T←Y+9
   senão
     M←L+4
     M←K×2
   fim{se}
 senão
   se (E≠6) então
     S←P+9
     T←P×8
     R←T÷7
   senão
     J←G÷5
     H←U÷3
   fim{se}
 fim{se}
```

Se os trechos acima forem equivalentes, some 32 à resposta.




```
 ex064
se (A≤2)∨(B≤3) então
   se (C≤1)∨(D>2) então
     S←S-6
     R←T×6
   senão
     L←M×4
   fim{se}
 senão
   Z←X-2
   Z←Y+7
 fim{se}
```

Se os trechos acima forem equivalentes, some 64 à resposta.



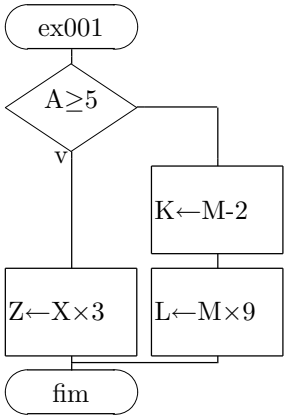

```
 ex128
se (A≠6)∧(B<9) então
   se (E<3)∧(F≥1) então
     R←P×5
     T←R÷8
     T←P+2
   senão
     J←U+9
   fim{se}
 senão
   se (C>9)∨(D≠4) então
     N←M÷8
   senão
     Y←X×4
     Z←Z×9
     X←Y×7
   fim{se}
 fim{se}
```

Se os trechos acima forem equivalentes, some 128 à resposta.

Responda a soma dos exercícios que são equivalentes

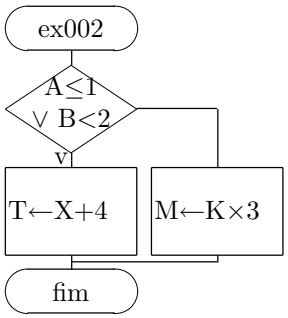
18.1.2 Exercício 2

A seguir, some o número dos exercícios em que o trecho de pseudocódigo é 100% equivalente ao fluxograma apresentado



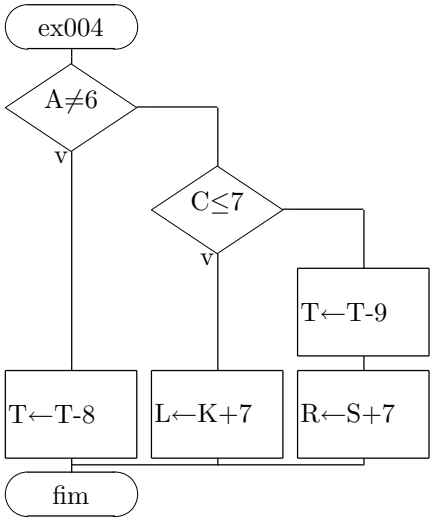

```
 ex001
se (A≥5) então
   Z←X×3
 senão
   K←M-2
   L←M×9
 fim{se}
```

Se os trechos acima forem equivalentes, some 1 à resposta.



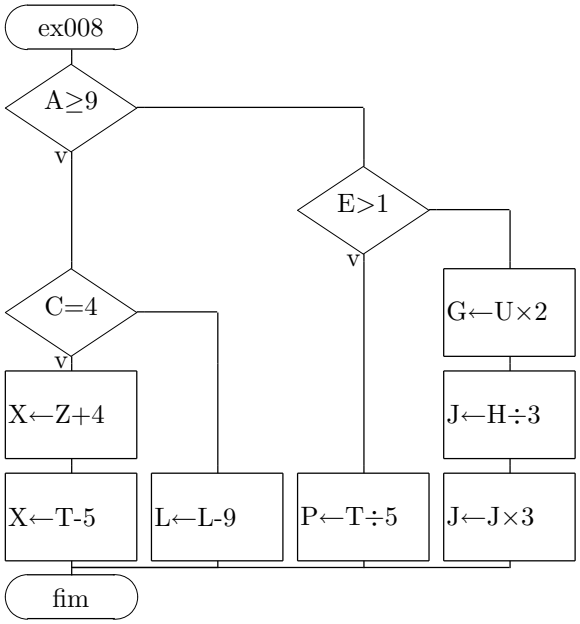

```
 ex002
se (A>1)∧(B≥2) então
   T←X+4
 senão
   M←K×3
 fim{se}
```

Se os trechos acima forem equivalentes, some 2 à resposta.



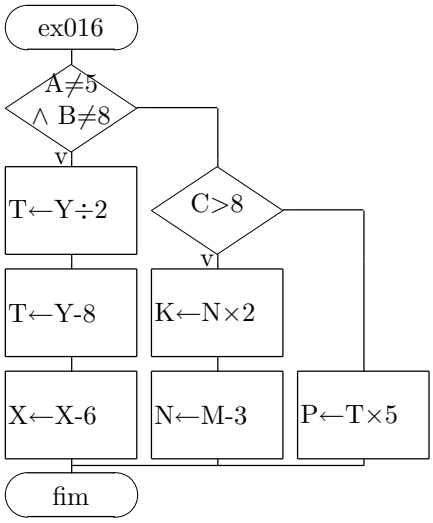

```
 ex004
se (A=6) então
   se (C>7) então
     T←T-9
     R←S+7
   senão
     L←K+7
   fim{se}
 senão
   T←T-8
 fim{se}
```

Se os trechos acima forem equivalentes, some 4 à resposta.




```
 ex008
se (A<9) então
   se (E≤1) então
     P←T÷5
   senão
     G←U×2
     J←H÷3
     J←J×3
   fim{se}
 senão
   se (C≠4) então
     L←L-9
   senão
     X←Z+4
     X←T-5
   fim{se}
 fim{se}
```

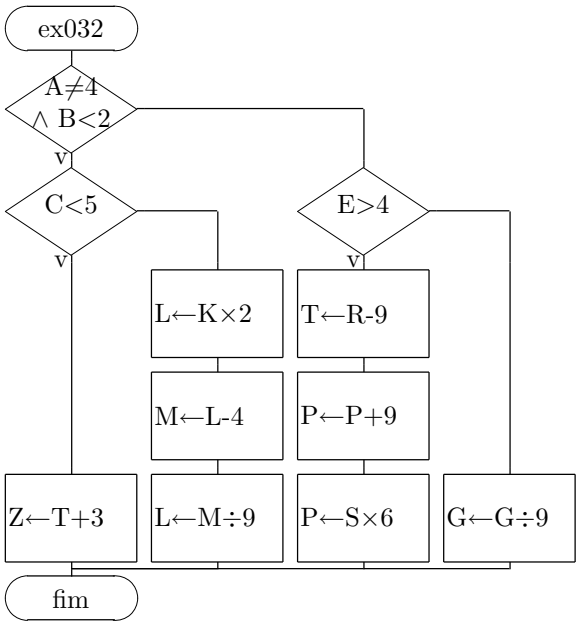
Se os trechos acima forem equivalentes, some 8 à resposta.




```
ex016
 se (A≠5)∧(B≠8) então
   T←Y÷2
   T←Y-8
   X←X-6
 senão
   se (C>8) então
     K←N×2
     N←M-3
   senão
     P←T×5
   fim{se}
```

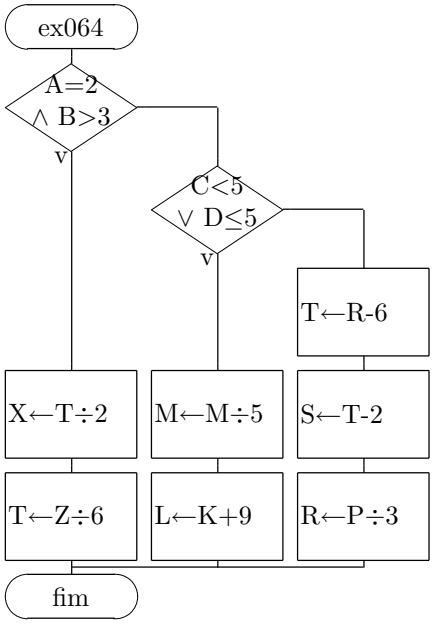
```
 fim{se}
```

Se os trechos acima forem equivalentes, some 16 à resposta.



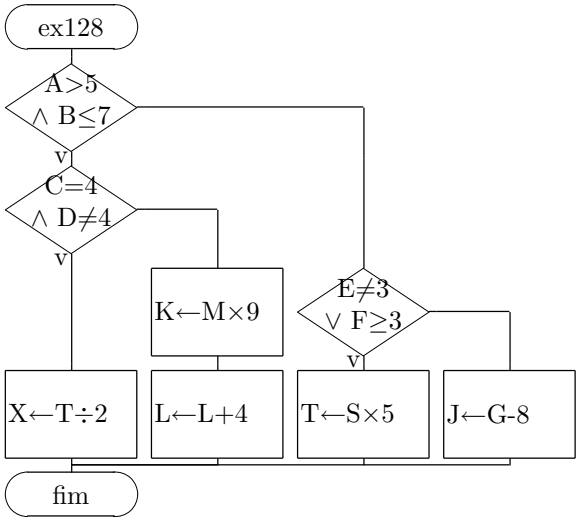

```
 ex032
se (A≠4)∧(B<2) então
   se (C<5) então
     T←R-9
     P←P+9
     P←S×6
   senão
     G←G÷9
   fim{se}
 senão
   se (E>4) então
     L←K×2
     M←L-4
     L←M÷9
   senão
     Z←T+3
   fim{se}
 fim{se}
```

Se os trechos acima forem equivalentes, some 32 à resposta.




```
 ex064
se (A≠2)∨(B≤3) então
   se (C≥5)∧(D>5) então
     T←R-6
     S←T-2
     R←P÷3
   senão
     M←M÷5
     L←K+9
   fim{se}
 senão
   X←T÷2
   T←Z÷6
 fim{se}
```

Se os trechos acima forem equivalentes, some 64 à resposta.




```
 ex128
se (A>5)∧(B≤7) então
   se (C=4)∧(D≠4) então
     T←S×5
   senão
     J←G-8
   fim{se}
 senão
   se (E≠3)∨(F≥3) então
     K←M×9
     L←L+4
   senão
     X←T÷2
   fim{se}
 fim{se}
```

Se os trechos acima forem equivalentes, some 128 à resposta.
Responda a soma dos exercícios que são equivalentes

18.1.3 Respostas

```
1       153
2        93
```


Capítulo 19

Exercícios Práticos: 018 - Nassi

Nassi e Pseudocódigo: A seguir, some o número dos exercícios em que o trecho de pseudo- código é 100% equivalente ao diagrama de Nassi apresentado

19.0.4 Exercício 1

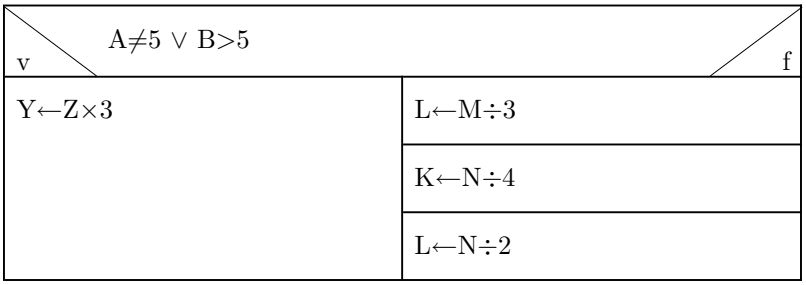
ex001 —

A≠1 v	f
Z←Z+4	M←K×8
T←Z÷5	K←N÷5
Z←X÷9	K←M×2

```
 ex001
se (A=1) então
   Z←Z+4
   T←Z÷5
   Z←X÷9
 senão
   M←K×8
   K←N÷5
   K←M×2
 fim{se}
```

Se os trechos acima forem equivalentes, some 1 à resposta.

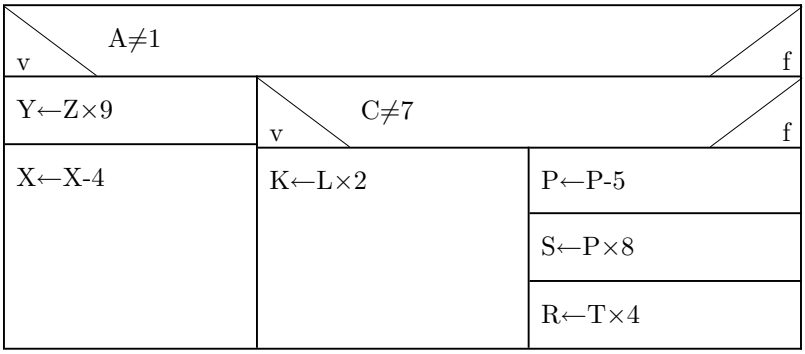
ex002 —




```
 ex002
se (A≠5)∨(B>5) então
   L←M÷3
   K←N÷4
   L←N÷2
 senão
   Y←Z×3
 fim{se}
```

Se os trechos acima forem equivalentes, some 2 à resposta.

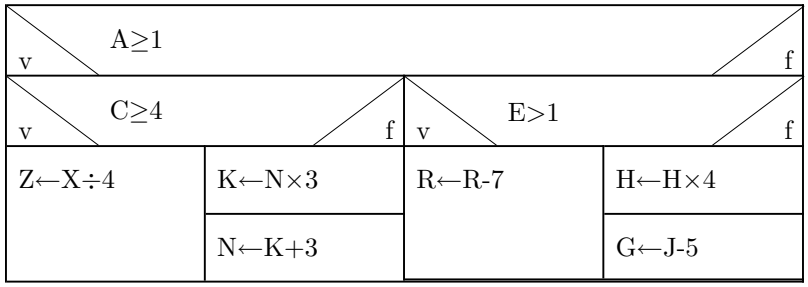
ex004 —




```
 ex004
se (A=1) então
   se (C=7) então
     P←P-5
     S←P×8
     R←T×4
   senão
     K←L×2
   fim{se}
 senão
   Y←Z×9
   X←X-4
 fim{se}
```

Se os trechos acima forem equivalentes, some 4 à resposta.

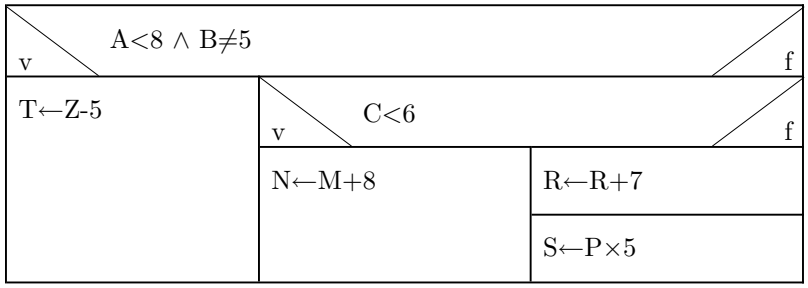
ex008 —




```
 ex008
se (A<1) então
   se (E≤1) então
     R←R-7
   senão
     H←H×4
     G←J-5
   fim{se}
 senão
   se (C<4) então
     K←N×3
     N←K+3
   senão
     Z←X÷4
   fim{se}
 fim{se}
```

Se os trechos acima forem equivalentes, some 8 à resposta.

ex016 —




```
 ex016
se (A≥8)∨(B=5) então
   T←Z-5
 senão
```



```
  se (C≥6) então
    N←M+8
  senão
    R←R+7
    S←P×5
  fim{se}
fim{se}
```

Se os trechos acima forem equivalentes, some 16 à resposta.

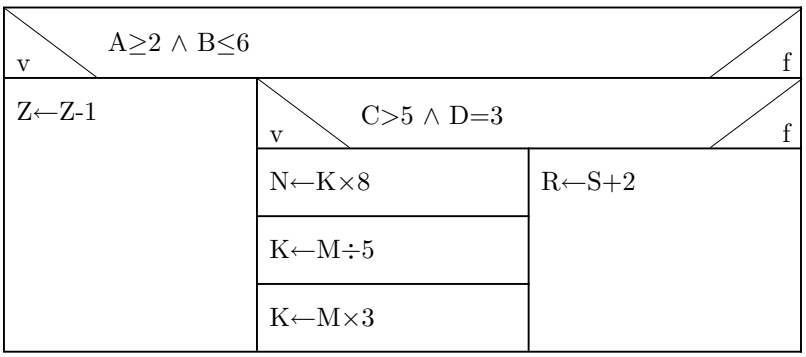
ex032 —

<table>
<tr><td colspan="4">A≥2 ∧ B≥4
v f</td></tr>
<tr><td colspan="2">C≥3
v f</td><td colspan="2">E≤7
v f</td></tr>
<tr><td>Y←T÷1</td><td>K←K×3</td><td>R←S×9</td><td>U←G-3</td></tr>
<tr><td rowspan="2"></td><td>N←M+8</td><td>T←R-3</td><td rowspan="2">J←U-2</td></tr>
<tr><td colspan="2"></td><td>P←T+6</td></tr>
</table>

```
 ex032
se (A<2)∨(B<4) então
  se (C≥3) então
    Y←T÷1
  senão
    K←K×3
    N←M+8
  fim{se}
 senão
  se (E>7) então
    R←S×9
    T←R-3
    P←T+6
  senão
    U←G-3
    J←U-2
  fim{se}
 fim{se}
```

Se os trechos acima forem equivalentes, some 32 à resposta.

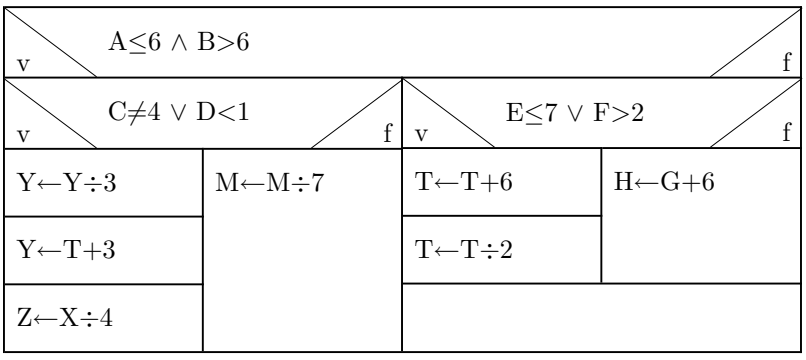
ex064 —




```
 ex064
se (A<2)∨(B>6) então
   Z←Z-1
 senão
   se (C≤5)∨(D≠3) então
     N←K×8
     K←M÷5
     K←M×3
   senão
     R←S+2
   fim{se}
 fim{se}
```

Se os trechos acima forem equivalentes, some 64 à resposta.

ex128 —




```
 ex128
se (A>6)∨(B≤6) então
   se (C≠4)∨(D<1) então
     Y←Y÷3
     Y←T+3
     Z←X÷4
```



```
  senão
    M←M÷7
  fim{se}
 senão
  se (E>7)∧(F≤2) então
    T←T+6
    T←T÷2
  senão
    H←G+6
  fim{se}
 fim{se}
```

Se os trechos acima forem equivalentes, some 128 à resposta.
Responda a soma dos exercícios que são equivalentes

19.0.5 Exercício 2

ex001 —

v A=8	f
X←X-9	K←K÷6
X←Z÷8	

```
 ex001
se (A≠8) então
  X←X-9
  X←Z÷8
 senão
  K←K÷6
 fim{se}
```

Se os trechos acima forem equivalentes, some 1 à resposta.

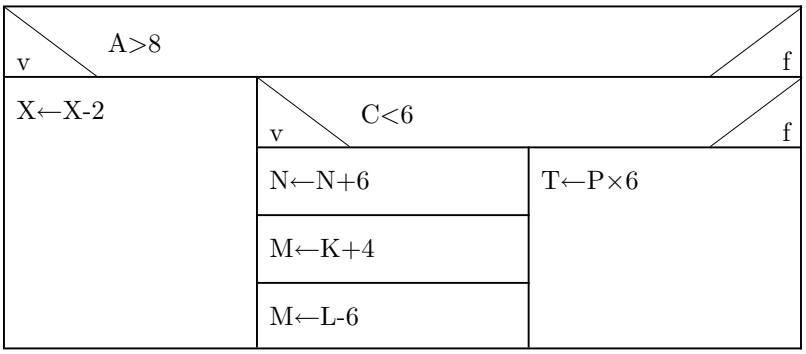
ex002 —

v A≥4 ∨ B=8	f
T←Z÷6	L←M÷4
T←Z×5	N←M×4
	N←K+2

```
 ex002
se (A≥4)∨(B=8) então
   L←M÷4
   N←M×4
   N←K+2
 senão
   T←Z÷6
   T←Z×5
 fim{se}
```

Se os trechos acima forem equivalentes, some 2 à resposta.

ex004 —




```
 ex004
se (A>8) então
   X←X-2
 senão
   se (C≥6) então
     N←N+6
     M←K+4
     M←L-6
   senão
     T←P×6
   fim{se}
 fim{se}
```

Se os trechos acima forem equivalentes, some 4 à resposta.

ex008 —

<table>
<tr><td colspan="4">A≠5
v f</td></tr>
<tr><td colspan="2">C≥1
v f</td><td colspan="2">E≠1
v f</td></tr>
<tr><td>Z←Y+2</td><td>L←L×7</td><td>R←S×8</td><td>J←G+7</td></tr>
<tr><td>Z←T+5</td><td rowspan="2">K←L+5</td><td>P←S+3</td><td rowspan="2">G←U÷6</td></tr>
<tr><td>X←Y×7</td><td>R←R+5</td></tr>
</table>

```
 ex008
se (A≠5) então
   se (C≥1) então
     Z←Y+2
     Z←T+5
     X←Y×7
   senão
     L←L×7
     K←L+5
   fim{se}
 senão
   se (E≠1) então
     R←S×8
     P←S+3
     R←R+5
   senão
     J←G+7
     G←U÷6
   fim{se}
 fim{se}
```

Se os trechos acima forem equivalentes, some 8 à resposta.

ex016 —

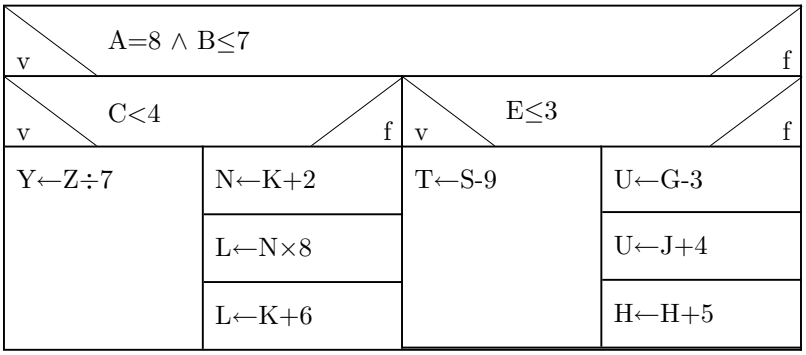
<table>
<tr><td colspan="3">A>6 ∧ B=1
v f</td></tr>
<tr><td rowspan="3">T←Z÷9</td><td colspan="2">C≤3
v f</td></tr>
<tr><td>N←L-7</td><td rowspan="2">R←T×4</td></tr>
<tr><td>N←K×3</td></tr>
</table>

```
 ex016
```

```
se (A>6)∧(B=1) então
   se (C≤3) então
     R←T×4
   senão
     N←L-7
     N←K×3
   fim{se}
 senão
   T←Z÷9
 fim{se}
```

Se os trechos acima forem equivalentes, some 16 à resposta.

ex032 —




```
 ex032
se (A≠8)∨(B>7) então
   se (E>3) então
     T←S-9
   senão
     U←G-3
     U←J+4
     H←H+5
   fim{se}
 senão
   se (C≥4) então
     N←K+2
     L←N×8
     L←K+6
   senão
     Y←Z÷7
   fim{se}
 fim{se}
```

Se os trechos acima forem equivalentes, some 32 à resposta.

ex064 —

<table>
<tr><td colspan="3">A>1 ∧ B=4
v / f</td></tr>
<tr><td>Y←X×7</td><td colspan="2">C=3 ∧ D=1
v / f</td></tr>
<tr><td rowspan="3">T←X÷8</td><td>N←N-2</td><td>R←S-9</td></tr>
<tr><td>N←K÷3</td><td>S←R×5</td></tr>
<tr><td></td><td>P←R÷4</td></tr>
</table>

```
 ex064
se (A≤1)∨(B≠4) então
   se (C≠3)∨(D≠1) então
     R←S-9
     S←R×5
     P←R÷4
   senão
     N←N-2
     N←K÷3
   fim{se}
 senão
   Y←X×7
   T←X÷8
 fim{se}
```

Se os trechos acima forem equivalentes, some 64 à resposta.

ex128 —

<table>
<tr><td colspan="4">A≥7 ∧ B≠7
v / f</td></tr>
<tr><td colspan="2">C≠9 ∨ D≠8
v / f</td><td colspan="2">E≠2 ∧ F≤4
v / f</td></tr>
<tr><td>Y←X÷4</td><td>L←M÷5</td><td>R←S×2</td><td>G←H+2</td></tr>
<tr><td>X←Y-9</td><td>M←L+7</td><td rowspan="2">T←P-2</td><td>H←U-2</td></tr>
<tr><td colspan="2"></td><td>G←J÷8</td></tr>
</table>

```
 ex128
se (A≥7)∧(B≠7) então
   se (C≠9)∨(D≠8) então
     Y←X÷4
```

```
    X←Y-9
  senão
    L←M÷5
    M←L+7
  fim{se}
senão
  se (E≠2)∧(F≤4) então
    R←S×2
    T←P-2
  senão
    G←H+2
    H←U-2
    G←J÷8
  fim{se}
fim{se}
```

Se os trechos acima forem equivalentes, some 128 à resposta.
Responda a soma dos exercícios que são equivalentes

19.0.6 Respostas

1	12
2	232

Capítulo 20

Exercícios práticos: 021 - 5 funções simples

20.1 Funções numéricas

Os exercícios a seguir estão indicados para desenvolver as habilidades de

- saber resolver condições lógicas
- adquirir destreza em avaliar comandos condicionais
- ganhar disciplina mental para seguir testes de mesma (chineses)

Os exercícios a seguir implementam funções. Em todas, as variáveis A, B e C são inicializadas no início da função. Você deve seguir a função até o final e neste ponto informar os valores de A, B e C.

20.1.1 Exercício 1

Exercício 1.1

Suponha a chamada à função CALC, com a seguinte definição

```
1: função CALC
2: inteiro A, B, C
3: A ← 1
4: B ← 8
5: C ← 7
6: se (B > 5) então
7:   se (C ≠ A)  então
8:     se (A ≥ B)  então
9:       C ← C × 5
10:    senão
11:      B ← B - B
12:    fimse
13:  senão
14:    se (B ≥ C)  então
15:      A ← B - 3
16:    senão
17:      B ← B + 2
18:    fimse
```

```
19:    fimse
20: senão
21:    se (∼ (C = B))  então
22:       A ← C × 2
23:    senão
24:       B ← A - 3
25:    fimse
26: fimse
27: imprima A, B, C
28: fim{função}
```

Exercício 1.2

Suponha a chamada à função CALC, com a seguinte definição

```
1: função CALC
2: inteiro A, B, C
3: A ← 5
4: B ← 2
5: C ← 2
6: se (B < C) então
7:    se (A ≥ C)  então
8:       se (C ≠ B)  então
9:          A ← C - 4
10:      senão
11:         A ← A - A
12:      fimse
13:   senão
14:      se (∼ (C = B))  então
15:         B ← A - 4
16:      senão
17:         B ← C + 3
18:      fimse
19:   fimse
20: senão
21:    se (A < 6)  então
22:       C ← B + 5
23:    senão
24:       A ← C × 4
25:    fimse
26: fimse
27: imprima A, B, C
28: fim{função}
```

Exercício 1.3

Suponha a chamada à função CALC, com a seguinte definição

```
1: função CALC
2: inteiro A, B, C
3: A ← 1
4: B ← 9
5: C ← 9
6: se (∼ (C > A)) então
```

```
 7:    se (C ≥ B) então
 8:      se (B = C) então
 9:        C ← B × 5
10:      senão
11:        B ← B + B
12:      fimse
13:    senão
14:      se (B > A) então
15:        C ← B - 2
16:      senão
17:        C ← C + B
18:      fimse
19:    fimse
20: senão
21:    se (∼ (A ≠ 2)) então
22:      A ← A × 5
23:    senão
24:      B ← A - 2
25:    fimse
26: fimse
27: imprima A, B, C
28: fim{função}
```

Exercício 1.4

Suponha a chamada à função CALC, com a seguinte definição

```
 1: função CALC
 2: inteiro A, B, C
 3: A ← 4
 4: B ← 8
 5: C ← 7
 6: se (∼ (B < A)) então
 7:    se (∼ (C < B)) então
 8:      se (B < C) então
 9:        B ← C + 4
10:      senão
11:        B ← A × B
12:      fimse
13:    senão
14:      se (B < A) então
15:        C ← A - B
16:      senão
17:        B ← C + 3
18:      fimse
19:    fimse
20: senão
21:    se (B = 1) então
22:      A ← A + A
23:    senão
24:      B ← C - 4
25:    fimse
26: fimse
```



```
27: imprima A, B, C
28: fim{função}
```

Exercício 1.5

Suponha a chamada à função CALC, com a seguinte definição

```
 1: função CALC
 2: inteiro A, B, C
 3: A ← 7
 4: B ← 8
 5: C ← 8
 6: se (A = C) então
 7:    se (C < 3)  então
 8:       se (A ≤ 4)  então
 9:          B ← A + 2
10:       senão
11:          C ← C × C
12:       fimse
13:    senão
14:       se (∼ (B = 1))  então
15:          A ← A + 2
16:       senão
17:          C ← C + 4
18:       fimse
19:    fimse
20: senão
21:    se (∼ (B ≤ 3))  então
22:       B ← C - 4
23:    senão
24:       B ← C × A
25:    fimse
26: fimse
27: imprima A, B, C
28: fim{função}
```

Respostas

Exerc 1	Exerc 2	Exerc 3	Exerc 4	Exerc 5
Var. B	Var. C	Var. B	Var. B	Var. B

20.1.2 Exercício 2

Exercício 2.1

Suponha a chamada à função CALC, com a seguinte definição

```
1: função CALC
2: inteiro A, B, C
3: A ← 8
4: B ← 4
```

```
 5: C ← 5
 6: se (A < C) então
 7:    se (∼ (B > C)) então
 8:       se (C < B) então
 9:          B ← B - 2
10:       senão
11:          B ← A + B
12:       fimse
13:    senão
14:       se (∼ (C ≤ A)) então
15:          C ← A + 5
16:       senão
17:          B ← B + B
18:       fimse
19:    fimse
20: senão
21:    se (B = A) então
22:       A ← A × A
23:    senão
24:       C ← B + C
25:    fimse
26: fimse
27: imprima A, B, C
28: fim{função}
```

Exercício 2.2

Suponha a chamada à função CALC, com a seguinte definição

```
 1: função CALC
 2: inteiro A, B, C
 3: A ← 4
 4: B ← 5
 5: C ← 7
 6: se (A = 1) então
 7:    se (∼ (B ≥ 2)) então
 8:       se (∼ (B = C)) então
 9:          B ← A + 4
10:       senão
11:          B ← C × 5
12:       fimse
13:    senão
14:       se (A > C) então
15:          C ← C + C
16:       senão
17:          C ← C × 3
18:       fimse
19:    fimse
20: senão
21:    se (B ≠ 2) então
22:       B ← C - B
23:    senão
24:       A ← C × 2
```



```
25:    fimse
26: fimse
27: imprima A, B, C
28: fim{função}
```

Exercício 2.3

Suponha a chamada à função CALC, com a seguinte definição

```
 1: função CALC
 2: inteiro A, B, C
 3: A ← 9
 4: B ← 8
 5: C ← 7
 6: se (B > C) então
 7:    se (∼ (A = 4))  então
 8:       se (∼ (B < 5))  então
 9:          B ← C × 4
10:       senão
11:          A ← B - 2
12:       fimse
13:    senão
14:       se (C < A)  então
15:          C ← B - 3
16:       senão
17:          A ← B × 5
18:       fimse
19:    fimse
20: senão
21:    se (∼ (A ≥ 1))  então
22:       B ← A - 5
23:    senão
24:       A ← A - 5
25:    fimse
26: fimse
27: imprima A, B, C
28: fim{função}
```

Exercício 2.4

Suponha a chamada à função CALC, com a seguinte definição

```
 1: função CALC
 2: inteiro A, B, C
 3: A ← 3
 4: B ← 2
 5: C ← 9
 6: se (A = 2) então
 7:    se (∼ (C > 1))  então
 8:       se (C ≤ 5)  então
 9:          C ← A × 2
10:       senão
11:          B ← B × 5
12:       fimse
```

```
13:    senão
14:       se (C ≠ 1)  então
15:          A ← A × A
16:       senão
17:          B ← A - 4
18:       fimse
19:    fimse
20: senão
21:    se (C = 2)  então
22:       A ← A + C
23:    senão
24:       C ← C × A
25:    fimse
26: fimse
27: imprima A, B, C
28: fim{função}
```

Exercício 2.5

Suponha a chamada à função CALC, com a seguinte definição

```
 1: função CALC
 2: inteiro A, B, C
 3: A ← 1
 4: B ← 6
 5: C ← 6
 6: se (∼ (C = 2)) então
 7:    se (C ≠ 3)  então
 8:       se (A = 2)  então
 9:          B ← A × 4
10:       senão
11:          B ← A - 5
12:       fimse
13:    senão
14:       se (A > C)  então
15:          C ← A + 3
16:       senão
17:          C ← C + 5
18:       fimse
19:    fimse
20: senão
21:    se (B = A)  então
22:       B ← B + B
23:    senão
24:       A ← A × B
25:    fimse
26: fimse
27: imprima A, B, C
28: fim{função}
```


Respostas

Exerc 1	Exerc 2	Exerc 3	Exerc 4	Exerc 5
Var. C	Var. B	Var. B	Var. C	Var. B

20.1.3 Respostas

```
   1
1.1   0
1.2 7
1.3 -1
1.4 10
1.5 4
   2
   2.1 9
2.2    2
2.3 28
2.4 27
2.5 -4
```

Capítulo 21

Exercícios Práticos: 024 -DVs

21.1 Dígitos Verificadores

Segurança da informação

Segurança de Informação está relacionada com a proteção existente ou necessária sobre dados que possuem valor para alguém ou uma organização.

Possui aspectos básicos como confidencialidade, integridade e disponibilidade da informação e não se aplica ou está restrita a sistemas computacionais, nem a informações eletrônicas ou qualquer outra forma mecânica de armazenamento. Ela se aplica a todos os aspectos de proteção e armazenamento de informações e dados, em qualquer forma.

Um dos padrões de segurança mais conhecidos é o BS7799, que estabelece melhores práticas para implementação e na gestão da segurança da informação. A série de normas ISO 27000, encabeçadas pela ISO 27001 estão sendo elaboradas para substituir e completar os padrões definidos pela BS7799.

Algumas normas definem aspectos que devem ser levados em consideração ao elaborar políticas de segurança. Entre essas normas estão a BS 7799 (elaborada pela British Standards Institution) e a NBR ISO/IEC 17799 (a versão brasileira desta primeira). A ISO começou a publicar a série de normas 27000, em substituição à ISO 17799 (e por conseguinte à BS 7799), das quais a primeira, ISO 27001, foi publicada em 2005.

Entende-se por informação todo e qualquer conteúdo ou dado que tenha valor para alguma organização ou pessoa. Ela pode estar guardada para uso restrito ou exposta ao público para consulta ou aquisição.

A segurança de uma determinada informação pode ser afetada por fatores comportamentais e de uso de quem se utiliza dela, pelo ambiente ou infra-estrutura que a cerca ou por pessoas mal intencionadas que tem o objetivo de furtar, destruir ou modificar a informação.

A tríade CIA (Confidentiality, Integrity and Availability) – Confidencialidade, Integridade e Disponibilidade – representa as principais propriedades que, atualmente, orientam a análise, o planejamento e a implementação da segurança para um determinado grupo de informações que se deseja protejer.

Outras propriedades estão sendo apresentadas (legitimidade e autenticidade) na medida em que o uso de transações comerciais em todo o mundo, através de redes eletrônicas (públicas ou privadas) se desenvolve.

- Confidencialidade - propriedade que limita o acesso a informação tão somente às entidades legítimas, ou seja, àquelas autorizadas pelo proprietário da informação.

- Integridade - propriedade que garante que a informação manipulada mantenha

todas as características originais estabelecidas pelo proprietário da informação, incluindo controle de mudanças e garantia do seu ciclo de vida (nascimento, manutenção e destruição).

- Disponibilidade - propriedade que garante que a informação esteja sempre disponívei para o uso legítimo, ou seja, por aqueles usuários autorizados pelo proprietário da informação.

Mecanismos de segurança O suporte para as recomendações de segurança pode ser encontrado em:

- Controles físicos: são barreiras que limitam o contato ou acesso direto a informação ou a infra-estrutura (que garante a existência da informação)que a suporta.

  Existem mecanismos de segurança que apóiam os controles físicos:

  Portas / trancas / paredes / blindagem / guardas / etc ..

- Controles lógicos: são barreiras que impedem ou limitam o acesso a informação, que está em ambiente controlado, geralmente eletrônico, e que, de outro modo, ficaria exposta a alteração não autorizada por elemento mal intencionado.

  Existem mecanismos de segurança que apóiam os controles lógicos:

  - Mecanismos de encriptação. Permitem a transformação reversível da informação de forma a torná-la ininteligível a terceiros. Utiliza-se para tal, algoritmos determinados e uma chave secreta para, a partir de um conjunto de dados não encriptados, produzir uma sequência de dados encriptados. A operação inversa é a desencriptação.
  - Assinatura digital. Um conjunto de dados encriptados, associados a um documento do qual são função, garantindo a integridade do documento associado, mas não a sua confidencialidade.
  - Mecanismos de garantia da integridade da informação. Usando funções de "Hashing"ou de checagem, consistindo na adição.
  - Mecanismos de controle de acesso. Palavras-chave, sistemas biométricos, firewalls, cartões inteligentes.
  - Mecanismos de certificação. Atesta a validade de um documento.
  - Integridade. Medida em que um serviço/informação é genuino, isto é, esta protegido contra a personificação por intrusos.

Existem duas filosofias por trás de qualquer política de segurança: a proibitiva (tudo que não é expressamente permitido é proibido) e a permissiva (tudo que não é proibido é permitido).

Integridade

- erros possíveis em digitação

erro	era	foi digitado
obliteração	1234	123
repetição	1234	12234
transposição	1234	1324
substituição	1234	7234
fraude	1234	5678

- Para minimizar estes erros, usa-se o conceito de dígito verificador. Trata-se de um dígito que é obtido como uma função matemática do código e é grudado nele.

  Pela simples análise do DV é possível detectar grande parte dos erros acima descritos.

- A função mais usada é MOD, já que é uma função de mão única. (não tem inversa) Por exemplo DOBRO de 2 é 4. Logo A METADE de 4 é 2. Já 8 MOD 7 é 1. Mas não é possível determinar x em X mod 7 = 1. x pode ser 8, 22, 29,...

Exemplos

UPC=Código de produto universal É o código de barras que encontramos nos produtos do supermercado. As 3 primeiras posições do código indicam o país de produção do bem. Neste caso, o Brasil é 789.

O último dígito (0 13^o) é assim obtido: 1. Soma-se os dígitos que ocupam posições ímpares (1, 3, 5...) e multiplica-se o resultado por 3. 2. Soma-se os dígitos que ocupam posições pares, e soma-se este resultado ao da etapa anterior 3. Subtrai-se o resultado do próximo múltiplo de 10. O resultado é o DV.

Sejam exemplos de um recipiente de tinta de carimbo, marca "Japan Stamp". Seu código é 7898076820584.

Calcule e ache o resultado correto (4).

Outro exemplo, um tubo de cola Tenaz. Seu código é 7891200304295.

Note-se que às vezes, a indústria procede diferente. Ao invés do código do produto ocupar as posições 4 a 12, sendo o DV a posição 13, o código do produto passa a ocupar as posições 5 a 13, e o DV (de maneira a que o último dígito do produto na posição 13) seja correto, o 5 dígito é convenientemente estabelecido.

ISBN 10 (International Standard Book Number) O último dígito da série de 10 do ISBN é o DV. Ele é calculado de maneira a que multiplicando cada dígito do código pela sua posição (começando da direita e em 1) e somando tudo, o resto desta soma por 11 deve ser 0.

Acompanhe o exemplo: Seja o ISBN 85-7001-926-? (idioma-editor-livro-dv). Multiplica-se 8 por 10, 5 por 9, 7 por 8, 1 por 5, 9 por 4, 2 por 3, 6 por 2 e o resultado é 240.

Dividindo 240 por 11 tem-se 21.8, e portanto o próximo inteiro divisível por 11 é 11 vezes 22, que é 242.

Fazendo-se 242-240=2 que é o DV procurado.

O ISBN 13 (em uso a partir de Janeiro de 2007) gera seu dígito da mesma maneira que o UPC.

Códigos de cartão de crédito Usa-se o algoritmo de Luhn, (Hans Peter Luhn, funcionário da IBM, 1896-1964), também conhecido como módulo 10, foi desenvolvido nos anos 60, como um método para validar códigos. Ele é usado nos números de cartão de crédito e no código de seguro social do Canadá. O algoritmo é de domínio público. Ele protege contra o erro acidental e não contra o ataque malicioso.

Começando com o dígito mais a direita (que é o DV), dobre o valor dos dígitos alternados. Para cada valor que ultrapasse 10, tome os seus dígitos juntos. Por exemplo 1111 gera 2121, enquanto 8763 gera 7733, (de 2x6=12 →1+2=3 e 2x8=16→1+6=7).

Some todos os dígitos juntos. Por exemplo, 1111 vira 2121 então 2+1+2+1 é 6. Já 8763 torna-se 7733, então 7+7+3+3 é 20.

Se o total termina em zero (é múltiplo de 10) o código completo é válido de acordo com a fórmula de Luhn, senão não é. Então 1111 não é válido enquanto 8763 é.

Para testar o algoritmo de Luhn, calcule o DV do cartão 4931 4701 2604 479?. A resposta deve ser 2.

Este algoritmo também é usado no sistema bancário da Noruega, no código ISSN (periódicos), no número de identificação do veículo em alguns estados americanos, no cartão de identificação de israelenses e no Yugoslav Unique Master Citizen Number (JMBG).

Código de barras da FEBRABAN Este código está presente no sistema bancário brasileiro e ele permite o intercâmbio de pagamentos entre os bancos (ou seja, um título de um banco pode ser pago em qualquer outro, até o vencimento).

Além do código de barras, adequado para uma leitura mecânica, existe um código numérico, que contém as mesmas informações, mas não na mesma ordem. Note-se também que os dígitos verificadores só aparecem no código numérico a digitar, mas não no código de barras. Aqui eles são desnecessários.

O código numérico é formado por 5 campos. O primeiro, de 10 dígitos é formado pelo código do banco (3 posições, B_1, B_2, B_3), código da moeda (9=real, M_4), e pelas posições L_{20}, L_{21},L_{22},L_{23},L_{24} do conteúdo livre (área do cliente), além de um DV obtido pelo método de base 10.

O segundo campo contendo 11 dígitos é formado pelo conjunto L_{25},...,L_{34} além de um novo DV base 10.

O terceiro campo contendo 11 dígitos é formado pelo conjunto L_{35},...,L_{44} além de um novo DV base 10.

O quarto campo é o DV do conjunto completo, calculado com o dígito 11 e pesos 2345678923... onde o 2 inicial corresponde ao caracter mais à direita. Nesta representação ele é o D_5.

Finalmente, o quinto campo é formado por 14 dígitos, sendo os 4 primeiros o fator de vencimento (dias decorridos desde 07/10/1997) até o dia de vencimento, ou seja F_6, F_7,F_8,F_9, e depois vem os 10 dígitos do valor, com centavos, ou seja, o V_{10}, V_{11}, V_{12}, V_{13}, V_{14}, V_{15}, V_{16}, V_{17}, V_{18}, V_{19}.

O código de barras, apresenta os dados na ordem em que eles foram apresentados acima, a saber:

B_1, B_2, B_3, M_4, D_5, F_6, F_7, F_8, F_9, V_{10}, V_{11}, V_{12}, V_{13}, V_{14}, V_{15}, V_{16}, V_{17}, V_{18}, V_{19}, L_{20}, L_{21},...,L_{44}. Além destes sinais, o código de barra começa por um start e termina por um stop. Este código é o chamado entrelaçãdo 5-2, já que os dígitos são calculados de 5 em 5 sinais, e tanto as barras quanto os espaços são usados para o reconhecimento.

No cálculo do DV mod 11, ao subtrair 11 menos o resto da divisão, se a resposta for 0, 1 ou maior do que 9, o dígito verificador é 1.

O que é obliteração ?

- 2 Na acepção 2 do Aurélio, obliterar é "destruir, eliminar, suprimir"e é sobre isto que nos fala esta história. Quantas vezes ao escrever uma palavra qualquer no micro ecrevemos (atenção revisão: é ecrevemos e não escrevemos) errado ? Você acabou de ver uma obliteração, alguém (meus dedos ?) sumiu com a letra "s"de escrevemos.

Isto é comum quando o pensamento é mais rápido que os dedos que digitam, isto é quase sempre. Via de regra, depois que se termina um texto, mandam os bons constumes que o digitador leia o que escreveu, e neste caso as obliterações são – em geral – descobertas e corrigidas. Quando o texto é muito importante, é comum pedir-se a outra pessoa que faça a revisão, já que muitos erros cometidos pelo digitador não podem ser facilmente descobertos por esse mesmo revisor. Há necessidade de um terceiro.

São cuidados mínimos para não pagar mico, e na sua falta, o implacável imponderável sempre dá as suas caras, como vai-se ver a seguir.

A cena é o exame final do curso de informática em uma universidade bem conceituada de Curitiba. Nesse curso, os alunos – para se formarem – precisam construir um software desde a sua concepção até sua operação sem nenhum erro grave. Eles têm 1 ano para essa tarefa e acreditem-me, geralmente é pouco tempo.

Há alguns anos, uma equipe estava na última banca para aprovação. Nós chamamos esta banca de magna, pois por ser a última é composta por todos os professores orientadores, que naquele ano eram em número de 9. A equipe em questão era formada por 3 alunas, bonitas, charmosas e também competentes, não esqueçamos de afirmar.

Estavam as 3 nervosas, roendo os dedos que as unhas já haviam acabado, numa espera imensa e infernal quando chegou a vez delas: Entraram na sala, instalaram o software, os micros, eram 3 que aquilo rodava em rede, o canhão, enfim todas essas parafernálias que todos tão bem conhecemos.

Rolava a apresentação, sem maiores problemas, o que por sí só já devia ser sério indício de que alguma coisa grave ia acontecer, essas coisas nunca rolam sem maiores problemas, quando... Mostrou-se uma transparência imensa cujo título era "processamento de pedidos", afinal o software era para automatizar uma pequena lojinha de bairro.

Nessa hora a obliteração (a maldita) introduziu-se na história. O digitador (o maldito) esquecera-se de digitar, obliterara uma letra no título. Maior gravidade não haveria se fosse qualquer letra, mas qual o quê: a letra roubada fora a terceira letra da terceira palavra, as alunas estavam apresentando uma transparência onde em letras garrafais se dizia "processamento de peidos".

Não preciso descrever como terminou a banca. Apenas informo que as alunas foram aprovadas, o software estava muito bom.

Pedro Kantek

Os principais métodos usados:

21.1.1 Módulo 10

- Separe os dígitos do código a processar. Ex: se o código é 13865, tem-se $d_1 = 1$, $d_2 = 3$, $d_3 = 8$, $d_4 = 6$ e $d_5 = 5$.
- Crie um vetor de pesos P, com o mesmo número de dígitos, e contendo 2 e 1, começando com 2 e alternando. Ex: $p_1 = 2, p_2 = 1, p_3 = 2, p_4 = 1, p_5 = 2$.
- Multiplique os dois vetores, tirando NOVES FORA em cada multiplicação. Ex: $m_1 = 1 \times 2 = 2$, $m_2 = 3 \times 1 = 3$, $m_3 = 8 \times 2 = 16$, noves fora=7, $m_4 = 1 \times 6 = 6$, $m_5 = 2 \times 5 = 1$
- Some os elementos do vetor multiplicação. Ex:$2 + 3 + 7 + 6 + 1 = 19$
- O que faltar para completar a próxima dezena será o DV *mod* 10. No exemplo, o que falta a 19 para completar 20 é 1. Logo $DV = 1$.
  Obs: 1: noves fora: se $x > 9$, $x \leftarrow x - 9$;
  2:se deu a dezena, a resposta é 0.

21.1.2 Módulo 11

- Separe os dígitos.
- Crie vetor P. O menor peso é 2, alocado ao dígito mais a direita. A seqüência de pesos cresce para a esquerda. Ex: $7, 6, 5, 4, 3, 2$. A lei de formação destes pesos PODE VARIAR em cada método.
- O $DV = 11-$ resto da soma dividido por 11.

- Se o resto é 0 ou 1, o DV é igual a 0. (No BB se R=0, DV=X)

Letra Chave

Este método tem como resposta uma letra e não um dígito.

- Separe os dígitos e crie um vetor de pesos adequado.
- Some e ache o resto de uma divisão por 26.
- A LC é obtida pela equivalência: $R = 0$, $LC = A$; $R = 1$, $LC = B$; ... $R = 25$,$LC = Z$.

Este método se encontra em desuso. Foi usado na Receita do PR até 96.

21.1.3 CPF (Cadastro de Pessoas Físicas)

a) São 2 DVs.
b) O primeiro é calculado pelo MOD 11, com pesos $= 10, 9, 8, 7, 6, 5, 4, 3, 2$.
c) O DV calculado é colocado no seu lugar e o processo refeito, agora com os pesos: $11, 10, 9, 8, 7, 6, 5, 4, 3, 2$.
d) O segundo dígito é colocado no seu lugar.
Obs: O nono dígito de um CPF é a região onde foi criado. 6=MG, 7=ES/RJ, 8=SP, 9=PR/SC e 0=RS. Por exemplo, seja o $CPF = 176.294.338$, quais os DVs ? $1 \times 10 + 7 \times 9 + 6 \times 8 + 2 \times 7 + 9 \times 6 + 4 \times 5 + 3 \times 4 + 3 \times 3 + 8 \times 2 = 246$, cuja divisão por 11 tem como resto 4. Logo o primeiro DV é $11 - 4 = 7$.
Refazendo $1 \times 11 + 7 \times 10 + 6 \times 9 + 2 \times 8 + 9 \times 7 + 4 \times 6 + 3 \times 5 + 3 \times 4 + 8 \times 3 + 7 \times 2 = 303$,cujo resto da divisão por 11 é 6. Assim, o DV é $11 - 6 = 5$. O CPF completo fica sendo $176.294.338 - 75$.

Curiosidade: CPF = 111 111 111, DVs=1,1. CPF = 222 222 222 - 22, 333 333 333 - 33, 444 444 444 - 44, e assim por diante até 999 999 999 - 99 e 000 000 000 - 00.

Veja se acerta:
CPF = 357 432 754 - _____ - _____. Resposta: 40.
CPF = 247 212 764 - _____ - _____. Resposta: 27.

21.1.4 CNPJ (antigo CGC)

a) São 3 DVs.
b) O DV1 ocupa a 8.posição do código e é um MOD 10 dos 7 dígitos iniciais.
c) O DV2 ocupa a 13.posição e é um MOD 11 de todos os anteriores com o vetor de $pesos = 543298765432$. d) O DV3 ocupa a 14.posição e é um MOD 11 de todos os anteriores com os $pesos = 6543298765432$.

Exemplo, seja o CNPJ da COPEL: 7 648 381. O $DV1 = 7 \times 2 = 14$, noves fora $= 5 + 6 \times 1 = 6$, $+4 \times 2 = 8$, $+8 \times 1 = 8$, $+3 \times 2 = 6$, $+8 \times 1 = 8$, $+1 \times 2 = 2$. A soma é 43, e o DV1 é o que falta para a próxima dezena, NESTE CASO 50, ou DV1=7. Com isso o primeiro código é: 76.483.817. Continuando:
A filial é 0001, e fica: 76.483.817/0001.

O DV2 é igual a: $= 7 \times 5 + 6 \times 4 + 4 \times 3 + 8 \times 2 + 3 \times 9 + 8 \times 8 + 1 \times 7 + 7 \times 6 + 0 \times 5 + 0 \times 4 + 0 \times 3 + 1 \times 2 = 229$. O DV2 é 11 menos o resto de 229 por 11 que é 2.

DV3=$7 \times 6 + 6 \times 5 + 4 \times 4 + 8 \times 3 + 3 \times 2 + 8 \times 9 + 1 \times 8 + 7 \times 7 + 0 \times 6 + 0 \times 5 + 0 \times 4 + 1 \times 3 + 2 \times 2 =$ 254. O resto de 254 dividido por 11 é 1, logo o DV3=11-1=10 ou zero. Assim, o CNPJ completo da COPEL é: 76.483.817/0001-20.

Veja se acerta os CNPJs a seguir:
Carrefour: 45.543.91___/0001 - ___ - ___. Resposta:581.
Lacta: 57.003.88___/0061- ___-___ Resposta:152.
BC: 00.038.16___/0001- ___-___ Resposta:605.

Observação: Alguns CNPJs inexplicavelmente não seguem o aqui escrito. Aparentemente foram criados antes da regra. Por exemplo, o CGC da Light: 66 444 437/0001-46.

21.2 Exercício 1

Calcule os dígitos dos Cadastros de Pessoa Física:
1= 3 5 6 9 8 2 1 0 9 - ___ ___.
2= 8 8 9 4 0 5 5 3 1 - ___ ___.
Calcule os dígitos dos seguintes CNPJs:
3= 7 4 7 7 8 3 4 ___ / 0 0 0 9 4 - ___-___.
4= 8 1 6 0 4 0 0 ___ / 0 0 0 6 9 - ___-___.

Some os 10 dígitos achados : _____

Exercício 2

Calcule os dígitos dos Cadastros de Pessoa Física:
1= 7 1 2 5 3 4 1 7 8 - ___ ___.
2= 6 2 5 4 9 3 2 4 3 - ___ ___.
Calcule os dígitos dos seguintes CNPJs:
3= 3 0 6 8 0 0 2 ___ / 0 0 0 5 - ___-___.
4= 5 2 1 3 4 8 6 ___ / 0 0 0 9 - ___-___.

Some os 10 dígitos achados : _____

Exercício 3

Calcule os dígitos dos Cadastros de Pessoa Física:
1= 5 5 2 9 5 1 3 3 4 - ___ ___.
2= 4 3 1 7 3 8 3 0 2 - ___ ___.
Calcule os dígitos dos seguintes CNPJs:
3= 5 6 5 2 2 6 4 ___ / 0 0 0 4 5 - ___-___.
4= 2 4 8 7 9 6 3 ___ / 0 0 0 4 2 - ___-___.

Some os 10 dígitos achados : _____

Exercício 4

Calcule os dígitos dos Cadastros de Pessoa Física:
1= 5 5 1 5 9 3 9 0 8 - ___ ___.
2= 8 6 5 4 7 1 0 3 3 - ___ ___.
Calcule os dígitos dos seguintes CNPJs:
3= 8 0 9 9 4 8 2 ___ / 0 0 0 9 3 - ___-___.
4= 9 8 8 2 7 6 7 ___ / 0 0 0 5 9 - ___-___.

Some os 10 dígitos achados : _____

Exercício 5

Calcule os dígitos dos Cadastros de Pessoa Física:
1= 3 7 6 9 5 4 9 4 4 - ___ ___.

2= 7 6 8 4 0 6 7 2 6 - ___ ___.
Calcule os dígitos dos seguintes CNPJs:
3= 6 6 1 6 2 3 4 ___ / 0 0 0 9 - ___-___.
4= 4 6 2 1 4 6 9 ___ / 0 0 0 8 - ___-___.

SOME OS 10 DÍGITOS ACHADOS : _____

Exercício 6

Calcule os dígitos dos Cadastros de Pessoa Física:
1= 4 9 6 9 5 5 5 0 2 - ___ ___.
2= 8 5 3 7 8 0 6 4 3 - ___ ___.
Calcule os dígitos dos seguintes CNPJs:
3= 6 1 5 6 4 0 3 ___ / 0 0 4 1 - ___-___.
4= 3 4 8 9 2 5 7 ___ / 0 0 0 2 - ___-___.

SOME OS 10 DÍGITOS ACHADOS : _____

Exercício 7

Calcule os dígitos dos Cadastros de Pessoa Física:
1= 6 6 7 6 1 6 4 2 2 - ___ ___.
2= 6 0 6 9 9 1 4 9 5 - ___ ___.
Calcule os dígitos dos seguintes CNPJs:
3= 6 6 9 5 0 0 3 ___ / 0 0 0 5 - ___-___.
4= 3 3 9 7 7 2 1 ___ / 0 0 0 3 - ___-___.

SOME OS 10 DÍGITOS ACHADOS : _____

Exercício 8

Calcule os dígitos dos Cadastros de Pessoa Física:
1= 5 2 2 8 4 2 5 3 4 - ___ ___.
2= 6 8 2 9 1 8 2 0 6 - ___ ___.
Calcule os dígitos dos seguintes CNPJs:
3= 7 2 0 8 7 2 7 ___ / 0 0 0 6 - ___-___.
4= 6 0 1 1 5 1 5 ___ / 0 0 2 8 - ___-___.

SOME OS 10 DÍGITOS ACHADOS : _____

Exercício 9

Calcule os dígitos dos Cadastros de Pessoa Física:
1= 8 0 5 4 5 2 6 7 3 - ___ ___.
2= 8 1 5 9 7 1 9 3 1 - ___ ___.
Calcule os dígitos dos seguintes CNPJs:
3= 6 0 8 8 8 3 2 ___ / 0 0 0 5 - ___-___.
4= 9 2 6 9 8 9 4 ___ / 0 0 8 6 - ___-___.

SOME OS 10 DÍGITOS ACHADOS : _____

Exercício 10

Calcule os dígitos dos Cadastros de Pessoa Física:
1= 4 5 1 2 4 4 4 0 2 - ___ ___.
2= 4 1 7 0 9 9 8 8 6 - ___ ___.
Calcule os dígitos dos seguintes CNPJs:

3= 5 8 1 2 1 0 9 ___ / 0 0 4 3 - ___-___.
4= 4 6 3 8 4 6 2 ___ / 0 0 3 9 - ___-___.

SOME OS 10 DÍGITOS ACHADOS : _____

21.2.1 Respostas

1	33	9301128135
2	50	0615956369
3	35	4302226736
4	47	5507526863
5	50	9654840815
6	43	8292535036
7	30	1333145622
8	39	9094330137
9	31	3100841347
10	38	5230685423

Capítulo 22

Exercícios Práticos: 027 - aritmética não decimal

22.1 Aritmética não decimal

A aritmética, tal como a conhecemos, não sofre nenhuma mudança conceitual ou operacional se passarmos a usar bases não decimais. O sistema de numeração usado, desde que posicional, funciona legal.

Para as operações a seguir, considere um conjunto de dígitos sujeito as seguintes regras:

1. O primeiro digito é sempre ZERO.
2. Existem tantos dígitos quanto é o valor da base.
3. Se forem necessários mais de 10 dígitos, o 11^o será a letra A, o 12^o a letra B e assim por diante até o 36^o dígito que será o Z.

Na soma individual, dígito a dígito, a regra é simples: se a soma for maior que a base, retira-se uma base do total e vai um...

Na subtração individual, dígito a dígito, se o número a subtrair é maior do que o número do qual vai ser feita a subtração, empresta-se uma base do vizinho esquerdo.

Exemplos:

```
27(10) + 45(10) =      2 7
                     +  4 5
                       -----
                        7 2
```

Como $7 + 5 = 12$ e 12 é maior do que a base, subtrai-se uma base (10) e o resultado é 2, e "vai um". A resposta final é 72.

```
94(10) - 59(10) = 9 4
                -    5 9
                    -----
                     3 5
```

Como $4 - 9$ não pode ser feito (pois $9 > 4$), a solução é emprestar uma base do vizinho. Com isto, tem-se $4 + 10 = 14$, e agora pode-se fazer $14 - 9$, cujo resultado é 5.

O um que foi emprestado do 9, deixa-o valendo apenas 8, e agora tem-se $8 - 5 = 3$. O resultado final é 35.

Para facilitar, use a seguinte tabela:

```
0..9 = 0..9   E = 14   J = 19   O = 24   T = 29  Y = 34
   A = 10     F = 15   K = 20   P = 25   U = 30  Z = 35
   B = 11     G = 16   L = 21   Q = 26   V = 31
   C = 12     H = 17   M = 22   R = 27   W = 32
   D = 13     I = 18   N = 23   S = 28   X = 33
```

Nos exercícios a seguir, tente fazer as conversões e as operações solicitadas:

```
1. Conversoes de Base Q para base 10

00122(18) = _____________________(10)

000O7(32) = _____________________(10)

00510(12) = _____________________(10)

001C4(24) = _____________________(10)

02120(07) = _____________________(10)

2. Conversoes de base 10 para base Q

00868(10) = _____________________(22)

00772(10) = _____________________(18)

00642(10) = _____________________(21)

00703(10) = _____________________(09)

00232(10) = _____________________(25)

3. Somas em base diferente de 10

01402(06) + 01113(06) = _____________(06)

01343(05) + 01104(05) = _____________(05)

001A1(15) + 001B9(15) = _____________(15)

0013I(22) + 00103(22) = _____________(22)

000I2(19) + 00138(19) = _____________(19)

4. Subtraçoes em base diferente de 10

00176(18) - 00128(18) = _____________(18)

00297(12) - 00181(12) = _____________(12)

000LF(26) - 000GP(26) = _____________(26)

000DD(27) - 00080(27) = _____________(27)
```

```
00671(09) - 00271(09) = _________________(09)
```

22.1.1 Programa VISUALG

A seguir um programa escrito em VISUALG que resolve as adições e subtrações em bases não decimais. Note-se que o programa não faz nenhuma consistência, e se algum erro aparecer, o programa abortará. Eis alguns dos possíveis erros:

- Usar números de comprimento maior que 9 dígitos
- Usar bases maiores do que 36
- Escrever operações de subtração cujo resultado seja negativo
- Incluir caracteres diferentes de "0..Z"nos operandos
- Usar operações diferentes de + e -
- Usar espaços em branco entre os dígitos dos operandos
- Usar digitos de valor absoluto maior ou igual do que a base

```
algoritmo "maismenosnaodecimal"
var
sinal: caracter
c1, c2 : caracter
b : inteiro
op1 : vetor [1..10] de inteiro
op2 : vetor [1..10] de inteiro
res : vetor [1..10] de inteiro
let : caracter
i, j : inteiro
funcao achaind (ind:caracter) : inteiro
var
k : inteiro
inicio
k <- 1
enquanto (copia(let;k,1) <> ind) faca
   k <- k + 1
fimenquanto
retorne k
fimfuncao
inicio
let <- "0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ"
escreval ("Soma e subtracao de numeros de base qualquer - v.1.1 - P.kantek - ago/07")
escreval ("escreva o operador 1: ")
leia (c1)
escreval ("mais (+) ou menos (-) ? ")
leia(sinal)
escreval ("escreva o operador 2: ")
leia (c2)
escreval ("a base: ")
leia(b)
```



```
i <- compr(c1)
j <- 10
enquanto (i >= 1) faca
  op1[j] <- achaind (copia(c1;i,1)) - 1
  i <- i - 1
  j <- j - 1
fimenquanto
i <- compr(c2)
j <- 10
enquanto (i >= 1) faca
  op2[j] <- achaind (copia(c2;i,1)) - 1
  i <- i - 1
  j <- j - 1
fimenquanto
se sinal = "+" entao
   i <- 10
   enquanto (i>=1) faca
      Se (op1[i] + op2[i]) >= b entao
         res[i] <- op1[i] + op2[i] - b
         op1[i-1] <- op1[i-1] + 1
      senao
         res[i] <- op1[i] + op2[i]
      fimse
      i <- i - 1
   fimenquanto
senao
   i <- 10
   enquanto (i>=1) faca
      Se op1[i]<op2[i] entao
         res[i] <- b + op1[i] - op2[i]
         op1[i-1] <- op1[i-1] - 1
      senao
         res[i] <- op1[i] - op2[i]
      fimse
      i <- i - 1
   fimenquanto
fimse
para i de 1 ate 10 faca
escreva (copia(let;(res[i]+1),1))
fimpara
fimalgoritmo
```

Algoritmo de conversão de base qualquer para base 10

```
algoritmo "conversãoQ10"
var
c1 : caracter
b : inteiro
op1 : vetor [1..10] de inteiro
let : caracter
i, j : inteiro
resu : real
funcao achaind (ind:caracter) : inteiro
```

```
var
k : inteiro
inicio
k <- 1
enquanto (copia(let;k,1) <> ind) faca
   k <- k + 1
fimenquanto
retorne k
fimfuncao
inicio
let <- "0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ"
escreval ("Conversao de numeros de base qualquer para base 10 - v.1.1 - P.kantek - ago/07")
escreval ("escreva o valor a converter ")
leia (c1)
escreval ("a base: ")
leia(b)

i <- compr(c1)
j <- 10
enquanto (i >= 1) faca
  op1[j] <- achaind (copia(c1;i,1)) - 1
  i <- i - 1
  j <- j - 1
fimenquanto
resu <- 0
para i de 1 ate 10 faca
   resu <- resu + op1[i] * b ^ (10-i)
fimpara
escreva (resu)
fimalgoritmo
```

Algoritmo de conversão de base 10 para base qualquer

```
algoritmo "conversão10Q"
var
c1 : inteiro
b : inteiro
op1 : vetor [1..10] de inteiro
let : caracter
i : inteiro
resto : inteiro
funcao achaind (ind:caracter) : inteiro
var
k : inteiro
inicio
k <- 1
enquanto (copia(let;k,1) <> ind) faca
   k <- k + 1
fimenquanto
retorne k
fimfuncao
inicio
let <- "0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ"
```



```
escreval ("Conversao de numeros de base 10 para base qualquer - v.1.1 - P.kantek - ago/07")
escreval ("escreva o valor a converter ")
leia (c1)
escreval ("a base: ")
leia(b)
resto <- 1
i <- 10
enquanto (resto <> 0) faca
   resto <- c1 mod b
   op1[i] <- resto
   c1 <- c1 div b
   i <- i - 1
fimenquanto

para i de 1 ate 10 faca
escreva (copia(let;(op1[i]+1),1))
fimpara

fimalgoritmo
```

22.2 Exercício 1

1. Conversões de Base Q para base 10

   460 $_{(15)}$=_____$_{(10)}$

   A3 $_{(27)}$=_____$_{(10)}$

2. Conversões de base 10 para base Q

   00888 $_{(10)}$ = _____$_{(11)}$

   00588 $_{(10)}$ = _____$_{(12)}$

3. Somas em base diferente de 10

   G4 $_{(29)}$ + GN $_{(29)}$ = _____$_{(29)}$

   163 $_{(19)}$ + 1A6 $_{(19)}$ = _____$_{(19)}$

4. Subtrações em base diferente de 10

   143 $_{(17)}$ - F8 $_{(17)}$ = _____$_{(17)}$

   13A $_{(20)}$ - I0 $_{(20)}$ = _____$_{(20)}$

22.3 Exercício 2

1. Conversões de Base Q para base 10

   175 $_{(28)}$=_____$_{(10)}$

   535 $_{(15)}$=_____$_{(10)}$

2. Conversões de base 10 para base Q

   00939 $_{(10)}$ = _____$_{(13)}$

   00319 $_{(10)}$ = _____$_{(32)}$

3. Somas em base diferente de 10

   108 $_{(22)}$ + 7F $_{(22)}$ = _____$_{(22)}$

   IM $_{(29)}$ + 7A $_{(29)}$ = _____$_{(29)}$

4. Subtrações em base diferente de 10

   210 $_{(13)}$ - 151 $_{(13)}$ = _____$_{(13)}$

   255 $_{(11)}$ - 172 $_{(11)}$ = _____$_{(11)}$

22.4 Exercício 3

1. Conversões de Base Q para base 10

   289 $_{(19)}$=_____$_{(10)}$

   42A $_{(11)}$=_____$_{(10)}$

2. Conversões de base 10 para base Q

   00783 $_{(10)}$ = _____$_{(13)}$

   00212 $_{(10)}$ = _____$_{(30)}$

3. Somas em base diferente de 10

   7J $_{(24)}$ + D3 $_{(24)}$ = _____$_{(24)}$

   152 $_{(14)}$ + 180 $_{(14)}$ = _____$_{(14)}$

4. Subtrações em base diferente de 10

   1E0 $_{(18)}$ - 173 $_{(18)}$ = _____$_{(18)}$

   JD $_{(25)}$ - BG $_{(25)}$ = _____$_{(25)}$

22.4.1 Respostas

```
1 990 273 738  410  13R  2G9  5C   5A
2 985 1175 573  9V   181  Q3   8C   93
3 883 516 483  72   KM   2D2  6F   7M
```

Capítulo 23

Exercícios Práticos: 031 - Manipulação de datas

23.1 Algoritmos de Calendário

Série de algoritmos e truques para calcular dias de semana e datas de feriados móveis em nosso calendário.

23.1.1 Cálculo do dia da semana

O algoritmo seguinte é devido ao astrônomo napolitano Aloysius Lilius e ao matemático alemão e jesuita Cristopher Clavius. Escrito no século XVI é usado pelas igrejas ocidentais para calcular o dia do domingo de Páscoa. Existiram outros algoritmos antes deste. Por exemplo o *Canon Paschalis* devido a Victorius de Aquitania escrito em cerca de 450 a.C.

Dada uma data no formato dia, mes, ano (onde ano > 1587), calcula-se o dia da semana usando a seguinte formulação:
$A \leftarrow \lfloor((12 - mes) \div 10)$
$B \leftarrow ano - A$
$C \leftarrow mes + (12 \times A)$
$D \leftarrow \lfloor(B \div 100)$
$E \leftarrow \lfloor(D \div 4)$
$F \leftarrow E + 2 - D$
$G \leftarrow \lfloor(365.25 \times B)$
$H \leftarrow \lfloor(30.6001 \times (C + 1))$
$I \leftarrow F + G + H + dia + 5$
$R \leftarrow I \ mod \ 7$

Se $R = 0$, dia, mes, ano é sábado, $R = 1$ é domingo, $R = 2$ é segunda, $R = 3$ é terça $R = 4$ é quarta, $R = 5$ é quinta e $R = 6$ é sexta-feira.

Exemplo: Calculemos o dia da semana de hoje,
dia ____ / ____ / ____

A ← _____ F ← _____

B ← _____ G ← _____

C ← _____ H ← _____

D ← _____ I ← _____

E ← _____ R ← _____

como $R =$_____, o dia em questão é _____.

23.1.2 Cálculo dos feriados móveis

Os 3 feriados móveis (terça de carnaval, sexta feira santa e Corpus Christi) são baseados todos no dia do domingo de Páscoa. Portanto, a primeira coisa a fazer é calcular em que dia cai a Páscoa.

Dado um ano com quatro digitos (maior que 1587), a Páscoa é:
$A \leftarrow ano \; mod \; 19$
$B \leftarrow \lfloor (ano \div 100)$
$C \leftarrow ano \; mod \; 100$
$D \leftarrow \lfloor (B \div 4)$
$E \leftarrow B \; mod \; 4$
$F \leftarrow \lfloor (B + 8) \div 25$
$G \leftarrow \lfloor (1 + B - F) \div 3$
$H \leftarrow ((19 \times A) + B + 15 - (D + G)) \; mod \; 30$
$I \leftarrow \lfloor (C \div 4)$
$K \leftarrow C \; mod \; 4$
$L \leftarrow (32 + (2 \times E) + (2 \times I) - (H + K)) \; mod \; 7$
$M \leftarrow \lfloor ((A + (11 \times H) + (22 \times L)) \div 451)$
$P \leftarrow \lfloor ((H + L + 114 - (7 \times M)) \div 31)$
$Q \leftarrow (H + L + 114 - (7 \times M)) \; mod \; 31$

A Páscoa é o dia Q+1 do mês P.

23.1.3 Bissexto

A regra do bissexto pode ser assim descrita: sejam
R4 ← resto da divisão do ano por 4
R100 ← resto da divisão do ano por 100 e
R400 ← resto da divisão do ano por 400.
SE R4=0 ∧ ((R100 ≠ 0) ∨ (R400 = 0)) o ano é bissexto senão não é.

Outra maneira de descrever o algoritmo é usando SEs encadeados

```
1:  se (ANO mod 400) = 0 então
2:     ... é bissexto
3:  senão
4:     se (ANO mod 100) = 0 então
5:        ... NÃO é bissexto
6:     senão
7:        se (ANO mod 4) = 0 então
8:           ...é bissexto
9:        senão
10:          NÃO é bissexto
11:       fimse
12:    fimse
```

13: **fimse**

Se lembrarmos que os dias de cada mês são 31, (28/29), 31, 30, 31, 30, 31, 31, 30, 31, 30, 31. A partir daqui pode-se criar um vetor auxiliar VA com os seguintes valores $VA \leftarrow 1, 32, 60, 91, 121, 152, 182, 213, 244, 274, 305, 335$ (anos não bissextos) ou $VA \leftarrow 1, 32, 61, 92, 122, 153, 183, 214, 245, 275, 306, 336$ (anos bissextos) e calcularmos $dc \leftarrow VA[P] + Q$ então dc é o dia absoluto a partir de 1/janeiro desse ano.

A Sexta feira santa é $dc - 2$, o carnaval é $dc - 47$ e Corpus Christi é $dc + 60$ Note que os dias da semana sempre serão sexta, terça e quinta respectivamente.

$dccarn \leftarrow dc - 47$, $dcsexs \leftarrow dc - 2$ e $dccorp \leftarrow dc + 60$

O mês de cada uma dessas datas, deve ser lido em VA vis-a-vis o valor de dcxxxx. Assim, se dcxxxx é 94, essa data corresponde a 4/abril, para um ano não bissexto.

Mais formalmente: A passagem de dd,mm,aa para dc é $dc = VA[mes] + dia - 1$. A passagem de dc para dd,mm,aa é: O dia corresponde a dc MENOS o maior elemento de VA menor ou igual a dc MAIS 1. Por exemplo, se dc é 100 e VA é 1 32 60 91 121... o dia será (100-91)+1= dia 10. O mês, por sua vez, corresponde ao índice do elemento acima.

Exemplo: Vamos calcular as 4 datas do ano de 2007 .

A ← _____	K ← _____
B ← _____	L ← _____
C ← _____	M ← _____
D ← _____	P ← _____
E ← _____	Q ← _____
F ← _____	dc ← _____
G ← _____	dccarn ←
H ← _____	dcsexs ← _____
I ← _____	dccorp ←

23.2 Exercício 1

1. Calcule dia da semana do dia 25/10/2112 e informe:_____ (0=sab, 1=dom,...6=sex)

2. Calcule o Carnaval de 2112 _____ / _____

E o Corpus-Christi de 2112 _____ / _____

23.3 Exercício 2

1. Calcule dia da semana do dia 28/ 2/1605 e informe:_____ (0=sab, 1=dom,...6=sex)

2. Calcule o Carnaval de 1605 _____ / _____

E o Corpus-Christi de 1605 _____ / _____

23.4 Exercício 3

1. Calcule dia da semana do dia 21/ 7/1893 e informe:_____ (0=sab, 1=dom,...6=sex)

2. Calcule o Carnaval de 1893 _____ / _____

E o Corpus-Christi de 1893 _____ / _____

23.5 Exercício 4

1. Calcule dia da semana do dia 10/ 7/2143 e informe:_____ (0=sab, 1=dom,...6=sex)

2. Calcule o Carnaval de 2143 _____ / _____

E o Corpus-Christi de 2143 _____ / _____

23.6 Exercício 5

1. Calcule dia da semana do dia 20/12/2131 e informe:_____ (0=sab, 1=dom,...6=sex)

2. Calcule o Carnaval de 2131 _____ / _____

E o Corpus-Christi de 2131 _____ / _____

23.7 Exercício 6

1. Calcule dia da semana do dia 17/10/2180 e informe:_____ (0=sab, 1=dom,...6=sex)

2. Calcule o Carnaval de 2180 _____ / _____

E o Corpus-Christi de 2180 _____ / _____

23.8 Exercício 7

1. Calcule dia da semana do dia 11/ 4/2114 e informe:_____ (0=sab, 1=dom,...6=sex)

2. Calcule o Carnaval de 2114 _____ / _____

E o Corpus-Christi de 2114 _____ / _____

23.9 Exercício 8

1. Calcule dia da semana do dia 16/ 7/1941 e informe:_____ (0=sab, 1=dom,...6=sex)

2. Calcule o Carnaval de 1941 _____ / _____

E o Corpus-Christi de 1941 _____ / _____

23.10 Exercício 9

1. Calcule dia da semana do dia 20/ 6/1884 e informe:_____ (0=sab, 1=dom,...6=sex)

2. Calcule o Carnaval de 1884 _____ / _____

E o Corpus-Christi de 1884 _____ / _____

23.11 Exercício 10

1. Calcule dia da semana do dia 29/10/1999 e informe:_____ (0=sab, 1=dom,...6=sex)

2. Calcule o Carnaval de 1999 _____ / _____

E o Corpus-Christi de 1999 _____ / _____

23.12 Respostas

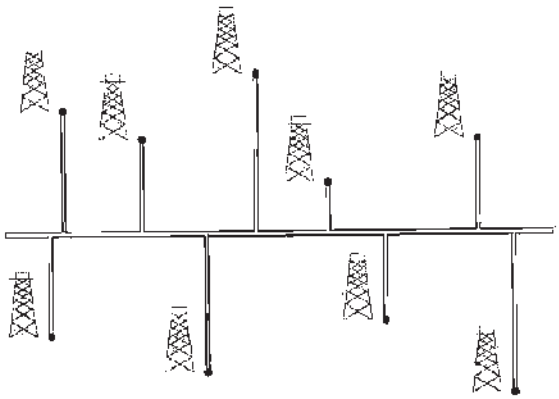
```
1     3 (terça)      1/3/2112   16/6/2112
2     2 (segunda)   22/2/1605    9/6/1605
3     6 (sexta)     14/2/1893    1/6/1893
4     4 (quarta)    12/2/2143   30/5/2143
5     5 (quinta)    27/2/2131   14/6/2131
6     3 (terça)     29/2/2180   15/6/2180
7     4 (quarta)     6/3/2114   21/6/2114
8     4 (quarta)    25/2/1941   12/6/1941
9     6 (sexta)     26/2/1884   12/6/1884
10    6 (sexta)     16/2/1999    3/6/1999
```


Capítulo 24

Exercícios Práticos: 035 - Localização de oleoduto

24.1 Localização de oleoduto

Suponha que você tem um terreno de 100 $km \times 100$ km contendo um certo número de poços de petróleo. O terreno irá ser atravessado integralmente por um duto mestre, ao qual irão se ligar (sempre de maneira transversal, isto é, fazendo 90 graus) ramais auxiliares que conectarão cada um dos poços ao duto mestre.



O objetivo do exercício é descobrir por onde deverá passar o duto mestre, de maneira a minimizar a quantidade de duto auxiliar necessário. O duto mestre sempre será instalado no sentido NORTE-SUL ou no LESTE-OESTE, conforme explicitado no exercício. Outra coisa que você deve calcular é a quantidade de duto auxiliar necessário (medido em kilômetros).

Os poços serão identificados pelas suas coordenadas (x, y), sendo que a origem dos índices, o ponto $0, 0$, estará localizado no canto inferior esquerdo do terreno. A primeira medida de cada poço (o x) indicará a distância no sentido leste-oeste e o segundo número (o y) indicará a distância no norte-sul.

Assim, se um poço estiver nas coordenadas (12,78), ele se encontrará 12 metros à direita da origem e 78 metros acima da mesma origem.

Veja-se o seguinte exemplo: Suponha-se a seguinte ditribuição de poços:

x	11	10	17	5	16	13	1	26	15
y	9	26	19	14	22	20	12	4	25

Supondo que o duto principal deva ser extendido no sentido leste-oeste, em qual ordenada ele deveria estar ?

Proposta de solução

1. Como o sentido é leste-oeste, podem-se desprezar as abcissas.

2. Ficando apenas com as ordenadas, têm-se: $9, 26, 19, 14, 22, 20, 12, 4, 25$.

3. Deslocando a origem dos indices, a fim de diminuir o trabalho braçal por meio de uma translação, fica: $5, 22, 15, 10, 18, 16, 8, 0, 21$

4. É razoável se supor que o duto esteja entre o ponto 0 e o ponto 22. Na realidade, entre 4 e 26, antes da translação.

5. Para descobrir onde deve passar suponha-o localizado em cada um dos 9 poços e meça a quantidade de duto auxiliar necessário. Neste caso temos: $80, 83, 54, 59, 61, 55, 65, 115, 76$

6. Como se pode ver o menor valor é de 54 km, correspondendo à ordenada 19, do terceiro poço.

7. A resposta esperada, portanto é 54,19, significando que são necessários 54m de duto auxiliar e que o duto principal deve estar entre os pontos 0,19 e 100,19.

Se o mesmo valor mínimo for encontrado para 2 poços, a resposta deverá ser a média entre as duas ordenadas. Por exemplo, no problema

x	19	10	4	1	7	18	8	21
y	13	12	17	5	23	20	16	11

Percebe-se que as distâncias mínimas são: $35, 37, 37, 77, 67, 49, 35, 41$ que correspondem aos poços 1 e 7. Com isso, a ordenada do duto principal deve ser a média das duas ordenadas dos poços: $13 + 16 = 29 \div 2 = 14.5\ km$. A resposta deveria ser 35, 14.5. Lembrando, o valor 35 corresponde a quantidade de duto auxiliar necessária.

Observações

Lembre que se o duto principal estiver na direção norte-sul, vale o mesmo raciocínio visto acima, apenas mudando "ordenadas" por "abcissas", e desprezando-se o segundo número das coordenadas de cada poço.

Finalmente, o algoritmo aqui esboçado, a quem poderíamos chamar de "força bruta", funciona e apresenta o resultado correto. Entretanto você é livre para procurar outro algoritmo que seja mais eficiente, (mas que funcione também, é claro.)

24.2 Exercícios

24.3 Exercício 1

Suponha o duto principal no eixo L-W . Os poços se encontram nas seguintes posições:

x	18	4	6	1	23	15	21	13	22	14
y	19	3	10	29	26	16	5	11	9	30

Para este caso, a metragem de duto auxiliar é de _____
e a posição do duto principal deve ser _____.

Mais, um:
Suponha o duto principal no eixo N-S . Os poços se encontram nas seguintes posições:

x	20	1	2	10	9	5	16	13	24
y	6	3	21	18	11	27	15	17	8

Para este caso, a metragem de duto auxiliar é de _____
e a posicao do duto principal deve ser _____.

Este exercício foi adaptado de CORMEN, T. et alli, Algoritmos, pág 155 (edição brasileira).

24.4 Exercício 2

Suponha o duto principal no eixo N-S . Os poços se encontram nas seguintes posições:

x	16	5	14	24	23	1	6	21
y	15	2	22	8	18	3	19	10

Para este caso, a metragem de duto auxiliar é de _____
e a posição do duto principal deve ser _____.

Mais, um:
Suponha o duto principal no eixo L-W . Os poços se encontram nas seguintes posições:

x	19	5	2	14	20	21	23	9	15
y	18	8	17	7	16	27	22	3	11

Para este caso, a metragem de duto auxiliar é de _____
e a posicao do duto principal deve ser _____.

24.5 Exercício 3

Suponha o duto principal no eixo N-S . Os poços se encontram nas seguintes posições:

x	13	1	19	10	12	3	22	4	20
y	24	5	11	23	14	18	2	15	7

Para este caso, a metragem de duto auxiliar é de _____
e a posição do duto principal deve ser _____.

Mais, um:
Suponha o duto principal no eixo L-W . Os poços se encontram nas seguintes posições:

x	23	10	17	18	2	20	22	30	5	4
y	16	14	27	25	26	8	13	1	21	12

Para este caso, a metragem de duto auxiliar é de _____
e a posicao do duto principal deve ser _____.

24.6 Respostas

```
1      82.0 13.5 56.0 10.0
2      58.0 15.0 55.0 16.0
3      56.0 12.0 67.0 15.0
```

Capítulo 25

Exercícios Práticos: 054 - Kumon de algoritmos

25.1 Treinamento básico em Algoritmos

Nos trechos de algoritmos a seguir, você deve seguir a lógica de cada trecho descobrindo qual o valor que é impresso ao final.

25.2 Exercício 1

1.
```
1: função A11
2: X ← 15
3: enquanto X >8 faça
4:    X ← X - 4
5: fimenquanto
6: imprima X
7: fimfunção
```

2.
```
1: função A21
2: X ← 5
3: repita
4:    X ← X + 4
5: até  X >18
6: imprima X
7: fimfunção
```

3.
```
1: função A32
2: X ← 0
3: para  Y de 5 a 19 passo 3 faça
4:    X ← X + Y
5: fimpara
6: imprima X
7: fimfunção
```

4.
```
1: função A12
```

```
2: X ← 14
3: Y ← 4
4: enquanto X >7 faça
5:    Y ← Y + 3
6:    X ← X - 4
7: fimenquanto
8: imprima Y
9: fimfunção
```

5.
```
1: função A33
2: X ← 5
3: para  Y de 6 a 20 passo 3 faça
4:    X ← X + Y +9
5: fimpara
6: imprima X
7: fimfunção
```

6.
```
1: função A13
2: X ← 20
3: Y ← 19
4: enquanto X > 4 faça
5:    Y ← Y + X + 3
6:    X ← X - 3
7: fimenquanto
8: imprima Y
9: fimfunção
```

7.
```
1: função A41
2: X ← 3
3: Y ← 2
4: se X > 5 então
5:    Y ← Y + X + 4
6: senão
7:    Y ← Y - (X + 12)
8: fimse
9: imprima Y
10: fimfunção
```

8.
```
1: função A22
2: X ← 4
3: Y ← 6
4: repita
5:    Y ← Y + 2
6:    X ← X + 4
7: até  X >17
8: imprima Y
9: fimfunção
```

9.
```
1: função A34
```

```
 2: X ← 5
 3: para  Y de 3 a 21 passo 3 faça
 4:    se (Y mod 2) = 0 então
 5:       X ← X + Y + 4
 6:    senão
 7:       X ← X + Y + 1
 8:    fimse
 9: fimpara
10: imprima X
11: fimfunção
```

10.
```
 1: função A41
 2: X ← 4
 3: Y ← 4
 4: se X > 8∧ Y>9 então
 5:    Y ← Y + X + 3
 6: senão
 7:    Y ← Y - (X + 12 )
 8: fimse
 9: imprima Y
10: fimfunção
```

11.
```
 1: função A23
 2: X ← 6
 3: Y ← 9
 4: repita
 5:    Y ← Y + X + 3
 6:    X ← X + 2
 7: até  X ≥ 17
 8: imprima Y
 9: fimfunção
```

12.
```
 1: função A11
 2: X ← 18
 3: enquanto X >5 faça
 4:    X ← X - 3
 5: fimenquanto
 6: imprima X
 7: fimfunção
```

13.
```
 1: função A32
 2: X ← 1
 3: para  Y de 17 a 4 passo -3 faça
 4:    X ← X + Y
 5: fimpara
 6: imprima X
 7: fimfunção
```

14.
```
 1: função A21
```



```
2: X ← 6
3: repita
4:    X ← X + 4
5: até  X >13
6: imprima X
7: fimfunção
```

15.
```
1: função A41
2: X ← 3
3: Y ← 4
4: se X ≤ 8 então
5:    Y ← Y + X + 5
6: senão
7:    Y ← Y - (X + 12)
8: fimse
9: imprima Y
10: fimfunção
```

16.
```
1: função A13
2: X ← 7
3: Y ← 5
4: enquanto X ≤ 16 faça
5:    Y ← Y + X + 4
6:    X ← X + 4
7: fimenquanto
8: imprima Y
9: fimfunção
```

17.
```
1: função A33
2: X ← 5
3: para  Y de 3 a 17 passo 3 faça
4:    X ← X + Y + 5
5: fimpara
6: imprima X
7: fimfunção
```

Respostas

1	2	3	4	5	6
7	8	9	10	11	12
13	14	15	16	17	

25.3 Exercício 2

1.
```
1: função A11
2: X ← 19
```

```
3: enquanto X >7 faça
4:    X ← X - 2
5: fimenquanto
6: imprima X
7: fimfunção
```

2.
```
1: função A21
2: X ← 10
3: repita
4:    X ← X + 3
5: até  X ≥13
6: imprima X
7: fimfunção
```

3.
```
1: função A32
2: X ← 0
3: para  Y de 3 a 18 passo 3 faça
4:    X ← X + Y
5: fimpara
6: imprima X
7: fimfunção
```

4.
```
1: função A12
2: X ← 4
3: Y ← 4
4: enquanto X ≤18 faça
5:    Y ← Y + 2
6:    X ← X + 3
7: fimenquanto
8: imprima Y
9: fimfunção
```

5.
```
1: função A33
2: X ← 3
3: para  Y de 17 a 7 passo -3 faça
4:    X ← X + Y + 6
5: fimpara
6: imprima X
7: fimfunção
```

6.
```
1: função A13
2: X ← 14
3: Y ← 20
4: enquanto X ≥ 3 faça
5:    Y ← Y + X + 2
6:    X ← X - 3
7: fimenquanto
8: imprima Y
```



```
 9: fimfunção
```

7.
```
 1: função A41
 2: X ← 7
 3: Y ← 1
 4: se X = 5 então
 5:    Y ← Y + X + 7
 6: senão
 7:    Y ← Y - (X + 11)
 8: fimse
 9: imprima Y
10: fimfunção
```

8.
```
 1: função A22
 2: X ← 17
 3: Y ← 20
 4: repita
 5:    Y ← Y + 3
 6:    X ← X - 4
 7: até  X ≤8
 8: imprima Y
 9: fimfunção
```

9.
```
 1: função A34
 2: X ← 7
 3: para  Y de 4 a 19 passo 2 faça
 4:    se (Y mod 2) = 0 então
 5:       X ← X + Y + 5
 6:    senão
 7:       X ← X + Y + 1
 8:    fimse
 9: fimpara
10: imprima X
11: fimfunção
```

10.
```
 1: função A41
 2: X ← 1
 3: Y ← 6
 4: se X = 4∨ Y<4 então
 5:    Y ← Y + X + 4
 6: senão
 7:    Y ← Y - (X + 10)
 8: fimse
 9: imprima Y
10: fimfunção
```

11.
```
 1: função A23
 2: X ← 19
 3: Y ← 19
```

```
4: repita
5:    Y ← Y + X + 4
6:    X ← X - 4
7: até  X <10
8: imprima Y
9: fimfunção
```

12.
```
1: função A11
2: X ← 9
3: enquanto X ≤13 faça
4:    X ← X + 2
5: fimenquanto
6: imprima X
7: fimfunção
```

13.
```
1: função A32
2: X ← 0
3: para  Y de 4 a 15 passo 2 faça
4:    X ← X + Y
5: fimpara
6: imprima X
7: fimfunção
```

14.
```
1: função A21
2: X ← 7
3: repita
4:    X ← X + 3
5: até  X >17
6: imprima X
7: fimfunção
```

15.
```
1: função A41
2: X ← 1
3: Y ← 2
4: se X > 2 então
5:    Y ← Y + X + 5
6: senão
7:    Y ← Y - (X + 12)
8: fimse
9: imprima Y
10: fimfunção
```

16.
```
1: função A13
2: X ← 16
3: Y ← 20
4: enquanto X ≥ 3 faça
5:    Y ← Y + X + 3
6:    X ← X - 3
7: fimenquanto
```



```
8: imprima Y
9: fimfunção
```

17.
```
1: função A33
2: X ← 4
3: para  Y de 5 a 14 passo 3 faça
4:    X ← X + Y +7
5: fimpara
6: imprima X
7: fimfunção
```

Respostas

1	2	3	4	5	6
7	8	9	10	11	12
13	14	15	16	17	

25.4 Exercício 3

1.
```
1: função A11
2: X ← 6
3: enquanto X <17 faça
4:    X ← X + 4
5: fimenquanto
6: imprima X
7: fimfunção
```

2.
```
1: função A21
2: X ← 18
3: repita
4:    X ← X - 2
5: até  X ≤7
6: imprima X
7: fimfunção
```

3.
```
1: função A32
2: X ← 0
3: para  Y de 3 a 18 passo 3 faça
4:    X ← X + Y
5: fimpara
6: imprima X
7: fimfunção
```

4.
```
1: função A12
2: X ← 14
```

```
3: Y ← 9
4: enquanto X ≥7 faça
5:    Y ← Y + 4
6:    X ← X - 4
7: fimenquanto
8: imprima Y
9: fimfunção
```

5.
```
1: função A33
2: X ← 4
3: para  Y de 5 a 14 passo 3 faça
4:    X ← X + Y + 9
5: fimpara
6: imprima X
7: fimfunção
```

6.
```
1: função A13
2: X ← 15
3: Y ← 16
4: enquanto X ≥ 6 faça
5:    Y ← Y + X + 4
6:    X ← X - 2
7: fimenquanto
8: imprima Y
9: fimfunção
```

7.
```
1: função A41
2: X ← 5
3: Y ← 5
4: se X = 1 então
5:    Y ← Y + X + 8
6: senão
7:    Y ← Y - (X + 11)
8: fimse
9: imprima Y
10: fimfunção
```

8.
```
1: função A22
2: X ← 20
3: Y ← 14
4: repita
5:    Y ← Y + 4
6:    X ← X - 4
7: até  X <10
8: imprima Y
9: fimfunção
```

9.
```
1: função A34
2: X ← 5
```



```
 3: para  Y de 4 a 16 passo 2 faça
 4:    se (Y mod 2) = 0 então
 5:       X ← X + Y + 4
 6:    senão
 7:       X ← X + Y + 2
 8:    fimse
 9: fimpara
10: imprima X
11: fimfunção
```

10.
```
 1: função A41
 2: X ← 1
 3: Y ← 7
 4: se X ≥ 9∨Y<4 então
 5:    Y ← Y + X + 4
 6: senão
 7:    Y ← Y - (X + 11)
 8: fimse
 9: imprima Y
10: fimfunção
```

11.
```
 1: função A23
 2: X ← 19
 3: Y ← 18
 4: repita
 5:    Y ← Y + X + 2
 6:    X ← X - 2
 7: até  X < 9
 8: imprima Y
 9: fimfunção
```

12.
```
 1: função A11
 2: X ← 7
 3: enquanto X <13 faça
 4:    X ← X + 3
 5: fimenquanto
 6: imprima X
 7: fimfunção
```

13.
```
 1: função A32
 2: X ← 0
 3: para  Y de 4 a 18 passo 3 faça
 4:    X ← X + Y
 5: fimpara
 6: imprima X
 7: fimfunção
```

14.
```
 1: função A21
 2: X ← 8
```

```
3: repita
4:    X ← X + 4
5: até  X >16
6: imprima X
7: fimfunção
```

15.
```
1: função A41
2: X ← 7
3: Y ← 6
4: se X > 9 então
5:    Y ← Y + X + 9
6: senão
7:    Y ← Y - (X + 12)
8: fimse
9: imprima Y
10: fimfunção
```

16.
```
1: função A13
2: X ← 7
3: Y ← 8
4: enquanto X ≤ 15 faça
5:    Y ← Y + X + 4
6:    X ← X + 4
7: fimenquanto
8: imprima Y
9: fimfunção
```

17.
```
1: função A33
2: X ← 7
3: para  Y de 4 a 20 passo 2 faça
4:    X ← X + Y + 5
5: fimpara
6: imprima X
7: fimfunção
```

Respostas

1	2	3	4	5	6
7	8	9	10	11	12
13	14	15	16	17	

25.5 Respostas

```
1      7  21  55  10 110 112 -13  14 105 -12  93   3  56  14  12  50  75
2      7  13  63  14  77  66 -17  29 135  -5  76  15  54  19 -11  85  70
3     18   6  63  17  78  91 -11  26 103  -5 114  13  50  20 -13  53 160
```


Capítulo 26

Exercícios Práticos: 057 - Continuar seqüências

Para este exercício, são apresentados 8 algoritmos distintos que geram seqüências de números inteiros. Em cada uma delas, você deve descobrir qual o 8^o elemento da seqüência e escrevê-lo na resposta.

26.1 Por exemplo

Seja o seguinte algoritmo:

```
 1: função seq1 (inteiro n incr)
 2: ct ← 1
 3: enquanto ct < 9 faça
 4:    y ← 2
 5:    x ← 2
 6:    enquanto y < n faça
 7:       se 0=n mod y então
 8:          x ← x+1
 9:       fimse
10:       y ← y+1
11:    fimenquanto
12:    imprima x
13:    n ← n+incr
14:    ct++
15: fimenquanto
16: fimfunção
```

Este algoritmo, se chamado `seq1(3,7)` dará como resposta 2, 4, 2, 8, ..., e o oitavo elemento será 6. Se chamado com `seq1(4,8)` a resposta será 3, 6, 6, 6, ... 12 e se chamado com `seq1(33,55)` dará 4, 8, 4, 12, ... 8.

26.2 Exercício 1

1.
```
 1: função seq1 (inteiro n incr)
 2: ct ← 1
 3: enquanto ct < 9 faça
 4:    y ← 2
 5:    x ← 2
```

```
 6:    enquanto y < n faça
 7:       se 0=n mod y então
 8:          x ← x+1
 9:       fimse
10:       y ← y+1
11:    fimenquanto
12:    imprima x
13:    n ← n+incr
14:    ct++
15: fimenquanto
16: fimfunção
```

Sendo chamado com **seq1(8 , 8)**
Gerando os 4 primeiros números: 4, 5, 8, 6

2.
```
 1: função seq2 (inteiro alfa xn2 xn1)
 2: ct ← 1
 3: enquanto ct < 9 faça
 4:    ax ← (alfa × xn2) + xn1
 5:    xn2 ← xn1
 6:    xn1 ← ax
 7:    imprima ax
 8:    ct++
 9: fimenquanto
10: fimfunção
```

Sendo chamado com **seq2(3 , 3 , 6)**
Gerando os 4 primeiros números: 15, 33, 78, 177

3.
```
 1: função seq3 (inteiro xn2 xn1)
 2: ct ← 1
 3: enquanto ct < 9 faça
 4:    ax ← (teto(sqrt(xn1))) × xn2
 5:    xn2 ← xn1
 6:    xn1 ← ax
 7:    imprima ax
 8:    ct++
 9: fimenquanto
10: fimfunção
```

Sendo chamado com **seq3(6 , 7)**
Gerando os 4 primeiros números: 18, 35, 108, 385

4.
```
 1: função seq4 (inteiro xn)
 2: ct ← 1
 3: enquanto ct < 9 faça
 4:    se xn mod 2 = 0 então
 5:       ax ← 1 + 3 × xn
 6:    senão
 7:       ax ← (4 × xn)-1
 8:    fimse
 9:    imprima ax
10:    xn ← ax
11:    ct++
12: fimenquanto
13: fimfunção
```

Sendo chamado com **seq4(3)**
Gerando os 4 primeiros números: 11, 43, 171, 683

5.
```
1: função seq5 (inteiro xn1)
2: ct ← 1
3: enquanto ct < 9 faça
4:    ax ← xn1 + (ct-1)^(ct mod 3)
5:    imprima ax
6:    xn1 ← ax
7:    ct++
8: fimenquanto
9: fimfunção
```

Sendo chamado com **seq5(8)**
Gerando os 4 primeiros números: 8, 9, 10, 13

6.
```
1: função seq6 (inteiro xn2 xn1)
2: ct ← 1
3: repita
4:    ax ← chao(sqrt(xn1))
5:    bx ← 3 × xn2
6:    cx ← ax + bx
7:    imprima cx
8:    xn2 ← xn1
9:    xn1 ← cx
10:   ct++
11: até ct > 8
12: fimfunção
```

Sendo chamado com **seq6(5 , 4)**
Gerando os 4 primeiros números: 17, 16, 55, 55

7.
```
1: função seq7 (inteiro x)
2: ct ← 1
3: repita
4:    se ct < 3 então
5:       ax ← x × 4
6:    senão
7:       se ct<6 então
8:          ax ← x × 3
9:       senão
10:         ax ← x × 2
11:      fimse
12:   fimse
13:   imprima ax
14:   x ← ax
15:   ct ← ct+2
16: até ct > 16
17: fimfunção
```

Sendo chamado com **seq7(5)**
Gerando os 4 primeiros números: 20, 60, 180, 360

8.
```
1: função seq8 (inteiro zz)
2: ct ← 1
3: enquanto ct < 9 faça
4:    se ct mod 2 = 0 então
```



```
5:      x ← (ct mod 3)+(ct mod 4)+(ct mod 5)
6:    senão
7:      x ← (ct mod 2)+(ct mod 3)+(ct mod 4)
8:    fimse
9:    x ← x+zz
10:   imprima x
11:   ct ← ct+1
12: fimenquanto
13: fimfunção
```

Sendo chamado com **seq8(9)**
Gerando os 4 primeiros números: 12, 15, 13, 14

Respostas

1	2	3	4	5	6	7	8

26.3 Exercício 2

1.
```
1: função seq1 (inteiro n incr)
2: ct ← 1
3: enquanto ct < 9 faça
4:   y ← 2
5:   x ← 2
6:   enquanto y < n faça
7:     se 0=n mod y então
8:       x ← x+1
9:     fimse
10:    y ← y+1
11:  fimenquanto
12:  imprima x
13:  n ← n+incr
14:  ct++
15: fimenquanto
16: fimfunção
```

Sendo chamado com **seq1(4 , 5)**
Gerando os 4 primeiros números: 3, 3, 4, 2

2.
```
1: função seq2 (inteiro alfa xn2 xn1)
2: ct ← 1
3: enquanto ct < 9 faça
4:   ax ← (alfa × xn2) + xn1
5:   xn2 ← xn1
6:   xn1 ← ax
7:   imprima ax
8:   ct++
9: fimenquanto
10: fimfunção
```

Sendo chamado com **seq2(5 , 4 , 5)**
Gerando os 4 primeiros números: 25, 50, 175, 425

3.
```
1: função seq3 (inteiro xn2 xn1)
2: ct ← 1
```

```
3: enquanto ct < 9 faça
4:    ax ← (teto(sqrt(xn1))) × xn2
5:    xn2 ← xn1
6:    xn1 ← ax
7:    imprima ax
8:    ct++
9: fimenquanto
10: fimfunção
```

Sendo chamado com **seq3(5 , 7)**
Gerando os 4 primeiros números: 15, 28, 90, 280

4.
```
1: função seq4 (inteiro xn)
2: ct ← 1
3: enquanto ct < 9 faça
4:    se xn mod 2 = 0 então
5:       ax ← 1 + 3 × xn
6:    senão
7:       ax ← (4 × xn)-1
8:    fimse
9:    imprima ax
10:   xn ← ax
11:   ct++
12: fimenquanto
13: fimfunção
```

Sendo chamado com **seq4(8)**
Gerando os 4 primeiros números: 25, 99, 395, 1579

5.
```
1: função seq5 (inteiro xn1)
2: ct ← 1
3: enquanto ct < 9 faça
4:    ax ← xn1 + (ct-1)^(ct mod 3)
5:    imprima ax
6:    xn1 ← ax
7:    ct++
8: fimenquanto
9: fimfunção
```

Sendo chamado com **seq5(8)**
Gerando os 4 primeiros números: 8, 9, 10, 13

6.
```
1: função seq6 (inteiro xn2 xn1)
2: ct ← 1
3: repita
4:    ax ← chao(sqrt(xn1))
5:    bx ← 3 × xn2
6:    cx ← ax + bx
7:    imprima cx
8:    xn2 ← xn1
9:    xn1 ← cx
10:   ct++
11: até ct > 8
12: fimfunção
```

Sendo chamado com **seq6(9 , 4)**
Gerando os 4 primeiros números: 29, 17, 91, 60

7.
```
1: função seq7 (inteiro x)
2: ct ← 1
3: repita
4:    se ct < 3 então
5:       ax ← x × 4
6:    senão
7:       se ct<6 então
8:          ax ← x × 3
9:       senão
10:         ax ← x × 2
11:      fimse
12:   fimse
13:   imprima ax
14:   x ← ax
15:   ct ← ct+2
16: até ct > 16
17: fimfunção
```

Sendo chamado com **seq7(5)**
Gerando os 4 primeiros números: 20, 60, 180, 360

8.
```
1: função seq8 (inteiro zz)
2: ct ← 1
3: enquanto ct < 9 faça
4:    se ct mod 2 = 0 então
5:       x ← (ct mod 3)+(ct mod 4)+(ct mod 5)
6:    senão
7:       x ← (ct mod 2)+(ct mod 3)+(ct mod 4)
8:    fimse
9:    x ← x+zz
10:   imprima x
11:   ct ← ct+1
12: fimenquanto
13: fimfunção
```

Sendo chamado com **seq8(10)**
Gerando os 4 primeiros números: 13, 16, 14, 15

Respostas

1	2	3	4	5	6	7	8

26.4 Exercício 3

1.
```
1: função seq1 (inteiro n incr)
2: ct ← 1
3: enquanto ct < 9 faça
4:    y ← 2
5:    x ← 2
6:    enquanto y < n faça
7:       se 0=n mod y então
8:          x ← x+1
9:       fimse
10:      y ← y+1
```

```
11:    fimenquanto
12:    imprima x
13:    n ← n+incr
14:    ct++
15: fimenquanto
16: fimfunção
```

Sendo chamado com **seq1(9 , 9)**
Gerando os 4 primeiros números: 3, 6, 4, 9

2.
```
 1: função seq2 (inteiro alfa xn2 xn1)
 2: ct ← 1
 3: enquanto ct < 9 faça
 4:    ax ← (alfa × xn2) + xn1
 5:    xn2 ← xn1
 6:    xn1 ← ax
 7:    imprima ax
 8:    ct++
 9: fimenquanto
10: fimfunção
```

Sendo chamado com **seq2(5 , 4 , 6)**
Gerando os 4 primeiros números: 26, 56, 186, 466

3.
```
 1: função seq3 (inteiro xn2 xn1)
 2: ct ← 1
 3: enquanto ct < 9 faça
 4:    ax ← (teto(sqrt(xn1))) × xn2
 5:    xn2 ← xn1
 6:    xn1 ← ax
 7:    imprima ax
 8:    ct++
 9: fimenquanto
10: fimfunção
```

Sendo chamado com **seq3(6 , 7)**
Gerando os 4 primeiros números: 18, 35, 108, 385

4.
```
 1: função seq4 (inteiro xn)
 2: ct ← 1
 3: enquanto ct < 9 faça
 4:    se xn mod 2 = 0 então
 5:       ax ← 1 + 3 × xn
 6:    senão
 7:       ax ← (4 × xn)-1
 8:    fimse
 9:    imprima ax
10:    xn ← ax
11:    ct++
12: fimenquanto
13: fimfunção
```

Sendo chamado com **seq4(6)**
Gerando os 4 primeiros números: 19, 75, 299, 1195

5.
```
 1: função seq5 (inteiro xn1)
 2: ct ← 1
 3: enquanto ct < 9 faça
```



```
4:    ax ← xn1 + (ct-1)^(ct mod 3)
5:    imprima ax
6:    xn1 ← ax
7:    ct++
8:  fimenquanto
9:  fimfunção
```

Sendo chamado com **seq5(4)**
Gerando os 4 primeiros números: 4, 5, 6, 9

6.
```
1:  função seq6 (inteiro xn2 xn1)
2:  ct ← 1
3:  repita
4:    ax ← chao(sqrt(xn1))
5:    bx ← 3 × xn2
6:    cx ← ax + bx
7:    imprima cx
8:    xn2 ← xn1
9:    xn1 ← cx
10:   ct++
11: até ct > 8
12: fimfunção
```

Sendo chamado com **seq6(6 , 5)**
Gerando os 4 primeiros números: 20, 19, 64, 65

7.
```
1:  função seq7 (inteiro x)
2:  ct ← 1
3:  repita
4:    se ct < 3 então
5:      ax ← x × 4
6:    senão
7:      se ct<6 então
8:        ax ← x × 3
9:      senão
10:       ax ← x × 2
11:     fimse
12:   fimse
13:   imprima ax
14:   x ← ax
15:   ct ← ct+2
16: até ct > 16
17: fimfunção
```

Sendo chamado com **seq7(7)**
Gerando os 4 primeiros números: 28, 84, 252, 504

8.
```
1:  função seq8 (inteiro zz)
2:  ct ← 1
3:  enquanto ct < 9 faça
4:    se ct mod 2 = 0 então
5:      x ← (ct mod 3)+(ct mod 4)+(ct mod 5)
6:    senão
7:      x ← (ct mod 2)+(ct mod 3)+(ct mod 4)
8:    fimse
9:    x ← x+zz
```

```
10:    imprima x
11:    ct ← ct+1
12: fimenquanto
13: fimfunção
```

Sendo chamado com **seq8(6)**
Gerando os 4 primeiros números: 9, 12, 10, 11

Respostas

1	2	3	4	5	6	7	8

26.5 Respostas

1	7	5001	9771300	174763	85	557	5760	14
2	4	27050	4513600	404139	85	617	5760	15
3	12	29336	9771300	305835	81	651	8064	11

Capítulo 27

Exercícios Práticos: 062 - Engenharia reversa

27.1 Engenharia Reversa de algoritmos

Se a programação fosse mais engenharia e menos arte, talvez não precisássemos tanto usar os dons de Sherlock Holmes para corrigir programas. Infelizmente (ou será felizmente ??) tal não ocorre, e um programador profissional gasta mais de 50% do seu tempo procurando e corrigindo erros em programas.

É tarefa difícil, sutil, complexa e delicada. Às vezes um ponto fora do lugar põe a perder um imenso projeto. É tão importante a tarefa que se diz que bom programador não é aquele de programa bem, mas aquele que depura bem. A propósito, depurar é **identificar, localizar e corrigir erros, de maneira continuada e integrada para produzir software de qualidade.**

O exercício de hoje, visa estimular esta habilidade. Você vai receber alguns códigos e alguns resultados, e deverá inferir quais as inicializações originais das variáveis em cada código.

Por exemplo, no algoritmo

```
1: função EXEMPLO
2: A ← ======.
3: B ← 2
4: C ← 3
5: enquanto A < 10 faça
6:    se (B mod 3) = 0 então
7:       A ← A + 1
8:    senão
9:       A ← A + 2
10:   fimse
11:   se (C mod 4) = 0 então
12:      B ← B + 2
13:   senão
14:      A ← A + 1
15:   fimse
16:   imprima A, B, C
17: fimenquanto
```

Analisando as saídas, que são:
4 2 3

7 2 3
10 2 3
Percebe-se que a inicialização original do A foi 1. A resposta deste exemplo portanto, é 1.

27.2 Exercício 1

A seguir, diversos algoritmos. Em cada um deles, você deve dizer qual o valor inicial da variável A.

1

```
 1: função A1
 2: A ← _____.
 3: B ← 7
 4: C ← 13
 5: enquanto (A + B) > 4 faça
 6:    se A > C então
 7:       A ← A - 1
 8:    senão
 9:       A ← A + 1
10:    fimse
11:    B ← B - 2
12:    imprima A, B, C
13: fimenquanto
```

Esta execução deu como resultado
6 5 13
7 3 13
8 1 13
9 -1 13
10 -3 13
11 -5 13
12 -7 13
13 -9 13

2

```
 1: função A2
 2: A ← _____.
 3: B ← 9
 4: C ← 11
 5: enquanto (C ≠ 0) ∧ (C ≠ 1) faça
 6:    se (B mod 5) ≠ 0 então
 7:       B ← B - 1
 8:    fimse
 9:    se (A mod 3) = 0 então
10:       A ← A + 1
11:    fimse
12:    se (A > B) então
13:       A ← A + 2
14:    fimse
15:    C ← C - 1
```

```
16:    imprima A, B, C
17: fimenquanto
```

Esta execução deu como resultado

```
13 8 10
15 7 9
18 6 8
21 5 7
24 5 6
27 5 5
30 5 4
33 5 3
36 5 2
39 5 1
```

3

```
 1: função A3
 2: A ← ________.
 3: B ← 3
 4: C ← 5
 5: repita
 6:    A ← A + 1
 7:    B ← B + 1
 8:    C ← C + 1
 9:    enquanto C > 10 faça
10:       C ← C - 2
11:    fimenquanto
12:    se (C ≠ 3) ∨ (B ≠ 2) então
13:       A ← A + 2
14:    fimse
15:    imprima A, B, C
16: até (A + B + C ) > 30
```

Esta execução deu como resultado

```
6 4 6
9 5 7
12 6 8
15 7 9
```

4

```
 1: função A4
 2: A ← ________.
 3: B ← 15
 4: C ← 13
 5: enquanto (A>10) ∨ (B>10) ∨ (C>10) faça
 6:    se A > B então
 7:       C ← C - 1
 8:    fimse
 9:    se (B mod 3) = 1 então
10:       A ← A - 1
11:    fimse
```



```
12:    se C > B então
13:       B ← B - 3
14:    fimse
15:    imprima A, B, C
16:    A ← A - 1
17:    B ← B - 1
18:    C ← C - 2
19: fimenquanto
```

Esta execução deu como resultado

```
11 15 13
10 14 11
8 13 9
7 12 7
6 11 5
```

Respostas Escreva aqui os valores presumidos para as 4 variáveis "A" de cada um dos 4 algoritmos

1	2	3	4

27.3 Exercício 2

A seguir, diversos algoritmos. Em cada um deles, você deve dizer qual o valor inicial da variável A.

1

```
1: função A1
2: A ← _____.
3: B ← 7
4: C ← 11
5: enquanto (A + B) > 4 faça
6:    se A > C então
7:       A ← A - 1
8:    senão
9:       A ← A + 1
10:   fimse
11:   B ← B - 2
12:   imprima A, B, C
13: fimenquanto
```

Esta execução deu como resultado

```
7 5 11
8 3 11
9 1 11
10 -1 11
11 -3 11
12 -5 11
11 -7 11
```

2

```
 1: função A2
 2: A ← ======.
 3: B ← 10
 4: C ← 9
 5: enquanto (C ≠ 0) ∧ (C ≠ 1) faça
 6:    se (B mod 5) ≠ 0 então
 7:       B ← B - 1
 8:    fimse
 9:    se (A mod 3) = 0 então
10:       A ← A + 1
11:    fimse
12:    se (A > B) então
13:       A ← A + 2
14:    fimse
15:    C ← C - 1
16:    imprima A, B, C
17: fimenquanto
```

Esta execução deu como resultado

16 10 8
18 10 7
21 10 6
24 10 5
27 10 4
30 10 3
33 10 2
36 10 1

3

```
 1: função A3
 2: A ← =====.
 3: B ← 2
 4: C ← 5
 5: repita
 6:    A ← A + 1
 7:    B ← B + 1
 8:    C ← C + 1
 9:    enquanto C > 10 faça
10:       C ← C - 2
11:    fimenquanto
12:    se (C ≠ 3) ∨ (B ≠ 2) então
13:       A ← A + 2
14:    fimse
15:    imprima A, B, C
16: até (A + B + C ) > 30
```

Esta execução deu como resultado

7 3 6
10 4 7
13 5 8
16 6 9

4

```
 1: função A4
 2: A ← ________.
 3: B ← 12
 4: C ← 13
 5: enquanto (A>10) ∨ (B>10) ∨ (C>10) faça
 6:    se A > B então
 7:       C ← C - 1
 8:    fimse
 9:    se (B mod 3) = 1 então
10:       A ← A - 1
11:    fimse
12:    se C > B então
13:       B ← B - 3
14:    fimse
15:    imprima A, B, C
16:    A ← A - 1
17:    B ← B - 1
18:    C ← C - 2
19: fimenquanto
```

Esta execução deu como resultado
12 9 13
11 5 10

Respostas Escreva aqui os valores presumidos para as 4 variáveis "A" de cada um dos 4 algoritmos

1	2	3	4

27.4 Exercício 3

A seguir, diversos algoritmos. Em cada um deles, você deve dizer qual o valor inicial da variável A.

1

```
 1: função A1
 2: A ← ________.
 3: B ← 8
 4: C ← 9
 5: enquanto (A + B) > 4 faça
 6:    se A > C então
 7:       A ← A - 1
 8:    senão
 9:       A ← A + 1
10:    fimse
11:    B ← B - 2
12:    imprima A, B, C
```

13: **fimenquanto**

Esta execução deu como resultado

7 6 9
8 4 9
9 2 9
10 0 9
9 -2 9
10 -4 9
9 -6 9

2

```
1: função A2
2: A ← ________.
3: B ← 9
4: C ← 10
5: enquanto (C ≠ 0) ∧ (C ≠ 1) faça
6:    se (B mod 5) ≠ 0 então
7:       B ← B - 1
8:    fimse
9:    se (A mod 3) = 0 então
10:      A ← A + 1
11:   fimse
12:   se (A > B) então
13:      A ← A + 2
14:   fimse
15:   C ← C - 1
16:   imprima A, B, C
17: fimenquanto
```

Esta execução deu como resultado

13 8 9
15 7 8
18 6 7
21 5 6
24 5 5
27 5 4
30 5 3
33 5 2
36 5 1

3

```
1: função A3
2: A ← ________.
3: B ← 3
4: C ← 2
5: repita
6:    A ← A + 1
7:    B ← B + 1
8:    C ← C + 1
9:    enquanto C > 10 faça
```



```
10:       C ← C - 2
11:    fimenquanto
12:    se (C ≠ 3) ∨ (B ≠ 2) então
13:       A ← A + 2
14:    fimse
15:    imprima A, B, C
16: até (A + B + C ) > 30
```

Esta execução deu como resultado

8 4 3
11 5 4
14 6 5
17 7 6
20 8 7

4

```
 1: função A4
 2: A ← ________.
 3: B ← 14
 4: C ← 15
 5: enquanto (A>10) ∨ (B>10) ∨ (C>10) faça
 6:    se A > B então
 7:       C ← C - 1
 8:    fimse
 9:    se (B mod 3) = 1 então
10:       A ← A - 1
11:    fimse
12:    se C > B então
13:       B ← B - 3
14:    fimse
15:    imprima A, B, C
16:    A ← A - 1
17:    B ← B - 1
18:    C ← C - 2
19: fimenquanto
```

Esta execução deu como resultado

15 14 14
13 13 11
12 12 9
11 11 7

Respostas Escreva aqui os valores presumidos para as 4 variáveis "A" de cada um dos 4 algoritmos

1	2	3	4

27.5 Respostas

```
1          5  11   3  11
```

```
2          6  14   4  12
3          6  11   5  15
```


Capítulo 28

Exercícios Práticos: 065 - Se, enquanto, repita

28.1 Exercício 1

1

```
 1: função programa11
 2: inteiro A,B,C,D,U,V,T,I,G,S,Q
 3: inteiro F,W,L
 4: A←1; B←3; C←3;D←2;U←2
 5: V←1; T←1; I←1;G←3;S←3
 6: Q←2; F←3; W←3;L←1;U←2
 7: repita
 8:    V←1
 9:    enquanto V<7 faça
10:       C←2
11:       se C≠4 então
12:          T←2
13:          enquanto T<2 faça
14:             I←I+2
15:             G←G+3
16:             S←S+A+3
17:             T←T+1
18:          fimenquanto
19:       senão
20:          Q←3
21:          repita
22:             F←F+D+3
23:             W←W+1
24:             L←L+2
25:             Q←Q+2
26:          até Q≥9
27:       fimse
28:       V←V+1
29:    fimenquanto
30:    U←U+1
31: até U>5
```

```
32: imprima L+Q+G+U
33: fimfunção
```

2

```
 1: função programa12
 2: inteiro A,B,C,D,P,H,M,T,E,O,K
 3: inteiro X,Y,N,G,R,I,V
 4: A←2; B←1; C←2;D←3;P←3
 5: H←2; M←3; T←2;E←2;O←3
 6: K←3; X←1; Y←2;N←3;G←1
 7: R←3
 8: I←2
 9: V←2
10: D←8
11: se D=4 então
12:    P←1
13:    repita
14:       H←2
15:       enquanto H≤8 faça
16:          C←3
17:          se C=3 então
18:             M←M+3
19:             T←T+2
20:          senão
21:             E←E+3
22:             O←O+C+2
23:             K←K+A+1
24:             X←X+D+3
25:             Y←Y+D+1
26:          fimse
27:          H←H+1
28:       fimenquanto
29:       P←P+1
30:    até P>6
31: senão
32:    N←3
33:    enquanto N<8 faça
34:       G←2
35:       enquanto G<5 faça
36:          R←R+B+1
37:          I←I+1
38:          V←V+3
39:          G←G+2
40:       fimenquanto
41:       N←N+1
42:    fimenquanto
43: fimse
44: imprima K+P+M+V
45: fimfunção
```

3

```
1: função programa13
```

```
 2: inteiro A,B,C,D,S,Q,U,Z,F,V,P
 3: inteiro I,O,K
 4: A←3; B←3; C←3;D←1;S←3
 5: Q←1; U←1; Z←2;F←3;V←3
 6: P←2; I←3; O←3;K←1;A←5
 7: se A=3 então
 8:    S←2
 9:    enquanto S≤3 faça
10:       Q←2
11:       enquanto Q<8 faça
12:          U←3
13:          enquanto U<3 faça
14:             Z←Z+3
15:             F←F+1
16:             V←V+B+3
17:             U←U+2
18:          fimenquanto
19:          Q←Q+2
20:       fimenquanto
21:       S←S+2
22:    fimenquanto
23: senão
24:    P←1
25:    repita
26:       I←1
27:       repita
28:          O←2
29:          repita
30:             K←3
31:             repita
32:                K←K+1
33:             até K≥8
34:             O←O+2
35:          até O≥9
36:          I←I+2
37:       até I≥2
38:       P←P+1
39:    até P≥6
40: fimse
41: imprima Z+D+I+F
42: fimfunção
```

4

```
 1: função programa14
 2: inteiro A,B,C,D,Z,V,W,F,N,H,K
 3: inteiro S,R,I,P,Q,E,Y
 4: A←2; B←1; C←3;D←1;Z←1
 5: V←2; W←1; F←2;N←2;H←3
 6: K←3; S←3; R←3;I←1;P←2
 7: Q←3
 8: E←1
 9: Y←3
```



```
10: A←8
11: se A>6 então
12:    A←7
13:    se A>9 então
14:       Z←1
15:       repita
16:          V←3
17:          enquanto V≤2 faça
18:             D←8
19:             se D≤9 então
20:                W←W+B+1
21:                F←F+A+1
22:                N←N+C+3
23:                H←H+C+2
24:                K←K+1
25:             senão
26:                S←S+B+2
27:                R←R+D+3
28:                I←I+A+1
29:                P←P+B+1
30:                Q←Q+2
31:             fimse
32:             V←V+1
33:          fimenquanto
34:          Z←Z+1
35:       até Z≥8
36:    fimse
37: senão
38:    B←7
39:    se B=9 então
40:       E←2
41:       repita
42:          Y←2
43:          repita
44:             Y←Y+2
45:          até Y≥2
46:          E←E+2
47:       até E>8
48:    fimse
49: fimse
50: imprima K+B+F+P
51: fimfunção
```

Respostas

1	2	3	4

28.2 Exercício 2

1

```
1: função programa11
```

```
 2: inteiro A,B,C,D,M,S,V,Q,Z,T,H
 3: inteiro K,I,J
 4: A←3; B←3; C←1;D←3;M←2
 5: S←2; V←1; Q←1;Z←2;T←3
 6: H←3; K←3; I←3;J←2;M←2
 7: repita
 8:    S←2
 9:    enquanto S≤8 faça
10:       C←1
11:       se C>7 então
12:          V←2
13:          enquanto V<1 faça
14:             Q←Q+D+3
15:             Z←Z+A+1
16:             T←T+A+3
17:             V←V+1
18:          fimenquanto
19:       senão
20:          H←3
21:          repita
22:             K←K+C+1
23:             I←I+D+1
24:             J←J+B+3
25:             H←H+2
26:          até H>7
27:       fimse
28:       S←S+1
29:    fimenquanto
30:    M←M+1
31: até M>9
32: imprima T+A+Z+H
33: fimfunção
```

2

```
 1: função programa12
 2: inteiro A,B,C,D,K,R,X,I,P,V,E
 3: inteiro F,H,G,L,M,J,S
 4: A←1; B←3; C←2;D←2;K←2
 5: R←3; X←3; I←2;P←2;V←1
 6: E←3; F←3; H←2;G←3;L←2
 7: M←2
 8: J←1
 9: S←3
10: A←5
11: se A≠1 então
12:    K←2
13:    repita
14:       R←1
15:       enquanto R≤4 faça
16:          A←4
17:          se A≤8 então
18:             X←X+B+3
```



```
19:             I←I+A+1
20:           senão
21:             P←P+B+1
22:             V←V+2
23:             E←E+3
24:             F←F+1
25:             H←H+1
26:           fimse
27:           R←R+1
28:       fimenquanto
29:       K←K+1
30:     até K≥8
31: senão
32:     G←2
33:     enquanto G<5 faça
34:       L←2
35:       enquanto L<7 faça
36:         M←M+A+2
37:         J←J+B+2
38:         S←S+1
39:         L←L+2
40:       fimenquanto
41:       G←G+1
42:     fimenquanto
43: fimse
44: imprima F+E+J+B
45: fimfunção
```

3

```
 1: função programa13
 2: inteiro A,B,C,D,M,U,N,J,V,E,X
 3: inteiro R,I,K
 4: A←2; B←2; C←3;D←1;M←2
 5: U←3; N←2; J←2;V←3;E←3
 6: X←3; R←2; I←1;K←1;C←4
 7: se C<6 então
 8:     M←2
 9:     enquanto M<9 faça
10:       U←3
11:       enquanto U<2 faça
12:         N←3
13:         enquanto N≤6 faça
14:           J←J+1
15:           V←V+3
16:           E←E+3
17:           N←N+2
18:         fimenquanto
19:         U←U+2
20:       fimenquanto
21:       M←M+2
22:     fimenquanto
23: senão
```

```
24:    X←3
25:    repita
26:      R←3
27:      repita
28:        I←3
29:        repita
30:          K←3
31:          repita
32:            K←K+2
33:          até K>7
34:          I←I+1
35:        até I≥8
36:        R←R+2
37:      até R≥8
38:      X←X+2
39:    até X≥4
40: fimse
41: imprima M+K+C+R
42: fimfunção
```

4

```
 1: função programa14
 2: inteiro A,B,C,D,N,F,V,H,G,E,Z
 3: inteiro X,O,M,I,W,S,Y
 4: A←2; B←2; C←1;D←3;N←3
 5: F←2; V←3; H←1;G←3;E←1
 6: Z←1; X←2; O←3;M←3;I←2
 7: W←3
 8: S←2
 9: Y←1
10: C←3
11: se C≥7 então
12:    C←5
13:    se C≥3 então
14:      N←2
15:      repita
16:        F←2
17:        enquanto F≤8 faça
18:          D←8
19:          se D=1 então
20:            V←V+3
21:            H←H+1
22:            G←G+D+3
23:            E←E+1
24:            Z←Z+D+1
25:          senão
26:            X←X+C+1
27:            O←O+3
28:            M←M+3
29:            I←I+2
30:            W←W+2
31:          fimse
```



```
32:             F←F+2
33:           fimenquanto
34:           N←N+1
35:         até N≥4
36:       fimse
37: senão
38:       C←4
39:       se C=4 então
40:         S←2
41:         repita
42:           Y←2
43:           repita
44:             Y←Y+2
45:           até Y>2
46:           S←S+2
47:         até S≥8
48:       fimse
49: fimse
50: imprima Z+O+D+Y
51: fimfunção
```

Respostas

1	2	3	4

28.3 Exercício 3

1

```
 1: função programa11
 2: inteiro A,B,C,D,E,M,F,L,G,Y,N
 3: inteiro X,K,V
 4: A←2; B←1; C←1;D←2;E←1
 5: M←1; F←2; L←2;G←2;Y←1
 6: N←3; X←2; K←3;V←2;E←1
 7: repita
 8:   M←1
 9:   enquanto M≤7 faça
10:     D←3
11:     se D<1 então
12:       F←3
13:       enquanto F<1 faça
14:         L←L+C+3
15:         G←G+C+1
16:         Y←Y+A+1
17:         F←F+2
18:       fimenquanto
19:     senão
20:       N←3
21:       repita
22:         X←X+A+3
23:         K←K+B+1
```

```
24:             V←V+C+2
25:             N←N+1
26:           até N>3
27:         fimse
28:         M←M+2
29:       fimenquanto
30:       E←E+1
31:   até E≥2
32:   imprima M+K+C+F
33:   fimfunção
```

2

```
 1: função programa12
 2: inteiro A,B,C,D,N,M,O,I,J,G,K
 3: inteiro P,E,T,F,U,Q,W
 4: A←2; B←2; C←3;D←3;N←3
 5: M←1; O←3; I←2;J←2;G←1
 6: K←2; P←2; E←1;T←2;F←2
 7: U←3
 8: Q←2
 9: W←2
10: D←5
11: se D=6 então
12:   N←2
13:   repita
14:     M←3
15:     enquanto M≤7 faça
16:       D←5
17:       se D≤6 então
18:         O←O+2
19:         I←I+1
20:       senão
21:         J←J+2
22:         G←G+B+2
23:         K←K+C+1
24:         P←P+1
25:         E←E+1
26:       fimse
27:       M←M+2
28:     fimenquanto
29:     N←N+1
30:   até N≥9
31: senão
32:   T←3
33:   enquanto T<5 faça
34:     F←2
35:     enquanto F≤3 faça
36:       U←U+B+2
37:       Q←Q+C+2
38:       W←W+3
39:       F←F+1
40:     fimenquanto
```



```
41:       T←T+1
42:     fimenquanto
43: fimse
44: imprima I+P+C+J
45: fimfunção
```

3

```
 1: função programa13
 2: inteiro A,B,C,D,P,N,U,M,Y,I,V
 3: inteiro F,H,S
 4: A←1; B←2; C←2;D←2;P←1
 5: N←1; U←3; M←1;Y←2;I←1
 6: V←2; F←3; H←1;S←3;D←3
 7: se D<7 então
 8:   P←3
 9:   enquanto P<9 faça
10:     N←3
11:     enquanto N<2 faça
12:       U←3
13:       enquanto U≤6 faça
14:         M←M+A+1
15:         Y←Y+1
16:         I←I+1
17:         U←U+2
18:       fimenquanto
19:       N←N+2
20:     fimenquanto
21:     P←P+2
22:   fimenquanto
23: senão
24:   V←3
25:   repita
26:     F←2
27:     repita
28:       H←3
29:       repita
30:         S←2
31:         repita
32:           S←S+1
33:         até S≥4
34:         H←H+2
35:       até H>9
36:       F←F+1
37:     até F≥6
38:     V←V+2
39:   até V≥3
40: fimse
41: imprima V+M+A+D
42: fimfunção
```

4

```
1: função programa14
```

```
 2: inteiro A,B,C,D,X,T,Y,L,I,H,Z
 3: inteiro Q,U,M,O,J,N,K
 4: A←2; B←2; C←1;D←1;X←1
 5: T←2; Y←3; L←1;I←3;H←3
 6: Z←2; Q←1; U←2;M←2;O←3
 7: J←2
 8: N←1
 9: K←3
10: C←8
11: se C≥8 então
12:    D←7
13:    se D≠6 então
14:       X←1
15:       repita
16:          T←3
17:          enquanto T<3 faça
18:             D←4
19:             se D≤4 então
20:                Y←Y+D+3
21:                L←L+1
22:                I←I+2
23:                H←H+D+1
24:                Z←Z+2
25:             senão
26:                Q←Q+2
27:                U←U+3
28:                M←M+3
29:                O←O+2
30:                J←J+3
31:             fimse
32:             T←T+1
33:          fimenquanto
34:          X←X+2
35:       até X≥1
36:    fimse
37: senão
38:    B←8
39:    se B<3 então
40:       N←3
41:       repita
42:          K←1
43:          repita
44:             K←K+1
45:          até K≥6
46:          N←N+2
47:       até N≥7
48:    fimse
49: fimse
50: imprima Y+D+X+Q
51: fimfunção
```


Respostas

1	2	3	4

28.4 Respostas

```
1        12    41     9     8
2        17    10    17    11
3        23     9     7    14
```

Capítulo 29

Exercícios Práticos: 069 - 4 algoritmos

29.1 Exercício 1

A seguir existem 4 algoritmos que foram gerados de maneira aleatória. Todos estão corretos, não entram em loop, geram valores perfeitamente determinados e todos têm complexidade aceitável. Execute os testes de mesa pedidos e ao final calcule as 4 somas pedidas.

Algoritmo 1

```
 1: Algoritmo 1
 2: inteiro A, B, C, D, E
 3: A ← 5
 4: B ← 3
 5: C ← 0
 6: D ← 16
 7: E ← 17
 8: enquanto (D > 0) faça
 9:    se (A < −7) então
10:       B ← B + 1
11:    fimse
12:    enquanto (A > (−4 − 6)) faça
13:       B ← A + B + 2
14:       A ← A − 2
15:       E ← E + 7
16:       se (D > (E − 13)) então
17:          C ← C + −1
18:       fimse
19:    fimenquanto
20:    D ← D − 4
21: fimenquanto
22: imprima (A + B + C + D + E)
```

SOMA = _____ (A+B+C+D+E)

Algoritmo 2

1: Algoritmo 2
2: inteiro F, G, H, I, J
3: $F \leftarrow -1$
4: $G \leftarrow -3$
5: $H \leftarrow 14$
6: $I \leftarrow -6$
7: $J \leftarrow 12$
8: **enquanto** $((H + J) \leq (3 \times J))$ **faça**
9: **se** $(F < 1)$ **então**
10: $G \leftarrow G + 1$
11: **fimse**
12: **enquanto** $((F > 4) \vee (G \leq 16))$ **faça**
13: $G \leftarrow G + 2$
14: $F \leftarrow F - 2$
15: **se** $(I > 13)$ **então**
16: $J \leftarrow J + 3 - F$
17: **fimse**
18: **fimenquanto**
19: $H \leftarrow H + 3$
20: **fimenquanto**
21: imprima $(F + G + H + I + J)$

SOMA = _____ (F+G+H+I+J)

Algoritmo 3

1: Algoritmo 3
2: inteiro K, L, M, N, P
3: $K \leftarrow 10$
4: $L \leftarrow 1$
5: $M \leftarrow 7$
6: $N \leftarrow -9$
7: $P \leftarrow 20$
8: **enquanto** $(K \geq -4)$ **faça**
9: $L \leftarrow L + 1$
10: **se** $(N \leq -3)$ **então**
11: **enquanto** $(M \leq 12)$ **faça**
12: $M \leftarrow M + 6$
13: **fimenquanto**
14: $M \leftarrow M + 1$
15: **fimse**
16: $K \leftarrow K - 3$
17: $N \leftarrow N + 1$
18: $P \leftarrow P + 2 - K$
19: **enquanto** $(N \bmod 3 \neq 0)$ **faça**
20: $N \leftarrow N + 1$
21: **fimenquanto**
22: **fimenquanto**
23: imprima $(K + L + M + N + P)$

Algoritmo 4

SOMA = _____ (K+L+M+N+P)

```
 1: Algoritmo 4
 2: inteiro Q, R, S, T, U
 3: Q ← −1
 4: R ← 13
 5: S ← 18
 6: T ← 3
 7: U ← −9
 8: repita
 9:    se (Q < −5) então
10:       R ← R + 1
11:    fimse
12:    se (T > 11) então
13:       S ← S + 2
14:    fimse
15:    enquanto (U ≤ 17) faça
16:       U ← U + 4
17:       se (S > 10) então
18:          T ← T + 1
19:       fimse
20:    fimenquanto
21:    Q ← Q + 2
22: até ((Q > 0) ∨ (R ≠ 0))
23: imprima (Q + R + S + T + U)
```

SOMA = _____ (Q+R+S+T+U)

29.2 Exercício 2

A seguir existem 4 algoritmos que foram gerados de maneira aleatória. Todos estão corretos, não entram em loop, geram valores perfeitamente determinados e todos têm complexidade aceitável. Execute os testes de mesa pedidos e ao final calcule as 4 somas pedidas.

Algoritmo 1

```
 1: Algoritmo 1
 2: inteiro A, B, C, D, E
 3: A ← 1
 4: B ← −5
 5: C ← 7
 6: D ← 11
 7: E ← −3
 8: enquanto (D > 0) faça
 9:    se (A < −1) então
10:       B ← B + 1
11:    fimse
12:    enquanto (A > (−6 − 6)) faça
13:       B ← A + B + 2
14:       A ← A − 2
```



```
15:       E ← E + −7
16:       se (D > (E − 0)) então
17:          C ← C + 8
18:       fimse
19:    fimenquanto
20:    D ← D − 4
21: fimenquanto
22: imprima (A + B + C + D + E)
```

SOMA = _____ (A+B+C+D+E)

Algoritmo 2

```
 1: Algoritmo 2
 2: inteiro F, G, H, I, J
 3: F ← 17
 4: G ← 0
 5: H ← −9
 6: I ← 2
 7: J ← 14
 8: enquanto ((H + J) ≤ (3 × J)) faça
 9:    se (F < 20) então
10:       G ← G + 1
11:    fimse
12:    enquanto ((F > 3) ∨ (G ≤ −5)) faça
13:       G ← G + 2
14:       F ← F − 2
15:       se (I > 5) então
16:          J ← J + 3 − F
17:       fimse
18:    fimenquanto
19:    H ← H + 3
20: fimenquanto
21: imprima (F + G + H + I + J)
```

SOMA = _____ (F+G+H+I+J)

Algoritmo 3

```
 1: Algoritmo 3
 2: inteiro K, L, M, N, P
 3: K ← 16
 4: L ← −2
 5: M ← 12
 6: N ← 8
 7: P ← 15
 8: enquanto (K ≥ 0) faça
 9:    L ← L + 1
10:    se (N ≤ 17) então
11:       enquanto (M ≤ −3) faça
12:          M ← M + 6
13:       fimenquanto
```

```
14:       M ← M + 1
15:    fimse
16:    K ← K − 3
17:    N ← N + 1
18:    P ← P + 2 − K
19:    enquanto (N mod 3 ≠ 0) faça
20:       N ← N + 1
21:    fimenquanto
22: fimenquanto
23: imprima (K + L + M + N + P)
```

SOMA = _____ (K+L+M+N+P)

Algoritmo 4

```
 1: Algoritmo 4
 2: inteiro Q, R, S, T, U
 3: Q ← 6
 4: R ← −1
 5: S ← −3
 6: T ← 11
 7: U ← −9
 8: repita
 9:    se (Q < 12) então
10:       R ← R + 1
11:    fimse
12:    se (T > 17) então
13:       S ← S + 2
14:    fimse
15:    enquanto (U ≤ 19) faça
16:       U ← U + 4
17:       se (S > −5) então
18:          T ← T + 1
19:       fimse
20:    fimenquanto
21:    Q ← Q + 2
22: até ((Q > 0) ∨ (R ≠ 0))
23: imprima (Q + R + S + T + U)
```

SOMA = _____ (Q+R+S+T+U)

29.3 Exercício 3

A seguir existem 4 algoritmos que foram gerados de maneira aleatória. Todos estão corretos, não entram em loop, geram valores perfeitamente determinados e todos têm complexidade aceitável. Execute os testes de mesa pedidos e ao final calcule as 4 somas pedidas.

Algoritmo 1

```
1: Algoritmo 1
```



```
 2: inteiro A, B, C, D, E
 3: A ← 14
 4: B ← −4
 5: C ← −1
 6: D ← 3
 7: E ← −5
 8: enquanto (D > 0) faça
 9:    se (A < 17) então
10:       B ← B + 1
11:    fimse
12:    enquanto (A > (2 − 6)) faça
13:       B ← A + B + 2
14:       A ← A − 2
15:       E ← E + −3
16:       se (D > (E − 20)) então
17:          C ← C + 10
18:       fimse
19:    fimenquanto
20:    D ← D − 4
21: fimenquanto
22: imprima (A + B + C + D + E)
```

SOMA = _____ (A+B+C+D+E)

Algoritmo 2

```
 1: Algoritmo 2
 2: inteiro F, G, H, I, J
 3: F ← 15
 4: G ← 7
 5: H ← 1
 6: I ← 13
 7: J ← 4
 8: enquanto ((H + J) ≤ (3 × J)) faça
 9:    se (F < 16) então
10:       G ← G + 1
11:    fimse
12:    enquanto ((F > 2) ∨ (G ≤ 18)) faça
13:       G ← G + 2
14:       F ← F − 2
15:       se (I > 8) então
16:          J ← J + 3 − F
17:       fimse
18:    fimenquanto
19:    H ← H + 3
20: fimenquanto
21: imprima (F + G + H + I + J)
```

SOMA = _____ (F+G+H+I+J)

Algoritmo 3

```
 1: Algoritmo 3
 2: inteiro K, L, M, N, P
 3: K ← 13
 4: L ← −9
 5: M ← 0
 6: N ← 19
 7: P ← 15
 8: enquanto (K ≥ −1) faça
 9:    L ← L + 1
10:    se (N ≤ −4) então
11:       enquanto (M ≤ 18) faça
12:          M ← M + 6
13:       fimenquanto
14:       M ← M + 1
15:    fimse
16:    K ← K − 3
17:    N ← N + 1
18:    P ← P + 2 − K
19:    enquanto (N mod 3 ≠ 0) faça
20:       N ← N + 1
21:    fimenquanto
22: fimenquanto
23: imprima (K + L + M + N + P)
```

SOMA = _____ (K+L+M+N+P)

Algoritmo 4

```
 1: Algoritmo 4
 2: inteiro Q, R, S, T, U
 3: Q ← 8
 4: R ← −1
 5: S ← 12
 6: T ← −9
 7: U ← −7
 8: repita
 9:    se (Q < 14) então
10:       R ← R + 1
11:    fimse
12:    se (T > 17) então
13:       S ← S + 2
14:    fimse
15:    enquanto (U ≤ 18) faça
16:       U ← U + 4
17:       se (S > 2) então
18:          T ← T + 1
19:       fimse
20:    fimenquanto
21:    Q ← Q + 2
22: até ((Q > 0) ∨ (R ≠ 0))
23: imprima (Q + R + S + T + U)
```


SOMA = _____ (Q+R+S+T+U)

29.4 Exercício 4

A seguir existem 4 algoritmos que foram gerados de maneira aleatória. Todos estão corretos, não entram em loop, geram valores perfeitamente determinados e todos têm complexidade aceitável. Execute os testes de mesa pedidos e ao final calcule as 4 somas pedidas.

Algoritmo 1

```
 1: Algoritmo 1
 2: inteiro A, B, C, D, E
 3: A ← 3
 4: B ← −2
 5: C ← 20
 6: D ← 11
 7: E ← 1
 8: enquanto (D > 0) faça
 9:    se (A < −4) então
10:       B ← B + 1
11:    fimse
12:    enquanto (A > (−9 − 6)) faça
13:       B ← A + B + 2
14:       A ← A − 2
15:       E ← E + −5
16:       se (D > (E − 12)) então
17:          C ← C + 7
18:       fimse
19:    fimenquanto
20:    D ← D − 4
21: fimenquanto
22: imprima (A + B + C + D + E)
```

SOMA = _____ (A+B+C+D+E)

Algoritmo 2

```
 1: Algoritmo 2
 2: inteiro F, G, H, I, J
 3: F ← 12
 4: G ← 15
 5: H ← −3
 6: I ← −6
 7: J ← 2
 8: enquanto ((H + J) ≤ (3 × J)) faça
 9:    se (F < 17) então
10:       G ← G + 1
11:    fimse
12:    enquanto ((F > −4) ∨ (G ≤ 1)) faça
13:       G ← G + 2
14:       F ← F − 2
```

```
15:       se (I > 8) então
16:          J ← J + 3 − F
17:       fimse
18:    fimenquanto
19:    H ← H + 3
20: fimenquanto
21: imprima (F + G + H + I + J)
```

SOMA = _____ (F+G+H+I+J)

Algoritmo 3

```
 1: Algoritmo 3
 2: inteiro K, L, M, N, P
 3: K ← 16
 4: L ← 6
 5: M ← 5
 6: N ← 13
 7: P ← 19
 8: enquanto (K ≥ 7) faça
 9:    L ← L + 1
10:    se (N ≤ 17) então
11:       enquanto (M ≤ 12) faça
12:          M ← M + 6
13:       fimenquanto
14:       M ← M + 1
15:    fimse
16:    K ← K − 3
17:    N ← N + 1
18:    P ← P + 2 − K
19:    enquanto (N mod 3 ≠ 0) faça
20:       N ← N + 1
21:    fimenquanto
22: fimenquanto
23: imprima (K + L + M + N + P)
```

SOMA = _____ (K+L+M+N+P)

Algoritmo 4

```
 1: Algoritmo 4
 2: inteiro Q, R, S, T, U
 3: Q ← −8
 4: R ← 7
 5: S ← 6
 6: T ← −5
 7: U ← −9
 8: repita
 9:    se (Q < 14) então
10:       R ← R + 1
11:    fimse
12:    se (T > −6) então
```


13: $S \leftarrow S + 2$
14: **fimse**
15: **enquanto** $(U \leq 18)$ **faça**
16: $U \leftarrow U + 4$
17: **se** $(S > -2)$ **então**
18: $T \leftarrow T + 1$
19: **fimse**
20: **fimenquanto**
21: $Q \leftarrow Q + 2$
22: **até** $((Q > 0) \vee (R \neq 0))$
23: imprima $(Q + R + S + T + U)$

SOMA = _____ (Q+R+S+T+U)

29.5 Respostas

1	67	32	48	61
2	-27	76	36	47
3	121	16	32	41
4	-4	32	50	31

Capítulo 30

Exercícios Práticos: 073 - achar o décimo termo

Elementar, meu caro Watson. Holmes, S.

30.1 Exemplos

Para este exercício, são apresentadas diversas seqüências de 7 números. Naturalmente, elas são objeto de um programa de computador. O programa é apresentado, mas faltam nele certos parâmetros (x, y e z), cujos valores serão inteiros e sempre estarão entre 2 e 7, inclusive. Pelo aspecto da seqüência e do programa mostrado, você deve inferir tais valores. Depois disso, a seqüência já impressa deve ser verificada (preferencialmente pelo computador). Estando os 7 primeiros valores corretos, devem estar os seguintes. Para cada seqüência, você deve responder qual será o décimo valor.

Seqüência 1

1: **para** I de 1 até 10 **faça**
2: imprima $x + (I^2)$
3: **fimpara**

Para a seqüência 5 8 13 20 29 40 53 68 85 104, pode-se ver que o 10^o valor da seqüência é 104, já que $x = 4$.

Seqüência 2

1: **para** I de 1 até 10 **faça**
2: imprima $x + (I^3)$
3: **fimpara**

Para a seqüência 4 11 30 67 128 219 346 515 732 1003, pode-se ver que o 10^o valor da seqüência é 1003, já que $x = 3$.

Seqüência 3

1: **para** I de 1 até 10 **faça**
2: imprima $(x + I)^2$
3: **fimpara**

Para a seqüência 16 25 36 49 64 81 100 121 144 169, pode-se ver que o 10^o valor da seqüência é 169, já que $x = 3$.

Seqüência 4

1: **para** I de 1 até 10 **faça**
2: imprima $(x + I)^3$
3: **fimpara**

Para a seqüência 512 729 1000 1331 1728 2197 2744 3375 4096 4913, pode-se ver que o 10^o valor da seqüência é 4913, já que $x = 7$.

Seqüência 5

1: **para** I de 1 até 10 **faça**
2: imprima $(x \times I)^2$
3: **fimpara**

Para a seqüência 36 144 324 576 900 1296 1764 2304 2916 3600, pode-se ver que o 10^o valor da seqüência é 3600, já que $x = 6$.

Seqüência 6

1: **para** I de 1 até 10 **faça**
2: imprima $(x \times I)^3$
3: **fimpara**

Para a seqüência 8 64 216 512 1000 1728 2744 4096 5832 8000, pode-se ver que o 10^o valor da seqüência é 8000, já que $x = 2$.

Seqüência 7

1: **para** I de 1 até 10 **faça**
2: imprima x^I
3: **fimpara**

Para a seqüência 2 4 8 16 32 64 128 256 512 1024, pode-se ver que o 10^o valor da seqüência é 1024, já que $x = 2$.

Seqüência 8

1: **para** I de 1 até 10 **faça**
2: imprima $((I^2) + x) \times y$
3: **fimpara**

Para a seqüência 12 18 28 42 60 82 108 138 172 210, pode-se ver que o 10^o valor da seqüência é 210, já que $x = 5$ e $y = 2$.

Seqüência 9

1: **para** I de 1 até 10 **faça**
2: imprima $((I^3) + x) \times y$
3: **fimpara**

Para a seqüência 15 50 145 330 635 1090 1725 2570 3655 5010, pode-se ver que o 10^o valor da seqüência é 5010, já que $x = 2$ e $y = 5$.

Seqüência 10

1: **para** I de 1 até 10 **faça**
2: imprima $((x + I)^2) \times y$
3: **fimpara**

Para a seqüência 180 245 320 405 500 605 720 845 980 1125, pode-se ver que o 10^o valor da seqüência é 1125, já que $x = 5$ e $y = 5$.

Seqüência 11

1: **para** I de 1 até 10 **faça**
2: imprima $((x + I)^3) \times y$
3: **fimpara**

Para a seqüência 2401 3584 5103 7000 9317 12096 15379 19208 23625 28672, pode-se ver que o 10^o valor da seqüência é 28672, já que $x = 6$ e $y = 7$.

Seqüência 12

1: **para** I de 1 até 10 **faça**
2: imprima $((x + I)^2) + y$
3: **fimpara**

Para a seqüência 22 31 42 55 70 87 106 127 150 175, pode-se ver que o 10^o valor da seqüência é 175, já que $x = 3$ e $y = 6$.

Seqüência 13

1: **para** I de 1 até 10 **faça**
2: imprima $((x + I)^3) + y$
3: **fimpara**

Para a seqüência 131 222 349 518 735 1006 1337 1734 2203 2750, pode-se ver que o 10^o valor da seqüência é 2750, já que $x = 4$ e $y = 6$.

Seqüência 14

1: **para** I de 1 até 10 **faça**
2: imprima $(x^I) + y$
3: **fimpara**

Para a seqüência 8 10 14 22 38 70 134 262 518 1030, pode-se ver que o 10^o valor da seqüência é 1030, já que $x = 2$ e $y = 6$.

Seqüência 15

1: **para** I de 1 até 10 **faça**
2: imprima x^{I+y}
3: **fimpara**

Para a seqüência 32 64 128 256 512 1024 2048 4096 8192 16384, pode-se ver que o 10^o valor da seqüência é 16384, já que $x = 2$ e $y = 4$.

Seqüência 16

1: **para** I de 1 até 10 **faça**
2: imprima $((x + (I^2)) \times y) + z$
3: **fimpara**

Para a seqüência 25 37 57 85 121 165 217 277 345 421, pode-se ver que o 10^o valor da seqüência é 421, já que $x = 4$, $y = 4$ e $z = 5$.

Seqüência 17

1: **para** I de 1 até 10 **faça**
2: imprima $((x + (I^3)) \times y) + z$
3: **fimpara**

Para a seqüência 31 59 135 283 527 891 1399 2075 2943 4027, pode-se ver que o 10^o valor da seqüência é 4027, já que $x = 6$, $y = 4$ e $z = 3$.

Seqüência 18

1: **para** I de 1 até 10 **faça**
2: imprima $(((x + I)^2) \times y) + z$
3: **fimpara**

Para a seqüência 255 346 451 570 703 850 1011 1186 1375 1578, pode-se ver que o 10^o valor da seqüência é 1578, já que $x = 5$, $y = 7$ e $z = 3$.

Seqüência 19

1: **para** I de 1 até 10 **faça**
2: imprima $(((x + I)^3) \times y) + z$
3: **fimpara**

Para a seqüência 2061 3075 4377 6003 7989 10371 13185 16467 20253 24579, pode-se ver

que o 10^o valor da seqüência é 24579, já que $x = 6$, $y = 6$ e $z = 3$.

Seqüência 20

```
1: para I de 1 até 10 faça
2:     imprima ((x^I) + y) × z
3: fimpara
```

Para a seqüência 10 14 22 38 70 134 262 518 1030 2054, pode-se ver que o 10^o valor da seqüência é 2054, já que $x = 2$, $y = 3$ e $z = 2$.

30.2 Exercício 1

seq	v_1	v_2	v_3	v_4	v_5	v_6	v_7	10^o ?
1	3	6	11	18	27	38	51	
2	6	13	32	69	130	221	348	
3	9	16	25	36	49	64	81	
4	64	125	216	343	512	729	1000	
5	9	36	81	144	225	324	441	
6	64	512	1728	4096	8000	13824	21952	
7	3	9	27	81	243	729	2187	
8	40	55	80	115	160	215	280	
9	15	50	145	330	635	1090	1725	
10	64	100	144	196	256	324	400	
11	256	500	864	1372	2048	2916	4000	
12	13	20	29	40	53	68	85	
13	514	731	1002	1333	1730	2199	2746	
14	8	14	32	86	248	734	2192	
15	16	32	64	128	256	512	1024	
16	14	23	38	59	86	119	158	
17	17	52	147	332	637	1092	1727	
18	105	135	169	207	249	295	345	
19	260	504	868	1376	2052	2920	4004	
20	12	18	30	54	102	198	390	

30.3 Exercício 2

seq	v_1	v_2	v_3	v_4	v_5	v_6	v_7	10^o ?
1	8	11	16	23	32	43	56	
2	8	15	34	71	132	223	350	
3	25	36	49	64	81	100	121	
4	27	64	125	216	343	512	729	
5	9	36	81	144	225	324	441	
6	64	512	1728	4096	8000	13824	21952	
7	2	4	8	16	32	64	128	
8	20	32	52	80	116	160	212	
9	6	20	58	132	254	436	690	
10	112	175	252	343	448	567	700	
11	81	192	375	648	1029	1536	2187	
12	28	39	52	67	84	103	124	
13	514	731	1002	1333	1730	2199	2746	
14	4	6	10	18	34	66	130	
15	64	128	256	512	1024	2048	4096	
16	16	25	40	61	88	121	160	
17	18	53	148	333	638	1093	1728	
18	74	100	130	164	202	244	290	
19	140	325	630	1085	1720	2565	3650	
20	20	28	44	76	140	268	524	

30.4 Exercício 3

seq	v_1	v_2	v_3	v_4	v_5	v_6	v_7	10^o ?
1	5	8	13	20	29	40	53	
2	6	13	32	69	130	221	348	
3	25	36	49	64	81	100	121	
4	64	125	216	343	512	729	1000	
5	4	16	36	64	100	144	196	
6	64	512	1728	4096	8000	13824	21952	
7	3	9	27	81	243	729	2187	
8	28	40	60	88	124	168	220	
9	42	91	224	483	910	1547	2436	
10	98	128	162	200	242	288	338	
11	81	192	375	648	1029	1536	2187	
12	30	41	54	69	86	105	126	
13	127	218	345	514	731	1002	1333	
14	7	13	31	85	247	733	2191	
15	16	32	64	128	256	512	1024	
16	35	53	83	125	179	245	323	
17	20	41	98	209	392	665	1046	
18	348	453	572	705	852	1013	1188	
19	85	196	379	652	1033	1540	2191	
20	49	63	91	147	259	483	931	

30.5 Exercício 4

seq	v_1	v_2	v_3	v_4	v_5	v_6	v_7	10^o ?
1	3	6	11	18	27	38	51	
2	7	14	33	70	131	222	349	
3	9	16	25	36	49	64	81	
4	64	125	216	343	512	729	1000	
5	36	144	324	576	900	1296	1764	
6	27	216	729	1728	3375	5832	9261	
7	2	4	8	16	32	64	128	
8	35	56	91	140	203	280	371	
9	16	30	68	142	264	446	700	
10	144	196	256	324	400	484	576	
11	1372	2048	2916	4000	5324	6912	8788	
12	54	69	86	105	126	149	174	
13	66	127	218	345	514	731	1002	
14	4	6	10	18	34	66	130	
15	128	256	512	1024	2048	4096	8192	
16	22	40	70	112	166	232	310	
17	45	94	227	486	913	1550	2439	
18	38	66	102	146	198	258	326	
19	256	438	692	1030	1464	2006	2668	
20	24	36	60	108	204	396	780	

30.6 Respostas

```
1     102  1005   144  2197   900 64000 59049   535  5010   676  8788   148
      4915 59054  8192   311  5012   519  8792  3078
2     107  1007   196  1728   900 64000  1024   416  2004  1183  5184   199
      4915  1026 32768   313  5013   452  8645  4108
3     104  1005   196  2197   400 64000 59049   424  7035   512  5184   201  2746
      59053  8192   629  3017  1797  5188  7203
4     102  1006   144  2197  3600 27000  1024   728  2014   900 16384   261
      2199  1026 65536   616  7038   578  5494  6156
```

Capítulo 31

Exercícios Práticos: 077 - Algoritmos

O objetivo deste exercício é o treinamento do "teste de mesa", também conhecido como chinês. Você deve seguir os algoritmos com os dados de entrada fornecidos e ao final deve imprimir o resultado gerado pelo algoritmo. Todos os algoritmos estão corretos e fazem alguma computação útil.

31.1 Exercício 1

31.1.1 Algoritmo 1

1: Algoritmo UM
2: cadeia C[80]
3: inteiro I, J
4: leia(C)
5: $I \leftarrow 0$
6: $J \leftarrow 1$
7: **enquanto** $I < 2$ **faça**
8: **se** $C[J] =$ " $-$ " **então**
9: $I++$
10: **fimse**
11: $J++$
12: **fimenquanto**
13: **enquanto** $C[J] \neq$ "O" **faça**
14: imprima $C[J]$
15: $J++$
16: **fimenquanto**

Para os dados

```
ONDE-ORA-SE-ADUNA-O-CONCILIO-GUERREIRO
```

O algoritmo UM deve imprimir _____

31.1.2 Algoritmo 2

1: Algoritmo $DOIS$

```
 2: inteiro VET[6]
 3: inteiro SOMA, I
 4: pontoflutuante MED
 5: leia(VET)
 6: SOMA ← 0
 7: I ← 1
 8: enquanto I < 7 faça
 9:    SOMA ← SOMA + VET[I]
10:    I++
11: fimenquanto
12: MED ← SOMA ÷ 6
13: I ← 1
14: enquanto (I < 7) faça
15:    se VET[I] > (MED − 5) então
16:       imprima VET[I]
17:    fimse
18:    I++
19: fimenquanto
```

Para os dados

```
117 190 121 108 167 198
```

O algoritmo DOIS deverá imprimir _____

31.1.3 Algoritmo 3

```
 1: Algoritmo TRES
 2: inteiro VET[6]
 3: inteiro I, J
 4: I ← 2
 5: enquanto I ≤ 6 faça
 6:    leia VET[I]
 7:    I++
 8: fimenquanto
 9: I ← 3
10: enquanto (I ≤ 6) faça
11:    VET[1] ← VET[I]
12:    J ← I − 1
13:    enquanto (VET[J] > VET[1]) faça
14:       VET[J + 1] ← VET[J]
15:       J − −
16:    fimenquanto
17:    VET[J + 1] ← VET[1]
18:    I++
19: fimenquanto
20: imprima VET[3]
```

Para os dados

```
53  43  44  55  68
```

O algoritmo TRES deverá imprimir _____

Algoritmo 4

```
1: Algoritmo QUATRO
2: inteiro MAT[4][5]
3: inteiro C, I, J, LIN, COL
4: leia (C, LIN, COL)
5: I ← 1
6: enquanto I ≤ 4 faça
7:    J ← 1
8:    enquanto (J ≤ 5) faça
9:       MAT[I][J] ← C
10:      C ← C + 9
11:      J + +
12:   fimenquanto
13:   I + +
14: fimenquanto
15: imprima MAT[LIN][COL]
```

Solução

```
1    2    2
```

O algoritmo QUATRO deverá imprimir _____

31.2 Exercício 2

31.2.1 Algoritmo 1

```
1: Algoritmo UM
2: cadeia C[80]
3: inteiro I, J
4: leia(C)
5: I ← 0
6: J ← 1
7: enquanto I < 2 faça
8:    se C[J] = "O" então
9:       I + +
10:   fimse
11:   J + +
12: fimenquanto
13: enquanto C[J] ≠ "A" faça
14:   imprima C[J]
15:   J + +
16: fimenquanto
```

Para os dados

```
AO-VELHO-COITADO-DE-PENAS-RALADO
```

O algoritmo UM deve imprimir _____

31.2.2 Algoritmo 2

```
 1: Algoritmo DOIS
 2: inteiro VET[6]
 3: inteiro SOMA, I
 4: pontoflutuante MED
 5: leia(VET)
 6: SOMA ← 0
 7: I ← 1
 8: enquanto I < 7 faça
 9:    SOMA ← SOMA + VET[I]
10:    I++
11: fimenquanto
12: MED ← SOMA ÷ 6
13: I ← 1
14: enquanto (I < 7) faça
15:    se VET[I] > (MED − 2) então
16:       imprima VET[I]
17:    fimse
18:    I++
19: fimenquanto
```

Para os dados

```
154 142 112 120 173 114
```

O algoritmo DOIS deverá imprimir _____

31.2.3 Algoritmo 3

```
 1: Algoritmo TRES
 2: inteiro VET[6]
 3: inteiro I, J
 4: I ← 2
 5: enquanto I ≤ 6 faça
 6:    leia VET[I]
 7:    I++
 8: fimenquanto
 9: I ← 3
10: enquanto (I ≤ 6) faça
11:    VET[1] ← VET[I]
12:    J ← I − 1
13:    enquanto (VET[J] < VET[1]) faça
14:       VET[J + 1] ← VET[J]
15:       J−−
16:    fimenquanto
17:    VET[J + 1] ← VET[1]
18:    I++
19: fimenquanto
20: imprima VET[3]
```

Para os dados

```
56  70  75  82  79
```

O algoritmo TRES deverá imprimir _____

31.2.4 Algoritmo 4

```
1: Algoritmo QUATRO
2: inteiro MAT[4][5]
3: inteiro C, I, J, LIN, COL
4: leia (C, LIN, COL)
5: I ← 1
6: enquanto I ≤ 4 faça
7:    J ← 1
8:    enquanto (J ≤ 5) faça
9:       MAT[I][J] ← C
10:      C ← C + 7
11:      J + +
12:   fimenquanto
13:   I + +
14: fimenquanto
15: imprima MAT[LIN][COL]
```

Para os dados

```
7   3   5
```

O algoritmo QUATRO deverá imprimir _____

31.3 Exercício 3

31.3.1 Algoritmo 1

```
1: Algoritmo UM
2: cadeia C[80]
3: inteiro I, J
4: leia(C)
5: I ← 0
6: J ← 1
7: enquanto I < 3 faça
8:    se C[J] = " − " então
9:       I + +
10:   fimse
11:   J + +
12: fimenquanto
13: enquanto C[J] ≠ "A" faça
14:   imprima C[J]
15:   J + +
16: fimenquanto
```

Para os dados

```
DOS-VENCIDOS-TAPUIAS-INDA-CHOREM
```

O algoritmo UM deve imprimir _____

31.3.2 Algoritmo 2

```
 1: Algoritmo DOIS
 2: inteiro VET[6]
 3: inteiro SOMA, I
 4: pontoflutuante MED
 5: leia(VET)
 6: SOMA ← 0
 7: I ← 1
 8: enquanto I < 7 faça
 9:    SOMA ← SOMA + VET[I]
10:    I++
11: fimenquanto
12: MED ← SOMA ÷ 6
13: I ← 1
14: enquanto (I < 7) faça
15:    se VET[I] > (MED + 3) então
16:       imprima VET[I]
17:    fimse
18:    I++
19: fimenquanto
```

Para os dados

```
169 186 105 179 160 187
```

O algoritmo DOIS deverá imprimir _____

31.3.3 Algoritmo 3

```
 1: Algoritmo TRES
 2: inteiro VET[6]
 3: inteiro I, J
 4: I ← 2
 5: enquanto I ≤ 6 faça
 6:    leia VET[I]
 7:    I++
 8: fimenquanto
 9: I ← 3
10: enquanto (I ≤ 6) faça
11:    VET[1] ← VET[I]
12:    J ← I − 1
13:    enquanto (VET[J] > VET[1]) faça
14:       VET[J + 1] ← VET[J]
15:       J−−
16:    fimenquanto
17:    VET[J + 1] ← VET[1]
18:    I++
19: fimenquanto
20: imprima VET[4]
```

Para os dados

```
55  45  42  37  59
```

O algoritmo TRES deverá imprimir _____
subsectionAlgoritmo 4

```
1: Algoritmo QUATRO
2: inteiro MAT[4][5]
3: inteiro C, I, J, LIN, COL
4: leia (C, LIN, COL)
5: I ← 1
6: enquanto I ≤ 4 faça
7:    J ← 1
8:    enquanto (J ≤ 5) faça
9:       MAT[I][J] ← C
10:      C ← C + 8
11:      J + +
12:   fimenquanto
13:   I + +
14: fimenquanto
15: imprima MAT[LIN][COL]
```

Para os dados

```
4   2   5
```

O algoritmo QUATRO deverá imprimir _____
sectionRespostas

```
1  SE-ADUNA-
   190 167 198
   44
   55

2  -COIT
   154 142 173
   79
   105

3  IND
   169 186 179 187
   45
   76
```


Capítulo 32

Exercício Prático: 084 Balance LIne

1. Considere o seguinte algoritmo

```
 1: Algoritmo Balance Line
 2: estrutura CAD
 3: inteiro CODIGO {número da pessoa}
 4: cadeia NOME[12]
 5: inteiro IDADE
 6: pflutuante DIVIDA
 7: fim {estrutura}
 8: estrutura MOV
 9: caracter CODMOV {pode ser I, A ou E}
10: CAD DA
11: fim {estrutura}
12: CAD ENT, SAI
13: MOV ME
14: inteiro função LECAD
15: leia ENT
16: se fim-de-arquivo então
17:    devolva 999999
18: senão
19:    devolva ENT.CODIGO
20: fimse
21: fim {função}
22: inteiro função LEMOV
23: leia ME
24: se fim-de-arquivo então
25:    devolva 999999
26: senão
27:    devolva ME.DA.CODIGO
28: fimse
29: fim {função}
30: inteiro CC, CM
31: CC ← LECAD
32: CM ← LEMOV
33: enquanto (CC ≠ 999999 ∨ CM ≠ 999999) faça
34:    se (CC < CM) então
```

```
35:       SAI ← ENT
36:       imprima (SAI)
37:       CC ← LECAD
38:     senão
39:       se (CM < CC) então
40:         se (ME.CODMOV = 'A') então
41:           "Erro: ", ME.DA.CODIGO, "nao existe para alterar"
42:         senão
43:           se (ME.CODMOV ='E') então
44:             "Erro: ", ME.DA.CODIGO, "nao existe p/ excluir"
45:           senão
46:             SAI.CODIGO ← ME.DA.CODIGO
47:             SAI.NOME ← ME.DA.NOME
48:             SAI.IDADE ← ME.DA.IDADE
49:             SAI.DIVIDA ← ME.DA.DIVIDA
50:             imprima (SAI)
51:           fimse
52:         fimse
53:         CM ← LEMOV
54:       senão
55:         se (ME.CODMOV = 'I') então
56:           "Erro: ",ME.DA.CODIGO,"ja existente"
57:           SAI ← ENT
58:           imprima (SAI)
59:         senão
60:           se (ME.CODMOV = 'A') então
61:             SAI ← ENT
62:             se (ME.DA.NOME ≠ " ") então
63:               SAI.NOME ← ME.DA.NOME
64:             fimse
65:             se (ME.DA.IDADE ≠ 0) então
66:               SAI.IDADE ← ME.DA.IDADE
67:             fimse
68:             se (ME.DA.DIVIDA ≠ 0.0) então
69:               SAI.DIVIDA ← ME.DA.DIVIDA
70:             fimse
71:             imprime (SAI)
72:           fimse
73:         fimse
74:         CC ← LECAD
75:         CM ← LEMOV
76:       fimse
77:     fimse
78: fimenquanto
79: fim algoritmo
```

2. Considere os seguintes dados:

```
==== CADASTRO ====         ==== MOVIMENTO ====
 4hugo     27  826.00      A  2         22  531.00
 6kica     20  384.00      E  5          0     .00
 7antonio  23  259.00      E  7          0     .00
 8ana      23  792.00      E  8          0     .00
```

```
12carmem  16  609.00     E  9         0     .00
13felipe  15  922.00     E 11         0     .00
15jorge   21  117.00     E 12         0     .00
17antonio 20  377.00     I 14        22 1041.00
19elaine  21  609.00     A 15        15     .00
20luis    23 1015.00     A 20        21     .00
24kica    15  602.00     A 21        24     .00
26ana     15  972.00     A 23        18  466.00
27sara    28 1016.00     A 24        27 1049.00
28paulo   16  372.00     A 25jorge   19     .00
30sara    16  406.00     A 26        15  919.00
31jose    16  766.00     I 29        19  283.00
32paula   27  384.00     I 30        19  791.00
37felipe  16  279.00     I 34        23  444.00
39zuza    26  259.00     I 35        22  440.00
43kica    15  232.00     A 39biba    21     .00
44carmem  17 1082.00     I 42        28  825.00
46luis    19  296.00     E 43         0     .00
48kica    25  792.00     E 44         0     .00
49kica    16  991.00     I 45        21  594.00
50pedro   29  791.00     I 46carmem  26  440.00
```

3. Numere o cadastro de saída e as mensagens de erro, e responda:

   - Qual a linha numerada do cadastro de saída ? 3 _____
     9 _____
     26 _____

   - Qual a mensagem de erro numerada ? 6 _____
     9 _____

4. Considere os seguintes dados:

```
==== CADASTRO ====        ==== MOVIMENTO ====
 5joao    22  558.00     E  1         0     .00
 7jane    25  431.00     I  2        27 1058.00
 8paulo   29  673.00     A  3hugo    18     .00
10jane    24  228.00     A  4jorge   22     .00
16elaine  27  669.00     E  9         0     .00
17paulo   20  855.00     I 10        25  364.00
18biba    23  885.00     E 11         0     .00
22paula   16  558.00     I 12jose    25  971.00
23kica    28  556.00     I 15        22  736.00
24carmem  29  934.00     I 18        21  963.00
29hugo    16  685.00     E 19         0     .00
31biba    19  126.00     E 20         0     .00
32helio   21  932.00     I 25sara    28  694.00
33sara    29  537.00     E 27         0     .00
34zuza    20  615.00     A 31        23  927.00
35pedro   24  745.00     I 32        17  694.00
```



```
37felipe  21  745.00     I 34        29 1075.00
38jane    16  406.00     A 36        18     .00
40ana     18 1097.00     I 37        17  856.00
41hugo    25  971.00     E 40         0     .00
43felipe  29  491.00     I 44        22  492.00
46sara    27  615.00     E 47         0     .00
47paula   15  212.00     I 48        25  764.00
48carmem  27  331.00     I 49        19  930.00
49elaine  24  685.00     A 50        15     .00
```

5. Numere o cadastro de saída e as mensagens de erro, e responda:

   - Qual a linha numerada do cadastro de saída ? 7 _____
     21 _____
     28 _____

   - Qual a mensagem de erro numerada ? 1 _____
     17 _____

6. Considere os seguintes dados:

```
==== CADASTRO ====        ==== MOVIMENTO ====
 1jose    19  629.00     E  1         0     .00
 2jose    18  736.00     E  2         0     .00
 5zuza    23  191.00     E  3         0     .00
 6elaine  16  653.00     E  4         0     .00
 8joao    15  211.00     A 11        21  650.00
10kica    18  764.00     I 14hugo    19  232.00
11helio   28  578.00     E 15         0     .00
12joao    18  533.00     E 19         0     .00
13hugo    20  250.00     A 22hugo    23     .00
17carmem  24  440.00     I 23        26  922.00
18zuza    16  218.00     A 26        24  932.00
19kica    19  218.00     A 27joao    29     .00
20elaine  18  861.00     I 28        29  740.00
23zuza    29  466.00     E 33         0     .00
24jane    27  861.00     A 34        16     .00
25luis    19  218.00     A 36        29  112.00
27paula   18  767.00     E 38         0     .00
38helio   28  368.00     A 41        27     .00
39paula   16  498.00     E 43         0     .00
40sara    27  465.00     E 44         0     .00
41jorge   24  528.00     E 45         0     .00
43paulo   23  549.00     E 46         0     .00
47biba    22  250.00     I 47        18  506.00
49luis    16  528.00     A 48        18     .00
50biba    24 1060.00     I 49        26  424.00
```

7. Numere o cadastro de saída e as mensagens de erro, e responda:

- Qual a linha numerada do cadastro de saída ? 5 _____
  11 _____
  22 _____

- Qual a mensagem de erro numerada ? 13 _____
  15 _____

8. Considere os seguintes dados:

```
==== CADASTRO ====        ==== MOVIMENTO ====
 2antonio 29  653.00     I  3        23  775.00
 3helio   29  868.00     E  4         0     .00
 5zuza    24  539.00     E  6         0     .00
 6carmem  24  392.00     A  8        21     .00
 7jose    15  963.00     E 10         0     .00
 8carmem  29  807.00     A 11        26  781.00
 9elaine  18  704.00     E 14         0     .00
10hugo    28  454.00     I 15        29  941.00
15ana     26  217.00     I 19jose    22  389.00
16ana     26  796.00     E 20         0     .00
18pedro   20  563.00     A 23        21  538.00
20ana     28  231.00     E 24         0     .00
21paulo   21  886.00     I 30        26  945.00
23felipe  25  617.00     E 31         0     .00
25felipe  29  430.00     I 33ana     18  805.00
31joao    23  724.00     E 34         0     .00
32antonio 25  994.00     E 36         0     .00
35sara    26  388.00     E 37         0     .00
39jorge   16  614.00     E 39         0     .00
40pedro   22  621.00     A 40        18  523.00
41sara    25  307.00     A 43        24     .00
42hugo    21  228.00     I 45        28  538.00
44paula   27  358.00     I 46        21  600.00
45kica    27  706.00     I 48        21  693.00
46ana     21  563.00     E 50         0     .00
```

9. Numere o cadastro de saída e as mensagens de erro, e responda:

- Qual a linha numerada do cadastro de saída ? 21 _____
  22 _____
  24 _____

- Qual a mensagem de erro numerada ? 11 _____
  13 _____

32.1 Respostas


```
2
/ 13felipe      15  922.00
/ 24kica        27 1049.00
/ 50pedro       29  791.00

/Alteracao 23 inex
/Inclusao 46 duplo

   4
/ 15            22  736.00
/ 37felipe      21  745.00
/ 49elaine      24  685.00

/Exclusao   1 inex
/Alteracao 50 inex

6
/ 11helio       21  650.00
/ 20elaine      18  861.00
/ 50biba        24 1060.00

/Inclusao 47 duplo
/Inclusao 49 duplo

8
/ 44paula       27  358.00
/ 45kica        27  706.00
/ 48            21  693.00

/Inclusao 45 duplo
/Exclusao  50 inex
```

Capítulo 33

Exercícios Práticos: 110 - indexação e indireção

Sejam os vetores:

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
V	3	7	1	4	5	2	5	3	20	14	12	9	5	17	4	10
T	3	5	5	2	2	5	1	5	2	9	12	8	8	3	18	9
Z	3	2	6	7	6	4	8	8	10	17	7	5	5	20	6	1

E, sejam ainda $A = 4$, $E = -5$, $H = 7$, $K = 9$ e $M = -9$

Acompanhe os seguintes exemplos

```
 1. Z [ 14 +  1  ]                                  6

 2. Z [ 14 -  1  ]                                  5

 3. T [  9 -  2  ]                                  1

 4. Z [ Z [  3 + A  ] ]                             8

 5. V [ V [  5 ] ]                                  5

 6. T [ T [ Z [ 10 +  1  ] + K  ] + A  ] + 4       12

 7. T [ 16 -  2  ]                                  3

 8. Z [ T [  8 ] ] - 5                              1

 9. Z [ Z [ T [  6 +  1  ] + H  ] ] + 4            12

10. V [  3 +  1  ] - 6                             -2

11. T [ V [  3 +  2  ] ] + 3                       5

12. Z [ T [ T [  8 +  1  ] +  2  ] ] + 8           10

13. V [  6 + H  ] - M                              14
```

```
14. Z [ V [  7 ] +  1  ] + 7                    11

15. Z [ V [ Z [ 13 + E  ] ] - M  ] - A           1

16. V [  8 +  2  ]                              14

17. Z [ V [ 12 - K  ] +  2  ] + E                1

18. V [ Z [ Z [  3 ] ] -  1  ] + 8               9

19. Z [ 11 -  1  ] - K                           8

20. T [ T [  8 -  2  ] +  1  ] + E               0

21. T [ V [ V [ 12 ] -  2  ] -  1  ] - K        -7

22. V [  4 -  1  ] - M                          10

23. V [ V [  4 - M  ] +  2  ] - 9               -4

24. T [ Z [ T [  4 - E  ] -  1  ] - M  ]         8
```

33.1 Exercício 1

Sejam os vetores:

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
V	6	8	8	5	3	5	8	3	2	2	6	2	14	8	12	20
T	6	4	6	8	2	4	3	1	7	12	10	9	1	19	16	19
Z	8	4	2	5	3	4	2	4	15	14	11	10	12	16	6	18

E, sejam ainda $A = 1$; $E = 5$; $H = -7$; $K = 1$; $M = -1$ Resolva,

```
Z [  8 ] + M                                    ________

V [  2 ] - 8                                    ________

V [ 12 ] - 7                                    ________

V [ T [  7 -  2  ] - H  ] - 1                   ________

Z [ Z [  8 +  1  ] - A  ]                       ________

T [ V [ Z [ 10 ] - A  ] + M  ] + 4              ________

V [ 13 + K  ] + 5                               ________

V [ Z [  3 + E  ] +  2  ] - K                   ________1._______

T [ T [ V [  3 + A  ] - A  ] +  1  ] + 4        ________

V [  5 - H  ]                                   ________
```

```
V [ V [  2 - A  ] + A  ]                    ________
V [ T [ Z [  4 ] -  1  ] ] + 6              ________
Z [ 11 - K  ]                               ________
V [ T [  2 ] -  1  ] + M                    ________
T [ V [ V [  3 + M  ] ] ] - H               ________
T [ 13 + A  ] - A                           ________2.________
Z [ V [ 14 ] +  2  ] - 6                    ________
V [ V [ V [  2 + A  ] + E  ] ]              ________
Z [  5 +  1  ] - H                          ________
T [ V [  3 -  1  ] ]                        ________
Z [ T [ V [  5 + K  ] +  1  ] ] - 6         ________
T [  6 -  1  ] + A                          ________
T [ V [ 10 ] +  2  ]                        ________
V [ V [ T [ 13 -  2  ] ] -  1  ] + 3        ________3.________
V [ 10 - E  ] + E                           ________
T [ Z [ 11 ] ] + 6                          ________
T [ V [ Z [  6 ] +  2  ] +  1  ]            ________
Z [ 10 + M  ]                               ________
T [ T [  7 ] ] - K                          ________
Z [ Z [ T [ 10 ] -  2  ] -  1  ]            ________
Z [ 18 + H  ] - 1                           ________
V [ T [  9 - E  ] + A  ]                    ________4.________
T [ Z [ Z [  2 ] - M  ] +  2  ] - 8         ________
V [  1 ] - 9                                ________
V [ V [  7 +  1  ] + A  ] - 3               ________
Z [ V [ Z [  9 - K  ] -  2  ] + K  ] + A    ________
```



```
T [  8 - K  ] - 7                          ________

V [ Z [  3 -  2  ] ] + M                   ________

T [ Z [ V [  5 ] +  1  ] ]                 ________

V [  5 ] + 5                               ________5.________

V [ V [ 10 + A  ] - E  ]                   ________

T [ V [ T [  3 + K  ] -  1  ] -  2  ]      ________

V [ 17 -  2  ] - 3                         ________

V [ T [  7 -  1  ] +  2  ] + 1             ________

Z [ Z [ V [  6 -  1  ] + E  ] -  1  ]      ________

V [  8 +  2  ]                             ________

V [ Z [  9 +  2  ] + A  ] + 8              ________

Z [ V [ V [  2 +  2  ] + E  ] +  2  ] - H  ________6.________
```

Respostas

1	2	3	4	5	6

33.2 Exercício 2

Sejam os vetores:

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
V	7	4	5	4	8	6	2	2	13	9	8	20	13	7	7	7
T	3	1	2	7	2	5	3	1	20	18	11	4	6	2	7	9
Z	1	8	5	2	2	1	6	5	2	8	7	17	19	6	14	20

E, sejam ainda $A = -7$; $E = -1$; $H = 7$; $K = 8$; $M = 0$ Resolva,

```
T [  8 + E  ]                              ________

Z [  2 ]                                   ________

T [ 13 -  2  ] - 8                         ________

T [ Z [ 15 + A  ] +  2  ] - A              ________

T [ V [  1 ] + E  ] + 7                    ________

V [ Z [ V [  7 -  1  ] ] +  2  ]           ________

T [  4 + E  ] + 8                          ________

V [ V [  9 - M  ] ]                        ________1.________
```

```
T [ Z [ T [  6 +  1  ] +  1  ] + E  ]        ________
T [ 10 ] + 7                                 ________
V [ T [ 11 ] - E  ] + 7                      ________
Z [ V [ V [  4 ] ] ]                         ________
Z [ 16 -  2  ]                               ________
V [ T [ 10 - E  ] +  1  ]                    ________
Z [ V [ V [  3 +  2  ] - M  ] - A  ]         ________
Z [ 15 + M  ] - M                            ________2.________
V [ Z [  8 -  1  ] -  2  ] - A               ________
V [ T [ Z [  6 -  1  ] + K  ] - K  ] + 3     ________
Z [ 16 -  1  ] + K                           ________
V [ V [ 17 -  1  ] -  1  ]                   ________
Z [ T [ T [ 10 +  2  ] + M  ] ] + A          ________
Z [ 13 ]                                     ________
T [ Z [ 11 ] +  2  ] - 2                     ________
V [ T [ V [ 15 - H  ] +  2  ] ]              ________3.________
V [ 10 +  2  ] - 6                           ________
T [ Z [  8 -  2  ] + M  ] - M                ________
V [ Z [ T [  1 +  2  ] - A  ] + E  ] - M     ________
V [ 12 + A  ] - H                            ________
V [ T [ 15 + E  ] +  1  ] + E                ________
T [ Z [ Z [  2 - E  ] + H  ] -  1  ] + H     ________
T [  1 +  1  ] + 6                           ________
Z [ V [ 10 -  1  ] ] - E                     ________4.________
V [ V [ Z [ 13 -  1  ] - K  ] ]              ________
Z [  6 ]                                     ________
```



```
T [ V [  6 -  2  ] ] + A                          ________

Z [ V [ V [ 15 - E  ] +  2  ] -  1  ] + 8         ________

V [  6 ]                                          ________

V [ T [ 13 ] + H  ] + 7                           ________

V [ Z [ Z [  6 +  1  ] - M  ] + M  ]              ________

Z [ 14 ]                                          ________5.________

T [ V [  1 +  1  ] - A  ]                         ________

T [ T [ T [ 16 ] + H  ] ] - E                     ________

V [ 17 - H  ] + 8                                 ________

T [ T [ 10 - H  ] ] + K                           ________

T [ Z [ Z [  6 ] ] + M  ]                         ________

T [ 11 +  2  ] + E                                ________

T [ V [  4 ] +  2  ] - 8                          ________

V [ V [ Z [  4 ] - M  ] +  2  ]                   ________6.________
```

Respostas

1	2	3	4	5	6

33.3 Exercício 3

Sejam os vetores:

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
V	3	4	3	2	3	5	4	5	5	18	20	2	5	14	14	4
T	3	2	7	2	3	8	5	3	15	13	18	6	10	9	15	19
Z	4	7	2	5	6	5	3	3	18	9	11	14	1	14	7	5

E, sejam ainda $A = 3$; $E = -6$; $H = -3$; $K = 3$; $M = -5$ Resolva,

```
V [ 13 + H  ]                                     ________

V [  4 - H  ] - H                                 ________

V [  8 -  2  ] + M                                ________

V [ V [  7 +  1  ] -  1  ] - H                    ________

Z [ V [  8 - M  ] - A  ] + 6                      ________

T [ V [ V [  5 -  1  ] ] - A  ] - A                ________
```

```
V [ 16 -  1  ]                          ________
T [ T [ 11 + E  ] + A  ] - 9            ________1.________
T [ Z [ T [  8 - K  ] -  1  ] + K  ]    ________
Z [  7 +  1  ] + 5                      ________
V [ V [ 14 ] - K  ]                     ________
V [ T [ Z [ 18 - A  ] + M  ] ] - A      ________
T [  2 +  2  ] + 5                      ________
T [ Z [  6 +  2  ] +  1  ]              ________
T [ Z [ T [ 10 +  2  ] +  1  ] - E  ]   ________
V [ 12 ]                                ________2.________
V [ V [ 14 -  1  ] +  2  ]              ________
T [ V [ V [  7 ] + A  ] +  2  ] + 4     ________
V [  4 +  2  ]                          ________
V [ Z [  8 +  2  ] ]                    ________
T [ T [ Z [ 15 -  1  ] + M  ] - A  ]    ________
Z [ 16 ]                                ________
V [ Z [  6 - E  ] -  2  ]               ________
Z [ T [ T [ 12 ] - K  ] + K  ]          ________3.________
V [  9 -  2  ]                          ________
V [ V [  8 ] ] - H                      ________
Z [ Z [ V [ 13 + A  ] ] -  2  ] - K     ________
V [  7 ] + E                            ________
V [ Z [  6 ] + K  ] + 1                 ________
V [ T [ V [ 15 -  2  ] - H  ] - M  ]    ________
V [  2 + A  ]                           ________
V [ V [  7 - H  ] - K  ] - 5            ________4.________
```



```
T [ Z [ V [  1 ] ] ] + H                    ________

V [  3 ]                                    ________

T [ V [  1 + A  ] ] + 5                     ________

Z [ Z [ T [  4 ] - E  ] + A  ]              ________

V [ 13 - H  ] + A                           ________

T [ T [ 17 - K  ] +  1  ] - E               ________

T [ V [ Z [ 14 ] ] + H  ]                   ________

Z [  8 ] - E                                ________5._______

V [ V [  8 + E  ] -  1  ] + 1               ________

Z [ Z [ Z [ 12 + M  ] ] +  1  ]             ________

V [ 13 ] - K                                ________

T [ V [ 19 - A  ] - H  ]                    ________

Z [ Z [ V [  1 +  1  ] - A  ] ] - 8         ________

Z [  5 - A  ] - M                           ________

Z [ Z [ 18 + H  ] - K  ] - M                ________

Z [ T [ V [  8 + E  ] -  2  ] +  2  ]       ________6._______
```

Respostas

1	2	3	4	5	6

33.4 Respostas

1	37	82	47	72	17	51
2	64	99	89	72	78	69
3	56	68	48	30	67	37

Capítulo 34

Exercícios Práticos: 116 - manuseio de tabelas

34.1 Manuseio de tabelas

A tabela é uma excelente ferramenta para ajudar na programação. Sempre que se necessita consultar, acumular, separar em classes e tarefas similares, a tabela é a solução.

A tabela pode ser de 1 dimensão e neste caso é chamada vetor. Por exemplo, Se sabendo existirem 1200 alunos no curso, quisermos saber quantos alunos há em cada um dos 5 anos do curso, e supondo que cada aluno é assim identificado

informação	significado
código de matricula	o número único que identifica o aluno na instituição
nome do aluno	
série do aluno	em que ano do curso ele está (inteiro?)

Como fazer ? Basta criar um vetor

```
1: inteiro QTDAL [5]
2: QTDAL ← 0
```

E, ao processar cada aluno, incluir o seguinte comando

```
1: QTDAL [serie-aluno] ← QTDAL [serie-aluno] + 1
```

Um segundo caso, algo mais complexo, ocorre quando os identificadores da classe não são conhecidos de antemão. Por exemplo, suponhamos querer saber qual o local de nascimento (município) mais freqüente entre os 1200 alunos do curso.

Aqui, uma esperteza: É de se esperar que existam uns poucos municípios que concentrem a maioria das pessoas (esta regra é conhecida como Regra de Pareto). O problema é que não sabemos quais são. Naturalmente, não precisamos criar um vetor com 5000 números (5000 são os municípios brasileiros, número aproximado.

Agora, o índice de acesso não é mais conhecido de antemão e precisa ser localizado. Suponhamos existir o campo MUNALU, como sendo o código numérico do município onde o aluno nasceu. Suponhamos que um limite exagerado (para a quantidade de municípios) seja 200.

Criaremos dois vetores, o primeiro contendo os códigos de município que forem surgindo e o segundo para guardar os contadores. Ambos os vetores serão acessados pelo mesmo índice.

```
1: inteiro MUNICIPIO[200], CONTADOR[200]
2: para I de 1 até 200 faça
3:    MUNICIPIO[I] ← CONTADOR[I] ← 0
4: fimpara
```

A partir deste ponto, os dois vetores estão prontos para receber os dados. Para cada MUNALU lido, deve-se pesquisar o vetor MUNICIPIO da matriz até que uma das três coisas aconteça: a) Achar um MUNICIPIO = MUNALU; b) Achar um MUNICIPIO = 0; c) Ultrapassar o limite de I = 200.

A primeira indica que este município já havia aparecido e seu contador deve ser incrementado em uma unidade. A segunda, informa ser a primeira vez que este município está aparecendo e seu contador deve ser 1. A terceira, não deveria ocorrer e se acontecer indicará um erro.

Eis o algoritmo

```
inteiro MUNICIPIO[200], CONTADOR[200]
para I de 1 até 200 faça
  MUNICIPIO[I] ← CONTADOR[I] ← 0
fimpara
leia MUNALU
enquanto MUNALU ≥ 0 faça
  I ← 1
  enquanto MUNALU ≠ MUNICIPIO[I] ∧ MUNICIPIO[I] ≠ 0 faça
    I++
    se I> 200 então
      ...erro...
    fimse
  fimenquanto
  CONTADOR[I] ← CONTADOR[I] + 1
  leia MUNALU
fimenquanto
// achar o mais freqüente
MELHOR ← −∞
QUAL ← 0
I ← 1
enquanto I ≤ 200 ∧ MUNICIPIO[I] ≠ 0 faça
  se CONTADOR[I] > MELHOR então
    MELHOR ← CONTADOR[I]
    QUAL ← I
  fimse
  I++
fimenquanto
imprima MUNICIPIO[QUAL]
```

Operações fundamentais: inclusão Para incluir um novo elemento em uma tabela, duas condições devem ocorrer:

a) Deve haver espaço na estrutura; b) A chave a incluir deve ser nova (isto é conhecido como atributo UNIQUE).

Em geral, o primeiro ponto é controlado por uma variável global, por exemplo QTDUSADA, que começa com zero, é incrementada a cada inclusão e pode crescer até o limite (200 no exemplo acima). Note-se que na inclusão de tabelas que sofrerão busca linear ordenada, a tabela deverá ser mantida ordenada, o que talvez implique reordenações.

Exclusão Se necessita diminuir o contador de linhas válidas e eventualmente reagrupar as linhas (caso a linha a excluir não seja a última), para evitar a existência de buracos na tabela

Pesquisa

Busca Linear Para este algoritmo o conjunto de chaves não é mantido ordenado. O algoritmo básico é:

```
 1: função BUSCALIN(inteiro CHAVE, inteiro TABELA[num])
 2: inteiro i ← 1; qtd ← 0
 3: enquanto i ≤ tamanho(TABELA) faça
 4:    qtd ← qtd + 1
 5:    se CHAVE = TABELA[i] então
 6:       ...ACHOU...
 7:       i ← tamanho(TABELA)
 8:    fimse
 9:    qtd ← qtd + 1
10:    i++
11: fimenquanto
12: devolva qtd
```

```
37 BUSCALIN   62 45 61 78  1 94 30 28  4 31 33 92 37  2 23 e  25
98 BUSCALIN   70 98 81 46 29 12 65 40 72 14 20  4 82  2 30 e   3
 3 BUSCALIN   29 66 71 42 48 44 69 35 30 37 94 68 36 19 21 e  30
```

Busca linear ordenada Neste algoritmo, o conjunto de chaves deverá estar ordenado. Eis o algoritmo:

```
 1: função BUSCALINORD (inteiro CHAVE, inteiro TABELA[num])
 2: inteiro i ← 1; qtd ← 0
 3: enquanto (i ≤ tamanho(TABELA)) ∧ (CHAVE > TABELA[i]) faça
 4:    qtd ← qtd + 2
 5:    i++
 6: fimenquanto
 7: se CHAVE = TABELA[i] então
 8:    ...ACHOU...
 9: fimse
10: devolva qtd
```

```
96 BUSCALINORD 5 10 26 34 39 48 56 60 68 70 73 89 92 93 96 e  28
58 BUSCALINORD 1  2  8 10 12 41 42 47 49 53 58 59 60 62 85 e  20
96 BUSCALINORD 2 12 21 26 27 37 43 46 52 54 65 79 83 84 89 e  30
```

Busca linear com sentinela A tabela não precisa estar ordenada, e a busca gasta apenas a metade dos testes da busca linear. A sentinela é a chave que se busca que é forçosamente incluída ao final da tabela. Assim, a saída do laço sempre se dará pela igualdade de chaves. Se a chave procurada for encontrada no meio da tabela, ...ACHOU... Se for encontrada no final da tabela, (onde foi explicitamente colocada) é porque ela não se encontrava lá antes do início do algoritmo.

```
1: função BUSCALINSEN (inteiro CHAVE, inteiro TABELA[num])
2: inteiro i ← 1; qtd ← 0
3: TABELA[ULTIMO+1] ← CHAVE {note que ULTIMO não é alterado}
```



```
4: enquanto CHAVE ≠ TABELA[i]) faça
5:    qtd++
6:    i++
7: fimenquanto
8: se i ≠ (ULTIMO + 1) então
9:    ...ACHOU...
10: fimse
11: devolva qtd
```

```
38 BUSCALINSEN 24 35 91 18  8 72 10 88 27 54 28 34 43  3 38 e 14
54 BUSCALINSEN 63 17 35 68 27 86 54 53  5 78 41 58 44 51 25 e  6
61 BUSCALINSEN 75 67 51 60 85 38 49 74 73 78 66 35 24  4  2 e 15
```

Alguns dados experimentais Para o teste a seguir, fiz 6 passagens de 1000 chaves. Para cada método, em cada passagem foram 3 tentativas: as primeiras 2 com sucesso e a última para uma chave inexistente. Eis os dados:

passagem	Lin	Lin Ord	Lin Sen
1	1265.3	1063.3	817.3
2	1777.3	749.3	621.7
3	1832.0	1314.0	674.0
4	1244.7	613.3	634.7
5	1200.7	1186.7	607.3
6	948.0	832.7	770.3
média	1378.0	959.9	687.6

34.2 Exercício 1

Calcule:

```
60 BUSCALIN       2 44 67 37 73 12 60 17 65 62 54 97  6 83 81  e _____
29 BUSCALINORD    5  8 14 18 25 31 32 41 62 65 66 71 73 76 99  e _____
98 BUSCALINSEN   65 86 60 18 10 67 87 98 56  5 80  8 16 52 37  e _____
```

34.3 Exercício 2

Calcule:

```
92 BUSCALIN      77 92 70 35 89 31 59 48 64 80  9 37 11 41 72  e _____
28 BUSCALINORD    1  4 24 25 27 28 29 30 33 35 40 43 50 53 56  e _____
29 BUSCALINSEN    3 69 18 25 88 28 97 35 13 72 39 38 52 51 73  e _____
```

34.4 Exercício 3

Calcule:

```
89 BUSCALIN      39 38 52 71 53 23 41 15 66 89  4 48 22 21 11  e _____
87 BUSCALINORD    4  6 14 15 20 46 50 51 59 60 77 80 87 90 92  e _____
60 BUSCALINSEN   63 52 64 99 60 20 28 81  8 49 30 26 50 10 68  e _____
```

34.5 Respostas

```
1        13   10    7
2         3   10   15
3        19   24    4
```


Capítulo 35

Exercício Prático: 119 - Manuseio de Tabelas II

Para todos os algoritmos abaixo suponha-se duas variáveis globais:

```
1: inteiro TABELA[num]
2: inteiro ULTIMO
```

Onde $TABELA$ contém os dados a serem acessados. num é um valor suficiente para conter todos os dados necessários e $ULTIMO$ é um valor que aponta para o último valor válido da tabela.

Inclusão de itens em tabelas desordenadas Esta inclusão é fácil, bastando acrescentar o item ao final da área útil da tabela.

```
1: função INCLUSAONORD (inteiro CHAVE)
2: ULTIMO ← ULTIMO + 1
3: se ULTIMO > num então
4:     ...erro...
5: fimse
6: TABELA[ULTIMO] ← CHAVE
```

Qual a quantidade de operações para fazer uma inclusão em tabela não ordenada no caso de uma tabela de

elementos	operações
100	
10.000	
1.000.000	
n	

Exclusão de itens em tabelas desordenadas Há duas estratégias aqui: a primeira é eliminar o item puxando os que ficaram abaixo da exclusão para "tampar o buraco".

```
1: função EXCLUSAO1NORD (inteiro CHAVE)
2: inteiro i ← 1
3: enquanto TABELA[i] ≠ CHAVE faça
4:     i++
5:     se i > ULTIMO então
6:         ...erro...
```

```
 7:    fimse
 8: fimenquanto
 9: enquanto i < ULTIMO faça
10:    TABELA[i] ← TABELA [i+1]
11:    i++
12: fimenquanto
13: ULTIMO ← ULTIMO - 1
```

Qual a quantidade de operações para fazer uma exclusão do tipo 1; qual o número médio de operações para achar um item que está na tabela e qual o número médio de operações para concluir que um dado número não está na tabela

elementos	ops para exclusão	ops para achar	ops para concluir que não está
100			
10.000			
1.000.000			
n			

A segunda estratégia é incluir algum "indicador" ($filler$) que informe que o item foi eliminado.

```
1: função EXCLUSAO2NORD (inteiro CHAVE)
2: inteiro i ← 1
3: enquanto TABELA[i] ≠ CHAVE faça
4:    i++
5:    se i > ULTIMO então
6:       ...erro...
7:    fimse
8: fimenquanto
9: TABELA[i] ← 9999999
```

Qual a quantidade de operações para fazer uma exclusão do tipo 2; qual o número médio de operações para achar um item que está na tabela e qual o número médio de operações para concluir que um dado número não está na tabela

elementos	ops para exclusão	ops para achar	ops para concluir que não está
100			
10.000			
1.000.000			
n			

A vantagem desta segunda alternativa é que a exclusão é mais rápida, mas em compensação as buscas tendem a ficar mais demoaradas.

Busca binária Conjunto ordenado, ótimo desempenho

```
1: função BUSCABIN(inteiro CHAVE)
```

```
2: inteiro INIC,METADE,FIM
3: INIC ← 1
4: FIM ← ULTIMO
5: repita
6:    METADE ← ⌊ ((INIC + FIM)/2)
7:    se CHAVE ≤ TABELA[METADE] então
8:       FIM ← METADE - 1
9:    senão
10:      INIC ← METADE + 1
11:   fimse
12: até TABELA[METADE] = CHAVE ∨ INIC > FIM
13: se TABELA[METADE] = CHAVE  então
14:   devolva METADE
15: senão
16:   devolva -1
17: fimse
```

Qual a quantidade de operações para fazer pesquisa e qual o número médio de operações para concluir que um dado número não está na tabela

elementos	ops para pesquisa	ops para concluir que não está
100		
10.000		
1.000.000		
n		

Inclusão em tabela ordenada A contrapartida para poder fazer a busca ordenada é criar e manter a tabela em ordem. Para isso, a inclusão de novos elementos deve ocorrer em seus locais específicos e não no final, como vimos acima.

```
1: função INCLUSAOORD (inteiro CHAVE)
2: ULTIMO ← ULTIMO + 1
3: se ULTIMO > num então
4:    ...erro...
5: fimse
6: inteiro i ← 1
7: enquanto TABELA[i] < CHAVE faça
8:    i++
9: fimenquanto
10: k ← ULTIMO
11: enquanto k > i faça
12:    TABELA[k] ← TABELA[k-1]
13:    k−−
14: fimenquanto
15: TABELA[k] ← CHAVE
```

Qual a quantidade de operações para fazer uma inclusão em tabela ordenada no caso de uma tabela de

elementos	operações
100	
10.000	
1.000.000	
n	

35.1 Exercício 1

Escreva a seguir um algoritmo de uma função que Receba uma matriz global, de nome MAT-ENT, cujo conteúdo é:

MAT-ENT é uma matriz de 3 colunas, sendo todas inteiros, formada por

código da transação	valor da transação	código da cidade

O algoritmo deverá criar e imprimir uma segunda matriz, de apenas duas colunas

MAT-SAI é uma matriz de 2 colunas, sendo ambos inteiros, formada por

código da cidade	total dos valores de transações

Nesta matriz de saída, os valores de débito (cód 2) deverão ser totalizados, e quando houver **quebra** de cidade, uma nova linha deve ser introduzida na matriz de saida com o total. Note que a matriz de entrada está ordenada por cidade. (ENUNCIADO 2)

35.2 Exercício 2

Escreva a seguir um algoritmo de uma função que Receba uma matriz global, de nome MAT-VENDAS, cujo conteúdo é:

MAT-ENT é uma matriz de 3 colunas, sendo todas inteiros, formada por

número de parcelas	valor da transação	código da cidade

O número de parcelas pode ser de 1 a 6.

O algoritmo deverá criar e imprimir uma segunda matriz, de apenas duas colunas

MAT-SAI é uma matriz de 2 colunas, por 2 linhas sendo ambos inteiros, formada por

número de parcelas	quantidade de vendas

Nesta matriz de saída, deverão ser mostrados o número de parcelas campeão positivo (em quantidade de vendas) e o campeão negativo (idem), associado ao número de vendas de cada um. (ENUNCIADO 3)

35.3 Exercício 3

Escreva a seguir um algoritmo de uma função que Receba uma matriz global, de nome MAT-ENT, cujo conteúdo é:

MAT-ENT é uma matriz de 3 colunas, sendo todas inteiros, formada por

código do estado	área do estado	população do estado

O algoritmo deverá imprimir o código do estado de maior densidade populacional e o código do estado de menor densidade populacional, imprimindo também quantas vezes o primeiro é mais populoso do que o segundo. (ENUNCIADO 4)

35.4 Exercício 4

Escreva a seguir um algoritmo de uma função que Receba duas matrizes globais, de nomes MAT1 e MAT2, cujos conteúdos são:

MAT1 é uma matriz de 3 colunas, sendo todas inteiros, formada por

código da transação	valor da transação	código da cidade

MAT2 é uma matriz de 2 colunas, sendo ambos inteiros, formada por

código da cidade	estado a que pertence

O algoritmo deverá criar e imprimir uma terceira matriz, de apenas duas colunas

MATSAIDA é uma matriz de 2 colunas, sendo ambos inteiros, formada por

estado onde foi efetuada a transação	valor da transação

Nesta segunda matriz, apenas as transações de débito deverão ser incluídas. (ENUNCIADO 1)

Capítulo 36

Exercícios Práticos: 125a - Correção de algoritmos

36.1 Exercício 1

Construindo e consertando algoritmos - parte 1

Nesta folha, você receberá 5 algoritmos, que têm algumas lacunas representadas pelas constantes AAA, BBB, CCC, ... até III. Abaixo de cada algoritmo, é listada uma legenda que permitirá a substituição das constantes AAA,...,III por valores determinados.

Após a substituição, de duas uma:

- O algoritmo está correto
- O algoritmo contém um erro. Este erro deriva-se de uma (e apenas uma) substituição de constante por um valor errado.

Você não precisa determinar onde está o erro (se bem que isso é um interessante desafio), mas precisa apenas determinar que o algoritmo está errado.

Uma boa maneira de verificar isto é acompanhar as 3 execuções do algoritmo (correto) que acompanham o enunciado. Trata-se de algo similar a um chines reverso.

Identifique qual(is) dos 5 algoritmos estão corretos definindo A_n como: $A_1 \leftarrow 1$ se o algoritmo 1 está correto e $A_1 \leftarrow 0$ senão. Idem para A_2, A_3, A_4 e A_5.

Responda o resultado R

$$R = (A_1 \times 1) + (A_2 \times 2) + (A_3 \times 4) + (A_4 \times 8) + (A_5 \times 16)$$

Algoritmo 1 Escreva uma função que receba um número N (inteiro e positivo, não é preciso testar) e devolva a cadeia 'primo' se ele for PRIMO e 'não primo' senão.

```
1: funcao A01PRIM (N : inteiro) : inteiro
2: M,X,T : inteiro
3: M ← 0
4: X ← teto(N^0.5)
5: para  T de AAA a X faça
6:    Z ← BBB mod T
7:    se Z = CCC então
8:       M ← M + 1
9:    fimse
10: fimpara
```

```
11: se DDD = 0 então
12:    devolva EEE
13: senão
14:    devolva FFF
15: fimse
16: fimfuncao
```

Esta funcao correta, deu os seguintes resultados:

A01PRIM(30) = NAOPRIMO ; A01PRIM(39) = NAOPRIMO ; A01PRIM(31) = PRIMO .

As substituições a fazer são:

AAA=2 ; BBB=N ; CCC=0 ; DDD=M ; EEE=primo e FFF=nao primo .

Algoritmo 2 Escreva uma função que receba um número N (inteiro e positivo, não é preciso testar) e devolva um inteiro contendo a soma dos divisores inteiros de N, excluindo-se desta soma ele mesmo e a unidade.

```
 1: funcao A02SOMD (N : inteiro) : inteiro
 2: M,T,Z : inteiro
 3: M ← AAA
 4: para  T de BBB a CCC faça
 5:    Z ← DDD mod EEE
 6:    se FFF = Z então
 7:       M ← GGG + HHH
 8:    fimse
 9: fimpara
10: devolva III
11: fimfuncao
```

Esta funcao correta, deu os seguintes resultados:

A02SOMD(16) = 14 ; A02SOMD(89) = 0 ; A02SOMD(29) = 0 .

As substituições a fazer são:

AAA=0 ; BBB=2 ; CCC=N-1 ; DDD=N ; EEE=T ; FFF=0 ; GGG=M ; HHH=T ; III=M .

Algoritmo 3 Escreva uma função que receba dois inteiros N e X Se X for maior que zero, a funcao deve calcular e devolver o número N^X e se não for deve devolver -1.

```
 1: funcao A03POTE (N,X : inteiro) : inteiro
 2: R : inteiro
 3: R ← AAA
 4: se X > 0 então
 5:    enquanto BBB >0 faça
 6:       R ← R CCC N
 7:       X ← X DDD 1
 8:    fimenquanto
 9:    devolva EEE
10: senão
11:    devolva FFF
12: fimse
13: fimfuncao
```

Esta funcao correta, deu os seguintes resultados:

A03POTE(3 4) = 81 ; A03POTE(5 6) = 15625 ; A03POTE(4 6) = 4096 .

As substituições a fazer são:

AAA=0 ; BBB=X ; CCC=vezes (multiplicacao) ; DDD=- (subtracao) ; EEE=R ; FFF=-1 .

Algoritmo 4 Escreva uma função que receba três inteiros D, M e A representando uma data (D é o dia, M é o mês e A é o ano). O algoritmo deve devolver 1 se a data estiver errada e 0 senão.

```
 1: funcao A04DATA (D, M, A : inteiro) : inteiro
 2: biss, erro : inteiro
 3: se 0 = (A mod AAA) então
 4:    biss←1
 5: senão
 6:    se 0 = (BBB mod 100) então
 7:       biss←0
 8:    senão
 9:       se 0 = (A mod 4) então
10:          biss←CCC
11:       senão
12:          biss←DDD
13:       fimse
14:    fimse
15: fimse
16: erro ← 0
17: se M = EEE então
18:    se biss = 1 então
19:       se D > FFF então
20:          erro ← 1
21:       fimse
22:    senão
23:       se D > GGG então
24:          erro ← 1
25:       fimse
26:    fimse
27: senão
28:    se (M=4)∨(M=6)∨(M=9)∨(M=11) então
29:       se D > HHH então
30:          erro ← 1
31:       fimse
32:    senão
33:       se D > III então
34:          erro ← 1
35:       fimse
36:    fimse
37: fimse
38: se M > JJJ então
39:    erro ← 1
40: fimse
41: retorne erro
42: fimfuncao
```

Esta funcao correta, deu os seguintes resultados:
A04DATA(29 10 7202) = 0 ; A04DATA(29 2 2012) = 0 ; A04DATA(29 2 1800) = 1
.

As substituições a fazer são:

AAA=400 ; BBB=A (ano) ; CCC=1 e DDD=0 ; EEE=2 ; FFF=29 ; GGG=28 ; HHH=30 ; III=31 ; JJJ=12 .

Brinquedos "PIRRALHOS ENDIABRADOS" é um grande distribuidor de presentes em todo o país. Recentemente, a empresa teve a oportunidade de comprar pequenos brinquedos, todos embalados em caixas retangulares. O objetivo da compra, foi colocar cada brinquedo em uma esfera colorida, para revendê-los como surpresa, mais ou menos como o Kinder ovo. Existem esferas de raios 10, 20 e 30 cm. Cada brinquedo, tem as suas 3 dimensões A, B e C, medidas em centímetros. Escreva uma função que receba A,B,C e retorne e o raio da menor esfera possível. Todos os brinquedos caberão em uma das esferas.

```
1: funcao A05DIAG (X, Y, Z : inteiro) : inteiro
2: inteiro diag
3: diag ← (X^AAA) + (Y^AAA) + (Z^AAA)
4: diag ← √diag
5: se diag > BBB então
6:    devolva 30
7: senão
8:    se diag > CCC então
9:       devolva 20
10:   senão
11:      devolva 10
12:   fimse
13: fimse
14: fimfuncao
```

Esta funcao correta, deu os seguintes resultados:

A05DIAG(8 5 5) = 10 ; A05DIAG(28 25 29) = 30 ; A05DIAG(4 5 10) = 10 .

As substituições a fazer são:

AAA=3 ; BBB=40 ; CCC=20 .

☞ **Responda**

R=

36.2 Exercício 2

Construindo e consertando algoritmos - parte 1

Nesta folha, você receberá 5 algoritmos, que têm algumas lacunas representadas pelas constantes AAA, BBB, CCC, ... até III. Abaixo de cada algoritmo, é listada uma legenda que permitirá a substituição das constantes AAA,...,III por valores determinados.

Após a substituição, de duas uma:

- O algoritmo está correto
- O algoritmo contém um erro. Este erro deriva-se de uma (e apenas uma) substituição de constante por um valor errado.

Você não precisa determinar onde está o erro (se bem que isso é um interessante desafio), mas precisa apenas determinar que o algoritmo está errado.

Uma boa maneira de verificar isto é acompanhar as 3 execuções do algoritmo (correto) que acompanham o enunciado. Trata-se de algo similar a um chines reverso.

Identifique qual(is) dos 5 algoritmos estão corretos definindo A_n como: $A_1 \leftarrow 1$ se o algoritmo 1 está correto e $A_1 \leftarrow 0$ senão. Idem para A_2, A_3, A_4 e A_5.

Responda o resultado R

$$R = (A_1 \times 1) + (A_2 \times 2) + (A_3 \times 4) + (A_4 \times 8) + (A_5 \times 16)$$

Algoritmo 1 Escreva uma função que receba um número N (inteiro e positivo, não é preciso testar) e devolva a cadeia 'primo' se ele for PRIMO e 'não primo' senão.

```
 1: funcao A01PRIM (N : inteiro) : inteiro
 2: M,X,T : inteiro
 3: M ← 0
 4: X ← teto(N^0.5)
 5: para  T de AAA a X faça
 6:    Z ← BBB mod T
 7:    se Z = CCC então
 8:       M ← M + 1
 9:    fimse
10: fimpara
11: se DDD = 0 então
12:    devolva EEE
13: senão
14:    devolva FFF
15: fimse
16: fimfuncao
```

Esta funcao correta, deu os seguintes resultados:
A01PRIM(17) = PRIMO ; A01PRIM(39) = NAOPRIMO ; A01PRIM(23) = PRIMO
.

As substituições a fazer são:
AAA=2 ; BBB=N ; CCC=0 ; DDD=M ; EEE=nao primo e FFF=primo .

Algoritmo 2 Escreva uma função que receba um número N (inteiro e positivo, não é preciso testar) e devolva um inteiro contendo a soma dos divisores inteiros de N, excluindo-se desta soma ele mesmo e a unidade.

```
1: funcao A02SOMD (N : inteiro) : inteiro
2: M,T,Z : inteiro
```



```
 3: M ← AAA
 4: para  T de BBB a CCC faça
 5:    Z ← DDD mod EEE
 6:    se FFF = Z então
 7:       M ← GGG + HHH
 8:    fimse
 9: fimpara
10: devolva III
11: fimfuncao
```

Esta funcao correta, deu os seguintes resultados:
A02SOMD(16) = 14 ; A02SOMD(86) = 45 ; A02SOMD(29) = 0 .
As substituições a fazer são:
AAA=0 ; BBB=2 ; CCC=N-1 ; DDD=N ; EEE=T ; FFF=0 ; GGG=M ; HHH=T ; III=M .

Algoritmo 3 Escreva uma função que receba dois inteiros N e X Se X for maior que zero, a funcao deve calcular e devolver o número N^X e se não for deve devolver -1.

```
 1: funcao A03POTE (N,X : inteiro) : inteiro
 2: R : inteiro
 3: R ← AAA
 4: se X > 0 então
 5:    enquanto BBB >0 faça
 6:       R ← R CCC N
 7:       X ← X DDD 1
 8:    fimenquanto
 9:    devolva EEE
10: senão
11:    devolva FFF
12: fimse
13: fimfuncao
```

Esta funcao correta, deu os seguintes resultados:
A03POTE(3 6) = 729 ; A03POTE(6 3) = 216 ; A03POTE(6 5) = 7776 .
As substituições a fazer são:
AAA=0 ; BBB=X ; CCC=vezes (multiplicacao) ; DDD=- (subtracao) ; EEE=R ; FFF=-1 .

Algoritmo 4 Escreva uma função que receba três inteiros D, M e A representando uma data (D é o dia, M é o mês e A é o ano). O algoritmo deve devolver 1 se a data estiver errada e 0 senão.

```
 1: funcao A04DATA (D, M, A : inteiro) : inteiro
 2: biss, erro : inteiro
 3: se 0 = (A mod AAA) então
 4:    biss←1
 5: senão
 6:    se 0 = (BBB mod 100) então
 7:       biss←0
 8:    senão
 9:       se 0 = (A mod 4) então
10:          biss←CCC
11:       senão
12:          biss←DDD
```

```
13:       fimse
14:    fimse
15: fimse
16: erro ← 0
17: se M = EEE então
18:    se biss = 1 então
19:       se D > FFF então
20:          erro ← 1
21:       fimse
22:    senão
23:       se D > GGG então
24:          erro ← 1
25:       fimse
26:    fimse
27: senão
28:    se (M=4)∨(M=6)∨(M=9)∨(M=11) então
29:       se D > HHH então
30:          erro ← 1
31:       fimse
32:    senão
33:       se D > III então
34:          erro ← 1
35:       fimse
36:    fimse
37: fimse
38: se M > JJJ então
39:    erro ← 1
40: fimse
41: retorne erro
42: fimfuncao
```

Esta funcao correta, deu os seguintes resultados:
A04DATA(28 11 7636) = 0 ; A04DATA(27 2 1852) = 0 ; A04DATA(29 2 2000) = 0
.

As substituições a fazer são:
AAA=400 ; BBB=A (ano) ; CCC=1 e DDD=0 ; EEE=2 ; FFF=29 ; GGG=28 ; HHH=30 ; III=30 ; JJJ=12 .

Brinquedos "PIRRALHOS ENDIABRADOS" é um grande distribuidor de presentes em todo o país. Recentemente, a empresa teve a oportunidade de comprar pequenos brinquedos, todos embalados em caixas retangulares. O objetivo da compra, foi colocar cada brinquedo em uma esfera colorida, para revendê-los como surpresa, mais ou menos como o Kinder ovo. Existem esferas de raios 10, 20 e 30 cm. Cada brinquedo, tem as suas 3 dimensões A, B e C, medidas em centímetros. Escreva uma função que receba A,B,C e retorne e o raio da menor esfera possível. Todos os brinquedos caberão em uma das esferas.

```
1: funcao A05DIAG (X, Y, Z : inteiro) : inteiro
2: inteiro diag
3: diag ← (X^AAA) + (Y^AAA) + (Z^AAA)
4: diag ← √diag
5: se diag > BBB então
6:    devolva 30
7: senão
```



```
 8:    se diag > CCC então
 9:       devolva 20
10:    senão
11:       devolva 10
12:    fimse
13: fimse
14: fimfuncao
```

Esta funcao correta, deu os seguintes resultados:
A05DIAG(2 6 8) = 10 ; A05DIAG(25 27 28) = 30 ; A05DIAG(5 1 7) = 10 .
As substituições a fazer são:
AAA=2 ; BBB=40 ; CCC=20 .

☞ **Responda**

R=

36.3 Exercício 3

Construindo e consertando algoritmos - parte 1

Nesta folha, você receberá 5 algoritmos, que têm algumas lacunas representadas pelas constantes AAA, BBB, CCC, ... até III. Abaixo de cada algoritmo, é listada uma legenda que permitirá a substituição das constantes AAA,...,III por valores determinados.

Após a substituição, de duas uma:

- O algoritmo está correto

- O algoritmo contém um erro. Este erro deriva-se de uma (e apenas uma) substituição de constante por um valor errado.

Você não precisa determinar onde está o erro (se bem que isso é um interessante desafio), mas precisa apenas determinar que o algoritmo está errado.

Uma boa maneira de verificar isto é acompanhar as 3 execuções do algoritmo (correto) que acompanham o enunciado. Trata-se de algo similar a um chines reverso.

Identifique qual(is) dos 5 algoritmos estão corretos definindo A_n como: $A_1 \leftarrow 1$ se o algoritmo 1 está correto e $A_1 \leftarrow 0$ senão. Idem para A_2, A_3, A_4 e A_5.

Responda o resultado R

$$R = (A_1 \times 1) + (A_2 \times 2) + (A_3 \times 4) + (A_4 \times 8) + (A_5 \times 16)$$

Algoritmo 1 Escreva uma função que receba um número N (inteiro e positivo, não é preciso testar) e devolva a cadeia 'primo' se ele for PRIMO e 'não primo' senão.

```
1: funcao A01PRIM (N : inteiro) : inteiro
2: M,X,T : inteiro
3: M ← 0
4: X ← teto(N^0.5)
5: para  T de AAA a X faça
6:    Z ← BBB mod T
7:    se Z = CCC então
8:       M ← M + 1
9:    fimse
10: fimpara
11: se DDD = 0 então
12:    devolva EEE
13: senão
14:    devolva FFF
15: fimse
16: fimfuncao
```

Esta funcao correta, deu os seguintes resultados:
A01PRIM(26) = NAOPRIMO ; A01PRIM(71) = PRIMO ; A01PRIM(37) = PRIMO
.

As substituições a fazer são:
AAA=2 ; BBB=N ; CCC=0 ; DDD=N ; EEE=primo e FFF=nao primo .

Algoritmo 2 Escreva uma função que receba um número N (inteiro e positivo, não é preciso testar) e devolva um inteiro contendo a soma dos divisores inteiros de N, excluindo-se desta soma ele mesmo e a unidade.

```
1: funcao A02SOMD (N : inteiro) : inteiro
2: M,T,Z : inteiro
3: M ← AAA
4: para  T de BBB a CCC faça
5:    Z ← DDD mod EEE
6:    se FFF = Z então
7:       M ← GGG + HHH
8:    fimse
9: fimpara
10: devolva III
11: fimfuncao
```


Esta funcao correta, deu os seguintes resultados:
A02SOMD(20) = 21 ; A02SOMD(27) = 12 ; A02SOMD(71) = 0 .
As substituições a fazer são:
AAA=0 ; BBB=2 ; CCC=N-1 ; DDD=N ; EEE=T ; FFF=0 ; GGG=M ; HHH=T ; III=2 vezes M .

Algoritmo 3 Escreva uma função que receba dois inteiros N e X Se X for maior que zero, a funcao deve calcular e devolver o número N^X e se não for deve devolver -1.

```
 1: funcao A03POTE (N,X : inteiro) : inteiro
 2: R : inteiro
 3: R ← AAA
 4: se X > 0 então
 5:    enquanto BBB >0 faça
 6:       R ← R CCC N
 7:       X ← X DDD 1
 8:    fimenquanto
 9:    devolva EEE
10: senão
11:    devolva FFF
12: fimse
13: fimfuncao
```

Esta funcao correta, deu os seguintes resultados:
A03POTE(4 4) = 256 ; A03POTE(6 4) = 1296 ; A03POTE(5 3) = 125 .
As substituições a fazer são:
AAA=1 ; BBB=X ; CCC=+ (adicao) ; DDD=- (subtracao) ; EEE=R ; FFF=-1 .

Algoritmo 4 Escreva uma função que receba três inteiros D, M e A representando uma data (D é o dia, M é o mês e A é o ano). O algoritmo deve devolver 1 se a data estiver errada e 0 senão.

```
 1: funcao A04DATA (D, M, A : inteiro) : inteiro
 2: biss, erro : inteiro
 3: se 0 = (A mod AAA) então
 4:    biss←1
 5: senão
 6:    se 0 = (BBB mod 100) então
 7:       biss←0
 8:    senão
 9:       se 0 = (A mod 4) então
10:          biss←CCC
11:       senão
12:          biss←DDD
13:       fimse
14:    fimse
15: fimse
16: erro ← 0
17: se M = EEE então
18:    se biss = 1 então
19:       se D > FFF então
20:          erro ← 1
21:       fimse
22:    senão
```

```
23:       se D > GGG então
24:         erro ← 1
25:       fimse
26:     fimse
27:   senão
28:     se (M=4)∨(M=6)∨(M=9)∨(M=11) então
29:       se D > HHH então
30:         erro ← 1
31:       fimse
32:     senão
33:       se D > III então
34:         erro ← 1
35:       fimse
36:     fimse
37:   fimse
38:   se M > JJJ então
39:     erro ← 1
40:   fimse
41:   retorne erro
42: fimfuncao
```

Esta funcao correta, deu os seguintes resultados:
A04DATA(25 12 8360) = 0 ; A04DATA(28 2 1968) = 0 ; A04DATA(29 2 2000) = 0 .

As substituições a fazer são:
AAA=400 ; BBB=A (ano) ; CCC=1 e DDD=0 ; EEE=2 ; FFF=29 ; GGG=28 ; HHH=30 ; III=31 ; JJJ=12 .

Brinquedos "PIRRALHOS ENDIABRADOS" é um grande distribuidor de presentes em todo o país. Recentemente, a empresa teve a oportunidade de comprar pequenos brinquedos, todos embalados em caixas retangulares. O objetivo da compra, foi colocar cada brinquedo em uma esfera colorida, para revendê-los como surpresa, mais ou menos como o Kinder ovo. Existem esferas de raios 10, 20 e 30 cm. Cada brinquedo, tem as suas 3 dimensões A, B e C, medidas em centímetros. Escreva uma função que receba A,B,C e retorne e o raio da menor esfera possível. Todos os brinquedos caberão em uma das esferas.

```
1: funcao A05DIAG (X, Y, Z : inteiro) : inteiro
2: inteiro diag
3: diag ← (X^AAA) + (Y^AAA) + (Z^AAA)
4: diag ← √diag
5: se diag > BBB então
6:   devolva 30
7: senão
8:   se diag > CCC então
9:     devolva 20
10:  senão
11:    devolva 10
12:  fimse
13: fimse
14: fimfuncao
```

Esta funcao correta, deu os seguintes resultados:
A05DIAG(4 2 8) = 10 ; A05DIAG(27 26 28) = 30 ; A05DIAG(9 7 7) = 10 .

As substituições a fazer são:
AAA=2 ; BBB=20 ; CCC=20 .

☞ **Responda**

R=

36.4 Exercício 4

Construindo e consertando algoritmos - parte 1

Nesta folha, você receberá 5 algoritmos, que têm algumas lacunas representadas pelas constantes AAA, BBB, CCC, ... até III. Abaixo de cada algoritmo, é listada uma legenda que permitirá a substituição das constantes AAA,...,III por valores determinados.

Após a substituição, de duas uma:

- O algoritmo está correto
- O algoritmo contém um erro. Este erro deriva-se de uma (e apenas uma) substituição de constante por um valor errado.

Você não precisa determinar onde está o erro (se bem que isso é um interessante desafio), mas precisa apenas determinar que o algoritmo está errado.

Uma boa maneira de verificar isto é acompanhar as 3 execuções do algoritmo (correto) que acompanham o enunciado. Trata-se de algo similar a um chines reverso.

Identifique qual(is) dos 5 algoritmos estão corretos definindo A_n como: $A_1 \leftarrow 1$ se o algoritmo 1 está correto e $A_1 \leftarrow 0$ senão. Idem para A_2, A_3, A_4 e A_5.

Responda o resultado R

$$R = (A_1 \times 1) + (A_2 \times 2) + (A_3 \times 4) + (A_4 \times 8) + (A_5 \times 16)$$

Algoritmo 1 Escreva uma função que receba um número N (inteiro e positivo, não é preciso testar) e devolva a cadeia 'primo' se ele for PRIMO e 'não primo' senão.

```
1: funcao A01PRIM (N : inteiro) : inteiro
2: M,X,T : inteiro
3: M ← 0
4: X ← teto(N^0.5)
5: para T de AAA a X faça
6:    Z ← BBB mod T
7:    se Z = CCC então
8:       M ← M + 1
9:    fimse
10: fimpara
11: se DDD = 0 então
12:    devolva EEE
13: senão
14:    devolva FFF
15: fimse
16: fimfuncao
```

Esta funcao correta, deu os seguintes resultados:

A01PRIM(20) = NAOPRIMO ; A01PRIM(54) = NAOPRIMO ; A01PRIM(67) = PRIMO .

As substituições a fazer são:

AAA=1 ; BBB=N ; CCC=0 ; DDD=M ; EEE=primo e FFF=nao primo .

Algoritmo 2 Escreva uma função que receba um número N (inteiro e positivo, não é preciso testar) e devolva um inteiro contendo a soma dos divisores inteiros de N, excluindo-se desta soma ele mesmo e a unidade.

```
1: funcao A02SOMD (N : inteiro) : inteiro
2: M,T,Z : inteiro
3: M ← AAA
4: para T de BBB a CCC faça
5:    Z ← DDD mod EEE
6:    se FFF = Z então
7:       M ← GGG + HHH
8:    fimse
9: fimpara
10: devolva III
11: fimfuncao
```

Esta funcao correta, deu os seguintes resultados:

A02SOMD(24) = 35 ; A02SOMD(85) = 22 ; A02SOMD(29) = 0 .

As substituições a fazer são:

AAA=0 ; BBB=2 ; CCC=N-1 ; DDD=N ; EEE=T ; FFF=0 ; GGG=N ; HHH=T ; III=M .

Algoritmo 3 Escreva uma função que receba dois inteiros N e X Se X for maior que zero, a funcao deve calcular e devolver o número N^X e se não for deve devolver -1.

```
1: funcao A03POTE (N,X : inteiro) : inteiro
```



```
 2: R : inteiro
 3: R ← AAA
 4: se X > 0 então
 5:    enquanto BBB >0 faça
 6:       R ← R CCC N
 7:       X ← X DDD 1
 8:    fimenquanto
 9:    devolva EEE
10: senão
11:    devolva FFF
12: fimse
13: fimfuncao
```

Esta funcao correta, deu os seguintes resultados:
A03POTE(6 3) = 216 ; A03POTE(5 3) = 125 ; A03POTE(3 3) = 27 .
As substituições a fazer são:
AAA=1 ; BBB=X ; CCC=vezes (multiplicacao) ; DDD=+ (adicao) ; EEE=R ; FFF=-1 .

Algoritmo 4 Escreva uma função que receba três inteiros D, M e A representando uma data (D é o dia, M é o mês e A é o ano). O algoritmo deve devolver 1 se a data estiver errada e 0 senão.

```
 1: funcao A04DATA (D, M, A : inteiro) : inteiro
 2: biss, erro : inteiro
 3: se 0 = (A mod AAA) então
 4:    biss←1
 5: senão
 6:    se 0 = (BBB mod 100) então
 7:       biss←0
 8:    senão
 9:       se 0 = (A mod 4) então
10:          biss←CCC
11:       senão
12:          biss←DDD
13:       fimse
14:    fimse
15: fimse
16: erro ← 0
17: se M = EEE então
18:    se biss = 1 então
19:       se D > FFF então
20:          erro ← 1
21:       fimse
22:    senão
23:       se D > GGG então
24:          erro ← 1
25:       fimse
26:    fimse
27: senão
28:    se (M=4)∨(M=6)∨(M=9)∨(M=11) então
29:       se D > HHH então
30:          erro ← 1
31:       fimse
```

```
32:    senão
33:      se D > III então
34:        erro ← 1
35:      fimse
36:    fimse
37: fimse
38: se M > JJJ então
39:    erro ← 1
40: fimse
41: retorne erro
42: fimfuncao
```

Esta funcao correta, deu os seguintes resultados:

A04DATA(25 10 32) = 0 ; A04DATA(29 2 2080) = 0 ; A04DATA(29 2 1900) = 1 .

As substituições a fazer são:

AAA=400 ; BBB=A (ano) ; CCC=1 e DDD=0 ; EEE=2 ; FFF=29 ; GGG=28 ; HHH=31 ; III=31 ; JJJ=12 .

Brinquedos "PIRRALHOS ENDIABRADOS" é um grande distribuidor de presentes em todo o país. Recentemente, a empresa teve a oportunidade de comprar pequenos brinquedos, todos embalados em caixas retangulares. O objetivo da compra, foi colocar cada brinquedo em uma esfera colorida, para revendê-los como surpresa, mais ou menos como o Kinder ovo. Existem esferas de raios 10, 20 e 30 cm. Cada brinquedo, tem as suas 3 dimensões A, B e C, medidas em centímetros. Escreva uma função que receba A,B,C e retorne e o raio da menor esfera possível. Todos os brinquedos caberão em uma das esferas.

```
1: funcao A05DIAG (X, Y, Z : inteiro) : inteiro
2: inteiro diag
3: diag ← (X^AAA) + (Y^AAA) + (Z^AAA)
4: diag ← √diag
5: se diag > BBB então
6:    devolva 30
7: senão
8:    se diag > CCC então
9:       devolva 20
10:   senão
11:      devolva 10
12:   fimse
13: fimse
14: fimfuncao
```

Esta funcao correta, deu os seguintes resultados:

A05DIAG(2 9 2) = 10 ; A05DIAG(28 28 26) = 30 ; A05DIAG(2 6 3) = 10 .

As substituições a fazer são:

AAA=2 ; BBB=40 ; CCC=20 .

☞ **Responda**

R=

36.5 Respostas

```
1    0 0 1 0 1   11
2    5 0 1 8 0   18
3    4 9 3 0 2    8
4    1 7 4 7 0   16
```

Capítulo 37

Exercícios Práticos: 125b - Correção de algoritmos

37.1 Exercício 1

Construindo e consertando algoritmos - parte 2

Nesta folha, você receberá 5 algoritmos, que têm algumas lacunas representadas pelas constantes AAA, BBB, CCC, ... até III. Abaixo de cada algoritmo, é listada uma legenda que permitirá a substituição das constantes AAA,...,III por valores determinados.

Após a substituição, de duas uma:

- O algoritmo está correto
- O algoritmo contém um erro. Este erro deriva-se de uma (e apenas uma) substituição de constante por um valor errado.

Você não precisa determinar onde está o erro (se bem que isso é um interessante desafio), mas precisa apenas determinar que o algoritmo está errado.

Uma boa maneira de verificar isto é acompanhar as 3 execuções do algoritmo (correto) que acompanham o enunciado. Trata-se de algo similar a um chines reverso.

Identifique qual(is) dos 5 algoritmos estão corretos definindo A_n como: $A_1 \leftarrow 1$ se o algoritmo 1 está correto e $A_1 \leftarrow 0$ senão. Idem para A_2, A_3, A_4 e A_5.

Responda o resultado R

$$R = (A_1 \times 1) + (A_2 \times 2) + (A_3 \times 4) + (A_4 \times 8) + (A_5 \times 16)$$

Algoritmo 1 Escreva uma função que receba um vetor de 10 números inteiros e devolva a posição daquele número que é o maior do vetor. Se houver dois ou mais iguais a este maior, deve-se retornar o endereço do primeiro. Por exemplo, se o vetor for 4, 7, 2, 19, 21, 8, 7, 19, 8, 6 a resposta deve ser 5, já que o 21 (o maior) é o quinto elemento. Se o vetor for 4, 12, 5, 7, 12, 7, 1, 2, 3, 8, a resposta deve ser 2.

```
1: funcao B01ACHM (N[10] : inteiro) : inteiro
2: MAI,QUA,I : inteiro
3: I ← AAA
4: BBB ← -99999
5: enquanto CCC < 11 faça
6:    se N[I] > DDD então
7:       MAI ← N[I]
```

```
 8:       EEE ← I
 9:    fimse
10:    FFF ← I + 1
11: fimenquanto
12: devolva GGG
13: fimfuncao
```

Esta funcao correta, deu os seguintes resultados:

B01ACHM(1 1 6 5 3 4 7 5 7 4) = 7 ; B01ACHM(18 8 17 12 4 7 19 10 13 6) = 7 ; B01ACHM(1 3 4 3 4 4 1 3 3 1) = 3 .

As substituições a fazer são:

AAA=1 ; BBB=MAI ; CCC=I ; DDD=MAI ; EEE=QUA ; FFF=I ; GGG=QUA .

Algoritmo 2 Escreva uma função que receba um número K (inteiro, não é preciso testar) e um vetor N de 10 inteiros e e devolva a posição de K em N. Se houver mais de 1 valor de K em N, deve retornar o primeiro. Se não houver K em N, retornar -1.

Por exemplo, Se K=8 e N=1 3 5 7 9 2 4 6 8 10, a resposta deve ser 9. Se K=7 e N é o mesmo, a resposta é 4. Se K=13 e N é o mesmo, a resposta é -1.

```
 1: funcao B02ACHKK (K, N[10] : inteiro) : inteiro
 2: inteiro RES,I
 3: RES ← AAA
 4: I ← BBB
 5: enquanto CCC < 11 faça
 6:    se N[I] = DDD então
 7:       RES ← EEE
 8:       I ← FFF
 9:    fimse
10:    I ← I + 1
11: fimenquanto
12: devolva GGG
13: fimfuncao
```

Esta funcao correta, deu os seguintes resultados:

B02ACHK(12,20 30 25 26 21 23 12 14 15 27)=7 ; B02ACHK(15,23 30 11 12 25 24 19 15 20 18)=8 ; B02ACHK(17,13 37 38 26 34 20 15 33 27 32)=-1 .

As substituições a fazer são:

AAA=-1 ; BBB=1 ; CCC=I ; DDD=K ; EEE=I ; FFF=11 (ou qualquer > 10) ; GGG=RES .

Algoritmo 3 Escreva uma funcao que receba um vetor N de 10 elementos inteiros, em princípio sem nenhuma ordem (ou seja, desordenado) e uma chave K também inteira e devolva a quantidade de vezes que K aparece em N.

Por exemplo, se N=1 3 5 7 2 3 4 5 6 10 e K=5 a resposta é 2. Se K=18, a resposta é 0 e se K=1, a reposta é 1.

```
 1: funcao B03CONK (K,N[10] : inteiro) : inteiro
 2: inteiro QTD,I
 3: QTD ← AAA
 4: para  I de BBB a CCC passo DDD faça
 5:    se N[I] = EEE então
 6:       QTD ← FFF+ 1
 7:    fimse
 8: fimpara
 9: devolva GGG
```

```
10: fimfuncao
```

Esta funcao correta, deu os seguintes resultados:
B03CONK(18,12 15 13 12 18 14 18 13 15 12)=2 ; B03CONK(15,12 18 13 16 16 15 12 15 13 18)=2 ; B03CONK(17,15 31 16 34 33 25 38 37 17 11)=1 .
As substituições a fazer são:
AAA=0 ; BBB=1 ; CCC=10 ; DDD=1 ; EEE=K ; FFF=QTD ; GGG=QTD .

Algoritmo 4 Escreva uma função que receba um vetor ordenado de 11 inteiros. A função deve devolver 3 valores (um real e dois inteiros), a saber:

média A soma dos elementos dividido por 11

moda o valor mais freqüente no vetor

mediana O valor do meio (neste caso o N[6])

```
 1: funcao B04ESTA (N[11] : inteiro) : real,inteiro,inteiro
 2: MED : real
 3: I,J,MOD,MAN,SOM,ASA,CTDR[11] : inteiro
 4: I ← 1
 5: SOM ← AAA
 6: enquanto I < BBB faça
 7:    SOM ← CCC + N[I]
 8:    I ← I + 1
 9: fimenquanto
10: DDD ← SOM ÷ 11
11: EEE ← N[6]
12: para  I de 1 a 11 faça
13:    SOM ← 0
14:    para  J de 1 a 11 faça
15:       se N[I] = N[J] então
16:          SOM ← SOM + 1
17:       fimse
18:    fimpara
19:    CTDR[I] ← FFF
20: fimpara
21: MAI ← -9999
22: para  J de 1 a 11 faça
23:    se CTDR[J] > GGG então
24:       MAI ← CTDR[J]
25:       ASA ← HHH
26:    fimse
27: fimpara
28: III ← N[ASA]
29: devolva MED,MOD,MAN
30: fimfuncao
```

Esta funcao correta, deu os seguintes resultados:
B04ESTA(1 2 2 4 4 6 6 7 7 7 8) = 4.9 7 6 ; B04ESTA(1 4 5 5 6 6 6 6 7 7 8) = 5.5 6 6 ; B04ESTA(1 2 2 2 3 3 4 4 6 6 6) = 3.5 2 3 .
As substituições a fazer são:
AAA=0 ; BBB=12 ; CCC=SOM ; DDD=MED ; EEE=MAN ; FFF=SOM ; GGG=MAI ; HHH=J ; III=MOD .

O vetor está em ordem ? Escreva o algoritmo de uma função que receba um vetor de 10 inteiros e devolva 0 se o vetor está em ordem crescente e 1 senão. Um vetor V está em ordem crescente para quaisquer i e j, sendo que $j > i$, é válida a expressão $V[j] \geq V[i]$. Note a presença da igualdade na condição. Além disso, essa expressão tem que ser válida para todos os pares dentro do vetor.

Por exemplo se o vetor for 1,3,4,5,6,7,9,10,16,21 a resposta é 0. Se o vetor for 1,3,4,5,6,8,7,10,16,21 a resposta é 1 e se for 2,2,2,2,2,2,2,2,2,2 a resposta é 0.

```
 1: funcao B05ESORN (N[10] : inteiro) : inteiro27
 2: inteiro I, RES
 3: RES ← AAA
 4: I ← 1
 5: enquanto I < BBB faça
 6:    se N[I] >N[I+1] então
 7:       RES ← 1
 8:    fimse
 9:    I ← I + 1
10: fimenquanto
11: devolva CCC
12: fimfuncao
```

Esta funcao correta, deu os seguintes resultados:

B05ESOR(2 5 6 7 12 27 35 44 60 86) = 0 ; B05ESOR(1 4 17 22 29 41 95 51 92 93) = 1 ; B05ESOR(5 7 20 35 37 38 68 55 78 79) = 1 .

As substituições a fazer são:

AAA=0 ; BBB=10 ; CCC=I .

☞ **Responda**

R=

37.2 Exercício 2

Construindo e consertando algoritmos - parte 2

Nesta folha, você receberá 5 algoritmos, que têm algumas lacunas representadas pelas constantes AAA, BBB, CCC, ... até III. Abaixo de cada algoritmo, é listada uma legenda que permitirá a substituição das constantes AAA,...,III por valores determinados.

Após a substituição, de duas uma:

- O algoritmo está correto
- O algoritmo contém um erro. Este erro deriva-se de uma (e apenas uma) substituição de constante por um valor errado.

Você não precisa determinar onde está o erro (se bem que isso é um interessante desafio), mas precisa apenas determinar que o algoritmo está errado.

Uma boa maneira de verificar isto é acompanhar as 3 execuções do algoritmo (correto) que acompanham o enunciado. Trata-se de algo similar a um chines reverso.

Identifique qual(is) dos 5 algoritmos estão corretos definindo A_n como: $A_1 \leftarrow 1$ se o algoritmo 1 está correto e $A_1 \leftarrow 0$ senão. Idem para A_2, A_3, A_4 e A_5.

Responda o resultado R

$$R = (A_1 \times 1) + (A_2 \times 2) + (A_3 \times 4) + (A_4 \times 8) + (A_5 \times 16)$$

Algoritmo 1 Escreva uma função que receba um vetor de 10 números inteiros e devolva a posição daquele número que é o maior do vetor. Se houver dois ou mais iguais a este maior, deve-se retornar o endereço do primeiro. Por exemplo, se o vetor for 4, 7, 2, 19, 21, 8, 7, 19, 8, 6 a resposta deve ser 5, já que o 21 (o maior) é o quinto elemento. Se o vetor for 4, 12, 5, 7, 12, 7, 1, 2, 3, 8, a resposta deve ser 2.

```
1: funcao B01ACHM (N[10] : inteiro) : inteiro
2: MAI,QUA,I : inteiro
3: I ← AAA
4: BBB ← -99999
5: enquanto CCC < 11 faça
6:    se N[I] > DDD então
7:       MAI ← N[I]
8:       EEE ← I
9:    fimse
10:   FFF ← I + 1
11: fimenquanto
12: devolva GGG
13: fimfuncao
```

Esta funcao correta, deu os seguintes resultados:

B01ACHM(2 4 6 6 1 2 6 4 1 7) = 10 ; B01ACHM(14 16 5 18 17 8 11 2 19 20) = 10 ; B01ACHM(4 2 1 5 4 2 5 1 2 2) = 4 .

As substituições a fazer são:

AAA=1 ; BBB=MAI ; CCC=I ; DDD=MAI ; EEE=MAI ; FFF=I ; GGG=QUA .

Algoritmo 2 Escreva uma função que receba um número K (inteiro, não é preciso testar) e um vetor N de 10 inteiros e e devolva a posição de K em N. Se houver mais de 1 valor de K em N, deve retornar o primeiro. Se não houver K em N, retornar -1.

Por exemplo, Se K=8 e N=1 3 5 7 9 2 4 6 8 10, a resposta deve ser 9. Se K=7 e N é o mesmo, a resposta é 4. Se K=13 e N é o mesmo, a resposta é -1.


```
1: funcao B02ACHKK (K, N[10] : inteiro) : inteiro
2: inteiro RES,I
3: RES ← AAA
4: I ← BBB
5: enquanto CCC < 11 faça
6:    se N[I] = DDD então
7:       RES ← EEE
8:       I ← FFF
9:    fimse
10:   I ← I + 1
11: fimenquanto
12: devolva GGG
13: fimfuncao
```

Esta funcao correta, deu os seguintes resultados:

B02ACHK(17,14 28 27 21 11 18 12 13 15 17)=10 ; B02ACHK(14,16 17 14 23 22 28 27 25 12 30)=3 ; B02ACHK(17,21 40 26 16 38 28 15 20 17 19)=9 .

As substituições a fazer são:

AAA=-1 ; BBB=1 ; CCC=I ; DDD=K ; EEE=I ; FFF=11 (ou qualquer > 10) ; GGG=I .

Algoritmo 3 Escreva uma funcao que receba um vetor N de 10 elementos inteiros, em princípio sem nenhuma ordem (ou seja, desordenado) e uma chave K também inteira e devolva a quantidade de vezes que K aparece em N.

Por exemplo, se N=1 3 5 7 2 3 4 5 6 10 e K=5 a resposta é 2. Se K=18, a resposta é 0 e se K=1, a reposta é 1.

```
1: funcao B03CONK (K,N[10] : inteiro) : inteiro
2: inteiro QTD,I
3: QTD ← AAA
4: para  I de BBB a CCC passo DDD faça
5:    se N[I] = EEE então
6:       QTD ← FFF+ 1
7:    fimse
8: fimpara
9: devolva GGG
10: fimfuncao
```

Esta funcao correta, deu os seguintes resultados:

B03CONK(18,14 18 16 18 16 13 13 16 11 17)=2 ; B03CONK(17,18 12 11 13 17 15 17 11 17 11)=3 ; B03CONK(17,23 29 14 24 37 26 15 36 30 20)=0 .

As substituições a fazer são:

AAA=0 ; BBB=1 ; CCC=10 ; DDD=1 ; EEE=I ; FFF=QTD ; GGG=QTD .

Algoritmo 4 Escreva uma função que receba um vetor ordenado de 11 inteiros. A função deve devolver 3 valores (um real e dois inteiros), a saber:

média A soma dos elementos dividido por 11

moda o valor mais freqüente no vetor

mediana O valor do meio (neste caso o N[6])

```
1: funcao B04ESTA (N[11] : inteiro) : real,inteiro,inteiro
2: MED : real
3: I,J,MOD,MAN,SOM,ASA,CTDR[11] : inteiro
```

```
 4: I ← 1
 5: SOM ← AAA
 6: enquanto I < BBB faça
 7:    SOM ← CCC + N[I]
 8:    I ← I + 1
 9: fimenquanto
10: DDD ← SOM ÷ 11
11: EEE ← N[6]
12: para  I de 1 a 11 faça
13:    SOM ← 0
14:    para  J de 1 a 11 faça
15:       se N[I] = N[J] então
16:          SOM ← SOM + 1
17:       fimse
18:    fimpara
19:    CTDR[I] ← FFF
20: fimpara
21: MAI ← -9999
22: para  J de 1 a 11 faça
23:    se CTDR[J] > GGG então
24:       MAI ← CTDR[J]
25:       ASA ← HHH
26:    fimse
27: fimpara
28: III ← N[ASA]
29: devolva MED,MOD,MAN
30: fimfuncao
```

Esta funcao correta, deu os seguintes resultados:

B04ESTA(1 2 3 3 4 4 4 4 6 7 7) = 4.1 4 4 ; B04ESTA(1 1 1 2 6 6 6 7 7 7 8) = 4.7 1 6 ; B04ESTA(1 1 2 3 4 5 6 6 7 8 8) = 4.6 1 5 .

As substituições a fazer são:

AAA=1 ; BBB=12 ; CCC=SOM ; DDD=MED ; EEE=MAN ; FFF=SOM ; GGG=MAI ; HHH=J ; III=MOD .

O vetor está em ordem ? Escreva o algoritmo de uma função que receba um vetor de 10 inteiros e devolva 0 se o vetor está em ordem crescente e 1 senão. Um vetor V está em ordem crescente para quaisquer i e j, sendo que $j > i$, é válida a expressão $V[j] \geq V[i]$. Note a presença da igualdade na condição. Além disso, essa expressão tem que ser válida para todos os pares dentro do vetor.

Por exemplo se o vetor for 1,3,4,5,6,7,9,10,16,21 a resposta é 0. Se o vetor for 1,3,4,5,6,8,7,10,16,21 a resposta é 1 e se for 2,2,2,2,2,2,2,2,2,2 a resposta é 0.

```
 1: funcao B05ESORN (N[10] : inteiro) : inteiro27
 2: inteiro I, RES
 3: RES ← AAA
 4: I ← 1
 5: enquanto I < BBB faça
 6:    se N[I] >N[I+1] então
 7:       RES ← 1
 8:    fimse
 9:    I ← I + 1
10: fimenquanto
11: devolva CCC
```


12: fimfuncao

Esta funcao correta, deu os seguintes resultados:

B05ESOR(7 8 36 47 48 50 65 70 89 90) = 0 ; B05ESOR(32 38 42 53 59 66 65 74 83 89) = 1 ; B05ESOR(12 13 41 47 52 66 21 87 90 97) = 1 .

As substituições a fazer são:

AAA=0 ; BBB=10 ; CCC=RES .

☞ **Responda**

R=

37.3 Exercício 3

Construindo e consertando algoritmos - parte 2

Nesta folha, você receberá 5 algoritmos, que têm algumas lacunas representadas pelas constantes AAA, BBB, CCC, ... até III. Abaixo de cada algoritmo, é listada uma legenda que permitirá a substituição das constantes AAA,...,III por valores determinados.

Após a substituição, de duas uma:

- O algoritmo está correto
- O algoritmo contém um erro. Este erro deriva-se de uma (e apenas uma) substituição de constante por um valor errado.

Você não precisa determinar onde está o erro (se bem que isso é um interessante desafio), mas precisa apenas determinar que o algoritmo está errado.

Uma boa maneira de verificar isto é acompanhar as 3 execuções do algoritmo (correto) que acompanham o enunciado. Trata-se de algo similar a um chines reverso.

Identifique qual(is) dos 5 algoritmos estão corretos definindo A_n como: $A_1 \leftarrow 1$ se o algoritmo 1 está correto e $A_1 \leftarrow 0$ senão. Idem para A_2, A_3, A_4 e A_5.

Responda o resultado R

$$R = (A_1 \times 1) + (A_2 \times 2) + (A_3 \times 4) + (A_4 \times 8) + (A_5 \times 16)$$

Algoritmo 1 Escreva uma função que receba um vetor de 10 números inteiros e devolva a posição daquele número que é o maior do vetor. Se houver dois ou mais iguais a este maior, deve-se retornar o endereço do primeiro. Por exemplo, se o vetor for 4, 7, 2, 19, 21, 8, 7, 19, 8, 6 a resposta deve ser 5, já que o 21 (o maior) é o quinto elemento. Se o vetor for 4, 12, 5, 7, 12, 7, 1, 2, 3, 8, a resposta deve ser 2.

```
1: funcao B01ACHM (N[10] : inteiro) : inteiro
2: MAI,QUA,I : inteiro
3: I ← AAA
4: BBB ← -99999
5: enquanto CCC < 11 faça
6:    se N[I] > DDD então
7:       MAI ← N[I]
8:       EEE ← I
9:    fimse
10:   FFF ← I + 1
11: fimenquanto
12: devolva GGG
13: fimfuncao
```

Esta funcao correta, deu os seguintes resultados:

B01ACHM(4 1 4 3 2 5 4 5 2 5) = 6 ; B01ACHM(9 8 18 16 1 13 3 5 14 11) = 3 ; B01ACHM(2 5 5 4 5 2 3 1 5 2) = 2 .

As substituições a fazer são:

AAA=1 ; BBB=MAI ; CCC=N[I] ; DDD=MAI ; EEE=QUA ; FFF=I ; GGG=QUA .

Algoritmo 2 Escreva uma função que receba um número K (inteiro, não é preciso testar) e um vetor N de 10 inteiros e e devolva a posição de K em N. Se houver mais de 1 valor de K em N, deve retornar o primeiro. Se não houver K em N, retornar -1.

Por exemplo, Se K=8 e N=1 3 5 7 9 2 4 6 8 10, a resposta deve ser 9. Se K=7 e N é o mesmo, a resposta é 4. Se K=13 e N é o mesmo, a resposta é -1.

```
1: funcao B02ACHKK (K, N[10] : inteiro) : inteiro
2: inteiro RES,I
3: RES ← AAA
4: I ← BBB
5: enquanto CCC < 11 faça
6:    se N[I] = DDD então
7:       RES ← EEE
8:       I ← FFF
9:    fimse
10:   I ← I + 1
11: fimenquanto
12: devolva GGG
13: fimfuncao
```


Esta funcao correta, deu os seguintes resultados:

B02ACHK(20,29 21 12 23 26 20 16 13 28 24)=6 ; B02ACHK(29,28 14 30 25 17 20 16 18 13 29)=10 ; B02ACHK(17,26 24 27 34 23 11 37 25 39 31)=-1 .

As substituições a fazer são:

AAA=-1 ; BBB=1 ; CCC=I ; DDD=K ; EEE=I ; FFF=11 (ou qualquer > 10) ; GGG=RES .

Algoritmo 3 Escreva uma funcao que receba um vetor N de 10 elementos inteiros, em princípio sem nenhuma ordem (ou seja, desordenado) e uma chave K também inteira e devolva a quantidade de vezes que K aparece em N.

Por exemplo, se N=1 3 5 7 2 3 4 5 6 10 e K=5 a resposta é 2. Se K=18, a resposta é 0 e se K=1, a reposta é 1.

```
1: funcao B03CONK (K,N[10] : inteiro) : inteiro
2: inteiro QTD,I
3: QTD ← AAA
4: para I de BBB a CCC passo DDD faça
5:    se N[I] = EEE então
6:       QTD ← FFF+ 1
7:    fimse
8: fimpara
9: devolva GGG
10: fimfuncao
```

Esta funcao correta, deu os seguintes resultados:

B03CONK(14,13 13 11 17 12 11 14 15 15 14)=2 ; B03CONK(18,18 15 17 14 14 17 18 14 12 18)=3 ; B03CONK(17,18 31 14 23 40 13 17 24 28 16)=1 .

As substituições a fazer são:

AAA=0 ; BBB=1 ; CCC=10 ; DDD=1 ; EEE=K ; FFF=QTD ; GGG=I .

Algoritmo 4 Escreva uma função que receba um vetor ordenado de 11 inteiros. A função deve devolver 3 valores (um real e dois inteiros), a saber:

média A soma dos elementos dividido por 11

moda o valor mais freqüente no vetor

mediana O valor do meio (neste caso o N[6])

```
1: funcao B04ESTA (N[11] : inteiro) : real,inteiro,inteiro
2: MED : real
3: I,J,MOD,MAN,SOM,ASA,CTDR[11] : inteiro
4: I ← 1
5: SOM ← AAA
6: enquanto I < BBB faça
7:    SOM ← CCC + N[I]
8:    I ← I + 1
9: fimenquanto
10: DDD ← SOM ÷ 11
11: EEE ← N[6]
12: para I de 1 a 11 faça
13:    SOM ← 0
14:    para J de 1 a 11 faça
15:       se N[I] = N[J] então
16:          SOM ← SOM + 1
```

```
17:       fimse
18:    fimpara
19:    CTDR[I] ← FFF
20: fimpara
21: MAI ← -9999
22: para  J de 1 a 11 faça
23:    se CTDR[J] > GGG então
24:       MAI ← CTDR[J]
25:       ASA ← HHH
26:    fimse
27: fimpara
28: III ← N[ASA]
29: devolva MED,MOD,MAN
30: fimfuncao
```

Esta funcao correta, deu os seguintes resultados:

B04ESTA(1 1 2 3 3 4 4 5 6 7 8) = 4.0 1 4 ; B04ESTA(2 3 4 5 5 5 6 6 8 8 8) = 5.5 5 5 ; B04ESTA(1 2 3 3 4 4 4 4 7 7 7) = 4.2 4 4 .

As substituições a fazer são:

AAA=1 ; BBB=12 ; CCC=SOM ; DDD=MED ; EEE=MAN ; FFF=SOM ; GGG=MAI ; HHH=J ; III=MOD .

O vetor está em ordem ? Escreva o algoritmo de uma função que receba um vetor de 10 inteiros e devolva 0 se o vetor está em ordem crescente e 1 senão. Um vetor V está em ordem crescente para quaisquer i e j, sendo que $j > i$, é válida a expressão $V[j] \geq V[i]$. Note a presença da igualdade na condição. Além disso, essa expressão tem que ser válida para todos os pares dentro do vetor.

Por exemplo se o vetor for 1,3,4,5,6,7,9,10,16,21 a resposta é 0. Se o vetor for 1,3,4,5,6,8,7,10,16,21 a resposta é 1 e se for 2,2,2,2,2,2,2,2,2,2 a resposta é 0.

```
1: funcao B05ESORN (N[10] : inteiro) : inteiro27
2: inteiro I, RES
3: RES ← AAA
4: I ← 1
5: enquanto I < BBB faça
6:    se N[I] >N[I+1] então
7:       RES ← 1
8:    fimse
9:    I ← I + 1
10: fimenquanto
11: devolva CCC
12: fimfuncao
```

Esta funcao correta, deu os seguintes resultados:

B05ESOR(11 14 26 43 50 56 63 66 73 96) = 0 ; B05ESOR(3 5 6 11 12 23 72 58 63 75) = 1 ; B05ESOR(23 26 44 46 49 50 20 61 69 70) = 1 .

As substituições a fazer são:

AAA=0 ; BBB=10 ; CCC=RES .

☞ **Responda**

R=

37.4 Exercício 4

Construindo e consertando algoritmos - parte 2

Nesta folha, você receberá 5 algoritmos, que têm algumas lacunas representadas pelas constantes AAA, BBB, CCC, ... até III. Abaixo de cada algoritmo, é listada uma legenda que permitirá a substituição das constantes AAA,...,III por valores determinados.

Após a substituição, de duas uma:

- O algoritmo está correto
- O algoritmo contém um erro. Este erro deriva-se de uma (e apenas uma) substituição de constante por um valor errado.

Você não precisa determinar onde está o erro (se bem que isso é um interessante desafio), mas precisa apenas determinar que o algoritmo está errado.

Uma boa maneira de verificar isto é acompanhar as 3 execuções do algoritmo (correto) que acompanham o enunciado. Trata-se de algo similar a um chines reverso.

Identifique qual(is) dos 5 algoritmos estão corretos definindo A_n como: $A_1 \leftarrow 1$ se o algoritmo 1 está correto e $A_1 \leftarrow 0$ senão. Idem para A_2, A_3, A_4 e A_5.

Responda o resultado R

$$R = (A_1 \times 1) + (A_2 \times 2) + (A_3 \times 4) + (A_4 \times 8) + (A_5 \times 16)$$

Algoritmo 1 Escreva uma função que receba um vetor de 10 números inteiros e devolva a posição daquele número que é o maior do vetor. Se houver dois ou mais iguais a este maior, deve-se retornar o endereço do primeiro. Por exemplo, se o vetor for 4, 7, 2, 19, 21, 8, 7, 19, 8, 6 a resposta deve ser 5, já que o 21 (o maior) é o quinto elemento. Se o vetor for 4, 12, 5, 7, 12, 7, 1, 2, 3, 8, a resposta deve ser 2.

```
1: funcao B01ACHM (N[10] : inteiro) : inteiro
2: MAI,QUA,I : inteiro
3: I ← AAA
4: BBB ← -99999
5: enquanto CCC < 11 faça
6:    se N[I] > DDD então
7:       MAI ← N[I]
8:       EEE ← I
9:    fimse
10:   FFF ← I + 1
11: fimenquanto
12: devolva GGG
13: fimfuncao
```

Esta funcao correta, deu os seguintes resultados:

B01ACHM(4 4 3 2 1 2 6 3 1 4) = 7 ; B01ACHM(11 10 13 12 19 7 5 16 6 17) = 5 ; B01ACHM(5 3 3 1 4 1 1 2 2 1) = 1 .

As substituições a fazer são:

AAA=1 ; BBB=MAI ; CCC=I ; DDD=MAI ; EEE=QUA ; FFF=I ; GGG=QUA .

Algoritmo 2 Escreva uma função que receba um número K (inteiro, não é preciso testar) e um vetor N de 10 inteiros e e devolva a posição de K em N. Se houver mais de 1 valor de K em N, deve retornar o primeiro. Se não houver K em N, retornar -1.

Por exemplo, Se K=8 e N=1 3 5 7 9 2 4 6 8 10, a resposta deve ser 9. Se K=7 e N é o mesmo, a resposta é 4. Se K=13 e N é o mesmo, a resposta é -1.

```
 1: funcao B02ACHKK (K, N[10] : inteiro) : inteiro
 2: inteiro RES,I
 3: RES ← AAA
 4: I ← BBB
 5: enquanto CCC < 11 faça
 6:    se N[I] = DDD então
 7:       RES ← EEE
 8:       I ← FFF
 9:    fimse
10:    I ← I + 1
11: fimenquanto
12: devolva GGG
13: fimfuncao
```

Esta funcao correta, deu os seguintes resultados:

B02ACHK(27,13 30 25 23 18 27 24 16 20 11)=6 ; B02ACHK(11,25 15 23 16 17 11 20 19 13 27)=6 ; B02ACHK(17,25 38 30 12 34 15 33 14 21 32)=-1 .

As substituições a fazer são:

AAA=-1 ; BBB=1 ; CCC=I ; DDD=K ; EEE=I ; FFF=11 (ou qualquer > 10) ; GGG=RES .

Algoritmo 3 Escreva uma funcao que receba um vetor N de 10 elementos inteiros, em princípio sem nenhuma ordem (ou seja, desordenado) e uma chave K também inteira e devolva a quantidade de vezes que K aparece em N.

Por exemplo, se N=1 3 5 7 2 3 4 5 6 10 e K=5 a resposta é 2. Se K=18, a resposta é 0 e se K=1, a reposta é 1.

```
 1: funcao B03CONK (K,N[10] : inteiro) : inteiro
 2: inteiro QTD,I
 3: QTD ← AAA
 4: para  I de BBB a CCC passo DDD faça
 5:    se N[I] = EEE então
 6:       QTD ← FFF+ 1
 7:    fimse
 8: fimpara
 9: devolva GGG
10: fimfuncao
```

Esta funcao correta, deu os seguintes resultados:

B03CONK(15,12 18 14 13 15 11 11 13 15 11)=2 ; B03CONK(12,18 17 14 16 12 13 12 17 14 12)=3 ; B03CONK(17,29 11 21 23 30 39 31 35 16 14)=0 .

As substituições a fazer são:

AAA=0 ; BBB=0 ; CCC=10 ; DDD=1 ; EEE=K ; FFF=QTD ; GGG=QTD .

Algoritmo 4 Escreva uma função que receba um vetor ordenado de 11 inteiros. A função deve devolver 3 valores (um real e dois inteiros), a saber:

média A soma dos elementos dividido por 11

moda o valor mais freqüente no vetor

mediana O valor do meio (neste caso o N[6])

```
1: funcao B04ESTA (N[11] : inteiro) : real,inteiro,inteiro
2: MED : real
3: I,J,MOD,MAN,SOM,ASA,CTDR[11] : inteiro
```



```
 4: I ← 1
 5: SOM ← AAA
 6: enquanto I < BBB faça
 7:    SOM ← CCC + N[I]
 8:    I ← I + 1
 9: fimenquanto
10: DDD ← SOM ÷ 11
11: EEE ← N[6]
12: para  I de 1 a 11 faça
13:    SOM ← 0
14:    para  J de 1 a 11 faça
15:       se N[I] = N[J] então
16:          SOM ← SOM + 1
17:       fimse
18:    fimpara
19:    CTDR[I] ← FFF
20: fimpara
21: MAI ← -9999
22: para  J de 1 a 11 faça
23:    se CTDR[J] > GGG então
24:       MAI ← CTDR[J]
25:       ASA ← HHH
26:    fimse
27: fimpara
28: III ← N[ASA]
29: devolva MED,MOD,MAN
30: fimfuncao
```

Esta funcao correta, deu os seguintes resultados:

B04ESTA(1 2 2 3 3 4 4 5 6 7 7) = 4.0 2 4 ; B04ESTA(2 2 2 3 3 4 5 6 6 8 8) = 4.5 2 4 ; B04ESTA(1 2 2 3 4 4 4 4 4 5 6) = 3.5 4 4 .

As substituições a fazer são:

AAA=0 ; BBB=12 ; CCC=SOM ; DDD=MED ; EEE=MAN ; FFF=SOM ; GGG=MAI ; HHH=J ; III=MOD .

O vetor está em ordem ? Escreva o algoritmo de uma função que receba um vetor de 10 inteiros e devolva 0 se o vetor está em ordem crescente e 1 senão. Um vetor V está em ordem crescente para quaisquer i e j, sendo que $j > i$, é válida a expressão $V[j] \geq V[i]$. Note a presença da igualdade na condição. Além disso, essa expressão tem que ser válida para todos os pares dentro do vetor.

Por exemplo se o vetor for 1,3,4,5,6,7,9,10,16,21 a resposta é 0. Se o vetor for 1,3,4,5,6,8,7,10,16,21 a resposta é 1 e se for 2,2,2,2,2,2,2,2,2,2 a resposta é 0.

```
 1: funcao B05ESORN (N[10] : inteiro) : inteiro27
 2: inteiro I, RES
 3: RES ← AAA
 4: I ← 1
 5: enquanto I < BBB faça
 6:    se N[I] >N[I+1] então
 7:       RES ← 1
 8:    fimse
 9:    I ← I + 1
10: fimenquanto
11: devolva CCC
```

12: fimfuncao

Esta funcao correta, deu os seguintes resultados:

B05ESOR(14 34 41 47 48 51 54 57 62 96) = 0 ; B05ESOR(28 39 41 42 49 51 34 78 81 89) = 1 ; B05ESOR(5 21 22 23 28 36 38 57 84 96) = 1 .

As substituições a fazer são:

AAA=1 ; BBB=10 ; CCC=RES .

☞ **Responda**

R=

37.5 Respostas

```
1    0 0 0 0 3   15
2    5 7 5 1 0   16
3    3 0 7 1 0   18
4    0 0 2 0 1   11
```


Capítulo 38

Exercícios Práticos: 125c - Correção de algoritmos

38.1 Exercício 1

Construindo e consertando algoritmos - parte 3

Nesta folha, você receberá 5 algoritmos, que têm algumas lacunas representadas pelas constantes AAA, BBB, CCC, ... até III. Abaixo de cada algoritmo, é listada uma legenda que permitirá a substituição das constantes AAA,...,III por valores determinados.

Após a substituição, de duas uma:

- O algoritmo está correto
- O algoritmo contém um erro. Este erro deriva-se de uma (e apenas uma) substituição de constante por um valor errado.

Você não precisa determinar onde está o erro (se bem que isso é um interessante desafio), mas precisa apenas determinar que o algoritmo está errado.

Uma boa maneira de verificar isto é acompanhar as 3 execuções do algoritmo (correto) que acompanham o enunciado. Trata-se de algo similar a um chines reverso.

Identifique qual(is) dos 5 algoritmos estão corretos definindo A_n como: $A_1 \leftarrow 1$ se o algoritmo 1 está correto e $A_1 \leftarrow 0$ senão. Idem para A_2, A_3, A_4 e A_5.

Responda o resultado R

$$R = (A_1 \times 1) + (A_2 \times 2) + (A_3 \times 4) + (A_4 \times 8) + (A_5 \times 16)$$

Algoritmo 1 Escreva uma função que receba uma frase (cadeia de até 80 caracteres, terminada por um ponto) e devolva 1 se a frase for um palíndromo e 0 senão. Um palíndromo é aquela frase que pode ser lida tanto da direita para a esquerda como ao contrário. Exemplos:
ROMA ME TEM AMOR
SOCORRAM ME SUBI NO ONIBUS EM MARROCOS
SAIRAM O TIO E OITO MARIAS
OVO
Observação: note que as posições do espaço em branco não são as mesmas na ida e na volta.

```
1: funcao C01PALI (X[80] : caracter) : inteiro
2: inteiro INI,FIM,EPAL,TAM
```

```
 3: TAM ← 1
 4: enquanto X[TAM] AAA '.' faça
 5:    TAM ← TAM + 1
 6: fimenquanto
 7: INI ← 1
 8: FIM ← BBB
 9: EPAL ← 1
10: enquanto INI < CCC faça
11:    enquanto X[DDD] = ' ' faça
12:       INI ← INI + 1
13:    fimenquanto
14:    enquanto X[EEE] = ' ' faça
15:       FIM ← FIM - 1
16:    fimenquanto
17:    se X[INI] FFF X[FIM] então
18:       EPAL ← 0
19:    fimse
20:    INI ← GGG
21:    FIM ← HHH
22: fimenquanto
23: devolva III
24: fimfuncao
```

Esta funcao correta, deu os seguintes resultados:

C01PALI(ABRACADABRA.) = 0 ; C01PALI(OTO COME MOCOTO.) = 1 ; C01PALI(ROMA AMOR.) = 1 .

As substituições a fazer são:

AAA=um sinal de $\neq$; BBB=TAM-1 ; CCC=FIM ; DDD=INI ; EEE=FIM ; FFF=um sinal de $\neq$; GGG=INI+1 ; HHH=FIM-1 ; III=EPAL .

Algoritmo 2 Escreva o algoritmo de uma função que receba uma chave K (Inteiro) e um vetor V (de 10 inteiros) crescente e devolva outro vetor de 11 inteiros, também crescente onde a chave K foi incluída em seu local correto.

```
 1: funcao C02INCL (K, V[10] : inteiro) : R[11] inteiro
 2: inteiro I,J
 3: para I de 1 a AAA faça
 4:    R[I] ← V[I]
 5: fimpara
 6: I ← BBB
 7: enquanto (R[I] ≤ CCC)∧(I<DDD) faça
 8:    I ← EEE
 9: fimenquanto
10: J ← FFF
11: enquanto J ≥ I faça
12:    R[J+1] ← R[J]
13:    J ← GGG
14: fimenquanto
15: R[J+1] ← HHH
16: devolva III
17: fimfuncao
```

Esta funcao correta, deu os seguintes resultados:

C02INCL(67,6 12 18 24 30 36 42 48 54 60)=6 12 18 24 30 36 42 48 54 60 67 ; C02INCL(154,7 14 21 28 35 42 49 56 63 70)=7 14 21 28 35 42 49 56 63 70 154 ; C02INCL(2,16 22 28 34 40 46 52 58 64 70)=2 16 22 28 34 40 46 52 58 64 70 .

As substituições a fazer são:

AAA=11 ; BBB=1 ; CCC=K ; DDD=11 ; EEE=I+1 ; FFF=10 ; GGG=J-1 ; HHH=K ; III=R .

Algoritmo 3 Escreva o algoritmo de uma função que receba uma frase (cadeia de 80 caracteres, o último útil é um ponto) e devolva o número da palavra que é a maior da frase. Se houver empate (mais de uma palavra com o mesmo comprimento), devolver o número da primeira. Por exemplo, na frase "IVO VIU A UVA", a maior palavra tem 3 letras é como há empate a resposta deve ser 1. Na frase "OUVIRAM DO IPIRANGA AS MARGENS"a resposta deve ser 3.

```
 1: funcao C03PALM (X[80] : caracter) : inteiro
 2: TMS[40] : inteiro
 3: I,J,TAM,CTP,MAI,QUA : inteiro
 4: TAM ← AAA
 5: enquanto X[TAM] ≠ '.' faça
 6:    TAM ← BBB
 7: fimenquanto
 8: CTP ← 1
 9: I ← 1
10: J ← 0
11: enquanto I ≤ TAM faça
12:    se X[I] CCC ' ' então
13:       TMS[CTP] ← I - J
14:       TMS[CTP] ← TMS[CTP]-1
15:       CTP ← DDD
16:       J ← EEE
17:    fimse
18:    I ← I + 1
19: fimenquanto
20: TMS[CTP] ← I - J
21: TMS[CTP] ← TMS[CTP] - FFF
22: MAI ← -99999
23: para  I de 1 a GGG faça
24:    se TMS[I] > MAI então
25:       MAI ← TMS[I]
26:       QUA ← HHH
27:    fimse
28: fimpara
29: devolva MAI
30: fimfuncao
```

Esta funcao correta, deu os seguintes resultados:

C03PALM(ATE O PONTO DE FAZER INFERENCIA.)=6 ; C03PALM(DECORRIDO ENTRE O DESAPARECIMENTO.)=4 ; C03PALM(EMBORA MENOS PERVERSO QUE.)=3 .

As substituições a fazer são:

AAA=1 ; BBB=TAM+1 ; CCC=≠ ; DDD=CTP+1 ; EEE=I ; FFF=2 ; GGG=CTP ; HHH=I ; III=QUA .

Algoritmo 4 Escreva o algoritmo de uma função que receba uma chave K (inteiro) e um vetor V de 10 inteiros. A função deve excluir de V todas as ocorrências (se houver) de K, incluindo zeros ao final se necessário. Por exemplo, se chamada com 22 e 1 7 21 22 23 24 28 29 22 90 deve responder 1 7 21 23 24 28 29 90 0 0.

```
 1: funcao C04EXCL (K, V[10] : inteiro) : R[10] inteiro
 2: I, J : inteiro
 3: I ← AAA
 4: J ← BBB
 5: enquanto I<CCC faça
 6:    se V[I] ≠ K então
 7:       R[J] ← V[I]
 8:       J ← DDD
 9:    fimse
10:    I ← EEE
11: fimenquanto
12: enquanto J < FFF faça
13:    R[J] ← 0
14:    J ← GGG
15: fimenquanto
16: devolva R
17: fimfuncao
```

Esta funcao correta, deu os seguintes resultados:

C04EXCL(1,3 7 5 3 8 3 8 1 6 2) = 3 7 5 3 8 3 8 6 2 0 ; C04EXCL(91,44 9 23 50 41 21 40 3 48 17) = 44 9 23 50 41 21 40 3 48 17 ; C04EXCL(4,3 5 5 8 8 1 6 1 10 6) = 3 5 5 8 8 1 6 1 10 6 .

As substituições a fazer são:

AAA=1 ; BBB=1 ; CCC=11 ; DDD=J+1 ; EEE=I+1 ; FFF=11 ; GGG=I+1 .

Algoritmo 5

```
 1: funcao C05AMPL (X[10] : inteiro) : inteiro
 2: inteiro I,MAI,MEN,AMP
 3: MAI ← AAA
 4: MEN ← BBB
 5: para I de 1 a 10 faça
 6:    se X[I] > CCC então
 7:       MAI ← X[I]
 8:    fimse
 9:    se X[I] < DDD então
10:       MEN ← X[I]
11:    fimse
12: fimpara
13: AMP ← MAI - MEN
14: RRRR ← AMP
15: fimfuncao
16:
```

Esta funcao correta, deu os seguintes resultados:

C05AMPL(2 45 21 97 52 51 54 44 99 14) = 97 ; C05AMPL(4 6 5 6 5 1 7 2 2 2) = 6 ; C05AMPL(35 64 39 91 51 36 68 55 79 62) = 56 .

As substituições a fazer são:

AAA=-9999 ; BBB=-9999 ; CCC=MAI ; DDD=MEN .

☞ **Responda**

R=

38.2 Exercício 2

Construindo e consertando algoritmos - parte 3

Nesta folha, você receberá 5 algoritmos, que têm algumas lacunas representadas pelas constantes AAA, BBB, CCC, ... até III. Abaixo de cada algoritmo, é listada uma legenda que permitirá a substituição das constantes AAA,...,III por valores determinados.

Após a substituição, de duas uma:

- O algoritmo está correto
- O algoritmo contém um erro. Este erro deriva-se de uma (e apenas uma) substituição de constante por um valor errado.

Você não precisa determinar onde está o erro (se bem que isso é um interessante desafio), mas precisa apenas determinar que o algoritmo está errado.

Uma boa maneira de verificar isto é acompanhar as 3 execuções do algoritmo (correto) que acompanham o enunciado. Trata-se de algo similar a um chines reverso.

Identifique qual(is) dos 5 algoritmos estão corretos definindo A_n como: $A_1 \leftarrow 1$ se o algoritmo 1 está correto e $A_1 \leftarrow 0$ senão. Idem para A_2, A_3, A_4 e A_5.

Responda o resultado R

$$R = (A_1 \times 1) + (A_2 \times 2) + (A_3 \times 4) + (A_4 \times 8) + (A_5 \times 16)$$

Algoritmo 1 Escreva uma função que receba uma frase (cadeia de até 80 caracteres, terminada por um ponto) e devolva 1 se a frase for um palíndromo e 0 senão. Um palíndromo é aquela frase que pode ser lida tanto da direita para a esquerda como ao

contrário. Exemplos:
ROMA ME TEM AMOR
SOCORRAM ME SUBI NO ONIBUS EM MARROCOS
SAIRAM O TIO E OITO MARIAS
OVO
Observação: note que as posições do espaço em branco não são as mesmas na ida e na volta.

```
1: funcao C01PALI (X[80] : caracter) : inteiro
2: inteiro INI,FIM,EPAL,TAM
3: TAM ← 1
4: enquanto X[TAM] AAA '.' faça
5:    TAM ← TAM + 1
6: fimenquanto
7: INI ← 1
8: FIM ← BBB
9: EPAL ← 1
10: enquanto INI < CCC faça
11:    enquanto X[DDD] = ' ' faça
12:       INI ← INI + 1
13:    fimenquanto
14:    enquanto X[EEE] = ' ' faça
15:       FIM ← FIM - 1
16:    fimenquanto
17:    se X[INI] FFF X[FIM] então
18:       EPAL ← 0
19:    fimse
20:    INI ← GGG
21:    FIM ← HHH
22: fimenquanto
23: devolva III
24: fimfuncao
```

Esta funcao correta, deu os seguintes resultados:

C01PALI(BRASIL E LISARB.) = 1 ; C01PALI(OTO COME MOCOTO.) = 1 ; C01PALI(XUCRICUNICO.) = 0 .

As substituições a fazer são:

AAA=um sinal de $\neq$; BBB=TAM-1 ; CCC=FIM ; DDD=FIM ; EEE=FIM ; FFF=um sinal de $\neq$; GGG=INI+1 ; HHH=FIM-1 ; III=EPAL .

Algoritmo 2 Escreva o algoritmo de uma função que receba uma chave K (Inteiro) e um vetor V (de 10 inteiros) crescente e devolva outro vetor de 11 inteiros, também crescente onde a chave K foi incluída em seu local correto.

```
1: funcao C02INCL (K, V[10] : inteiro) : R[11] inteiro
2: inteiro I,J
3: para I de 1 a AAA faça
4:    R[I] ← V[I]
5: fimpara
6: I ← BBB
7: enquanto (R[I] ≤ CCC)∧(I<DDD) faça
8:    I ← EEE
9: fimenquanto
10: J ← FFF
```

```
11: enquanto J ≥ I faça
12:    R[J+1] ← R[J]
13:    J ← GGG
14: fimenquanto
15: R[J+1] ← HHH
16: devolva III
17: fimfuncao
```

Esta funcao correta, deu os seguintes resultados:

C02INCL(41,6 12 18 24 30 36 42 48 54 60)=6 12 18 24 30 36 41 42 48 54 60 ; C02INCL(174,8 16 24 32 40 48 56 64 72 80)=8 16 24 32 40 48 56 64 72 80 174 ; C02INCL(5,15 20 25 30 35 40 45 50 55 60)=5 15 20 25 30 35 40 45 50 55 60 .

As substituições a fazer são:

AAA=10 ; BBB=1 ; CCC=K ; DDD=11 ; EEE=I+1 ; FFF=10 ; GGG=J+1 ; HHH=K ; III=R .

Algoritmo 3 Escreva o algoritmo de uma função que receba uma frase (cadeia de 80 caracteres, o último útil é um ponto) e devolva o número da palavra que é a maior da frase. Se houver empate (mais de uma palavra com o mesmo comprimento), devolver o número da primeira. Por exemplo, na frase "IVO VIU A UVA", a maior palavra tem 3 letras é como há empate a resposta deve ser 1. Na frase "OUVIRAM DO IPIRANGA AS MARGENS"a resposta deve ser 3.

```
1: funcao C03PALM (X[80] : caracter) : inteiro
2: TMS[40] : inteiro
3: I,J,TAM,CTP,MAI,QUA : inteiro
4: TAM ← AAA
5: enquanto X[TAM] ≠ '.' faça
6:    TAM ← BBB
7: fimenquanto
8: CTP ← 1
9: I ← 1
10: J ← 0
11: enquanto I ≤ TAM faça
12:    se X[I] CCC ' ' então
13:       TMS[CTP] ← I - J
14:       TMS[CTP] ← TMS[CTP]-1
15:       CTP ← DDD
16:       J ← EEE
17:    fimse
18:    I ← I + 1
19: fimenquanto
20: TMS[CTP] ← I - J
21: TMS[CTP] ← TMS[CTP] - FFF
22: MAI ← -99999
23: para  I de 1 a GGG faça
24:    se TMS[I] > MAI então
25:       MAI ← TMS[I]
26:       QUA ← HHH
27:    fimse
28: fimpara
29: devolva MAI
30: fimfuncao
```


Esta funcao correta, deu os seguintes resultados:

C03PALM(DECORRIDO ENTRE O DESAPARECIMENTO.)=4 ; C03PALM(REDATOR E MOSTRAR TENDO.)=1 ; C03PALM(MAS SE NAO PODEMOS IMAGINAR.)=5 .

As substituições a fazer são:

AAA=1 ; BBB=TAM+1 ; CCC=sinal de igual (=) ; DDD=CTP+1 ; EEE=I ; FFF=2 ; GGG=CTP ; HHH=I ; III=QUA .

Algoritmo 4 Escreva o algoritmo de uma função que receba uma chave K (inteiro) e um vetor V de 10 inteiros. A função deve excluir de V todas as ocorrências (se houver) de K, incluindo zeros ao final se necessário. Por exemplo, se chamada com 22 e 1 7 21 22 23 24 28 29 22 90 deve responder 1 7 21 23 24 28 29 90 0 0.

```
 1: funcao C04EXCL (K, V[10] : inteiro) : R[10] inteiro
 2: I, J : inteiro
 3: I ← AAA
 4: J ← BBB
 5: enquanto I<CCC faça
 6:    se V[I] ≠ K então
 7:       R[J] ← V[I]
 8:       J ← DDD
 9:    fimse
10:    I ← EEE
11: fimenquanto
12: enquanto J < FFF faça
13:    R[J] ← 0
14:    J ← GGG
15: fimenquanto
16: devolva R
17: fimfuncao
```

Esta funcao correta, deu os seguintes resultados:

C04EXCL(2,5 1 8 3 5 8 6 8 2 5) = 5 1 8 3 5 8 6 8 5 0 ; C04EXCL(82,31 47 13 45 18 40 2 12 7 17) = 31 47 13 45 18 40 2 12 7 17 ; C04EXCL(6,2 2 3 9 9 7 9 3 8 5) = 2 2 3 9 9 7 9 3 8 5 .

As substituições a fazer são:

AAA=1 ; BBB=1 ; CCC=11 ; DDD=J+1 ; EEE=I+1 ; FFF=11 ; GGG=J+1 .

Algoritmo 5

```
 1: funcao C05AMPL (X[10] : inteiro) : inteiro
 2: inteiro I,MAI,MEN,AMP
 3: MAI ← AAA
 4: MEN ← BBB
 5: para I de 1 a 10 faça
 6:    se X[I] > CCC então
 7:       MAI ← X[I]
 8:    fimse
 9:    se X[I] < DDD então
10:       MEN ← X[I]
11:    fimse
12: fimpara
13: AMP ← MAI - MEN
14: RRRR ← AMP
```

```
15: fimfuncao
16:
```

Esta funcao correta, deu os seguintes resultados:
C05AMPL(11 65 45 96 60 59 88 89 80 35) = 85 ; C05AMPL(3 4 7 3 2 2 5 8 7 1) = 7 ; C05AMPL(81 32 36 13 55 83 6 27 23 30) = 77 .
As substituições a fazer são:
AAA=-9999 ; BBB=+9999 ; CCC=MAI ; DDD=MEN .

☞ **Responda**

R=

38.3 Exercício 3

Construindo e consertando algoritmos - parte 3

Nesta folha, você receberá 5 algoritmos, que têm algumas lacunas representadas pelas constantes AAA, BBB, CCC, ... até III. Abaixo de cada algoritmo, é listada uma legenda que permitirá a substituição das constantes AAA,...,III por valores determinados.

Após a substituição, de duas uma:

- O algoritmo está correto
- O algoritmo contém um erro. Este erro deriva-se de uma (e apenas uma) substituição de constante por um valor errado.

Você não precisa determinar onde está o erro (se bem que isso é um interessante desafio), mas precisa apenas determinar que o algoritmo está errado.

Uma boa maneira de verificar isto é acompanhar as 3 execuções do algoritmo (correto) que acompanham o enunciado. Trata-se de algo similar a um chines reverso.

Identifique qual(is) dos 5 algoritmos estão corretos definindo A_n como: $A_1 \leftarrow 1$ se o algoritmo 1 está correto e $A_1 \leftarrow 0$ senão. Idem para A_2, A_3, A_4 e A_5.

Responda o resultado R

$$R = (A_1 \times 1) + (A_2 \times 2) + (A_3 \times 4) + (A_4 \times 8) + (A_5 \times 16)$$

Algoritmo 1 Escreva uma função que receba uma frase (cadeia de até 80 caracteres, terminada por um ponto) e devolva 1 se a frase for um palíndromo e 0 senão. Um palíndromo é aquela frase que pode ser lida tanto da direita para a esquerda como ao contrário. Exemplos:
ROMA ME TEM AMOR
SOCORRAM ME SUBI NO ONIBUS EM MARROCOS
SAIRAM O TIO E OITO MARIAS
OVO
Observação: note que as posições do espaço em branco não são as mesmas na ida e na volta.

```
 1: funcao C01PALI (X[80] : caracter) : inteiro
 2: inteiro INI,FIM,EPAL,TAM
 3: TAM ← 1
 4: enquanto X[TAM] AAA '.' faça
 5:    TAM ← TAM + 1
 6: fimenquanto
 7: INI ← 1
 8: FIM ← BBB
 9: EPAL ← 1
10: enquanto INI < CCC faça
11:    enquanto X[DDD] = ' ' faça
12:       INI ← INI + 1
13:    fimenquanto
14:    enquanto X[EEE] = ' ' faça
15:       FIM ← FIM - 1
16:    fimenquanto
17:    se X[INI] FFF X[FIM] então
18:       EPAL ← 0
19:    fimse
20:    INI ← GGG
21:    FIM ← HHH
22: fimenquanto
23: devolva III
24: fimfuncao
```

Esta funcao correta, deu os seguintes resultados:

C01PALI(ROTOR.) = 1 ; C01PALI(XUCRICUNICO.) = 0 ; C01PALI(BRASIL E LISARB.) = 1 .

As substituições a fazer são:

AAA=um sinal de $\neq$; BBB=TAM-1 ; CCC=FIM ; DDD=INI ; EEE=FIM ; FFF=um sinal de $\neq$; GGG=INI+1 ; HHH=FIM-1 ; III=EPAL .

Algoritmo 2 Escreva o algoritmo de uma função que receba uma chave K (Inteiro) e um vetor V (de 10 inteiros) crescente e devolva outro vetor de 11 inteiros, também crescente onde a chave K foi incluída em seu local correto.

```
1: funcao C02INCL (K, V[10] : inteiro) : R[11] inteiro
2: inteiro I,J
3: para I de 1 a AAA faça
4:    R[I] ← V[I]
5: fimpara
6: I ← BBB
7: enquanto (R[I] ≤ CCC)∧(I<DDD) faça
```

```
 8:    I ← EEE
 9: fimenquanto
10: J ← FFF
11: enquanto J ≥ I faça
12:    R[J+1] ← R[J]
13:    J ← GGG
14: fimenquanto
15: R[J+1] ← HHH
16: devolva III
17: fimfuncao
```

Esta funcao correta, deu os seguintes resultados:

C02INCL(14,7 14 21 28 35 42 49 56 63 70)=7 14 14 21 28 35 42 49 56 63 70 ; C02INCL(183,5 10 15 20 25 30 35 40 45 50)=5 10 15 20 25 30 35 40 45 50 183 ; C02INCL(2,18 26 34 42 50 58 66 74 82 90)=2 18 26 34 42 50 58 66 74 82 90 .

As substituições a fazer são:

AAA=10 ; BBB=1 ; CCC=K ; DDD=11 ; EEE=I+1 ; FFF=10 ; GGG=J-1 ; HHH=K ; III=R .

Algoritmo 3 Escreva o algoritmo de uma função que receba uma frase (cadeia de 80 caracteres, o último útil é um ponto) e devolva o número da palavra que é a maior da frase. Se houver empate (mais de uma palavra com o mesmo comprimento), devolver o número da primeira. Por exemplo, na frase "IVO VIU A UVA", a maior palavra tem 3 letras é como há empate a resposta deve ser 1. Na frase "OUVIRAM DO IPIRANGA AS MARGENS"a resposta deve ser 3.

```
 1: funcao C03PALM (X[80] : caracter) : inteiro
 2: TMS[40] : inteiro
 3: I,J,TAM,CTP,MAI,QUA : inteiro
 4: TAM ← AAA
 5: enquanto X[TAM] ≠ '.' faça
 6:    TAM ← BBB
 7: fimenquanto
 8: CTP ← 1
 9: I ← 1
10: J ← 0
11: enquanto I ≤ TAM faça
12:    se X[I] CCC ' ' então
13:       TMS[CTP] ← I - J
14:       TMS[CTP] ← TMS[CTP]-1
15:       CTP ← DDD
16:       J ← EEE
17:    fimse
18:    I ← I + 1
19: fimenquanto
20: TMS[CTP] ← I - J
21: TMS[CTP] ← TMS[CTP] - FFF
22: MAI ← -99999
23: para  I de 1 a GGG faça
24:    se TMS[I] > MAI então
25:       MAI ← TMS[I]
26:       QUA ← HHH
27:    fimse
28: fimpara
```



```
29: devolva MAI
30: fimfuncao
```

Esta funcao correta, deu os seguintes resultados:

C03PALM(DE DOMINGO E ANTES DA MEIA NOITE.)=2 ; C03PALM(DECORRIDO ENTRE O DESAPARECIMENTO.)=4 ; C03PALM(VOCE OBSERVOU EM SUAS NOTAS.)=2 .

As substituições a fazer são:

AAA=1 ; BBB=TAM+1 ; CCC=sinal de igual (=) ; DDD=CTP+1 ; EEE=I ; FFF=1 ; GGG=CTP ; HHH=I ; III=QUA .

Algoritmo 4 Escreva o algoritmo de uma função que receba uma chave K (inteiro) e um vetor V de 10 inteiros. A função deve excluir de V todas as ocorrências (se houver) de K, incluindo zeros ao final se necessário. Por exemplo, se chamada com 22 e 1 7 21 22 23 24 28 29 22 90 deve responder 1 7 21 23 24 28 29 90 0 0.

```
 1: funcao C04EXCL (K, V[10] : inteiro) : R[10] inteiro
 2: I, J : inteiro
 3: I ← AAA
 4: J ← BBB
 5: enquanto I<CCC faça
 6:    se V[I] ≠ K então
 7:       R[J] ← V[I]
 8:       J ← DDD
 9:    fimse
10:    I ← EEE
11: fimenquanto
12: enquanto J < FFF faça
13:    R[J] ← 0
14:    J ← GGG
15: fimenquanto
16: devolva R
17: fimfuncao
```

Esta funcao correta, deu os seguintes resultados:

C04EXCL(3,5 7 7 7 5 5 8 7 2 5) = 5 7 7 7 5 5 8 7 2 5 ; C04EXCL(35,40 27 3 29 49 18 8 32 11 26) = 40 27 3 29 49 18 8 32 11 26 ; C04EXCL(9,3 10 6 10 1 4 10 4 3 2) = 3 10 6 10 1 4 10 4 3 2 .

As substituições a fazer são:

AAA=1 ; BBB=1 ; CCC=11 ; DDD=J+1 ; EEE=I+1 ; FFF=11 ; GGG=J+1 .

Algoritmo 5

```
 1: funcao C05AMPL (X[10] : inteiro) : inteiro
 2: inteiro I,MAI,MEN,AMP
 3: MAI ← AAA
 4: MEN ← BBB
 5: para I de 1 a 10 faça
 6:    se X[I] > CCC então
 7:       MAI ← X[I]
 8:    fimse
 9:    se X[I] < DDD então
10:       MEN ← X[I]
11:    fimse
12: fimpara
```

13: AMP ← MAI - MEN
14: RRRR ← AMP
15: fimfuncao
16:

Esta funcao correta, deu os seguintes resultados:

C05AMPL(19 10 60 68 27 48 75 15 22 8) = 67 ; C05AMPL(1 5 1 8 7 3 6 5 3 2) = 7 ; C05AMPL(20 7 17 43 83 93 85 75 37 66) = 86 .

As substituições a fazer são:

AAA=-9999 ; BBB=+9999 ; CCC=MEN ; DDD=MEN .

☞ **Responda**

R=

38.4 Exercício 4

Construindo e consertando algoritmos - parte 3

Nesta folha, você receberá 5 algoritmos, que têm algumas lacunas representadas pelas constantes AAA, BBB, CCC, ... até III. Abaixo de cada algoritmo, é listada uma legenda que permitirá a substituição das constantes AAA,...,III por valores determinados.

Após a substituição, de duas uma:

- O algoritmo está correto
- O algoritmo contém um erro. Este erro deriva-se de uma (e apenas uma) substituição de constante por um valor errado.

Você não precisa determinar onde está o erro (se bem que isso é um interessante desafio), mas precisa apenas determinar que o algoritmo está errado.

Uma boa maneira de verificar isto é acompanhar as 3 execuções do algoritmo (correto) que acompanham o enunciado. Trata-se de algo similar a um chines reverso.

Identifique qual(is) dos 5 algoritmos estão corretos definindo A_n como: $A_1 \leftarrow 1$ se o algoritmo 1 está correto e $A_1 \leftarrow 0$ senão. Idem para A_2, A_3, A_4 e A_5.

Responda o resultado R

$$R = (A_1 \times 1) + (A_2 \times 2) + (A_3 \times 4) + (A_4 \times 8) + (A_5 \times 16)$$

Algoritmo 1 Escreva uma função que receba uma frase (cadeia de até 80 caracteres, terminada por um ponto) e devolva 1 se a frase for um palíndromo e 0 senão. Um palíndromo é aquela frase que pode ser lida tanto da direita para a esquerda como ao contrário. Exemplos:
ROMA ME TEM AMOR
SOCORRAM ME SUBI NO ONIBUS EM MARROCOS
SAIRAM O TIO E OITO MARIAS
OVO
Observação: note que as posições do espaço em branco não são as mesmas na ida e na volta.

```
 1: funcao C01PALI (X[80] : caracter) : inteiro
 2: inteiro INI,FIM,EPAL,TAM
 3: TAM ← 1
 4: enquanto X[TAM] AAA '.' faça
 5:    TAM ← TAM + 1
 6: fimenquanto
 7: INI ← 1
 8: FIM ← BBB
 9: EPAL ← 1
10: enquanto INI < CCC faça
11:    enquanto X[DDD] = ' ' faça
12:       INI ← INI + 1
13:    fimenquanto
14:    enquanto X[EEE] = ' ' faça
15:       FIM ← FIM - 1
16:    fimenquanto
17:    se X[INI] FFF X[FIM] então
18:       EPAL ← 0
19:    fimse
20:    INI ← GGG
21:    FIM ← HHH
22: fimenquanto
23: devolva III
24: fimfuncao
```

Esta funcao correta, deu os seguintes resultados:
C01PALI(ARA.) = 1 ; C01PALI(OTO COME MOCOTO.) = 1 ; C01PALI(XUCRICUNICO.) = 0 .
As substituições a fazer são:
AAA=um sinal de $\neq$; BBB=TAM-1 ; CCC=FIM ; DDD=INI ; EEE=INI ; FFF=um sinal de $\neq$; GGG=INI+1 ; HHH=FIM-1 ; III=EPAL .

Algoritmo 2 Escreva o algoritmo de uma função que receba uma chave K (Inteiro) e um vetor V (de 10 inteiros) crescente e devolva outro vetor de 11 inteiros, também crescente onde a chave K foi incluída em seu local correto.

```
1: funcao C02INCL (K, V[10] : inteiro) : R[11] inteiro
2: inteiro I,J
3: para I de 1 a AAA faça
4:    R[I] ← V[I]
```

```
 5: fimpara
 6: I ← BBB
 7: enquanto (R[I] ≤ CCC)∧(I<DDD) faça
 8:    I ← EEE
 9: fimenquanto
10: J ← FFF
11: enquanto J ≥ I faça
12:    R[J+1] ← R[J]
13:    J ← GGG
14: fimenquanto
15: R[J+1] ← HHH
16: devolva III
17: fimfuncao
```

Esta funcao correta, deu os seguintes resultados:

C02INCL(98,7 14 21 28 35 42 49 56 63 70)=7 14 21 28 35 42 49 56 63 70 98 ; C02INCL(176,7 14 21 28 35 42 49 56 63 70)=7 14 21 28 35 42 49 56 63 70 176 ; C02INCL(7,15 20 25 30 35 40 45 50 55 60)=7 15 20 25 30 35 40 45 50 55 60 .

As substituições a fazer são:

AAA=10 ; BBB=1 ; CCC=K ; DDD=11 ; EEE=I+1 ; FFF=10 ; GGG=J+1 ; HHH=K ; III=R .

Algoritmo 3 Escreva o algoritmo de uma função que receba uma frase (cadeia de 80 caracteres, o último útil é um ponto) e devolva o número da palavra que é a maior da frase. Se houver empate (mais de uma palavra com o mesmo comprimento), devolver o número da primeira. Por exemplo, na frase "IVO VIU A UVA", a maior palavra tem 3 letras é como há empate a resposta deve ser 1. Na frase "OUVIRAM DO IPIRANGA AS MARGENS"a resposta deve ser 3.

```
 1: funcao C03PALM (X[80] : caracter) : inteiro
 2: TMS[40] : inteiro
 3: I,J,TAM,CTP,MAI,QUA : inteiro
 4: TAM ← AAA
 5: enquanto X[TAM] ≠ '.' faça
 6:    TAM ← BBB
 7: fimenquanto
 8: CTP ← 1
 9: I ← 1
10: J ← 0
11: enquanto I ≤ TAM faça
12:    se X[I] CCC ' ' então
13:       TMS[CTP] ← I - J
14:       TMS[CTP] ← TMS[CTP]-1
15:       CTP ← DDD
16:       J ← EEE
17:    fimse
18:    I ← I + 1
19: fimenquanto
20: TMS[CTP] ← I - J
21: TMS[CTP] ← TMS[CTP] - FFF
22: MAI ← -99999
23: para  I de 1 a GGG faça
24:    se TMS[I] > MAI então
25:       MAI ← TMS[I]
```



```
26:       QUA ← HHH
27:    fimse
28: fimpara
29: devolva MAI
30: fimfuncao
```

Esta funcao correta, deu os seguintes resultados:

C03PALM(SE PERMITIRMOS QUE SUPONHA TAL.)=2 ; C03PALM(DE DOMINGO E ANTES DA MEIA NOITE.)=2 ; C03PALM(REDATOR E MOSTRAR TENDO.)=1 .

As substituições a fazer são:

AAA=1 ; BBB=TAM+1 ; CCC=sinal de igual (=) ; DDD=CTP+1 ; EEE=I ; FFF=2 ; GGG=CTP ; HHH=I ; III=MAI .

Algoritmo 4 Escreva o algoritmo de uma função que receba uma chave K (inteiro) e um vetor V de 10 inteiros. A função deve excluir de V todas as ocorrências (se houver) de K, incluindo zeros ao final se necessário. Por exemplo, se chamada com 22 e 1 7 21 22 23 24 28 29 22 90 deve responder 1 7 21 23 24 28 29 90 0 0.

```
1: funcao C04EXCL (K, V[10] : inteiro) : R[10] inteiro
2: I, J : inteiro
3: I ← AAA
4: J ← BBB
5: enquanto I<CCC faça
6:    se V[I] ≠ K então
7:       R[J] ← V[I]
8:       J ← DDD
9:    fimse
10:   I ← EEE
11: fimenquanto
12: enquanto J < FFF faça
13:    R[J] ← 0
14:    J ← GGG
15: fimenquanto
16: devolva R
17: fimfuncao
```

Esta funcao correta, deu os seguintes resultados:

C04EXCL(6,4 1 7 7 1 8 6 4 1 6) = 4 1 7 7 1 8 4 1 0 0 ; C04EXCL(68,21 19 18 11 29 8 43 10 39 27) = 21 19 18 11 29 8 43 10 39 27 ; C04EXCL(9,10 3 9 9 9 5 8 7 6 3) = 10 3 5 8 7 6 3 0 0 0 .

As substituições a fazer são:

AAA=1 ; BBB=1 ; CCC=11 ; DDD=J+1 ; EEE=I+1 ; FFF=11 ; GGG=J+1 .

Algoritmo 5

```
1: funcao C05AMPL (X[10] : inteiro) : inteiro
2: inteiro I,MAI,MEN,AMP
3: MAI ← AAA
4: MEN ← BBB
5: para I de 1 a 10 faça
6:    se X[I] > CCC então
7:       MAI ← X[I]
8:    fimse
9:    se X[I] < DDD então
```

```
10:       MEN ← X[I]
11:     fimse
12: fimpara
13: AMP ← MAI - MEN
14: RRRR ← AMP
15: fimfuncao
16:
```

Esta funcao correta, deu os seguintes resultados:
C05AMPL(76 38 36 62 50 19 55 79 73 78) = 60 ; C05AMPL(7 6 5 3 5 1 2 6 4 1) = 6 ; C05AMPL(30 61 95 72 10 43 33 64 48 73) = 85 .
As substituições a fazer são:
AAA=-9999 ; BBB=+9999 ; CCC=MAI ; DDD=MEN .

☞ **Responda**

R=

38.5 Respostas

1	0	1	3	7	2	1
2	4	7	0	0	0	28
3	0	0	6	0	3	11
4	5	7	9	0	0	24

Capítulo 39

Exercícios Práticos 125d - Correção de Algoritmos

39.1 Exercício 1

Construindo e consertando algoritmos - parte 4

Nesta folha, você receberá 5 algoritmos, que têm algumas lacunas representadas pelas constantes AAA, BBB, CCC, ... até III. Abaixo de cada algoritmo, é listada uma legenda que permitirá a substituição das constantes AAA,...,III por valores determinados.

Após a substituição, de duas uma:

- O algoritmo está correto
- O algoritmo contém um erro. Este erro deriva-se de uma (e apenas uma) substituição de constante por um valor errado.

Você não precisa determinar onde está o erro (se bem que isso é um interessante desafio), mas precisa apenas determinar que o algoritmo está errado.

Uma boa maneira de verificar isto é acompanhar as 3 execuções do algoritmo (correto) que acompanham o enunciado. Trata-se de algo similar a um chines reverso.

Identifique qual(is) dos 5 algoritmos estão corretos definindo A_n como: $A_1 \leftarrow 1$ se o algoritmo 1 está correto e $A_1 \leftarrow 0$ senão. Idem para A_2, A_3, A_4 e A_5.

Responda o resultado R

$$R = (A_1 \times 1) + (A_2 \times 2) + (A_3 \times 4) + (A_4 \times 8) + (A_5 \times 16)$$

Algoritmo 1 Escreva o algoritmo de uma função que receba uma matriz de 9 inteiros (3 linhas por 3 colunas), totalize as colunas e devolva um vetor de 3 totais.

```
1: função D01VESO (M [1..3] [1..3] : inteiro) V[1..3]:inteiro
2: I,J,SOM : inteiro
3: V ← 0,0,0
4: para  AAA de 1 a 3 faça
5:    BBB ← 0
6:    para  CCC de 1 a 3 faça
7:       SOM ← SOM + M[I] [J]
8:    fimpara
9:    V[DDD] ← SOM
10: fimpara
```

```
11: retorne V
12: fimfunção
```

As substituições a fazer são:
AAA=J ; BBB=SOM ; CCC=I ; DDD=J .
Esta função correta, deu os seguintes resultados:
D01VESO (6 4 9,9 5 3,1 7 8) = 16 16 20 ; D01VESO (5 6 1,9 3 3,9 3 7) = 23 12 11 ; D01VESO (3 3 7,3 5 7,1 5 3) = 7 13 17 .

Algoritmo 2 Escreva o algoritmo de uma função que receba uma matriz de 4 linhas por 4 colunas de números inteiros. A função deve gerar um vetor de 4 números reais. O elemento k da resposta é calculado como a soma dos elementos da coluna k, que estão acima da linha k, dividido pela soma dos elementos que estão na coluna k e nas linhas k, $k+1$, ..., 4.

```
 1: função D02MSMI (M[1..4] [1..4] : inteiro) V[1..4]:real
 2: J,NUM,DEN,I : inteiro
 3: para  J de 1 a 4 faça
 4:    NUM ← AAA
 5:    DEN ← 0
 6:    para  I de 1 a BBB faça
 7:       NUM ← NUM + M[I][EEE]
 8:    fimpara
 9:    para  I de CCC a 4 faça
10:       DEN ← DEN + M[DDD][J]
11:    fimpara
12:    V[J] ← FFF ÷ DEN
13: fimpara
14: retorne V
15: fimfunção
```

As substituições a fazer são:
AAA=0 (zero) ; BBB=J-1 ; CCC=1 (um) ; DDD=I ; EEE=J ; FFF=NUM .
Esta função correta, deu os seguintes resultados:
D02MSMI (5 1 12 13, 6 4 1 15, 4 12 1 3,10 4 13 10)= .0 .1 .9 3.1 ; D02MSMI (18 20 3 7,18 18 18 4,14 9 4 1,10 1 13 10)= .0 .7 1.2 1.2 ; D02MSMI (9 1 11 3,20 3 3 17, 2 16 18 4,19 8 13 16)= .0 .0 .5 1.5 .

Algoritmo 3 Escreva o algoritmo de uma função que receba uma matriz de 4 linhas por 5 colunas contendo inteiros e devolva um vetor de 4 inteiros que contém o menor valor em cada linha.

```
 1: função D03MEHO (M[1..4][1..5]:inteiro) V[1..4]:inteiro
 2: I,J,MEN : inteiro
 3: para  I de 1 a 4 faça
 4:    MEN ← AAA
 5:    para  J de 1 a BBB faça
 6:       se M[I][CCC] < MEN então
 7:          MEN ← M[I][J]
 8:       fimse
 9:    fimpara
10:    V[I] ← DDD
11: fimpara
12: retorne V
13: fimfunção
```

As substituições a fazer são:
AAA=999999 ; BBB=5 ; CCC=J ; DDD=J .
Esta função correta, deu os seguintes resultados:
D03MEHO (3 2 2 3 2, 5 7 2 6 10,10 3 1 10 3, 9 6 6 9 1)=2 2 1 1 ; D03MEHO (7 2 1 9 8, 4 4 8 3 8, 9 7 4 9 1,10 9 3 3 1)=1 3 1 1 ; D03MEHO (4 6 7 4 10,2 4 10 5 9,9 4 3 7 10,3 2 10 3 3)=4 2 3 2 .

Algoritmo 4 Escreva uma função que receba duas matrizes conformáveis, isto é a primeiro de 3 linhas por 4 colunas e a segunda de 4 linhas por 3 colunas. A função deve multiplicar as duas matrizes, gerando o resultado que terá 3 linhas por 3 colunas.

```
1: função D04MUMA (A[1..3][1..4]:inteiro,
2: B[1..4][1..3]:inteiro)
3: V[1..3][1..3]:real // até aqui é o cabeçalho...
4: I,J,K,SOM:inteiro
5: para I de 1 a AAA faça
6:    para J de 1 a BBB faça
7:       SOM ← 0
8:       para K de 1 a CCC faça
9:          SOM ← SOM+(A[I][DDD]×B[EEE][J])
10:      fimpara
11:      V[FFF][J] ← SOM
12:   fimpara
13: fimpara
14: retorne V
15: fimfunção
```

As substituições a fazer são:
AAA=3 ; BBB=3 ; CCC=4 ; DDD=K ; EEE=K ; FFF=K .
Esta função correta, deu os seguintes resultados:
D04MUMA (5 6 6 5,3 4 1 5,2 5 1 2;1 3 5,5 1 6,4 1 0,4 3 5) = 79 42 86,47 29 64,39 18 50 ; D04MUMA (0 1 5 3,5 2 6 2,2 4 2 5;4 1 2,5 1 2,6 5 6,1 5 4) = 38 41 44,68 47 58,45 41 44 ; D04MUMA (2 0 0 3,4 6 6 2,6 2 0 6;6 6 3,5 4 6,4 3 3,2 4 2) = 18 24 12,82 74 70,58 68 42 .

Algoritmo 5 Escreva o algoritmo de uma função que receba uma matriz de 6 linhas por 2 colunas contendo o resultado de uma eleição. A primeira coluna corresponde ao número do candidato, sendo que nem todas as linhas estarão preenchidas. Neste caso as linhas não usadas conterão zeros. Existem dois códigos em especial que estarão sempre presentes e que são 998=votos em branco e 999=votos nulos. A segunda coluna contém a quantidade de votos a cada candidato e também brancos e nulos.

A função deve devolver o número do candidato eleito, ou -1 se a eleição não for válida. O eleito é aquele que obteve o maior número de votos se a eleição foi válida. Uma eleição é válida quando o total de votos dados a candidatos mais os votos em branco corresponderem a 50% mais 1 voto do total de votantes. Se o vencedor for o branco ou o nulo, vence o candidato da linha 1 (por hipótese é o mais velho).

```
1: função D05ELEI (VOT[1..6][1..2]:inteiro):inteiro
2: VAL,I,NUL,TOT,MAI,QUEM:inteiro
3: VAL ← 0
4: para I de 1 a AAA faça
5:    se VOT[I][1] ≠ BBB então
6:       VAL ← VAL + VOT[I][CCC]
7:    senão
```



```
 8:       NUL ← VOT[I][2]
 9:    fimse
10: fimpara
11: TOT ← VAL ÷ 2
12: se NUL > DDD então
13:    retorne -1
14: senão
15:    MAI ← -999999
16:    para  I de 1 a 6 faça
17:       se VOT[I][EEE] > MAI então
18:          FFF ← VOT[I][2]
19:          QUEM ← VOT[I][1]
20:       fimse
21:    fimpara
22:    se (GGG = 998) ∨ (GGG=999) então
23:       retorne 1
24:    senão
25:       retorne QUEM
26:    fimse
27: fimse
28: fimfunção
```

As substituições a fazer são:
AAA=6 ; BBB=999 ; CCC=2 ; DDD=NUL ; EEE=2 ; FFF=MAI ; GGG=QUEM .

Esta função correta, deu os seguintes resultados:
D05ELEI (40 1400, 10 800, 0 0, 0 0,998 180,999 120) = 40 ; D05ELEI (70 80, 60 220, 80 320, 10 200,998 80,999 390) = 1 ; D05ELEI (60 60, 10 100, 70 140, 0 0,998 320,999 60) = 1 .

☞ **Responda**

R=

39.2 Exercício 2

Construindo e consertando algoritmos - parte 4

Nesta folha, você receberá 5 algoritmos, que têm algumas lacunas representadas pelas constantes AAA, BBB, CCC, ... até III. Abaixo de cada algoritmo, é listada uma legenda que permitirá a substituição das constantes AAA,...,III por valores determinados.

Após a substituição, de duas uma:

- O algoritmo está correto
- O algoritmo contém um erro. Este erro deriva-se de uma (e apenas uma) substituição de constante por um valor errado.

Você não precisa determinar onde está o erro (se bem que isso é um interessante desafio), mas precisa apenas determinar que o algoritmo está errado.

Uma boa maneira de verificar isto é acompanhar as 3 execuções do algoritmo (correto) que acompanham o enunciado. Trata-se de algo similar a um chines reverso.

Identifique qual(is) dos 5 algoritmos estão corretos definindo A_n como: $A_1 \leftarrow 1$ se o algoritmo 1 está correto e $A_1 \leftarrow 0$ senão. Idem para A_2, A_3, A_4 e A_5.

Responda o resultado R

$$R = (A_1 \times 1) + (A_2 \times 2) + (A_3 \times 4) + (A_4 \times 8) + (A_5 \times 16)$$

Algoritmo 1 Escreva o algoritmo de uma função que receba uma matriz de 9 inteiros (3 linhas por 3 colunas), totalize as colunas e devolva um vetor de 3 totais.

```
 1: função D01VESO (M [1..3] [1..3] : inteiro) V[1..3]:inteiro
 2: I,J,SOM : inteiro
 3: V ← 0,0,0
 4: para  AAA de 1 a 3 faça
 5:    BBB ← 0
 6:    para  CCC de 1 a 3 faça
 7:       SOM ← SOM + M[I] [J]
 8:    fimpara
 9:    V[DDD] ← SOM
10: fimpara
11: retorne V
12: fimfunção
```

As substituições a fazer são:

AAA=J ; BBB=SOM ; CCC=J ; DDD=J .

Esta função correta, deu os seguintes resultados:

D01VESO (7 2 2,1 1 1,3 7 2) = 11 10 5 ; D01VESO (3 5 5,5 6 9,9 1 9) = 17 12 23 ; D01VESO (6 4 7,3 7 8,5 5 3) = 14 16 18 .

Algoritmo 2 Escreva o algoritmo de uma função que receba uma matriz de 4 linhas por 4 colunas de números inteiros. A função deve gerar um vetor de 4 números reais. O elemento k da resposta é calculado como a soma dos elementos da coluna k, que estão acima da linha k, dividido pela soma dos elementos que estão na coluna k e nas linhas k, $k+1$, ..., 4.

```
 1: função D02MSMI (M[1..4] [1..4] : inteiro) V[1..4]:real
 2: J,NUM,DEN,I : inteiro
 3: para  J de 1 a 4 faça
 4:    NUM ← AAA
 5:    DEN ← 0
 6:    para  I de 1 a BBB faça
 7:       NUM ← NUM + M[I][EEE]
 8:    fimpara
 9:    para  I de CCC a 4 faça
10:       DEN ← DEN + M[DDD][J]
11:    fimpara
12:    V[J] ← FFF ÷ DEN
13: fimpara
14: retorne V
15: fimfunção
```

As substituições a fazer são:

AAA=0 (zero) ; BBB=J-1 ; CCC=J ; DDD=J ; EEE=J ; FFF=NUM .

Esta função correta, deu os seguintes resultados:

D02MSMI (11 5 15 5,11 18 17 15,12 3 4 11,11 17 8 14)= .0 .1 2.7 2.2 ; D02MSMI (3 17 16 5,11 10 6 16,17 20 4 13, 8 3 7 14)= .0 .5 2.0 2.4 ; D02MSMI (16 13 18 5, 3 4 18 17,16 18 20 20, 1 2 20 15)= .0 .5 .9 2.8 .

Algoritmo 3 Escreva o algoritmo de uma função que receba uma matriz de 4 linhas por 5 colunas contendo inteiros e devolva um vetor de 4 inteiros que contém o menor valor em cada linha.

```
 1: função D03MEHO (M[1..4][1..5]:inteiro) V[1..4]:inteiro
 2: I,J,MEN : inteiro
 3: para I de 1 a 4 faça
 4:    MEN ← AAA
 5:    para J de 1 a BBB faça
 6:       se M[I][CCC] < MEN então
 7:          MEN ← M[I][J]
 8:       fimse
 9:    fimpara
10:    V[I] ← DDD
11: fimpara
12: retorne V
13: fimfunção
```

As substituições a fazer são:

AAA=0 (zero) ; BBB=5 ; CCC=J ; DDD=MEN .

Esta função correta, deu os seguintes resultados:

D03MEHO (5 3 4 7 3,8 7 4 9 5,3 2 10 3 1,9 2 9 2 4)=3 4 1 2 ; D03MEHO (5 9 6 5 3, 9 7 2 4 5,10 9 9 2 8, 7 8 8 5 10)=3 2 2 5 ; D03MEHO (2 10 2 10 5,6 2 1 1 4,8 8 10 7 1,3 2 1 6 3)=2 1 1 1 .

Algoritmo 4 Escreva uma função que receba duas matrizes conformáveis, isto é a primeiro de 3 linhas por 4 colunas e a segunda de 4 linhas por 3 colunas. A função deve multiplicar as duas matrizes, gerando o resultado que terá 3 linhas por 3 colunas.

```
 1: função D04MUMA (A[1..3][1..4]:inteiro,
 2: B[1..4][1..3]:inteiro)
 3: V[1..3][1..3]:real // até aqui é o cabeçalho...
 4: I,J,K,SOM:inteiro
 5: para I de 1 a AAA faça
 6:    para J de 1 a BBB faça
 7:       SOM ← 0
 8:       para K de 1 a CCC faça
 9:          SOM ← SOM+(A[I][DDD]×B[EEE][J])
10:       fimpara
11:       V[FFF][J] ← SOM
12:    fimpara
13: fimpara
14: retorne V
15: fimfunção
```

As substituições a fazer são:

AAA=3 ; BBB=3 ; CCC=4 ; DDD=K ; EEE=K ; FFF=I .

Esta função correta, deu os seguintes resultados:

D04MUMA (6 2 1 1,3 3 1 1,3 2 5 3;6 6 2,1 3 1,0 5 4,6 3 0) = 44 50 18,27 35 13,38 58 28 ; D04MUMA (0 6 5 1,3 5 2 0,4 1 3 1;0 3 2,2 2 2,1 6 3,0 3 1) = 17 45 28,12 31 22, 5 35 20 ; D04MUMA (1 0 4 2,6 3 5 5,1 5 1 6;3 0 4,0 1 3,3 4 5,0 4 0) = 15 24 24,33 43 58, 6 33 24 .

Algoritmo 5 Escreva o algoritmo de uma função que receba uma matriz de 6 linhas por 2 colunas contendo o resultado de uma eleição. A primeira coluna corresponde ao

número do candidato, sendo que nem todas as linhas estarão preenchidas. Neste caso as linhas não usadas conterão zeros. Existem dois códigos em especial que estarão sempre presentes e que são 998=votos em branco e 999=votos nulos. A segunda coluna contém a quantidade de votos a cada candidato e também brancos e nulos.

A função deve devolver o número do candidato eleito, ou -1 se a eleição não for válida. O eleito é aquele que obteve o maior número de votos se a eleição foi válida. Uma eleição é válida quando o total de votos dados a candidatos mais os votos em branco corresponderem a 50% mais 1 voto do total de votantes. Se o vencedor for o branco ou o nulo, vence o candidato da linha 1 (por hipótese é o mais velho).

```
 1: função D05ELEI (VOT[1..6][1..2]:inteiro):inteiro
 2: VAL,I,NUL,TOT,MAI,QUEM:inteiro
 3: VAL ← 0
 4: para  I de 1 a AAA faça
 5:    se VOT[I][1] ≠ BBB então
 6:       VAL ← VAL + VOT[I][CCC]
 7:    senão
 8:       NUL ← VOT[I][2]
 9:    fimse
10: fimpara
11: TOT ← VAL ÷ 2
12: se NUL > DDD então
13:    retorne -1
14: senão
15:    MAI ← -999999
16:    para  I de 1 a 6 faça
17:       se VOT[I][EEE] > MAI então
18:          FFF ← VOT[I][2]
19:          QUEM ← VOT[I][1]
20:       fimse
21:    fimpara
22:    se (GGG = 998) ∨ (GGG=999) então
23:       retorne 1
24:    senão
25:       retorne QUEM
26:    fimse
27: fimse
28: fimfunção
```

As substituições a fazer são:

AAA=6 ; BBB=999 ; CCC=2 ; DDD=TOT ; EEE=2 ; FFF=MAI ; GGG=QUEM .

Esta função correta, deu os seguintes resultados:

D05ELEI (40 160, 60 80, 20 260, 50 180,998 340,999 60) = 1 ; D05ELEI (90 360, 60 400, 10 220, 20 40,998 280,999 450) = 1 ; D05ELEI (40 340, 50 360, 30 20, 70 260,998 200,999 210) = 50 .

☞ **Responda**

R=

39.3 Exercício 2

Construindo e consertando algoritmos - parte 4

Nesta folha, você receberá 5 algoritmos, que têm algumas lacunas representadas pelas constantes AAA, BBB, CCC, ... até III. Abaixo de cada algoritmo, é listada uma legenda que permitirá a substituição das constantes AAA,...,III por valores determinados.

Após a substituição, de duas uma:

- O algoritmo está correto
- O algoritmo contém um erro. Este erro deriva-se de uma (e apenas uma) substituição de constante por um valor errado.

Você não precisa determinar onde está o erro (se bem que isso é um interessante desafio), mas precisa apenas determinar que o algoritmo está errado.

Uma boa maneira de verificar isto é acompanhar as 3 execuções do algoritmo (correto) que acompanham o enunciado. Trata-se de algo similar a um chines reverso.

Identifique qual(is) dos 5 algoritmos estão corretos definindo A_n como: $A_1 \leftarrow 1$ se o algoritmo 1 está correto e $A_1 \leftarrow 0$ senão. Idem para A_2, A_3, A_4 e A_5.

Responda o resultado R

$$R = (A_1 \times 1) + (A_2 \times 2) + (A_3 \times 4) + (A_4 \times 8) + (A_5 \times 16)$$

Algoritmo 1 Escreva o algoritmo de uma função que receba uma matriz de 9 inteiros (3 linhas por 3 colunas), totalize as colunas e devolva um vetor de 3 totais.

```
1: função D01VESO (M [1..3] [1..3] : inteiro) V[1..3]:inteiro
2: I,J,SOM : inteiro
3: V ← 0,0,0
4: para AAA de 1 a 3 faça
5:    BBB ← 0
6:    para CCC de 1 a 3 faça
7:       SOM ← SOM + M[I] [J]
8:    fimpara
9:    V[DDD] ← SOM
10: fimpara
11: retorne V
12: fimfunção
```

As substituições a fazer são:
AAA=J ; BBB=SOM ; CCC=I ; DDD=J .
Esta função correta, deu os seguintes resultados:
D01VESO (2 8 8,2 5 6,4 9 8) = 8 22 22 ; D01VESO (6 5 5,7 3 4,1 3 2) = 14 11 11 ; D01VESO (3 5 7,9 1 2,4 7 9) = 16 13 18 .

Algoritmo 2 Escreva o algoritmo de uma função que receba uma matriz de 4 linhas por 4 colunas de números inteiros. A função deve gerar um vetor de 4 números reais. O elemento k da resposta é calculado como a soma dos elementos da coluna k, que estão acima da linha k, dividido pela soma dos elementos que estão na coluna k e nas linhas k, $k+1$, ..., 4.

```
1: função D02MSMI (M[1..4] [1..4] : inteiro) V[1..4]:real
2: J,NUM,DEN,I : inteiro
3: para J de 1 a 4 faça
4:    NUM ← AAA
```

```
 5:    DEN ← 0
 6:    para  I de 1 a BBB faça
 7:       NUM ← NUM + M[I][EEE]
 8:    fimpara
 9:    para  I de CCC a 4 faça
10:       DEN ← DEN + M[DDD][J]
11:    fimpara
12:    V[J] ← FFF ÷ DEN
13: fimpara
14: retorne V
15: fimfunção
```

As substituições a fazer são:
AAA=1 (um) ; BBB=J-1 ; CCC=J ; DDD=I ; EEE=J ; FFF=NUM .
Esta função correta, deu os seguintes resultados:
D02MSMI (20 12 6 6,17 10 1 12,13 8 8 12, 8 15 1 16)= .0 .4 .8 1.9 ; D02MSMI (16 16 5 19, 9 8 3 3, 2 19 17 16,20 7 12 13)= .0 .5 .3 2.9 ; D02MSMI (18 20 7 10,14 5 4 9, 2 13 5 18,20 12 11 17)= .0 .7 .7 2.2 .

Algoritmo 3 Escreva o algoritmo de uma função que receba uma matriz de 4 linhas por 5 colunas contendo inteiros e devolva um vetor de 4 inteiros que contém o menor valor em cada linha.

```
 1: função D03MEHO (M[1..4][1..5]:inteiro) V[1..4]:inteiro
 2: I,J,MEN : inteiro
 3: para  I de 1 a 4 faça
 4:    MEN ← AAA
 5:    para  J de 1 a BBB faça
 6:       se M[I][CCC] < MEN então
 7:          MEN ← M[I][J]
 8:       fimse
 9:    fimpara
10:    V[I] ← DDD
11: fimpara
12: retorne V
13: fimfunção
```

As substituições a fazer são:
AAA=999999 ; BBB=5 ; CCC=J ; DDD=J .
Esta função correta, deu os seguintes resultados:
D03MEHO (2 10 8 3 7, 7 9 9 5 3, 4 7 6 7 3,10 10 7 6 2)=2 3 3 2 ; D03MEHO (5 2 10 5 8,7 1 6 1 3,3 5 4 6 3,5 7 4 10 6)=2 1 3 4 ; D03MEHO (3 2 7 7 7,10 4 3 10 3,10 4 2 1 1, 1 3 4 4 8)=2 3 1 1 .

Algoritmo 4 Escreva uma função que receba duas matrizes conformáveis, isto é a primeiro de 3 linhas por 4 colunas e a segunda de 4 linhas por 3 colunas. A função deve multiplicar as duas matrizes, gerando o resultado que terá 3 linhas por 3 colunas.

```
 1: função D04MUMA (A[1..3][1..4]:inteiro,
 2: B[1..4][1..3]:inteiro)
 3: V[1..3][1..3]:real // até aqui é o cabeçalho...
 4: I,J,K,SOM:inteiro
 5: para  I de 1 a AAA faça
 6:    para  J de 1 a BBB faça
 7:       SOM ← 0
```



```
 8:       para  K de 1 a CCC faça
 9:         SOM ← SOM+(A[I][DDD]×B[EEE][J])
10:       fimpara
11:       V[FFF][J] ← SOM
12:     fimpara
13: fimpara
14: retorne V
15: fimfunção
```

As substituições a fazer são:
AAA=3 ; BBB=4 ; CCC=4 ; DDD=K ; EEE=K ; FFF=I .
Esta função correta, deu os seguintes resultados:
D04MUMA (2 1 5 3,0 1 4 6,0 0 4 1;0 3 2,0 3 5,1 3 3,6 4 4) = 23 36 36,40 39 41,10 16 16 ; D04MUMA (0 3 1 2,4 2 2 3,0 5 3 1;4 0 1,1 3 0,6 5 4,5 4 3) = 19 22 10,45 28 21,28 34 15 ; D04MUMA (3 6 3 1,1 5 0 0,0 1 4 4;5 3 2,3 1 3,5 5 4,2 6 4) = 50 36 40,20 8 17,31 45 35 .

Algoritmo 5 Escreva o algoritmo de uma função que receba uma matriz de 6 linhas por 2 colunas contendo o resultado de uma eleição. A primeira coluna corresponde ao número do candidato, sendo que nem todas as linhas estarão preenchidas. Neste caso as linhas não usadas conterão zeros. Existem dois códigos em especial que estarão sempre presentes e que são 998=votos em branco e 999=votos nulos. A segunda coluna contém a quantidade de votos a cada candidato e também brancos e nulos.

A função deve devolver o número do candidato eleito, ou -1 se a eleição não for válida. O eleito é aquele que obteve o maior número de votos se a eleição foi válida. Uma eleição é válida quando o total de votos dados a candidatos mais os votos em branco corresponderem a 50% mais 1 voto do total de votantes. Se o vencedor for o branco ou o nulo, vence o candidato da linha 1 (por hipótese é o mais velho).

```
 1: função D05ELEI (VOT[1..6][1..2]:inteiro):inteiro
 2: VAL,I,NUL,TOT,MAI,QUEM:inteiro
 3: VAL ← 0
 4: para  I de 1 a AAA faça
 5:   se VOT[I][1] ≠ BBB então
 6:     VAL ← VAL + VOT[I][CCC]
 7:   senão
 8:     NUL ← VOT[I][2]
 9:   fimse
10: fimpara
11: TOT ← VAL ÷ 2
12: se NUL > DDD então
13:   retorne -1
14: senão
15:   MAI ← -999999
16:   para  I de 1 a 6 faça
17:     se VOT[I][EEE] > MAI então
18:       FFF ← VOT[I][2]
19:       QUEM ← VOT[I][1]
20:     fimse
21:   fimpara
22:   se (GGG = 998) ∨ (GGG=999) então
23:     retorne 1
24:   senão
25:     retorne QUEM
```

```
26:    fimse
27: fimse
28: fimfunção
```

As substituições a fazer são:

AAA=6 ; BBB=999 ; CCC=2 ; DDD=TOT ; EEE=2 ; FFF=MAI ; GGG=QUEM .

Esta função correta, deu os seguintes resultados:

D05ELEI (20 320, 40 260, 10 300, 50 220,998 240,999 330) = 1 ; D05ELEI (50 160, 30 360, 90 300, 40 60,998 380,999 240) = 1 ; D05ELEI (40 120, 20 40, 30 300, 80 320,998 380,999 570) = 1 .

☞ **Responda**

R=

39.4 Exercício 3

Construindo e consertando algoritmos - parte 4

Nesta folha, você receberá 5 algoritmos, que têm algumas lacunas representadas pelas constantes AAA, BBB, CCC, ... até III. Abaixo de cada algoritmo, é listada uma legenda que permitirá a substituição das constantes AAA,...,III por valores determinados.

Após a substituição, de duas uma:

- O algoritmo está correto
- O algoritmo contém um erro. Este erro deriva-se de uma (e apenas uma) substituição de constante por um valor errado.

Você não precisa determinar onde está o erro (se bem que isso é um interessante desafio), mas precisa apenas determinar que o algoritmo está errado.

Uma boa maneira de verificar isto é acompanhar as 3 execuções do algoritmo (correto) que acompanham o enunciado. Trata-se de algo similar a um chines reverso.

Identifique qual(is) dos 5 algoritmos estão corretos definindo A_n como: $A_1 \leftarrow 1$ se o algoritmo 1 está correto e $A_1 \leftarrow 0$ senão. Idem para A_2, A_3, A_4 e A_5.

Responda o resultado R

$$R = (A_1 \times 1) + (A_2 \times 2) + (A_3 \times 4) + (A_4 \times 8) + (A_5 \times 16)$$

Algoritmo 1 Escreva o algoritmo de uma função que receba uma matriz de 9 inteiros (3 linhas por 3 colunas), totalize as colunas e devolva um vetor de 3 totais.

```
 1: função D01VESO (M [1..3] [1..3] : inteiro) V[1..3]:inteiro
 2: I,J,SOM : inteiro
 3: V ← 0,0,0
 4: para  AAA de 1 a 3 faça
 5:    BBB ← 0
 6:    para  CCC de 1 a 3 faça
 7:       SOM ← SOM + M[I] [J]
 8:    fimpara
 9:    V[DDD] ← SOM
10: fimpara
```



```
11: retorne V
12: fimfunção
```

As substituições a fazer são:
AAA=I ; BBB=SOM ; CCC=I ; DDD=J .
Esta função correta, deu os seguintes resultados:
D01VESO (2 1 9,3 8 2,8 4 2) = 13 13 13 ; D01VESO (2 8 9,6 6 5,4 7 8) = 12 21 22 ; D01VESO (2 1 1,4 9 5,5 7 4) = 11 17 10 .

Algoritmo 2 Escreva o algoritmo de uma função que receba uma matriz de 4 linhas por 4 colunas de números inteiros. A função deve gerar um vetor de 4 números reais. O elemento k da resposta é calculado como a soma dos elementos da coluna k, que estão acima da linha k, dividido pela soma dos elementos que estão na coluna k e nas linhas k, $k+1$, ..., 4.

```
 1: função D02MSMI (M[1..4] [1..4] : inteiro) V[1..4]:real
 2: J,NUM,DEN,I : inteiro
 3: para  J de 1 a 4 faça
 4:    NUM ← AAA
 5:    DEN ← 0
 6:    para  I de 1 a BBB faça
 7:       NUM ← NUM + M[I][EEE]
 8:    fimpara
 9:    para  I de CCC a 4 faça
10:       DEN ← DEN + M[DDD][J]
11:    fimpara
12:    V[J] ← FFF ÷ DEN
13: fimpara
14: retorne V
15: fimfunção
```

As substituições a fazer são:
AAA=0 (zero) ; BBB=J-1 ; CCC=J ; DDD=I ; EEE=J ; FFF=NUM .
Esta função correta, deu os seguintes resultados:
D02MSMI (11 12 11 13, 9 5 4 13,11 5 4 11,10 20 2 1)= .0 .4 2.5 37.0 ; D02MSMI (16 11 13 5, 2 13 8 12,14 7 13 11, 3 14 15 2)= .0 .3 .8 14.0 ; D02MSMI (2 18 8 19,14 19 15 3, 2 4 14 20,12 9 7 8)= .0 .6 1.1 5.3 .

Algoritmo 3 Escreva o algoritmo de uma função que receba uma matriz de 4 linhas por 5 colunas contendo inteiros e devolva um vetor de 4 inteiros que contém o menor valor em cada linha.

```
 1: função D03MEHO (M[1..4][1..5]:inteiro) V[1..4]:inteiro
 2: I,J,MEN : inteiro
 3: para  I de 1 a 4 faça
 4:    MEN ← AAA
 5:    para  J de 1 a BBB faça
 6:       se M[I][CCC] < MEN então
 7:          MEN ← M[I][J]
 8:       fimse
 9:    fimpara
10:    V[I] ← DDD
11: fimpara
12: retorne V
13: fimfunção
```

As substituições a fazer são:
AAA=999999 ; BBB=5 ; CCC=I ; DDD=MEN .
Esta função correta, deu os seguintes resultados:
D03MEHO (2 8 4 9 1,6 2 5 5 1,5 4 2 7 6,7 10 8 9 5)=1 1 2 5 ; D03MEHO (9 6 7 4 1,9 7 4 7 7,1 9 2 3 7,7 7 7 8 9)=1 4 1 7 ; D03MEHO (8 10 6 3 3, 9 7 1 2 5,10 2 10 1 9, 8 4 2 3 3)=3 1 1 2 .

Algoritmo 4 Escreva uma função que receba duas matrizes conformáveis, isto é a primeiro de 3 linhas por 4 colunas e a segunda de 4 linhas por 3 colunas. A função deve multiplicar as duas matrizes, gerando o resultado que terá 3 linhas por 3 colunas.

```
1: função D04MUMA (A[1..3][1..4]:inteiro,
2: B[1..4][1..3]:inteiro)
3: V[1..3][1..3]:real // até aqui é o cabeçalho...
4: I,J,K,SOM:inteiro
5: para I de 1 a AAA faça
6:    para J de 1 a BBB faça
7:       SOM ← 0
8:       para K de 1 a CCC faça
9:          SOM ← SOM+(A[I][DDD]×B[EEE][J])
10:      fimpara
11:      V[FFF][J] ← SOM
12:   fimpara
13: fimpara
14: retorne V
15: fimfunção
```

As substituições a fazer são:
AAA=4 ; BBB=3 ; CCC=4 ; DDD=K ; EEE=K ; FFF=I .
Esta função correta, deu os seguintes resultados:
D04MUMA (5 1 1 3,0 5 0 3,6 2 1 3;1 3 2,6 2 2,3 3 5,3 4 6) = 23 32 35,39 22 28,30 37 39 ; D04MUMA (5 0 5 4,4 6 6 5,4 4 6 1;0 0 6,5 3 2,2 6 2,3 1 6) = 22 34 64,57 59 78,35 49 50 ; D04MUMA (2 3 1 4,4 4 6 5,2 1 0 1;3 5 3,1 6 2,1 6 4,2 5 0) = 18 54 16,32 105 44, 9 21 8 .

Algoritmo 5 Escreva o algoritmo de uma função que receba uma matriz de 6 linhas por 2 colunas contendo o resultado de uma eleição. A primeira coluna corresponde ao número do candidato, sendo que nem todas as linhas estarão preenchidas. Neste caso as linhas não usadas conterão zeros. Existem dois códigos em especial que estarão sempre presentes e que são 998=votos em branco e 999=votos nulos. A segunda coluna contém a quantidade de votos a cada candidato e também brancos e nulos.

A função deve devolver o número do candidato eleito, ou -1 se a eleição não for válida. O eleito é aquele que obteve o maior número de votos se a eleição foi válida. Uma eleição é válida quando o total de votos dados a candidatos mais os votos em branco corresponderem a 50% mais 1 voto do total de votantes. Se o vencedor for o branco ou o nulo, vence o candidato da linha 1 (por hipótese é o mais velho).

```
1: função D05ELEI (VOT[1..6][1..2]:inteiro):inteiro
2: VAL,I,NUL,TOT,MAI,QUEM:inteiro
3: VAL ← 0
4: para I de 1 a AAA faça
5:    se VOT[I][1] ≠ BBB então
6:       VAL ← VAL + VOT[I][CCC]
7:    senão
```



```
 8:       NUL ← VOT[I][2]
 9:    fimse
10: fimpara
11: TOT ← VAL ÷ 2
12: se NUL > DDD então
13:    retorne -1
14: senão
15:    MAI ← -999999
16:    para  I de 1 a 6 faça
17:       se VOT[I][EEE] > MAI então
18:          FFF ← VOT[I][2]
19:          QUEM ← VOT[I][1]
20:       fimse
21:    fimpara
22:    se (GGG = 998) ∨ (GGG=999) então
23:       retorne 1
24:    senão
25:       retorne QUEM
26:    fimse
27: fimse
28: fimfunção
```

As substituições a fazer são:
AAA=6 ; BBB=999 ; CCC=2 ; DDD=TOT ; EEE=2 ; FFF=MAI ; GGG=QUEM .

Esta função correta, deu os seguintes resultados:
D05ELEI (70 380, 20 400, 50 80, 0 0,998 300,999 90) = 20 ; D05ELEI (10 120, 40 260, 30 140, 50 340,998 320,999 180) = 50 ; D05ELEI (90 1300, 30 2000, 0 0, 0 0,998 180,999 60) = 30 .

☞ **Responda**

R=

39.5 Respostas

número do exercício	linhas onde ocorreu o erro	soma dos certos
1	0 3 4 6 4	1
2	3 4 1 0 0	24
3	0 1 4 2 0	17
4	1 0 3 1 0	18

Capítulo 40

Exercícios Práticos: 128 - Matrizes

Totalização de matrizes É comum ter-se uma matriz de valores referentes a um dado fenômeno e pretender-se obter totais em qualquer uma das duas dimensões. Acompanhe: Seja a matriz M composta pelo resultado das vendas de 7 filiais (7 colunas) nos 22 dias úteis (linhas) do mês passado. Pretende-se responder algumas perguntas:

a) Qual a maior filial ?; b) qual o dia do mês de maior movimento?; c) Qual a amplitude de filiais ? (a maior menos a menor); d) qual a amplitude de dias ? e) Quantos dias respondem por 50% das vendas ? etc etc.

Para responder a perguntas como essas, há que se criar um vetor de 7 valores (soma das filiais) e um de 22 valores para a soma dos dias. Acompanhe o algoritmo

```
1: real VF[7], VD[22] função SOMA (real VENDAS[22][7])
2: inteiro J, K
3: VF ← 0
4: VD ← 0
5: para J de 1 até 22 faça
6:    para K de 1 até 7 faça
7:       VD[J] ← VD[J] + VENDAS[J][K]
8:       VF[K] ← VF[K] + VENDAS[J][K]
9:    fimpara
10: fimpara
11: fim {função}
```

Transposta Supondo a matriz M de i linhas por j colunas, a matriz transposta de M, reconhecida como $\phi M = M'$ é aquela obtida rotacionando-se seus elementos. M' terá j linhas por i colunas. Acompanhe o algoritmo

```
1: (real TRA[j][i]) função ACHATRANSP (real M[i][j])
2: inteiro K, L
3: para K de 1 até i faça
4:    para L de 1 até j faça
5:       TRA[L][K] ← M[K][L]
6:    fimpara
7: fimpara
8: devolva TRA
9: fim{função}
```

Zerando a matriz abaixo da diagonal principal Este é um problema típico em teoria de grafos. A matriz de adjacência de um digrafo (grafo direcionado) usa todas as linhas da matriz. Se este grafo passar a ser considerado como não direcionado, a parte abaixo da diagonal principal precisa ser zerada. Eis o algoritmo que o faz:

```
real MZ[i][i] função ZERAB (real M[i][i])
inteiro J, K
para J de 1 até i faça
   para K de 1 até i faça
      se K < J então
         MZ[J][K] ← 0
      senão
         MZ[J][K] ← M[J][K]
      fimse
   fimpara
fimpara
devolva MZ
fim{função}
```

multiplicação matricial Este é um algoritmo famoso na matemática. Dadas as matrizes A (de i linhas por j colunas) e B (de k linhas por m colunas), onde $j = k$, obter a matriz C, multiplicação de A por B, de i linhas por m colunas, a partir de $C[i][m] = \sum_{x=1}^{j} A[i][x] \times B[x][m]$

Para implementá-lo, tem-se o

```
real C [i][m] função MM (real A[i][j], B[k][m]) {j tem que ser igual a k}
inteiro AUX
inteiro IND1, IND2, IND3
para IND1 = 1 até i faça
   para IND2 = 1 até m faça
      AUX ← 0
      para IND3 = 1 até j faça
         AUX ← AUX + A[IND1][IND3] × B[IND3][IND2]
      fimpara
      C[IND1][IND2] ← AUX
   fimpara
fimpara
devolva C
fim{função}
```

Solução de sistema de equações lineares Seja A a matriz de coeficientes e seja B a matriz coluna de termos independentes em um sistema $Ax + B = 0$. O parâmetro do algoritmo é uma matriz de n linhas e $n + 1$ colunas, onde a matriz coluna B ocupa a $(n + 1)$-ésima coluna de A.

```
real R[n] função RSEL (real A[n][n+1])
real OLA
inteiro I,J,K
I ← 1
enquanto I < n faça
   J ← n
   enquanto J > I faça
      se A[J-1][I] = 0 então
```

```
9:          troque as linhas A[J] e A[J-1]
10:       fimse
11:       se A[J-1][I] = 0 então
12:         OLA ← 0
13:       senão
14:         OLA ← −A[J][I] ÷ A[J-1][I]
15:       fimse
16:       para K de 1 até n+1 faça
17:         A[J][K] ← A[J][K]+(A[J-1][K]× OLA)
18:       fimpara
19:       J ← J - 1
20:     fimenquanto
21:     I ← I + 1
22: fimenquanto
23: I ← n
24: enquanto I≥ 1 faça
25:     R[I] ← A[I][I+1] ÷ A[I][I]
26:     para K = 1 até n faça
27:       A[K][I] ← A[K][I+1]-(A[K][I]× R[I])
28:     fimpara
29:     I ← I - 1
30: fimenquanto
31: devolva R
32: fim{função}
```

Este algoritmo, pela sua complexidade vale alguma explicação. Seja o seguinte sistema:

——— A solução de um sistema de equações lineares, do tipo

$$\begin{array}{lllllllll} c_{11}.x_1 & + & c_{12}.x_2 & + & c_{13}.x_3 & + & ... & = & t_1 \\ c_{21}.x_1 & + & c_{22}.x_2 & + & c_{23}.x_3 & + & ... & = & t_2 \\ c_{31}.x_1 & + & c_{32}.x_2 & + & c_{33}.x_3 & + & ... & = & t_3 \\ ... & & & & & & & & \\ c_{n1}.x_1 & + & c_{n2}.x_2 & + & c_{n3}.x_3 & + & ... & = & t_n \end{array}$$

Passa pela estratégia de diagonalização, como segue. Seja

$$\begin{array}{lllllll} a.x & + & b.y & + & c.z & = & M \\ d.x & + & e.y & + & f.z & = & N \\ g.x & + & h.y & + & i.z & = & P \end{array}$$

O objetivo inicial é zerar g, e para isso, a nova linha (3) passará a ser a linha (3) somada com a linha (2) esta devidamente multiplicada por $\frac{-g}{d}$ e fica

$$\begin{array}{lllllll} a.x & + & b.y & + & c.z & = & M \\ d.x & + & e.y & + & f.z & = & N \\ 0.x & + & h'.y & + & i'.z & = & P' \end{array}$$

Depois, o objetivo é zerar d e para isso a nova linha (2) passa a ser a linha (2) somada com a linha (1) esta devidamente multiplicada por $\frac{-d}{a}$ e fica

$$\begin{array}{lllllll} a.x & + & b.y & + & c.z & = & M \\ 0.x & + & e'.y & + & f'.z & = & N' \\ 0.x & + & h'.y & + & i'.z & = & P' \end{array}$$

Finalmente, zera-se h' e para isso a linha (3) passa a ser a linha (3) somada com a linha (2) esta devidamente multiplicada por $\frac{-h}{e'}$ e fica

$$\begin{array}{lllllll} a.x & + & b.y & + & c.z & = & M \\ 0.x & + & e'.y & + & f'.z & = & N' \\ 0.x & + & 0.y & + & i''.z & = & P'' \end{array}$$

Note que neste ponto, z pode ser calculado, fazendo-se $z = \frac{P''}{i''}$.

Conhecido z, a coluna 3 da matriz pode ser toda calculada e jogada para a coluna do termo independente. Ao fazer isto, passa-se a ter um sistema de 2 equações e 2 incógnitas, x e y.

Agora o processo se repete até o final. Vamos ver um exemplo disto. Seja o sistema

2x + 3y + 1z = 11
5x + -2y + 3z = 10
2x + y + -z = 1

Ao se zerar a primeira coluna

2x + 3y + 1z = 11
+ -9.5y + 0.5z = -17.5
+ 1.8y + -2.2z = -3

Zerando a segunda coluna, fica:

2x + 3y + 1z = 11
-9.5y + 0.5z = -17.5
-2.11z = -6.33

Com isto, recupera-se o valor $z = 3$, que aplicado no sistema,

2x + 3y = 8
-9.5y = -19

E daqui, sai $y = 2$. Aplicando-o fica 2x = 2 E finalmente $x = 1$.

40.1 Exercício 1

Resolva (a mão ou via computador, você decide ;-) o seguinte sistema de 8 equações lineares a 8 incógnitas:

-1	2	1	2	-1	1	1	2	47
4	1	1	-5	3	-1	1	1	26
2	2	-2	3	-1	-1	2	1	39
5	-5	1	1	2	1	2	2	67
3	2	-3	2	1	3	2	-1	31
2	-2	1	2	2	3	1	3	78
1	-1	5	4	2	-2	2	-2	88
-2	3	2	3	3	4	-4	-3	34

E, informe o valor das 8 variáveis encontradas:

x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8

40.2 Exercício 2

Resolva (a mão ou via computador, você decide ;-) o seguinte sistema de 8 equações lineares a 8 incógnitas:

2	1	1	2	2	2	1	-3	41
2	1	3	4	6	1	-1	1	104
2	1	2	2	3	-3	2	1	43
5	2	1	-2	-2	2	1	3	38
3	2	-4	-1	4	4	5	-1	51
1	-1	2	1	3	1	2	4	89
-2	-3	5	3	2	-2	1	1	60
-1	2	2	1	-4	4	-2	-2	8

E, informe o valor das 8 variáveis encontradas:

x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8

40.3 Exercício 3

Resolva (a mão ou via computador, você decide ;-) o seguinte sistema de 8 equações lineares a 8 incógnitas:

1	2	1	1	2	3	-1	3	65
3	2	4	3	1	1	2	2	100
1	1	3	1	-2	-3	1	1	3
5	-4	1	-1	1	1	2	3	68
2	2	-3	1	2	1	6	-1	75
2	-2	-1	1	3	3	2	2	81
1	-3	1	1	1	-1	2	-1	24
-2	3	3	2	2	2	3	-1	84

E, informe o valor das 8 variáveis encontradas:

x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8

40.4 Exercício 4

Resolva (a mão ou via computador, você decide ;-) o seguinte sistema de 8 equações lineares a 8 incógnitas:

2	1	2	1	1	3	3	2	75
2	3	4	-5	1	2	-1	1	43
2	2	3	3	2	3	2	3	107
3	-3	2	-1	2	2	2	1	15
-2	3	-2	-2	3	5	5	-2	35
1	2	1	1	2	1	-1	3	52
-1	1	3	2	2	2	2	-1	64
-2	1	3	2	1	3	-2	1	42

E, informe o valor das 8 variáveis encontradas:

x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8

40.5 Respostas

1	7	5	10	9	6	2	4	8
2	1	2	10	3	8	9	5	6
3	4	2	7	3	9	8	10	5
4	2	10	9	6	3	1	7	5

Capítulo 41

Exercícios práticos: 135 - O cubo RUBIK

Criado por Erno Rubik em 1974, tem um mecanismo simples que surpreende tanto do ponto de vista mecânico – ao estudar seu interior, como pela complexidade das combinações que se conseguem ao girar suas faces.

O invento, descendente de um protótipo que tinha apenas duas faces é um tipo de quebracabeças que consiste de um cubo em que cada uma das 6 faces está dividida em 9 peças. Elas se articulam graças ao mecanismo da peça interior central que fica oculta dentro do cubo. O resto das peças são visíveis e pode-se observar 3 tipos de peças que não perdem sua condição, a despeito de qualquer movimento que o cubo sofra. São elas:

Central As 6 peças centrais definem a cor da face. Mantém sempre a orientação relativa entre elas. No modelo original o branco se opunha ao amarelo, o vermelho ao laranja e o verde ao azul.

Aresta Peças formadas por 2 cores, em número de 12.

Vértices Peças dos cantos do cubo, em número de 8 e formadas por 3 cores.

Representação

Para representar em 2D um cubo (que é 3D), vai-se fazer o seguinte mapeamento:

1. As 6 cores serão representadas como `123456`.
2. Cada instância de cubro vai ser mostrado como

```
    123                 SSS
    456                 SSS
    789                 SSS
abc ABC jkl JKL      EEE FFF DDD PPP
def DEF mno MNO  ou  EEE FFF DDD PPP
ghi GHI pqr PQR      EEE FFF DDD PPP
    stu                 III
    vxy                 III
    z()                 III
```

   onde a face marcada com `1` corresponde à aresta superior do cubo. A `2` é a face esquerda, a `3` é a frontal, a `4` é a direita e a `5` é a face posterior. Finalmente, a face `6` é a face inferior do cubo.

3. Note que as 6 peças centrais das 6 faces são fixas e não mudam de orientação entre si (umas em relação à outras).

4. Finalmente, para economia de papel, a configuração acima vai ser apresentada como `123456789abcdefghiABCDEFGHI jklmnopqrJKLMNOPQRstuvxyz()`

Operadores

Na tentativa de escrever um programa de computador que resolva o cubo de Rubik, usa-se em geral o algoritmo A* (Hart, 68 e 72). Uma parte importante do algoritmo é a lista de operadores que se aplicam a um estado qualquer.

Para os operadores que virão a seguir, considere o estado inicial como

```
    123
    456
    789
abc ABC jkl JKL
def DEF mno MNO
ghi GHI pqr PQR
    stu
    vxy
    z()
```

Podem-se definir os seguintes operadores:

nome	como ficou
1=cubo vertical anti-horário	369 258 147 JKL abc ABC jkl MNO def DEF mno PQR ghi GHI pqr zvs (xt)yu
2=cubo frontal anti-horário	lor knq jmp 369 CFI uy) PMJ 258 BEH tx(QNK 147 ADG svz ROL cfi beh adg
3=cubo lateral horário anti-horário (observador à esquerda)	ABC DEF GHI cfi stu pmj 987 beh vxy qnk 654 adg z() rol 321 RQP ONM LKJ
4=face superior anti-horário	369 258 147 JKL abc ABC jkl def DEF mno MNO ghi GHI pqr PQR stu vxy z()
5=face frontal anti-horário	123 456 jmp ab9 CFI ukl JKL de8 BEH tno MNO gh7 ADG sqr PQR cfi vxy z()
6=face lateralD anti-horário (observador à esquerda)	12C 45F 78I abc ABu pmj 9KL def DEy qnk 6NO ghi GH) rol 3QR stP vxM z(J
7=face superior horário	741 852 963 ABC jkl JKL abc def DEF mno MNO ghi GHI pqr PQR stu vxy z()
8=face frontal horário	123 456 ifc abs GDA 7kl JKL det HEB 8no MNO ghu IFC 9qr PQR pmj vxy z()
9=face lateral direita horário (observador à esquerda)	12P 45M 78J abc AB3 lor)KL def DE6 knq yNO ghi GH9 jmp uQR stC vxF z(I

Observação importante: Note que quando ocorrem giros na face direita, a posição do operador SEMPRE é a esquerda do cubo. Se tiver dúvidas sobre este operador, olhe as tabelas acima nos casos 3, 6 e 9.

41.1 Exercício 1

1. Aplique o operador **8** ao cubo

```
    )H1
    yEB
    ubC
rqp I4j 98L atP
onm F5f Deh vxM
lkA c2i Gdg z(J
    7Ks
    6NO
    3QR
```

e obterá como resposta um cubo que terá, na face **frontal** na posição L= **2** e C= **3** o valor [ ]

2. Aplique o operador **6** ao cubo

```
    lO7
    852
    Lmg
3Ba 1FR zKc AdJ
4no MN6 kef DEb
jqr PQp uhi GH9
    )(I
    yxv
    Cts
```

e obterá como resposta um cubo que terá, na face **direita** na posição L= **2** e C= **3** o valor [ ]

3. Quais (2) operadores devem ser usados no cubo

```
    36c
    25F
    14I
JKL abu pmA 7kl
def DEy qnB 8NO
ghi GH) roC 9QR
    stP
    vxM
    z(j
```

a fim de resolvê-lo ?

[|]

Exercício 2

1. Aplique o operador **8** ao cubo

```
    IHA
    KNd
    g43
```

```
p2R zbl JO7 ctu
knQ (xo Mef D56
jmP )y1 ahi G89
    rqL
    FEv
    CBs
```

e obterá como resposta um cubo que terá, na face **superior** na posição L= **1** e C= **3** o valor ☐

2. Aplique o operador **9** ao cubo

```
    741
    heB
    ro3
Av) PMl J8L abc
t5( QNF mx6 kEH
s2z ROI py9 jDG
    gdu
    Knq
    ifC
```

e obterá como resposta um cubo que terá, na face **superior** na posição L= **3** e C= **1** o valor ☐

3. Quais (2) operadores devem ser usados no cubo

```
    741
    852
    IFC
ABu pmj 9KL abc
def DEy qnk 6NO
ghi GH) rol 3QR
    stP
    vxM
    z(J
```

a fim de resolvê-lo ?

41.2 Exercício 3

1. Aplique o operador **2** ao cubo

```
    AB3
    DEh
    GHg
cfi stR zvl J87
y54 beQ (xo Mnq
u21 adC 96r Pmp
    LOj
    KNk
    IF)
```


e obterá como resposta um cubo que terá, na face **esquerdo** na posição L= **1** e C= **3** o valor

2. Aplique o operador **4** ao cubo

```
    cfl
    yEo
    uBr
7qp I8) PMJ 3DA
F5O deH tx( Qnm
C2R ghG sba Lkj
    zvi
    KN4
    961
```

e obterá como resposta um cubo que terá, na face **superior** na posição L= **2** e C= **3** o valor

3. Quais (2) operadores devem ser usados no cubo

```
    987
    654
    321
jkl JKL abc ABC
def DEF mno MNO
ghi GHI pqr PQR
    stu
    vxy
    z()
```

a fim de resolvê-lo ?

41.3 Exercício 4

1. Aplique o operador **4** ao cubo

```
    pms
    bEt
    aBC
u41 L8j 9HG iFI
M52 KeD fx( Qno
P63 JdA cvz Rqr
    lO7
    kNh
    )yg
```

e obterá como resposta um cubo que terá, na face **direita** na posição L= **1** e C= **1** o valor

2. Aplique o operador **6** ao cubo

```
    uy)
    tx(
    RFC
IHz gmj 9QP rqp
ONv hnk 6ED fed
LKs iol 3BA cba
    GMJ
    258
    147
```

e obterá como resposta um cubo que terá, na face **frontal** na posição L= **3** e C= **1** o valor

3. Quais (2) operadores devem ser usados no cubo

```
    12A
    45B
    ifC
abs GDj 987 cKL
det HEy qnk 6NO
ghu IF) rol 3QR
    pmP
    vxM
    z(J
```

a fim de resolvê-lo ?

Respostas

```
1 4 K 9 7
2 A r 4 9
3 g B 7 7
4 L i 9 5
```


Capítulo 42

Exercícios práticos: 139 - Ostras

42.1 A fazenda de ostras de Zing Zhu

Zing Zhu possue uma ilha que vem a ser um pedaço de terra chata. Todos os dias, quando a maré sobe a ilha é inundada pela água do mar. Depois de muito pensar e também pedir conselhos a alguns membros de sua família, Zing Zhu decidiu estabelecer uma fazenda de ostras na ilha. Ele usará um sofisticado sistema de cercas de plástico, à prova d'água para controlar as áreas que deverão ser inundadas e as áreas que deverão ficar secas durante a subida da maré. As cercas usadas pelo Zing serão verticais ou horizontais e vêm em tiras que tem diversas alturas e comprimentos. Duas cercas podem se interceptar em um ponto não necessariamente no seu final ou começo.

Você foi contactado pelo Zing para calcular, dada a altura que a maré vai alcançar e a posição e altura de todas as cercas, a área total de terra que NÃO vai ser inudada durante a maré alta. Você pode assumir que a largura das cercas é muito fina quando comparada aos tamanhos de terra envolvida e para o propósito do cálculo da área total, as cercas podem ser consideradas como tendo largura zero.

A entrada do algoritmo contém diversas linhas: a primeira linha de um caso de teste contém um inteiro N indicando o número de cercas na ilha ($1 \leq \mathrm{N} \leq 20$). Cada uma das N próximas linhas contém 5 inteiros: X_1, Y_1, X_2, Y_2 e H representando respectivamente o ponto de início de uma cerca (X_1, Y_1) e o ponto de fim (X_2, Y_2), e a altura da cerca. A última linha da entrada contém um inteiro W que representa a altura da maré. As coordenadas são dadas em metros, as alturas em centímetros. Além disso $X_1 = X_2$ ou $Y_1 = Y_2$ (mas não ambos). $-500 \leq X_1, Y_1, X_2, Y_2 \leq 500$; e $1 \leq W$, $H \leq 1000$. O fim dos dados é indicado por $N = 0$

A saída Para cada conjunto de dados de entrada, o programa deve devolver a quantidade total de área de ilha que vai permanecer seca (i.é, que não vai ser inundada).

Exemplo Seja a entrada

```
4
-20 20 20 20 200
20 20 20 -20 200
0 0 0 20 100
```

```
-10 0 20 0 200
99
0
```

E a resposta neste caso deverá ser 400 m^2.

42.1.1 Resolvendo...

Este problema apresenta algumas novidades interessantes. A primeira, é ele ter sido extraído da lista de problemas da maratona de programação da ACM de 2004. É uma disputa mundial com fases locais, regionais e a fase final que envolve o planeta. São 6 problemas similares a este com cerca de 5 horas para sua solução em equipes de 4 pessoas mais um técnico.

A segunda, que nem é tão novidade assim, mas que não costuma freqüentar as nossas aulas é o fato de não existir nenhuma estratégia "evidente" para resolver o problema. Embora a vida real aí fora seja exatamente assim, os nossos problemas de sala de aula podem dar a entender que o mundo é certinho, sempre tem solução, a solução é única e 100% correta e assim por diante. Isso não é verdade.

A terceira observação, também importante, é que para problemas complexos nem sempre se acha uma solução 100% correta, e mais do que isso, conhece-se de antemão ONDE o algoritmo achado falha e quais problemas ele deixa de resolver. Age-se assim, quando na falta de solução melhor, usa-se a que se tem, na expectativa de que uma quase-solução é melhor do que nenhuma-solução.

A proposta de solução que vai ser sugerida aqui é a de construir uma nova matriz de dimensões $2 \times L + 1$ e $2 \times C + 1$, onde L e C são o número de linhas e colunas da matriz proposta originalmente pelo problema. Age-se assim para criar um espaço adicional onde representar as cercas (que lembrando, no problema original têm largura 0). Nesta nova matriz, as cercas são dispostas, e a seguir, para cada espaço da matriz é feita uma análise de submersão. Vem em nosso socorro o fato de que as cercas são verticais ou horizontais apenas. Assim, só 4 direções precisam ser examinadas. Para cada ponto examinado, escreve-se em qual altura de maré ele ficará submerso (na verdade, o menor valor dos 4 retornados na pesquisa acima). Esta estratégia pressupõe que após a primeira cerca está o mar (o que nem sempre pode ser correto, aqui a falha deste algoritmo). Fica como sugestão melhorar este aspecto.

Estabelecido este valor, é hora de retornar à matriz original (já sem as cercas), mas com os valores de cota de submersão. Agora, a resposta ao pedido é apenas uma operação de comparação (da cota com a altura da maré) e de contagem de metros quadrados, já que lembrando, todas as dimensões são de números inteiros.

42.1.2 Truques usados na implementação

Para passar das coordenados do mundo ($-500 \leq X_1, Y_1, X_2, Y_2 \leq 500$) para as coordenadas 1..L e 1..C da matriz de trabalho, usam-se as expressões:

ESCADA $\leftarrow 1 + (2 \times MAIORY - MENORY)$
L$\leftarrow 1 + abs((1 + (2 \times (Y - MENORY))) - ESCADA)$
C $\leftarrow 1 + (2 \times (X - MENORX))$

Note o cálculo da $ESCADA$ que serve para inverter a ordem de crescimento das linhas (no mundo cartesiano, as linhas crescem de baixo paa cima e no mundo da programação crescem de cima para baixo.

Feita esta conversão, os dados do problema (as cercas) são transpostas à suas novas coordenadas da matriz. É um mapeamento simples.

Nas linhas (e colunas) ímpares desta nova matriz está o espaço para a colocação das cercas. Nas linhas (e colunas) pares estão representados os metros quadrados da fazenda.

Lançadas as cercas, anda-se nas linhas e colunas pares, olhando para cima, direita, baixo e esquerda, olhando o que (altura) impede a entrada da água. O menor desses quatro valores estabelece quando a água vai chegar.

Toda a fazenda é assim mapeada. Depois, pela retirada das cercas (linhas e colunas ímpares) o que sobra é área útil da fazenda.

42.2 Exercício 1

Na instância a seguir, o problema foi ligeiramente adaptado às características usuais da família VIVO de problemas. O enunciado segue mais ou menos o mesmo.

Suponha então uma fazenda de ostras, cujas cercas têm a seguinte distribuição

```
6
  10    3   10   17  240
   2    9   18    9  400
   4   13   14   13  160
  14   13   14    4  100
  14    4    4    4  100
   4    4    4   13  200
```

O senhor Zing Zhu quer saber qual a área seca quando a maré chegar a

altura maré	área seca
90 cm	m^2
110 cm	m^2
170 cm	m^2
230 cm	m^2

42.3 Exercício 2

Na instância a seguir, o problema foi ligeiramente adaptado às características usuais da família VIVO de problemas. O enunciado segue mais ou menos o mesmo.

Suponha então uma fazenda de ostras, cujas cercas têm a seguinte distribuição

```
6
  10    3   10   18  400
   3   10   19   10  360
   5   13   14   13  200
  14   13   14    5  160
  14    5    5    5   80
   5    5    5   13   80
```

O senhor Zing Zhu quer saber qual a área seca quando a maré chegar a

altura maré	área seca
70 cm	m^2
90 cm	m^2
170 cm	m^2
230 cm	m^2

42.4 Exercício 3

Na instância a seguir, o problema foi ligeiramente adaptado às características usuais da família VIVO de problemas. O enunciado segue mais ou menos o mesmo.

Suponha então uma fazenda de ostras, cujas cercas têm a seguinte distribuição

```
6
  10    2   10   19  380
   3   10   17   10  340
   3   14   13   14  140
  13   14   13    5   80
  13    5    3    5  140
   3    5    3   14  160
```

O senhor Zing Zhu quer saber qual a área seca quando a maré chegar a

altura maré	área seca
70 cm	m^2
90 cm	m^2
150 cm	m^2
230 cm	m^2

42.5 Exercício 4

Na instância a seguir, o problema foi ligeiramente adaptado às características usuais da família VIVO de problemas. O enunciado segue mais ou menos o mesmo.

Suponha então uma fazenda de ostras, cujas cercas têm a seguinte distribuição

```
6
   9    3    9   17  380
   3    9   17    9  280
   3   13   14   13   80
  14   13   14    4  100
  14    4    3    4  180
   3    4    3   13  200
```

O senhor Zing Zhu quer saber qual a área seca quando a maré chegar a

altura maré	área seca
70 cm	m^2
90 cm	m^2
110 cm	m^2
190 cm	m^2

42.6 Respostas

```
1      90  24   0   0
2      72  12   0   0
3      90  63   0   0
4      99  55  30   0
```


Capítulo 43

Exercícios práticos: 144 - romanos

O sistema de numeração romana desenvolveu-se na antiga Roma e utilizou-se em todo o seu império. Neste sistema as cifras escrevem-se com determinadas letras, que representam os números. As letras são sempre maiúsculas, já que no alfabeto romano não existem as minúsculas.

As equivalências dos numerais romanos com o sistema decimal são as seguintes:

Decimal	Romana	Decimal	Romana	Decimal	Romana
1	I	10	X	100	C
5	V	50	L	500	D
1000	M				

No sistema de numeração romano as letras devem situar-se da ordem de maior valor para a de menor valor. Não se devem escrever mais de três I, ou três X, ou três C em qualquer número. Se estas letras se situam à frente de um V, um L, ou um D, respectivamente, subtrai-se o seu valor à cifra das ditas letras.

Os romanos desconheciam o zero, introduzido posteriormente pelos árabes, de forma que não existe nenhuma forma de representação deste valor.

Para cifras elevadas os romanos utilizavam um hífen colocado por cima da letra correspondente. O hífen multiplicava o valor da letra por 1.000. Por exemplo, um "C"com hífen superior correspondia ao valor 100.000 (100×1.000), e um "M"com hífen superior correspondia ao valor 1.000.000 (1.000×1.000). Este método permitia escrever cifras realmente altas.

Apresentam-se vários exemplos de números romanos, com as suas equivalências decimais:

Decimal	Romana	Decimal	Romana	Decimal	Romana
1	I	2	II	3	III
4	IV	5	V	6	VI
7	VII	8	VIII	9	IX
10	X	104	CIV	1444	MCDXLIV

Deve-se notar que cada numeral romano básico é superior a todos os numerais romanos básicos de menor valor e cada numeral romano básico é subordinado a todos os numerais romanos básicos de maior valor.

Uma característica importante é que os números romanos não são únicos. Por exemplo, podemos considerar corretas as seguintes representações de 499: `CDXCIC`, `LDVLIV`, `XCIX`, `VDIV` e finalmente `ID`.

43.0.1 Conversão de romano para arábico

```
1: inteiro função R2A (cadeia NROM[n])
2: cadeia CADAUX [7] ← 'IVXLCDM'
3: inteiro VALAUX[7] ← 1 5 10 50 100 500 1000
4: inteiro SUM, I, K, TAM
5: TAM ← tamanho da cadeia NROM
6: SUM ← 0
7: I ← 1
8: enquanto I < TAM faça
9:    K ← 1
10:   enquanto CADAUX[K] ≠ NROM[I] faça
11:      K++
12:   fimenquanto
13:   se (I+1)≥TAM então
14:      M ← 1
15:      enquanto CADAUX[M] ≠ NROM[I+1] faça
16:         M++
17:      fimenquanto
18:      se VALAUX[K] < VALAUX[M] então
19:         SUM ← SUM + VALAUX[M] - VALAUX[K]
20:         I ← I + 2
21:      senão
22:         SUM ← SUM + VALAUX[K]
23:         I ← I + 1
24:      fimse
25:   senão
26:      SUM ← SUM + VALAUX[K]
27:      I ← I + 1
28:   fimse
29: fimenquanto
30: devolva SUM
31: fim {função}
```

Exercícios de aquecimento Converta para arábico:

XVI	
XXXVII	
XLVI	
XCVI	
CI	
CDXCVI	
CIV	
XXV	
XLV	
MDXI	

43.0.2 Conversão de arábico para romano

```
1: (cadeia RESP[10]) função ARD (inteiro A, cadeia LETRAS[3])
2: cadeia AUX[10]
3: inteiro K
4: se A ≥ 0 então
```

```
5:    se A=0 então
6:       devolva ”
7:    senão se A=1 então
8:       devolva LETRAS[1]
9:    senão se A=2 então
10:      devolva LETRAS[1]+LETRAS[1]
11:   senão se A=3 então
12:      devolva LETRAS[1]+LETRAS[1]+LETRAS[1]
13:   senão se A=4 então
14:      devolva LETRAS[1]+LETRAS[2]
15:   senão se A=5 então
16:      devolva LETRAS[2]
17:   senão se A=6 então
18:      devolva LETRAS[2]+LETRAS[1]
19:   senão se A=7 então
20:      devolva LETRAS[2]+LETRAS[1]+LETRAS[1]
21:   senão se A=8 então
22:      devolva LETRAS[2]+LETRAS[1]+LETRAS[1]+LETRAS[1]
23:   senão se A=9 então
24:      devolva LETRAS[1]+LETRAS[3]
25:   fimse
26: senão
27:   K ← 1
28:   enquanto K < abs(A) faça
29:      AUX[K] ← LETRA[1]
30:      K++
31:   fimenquanto
32:   devolva AUX
33: fimse
34: fim{função}
```

```
1:  (cadeia RESP[10]) função A2R (inteiro V)
2:  inteiro D
3:  cadeia AUX[10]
4:  AUX ← ”
5:  D ← ⌊ V ÷ 1000
6:  V ← V - D × 1000
7:  AUX ← AUX + (-D) ARD ’M’
8:  D ← ⌊ V ÷ 100
9:  V ← V - D × 100
10: AUX ← AUX + D ARD ’CDM’
11: D ← ⌊ V ÷ 10
12: V ← V - D × 10
13: AUX ← AUX + D ARD ’XLC’
14: AUX ← AUX + V ARD ’IVX’
15: devolva AUX
16: fim{função}
```


Exercícios de aquecimento Converta para romano:

38	
44	
35	
104	
496	
107	
665	
388	
1002	
843	

43.1 Exercício 1

Implemente os programas acima e a seguir responda:

Converta para arábico Responda:

CCIX	
CDIX	
MMCCXCV	
DCCXL	
CCCXIII	
MMDCCCXXIX	

Converta para romano Responda:

2404	
382	
187	
85	
466	
2510	

43.2 Exercício 2

Implemente os programas acima e a seguir responda:

Converta para arábico Responda:

CDLXVII	
DCCCXCIII	
MCDLVIII	
CDLXV	
DXIX	
MDCX	

Converta para romano Responda:

3402	
844	
805	
410	
198	
3261	

43.3 Exercício 3

Implemente os programas acima e a seguir responda:

Converta para arábico Responda:

DCCXIX	
XLII	
MMDCCXCIII	
DCCCLXIV	
IV	
MMIX	

Converta para romano Responda:

2091	
41	
97	
1931	
547	
859	

43.4 Respostas

```
1  209, 409,2295, 740, 313,2829,MMCDIV,CCCLXXXII,CLXXXVII,LXXXV,CDLXVI,MMDX
2  467, 893,1458, 465, 519,1610,MMMCDII,DCCCXLIV,DCCCV,CDX,CXCVIII,MMMCCLXI
3  719,  42,2793, 864,   4,2009,MMXCI,XLI,XCVII,MCMXXXI,DXLVII,DCCCLIX
```


Capítulo 44

Exercícios Práticos: 148 - diversos

44.1 Universidade de Pinguinhos / cubos

Estes dois exercícios foram retirados da Maratona de Programação da ACM de 2005, onde apareceram com os nomes de "Curso Universitário" e "cubos coloridos".

Há alguns anos atrás, a Universidade de Pinguinhos introduziu um novo sistema flexível de créditos para os alunos ingressantes de cursos de graduação. No novo sistema os alunos podem escolher as disciplinas que desejam cursar em um semestre com a única restrição de não poderem cursar uma dada disciplina A sem antes terem cursado todas as disciplinas que tiverem sido estabelecidas como pré-requisitos de A. Após alguns semestres o reitor notou que muitos estudantes estavam sendo reprovados em muitas disciplinas, simplesmente porque os estudantes estavam cursando muitas disciplinas por semestre. Alguns estudantes chegavam a se matricular em até quinze disciplinas em um semestre. Sendo muito sábio, este ano o reitor introduziu uma regra adicional limitando o número de disciplinas que cada estudante pode cursar por semestre a um certo valor N. Essa regra adicional, no entanto, fez com que os alunos ficassem muito confusos na hora de escolher as disciplinas a serem cursadas em cada semestre. É aí que você entra na estória. O reitor resolveu disponibilizar um programa de computador para ajudar os alunos a fazerem suas escolhas de disciplinas, e solicitou sua ajuda. Mais precisamente, o reitor quer que o programa sugira as disciplinas a serem cursadas durante o curso da seguinte forma. A cada disciplina é atribuída uma prioridade. Se mais do que N disciplinas podem ser cursadas em um determinado semestre (obedecendo ao sistema pré-requisitos), o programa deve sugerir que o aluno matricule-se nas N disciplinas de maior prioridade. Se N ou menos disciplinas podem ser cursadas em um determinado semestre, o programa deve sugerir que o aluno matricule-se em todas as disciplinas disponíveis. Portanto, dadas a descrição de pré-requisitos para cada disciplina, a prioridade de cada disciplina, e o número máximo de disciplinas por semestre, seu programa deve calcular o número necessário de semestres para concluir o curso, segundo a sugestão do reitor, e a lista de disciplinas que o aluno deve matricular-se em cada semestre.

Entrada A entrada, contém vários casos de teste. Se uma disciplina não tem qualquer pré-requisito ela é denominada básica - caso contrário ela é denominada avançada. A primeira linha de um caso de teste contém dois inteiros $1 \leq N \leq 100$ e $1 \leq M \leq 10$, indicando respectivamente o número de disciplinas avançadas do curso e o número máximo de disciplinas que podem ser cursadas por semestre. Cada uma das N linhas

seguintes tem o formato

`STR0 K STR1 STR2 ... STRK`

onde `STR0` é o nome de uma disciplina avançada, $1 \leq K \leq 15$ é o número de disciplinas que são pré-requisitos de `STR0` e `STR1, STR2, ... STRK` são os nomes das disciplinas que são pré-requisitos de `STR0`. O nome de uma disciplina é uma cadeia com no mínimo um e no máximo sete caracteres alfanuméricos maiúsculos (`'A'-'Z'` e `'0'-'9'`). Note que as disciplinas básicas são aquelas que aparecem apenas como pré-requisito de alguma disciplina avançada. Para concluir o curso, o aluno deve cursar (e passar!) todas as disciplinas básicas e avançadas. A prioridade das disciplinas é determinada pela ordem em que elas aparecem pela primeira vez na entrada: a que aparece primeiro tem maior prioridade, e a que aparece por último tem a menor prioridade. Não há circularidade nos pré-requisitos (ou seja, se a disciplina B tem como pré-requisito a disciplina A então A não tem B como pré-requisito, direta ou indiretamente). O número total de disciplinas é no máximo igual a 200. O final da entrada é indicado por `N = M = 0`.

A entrada deve ser lida da entrada padrão.

Saída Para cada caso de teste da entrada seu programa deve produzir a saída na seguinte forma. A primeira linha deve conter a frase 'Formatura em S semestres', onde S é o número de semestres necessários para concluir o curso segundo a sugestão do reitor. As S linhas seguintes (levem conter a descrição das disciplinas a serem cursadas em cada semestre, um semestre por linha, no formato mostrado no exemplo de saída abaixo. Para cada semestre, a lista das disciplinas deve ser dada em **ordem lexicográfica**.

Definição: considere as cadeias de caracteres $S_a = a_l a_2 \ldots a_m$ e $S_b = b_l b_2 \ldots b_n$. S_a precede S_b em ordem lexicográfica se e apenas se S_b é não-vazia e uma das seguintes condições é verdadeira:

- S_a é uma cadeia vazia;
- $a_1 < b_1$ na ordem $'O' <' l' <' 2' < \ldots <' 9' <' A' <' B' < \ldots <' Z'$;
- $a_1 = b_1$ e a cadeia $a_2 a_3 \ldots a_m$ precede a cadeia $b_2 b_3 \ldots b_n$.

A saída deve ser escrita na saída padrão.

Exemplo

```
Exemplo de entrada            Saída para o exemplo

2 2                           Formatura em 4 semestres
B02 3 A01 A02 A03             Semestre 1 : A01 A02
C01 2 B02 B01                 Semestre 2 : A03 B01
                              Semestre 3 : B02
                              Semestre 4 : C01
3 2                          Formatura em 4 semestres
ARTE2 1 ARTE1                 Semestre 1 : ARTE1 MAT1
PROG3 1 PROG2                 Semestre 2 : ARTE2 PROG1
PROG2 2 MAT1 PROG1            Semestre 3 : PROG2
0 0                           Semestre 4 : PROG3
```

44.2 Exercício 1

1. Seja uma instância composta de

```
    6    4
MEDI 4 ENGM ENFE GEOL ENGA
ENGC 2 DISC CALE
CALC 2 CARI COBO
GEOA 3 LOGI CALE CARI
PORT 4 ENGM LATI GEOM RETO
INGL 3 CALE ENGE LATI
0 0
```

Resolva-a e informe:

semestres	materia 3 do semestre 3	materia 3 do semestre 2

2. Seja uma instância composta de

```
    6    4
FORT 2 PORT ENGC
QUIM 4 FRAN MATB ENFE GEOM
ENGE 4 CMAM DISC CALC MATB
MEDI 2 PORT INGL
ENGA 4 BAND GEOL MATB ALEM
RETO 4 BAND ALGO INGL INFO
0 0
```

Resolva-a e informe:

semestres	materia 1 do semestre 5	materia 1 do semestre 3

44.2.1 Cubos Coloridos

Crianças adoram brincar com pequenos cubos. Elas passam horas criando 'casas', 'prédios', etc. O irmãozinho de Tomaz acabou de ganhar um conjunto de blocos coloridos no seu aniversário. Cada face de cada cubo é de uma cor. Como Tomaz é uma criança muito analítica, ele decidiu descobrir quantos "tipos"diferentes de cubos o seu irmãozinho ganhou. Você pode ajudá-lo? Dois cubos são considerados do mesmo tipo se for possível rotacionar um deles de forma que as cores nas faces respectivas dos dois cubos sejam iguais.

Entrada

A entrada contém vários casos de teste. A primeira linha do caso de teste contém um inteiro N especificando o número de cubos no conjunto ($1 < N < 1000$). As próximas $3 \times N$ linhas descrevem os cubos do conjunto. Na descrição as cores serão identificadas pelos números de O a 9. A descrição de cada cubo será dada em três linhas mostrando as cores das seis faces do cubo "aberto", no formato dado no exemplo abaixo. No exemplo abaixo, as faces do cubo têm cores de 1 a 6, a face com cor 1 está no lado oposto da face com a cor 3, e a, face com cor 2 é vizinha das faces 1, 3, 4 e 6, e está no lado oposto da face com cor 5.

```
1
2 4 5 6
3
```


O final da entrada é indicado por N = 0.

A entrada deve ser lida da entrada padrão.

Saída

Para cada caso de teste seu programa deve imprimir uma linha contendo um único inteiro, correspondendo ao número de tipos de cubos no conjunto dado.

A saída deve ser escrita na saída padrão.

Exemplos

```
3
0          0          3
0 7 2 3    1 2 3 7    0 0 2 1 , resposta = 2
1          0          7
```

44.3 Exercício 1a

```
4
5                  1                  9                  4
9 1 4 9            4 4 9 5            4 4 9 5            4 5 1 9
4                  9                  1                  9
```

Quantos cubos diferentes ?

44.4 Exercício 2

1. Seja uma instância composta de

```
    6    3
CULI 4 CALE RETO INFO ENGA
ENGE 4 ALGO COBO FRAN QUIM
ENFE 3 BAND DISC COBO
GEOA 2 COBO GEOL
GEOM 2 PORT CMAM
ALEM 4 FISI DBAS MEDI CALE
0 0
```

Resolva-a e informe:

semestres	materia 3 do semestre 4	materia 2 do semestre 3

2. Seja uma instância composta de

```
    6    3
GEOL 2 PORT CALE
ESDA 2 ENGC CALC
LOGI 3 FRAN INFO INGL
DISC 4 PORT LATI MEDI CARI
ENGE 4 FRAN COBO LATI PORT
```

```
ALGO 3 INGL CMAM ENFE
0 0
```

Resolva-a e informe:

semestres	materia 1 do semestre 5	materia 3 do semestre 3

44.5 Exercício 2a

```
4
2                   3                   2                   8
4 8 8 3             8 4 8 2             4 4 8 3             4 3 4 8
4                   4                   8                   2
```

Quantos cubos diferentes ?

44.6 Exercício 3

1. Seja uma instância composta de

```
    6    3
GEOA 4 CALE INFO QUIM LOGI
ALGO 3 CARI BAND ENGA
LATI 4 COBO BAND CMAM GEOM
ESDA 2 ENGE INGL
MEDI 3 DBAS FRAN CMAM
CALC 2 FRAN COBO
0 0
```

Resolva-a e informe:

semestres	materia 1 do semestre 3	materia 2 do semestre 6

2. Seja uma instância composta de

```
    6    4
CALE 3 CARI QUIM ENFE
INFO 2 QUIM ENGE
PORT 4 CULI INGL BAND ENGM
GEOA 2 ALGO GEOL
CALC 2 DBAS ESDA
RETO 4 FRAN GEOM MATB ENGC
0 0
```


Resolva-a e informe:

semestres	materia 2 do semestre 4	materia 1 do semestre 7

44.7 Exercício 3a

```
4
9                   9                   9                   1
9 7 9 6             6 9 7 1             9 1 9 7             9 7 9 6
1                   9                   6                   9
```

Quantos cubos diferentes ?

44.8 Respostas

```
1   6 LATI DISC    6 ALEM CALC
1a  3 13
2   8 ENGE FRAN    7 CARI INGL
2a  4
3   7 COBO ESDA    7 ESDA RETO
3a  2 12 14 24
```

Capítulo 45

Exercícios Práticos: 151 - diversos

Estes exercícios foram retirados da Maratona de Programação da ACM de 2005 e 2004.

45.1 A Piscina

O Centro Comunitário decidiu construir uma nova piscina, em tempo para o verão do ano que vem. A nova piscina será retangular, com dimensões X por Y e com profundidade Z. A piscina será recoberta com um novo tipo de azulejo cerâmico de alta tecnologia que é produzido em três tamanhos distintos: 5×5, 15×15 e 30×30 (em centímetros). Cada azulejo desses tamanhos custa 2 centavos, 15 centavos e 50 centavos respectivamente. Os azulejos são de alta qualidade, feitos de um material que não pode ser cortado (ou seja, os azulejos devem ser usados inteiros).

A Única loja local que vende esse tipo de azulejo tem em estoque uma certa quantidade de azulejos de cada tamanho. Você deve escrever um programa que determine se o estoque de azulejos disponível na loja é suficiente para azulejar toda a piscina. Se o estoque for suficiente, seu programa deve determinar também o número de azulejos de cada tamanho que são necessários para que o custo de azulejar a piscina seja o menor possível.

Os azulejos devem ser usados para recobrir completamente toda a superfície da piscina sem deixar qualquer espaço sem azulejos e sem deixar sobras de azulejos transpassando as bordas da piscina.

Entrada A entrada contém vários casos de teste. Cada caso de teste é composto por duas linhas. A primeira linha contém três números reais X, Y e Z, representando as dimensões e a profundidade da piscina, em metros, com precisão de uma casa decimal ($0 < X, Y \leq 50.0$ e $0 < Z \leq 2.0$). A segunda linha contém três números inteiros P, M e G, representando a quantidade disponível de azulejos de tamanho pequeno, médio e grande ($0 \leq P, M, G \leq 2000000$), respectivamente. O final da entrada é indicado por $X = Y = Z = 0$.

Saída Para cada caso de teste da entrada seu programa deve produzir uma linha de saída. Se é possível recobrir completamente a piscina com o estoque disponível, imprima uma linha com três inteiros descrevendo respectivamente as quantidades de azulejos pequenos, médios e grandes para recobrir toda a piscina com o menor custo

possível. Caso contrário, imprima uma linha contendo a palavra `impossivel` (note a ausência de acentuação).

Alguns exemplos Para as entradas

```
3.0 4.0 1.0
1000 1000 1000
```

a resposta deverá ser

```
752 0 268
```

```
3.0 3.0 0.9
300 300 300
```

a resposta deverá ser

```
0 0 220
```

```
12.5 12.5 1.6
5000 0 3000
```

a resposta deverá ser

```
4464 0 2501
```

```
3.0 3.0 1.0
300 300 300
0 0 0
```

a resposta deverá ser

```
impossivel
```

45.2 Exercício 1

Seja a entrada

```
   5.0   5.9   2.0
  1200   700   800
   5.0   6.0   2.4
  1300   700   400
   4.1   6.1   1.5
   900   700   500
0 0 0
```

Resolva-as e informe:

inst	az. peq	az. médio	az. grande
1			
2			
3			

45.3 Mágico

Um mágico inventou um novo truque de cartas e apresentou-o na prestigiosa American Conference of Magicians (ACM). O truque é muito interessante e ele recebeu o prêmio "Best Magic Award"na conferência. O truque requer 3 participantes: o mágico, um expectador e um assistente. Durante o truque o expectador deve embaralhar um conjunto de 52 cartas (baralho completo sem os coringas) e depois escolher 5 cartas aleatórias. Elas são passadas ao assistente, sem que o mágico as veja. Após mostrar 4 dessas cartas ao mágico, ele magicamente adivinha a quinta carta.

O truque funciona pelo fato do assistente, depois de olhar as 5 cartas, escolher quais quatro serão mostradas, e ele as escolhe de modo a codificar a informação para o mágico sobre a quinta carta. O código é baseado em uma dada ordenação do baralho. As cartas são ordenadas **primeiro por naipe** e depois pelo seu valor de face.

Usar-se-á a seguinte ordem

- $\diamondsuit < \spadesuit < \heartsuit < \clubsuit$ (ouros, espadas, copas e paus); e
- $A < 2 < ... < 9 < T < J < Q < K$, onde A=Ás, T=10, J=valete, Q=dama e K=rei.

Assuma que o expectador escolheu as cartas J$\heartsuit$, 8$\clubsuit$, 7$\diamondsuit$, 8$\spadesuit$ e Q$\diamondsuit$ (valete de copas, 8 de paus, 7 de ouros, 8 de espadas e dama de ouros). A estratégia do assistente é a seguinte:

- Ache o naipe n que aparece ao menos duas vezes no conjunto de cartas escolhidas (ouros, no exemplo). Se mais do que um naipe aparecem duas vezes, escolha o de menor ordem.
- esconda a carta x, do naipe n que esteja a no máximo 6 posições **à direita** na tabela cíclica $A < 2 < ... < T < J < Q < K < A < 2 < ...$ de outra carta y do mesmo naipe. Isto será sempre possível, já que existem apenas 13 cartas do mesmo naipe. (No exemplo, o assistente esconde a dama de ouros). Se duas ou mais cartas satisfazem o critério acima, escolha a que tem o **menor** valor de face.
- mostre y ao mágico. Neste ponto ele saberá o naipe da carta escondida (o mesmo de y). Saberá também que o valor da carta escondida não está a mais de 6 posições à direita de y.
- com as 3 cartas sobrantes, o assistente codifica um número entre 1 e 6. Para fazer isto ele ordena as três cartas (digamos z_1, z_2 e z_3) em ordem, onde $z_1 < z_2 < z_3$. Cada uma das 6 ordens possíveis em que as cartas são mostradas deve ser interpretada pelo mágico como sendo um número:
  - z_1, z_2, z_3 significa 1,
  - z_1, z_3, z_2 significa 2,
  - z_2, z_1, z_3 significa 3,
  - z_2, z_3, z_1 significa 4,
  - z_3, z_1, z_2 significa 5,
  - z_3, z_2, z_1 significa 6.

Desta maneira, mostrando as 4 cartas, em uma determinada ordem, ele informa ao mágico qual é a quinta carta.

Sua tarefa é desenvolver um programa que receba as 4 cartas mostradas pelo assistente e informe ao mágico qual a quinta carta.

Entrada A entrada contém diversas instâncias. A primeira linha contem N, onde N é a quantidade de instâncias que virão a seguir ($1 \leq N \leq 10000$). Cada instância é composta por uma única linha na qual estão 4 cartas separadas por brancos. Cada carta tem 1 caracteres para seu valor e um caracter para o seu naipe, como visto acima. As cartas vem na ordem em que são apresentadas pelo assistente.

Saída A saída deve ser composta pela quinta carta.

Exemplos Para a entrada
2
7♢, 8♣, 8♠, J♡
T♠, 2♡, A♣, 5♢

o programa deverá responder $Q\diamondsuit$ e $A\spadesuit$, respectivamente.

45.4 Exercício 1a

Seja a entrada
3
2♡ K♠ A♢ 9♣
Q♢ 7♠ 9♠ 2♡
4♠ 4♡ 6♡ 4♢
Resolva-a e informe:

inst1	inst2	inst3

45.5 Exercício 2

Seja a entrada

```
   5.0   6.5   1.7
   900   800   300
   4.2   6.4   2.3
  1000   600   900
   4.5   5.9   2.4
  1200   900  1000
0 0 0
```

Resolva-as e informe:

inst	az. peq	az. médio	az. grande
1			
2			
3			

45.6 Exercício 2a

Seja a entrada

3
5♢ Q♡ Q♣ 7♠
5♣ 6♠ Q♡ K♢
8♢ 4♡ T♣ K♠
Resolva-a e informe:

inst1	inst2	inst3

45.7 Exercício 3

Seja a entrada

```
   5.0   6.1   1.9
  1100   500   800
   4.5   5.9   1.9
  1400   700   900
   4.9   5.9   2.0
  1100   800  1000
0 0 0
```

Resolva-as e informe:

inst	az. peq	az. médio	az. grande
1			
2			
3			

45.8 Exercício 3a

Seja a entrada

3
T♣ 8♢ 5♠ Q♡
T♠ 4♢ T♣ K♡
J♢ 6♠ 7♠ A♣
Resolva-a e informe:

inst1	inst2	inst3

45.9 Respostas

```
1       809  263  724        -1    0    0       464  420  500
1a 6O 2E JP
2        -1    0    0       772  140  784       186   62  829
2a 9E 7E JC
3        -1    0    0       994   54  693       998  198  724
3a QE 3O 3C
```

Capítulo 46

Exercícios Práticos: 152 - Regata de cientistas e Luzes da Festa

46.1 Exercício 1

Este exercício foi retirado da Maratona de Programação da ACM de 2005.

46.2 Regata de cientistas

Todos os anos desde 1996, cientistas da computação do mundo todo se encontram para a famosa Regata dos Cientistas. A competição consiste em uma corrida de barcos com obstáculos pelo oceano, onde o objetivo de cada equipe é, partindo de um ponto em comum, alcançar o ponto de chegada sem que nenhum obstáculo seja tocado ou transpassado. Uma equipe que toca ou transpassa um obstáculo é automaticamente desclassificada. A equipe vencedora é aquela que primeiro atinge o ponto de chegada (o ponto de chegada é distinto do ponto de início). Você foi contratado pela equipe brasileira para desenvolver um programa que calcule o comprimento da menor rota válida possível do ponto de partida ao ponto de chegada. O oceano é considerado um plano infinito, onde cada obstáculo é localizado em uma posição fixa e representado por um segmento de reta dado pelos seus dois extremos (x_1, y_1) e (x_2, y_2). Os barcos são adimensionais, representados por um ponto no plano e os obstáculos possuem espessura desprezível. Os obstáculos estão dispostos de tal forma que os mesmos não se interceptam. De forma similar, os pontos de início e chegada da competição não são interceptados por nenhum obstáculo.

Entrada A entrada é composta por vários casos de teste. A primeira linha de um caso de teste contém 5 números inteiros x_i, y_i, x_f, y_f, n, representando respectivamente as coordenadas do ponto de início (x_i, y_i), as coordenadas do ponto de chegada (x_f, y_f) e a quantidade de obstáculos n ($n \leq 150$). Cada uma das n linhas seguintes de um caso de teste corresponde a quatro números inteiros (x_1, y_1, x_2, y_2) que são as coordenadas dos dois extremos de um obstáculo. Considere que as coordenadas x, y de qualquer ponto, satisfazem $-5000 \leq x, y \leq 5000$. O final da entrada é representada por uma linha contendo $x_i = y_i = x_f = y_f = n = 0$.

Saída Para cada caso de teste, imprima uma linha contendo o comprimento da menor rota válida possível, arredondada para duas casas decimais.

Exemplos

```
0 0 10 0 1
5 -1 5 1
0 0 10 0 1
5 0 5 1
0 0 0 0 0
```

As respostas a estes dois casos são `10.20` e `10.00` respectivamente.

46.3 Iluminação da festa

Este exercício foi retirado da Maratona de Programação da IOI de 1998.

Na próxima final da IOI, haverá uma festa de congraçamento, para comemorar as premiações. Nesta festa, ter-se-á uma iluminação feérica que precisa ser controlada. A iluminação será composta por N lâmpadas. No início da festa, todas as lâmpadas começam acesas. Para ajustar o acende-apaga, há 4 botões cujos comportamentos são:

botão 1 Todas as lâmpadas trocam de estado, ou seja as acesas apagam e as apagadas acendem.

botão 2 Trocam de estado as lâmpadas de número ímpar (1, 3, ...)

botão 3 Trocam de estado as lâmpadas de número par (2, 4, ...)

botão 4 Trocam de estado as lâmpadas de cujo número obedece à regra $3K + 1$, com $K \geq 0$, ou seja as lâmpadas 1, 4, 7, ...

Cada vez que um dos botões é pressionado, incrementa-se um contador C, que no início sempre contém o valor 0 (zero). Você deve idealizar um algoritmo que receba informações de entrada conforme descritas baixo e mostre quais as configurações possíveis de lâmpadas acesas e apagadas são possíveis.

Entrada A entrada é composta por 4 linhas, a saber:

1. A primeira linha contém o número N de lâmpadas ($10 \leq N \leq 100$)
2. A segunda, contém o valor final do contador C de botões ($1 \leq C \leq 10000$)
3. A terceira contém uma seqüência de nenhuma, uma ou duas lâmpadas que deverão estar ACESAS ao final. A seqüência termina com -1.
4. A última contém uma seqüência de nenhuma, uma ou duas lâmpadas que deverão estar APAGADAS ao final. A seqüência termina com -1.

Saída A saída deverá ter todas as configurações possíveis de serem alcançadas apertando C vezes algum dos quatro botões e que satisfaçam as restrições de aceso e apagado da entrada.

Por exemplo, supondo a entrada

```
12
3
1 6 -1
2 -1
```

Sugere que haverá 12 lâmpadas, os botões serão pressionados 3 vezes, e ao final as lâmpadas 1 e 6 deverão estar acesas e a lâmpada 2 deverá estar apagada. Naturalmente, as lâmpadas não citadas aqui, poderão estar em qualquer estado.

Outro exemplo, dada a entrada:

```
11
 3
 4    9   -1
 8   -1
```

A resposta será

```
00111000111
```

Que corresponde às aplicações dos botões 1, 3 e 4.

☞ **Para você fazer**

Considere a seguinte entrada, para o problema da regata

```
 -60   -40    80   160     3
-190   120   -70    40
  10    60    60    20
  30   120   230   -20
 -10     0   130   230     3
  20    60   100    10
 100    90   280   -20
 240    80   340    20
   0     0     0     0     0
```

Considere a seguinte entrada para as luzes

```
11
 3
 3    6   -1
 7   10   -1
```

regata 1	regata 2	luzes

46.4 Exercício 2

Regata de cientistas

☞ **Para você fazer**

Considere a seguinte entrada, para o problema da regata

```
-20   -10   200   220     3
-40   110    70    10
130    60   210   -10
230    80   330   -20
 10    10   240   150     3
 30    90    80    20
```



```
130    70   140    40
190    90   240    10
  0     0     0     0     0
```

Considere a seguinte entrada para as luzes

```
13
 3
 5    8    9   -1
13   -1
```

regata 1	regata 2	luzes

46.5 Exercício 3

☞ **Para você fazer**

Considere a seguinte entrada, para o problema da regata

```
 -70   -50    90   190     3
-110    60     0   -10
 -50   110    70    30
 120    80   230    10
 -70    20    90   210     3
 -50    80    40     0
  60    70   140     0
 140    80   260   -20
   0     0     0     0     0
```

Considere a seguinte entrada para as luzes

```
11
 3
 9   -1
 6   10   -1
```

regata 1	regata 2	luzes

46.6 Exercício 4

☞ **Para você fazer**

Considere a seguinte entrada, para o problema da regata

```
-60   -20   170   140     3
-60   100   -10    20
 20   100    60    40
130    60   160    20
 20    10   240   180     3
```

```
 30   100    80    40
150    60   180    30
220    80   290     0
  0     0     0     0     0
```

Considere a seguinte entrada para as luzes

```
10
 3
 3    5   -1
 4    7   -1
```

regata 1	regata 2	luzes

46.7 Respostas

```
1    247.61    295.70    01101101101      1 1 4
2    318.28    276.87    0110110110110    1 1 4
3    288.44    254.30    10101010101      1 1 3
4    280.18    278.03    0110110110       1 1 4
```


Capítulo 47

Exercícios práticos: 153a - Jogo do Retângulo

Exercício extraído da 17. IOI, Polônia 2005.

47.1 Jogo do retângulo

Considere-se um jogo de 2 jogadores. Há um retãngulo $x \times y$, onde x e y são inteiros e positivos. Cada jogador faz um corte no retângulo. e repassa o maior dos dois retângulos para que o adversário faça o seu corte. O corte deve ser único, vertical ou horizontal. Os retângulos resultados devem ambos ter dimensões inteiras. Se o corte dividiu o retângulo em duas metades, uma delas é descartada. O jogador que recebe um retângulo de 1×1, e que portanto não pode mais fazer cortes, perde o jogo. A tarefa é escrever um programa que jogue e ganhe o jogo do retângulo. As dimensões x e y serão inteiros emtre 1 e 100.000.000. Ao menos uma das dimensões é maior do que 1.

Quando o programa começa, ele recebe o retângulo original e deve fazer o primeiro corte. O maior pedaço é preservado e então apresentado ao seu oponente.

Exemplo de interação:

1. O jogo começa com $4, 3$. Você deve jogar algo, digamos *vertical*, 1 e o novo retângulo é $3, 3$
2. Seu oponente joga *horizontal*, 1 e o novo retângulo é $3, 2$.
3. Nova jogada sua, digamos *horizontal*, 1 e fica $3, 1$.
4. O outro joga *vertical*, 1 e fica $2, 1$
5. Você joga *vertical*, 1 e fica $1, 1$, e você acaba de ganhar o jogo

Uma estratégia de solução

Construa-se uma tabela, digamos 15×15, representando possíveis 225 tabuleiros (de dimensão máxima 15) cada. Assinale com um `X` a primeira posição perdedora, que é – de acordo com a definição – a $1, 1$. Marque com uma `o` as posições vencedoras que conduzem à perdedora, no caso as posições $1, 2$ e $2, 1$. Marque a posição $2, 2$ como perdedora, pois ela invariavelmente conduzirá à $1, 2$ ou $2, 1$ e estas são vencedoras.

Prossiga até montar o mapa completamente. Note que como a matriz é absolutamente simétrica em relação à diagonal principal (tanto faz usar a vertical ou a horizontal,

elas são intercambiáveis), basta montar um dos triângulos (acima ou abaixo da diagonal principal).

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1															
2															
3															
4															
5															
6															
7															
8															
9															
10															
11															
12															
13															
14															
15															

Depois de ter preenchido este quadro, verifique o que caracteriza as posições perdedoras. Note que para que m e n sejam um retângulo perdedor, deve existir um inteiro k que torne verdadeira a expressão

$$m + 1 = (n + 1) \times 2^k$$

.

É baseada nesta fórmula, que você deve procurar a jogada a fazer. Use o algoritmo:

```
1: função JOGUE (inteiro M, N): inteiro
2: inteiro LIMM,LIMN,I,J,XX
3: LIMM ← ⌊ M ÷ 2
4: LIMN ← ⌊ N ÷ 2
5: I ← 1
6: enquanto (I ≤ LIMM) faça
7:    XX ← podeparar (M-I, N)
8:    se XX = 1 então
9:       jogue (M-(I+1)),N
10:      caiafora
11:   fimse
12:   I++
13: fimenquanto
14: J ← 1
15: enquanto (J ≤ LIMN) faça
16:   XX ← podeparar (M, N-J)
17:   se XX = 1 então
18:      jogue M, N-(J+1)
19:      caiafora
20:   fimse
21:   J++
22: fimenquanto{casos perdedores}
23: se M=1 ∧ N=1 então
```

```
24:    perdi
25:    caiafora
26: fimse
27: se x < 2 então
28:    jogue M,(N-1) // e seja o que Deus quiser
29: senão
30:    jogue (M-1), N // e seja o que Deus quiser
31: fimse
32: fimfunção
```

```
 1: função podeparar (inteiro: A,B) : inteiro
 2: inteiro VOLTA, MAIOR, MENOR, K
 3: se A ≥ B então
 4:    MAIOR ← A
 5:    MENOR ← B
 6: senão
 7:    MAIOR ← B
 8:    MENOR ← A
 9: fimse
10: K ← 0
11: VOLTA ← 0
12: enquanto 2^K ≤ MAIOR faça
13:    se MAIOR = ((2^K)× MENOR) + (2^K) − 1 então
14:       VOLTA ← 1
15:       K ← ∞
16:    fimse
17:    K++
18: fimenquanto
19: retorne VOLTA
20: fimfunção
```

47.2 Exercício 1

1. **Seguindo os algoritmos acima**, faça o primeiro lance para

   `39,30`

   Trate de ganhar o jogo ! Se for impossível ganhar esta instância, responda "PERDI" no quadro próprio.

2. Idem para

   `18,37`

3. E para

   `38,33`

1	2	3

47.3 Exercício 2

1. **Seguindo os algoritmos acima**, faça o primeiro lance para

   ```
   25,12
   ```

   Trate de ganhar o jogo ! Se for impossível ganhar esta instância, responda "PERDI" no quadro próprio.

2. Idem para

   ```
   31,33
   ```

3. E para

   ```
   25,32
   ```

1	2	3

47.4 Exercício 3

1. **Seguindo os algoritmos acima**, faça o primeiro lance para

   ```
   40,39
   ```

   Trate de ganhar o jogo ! Se for impossível ganhar esta instância, responda "PERDI" no quadro próprio.

2. Idem para

   ```
   35,30
   ```

3. E para

   ```
   37,18
   ```

1	2	3

47.5 Exercício 4

1. **Seguindo os algoritmos acima**, faça o primeiro lance para

   ```
   25,33
   ```

   Trate de ganhar o jogo ! Se for impossível ganhar esta instância, responda "PERDI" no quadro próprio.

2. Idem para

```
31,35
```

3. E para

```
21,10
```

1	2	3

47.6 Respostas

```
1     30 30    0  0   33 33
2      0  0   16 33   25 25
3     39 39   30 30    0  0
4     16 33   17 35    0  0
```


Capítulo 48

Exercícios Práticos: 155a - Caminhos no tabuleiro e Descarga do vulcão

48.1 Exercício 1

Estes exercícios foram baseados em similares pedidos na Olimpíada Espanhola de Informática.

Caminhos

Seja um tabuleiro de dimensões $M \times N$, onde $1 \leq M \leq 20$ e $1 \leq N \leq 20$, tal que cada casa contenha uma letra maiúscula. Aquela casa que está na linha M e na coluna N é identificada pela dupla (m, n).

Duas casas diferentes (m_i, n_i) e (m_j, n_j) são adjacentes se para a primeira componente:

- $|m_i - m_j| \leq 1$ ou então $|m_i - m_j| = M - 1$.

A segunda componente tem expressão semelhante:

- $|n_i - n_j| \leq 1$ ou então $|n_i - n_j| = N - 1$.

Em outras palavras, são adjacentes todas as casas que rodeiam uma dada, imaginando que no tabuleiro, a última fila está unida à primeira, e a mesma coisa para as colunas. No desenho a seguir, marca-se com um asterisco, as casas adjacentes à (2,3), à esquerda e (1,1) à direita, ambas em tabuleiros 4×4.

```
_  *  *  *          _  *  _  *
_  *  _  *          *  *  _  *
_  *  *  *          _  _  _  _
_  _  _  _          *  *  _  *
```

Dada uma palavra A de k letras maiúsculas $A = a_1, a_2, ...a_k, k \geq 1$ diz-se que A está contida no tabuleiro se é verdade que

- existe uma casa (m_1, n_1) que contém a letra a_1
- para cada letra $a_1 + 1, 1 \leq i \leq k$, existe uma casa (m_{i+1}, n_{i+1}) que contém a_{i+1} sendo que (m_i, n_i) e (m_{i+1}, n_{i+1}) são adjacentes no tabuleiro e

A seqüência de casas $(m_1, n_1), ..., (m_k, n_k)$ será chamada de caminho de A no tabuleiro.

Desta maneira, dado o tabuleiro 4×4 a seguir, as cadeias "SOLA", "HOLA"e "ADIOS"estão contidas nele, mas não ocorre o mesmo com "GOZA", "HORA"nem "HALA".

```
S H A Z
I O L G
E Z E F
O H D I
```

No caso de "SOLA", as casas que formam o caminho são $(1,1),(2,2),(2,3)$ e $(3,1)$. Para "HOLA"são $(1,2),(2,2),(2,3),(1,3)$. Para "ADIOS"o caminho é $(1,3),(4,3),(4,4),(4,1),(1,1)$.

Dado um tabuleiro com as características citadas e uma palavra A composta de letras maiúsculas, se pede calcular o caminho de A. Quando construir o programa, pode-se afirmar que A está contida no tabuleiro, e que o caminho A é unico.

Exemplo Seja procurar a palavra PRECISAM no tabuleiro

```
    1   2   3   4   5   6   7   8   9  10  11  12
  +-----------------------------------------------
 1| D   S   T   Q   B   V   W   Y   G   C   Z   Z
 2| Y   Y   Q   Z   L   I   D   C   O   P   C   Y
 3| R   D   H   V   C   A   S   M   X   S   F   Q
 4| S   J   Z   Y   Y   F   E   U   Q   U   J   F
 5| L   X   B   G   Z   X   X   E   R   V   Y   D
 6| F   H   M   Q   Z   F   G   J   I   U   Z   J
 7| F   A   O   S   H   P   K   D   P   E   L   H
 8| S   I   L   V   Y   S   J   L   A   P   C   G
 9| R   C   Q   X   D   Y   P   K   O   A   V   O
10| E   A   N   Y   S   S   Y   F   Q   A   P   J
11| J   W   P   Y   H   Q   D   H   G   W   L   R
12| M   A   E   R   E   X   R   D   V   U   W   P
```

O caminho é (12,12);(11,12);(10,1);(9,2);(8,2);(8,1);(7,2) e (6,3).

Vulcão

Neste caso se deseja simular o deslocamento da lava de um vulcão ao largo da encosta do mesmo. Representar-se-á o vulcão por uma superfície retangular quadriculada. Para cada uma das casas, conhece-se a altura dela em relação ao nível do mar, sendo este número entre 1 e 9.

O comportamento da lava se rege pelas seguintes regras:

1. A lava flue desde a casa na qual está situada a cratera do vulcão
2. Dada uma casa alcançada pela lava, a mesma flue para qualquer um dos 8 vizinhos que tem altura menor que a casa original.
3. A cada instante de tempo, a lava se desloca 1 casa nas 8 direções.
4. No instante de tempo t=1, apenas 1 casa (a cratera) contém lava.

Pede-se a confecção de um mapa das encostas do vulcão que identifique os pontos alcançados pela lava, em todos os instantes de tempo.

Entrada A entrada é uma matriz de 12 linhas, contendo a altura dos 144 platôs das encostas. Informa-se também a coordenada (linha,coluna) da cratera.

Saída A quantidade de platos que a lava atinge, nos diversos instantes de tempo. Exemplo

	1	2	3	4	5	6	7	8	9	10	11	12
1	5	8	1	9	9	4	2	5	3	4	3	9
2	6	9	7	9	7	6	5	2	8	5	9	4
3	2	1	8	3	8	1	5	4	8	5	5	5
4	6	9	4	3	7	8	6	5	9	1	1	7
5	9	5	5	4	4	5	1	2	4	3	2	9
6	2	5	7	1	8	7	2	5	7	2	2	3
7	8	1	4	4	3	7	1	6	8	7	3	5
8	9	9	4	5	5	3	2	1	1	8	5	8
9	7	6	1	3	9	2	8	8	9	8	4	3
10	9	7	7	7	2	6	9	5	1	6	4	7
11	8	4	7	8	8	2	9	7	3	5	9	7
12	9	2	7	8	1	7	8	5	3	2	3	2

Com a cratera disposta em (9,10), gerou os deslocamentos de lava: para t=1, Quantidade de platôs imersos=1; para t=2, Q=7; para t=3, Q=11; para t=4, Q=16 e para t=5, Q=17.

Neste ponto, o programa pára, pois a lava não tem para onde ir.

☞ Para você fazer

Seja a palavra ACIONADAS e o tabuleiro

	1	2	3	4	5	6	7	8	9	10	11	12
1	L	H	J	L	X	R	E	B	F	J	O	Q
2	C	I	F	R	H	X	E	T	U	J	S	Z
3	X	A	N	X	K	K	X	C	B	C	A	I
4	O	A	D	X	E	U	F	G	T	Q	D	N
5	S	C	T	O	W	E	Y	U	Q	B	A	N
6	O	B	P	B	F	O	E	W	G	C	X	Z
7	O	E	D	H	F	K	P	Q	P	L	Q	V
8	C	N	S	Y	F	H	M	B	T	N	O	P
9	O	C	P	M	B	G	Z	V	J	E	H	S
10	G	D	L	C	E	R	L	P	S	N	H	L
11	J	A	F	C	L	M	Y	G	F	P	N	W
12	F	C	T	I	N	S	X	B	E	T	P	O

Pergunta-se: Qual a linha e coluna da sexta letra da palavra ?

Seja o vulcão

	1	2	3	4	5	6	7	8	9	10	11	12
1	2	9	6	7	8	9	4	3	6	9	5	4
2	2	7	4	3	9	8	3	3	8	1	3	2
3	9	7	5	1	5	1	9	9	8	9	7	6
4	4	3	3	6	1	4	6	6	4	2	8	3


```
 5|    7   7   9   3   6   2   7   1   9   6   2   2
 6|    8   5   4   4   9   1   9   9   7   7   7   3
 7|    6   2   4   3   1   4   3   9   1   3   4   1
 8|    5   3   6   3   2   2   6   7   9   4   3   1
 9|    8   5   9   1   5   5   4   7   8   4   7   2
10|    5   1   6   3   8   6   6   6   5   2   7   9
11|    5   3   4   6   3   1   7   3   1   4   5   2
12|    6   2   7   1   1   7   1   5   6   9   9   8
```

com a cratera disposta no plato (linha,coluna): 2 5

Pergunta-se quantos platôs são atingidos nos instantes t=3, t=4 e t=5 ?

caminhos: linha e coluna da sexta letra?	vulcão: em t=3, t=4 e t=5?

48.2 Exercício 2

Caminhos

☞ **Para você fazer**

Seja a palavra COMPLETA e o tabuleiro

```
    1   2   3   4   5   6   7   8   9  10  11  12
  +-----------------------------------------------
 1| N   B   L   X   B   R   R   S   D   V   N   P
 2| L   I   P   R   N   K   I   I   S   G   B   K
 3| N   Q   J   C   Z   V   P   H   Y   J   C   Z
 4| Z   K   U   H   N   O   Z   K   V   A   P   B
 5| D   Q   T   H   B   N   S   H   O   Z   U   X
 6| J   F   Z   F   V   A   H   N   P   O   Q   A
 7| H   A   M   P   Y   O   V   G   X   Z   H   M
 8| L   T   Z   T   X   E   H   Y   Z   L   O   P
 9| E   O   I   O   X   Z   G   N   G   J   P   C
10| K   I   O   G   C   Y   R   F   V   L   E   E
11| H   Q   J   M   W   U   Y   D   Z   C   Y   C
12| M   O   W   A   F   Q   R   I   M   G   E   Y
```

Pergunta-se: Qual a linha e coluna da sexta letra da palavra ?

Seja o vulcão

```
    1   2   3   4   5   6   7   8   9  10  11  12
  +-----------------------------------------------
 1|    2   8   9   5   4   6   9   4   4   5   9   6
 2|    1   6   7   2   8   8   7   6   2   7   1   6
 3|    1   5   8   5   9   5   4   2   1   6   5   5
 4|    7   4   9   2   2   9   7   8   8   7   7   4
 5|    1   3   6   9   2   7   2   1   4   2   7   1
 6|    9   1   6   4   3   2   5   1   7   3   4   6
 7|    1   3   1   2   7   9   1   3   8   1   6   7
```

```
 8|    8   6   2   6   1   6   9   4   8   2   5   6
 9|    7   2   8   6   9   4   1   9   7   2   8   4
10|    9   7   8   8   6   8   6   2   5   1   2   9
11|    8   5   4   2   6   8   2   4   1   8   7   9
12|    1   5   5   3   3   9   6   9   8   1   9   8
```

com a cratera disposta no plato (linha,coluna): 10 1

Pergunta-se quantos platôs são atingidos nos instantes t=3, t=4 e t=5 ?

caminhos: linha e coluna da sexta letra?	vulcão: em t=3, t=4 e t=5?

48.3 Exercício 3

Caminhos

☞ **Para você fazer**

Seja a palavra TRABALHAM e o tabuleiro

```
    1   2   3   4   5   6   7   8   9  10  11  12
  +-----------------------------------------------
 1| E   T   R   Z   J   G   C   W   S   S   U   A
 2| C   E   F   Y   C   I   I   X   T   L   D   H
 3| K   A   Q   M   O   O   U   L   M   C   S   U
 4| R   R   C   G   A   A   C   A   H   W   W   S
 5| Q   N   B   T   F   B   X   A   M   Q   U   L
 6| N   H   X   A   G   H   U   T   Z   Y   G   K
 7| C   I   U   A   H   H   S   W   K   U   H   L
 8| A   W   M   P   L   B   A   X   M   V   N   A
 9| B   P   P   A   O   A   A   M   U   G   X   O
10| F   X   M   H   X   R   F   F   I   R   V   Z
11| U   U   D   D   O   T   T   O   I   Z   U   F
12| Q   L   F   M   T   H   R   H   O   S   N   Y
```

Pergunta-se: Qual a linha e coluna da sexta letra da palavra ?

Seja o vulcão

```
    1   2   3   4   5   6   7   8   9  10  11  12
  +-----------------------------------------------
 1|    6   7   3   2   1   4   8   8   5   8   2   5
 2|    8   8   8   8   1   1   6   3   8   2   3   6
 3|    7   2   6   9   7   8   5   6   1   5   1   9
 4|    5   4   3   4   9   8   4   6   8   6   5   3
 5|    9   7   9   4   3   8   4   4   3   4   4   7
 6|    4   7   6   3   2   1   6   2   9   4   7   6
 7|    2   8   7   5   1   3   3   7   5   5   9   3
 8|    1   6   8   4   1   8   8   5   9   6   1   4
 9|    6   6   3   5   6   5   9   9   2   8   6   6
10|    4   3   5   2   6   9   1   6   8   8   9   4
```



```
11|    5   4   1   4   9   3   2   6   2   1   8   3
12|    3   2   3   7   6   3   5   9   2   8   8   9
```

com a cratera disposta no plato (linha,coluna): 5 3

Pergunta-se quantos platôs são atingidos nos instantes t=3, t=4 e t=5 ?

caminhos: linha e coluna da sexta letra?	vulcão: em t=3, t=4 e t=5?

48.4 Exercício 4

Caminhos

☞ **Para você fazer**

Seja a palavra SERVINDO e o tabuleiro

```
    1   2   3   4   5   6   7   8   9  10  11  12
  +-----------------------------------------------
 1| S   M   V   H   B   Z   X   E   I   R   W   H
 2| S   U   B   U   H   N   O   H   N   H   K   L
 3| R   D   Z   F   R   G   W   D   A   E   N   Y
 4| I   D   S   X   O   Q   D   U   T   Q   H   D
 5| N   A   W   V   R   K   T   P   R   R   G   G
 6| I   V   W   G   I   U   X   F   K   N   R   B
 7| C   H   I   D   W   E   P   Y   I   O   D   E
 8| G   O   R   E   L   N   Q   R   K   S   V   K
 9| V   C   H   J   X   D   R   U   V   S   U   J
10| A   E   T   K   O   T   G   H   T   I   D   J
11| G   B   S   S   N   H   W   A   G   D   N   S
12| A   O   O   E   S   S   W   L   T   V   E   O
```

Pergunta-se: Qual a linha e coluna da sexta letra da palavra ?

Seja o vulcão

```
    1   2   3   4   5   6   7   8   9  10  11  12
  +-----------------------------------------------
 1|    3   1   7   4   2   7   9   9   8   8   1   5
 2|    5   8   2   6   8   4   3   4   9   8   4   9
 3|    5   2   4   3   7   3   7   6   9   9   1   4
 4|    8   9   2   4   5   3   8   4   9   3   6   1
 5|    3   3   9   9   7   1   8   7   7   6   2   3
 6|    5   7   8   2   8   7   1   4   3   2   7   1
 7|    5   6   8   4   8   8   6   1   8   2   9   2
 8|    3   8   7   1   7   8   9   7   2   6   7   6
 9|    4   8   8   3   9   8   3   8   9   8   2   6
10|    7   9   7   1   3   7   1   2   8   6   2   1
11|    1   1   2   5   5   6   2   9   7   5   8   8
12|    7   9   3   7   7   7   6   1   8   6   8   7
```

com a cratera disposta no plato (linha,coluna): 711

Pergunta-se quantos platôs são atingidos nos instantes t=3, t=4 e t=5 ?

caminhos: linha e coluna da sexta letra?	vulcão: em t=3, t=4 e t=5?

48.5 Respostas

```
1       5 11      14 17 18
2       9  1      10 16 17
3       8  5      17 21 22
4       2  9      15 21 22
```


Capítulo 49

Exercício prático: 156 - Genoma e Palavras Cruzadas

49.1 Projeto Genoma

Um grande projeto mundial está em curso para mapear todo o material genético do ser humano: o Projeto Genoma Humano. As moléculas de DNA (moléculas que contêm material genético) podem ser representadas por cadeias de caracteres que usam um alfabeto de apenas 4 letras: 'A', 'C', 'T' e 'G'. Um exemplo de uma tal cadeia é:

```
TCATATGCAAATAGCTGCATACCGA
```

Nesta tarefa você deverá produzir uma ferramenta muito utilizada no projeto Genoma: um programa que procura ocorrências de uma pequena cadeia de DNA (que vamos chamar de p) dentro de uma outra cadeia de DNA (que vamos chamar de t). Você deverá procurar dois tipos de ocorrência: a "direta"e a "complementar invertida". Uma ocorrência *direta* é quando a cadeia p aparece como subcadeia dentro de t. Por exemplo, se

```
p = CATA          e          t = TCATATGCAAATAGCTGCATACCGA
```

então p ocorre na forma direta na posição 2 e na posição 18 de t. Uma ocorrência *complementar invertida* depende da seguinte correspondência entre as letras do DNA: 'A' $\iff$ 'T' e 'G' $\iff$ 'C'. "Complementar o DNA"significa trocar as letras de uma cadeia de DNA seguindo essa correspondência. Se complementarmos a cadeia CATA, vamos obter GTAT. Mas além de complementar, é preciso também inverter, ou seja, de GTAT obter TATG. E é esta cadeia que deverá ser procurada, no caso da ocorrência complementar invertida. Assim, se p e t são as mesmas cadeias do exemplo anterior, então p ocorre na forma complementar invertida na posição 4 de t.

Tarefa: Sua tarefa é escrever um programa que, dadas duas cadeias p e t, onde o comprimento de p é menor ou igual ao comprimento de t, procura todas as ocorrências diretas e todas as ocorrências complementares invertidas de p em t.

Entrada de Dados: A entrada contém vários conjuntos de teste. Cada conjunto de teste é composto por três linhas. A primeira linha contém dois inteiros positivos, M e N, M N, que indicam respectivamente o comprimento das cadeias de DNA p e t, conforme descrito acima. A segunda linha do conjunto de teste contém a cadeia p, e a terceira linha contém a cadeia t, onde p e t são compostas utilizando apenas os caracteres 'A', 'C', 'G' e 'T'. O final do arquivo de testes é indicado quando M = N = 0 (este último conjunto de testes não é válido e não deve ser processado).

```
Exemplo de Entrada
2 4
AC
TGGT
4 25
CATA
TCATATGCAAATAGCTGCATACCGA
0 0
```

Saída de Dados: Seu programa deve produzir uma saída onde para cada conjunto de teste do arquivo de entrada seu programa deve produzir duas linhas. Na primeira linha deve aparecer a lista, em ordem crescente, com a posição inicial de cada ocorrência, na forma direta, do padrão p na seqüência t. Na segunda linha deve aparecer a lista, em ordem crescente, com a posição inicial de cada ocorrência, na forma complementar invertida, do padrão p na seqüência t.

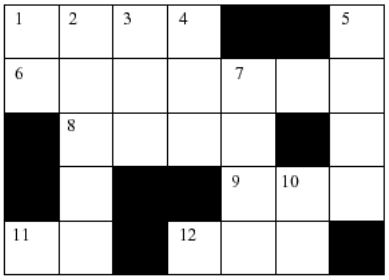
```
Exemplo de Saída
Teste 1
ocorrencia direta: 0
ocorrencia complementar invertida: 3
Teste 2
ocorrencia direta: 2 18
ocorrencia complementar invertida: 4
(esta saída corresponde ao exemplo de entrada acima)
```

Este exercício é igual ao proposto na OBI99.

49.2 Palavras Cruzadas

O conhecido passatempo de palavras cruzadas é composto por uma grade retangular de quadrados brancos e pretos e duas listas de definições. Uma das listas de definições é para palavras escritas da esquerda para a direita nos quadrados brancos (nas linhas) e a outra lista é para palavras que devem ser escritas de cima para baixo nos quadrados brancos (nas colunas). Uma palavra é uma seqüência de dois ou mais caracteres do alfabeto. Para resolver um jogo de palavras cruzadas, as palavras correspondentes às definições devem ser escritas nos quadrados brancos da grade.

As definições correspondem às posições das palavras na grade. As posições são definidas por meio de números inteiros seqüenciais colocados em alguns quadrados brancos. Um quadrado branco é numerado se uma das seguintes condições é verificada: (a) tem como vizinho à esquerda um quadrado preto e como vizinho à direita um quadrado branco; (b) tem como vizinho acima um quadrado preto e como vizinho abaixo um quadrado branco; (c) é um quadrado da primeira coluna à esquerda e tem como vizinho à direita um quadrado branco; d) é um quadrado da primeira linha acima e tem como vizinho abaixo um quadrado branco. Nenhum outro quadrado é numerado. A numeração começa em 1 e segue seqüencialmente da esquerda para a direita, de cima para baixo. A figura abaixo ilustra um jogo de palavras cruzadas com numeração apropriada.



Uma palavra horizontal é escrita em uma seqüência de quadrados brancos em uma linha, iniciando-se em um quadrado numerado que tem um quadrado preto à esquerda ou que está na primeira coluna à esquerda. A seqüência de quadrados para essa palavra continua da esquerda para a direita, terminando no quadrado branco imediatamente anterior a um quadrado preto, ou no quadrado branco da coluna mais à direita da grade.

Uma palavra vertical é escrita em uma seqüência de quadrados brancos em uma coluna, iniciando-se em um quadrado numerado que tem um quadrado preto acima ou que está na primeira linha acima. A seqüência de quadrados para essa palavra continua de cima para baixo, terminando no quadrado branco imediatamente anterior a um quadrado preto, ou no quadrado branco da coluna mais abaixo da grade.

Tarefa: Sua tarefa é escrever um programa que recebe como entrada uma especificação de um tabuleiro de palavras cruzadas e deve produzir 3 coisas:

* A quantidade total de casas numeradas
* a lista de casas iniciais para as palavras HORIZONTAIS
* a lista de casas iniciais para as palavras VERTICAIS

Entrada de Dados: A entrada contém 2 linhas. Na primeira linha há dois inteiros positivos, M e N, que indicam respectivamente o número de linhas e o número de colunas do jogo de palavras cruzadas. A segunda linha contém os números das casas pretas do tabuleiro. As casas do tabuleiro foram numeradas de 1 (a mais alta à esquerda) ate $M \times N$, a mais baixa à direita.

```
Exemplo de Entrada
10 10
 2 13 18 20 23 24 37 47 57 72 75 77 86 89 95
```

Saída de Dados: A saida deve produzir o maior número achado, a lista de casas que iniciam palavras horizontais e a lista ... verticais.

```
30
  2  8 10 11 12 15 18 19 20 21 22 23 24 25 26 28 29 30
  1  3  4  5  6  7  9 13 14 16 17 27 28
(esta saída corresponde ao exemplo de entrada acima)
```

Apenas para referência, eis como ficou a numeração do tabuleiro acima (99 é preto)

```
  1  99   2   3   4   5   6   0   7   0
  8   9  99  10   0   0   0  99   0  99
 11   0  99  99  12   0   0  13   0  14
 15   0  16  17   0   0  99  18   0   0
 19   0   0   0   0   0  99  20   0   0
 21   0   0   0   0   0  99  22   0   0
```



```
 23   0   0   0   0   0   0   0   0   0
  0  99  24   0  99   0  99  25   0   0
 26  27   0   0   0  99  28   0  99   0
 29   0   0   0  99  30   0   0   0   0
```

Este exercício está baseado (mas lá é diferente), na prova da OBI 99.

49.3 Exercício 1

Para o problema do Genoma, seja a entrada:

```
  4 40
TGGT
GAAGGGGATGGTGCAGGGCATGGTGTACCATTGTTTCGTC
  5 44
GGACT
CTGTGGACAGTAGTATTGAGTCCGGACTCTGACTTGTCAAGCAG
0 0
```

Pergunta-se: quais as 4 linhas da resposta (2 linhas para cada instância)

Sejam os tabuleiros de palavras cruzadas

```
  7  9
  7  9 15 20 38 40 46 52 57 59
  8  8
 10 13 24 27 29 34 39 50 53 56
  0 0
```

Pergunta-se 3 valores para cada instância: o valor da linha 1 e os PENÚLTIMOS valores da segunda e terceira linha.

L1 i 1	L2 i 1	L1 i 2	L2 i 2	L1, p/L2 p/L3 i 1	L1, p/L2 p/L3 i 2

49.4 Exercício 2

Para o problema do Genoma, seja a entrada:

```
  5 34
AAAGT
ATGTTTGTTAAAGTAAGTAAAGTTACTTTGCTAG
  6 46
CATGCC
TGCAATCCTAGACCATGCCGAGCTGGCTGCAGGCATGATGCCCGAC
0 0
```

Pergunta-se: quais as 4 linhas da resposta (2 linhas para cada instância)

Sejam os tabuleiros de palavras cruzadas

```
 7  7
8 20 21 27 31 32 40 45
8  8
3  4 12 16 17 22 37 43 46 61
0 0
```

Pergunta-se 3 valores para cada instância: o valor da linha 1 e os PENÚLTIMOS valores da segunda e terceira linha.

L1 i 1	L2 i 1	L1 i 2	L2 i 2	L1, p/L2 p/L3 i 1	L1, p/L2 p/L3 i 2

49.5 Exercício 3

Para o problema do Genoma, seja a entrada:

```
  4 40
CCTT
CTGGGAGTGCGCACGTGACTGAAGGTATCCTTTTATTTAT
  5 49
TGGCA
ACAGTGGCAGGGTGCCAGACGATTGTGGCACGTTGATTTGCAGGGCCCC
0 0
```

Pergunta-se: quais as 4 linhas da resposta (2 linhas para cada instância)
Sejam os tabuleiros de palavras cruzadas

```
 6  8
 3  9 10 11 16 18 26 41
10 10
14 18 21 22 24 27 58 61 63 69 71 74 75 78 88
 0 0
```

Pergunta-se 3 valores para cada instância: o valor da linha 1 e os PENÚLTIMOS valores da segunda e terceira linha.

L1 i 1	L2 i 1	L1 i 2	L2 i 2	L1, p/L2 p/L3 i 1	L1, p/L2 p/L3 i 2

49.6 Exercício 4

Para o problema do Genoma, seja a entrada:


```
  4 38
GGTT
GTTCCCCGTAGAAGTGTAGTAGAACCGGGTTGCCAGCA
  5 50
GTAGA
TCTACGTCTACATGTAGTAGATAGATGTAACAAGTACTAAGTAACGATGA
0 0
```

Pergunta-se: quais as 4 linhas da resposta (2 linhas para cada instância)

Sejam os tabuleiros de palavras cruzadas

```
8  7
5  7 10 12 23 28 35 42 55
8  8
4  8 13 31 33 36 41 49 53 56
0 0
```

Pergunta-se 3 valores para cada instância: o valor da linha 1 e os PENÚLTIMOS valores da segunda e terceira linha.

L1 i 1	L2 i 1	L1 i 2	L2 i 2	L1, p/L2 p/L3 i 1	L1, p/L2 p/L3 i 2

49.7 Respostas

```
1     9 21  0    27  0  0    24  0  0    19  0  0    18 17 14    20 19 15
2    10 19  0    25  0  0    14  0  0    32  0  0    19 18 14    25 24 22
3    29  0  0    22  0  0     5 26  0    13  0  0    13 11  9    30 29 27
4    28  0  0    23  0  0    17  0  0     1  7  0    17 15 13    21 20 16
```

Capítulo 50

Exercício prático 157

50.1 Trem ou Caminhão?

A produtora de refrigerantes CaraCola precisa enviar com freqüência grandes carregamentos para as suas distribuidoras em outros estados. Para isso ela pode utilizar uma transportadora que trabalha com caminhões ou uma transportadora que trabalha com trens. As duas transportadoras competem agressivamente para conseguir o serviço, mas seus custos dependem do momento (por exemplo, se há ou não caminhões disponíveis, etc.). A cada carregamento, a CaraCola consulta as duas transportadoras, que informam as condições de preço vigentes no momento, para o estado desejado. Sua tarefa é escrever um programa que, baseado nas informações das transportadoras, decida se o melhor é enviar o carregamento por trem ou por caminhão.

As transportadoras informam os seus custos na forma de duas variáveis, representando duas parcelas. Uma parcela é um custo fixo A que independe do peso do carregamento, e a outra parcela é um custo variável B que depende do peso do carregamento, em kilogramas. A CaraCola utiliza o peso do carregamento para calcular o custo dos transporte por trem e por caminhão e decidir qual empresa transportadora contratar. Por exemplo, suponha que a transportadora por trem informa que o seu custo fixo é A = R$ 450,00 e o seu custo por kilograma é B = R$ 3,50. Suponha ainda que a transportadora por caminhão informa que seu custo fixo é A = R$ 230,00 e o seu custo por kilograma é B = R$ 3,70. Neste caso, para um carregamento que pesa 2354 kg a CaraCola decide que é melhor fazer o envio por trem, pois $450 + 3,50 \times 2354 < 230 + 3,70 \times 2354$. Se a diferença entre os custos for menor do que R$ 1,00 a CaraCola prefere o transporte por trem.

Tarefa Sua tarefa é escrever um programa que recebe como entrada vários casos, cada um apresentando uma lista de custos, e determina se a CaraCola deve enviar o carregamento por trem ou por caminhão.

Entrada de Dados A entrada contém vários conjuntos de teste. Cada conjunto de teste é composto por uma linha, que contém cinco valores. O primeiro valor é um número inteiro positivo K que representa o peso, em kilogramas, do carregamento. Os quatro valores restantes são números reais A, B, C e D que representam os custos informados pelas empresas de transporte. A e B representam respectivamente o custo fixo e o custo variável por kilograma informado pela empresa que utiliza trem. C e D representam respectivamente o custo fixo e o custo variável por kilograma informado pela empresa que utiliza caminhão. Os custos são apresentados sempre com precisão de dois algarismos decimais. O final do arquivo de testes é indicado quando K = 0 (este último conjunto de testes não é válido e não deve ser processado).

`Exemplo de Entrada`

```
2354   450.00  3.50   230.00  3.70
1000   411.50  2.85   411.50  2.85
2327   325.00  3.10   556.50  3.00
0
```

Saída de Dados Para cada conjunto de teste do arquivo de entrada seu programa deve produzir a informação 1=trem ou 2=caminhão.

```
Exemplo de Saída: 1, 1 e 2 (corresponde à entrada acima)
```

50.2 RoboCoffee

Dona Mercedes, a eficiente funcionária responsável pela distribuição de cafezinho na empresa RoboCamp, fabricante de robôs industriais, vai aposentar-se em breve. Para ocupar a função de Dona Mercedes os engenheiros da empresa adaptaram um robô existente, ao qual acoplaram uma máquina de café. Para diminuir os custos de fabricação, o novo robô tem uma operação bem simples, sendo capaz de apenas três movimentos básicos: ficar parado (para que os empregados possam servir-se de café), andar para a frente, e girar sobre o seu eixo (para colocar-se de frente para o seu próximo ponto destino).

A programação do robô é feita através de uma seqüência de N pontos no plano, numerados de 0 a N-1 (o robô desenvolvido não sobe escadas, mas felizmente a empresa ocupa um único pavimento). Cada ponto é determinado por coordenadas inteiras (X,Y). O ponto i é sempre distinto do ponto i+1, para $0 \leq i < N$, e o ponto N-1 é distinto do ponto 0 (veja a regra 4 abaixo).

O robô move-se a partir do ponto 0, através de todos os pontos dados, observando as seguintes regras:

1. O robô inicia no ponto 0 de frente para o ponto 1;
2. O robô move-se sempre para a frente;
3. Chegando ao ponto i ($0 \leq i \leq N-1$) o robô gira sobre seu eixo, no sentido do relógio, de um ângulo α ($0o \leq \alpha < 360o$), de modo a ficar de frente para o ponto (i+1) mod N, e faz uma pausa para que os usuários possam servir-se de café;
4. No final, para completar o percurso, o robô movimenta-se do ponto N-1 para o ponto 0, e gira de modo a ficar de frente para o ponto 1.

Tarefa Sua tarefa é escrever um programa que, dada a seqüência de N pontos no plano que corresponde à programação do robô, determina quantas voltas completas sobre o seu eixo o robô perfaz durante seu percurso.

Entrada de Dados Cada conjunto de teste corresponde a uma programação do robô. A linha de um conjunto de testes contém um inteiro positivo, N, que indica o número de pontos presentes no conjunto de teste, seguido por N pares de números inteiros que correspondem ao valor das coordenadas X e Y de um ponto, separados por espaço em branco ($-1000 \leq X \leq 1000$, $-1000 \leq Y \leq 1000$). O final do arquivo de testes é indicado quando $N = 0$ (este último conjunto de teste não é válido e não deve ser processado).

```
Exemplo de Entrada
4 , 1 1 , 1 0 , 0 0 , 0 1
4 , 2 -3 , 2 2 , -3 3 , -2 -1
0
```

Saída de Dados Para cada conjunto de teste do arquivo de entrada seu programa deve indicar o número de rotações completas efetuadas pelo robô durante seu percurso, encontrado pelo seu programa.

```
Exemplo de Saída: 1 e 3 (corresponde à entrada)
```

50.3 Restaurante

A únicoMP (Universidade Independente de Computação) possui vários refeitórios que servem seus milhares de alunos e professores. Para melhorar o atendimento a únicoMP planeja fazer uma reforma nos refeitórios, mas para isso necessita saber qual o número máximo de pessoas que são atendidas simultaneamente em um mesmo refeitório. Para isso a únicoMP, que possui catracas eletrônicas, coletou os seguintes dados:

- Um vetor E, ordenado crescentemente, em que E[i] representa o instante de tempo em que a pessoa i entrou no restaurante;
- Um vetor S, em que S[i] representa o instante de tempo em que a pessoa i saiu do restaurante.

Os elementos de E e S são inteiros positivos que indicam o número de minutos transcorridos desde a abertura do restaurante. A entrada e a saída do restaurante se faz por uma única catraca, onde passa apenas uma pessoa por vez. Se acontecer de uma pessoa entrar e outra sair no mesmo minuto, quem entra tem a preferência de uso da catraca. Nos elementos de E não há valores repetidos. A mesma coisa para S.

Tarefa Sua tarefa é escrever um programa que, dados dois vetores de inteiros E e S, ambos de comprimento igual a N, calcula o número máximo de pessoas que estão presentes ao mesmo tempo dentro do restaurante.

Entrada de Dados Cada conjunto de teste é composto por três linhas. A primeira linha contém um inteiro positivo, N, que indica o comprimento dos vetores E e S, conforme descrito acima. A segunda linha do conjunto de teste contém os elementos do vetor E, separados por espaço em branco, e a terceira linha contém os elementos do vetor S, separados por espaço em branco. O final do arquivo de testes é indicado quando N = 0 (este último conjunto de testes não é válido e não deve ser processado).

```
Exemplo de Entrada
3
   14   67   98
 1890 1900 2123
2
  200 1800
 1543 2324
0
```

Saída de Dados Para cada conjunto de teste do arquivo de entrada seu programa deve produzir o número máximo de pessoas presentes simultaneamente no restaurante, encontrado pelo seu programa.

```
Exemplo de Saída: 3 e 1 (corresponde à entrada acima)
```

50.4 Exercício 1

Transporte:


```
   1471      930   1.34      308   3.27
   1116      824   1.48      270   2.49
   1975      900   1.75      290   1.56
 0 0
```

Robot

```
 5  -4  1, -2  2, -5 -3,  2 -4,  0 -2
 4   4 -1, -3 -2,  2  3,  0 -3
 4  -2  0,  1  3, -1  2,  4  4
 0 0
```

Restaurante

```
 12
 4  7 11 12 13 18 23 26 27 31 32 37
13 14 19 23 27 29 34 39 41 45 47 50
 11
 3  6 10 13 17 21 22 23 24 29 30
12 14 16 17 19 20 23 24 28 30 33
 10
 4  9 13 16 21 22 26 29 32 34
14 19 23 25 29 34 38 42 45 48
 0 0
```

Responda

T1	T2	T3	Rb1	Rb2	Rb3	R1	R2	R3

50.5 Exercício 2

Transporte:

```
    483      965   1.35      171   1.64
   1002      721   2.58      438   1.99
   1090      653   2.38      314   2.47
 0 0
```

Robot

```
6   2 -4, -4  3,  3  4,  0 -1, -3 -2,  1  2
5  -2  0, -5  1, -1  3,  0 -5, -3 -2
4   0  3, -2  2, -3 -5, -1 -1
 0 0
```

Restaurante

```
 10
 5 10 14 15 19 22 24 25 28 30
13 15 17 21 24 27 28 33 38 41
 10
 1  5  7  9 11 16 18 23 25 27
15 16 18 19 24 28 33 38 43 46
 11
```

```
 1  6  7 12 14 17 21 25 26 27 28
11 12 13 16 17 21 24 25 30 34 39
 0 0
```

Responda

T1	T2	T3	Rb1	Rb2	Rb3	R1	R2	R3

50.6 Exercício 3

Transporte:

```
    677      930   1.71      353   2.31
    459      852   1.48      412   2.76
    668      711   2.36      123   2.86
 0 0
```

Robot

```
6   1 -1,  0 -3, -1  0, -2  2,  2 -2,  3  1
5   2  3,  0 -4, -4  4,  4 -3,  3  1
4  -2 -2,  2  0,  1 -3, -1  2
 0 0
```

Restaurante

```
 10
 2  5  6  9 10 12 13 18 23 25
14 19 23 25 29 32 33 38 40 43
 10
 1  6 10 12 15 18 19 24 28 31
11 15 16 21 25 26 27 28 30 32
 12
 3  5  6  8 13 17 18 19 20 22 27 29
11 13 14 19 20 25 28 29 31 34 35 37
 0 0
```

Responda

T1	T2	T3	Rb1	Rb2	Rb3	R1	R2	R3

50.7 Exercício 4

Transporte:

```
    961      690   2.15      271   2.24
    692      690   1.72      417   3.19
   1943      624   1.28      294   3.80
 0 0
```

Robot


```
4   1 -4, -5 -2,  2 -3, -3  3
4   3 -3,  2 -1, -3 -2,  4  5
7  -1  1, -2  4, -3 -1,  1  0,  4 -3,  5  5,  0 -2
 0 0
```

Restaurante

```
 11
 3  4  8 13 17 20 24 28 33 36 40
15 17 20 25 29 32 35 40 43 45 48
 10
 3  8  9 10 13 15 19 22 24 27
11 13 17 20 21 22 24 28 32 37
 13
 1  5  6 10 15 20 24 27 31 32 33 38 42
15 20 25 30 32 34 39 43 44 47 50 54 56
 0 0
```

Responda

T1	T2	T3	Rb1	Rb2	Rb3	R1	R2	R3

50.8 Respostas

```
1      1  1  2      3  2  2      5  3  5
2      2  2  2      2  2  3      3  5  3
3      2  1  2      4  3  2      7  4  5
4      2  1  1      2  2  3      4  4  6
```

Capítulo 51

Exercício prático 158

51.1 Seqüências

Uma seqüência de 0's e 1's é chamada de seqüência-H se é composta por um único 0 ou se é composta por um 1 seguido de duas seqüências-H. Por exemplo, 0, 100 e 10100 são seqüências-H, mas 10, 111 e 10010 não são.

Tarefa Sua tarefa é determinar se uma dada seqüência é ou não uma seqüência-H.

Entrada de Dados Cada conjunto de teste é composto por uma única linha, que contém a seqüência a ser testada, composta de dígitos 0's e 1's (sem espaços em branco intermediários). O final da seqüência é indicado pelo primeiro espaço em branco da linha. O final da entrada é indicado quando os primeiros caracteres da linha de teste são -1.

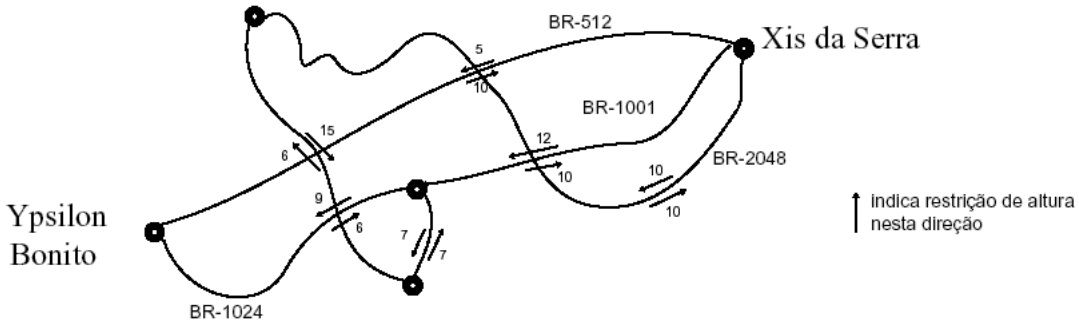
```
Exemplo de Entrada
0
10100
10010
-1
```

Saída de Dados Para cada conjunto de teste do arquivo de entrada seu programa deve produzir o dígito 1 se a seqüência é uma seqüência-H, ou o dígito 0 se a seqüência não é uma seqüência-H.

```
Exemplo de Saída
1 , 1 , 0   (esta saída corresponde ao exemplo de entrada acima)
```

51.2 Carga Pesada

Um dos maiores problemas no transporte de cargas pesadas por rodovias é a altura dos viadutos, pois muitas vezes a carga é mais alta do que o vão do viaduto sob o qual o caminhão deve passar. Considere o mapa abaixo. É possível tranportar uma turbina de hidroelétrica, que em cima do caminhão mede 7 metros de altura, da cidade Xis da Serra para a cidade Ypsilon Bonito?



Neste caso é fácil perceber que a resposta é sim, pois o menor vão encontrado no caminho, se utilizarmos as estradas BR-1001 e BR-1024, é de 9 metros. No caso geral, com dezenas de cidades e estradas, a resposta pode não ser tão evidente.

Tarefa Sua tarefa é determinar, para um dado par de cidades X e Y, qual a carga mais alta que pode ser transportada de X para Y por meio rodoviário, conhecendo todas as estradas da região e a altura de todos os túneis e viadutos dessas estradas. Considere que todas as estradas têm limitação de altura e que a interligação das estradas ao redor das cidades não tem restrição de altura.

Entrada de Dados O número máximo de cidades, N, em cada teste é 100. As cidades são numeradas de 1 a N. O primeiro par do conjunto de teste contém dois inteiros positivos X e Y que representam respectivamente as cidades origem e a cidade destino da carga ($1 \leq X \leq N$, $1 \leq Y \leq N$). As triplas seguintes contêm cada uma a descrição de uma estrada. Cada descrição é composta por três inteiros A, B e C, representando respectivamente a cidade onde a estrada inicia, a cidade onde a estrada termina e a altura do viaduto ou túnel mais baixo no trajeto de A para B. Cada instância do problema ocupa 2 linhas. O final do arquivo de testes é indicado quando X = Y = 0 (este último conjunto de testes não é válido e não deve ser processado).

```
Exemplo de Entrada
2 4 ; 1 4 5 ; 2 4 12 ; 0 0 0
1 3 ; 1 2 10 ; 1 3 8 ; 2 3 12 ; 3 1 5 ; 0 0 0
0 0
```

Saída de Dados Na resposta deve aparecer a altura máxima da carga, encontrada pelo seu programa.

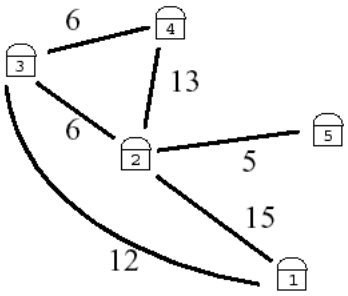
```
Exemplo de Saída
 12 , 10
(esta saída corresponde ao exemplo de entrada acima)
```

51.3 Rede ótica

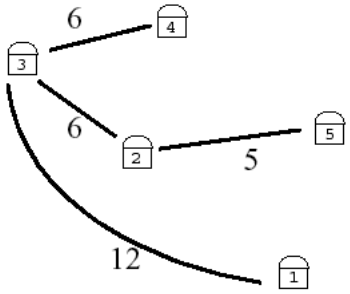
Os caciques da região de Tutuaçu pretendem integrar suas tribos à chamada "aldeia global". A primeira providência foi a distribuição de telefones celulares a todos os pajés. Agora, planejam montar uma rede de fibra ótica interligando todas as tabas. Esta empreitada requer que sejam abertas novas picadas na mata, passando por reservas de flora e fauna. Conscientes da necessidade de preservar o máximo possível o meio ambiente, os caciques encomendaram um estudo do impacto ambiental do projeto. Será que você consegue ajudá-los a projetar a rede de fibra ótica?

Tarefa Vamos denominar uma ligação de fibra ótica entre duas tabas de um ramo de rede. Para possibilitar a comunicação entre todas as tabas é necessário que todas elas

estejam interligadas, direta (utilizando um ramo de rede) ou indiretamente (utilizando mais de um ramo). Os caciques conseguiram a informação do impacto ambiental que causará a construção dos ramos. Alguns ramos, no entanto, nem foram considerados no estudo ambiental, pois sua construção é impossível.



Ramos de rede possíveis com custo ambiental associado



Interligação das tabas com menor custo ambiental

Sua tarefa é escrever um programa para determinar quais ramos devem ser construídos, de forma a possibilitar a comúnicação entre todas as tabas, causando o menor impacto ambiental possível.

Entrada A entrada é composta de vários conjuntos de teste. A primeira linha de um conjunto de teste contém o número de tabas. As tabas são numeradas de 1 a N. As M indicações seguintes contêm três inteiros positivos X, Y e Z, que indicam que o ramo de rede que liga a taba X à taba Y tem impacto ambiental Z. Com os conjuntos de teste dados sempre é possível interligar todas as tabas. O final da entrada é indicado quando N = 0.

```
Exemplo de Entrada
3 ; 1 2 10 ; 2 3 10 ; 3 1 10
5 ; 1 2 15 ; 1 3 12 ; 2 4 13 ; 2 5 5 ; 3 2 6 ; 3 4 6
0
```

Saída Para cada conjunto de teste da entrada seu programa deve produzir uma lista dos ramos de redes que devem ser construídos. Um ramo é descrito por um par de tabas X e Y , com X < Y. Os ramos de rede podem ser listados em qualquer ordem, mas não deve haver repetição. Se houver mais de uma solução possível, imprima apenas uma delas.

```
Exemplo de Saída
1 2 ; 1 3
1 3 ; 2 3 ; 2 5 ; 3 4
(esta saída corresponde ao exemplo de entrada acima)
```

51.4 Exercício 1

Seqüência-H:

```
111011001000100
11010111000110110111110010010000
11010100101011001101001101010000
11101100111010001001000
-1
```


Carga Pesada

```
5 3; 1 4 12;2 1  3;2 3 11;3 1  3;3 5  5;
4 1  6;4 2  9;5 1 10;5 2  4;5 3  7;0 0  0

5 2; 1 2 6;1 3 4;1 5 5;2 1 2;3 4 7;
4 1 9;4 5 4;5 2 5;5 3 8;0 0 0
0 0
```

Rede ótica

```
6; 4 6  4;1 4  5;1 2  6;3 5  7;1 3  9;2 6 10;2 5 11;
1 5 12;2 3 13;3 6 14;2 4 15;1 6 16;5 6 17;3 4 19;4 5 20

6; 2 3  4;2 5  6;1 6  7;2 4  8;3 6  9;1 5 11;3 4 12;
4 5 14;1 2 16;1 4 17;3 5 19;4 6 20;2 6 21;5 6 22;1 3 23
 0
```

Responda

S1	S2	S3	S4	C1	C2	R1	R2

51.5 Exercício 2

Seqüência-H:

```
11010100111101100100111100100110000000
110011001110100100100
11100101111010101100000100110000
1110010100100
-1
```

Carga Pesada

```
5 4; 1 2 3;1 3 9;1 4 6;2 1 6;2 4 8;2 5 5;
3 2 7;3 5 5;4 5 4;5 1 6;5 4 1;0 0 0

4 3; 1 3  9;1 5  5;2 3  4;2 5 12;3 5  6;
4 1  3;4 2 10;4 3  8;5 1 11;5 3  3;0 0  0
0 0
```

Rede ótica

```
6; 4 5  4;1 3  5;3 4  6;5 6  7;4 6  8;1 5  9;2 6 10;
2 5 11;1 4 12;3 5 15;1 2 18;2 3 19;1 6 20;2 4 21;3 6 22

6; 4 6  4;3 5  5;3 6  6;2 4  8;1 3  9;2 3 10;1 2 11;
2 5 12;2 6 14;3 4 15;1 4 17;1 5 18;4 5 19;5 6 21;1 6 23
 0
```

Responda

S1	S2	S3	S4	C1	C2	R1	R2

51.6 Exercício 3

Seqüência-H:

```
1011110000100
1110011100000010100
1111010100101100100100
11110101010010001100101011010000
-1
```

Carga Pesada

```
4 1; 1 2 4;2 1 7;2 3 5;2 4 3;3 2 5;
3 5 8;4 1 1;4 3 6;5 1 4;5 2 5;0 0 0

4 1; 1 3 6;2 3 6;2 4 3;2 5 5;3 1 8;
4 1 3;4 3 3;4 5 7;5 1 4;5 2 9;0 0 0
0 0
```

Rede ótica

```
6; 3 6  5;4 5  6;3 4  7;3 5  8;2 5  9;1 4 10;1 2 11;
1 6 13;1 3 15;2 3 16;5 6 17;1 5 19;2 6 20;4 6 21;2 4 23

6; 3 4  4;1 6  5;2 3  6;3 6  7;2 4  8;2 5  9;1 4 10;
1 5 12;1 2 13;4 5 15;4 6 16;1 3 19;5 6 20;2 6 22;3 5 23
 0
```

Responda

S1	S2	S3	S4	C1	C2	R1	R2

51.7 Exercício 4

Seqüência-H:

```
1110011001000
11010111001001011000000
1100101111001000100
1100101101000101000
-1
```

Carga Pesada

```
2 5; 1 2 3;1 3 5;1 4 7;1 5 5;2 1 8;
2 5 4;3 2 6;3 5 9;4 3 6;5 1 4;0 0 0

1 4; 1 2 8;1 4 2;2 3 7;2 5 5;3 2 4;
3 4 5;3 5 9;4 3 3;5 3 6;5 4 6;0 0 0
0 0
```

Rede ótica

```
6; 2 6  4;1 4  5;1 3  6;3 5  7;1 5  9;2 3 10;1 2 12;
2 4 13;3 6 14;4 6 15;5 6 16;2 5 17;1 6 20;4 5 22;3 4 23

6; 4 5  4;2 4  5;3 4  6;1 3  7;1 6  8;1 4  9;2 6 10;
2 5 11;5 6 12;4 6 17;1 2 18;3 6 20;1 5 21;3 5 22;2 3 23
 0
```

Responda

S1	S2	S3	S4	C1	C2	R1	R2

51.8 Respostas

```
1   1 0 1 1      9  6   46 14 12 35 13   23 25 16 24 36
2   1 1 1 1      6  9   45 13 34 56 26   46 35 36 24 13
3   1 0 1 1      5  6   36 45 34 25 14   34 16 23 36 25
4   1 0 1 0      6  6   26 14 13 35 23   45 24 34 13 16
```

Capítulo 52

Exercício prático - 159

52.1 Quermesse

Os alunos do último ano resolveram organizar uma quermesse para arrecadar fundos para a festa de formatura. A festa prometia ser um sucesso, pois o pai de um dos formandos, Teófilo, dono de uma loja de informática, decidiu doar um computador para ser sorteado entre os que comparecessem. Os alunos prepararam barracas de quentão, pipoca, doces, ensaiaram a quadrilha e colocaram à venda ingressos numerados sequencialmente a partir de 1. O número do ingresso serviria para o sorteio do computador. Ficou acertado que Teófilo decidiria o método de sorteio; em princípio o sorteio seria, claro, computadorizado.

O local escolhido para a festa foi o ginásio da escola. A entrada dos participantes foi pela porta principal, que possui uma roleta, onde passa uma pessoa por vez. Na entrada, um funcionário inseriu, em uma lista no computador da escola, o número do ingresso, na ordem de chegada dos participantes. Depois da entrada de todos os participantes, Teófilo começou a trabalhar no computador para preparar o sorteio. Verificando a lista de presentes, notou uma característica notável: havia apenas um caso, em toda a lista, em que o participante que possuia o ingresso numerado com i, havia sido a i-ésima pessoa a entrar no ginásio. Teófilo ficou tão encantado com a coincidência que decidiu que o sorteio não seria necessário: esta pessoa seria o ganhador do computador.

Tarefa Conhecendo a lista de participantes, por ordem de chegada, sua tarefa é determinar o número do ingresso premiado, sabendo que o ganhador é o único participante que tem o número do ingresso igual à sua posição de entrada na festa.

Entrada A entrada é composta de vários conjuntos de teste. A primeira linha de um conjunto de teste contém um número inteiro positivo N que indica o número de participantes da festa. A linha seguinte contém a sequência, em ordem de entrada, dos N ingressos das pessoas que participaram da festa.

O final da entrada é indicado quando N = 0. Para cada conjunto de teste da entrada haverá um único ganhador.

```
Exemplo de Entrada
4
4 5 3 1
10
9 8 7 6 1 4 3 2 12 10
0
```

Saída A saída deve conter o número do ingresso do ganhador, conforme determinado pelo seu programa.

```
Exemplo de Saída
3 e 10      (esta saída corresponde ao exemplo de entrada acima)
```

52.2 Bits Trocados

As Ilhas Weblands formam um reino independente nos mares do Pacífico. Como é um reino recente, a sociedade é muito influenciada pela informática. A moeda oficial é o Bit; existem notas de B$ 50,00, B$10,00, B$5,00 e B$1,00. Você foi contratado(a) para ajudar na programação dos caixas automáticos de um grande banco das Ilhas Weblands.

Tarefa Os caixas eletrônicos das Ilhas Weblands operam com todos os tipos de notas disponíveis, mantendo um estoque de cédulas para cada valor (B$ 50,00, B$10,00, B$5,00 e B$1,00). Os clientes do banco utilizam os caixas eletrônicos para efetuar retiradas de um certo número inteiro de Bits. Sua tarefa é escrever um programa que, dado o valor de Bits desejado pelo cliente, determine o número de cada uma das notas necessário para totalizar esse valor, de modo a minimizar a quantidade de cédulas entregues. Por exemplo, se o cliente deseja retirar B$50,00, basta entregar uma única nota de cinquenta Bits. Se o cliente deseja retirar B$72,00, é necessário entregar uma nota de B$50,00, duas de B$10,00 e duas de B$1,00.

Entrada A entrada é composta de vários conjuntos de teste. Cada conjunto de teste é composto por uma única linha, que contém um número inteiro positivo V, que indica o valor solicitado pelo cliente. O final da entrada é indicado por V = 0.

```
Exemplo de Entrada
1
72
0
```

Saída Para cada conjunto de teste da entrada seu programa deve produzir quatro inteiros I, J, K e L que representam o resultado encontrado pelo seu programa: I indica o número de cédulas de B$50,00, J indica o número de cédulas de B$10,00, K indica o número de cédulas de B$5,00 e L indica o número de cédulas de B$1,00.

```
Exemplo de Saída
0 0 0 1 e 1 2 0 2   (esta saída corresponde ao exemplo de entrada acima)
```

52.3 Saldo de gols

Hipólito é um torcedor fanático de um grande clube curitibano. Coleciona flâmulas, bandeiras, recortes de jornal, figurinhas de jogadores, camisetas e tudo o mais que se refira a seu time preferido. Quando ganhou um computador de presente em uma festa, resolveu montar um banco de dados com os resultados de todos os jogos de seu time ocorridos desde a sua fundação, em 1909. Depois de inseridos os dados, Hipólito começou a ficar curioso sobre estatísticas de desempenho do time. Por exemplo, ele deseja saber qual foi o período em que o seu time acumulou o maior saldo de gols. Como Hipólito tem o computador há muito pouco tempo, não sabe programar muito bem, e precisa de sua ajuda.

Tarefa É dada uma lista, numerada seqüencialmente a partir de 1, com os resultados de todos os jogos do time (primeira partida: 3 x 0, segunda partida: 1 x 2, terceira partida: 0 x 5 ...). Sua tarefa é escrever um programa que determine em qual período o time conseguiu acumular o maior saldo de gols. Um período é definido pelos números de seqüência de duas partidas, A e B, onde A B. O saldo de gols acumulado entre A e B é dado pela soma dos gols marcados pelo time em todas as partidas realizadas entre

A e B (incluindo as mesmas) menos a soma dos gols marcados pelos times adversários no período. Se houver mais de um período com o mesmo saldo de gols, escolha o maior período (ou seja, o período em que B - A é maior). Se ainda assim houver mais de uma solução possível, escolha qualquer uma delas como resposta.

Entrada Seu programa deve ler vários conjuntos de teste. A primeira linha de um conjunto de teste contém um inteiro não negativo, N, que indica o número de partidas realizadas pelo time (o valor N = 0 indica o final da entrada). Seguem-se N linhas, cada uma contendo um par de números inteiros não negativos X e Y que representam o resultado da partida: X são os gols a favor e Y os gols contra o time de Hipólito. As partidas são numeradas sequencialmente a partir de 1, na ordem em que aparecem na entrada.

```
Exemplo de Entrada
 2
 2 3 ; 7 1
 9
 2 2 ; 0 5 ; 6 2 ; 1 4 ; 0 0 ; 5 1 ; 1 5 ; 6 2 ; 0 5
 3
 0 2 ; 0 3 ; 0 4
 0
```

Saída Para cada conjunto de teste da entrada seu programa deve produzir um par de inteiros I e J que indicam respectivamente a primeira e última partidas do melhor período, conforme determinado pelo seu programa, exceto quando o saldo de gols do melhor período for menor ou igual a zero; neste caso a segunda linha deve conter a expressão "nenhum".

```
Exemplo de Saída
2 2, 3 8 e nenhum  (esta saída corresponde ao exemplo de entrada
acima)
```

52.4 Exercício 1

Quermesse

```
 17 10  2  8 16 11 19  7 14  4 12 13 18  3 15  6  5  9 20  1
 11  7 17  3  9 15 19  1 12  8  6  5 13 20  2 14  4 16 18 10
 20  5  9 16  4 13  2 15  1 17 14 12  7 10  3 11  6 19 18  8
  6  5 10  3 18  8 20  1  4 16 11  7 14 13  2 12  9 17 15 19
 0
```

Notas

```
 185
 254
 130
 168
 0
```

Saldo de gols

```
7
7 1;2 1;1 2;2 3;2 2;1 1;5 3
8
```



```
1 4;2 3;1 1;5 5;4 4;1 2;3 6;4 2
9
9 6;5 3;3 1;5 5;5 2;2 1;1 6;1 3;1 1
0
```

Responda

q1	q2	q3	q4	n1	n2	n3	n4	s1	s2	s3

52.5 Exercício 2

Quermesse

```
 13  6 17 16  7 14  1 20  2  3 10 15 18  4  8 11 12  5 19  9
 17  5  9 16  6  4  2 11 12 15  3 19  8  7 14 18 10 13  1 20
 10  8  2 12 20 18 17 13  3  9 15  6  7 14 11  1 16  5  4 19
  5 16  6 18  7 12  4 19  1 14  9 17 13 15  3 20  8  2 10 11
 0
```

Notas

```
 255
 247
 239
 271
 0
```

Saldo de gols

```
7
9 6;1 2;2 1;2 3;1 4;2 2;5 5
7
2 4;5 1;1 1;5 5;2 3;2 1;5 3
9
6 1;4 3;3 2;2 3;5 4;1 1;2 1;1 1;3 1
0
```

Responda

q1	q2	q3	q4	n1	n2	n3	n4	s1	s2	s3

52.6 Exercício 3

Quermesse

```
  5 11  1 13  2 18 16 15  8  9  3 12  7 10 17 19 20 14  4  6
 15  8 16 10  6 18 14  7 13  2 11  5  4 17 12  3 19 20  9  1
  3 12  2 11 10  5  4 17 18  9 14 15  8 13 16  1  7 20 19  6
  9 17  4  6 13  7  2 11 20 16 15  8 18 14  5 12 10  1  3 19
 0
```

Notas

```
281
197
228
250
0
```

Saldo de gols

```
6
4 7;5 3;1 1;1 7;3 3;3 2
9
6 3;1 2;2 2;1 6;5 3;1 1;6 3;4 2;2 1
9
6 4;1 3;3 2;5 4;4 4;3 1;7 2;2 1;3 2
0
```

Responda

q1	q2	q3	q4	n1	n2	n3	n4	s1	s2	s3

52.7 Exercício 4

Quermesse

```
16 18  5 17  1  3 15 11 12 20  4  9 10 14  7  6  8 19  2 13
10 13  4 18  1 17  6 11 20  3 12 19  5  7  8 16  9  2 14 15
 2 11  7  5 19 16  1 18  8  6  4 12 17 13 10  9 15 20  3 14
 2 14 17 20  9  7 16  5  4 18 13 11  8  6 12 10  3 15 19  1
0
```

Notas

```
289
123
208
236
0
```

Saldo de gols

```
7
1 6;2 4;2 1;8 2;3 1;1 1;8 6
9
9 4;3 4;2 1;8 2;1 4;3 2;1 3;1 3;2 1
9
1 7;2 2;1 2;8 3;2 2;3 1;6 6;3 3;3 1
0
```


Responda

q1	q2	q3	q4	n1	n2	n3	n4	s1	s2	s3

52.8 Respostas

```
1   15 13 12 11  3 3 1 0  5 0 0 4  2 3 0 0  3 1 1 3  1 9  8 9  1 6
2   19 20 14 13  5 0 1 0  4 4 1 2  4 3 1 4  5 2 0 1  1 5  2 9  1 9
3   12 11 19 14  5 3 0 1  3 4 1 2  4 2 1 3  5 0 0 0  2 6  5 9  1 9
4   14 16 12 19  5 3 1 4  2 2 0 3  4 0 1 3  4 3 1 1  3 9  1 4  4 9
```

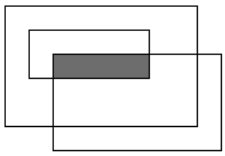
Capítulo 53

Exercício prático - 160

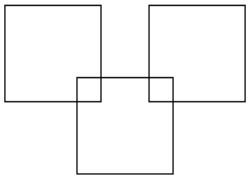
53.1 Macaco-prego

O macaco-prego é um animal irrequieto e barulhento, merecedor também dos adjetivos desordeiro e despudorado. A sua cabeça, encimada por uma densa pelagem negra ou marrom-escura, semelhante a um gorro, torna seu aspecto inconfundível. Apesar de ser o macaco mais comum nas matas do país, uma de suas sub-espécies encontra-se seriamente ameaçada de extinção: o macaco-prego-do-peito-amarelo, que se distingue das demais pela coloração amarelada do peito e da parte anterior dos braços. Um grande esforço foi feito pelos primatologistas para aumentar a população dos macacos-prego-do-peito-amarelo. Sabe-se que eles se alimentam de plantas, das quais consomem preferencialmente frutos e brotos. Alimentam-se também de muitos animais, preferencialmente lesmas, lagartas e rãs, e preferem as florestas mais densas. Para determinar o melhor local do país para criar uma nova reserva ambiental para os macacos-prego-do-peito-amarelo, o governo fez um levantamento das regiões no país onde as condições preferidas desses animais ocorrem: regiões de floresta densa, regiões com frutos, regiões com muitos brotos, etc. Ajude a salvar os macacos-prego-do-peito-amarelo.

Tarefa As regiões propícias para o macaco-prego-do-peito-amarelo foram determinadas como retângulos cujos lados são todos verticais ou horizontais. Sua tarefa é encontrar o local ideal para a reserva ambiental, definida como a interseção de todas as regiões dadas.



Conjunto de três regiões com interseção preenchida em cinza



Conjunto de três regiões com interseção vazia

Entrada Seu programa deve ler vários conjuntos de teste. A primeira linha de um conjunto de teste contém um inteiro não negativo, N, que indica o número de regiões (o valor N = 0 indica o final da entrada). Seguem-se N linhas, cada uma contendo quatro números inteiros X, Y, U e V que descrevem uma região: o par X, Y representa a coordenada do canto superior esquerdo e o par U, V representa a coordenada do canto inferior direito de um retângulo.

`Exemplo de Entrada (horizontalizados...)`

```
3               3               0
0 6 8 1         0 4 4 0
1 5 6 3         3 1 7 -3
2 4 9 0         6 4 10 0
```

Saída Para cada conjunto de teste da entrada seu programa deve produzir as coordenadas do retângulo de interseção encontrado pelo seu programa, no mesmo formato utilizado na entrada. Caso a interseção seja vazia, a segunda linha deve conter a expressão "nenhum".

```
Exemplo de Saída
2 4 6 3 e nenhum (esta saída corresponde ao exemplo de entrada acima)
```

53.2 MASP

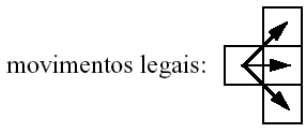
O MASP (Museu de Arte de São Paulo) tem o melhor acervo de obras de arte da América Latina, sendo reconhecido mundialmente. Além disso, o MASP é diferente de museus tradicionais porque seus quadros não são pendurados em paredes (já que as 'paredes' do MASP são janelas de vidro), mas sim apresentados em cavaletes no meio da sala de exposição. Uma grande exposição está em curso, ocupando todo o salão de exposições. Considere que:

a) as obras estão organizadas no salão de exposições no formato de uma matriz de N linhas por M colunas;

b) o visitante pode apenas mover-se da esquerda para a direita, iniciando na coluna 1 (primeira coluna) e terminando na coluna M (última coluna);

c) uma trajetória de visita é composta de uma seqüência de passos; a cada passo o visitante aprecia uma obra;

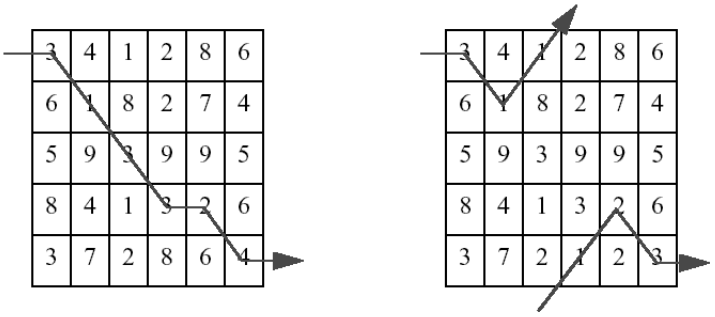
d) um passo consiste em mover-se da coluna i para a coluna i+1 em uma linha adjacente. Ou seja, o visitante pode efetuar um movimento horizontal ou diagonal. Movimentos legais são mostrados na figura abaixo:



e) para dificultar ainda um pouco mais o problema, a primeira e última linhas (linhas de número 1 e N) da matriz são consideradas adjacentes (o que não é possível no MASP!);

f) cada obra tem associada uma prioridade de visita. A prioridade é um número inteiro; quanto menor o número, maior a prioridade de que a obra seja apreciada (note que a prioridade pode ser negativa);

Tarefa Sua tarefa é escrever um programa que determine a trajetória ótima para uma visita ao museu, obedecendo as regras acima. A trajetória ótima é aquela que tem a menor soma total das prioridades das obras visitadas. Como exemplo, considere as duas exposições abaixo (a única diferença entre as duas exposições é a prioridade das obras da última linha):



As trajetórias ótimas para as duas exposições são indicadas na figura. Note que a trajetória ótima para a exposição da direita utiliza a propriedade da adjacência entre a primeira e a última linhas.

No caso de haver mais de uma trajetória ótima possível, seu programa deve imprimir a trajetória de menor ordem lexicográfica (veja o formato de Saída, abaixo).

Entrada A entrada é composta de vários conjuntos de teste. A primeira linha de um conjunto de teste contém dois números inteiros positivos N e M, que indicam respectivamente o número de linhas e o número de colunas da matriz. As N linhas seguintes contém cada uma M números inteiros que representam as prioridades das obras. O final da entrada é indicado por N = M = 0.

```
Exemplo de Entrada  (horizontalizados...)
5 6                 5 6                 2 2           0 0
3 4 1 2 8 6         3 4 1 2 8 6         9 10
6 1 8 2 7 4         6 1 8 2 7 4         9 10
5 9 3 9 9 5         5 9 3 9 9 5
8 4 1 3 2 6         8 4 1 3 2 6
3 7 2 8 6 4         3 7 2 8 6 4
```

Saída Para cada conjunto de teste da entrada seu programa deve produzir a trajetória ótima, descrita por uma seqüência de M inteiros (separados por um espaço em branco), representando as linhas da matriz que constituem a trajetória ótima. No caso de haver mais de uma trajetória ótima possível, seu programa deve imprimir a trajetória de menor ordem lexicográfica (a primeira achada).

```
Exemplo de Saída
Teste 1
1 2 3 4 4 5
Teste 2
1 2 1 5 4 5
Teste 3
1 1
(esta saída corresponde ao exemplo de entrada acima)
```

53.3 Exercício 1

Macaco prego

```
4
   3  10  15  -6
 -12  10   9   0
 -11  10   5  -1
```



```
  -2  10  13  -2
5
  -8  13   5   4
 -13   5   6 -15
  -5   5  14  -5
   0  11  10  -4
   4  14  15 -15
0
```

Masp

```
6 6
 9 2 5 9 7 8
 8 8 5 3 6 8
 4 2 4 7 9 2
 8 4 1 5 5 6
 8 1 1 8 7 9
 1 2 7 8 8 9
6 6
 6 2 4 8 1 7
 8 1 5 7 9 7
 5 7 7 3 2 4
 6 6 3 9 2 9
 1 1 3 9 8 9
 4 2 3 9 4 7
0 0
```

Responda

macaco1	macaco2	masp1	masp2

53.4 Exercício 2

Macaco prego

```
4
  -5   6   9 -13
  -2  13   5 -14
 -14  12   6  -9
  -5   8  14  -8
5
  -5   8   9 -13
 -13   8  13 -11
   3   9  11  -5
 -13   8   9 -14
  -7  14   8  -3
0
```

Masp

```
6 6
 8 4 5 8 7 5
```

```
 9 4 7 1 2 8
 1 7 6 3 3 2
 8 5 4 5 7 4
 1 5 9 9 3 9
 2 3 5 5 2 7
6 6
 6 7 4 8 7 7
 4 3 2 2 1 6
 2 5 1 5 1 1
 5 7 5 2 7 8
 4 2 5 1 8 3
 2 1 1 7 6 3
0 0
```

Responda

macaco1	macaco2	masp1	masp2

53.5 Exercício 3

Macaco prego

```
4
  -9  13  10 -11
  -4  14  13  -1
  -6  12   7 -13
 -13   5  10  -8
5
 -14  15   7 -13
  -1   6  13 -10
 -14  15  12  -9
   1  15  10  -4
 -12   5  12  -7
0
```

Masp

```
6 6
 6 2 7 5 5 7
 9 5 5 5 8 5
 9 6 9 7 2 4
 4 8 4 7 1 5
 3 9 3 6 9 6
 6 8 7 7 4 2
6 6
 6 6 2 1 2 2
 8 4 4 6 4 8
 8 4 4 3 7 1
 1 8 3 6 6 5
 2 6 2 2 6 4
 8 2 9 3 6 3
0 0
```


Responda

macaco1	macaco2	masp1	masp2

53.6 Exercício 4

Macaco prego

```
4
 -10  12   8 -13
   0   8  11  -2
 -15   9  15   4
  -6  10  15   0
5
   2  12  15   1
  -7   7   5   0
  -9   7   8 -15
  -9   8  15 -15
  -6  12  13 -12
0
```

Masp

```
6 6
 1 4 3 8 9 2
 9 8 6 4 3 7
 7 8 5 7 3 1
 5 3 6 2 5 7
 9 6 9 4 5 9
 6 4 6 4 9 5
6 6
 9 3 9 5 9 4
 5 8 9 2 6 2
 8 9 3 3 4 2
 1 1 9 5 5 5
 4 1 7 2 4 4
 8 8 9 2 3 3
0 0
```

Responda

macaco1	macaco2	masp1	masp2

53.7 Respostas

```
 1    3 10  5  0    4  5  5  4   6 5 4 4 4 3   5 5 4 3 3 3
 2nenhum           3  8  8 -3   5 6 1 2 2 3   3 2 3 2 2 3
 3   -4  5  7 -1    1  5  7 -4   1 1 2 1 6 6   5 6 1 1 1 1
 4    0  8  8  4    2  7  5  1   1 1 1 2 2 3   4 4 3 2 3 2
```

Capítulo 54

Exercício prático - 161

54.1 Anéis quadrados

Considere os seguintes anéis quadrados, cada um definido em uma matriz de 8 colunas por 9 linhas, conforme a figura abaixo. Agora coloque um anel sobre o outro, começando pelo anel 1 (que fica por baixo de todos) e terminando com o anel 5 (em cima de todos). Portanto, olhando a pilha de cinco anéis pelo lado de cima vemos o seguinte:

```
........  ........  ........  ........  .CCC....             .CCC....
EEEEEE..  ........  ........  ..BBBB..  .C.C....             ECBCBB..
E....E..  DDDDDD..  ........  ..B..B..  .C.C....             DCBCDB..
E....E..  D....D..  ........  ..B..B..  .CCC....             DCCC.B..
E....E..  D....D..  ....AAAA  ..B..B..  ........   ve-se =>  D.B.ABAA
E....E..  D....D..  ....A..A  ..BBBB..  ........             D.BBBB.A
E....E..  DDDDDD..  ....A..A  ........  ........             DDDDAD.A
E....E..  ........  ....AAAA  ........  ........             E...AAAA
EEEEEE..  ........  ........  ........  ........             EEEEEE..
  1         2         3         4         5
```

Os anéis são formados por letras maiúsculas, cada anel com uma letra diferente. O caractere ponto (‘.’) é utilizado para representrar espaços vazios. A espessura das paredes do anel é de exatamente um caractere e o comprimento dos lados nunca é menor do que três caracteres. Pelo menos uma parte de cada um dos lados do anel é visível (note que um canto conta como visível para dois lados). **Tarefa** Sua tarefa é escrever um programa que, dada a configuração de uma pilha de anéis, determine qual a seqüência de empilhamento (de baixo para cima) que foi utilizada na construção da pilha. No exemplo acima a resposta é EDABC. So há uma ordem possível de empilhamento.

Entrada A entrada é composta de vários conjuntos de teste. A primeira linha de um conjunto de teste contém dois números inteiros positivos X e Y que indicam, respectivamente, a altura e a largura da matriz que contém os anéis. As X linhas seguintes contêm Y caracteres cada, representando a vista superior da pilha de anéis. O final da entrada é indicado quando X = Y = 0.

```
Exemplo de Entrada (horizontalizado...)
9 8               10 10               0 0
.CCC....          ..AAAAA...
ECBCBB..          ..ACCCA...
DCBCDB..          ..AC.CA...
DCCC.B..          ..AC.CA...
```

```
D.B.ABAA          ..ACCCA...
D.BBBB.A          ..AAAAA...
DDDDAD.A          ..........
E...AAAA          ...BBB....
EEEEEE..          ...B.B....
                  ...BBB....
```

Saída Para cada conjunto de teste da entrada seu programa deve produzir uma lista das letras dos anéis, na ordem em que estes foram empilhados, do mais abaixo para o mais acima.

```
Exemplo de Saída
EDABC  e  (ABC ou ACB ou BAC ou BCA ou CAB ou CBA)
```

54.2 Balaio

Maria mora no interior de Minas Gerais e é especialista em fabricar balaios de junco. Os balaios de Maria são muito bem feitos e têm grande aceitação na região. Cada balaio demora exatamente um dia de trabalho para ser confeccionado: Maria começa a tecer um balaio no início do dia e no final do dia entrega o produto para um cliente. Com a crescente demanda, Maria começou a aceitar pedidos para o futuro: Narciso precisa de um balaio para o dia 10, Coronel Zoio precisa de um para o dia 4, Esmeralda para o dia 6, e assim por diante. Todos os pedidos, com as datas-limite de entrega, estão anotados computador que o filho de Maria comprou. Maria não sabe dizer não, e agora percebeu que aceitou mais pedidos do que vai conseguir produzir, se for considerar as datas-limite impostas pelos seus clientes. Alguém poderia ajudar Maria?

Tarefa Sua tarefa é escrever um programa que determine qual a melhor ordem de entrega dos balaios de Maria, de forma a minimizar a chateação total causada por eventuais atrasos na entrega. A medida da chateação utiliza um sistema de medição desenvolvido por Maria, por experiência anterior: ela sabe que se atrasar o balaio de Narciso, isso vai causar uma chateação de nível 13; Esmeralda é muito boazinha e, se Maria atrasar a sua entrega, a chateação será nível 0. No entanto, se atrasar o balaio do Coronel Zoio, ela terá uma chateação de nível 125. A chateação total é dada pela soma das chateações causadas por todos os atrasos. Considere que Maria trabalha todos os dias, sem descanso, e os dias são numerados sequencialmente a partir de 1.

Entrada A entrada é composta de vários conjuntos de teste. A primeira linha de um conjunto de teste contém um número inteiro positivo N, que indica quantidade de pedidos pendentes. A segunda linha contém o vetor de inteiros positivos L, em que L[i] que indica a data-limite para entrega do pedido i ($1 \leq i \leq N$). A terceira linha contém o vetor de inteiros positivos C, em que C[i] indica o nível de chateação ocasionado se o prazo L[i] não for obedecido ($1 \leq i \leq N$). O final da entrada é indicado por $N = 0$.

```
Exemplo de Entrada (horizontalizado)
3                  7                        0
10 6 4             4  2  4  3  1  4  6
21 0 125          10 60 50 40 30 20 10
```

Saída Para cada conjunto de teste da entrada seu programa deve produzir a chateação total mínima daquela entrada. Pode ocorrer que mais de uma seqüência de produção gere a mesma chateação mínima, mas como a resposta deve ser o valor do mínimo, este fato não tem importância.

```
Exemplo de Saída
0 e 30 (a seq. neste caso pode ser 5 2 4 3 6 7 1)
```

54.3 Meteoros

Em noites sem nuvens pode-se muitas vezes observar pontos brilhantes no céu que se deslocam com grande velocidade, e em poucos segundos desaparecem de vista: são as chamadas estrelas cadentes, ou meteoros. Meteoros são na verdade partículas de poeira de pequenas dimensões que, ao penetrar na atmosfera terrestre, queimam-se rapidamente (normalmente a uma altura entre 60 e 120 quilômetros). Se os meteoros são suficientemente grandes, podem não queimar-se completamente na atmosfera e dessa forma atingem a superfície terrestre: nesse caso são chamados de meteoritos.

Zé Felício é um fazendeiro que adora astronomia e descobriu um portal na Internet que fornece uma lista das posições onde caíram meteoritos. Com base nessa lista, e conhecendo a localização de sua fazenda, Zé Felício deseja saber quantos meteoritos caíram dentro de sua propriedade. Ele precisa de sua ajuda para escrever um programa de computador que faça essa verificação automaticamente.

Tarefa São dados:

- uma lista de pontos no plano cartesiano, onde cada ponto corresponde à posição onde caiu um meteorito;
- as coordenadas de um retângulo que delimita uma fazenda.

As linhas que delimitam a fazenda são paralelas aos eixos cartesianos. Sua tarefa é escrever um programa que determine quantos meteoritos caíram dentro da fazenda (incluindo meteoritos que caíram exatamente sobre as linhas que delimitam a fazenda).

Entrada Seu programa deve ler vários conjuntos de testes. A primeira linha de um conjunto de testes quatro números inteiros X1, Y1, X2 e Y2, onde (X1, Y1) é a coordenada do canto superior esquerdo e (X2, Y2) é a coordenada do canto inferior direito do retângulo que delimita a fazenda. A segunda linha contém um inteiro, N, que indica o número de meteoritos. Seguem-se N linhas, cada uma contendo dois números inteiros X e Y, correspondendo às coordenadas de cada meteorito. O final da entrada é indicado por X1 = Y1 = X2 =Y2 = 0.

```
Exemplo de Entrada (horizontalizado)
2 4 5 1          2 4 3 2          0 0 0 0
2                3
1 2              1 1
3 3              2 2
                 3 3
```

Saída Para cada conjunto de teste da entrada seu programa deve produzir o número de meteoritos que caíram dentro da fazenda.

```
Exemplo de Saída
1 e 2  (esta saída corresponde ao exemplo de entrada acima)
```

54.4 Exercício 1

Aneis quadrados

```
10 10
SSSSSSSS..
S.XX...S..
SSXXSSSS..
CBXXVCVV..
```



```
C.XX.C.V..
C.XX.C.V..
CCXXCCVV..
BBXXBBBB..
..XX......
..XX......
10 10
DDDDD.....
D.ZZZZZZZ.
D.ZNNNNNN.
D.ZND..BN.
D.ZND..BN.
D.ZNNNNNN.
D.ZZZZZZZ.
D...D.....
DDDDD.....
..........
0 0
```

Balaio

```
7
  3  5  3  2  1  3  1
 80 34 17 43 90 54 73
7
  1  3  2  4  2  3  1
 70 17 74 94 92 84 29
0
```

Meteoros

```
   9  26  22   6
6
  18  14
  10  23
  16  22
   8  28
  19  17
   6  11
   1  23  30   2
6
  10   7
  25  22
  12  10
  23   7
  26  27
  22  11
0 0 0 0
```

Responda

aneis1	aneis2	balaio1	balaio2	met1	met2

54.5 Exercício 2

Aneis quadrados

```
10 10
.YYYYYYY..
.YL....Y..
.YRRRRMYMM
CYMMMRMYMM
CYRRRRCY..
CYYYYYYY..
C.L...CL..
C.L...CL..
CCLLLLLL..
..........
10 10
IIIIII....
IZZWWIWW..
IVVW.I.W..
IZVDDDDDDD
IIIDII.W.D
VZVD.Z.W.D
VZVDWWWW.D
VVVD.Z...D
.ZZDDDDDDD
..........
0 0
```

Balaio

```
7
  3  3  2  1  5  5  3
 18 21 69 82 39 72 28
7
  4  2  1  4  4  5  5
 68 72 72 95 78 59 70
0
```

Meteoros

```
   6  25  26   9
6
  17  28
  23  10
  18  22
   7  26
  30  30
   8  27
   2  21  29   3
6
  27   6
  13  28
   7   2
  27  13
```



```
   5   6
  17  15
0 0 0 0
```

Responda

aneis1	aneis2	balaio1	balaio2	met1	met2

54.6 Exercício 3

Aneis quadrados

```
10 10
...LLLLLL.
.III....L.
.I.I....L.
YIOOOOOOL.
YIOOOOOOL.
.I.MM...L.
.IIMM...L.
...L....L.
...L....L.
...LLLLLL.
10 10
..TTT.....
..TNT.....
.DTFTFFF..
.DTFTH.F..
.DTFTH.F..
.DTFTH.F..
.DTTTFFF..
.D.....D..
.D.....D..
.DDDDDDD..
0 0
```

Balaio

```
7
  1  1  5  2  5  4  3
 49 87 28 62 84 39 14
7
  4  2  3  4  5  4  4
 11 12 58 63 34 27 79
0
```

Meteoros

```
   5  28  21   1
6
  25  12
  15   1
```

```
  15   6
  20  13
  15  24
  30  13
   1  25  25   4
6
   3   8
  13  26
  23  20
  30  12
   5  24
   7  30
0 0 0 0
```

Responda

aneis1	aneis2	balaio1	balaio2	met1	met2

54.7 Exercício 4

Aneis quadrados

```
10 10
VVVV......
V..V......
V.SVSSSSS.
V.KKKUUUS.
V.KVK...S.
V.KVK...S.
V.KVK...S.
V.KKK...S.
V.SVUUUUS.
VVVVSSSSS.
10 10
OOOOOOOOO.
OMMMMMIIO.
OMISSM.IO.
OMI..M.IO.
OMI..M.IO.
OMOOOMOOO.
.MI..M.I..
.MMMMM.I..
..I....I..
..IIIIII..
0 0
```

Balaio

```
7
  3  5  4  5  3  3  4
 22 63 58 96 96 78 28
```



```
7
  5  1  5  4  2  3  3
 78 74 50 36 72 17 84
0
```

Meteoros

```
   7  22  27  10
6
  11  12
  11   8
  26  21
   4  27
   8  16
   2  16
   1  28  21   5
6
  22   3
  21  25
  15  12
  29  16
   4  19
  18  14
0 0 0 0
```

Responda

aneis1	aneis2	balaio1	balaio2	met1	met2

54.8 Respostas

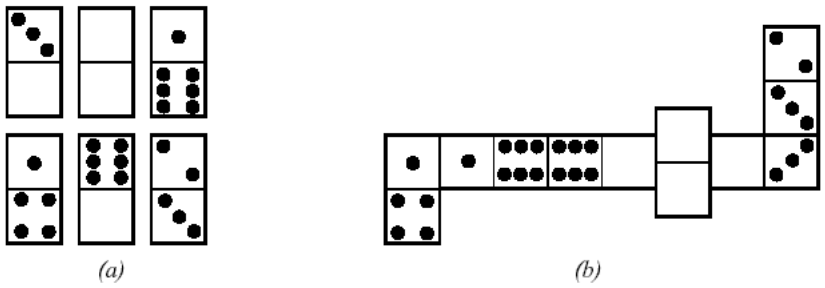
```
 1 XSCVB NZDFB 133 116  5  5
 2 YRMLC DIVWZ  39 127  3  4
 3 OMIYL TFDHN  63  23  5  4
 4 KVSHU MOSIF  50  53  4  4
```

Capítulo 55

Exercícios práticos - 162

55.1 Dominó

Todos conhecem o jogo de dominós, em que peças com dois valores devem ser colocadas na mesa em seqüência, de tal forma que os valores de peças imediatamente vizinhas sejam iguais. O objetivo desta tarefa é determinar se é possível colocar todas as peças de um conjunto dado em uma formação válida.



Conjunto de seis peças (a) e uma formação utilizando todas as seis peças (b)

Tarefa É dado um conjuto de peças de dominó. Cada peça tem dois valores X e Y, com X e Y variando de 0 a 6 (X pode ser igual a Y). Sua tarefa é escrever um programa que determine se é possível organizar todas as peças recebidas em seqüência, obedecendo as regras do jogo de dominó.

Entrada A entrada é composta de vários conjuntos de teste. A primeira linha de um conjunto de testes contém um número inteiro N que indica a quantidade de peças do conjunto. As N linhas seguintes contêm, cada uma, a descrição de uma peça. Uma peça é descrita por dois inteiros X e Y ($0 \leq X \leq 6$ e $0 \leq Y \leq 6$) que representam os valores de cada lado da peça. O final da entrada é indicado por $N = 0$.

```
Exemplo de Entrada (horizontalizado)
3 ; 0 1 ; 2 1 ; 2 1
2 ; 1 1 ; 0 0
6 ; 3 0 ; 0 0 ; 1 6 ; 4 1 ; 0 6 ; 2 3
0
```

Saída Para cada conjunto de teste da entrada seu programa deve produzir a expressão ”sim”se for possível organizar todas as peças em uma formação válida ou a expressão ”nao”caso contrário.

```
Exemplo de Saída
sim, nao e sim (esta saída corresponde ao exemplo de entrada acima)
```

55.2 Sorvete

Joãozinho é um menino que costuma ir à praia todos os finais de semana com seus pais. Eles freqüentam sempre a mesma praia, mas cada semana o pai de Joãozinho estaciona o carro em um local diferente ao longo da praia, e instala sua família em um ponto na praia em frente ao carro. Joãozinho é muito comilão, e adora de tomar sorvete na praia. Contudo, alguns dias acontece de nenhum sorveteiro passar pelo local onde eles estão. Intrigado com isto, e não querendo mais ficar sem tomar seu sorvete semanal, Joãozinho foi até a Associação dos Sorveteiros da Praia (ASP), onde ficou sabendo que cada sorveteiro passa o dia percorrendo uma mesma região da praia, indo e voltando. Além disto, cada sorveteiro percorre todos os dias a mesma região. Joãozinho conseguiu ainda a informação dos pontos de início e fim da região percorrida por cada um dos sorveteiros.

Com base nestes dados, Joãozinho quer descobrir os locais da praia onde o pai dele deve parar o carro, de forma que pelo menos um sorveteiro passe naquele local. Só que o volume de dados é muito grande, e Joãozinho está pensando se seria possível utilizar o computador para ajudá-lo nesta tarefa. No entanto Joãozinho não sabe programar, e está pedindo a sua ajuda.

Tarefa Você deve escrever um programa que leia os dados obtidos pelo Joãozinho e imprima uma lista de intervalos da praia por onde passa pelo menos um sorveteiro.

Entrada Seu programa deve ler vários conjuntos de teste. A primeira linha de um conjunto de teste contém dois inteiros não negativos, P e S, que indicam respectivamente o comprimento em metros da praia e o número de sorveteiros. Seguem-se S linhas, cada uma contendo dois números inteiros U e V que descrevem o intervalo de trabalho de cada um dos sorveteiros, em metros contados a partir do início da praia ($U < V$, $0 \leq U \leq P$ e $0 \leq V \leq P$). O final da entrada é indicado por S=0 e P=0.

```
Exemplo de Entrada (horizontalizada)
200 2 ; 0 21 ; 110 180
1000 3 ; 10 400 ; 80 200 ; 400 1000
10 2 ; 1 4 ; 5 6
0 0
```

Saída Para cada conjunto de teste da entrada seu programa deve indicar O MAIOR intervalo contíguo de praia que é servido por um sorveiteiro. É obvio que se só houver um intervalor, este é que será a resposta.

```
Exemplo de Saída
110 180 ; 10 1000 ; 1 4 (esta saída corresponde ao exemplo acima)
```

55.3 Pirâmide

Joana quer ser artista plástica, mas enquanto estuda procura trabalhos temporários durante suas férias escolares. Joana conseguiu emprego como auxiliar de almoxarifado em uma grande transportadora. A transportadora recebeu um enorme carregamento de caixas de mesma altura mas com largura e profundidades diferentes. As caixas podem ser empilhadas indefinidamente mas não podem ser deitadas em outra posição (todas têm que ser armazenadas obedecendo à indicação "Este lado para cima"). Joana é a responsável por armazenar o carregamento de caixas, e, seguindo seu senso artístico,

quer construir com as caixas a pilha mais alta possível na forma de uma "pirâmide", ou seja, uma pilha construída de tal forma que uma caixa A é empilhada sobre uma outra caixa B somente se as dimensões de A (largura e a profundidade) não são maiores do que as dimensões de B (as caixas podem ser viradas de forma a trocar a profundidade com a largura). Você pode ajudá-la?

Tarefa É dado um conjuto de caixas de mesma altura mas com largura e profundidades diferentes. Sua tarefa é escrever um programa que determine qual a pilha de caixas mais alta que Joana pode construir com as restrições acima. **Entrada** A entrada é composta de vários conjuntos de teste. A primeira linha de um conjunto de testes contém um número inteiro N que indica a quantidade de caixas do conjunto. As N linhas seguintes contêm, cada uma, a descrição de uma caixa. Uma caixa é descrita por dois inteiros X e Y ($1 \leq X \leq 15000$ e $1 \leq Y \leq 15000$) que representam os valores de cada lado da peça. O final da entrada é indicado por $N = 0$.

```
Exemplo de Entrada (horizontalizada)
3 ; 100 100 ; 1000 2000 ; 2000 500
6 ; 3 4 ; 5 7 ; 7 5 ; 1 5 ; 4 4 ; 10 2
0
```

Saída Para cada conjunto de teste da entrada seu programa deve produzir o número máximo de caixas que podem ser empilhadas na forma de uma pirâmide, conforme determinado pelo seu programa.

```
Exemplo de Saída
3 ; 4 (esta saída corresponde ao exemplo de entrada acima)
```

55.4 Exercício 1

Dominó

```
6 ; 1 3;3 1;1 3;4 5;5 2;2 6 ; 0
7 ; 1 2;0 4;0 2;6 1;3 5;6 1;3 4 ; 0
8 ; 4 0;0 6;1 4;1 4;3 0;6 6;6 5;3 1 ; 0
```

Sorvete

```
960   4;470 640;130 200;210 380;170 220
790   5; 50  70;340 430;390 500;160 240;110 120
780   6;380 460;230 270;250 400;240 420;200 250;270 310
0 0
```

Pirâmide de caixas

```
 5 ; 19 14;13 20; 8 12; 8  4;14  3 ; 0
 6 ;  3  4; 3  7; 5 10; 7 10;18  9;15 12 ; 0
 7 ;  3 19; 2  7;17  2;13 19;11  3; 8  2; 1  2 ; 0
```

Responda

d1	d2	d3	s1	s2	s3	p1	p2	p3

55.5 Exercício 2

Dominó

```
6 ; 4 1;6 6;4 5;6 1;0 2;0 3 ; 0
7 ; 1 0;3 1;3 1;4 6;0 5;5 3;0 6 ; 0
8 ; 6 0;1 4;4 0;2 6;1 2;4 1;3 1;2 0 ; 0
```

Sorvete

```
1000   4; 480 540; 100 120; 380 550; 250 410
770   5;360 390;150 190; 80 150;200 210;290 310
580   6;180 200;150 180;200 390; 40 170;200 220;100 260
0 0
```

Pirâmide de caixas

```
 5 ; 13 17; 2  5;14  6; 9  8; 1 12 ; 0
 6 ; 20  8;16  6; 3 13;11 13; 6  7;15  1 ; 0
 7 ; 11  4; 2  5;12 20;18 11; 5 19;14  5; 1 17 ; 0
```

Responda

d1	d2	d3	s1	s2	s3	p1	p2	p3

55.6 Exercício 3

Dominó

```
6 ; 3 5;6 6;4 6;5 5;4 3;6 4 ; 0
7 ; 1 5;0 2;4 4;6 4;2 1;1 0;6 4 ; 0
8 ; 3 3;0 2;1 1;4 5;1 5;5 0;1 1;3 4 ; 0
```

Sorvete

```
930   4;220 410;220 410;170 280;460 650
970   5;350 430;220 270;280 300;280 370;280 350
750   6;280 450;220 390;230 320; 70 170;360 460;150 310
0 0
```

Pirâmide de caixas

```
 5 ; 18  1; 4  9; 5  8; 9  8;11 12 ; 0
 6 ;  1  2; 5 15;13 19; 8 14;11 20; 7  2 ; 0
 7 ;  1  6;19  4;19  7; 2  8;16  8;18 17; 9  6 ; 0
```

Responda

d1	d2	d3	s1	s2	s3	p1	p2	p3

55.7 Exercício 4

Dominó

```
6 ; 2 4;5 4;4 6;6 3;2 4;4 1 ; 0
7 ; 3 4;1 1;2 3;0 5;4 3;4 3;4 3 ; 0
8 ; 4 5;0 2;4 3;6 3;6 2;1 4;0 2;2 1 ; 0
```

Sorvete

```
870   4;440 450;100 270;150 250;410 460
770   5;190 250; 20 220;350 460;370 560;250 330
920   6;170 220; 40 190;320 430;250 270;190 200; 90 180
0 0
```

Pirâmide de caixas

```
 5 ; 10  9;11 13; 9 15;11 14; 1 12 ; 0
 6 ;  1  4;18  4; 5  8;18  3; 1 16; 5  8 ; 0
 7 ;  9  8; 7 14; 1 12; 6 16;13  3;10  5; 6 10 ; 0
```

Responda

d1	d2	d3	s1	s2	s3	p1	p2	p3

55.8 Respostas

```
1     0 1 0   130 380  340 500  200 460     3 4 5
2     0 0 0   250 550   80 190   40 390     3 2 5
3     1 1 0   170 410  280 430   70 460     3 4 5
4     0 0 1   100 270   20 330   40 220     3 3 1
```

