Programação C++

P. Kantek

20 de abril de 2024

Sumário

Let us change our traditional attitude to the construction of programs. Instead of imagining that our main task is to instruct a computer what to do, let us imagine that our main task is to explain to human beings what we want a computer to do.

— Donald E. Knuth, Literate Programming

Mudemos nossa atitude tradicional em relação à construção de programas. Em vez de imaginar que nossa principal tarefa é instruir um computador sobre o que fazer, imaginemos que nossa principal tarefa seja explicar aos seres humanos o que queremos que um computador faça.

— Donald E. Knuth, Literate Programming

Capítulo 1

Introdução

Esta aula apresenta Turing e deve ser apoiada pelas apresentações:
aula_inicial_ufpr.pdf residente no diretório acima deste OU
/p/n/715/aula_inaug_enigma.pdf (com apoio de arquivo auxiliar no mesmo diretório) OU
/p/n/715/enigma_calouros.pdf
Tem um filme que pode ser apresentado em
/p/n/715/a maquina de critografia enigma
além dela programada em C, APL e Python em
/p/n/715/a maquina enigma programada em 3 linguagens.pdf
além de outras cositas mais...
Finalmente, pode se dar um exercício individual para os alunos VIVO715...

Outra alternativa é basear-se nas aulas inaugurais de Fundamentos (CI240) e/ou Métodos Numéricos (CI202) e aproveitar de lá o que for interessante. Deixa-se de repetir aqui o que está lá, para evitar duplicidade.

1.1 O que a ciência diz sobre "como se aprende"

Benjamin Bloom , atual. Escalada de conteúdos: fatos, conceitos, procedimentos, meta-cognição.

Whitehead ,1861-1947. O que for ensinado, que o seja em profundidade.

William Oesler ,1849-1919, médico canadense. Não dá para aprender medicina sem um paciente.

Aristóteles ,séc. IV a.C. aprendemos fazendo.

Capítulo 2

Computadores, algoritmos e programas

Esta disciplina visa habilitar os participantes a PROGRAMAR um computador. Esta é uma habilidade nova na humanidade, não tem 70 anos ainda.

Se você olhar para o mercado de emprego, perceba que nos últimos anos, um batalhão de escriturários, caixas, despachantes, desenhistas, analistas de crédito, motoristas (?), ... perderam seus empregos e foram substituídos por softwares, que fazem a mesma coisa que eles, sem erro (?) sem greve, sem reclamar direitos, sem cansar, com disponibilidade geográfica mundial e em um regime de 24/7.

Hora de contar a história da caçada de tigres.

Sem entrar no mérito MORAL (bem x mal) envolvido, o que se quer é garantir o meu...

A importância da computação no século XXI reside em

- Como programador profissional (uma carreira fascinante, que NUNCA terá falta de empregos). A beleza aqui é **criador de universos**
- Como engenheiro, técnico, médico, advogado, etc, etc, com habilidades de processar dados em alta escala e sem se intimidar pelo computador
- Como curioso, brincando de computar
- Abraçando uma das carreiras novas, e as que ainda virão: analista de dados, analista de IA, logístico, matemático aplicado, ...
- Participar das atividades de filhos e netos (para os quais o computador tende a ser mais amigável do que é hoje)

A melhor maneira de encarar esta disciplina é pelo que ela é: a habilidade de dominar um novo idioma (tal como inglês ou espanhol, ou chinês). Se por um lado é mais fácil (já que os elementos da linguagem são poucos) e por outro é mais difícil, já que é pura matemática aplicada.

Gosto muito da citação de Ivan Savov que diz *Não importa quanto tempo você gastou e gasta estudando matemática. Sempre é um tempo excelentemente gasto*. A nossa história começa nos anos 30, com a teoria de Alan Turing. Nos anos 50, ele começa a construir computadores em universidades inglesas. Ao mesmo tempo, nos EUA começa a montagem de computadores também: o ENIAC entre outros.

Falar aqui do IBM604

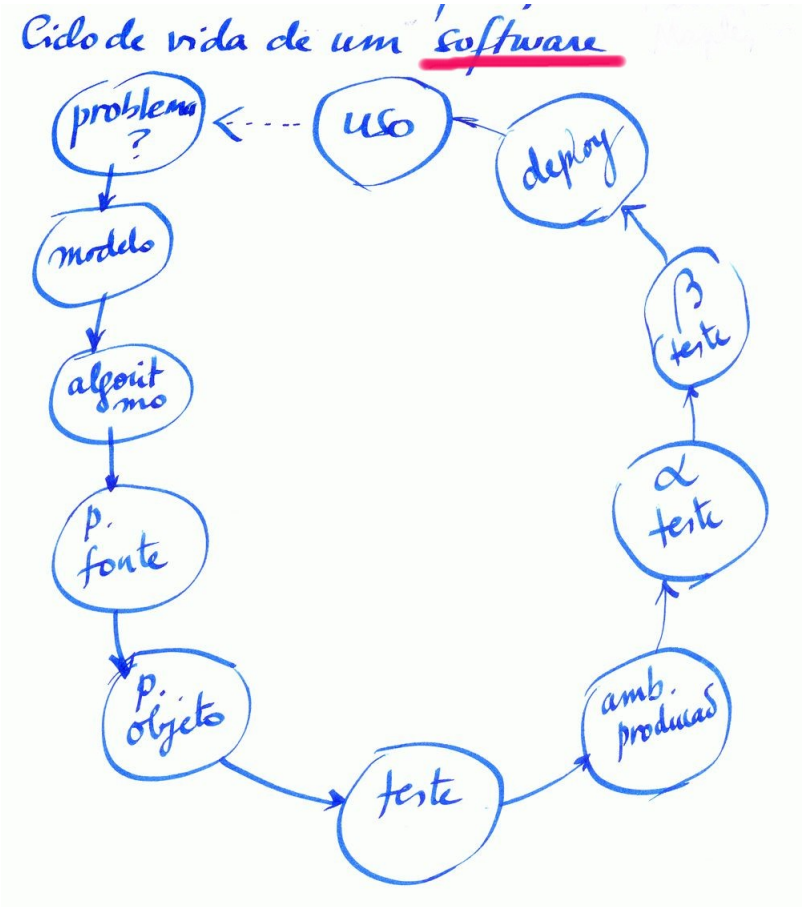
Nessa mesma época, Von Neumann lança sua assim denominada "Arquitetura de Von Neumann". A grande sacada aqui é tratar instruções como se tratam dados. Este fato simples abre a possibilidade de construir compiladores e consequentemente as primeiras linguagens de programação (Fortran, anos 50; LISP, logo depois). O surgimento das linguagens significa aprender a programar em termos de dias/semanas ao invés de meses/anos.

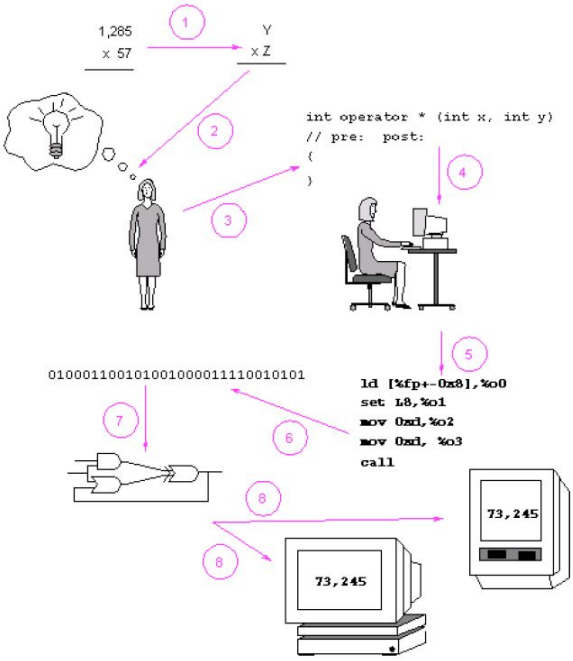
A partir de agora, uma CPU contém: UC+ULA+memória, além dos periféricos e da memória secundária.

Antes disso, havia que se dominar uma linguagem de baixo nível: dezenas (ou até centenas) de instruções de máquina, detalhes idiossincráticos de cada plataforma.

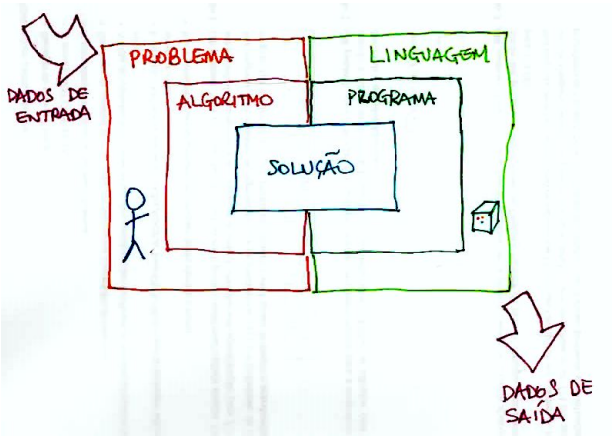
Programar é "emprestar" um pouco da inteligência humana para o computador. Salta aos olhos o fato de que para escrever um programa que resolve um problema, há que saber previamente como esse problema se resolve. (Possíveis exceções usando engenhos de IA).

Então o nossa primeira questão: como resolver o problema ? As soluções existem (e devem ser recuperadas) ou precisam ser inventadas/descobertas ? Aqui a primeira (e muitas vezes crucial) etapa: desenvolver o algoritmo. Depois a conversão desse algoritmo em uma linguagem de programação, constituindo o programa de computador. Esse programa é executado e gera (ou não) resultados. Esses resultados são criticados (a primeira resposta de um computador em geral está ERRADA). Depois de uma primeira aceitação, executa-se o teste α e depois o teste β. Depois o programa sofre um deploy e começa a ser usado em regime de produção. Quando o meio ambiente se modifica, o ciclo todo recomeça. Eis isso tudo graficamente





Outra maneira de entender onde estamos entrando:



2.1 Algoritmo

Um algoritmo é a idéia que está por trás de um programa de computador. É o que permanece igual quer você esteja programando em Python em um celular aqui na UFPR ou usando Java em um computador da NASA.

É a idéia que sobra de um programa de computador quando toda a tralha da computação é retirada. É a receita do bolo. Vejamos um exemplo: Suponha que o técnico de futebol do colégio precisa cancelar um treino anteriormente marcado:

A o técnico liga para os 45 jogadores (força bruta).

B instituí-se uma cadeia: $T \to J_1$, $J_1 \to J_2$, ..., $J_{44} \to J_{45}$.

C igual, mas exige-se que $A_{45} \to T$.

D igual, mas se A_i não responde, então $A_{i-1} \to A_{i+1}$

E árvore de chamadas: $A_1 \to A_2$ e também $A_1 \to A_3$. Daí, $A_2 \to A_4$ e $A_2 \to A_5$ e assim por diante.

F igual, mas com retorno. Depois de notificar todos abaixo, chame seu ascendente.

... ...

Primeira observação: sempre se fala em UM algoritmo e não nO algoritmo.
História da aluna no Positivo que usava resmas de papel na solução.
EFICÁCIA x EFICIÊNCIA: sempre eficiência. No passado era também eficácia, mas hoje perdeu importância.
COMO SE APRENDE ALGORITMOS ? É uma arte. Logo, como se aprende a desenhar ? ou a compor música ? Sugestões:

- estudar algoritmos ou melhor: programas
- escrever pequenos programas (complexidade crescente)
- treinar entender enunciados
- buscar intimidade com ambiente computacional

Máquina de Turing ? Será o caso de descrevê-la ? É a primeira formalização matemática do que é um algoritmo.

Teorema da incompletude de Gödel ? Será o caso de descrevê-lo ? É a dica para desistir de obter a última prova (citar a resposta para a pergunta fundamental "sobre a vida, o universo e tudo mais"). Ponha isso no google, se duvidar...

Formas de representar um algoritmo: descrição narrativa em português, fluxograma (norma americana e brasileira, desenho, extenso, difícil de atualizar), pseudo-código, um miniportuguês estruturado e restrito).

2.2 Linguagens

Fortran fórmula translation. É a avó, inclusive na disciplina. Década de 50

LISP List Processor. Usado em IA. Muito antigo também

APL Notação matemática atualizada. À frente do seu tempo (ainda)

BASIC Surgiu com os microcomputadores, década de 80

PASCAL Ferramenta para aprendizado

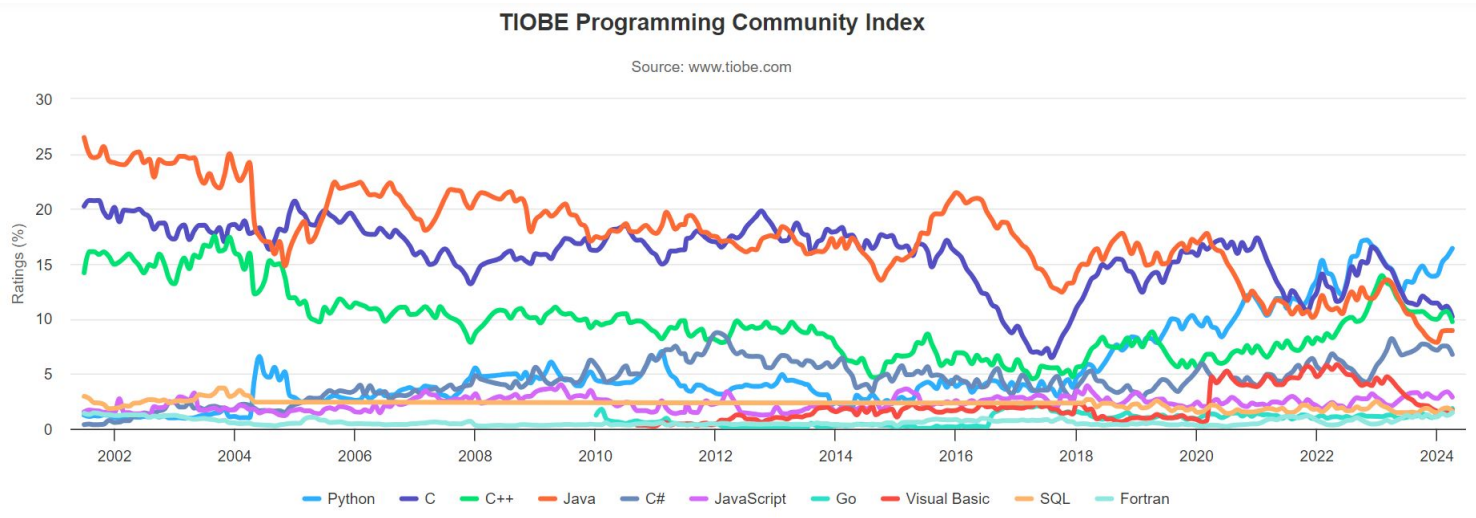
COBOL Processamento comercial. A primeira grande onda. Hoje só se mantém graças ao legado

PROLOG Processador lógico

MAPLE O primeiro processador simbólico

MATLAB Uso em cursos de engenharia

PYTHON A mais moderna. Usada em 80% das universidades americanas. E no mundo. (TIOBE)



Índice TIOBE de abril de 2024 (https://www.tiobe.com/tiobe-index/).

2.3 Variáveis

Em um programa de computador precisamos de lugares para guardar as coisas. Na matemática é igual quando se escrever $y = ax^2 + bx + c = 0$. a, b e c são nomes para valores que serão fornecidos depois (ou não serão fornecidos). Em um algoritmo/programa é igual: precisamos nomear os valores que vão ser manuseados. Então, uma variável é um pedaço de memória, que vai ter um certo conteúdo (variável) e quando este nome for citado, ele será substituído pelo conteúdo ATUAL deste pedaço de memória.

2.3.1 Criação e modificação

Uma variável é criada ou modificada pelo caracter de assinalamento. Deveria ser $\leftarrow$, mas como este caractere não existe no teclado foi substituido por =. (Em pascal é :=). Como consequência o igual deixa de ser a igualdade (em C++ é ==).

2.3.2 Nome

Uma palavra (não se admitem espaços), começando por uma letra, podendo ter letras e números e eventualmente o caracter sublinha (_). Em geral há distinção de caixa, pelo que OBA $\neq$ Oba $\neq$ oba. Por esta razão, sugere-se a criação de uma regra PESSOAL. Não se devem usar caracteres acentuados como nomes.
Exemplos bons: RAIZES, X, ALFA, B22, PORTADOR. Exemplos ruins: A, SALDOALIBERARDEPOISDOMES, 22

2.3.3 Tipo

Numérico ou alfanumérico: decidido pela presença da ASPA dupla.

```
A = 55
B = "Curitiba Paraná"
C = A'
```

Os numéricos se dividem em inteiros e reais, aqui chamados de float (de floating point) outras linguagens têm o tipo lógico (True e False), o C++ não tem: então aqui, 0 é falso e $\neq 0$ é verdadeiro.

2.3.4 Tamanho

os numéricos usam tamanhos pré-definidos: usualmente 32 ou 64 bits. Já os alfanuméricos têm o seu tamanho explicitamente estabelecidos.

Aqui é hora de olhar a SUMA02 de métodos numéricos.

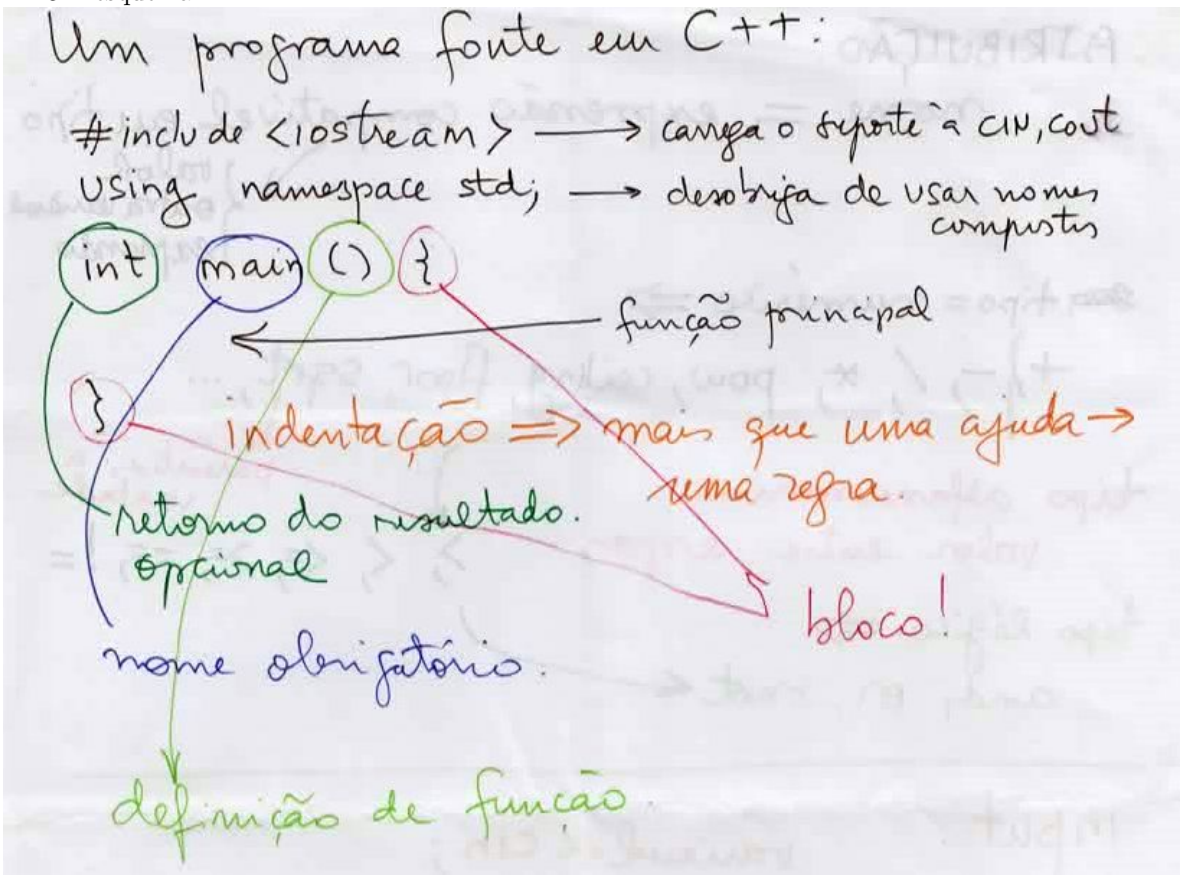
Capítulo 3

Primeiros programas

Um programa FONTE em C++ é um texto **plano** (isto é sem nenhum caracter de controle). Acompanhe cada um dos componentes de um programa

```
#include<iostream>
using namespace std;
[definições de constantes;]
definições de variáveis;
[definições de funções]
int main(){
    sentenças;
    ...
    return 0;
}
```

Um esquema:



3.1 Ambiente de trabalho

Um computador com acesso à internet (ou não). Buscar o programa CODEBLOCKS e instalá-lo.
(https://www.codeblocks.org/downloads/). Provavelmente deva ser instalada a versão contendo o mingw, que é o com-

pilador C++ necessário para a compilação de nossos programas. Na instalação deve-se escolher o sistema operacional (windows, linux ou Mac) e nestes a arquitetura disponível (32 ou 64 bits). Se você não sabe qual é sua arquitetura, (sob windows:) vá em iniciar, configurações, sobre e leia na descrição. Lá aparece com todas as letras. Se não conseguir, pergunte a alguém.

3.1.1 Tiny C Compiler

Para os corajosos ou destemidos, há um ambiente sob medida: é o *Tiny C Compiler* também conhecido como tcc. Ele mora em `https://bellard.org/tcc/` e é mínimo. Compila programas *C* (não *C++*), logo prepare-se para entradas via `scanf` e saídas via `printf`. É muito rápido. Você deve criar os fontes.c e salvá-los onde o *tcc* os ache. Muito rapidamente ele gera o fontes.exe.

C ou C++?

Algumas considerações sobre este tema

- C é mais antigo (década de 70) enquanto C++ é uma evolução obtida a partir de deficências da linguagem original e construída cerca de 15 anos após. A propósito, o nome é uma brincadeira com este fato. C++ é C mais 1.
- C++ é 100% retrocompatível com C. Então isso significa que 100% dos programas escritos em C são integralmente compilados e executados no compilador C++. Note que o contrário NÃO é verdadeiro. Um programa escrito em C++ (e usando alguma característica específica de C++) não será compilado em um compilador C.
- A principal diferença conceitual na mudança é a utilização (opcional) do paradigma orientado a objetos em C++. Este fato introduz uma mudança filosófica e tanto. Também permitiu o uso de tipos de dados abstratos (inclusive a partir de bibliotecas) – veja uma boa descrição em Skiena12.
- No que nos diz respeito (nós não vamos usar a orientação ao objeto), uma mudança bem importante é a simplificação das operações de entrada/saida no C++ quando comparadas à mesma coisa no C. Acompanhe

```
int idade;

printf("Idade: %d\n", idade);  // em C
cout<<idade;                   // em C++

scanf("%d", &idade);     // em C
cin>>idade;              // em C++
```

- Outra novidade que nos ajuda e muito é que em C++ apareceu o tipo `string` o que facilita e simplifica a manipulação de strings.
- Finalmente, C++ aponta para o futuro, razão pela qual vale a pena fazer a migração C → C++.

3.2 C++ ou Python ?

Aqui a coisa fica diferente

- C++ é uma ou duas ordens (10 a 100 vezes) mais eficiente que Python. Comentário sobre a maratona de programação.
- Python é menos idiossincrática e mais orientada ao problema: Logo é mais fácil de aprender e muito mais produtiva do ponto de vista humano.
- Há 30 anos, a maioria das universidades usava C/C++, hoje mais de 80% delas usam Python.
- O elemento raro na equação se deslocou de computador raro (anos 90) para inteligência humana rara (hoje).
- Programas modernos são desenvolvidos principalmente em Python (por exemplo o Tensorflow).
- C++ permanece imbatível em situações de uso MUITO frequente (por exemplo, a máquina de pesquisa do Google)

Compare o que se disse acima, olhando para o mesmo programa (a obtenção da hipotenusa de um triângulo retângulo a partir de seus dois catetos), uma aplicação trivial do Teorema de Pitágoras

Em C++

```
#include<iostream>
#include<cmath>
using namespace std;
float pita(float a, float b){
   return sqrt(pow(a,2)+pow(b,2));
   }
int main(){
   float ca,cb,hp;
   cout<<"Informe o cateto A: "<< endl;
   cin>>ca;
   cout<<"Informe o cateto B: "<< endl;
   cin>>cb;
   hp = pita(ca,cb);
   cout<<"A hipotenusa eh: "<<hp;
}
```

Em Python

```
def pita(a,b):
  return (a**2+b**2)**0.5
ca=float(input("Informe o cateto A: "))
cb=float(input("Informe o cateto B: "))
hp=pita(ca,cb)
print("A hipotenusa é: ",hp)
```

3.2.1 Alô mundo

Este é tradicionalmente o primeiro programa escrito em qualquer ambiente novo. Ele não faz nada, mas garante que o "caminho das pedras" da compilação e execução está dominado.

No CODEBLOCKS, escreva: File-New-File. Depois escolha C/C++ source, opção C++, e finalmente forneça o nome (completo) do seu arquivo, que deverá terminar por `.cpp`.

Na tela em branco, digite

```
#include<iostream>
using namespace std;
int main() {
   cout<<"Alo mundo"<<endl;
}
```

Depois de ter digitado exatamente o acima, clique no botão que contém uma engrenagem (compilar) atrás de um triângulo verde (executar).

Se tudo der certo, deve-se abrir uma tela preta com a mensagem "Alo mundo", o código retornado pelo programa (0, zero) e o tempo gasto na execução do programa (0.02 segundos, no meu caso).

Esta mensagem simples, atesta que você já sabe escrever, compilar e executar um programa escrito em C++. Não é pouca coisa.

Antes de prosseguir, vamos conceituar o que foi feito: `#include<iostream>` é uma diretiva de compilação (`#`) que manda incluir (`include`) a classe (módulo) `iostream` que contém os objetos `cin` e `cout` responsáveis pelas operações de entrada e saída, que serão vistas daqui a pouco. Sem este include, seu programa ficará cego e mudo. Neste ponto do curso, não é necessário nenhum detalhe a mais, basta incluir este comando *as is* e fim de papo.

O segundo comando é `using namespace std;` é para dizer ao compilador que o nosso espaço de nomes é o standard (padrão). Essa informação tem a ver na regra de formação de nomes de classes quando houver conflito de nomes entre diversas classes. Por enquanto é o que basta, ou seja você deve incluir este comando neste lugar sempre e fim de papo.

O operador `<<` é chamado operador de inserção no stream. Quando este programa é executado, o valor à direita de `<<` é inserido no stream de saída...

3.3 Sentenças

uma sentença em C++ é uma instrução ou comando, terminada por um `;` (ponto e vírgula). Uma sentença composta é formada por mais de uma sentença simples, e agora a sentença composta começa por um { e termina pelo } equivalente.

Uma providência boa e evitadora de problemas futuros é sempre limitar as sentenças sejam simples ou compostas por um { e um }. A explicação para isto: uma sentença que nasceu simples, pode evoluir mais tarde para composta. Aí essa evolução fica facilitada se já existirem as chaves.

Dois comandos podem estar na mesma linha ou em duas linhas separadas (em ambos casos separados por ;). A segunda alternativa é melhor.

A função main(){ ... } é um exemplo de sentença composta.

É claro que podem existir (e existem) sentenças dentro de sentenças dentro de sentenças. Daí a importância FUNDAMENTAL de entender e colocar corretamente as chaves. O editor do codeblocks ajuda aqui, já que quando o cursor

está em uma das chaves, a sua equivalente é iluminada. Faça a experiência. Aliás, outro detalhe importante do editor do codeblocks é o uso inteligente de cores durante a digitação. Atente para este detalhe.

3.3.1 Comentários

O compilador os ignora. Eles servem para olhos humanos. Servem para explicar uma passagem eventualmente obscura ou servem para documentar algum trecho de código. Na minha opinião, devem ser usados com alguma parcimônia e devem fugir do óbvio. Há uma corrente na programação que defende que a única documentação seja o código. Pode ser de dois tipos:

linha única é construído por um `//` e termina quando a linha acaba.

multilinha começa com `/*` e termina em qualquer linha a seguir por `*/`.

3.4 Variáveis

Como vimos na aula passada, uma variável (em C++) tem um nome, um tipo e um determinado conteúdo. Isto tudo pode ser definido pela sentença C++:

```
tipo nome [= valor];
```

`tipo` pode ser int (inteiro), float (real) ou char (alfanumérico). Este último será visto com mais detalhes no futuro. Vamos nos concentrar agora em tipos numéricos (int e float).
`nome` é um nome escolhido por você
`=` é o comando de atribuição
`valor` é a inicialização desta variável
`;` é o final da sentença
Note que `[=valor]` está entre colchetes o que sinaliza que é opcional. De fato, uma variável pode ser definida antes e inicializada depois.

Lembrar que inteiros são enumeráveis, mas têm intervalos finitos (porém grandes) de existência, e são muito mais rápidos em termos de execução. Reais não são enumeráveis mais têm intervalos muito mais largos.

Se houverem mais de uma variável de mesmo tipo elas podem ser criadas na mesma sentença como em

```
int a, b, indice;
```

Finalmente, o caso de várias variáveis, com algumas inicializadas:

```
int a,b=3,c,d=22,x;
```

Depois que a variável foi criada, ela pode ser usada em comandos e operações, seja fornecendo seu valor, seja recebendo um novo valor. A única coisa que acarreta erro de compilação é usar uma variável ANTES dela ter sido criada.

Acompanhe os tamanhos

```
#include<iostream>
using namespace std;
int main(){
   float a;
   double b;
   int c;
   long int d;
   long long int e;
   long double f;
   string s="oi mundo aqui vamos nos...";
   cout<<sizeof(a)<<endl;  // deu 4
   cout<<sizeof(b)<<endl;  // deu 8
   cout<<sizeof(c)<<endl;  // deu 4
   cout<<sizeof(d)<<endl;  // deu 4
   cout<<sizeof(e)<<endl;  // deu 8
   cout<<sizeof(f)<<endl;  // deu 16
   cout<<sizeof(s)<<endl;  // deu 32 -- tamanho padrão de apontador para string
   cout<<s.length()<<endl; // deu 26
}
```

3.5 Entrada e Saída

Programas de computador usualmente recebem dados do mundo externo (input) e geram dados para o mundo exterior (output). Existem muitas maneiras de fazer isso (input: teclado, mouse, rede com fio e sem fio, unidade de cd, disco rígido, pendrive, interpretador de fala humana, etc etc; output: monitor de vídeo, gerador de vibração no celular, rede com fio e sem fio, disco rígido, pendrive, sintetizador de fala humana, etc etc), mas neste ponto do curso, vamos nos limitar às operações de E/S (ou I/O) mais simples que são: a entrada via teclado e a saída via monitor de vídeo.

3.5.1 Entrada

o comando é `cin>>variável`, que significa que:

- O programa deve ser adormecido neste ponto até que o operador digite algo
- Quando o operador digitar `<enter>`, depois de algum conteúdo, o programa é despertado
- O conteúdo digitado é o novo valor da variável citada no comando (Tudo se passa como se o conteúdo fosse "movido" para a variável).
- A próxima sentença é executada normalmente.

3.5.2 Saída

O comando é `cout<<valor`, sendo que valor agora pode ser uma constante, uma variável ou uma expressão. Em qualquer um dos dois casos, a constante ou o conteúdo da variável citada será mostrada no vídeo. Não há interrupção nenhuma no programa e a próxima sentença é executada. Importante citar que não há salto de linha após a apresentação. Outro `cout` na sequência colará sua saída nesta. Para evitar esse problema, a solução é colocar um `<<endl` após a apresentação anterior. O endl, faz como que a linha seja saltada. Acompanhe o exemplo

```
#include<iostream>
using namespace std;
int main() {
int a=33;
cout<<"ola"<<endl<<"mundo";
cout<<"bom"<<"dia";
cout<<endl;
cout<<a;
}
```

vai imprimir

```
ola
mundobomdia
33
```

Note que tanto `cin` como `cout` devem ser escritos em minúsculo.

3.6 Um programa completo

Abra o C++ e digite:

```
#include<iostream>
#include<cmath>
using namespace std;
int main() {
  float x1,x2,delta;
  int a,b,c;
  cin>>a>>b>>c;
  delta = b*b - 4*a*c;
  x1 = (-b)+sqrt(delta)/2*a;
  x2 = (-b)-sqrt(delta)/2*a;
  cout<<"x1="<<x1<<endl;
  cout<<"x2="<<x2<<endl;
  return 0;  // opcional
}
```

Depois da compilação ter dado certo, digite: 1 <enter> 10 <enter> 3 <enter> e o programa deverá responder x1=-5.30958 e x2=-14.6904.

Observações

- Para que a chamada à função `sqrt` (square root=raiz quadrada) não dê erro, é necessário incluir o módulo `cmath` no cabeçalho do programa.
- Note os operadores numéricos usado: `+`=adição, `-`=subtração ou negativo, `*`=multiplicação, `/`=divisão, além de parênteses à vontade.
- a, b e c foram definidas como inteiro por desejo do programador (criador de universos !)
- Não se está testando se delta é positivo. Se não for a resposta será NaN que significa `not a number`, o que não é exatamente a verdade, mas enfim... Este teste virá na próxima aula.
- Funcionaria igual se você digitasse: 1 espaço 10 espaço 3 <enter>. Experimente
- O último comando `return 0;` é opcional. Se você não o escrever o retorno será 0. Se quiser retornar algo diferente de zero (sucesso), deixa de ser opcional
- Note a presença de uma informação após duas barras: isso é conhecido como comentário, é ignorado pelo compilador e é destinado a olhos humanos.
- Finalmente, o grande barato de usar o computador é a resposta a perguntas do tipo "E SE ?" Fique à vontade e faça experiências. Pense num estudante de medicina com uma ferramenta destas.
- Veja que às vezes o botão compilar-rodar está desabilitado. Isso ocorre enquanto a janela (preta) de execução está aberta. Ela precisa ser fechada para o botão ser reabilitado.

3.6.1 atribuição

Usa o sinal de `=` para criar ou alterar o valor de uma variável. Quando em `A=B` transfere o valor de `B` para a variável `A`. Este é o comando para manipular variáveis e expressões em geral. Seu formato é uma das 3 seguintes formas:

nomvar = valvar Neste caso, a variável de nome `nomvar` é criada (se não existir) ou tem o seu valor atualizado (se já existia). Em ambos os casos, o novo valor é `valvar`. Veja o exemplo

```
int a, b, c;
float x, y, z;
char n,m;
a = 2;
x = 3.1415;
n = 'a';
```

nomvar = outrvar Agora, precisa existir uma variável prévia de nome `outrvar` e o valor dela será copiado sobre a variável `nomvar` que pode ou não existir previamente. Em uma linguagem informal diz-se que o valor de outrvar é "movido" para nomvar. Exemplos

```
int a, b, c;
float x, y, z;
char n,m;
a=1;
c=a;
x=a;
y=3.14;
b=x;
```

nomvar = expressão compatível Neste caso, a `expressão compatível` é qualquer conjunto de variáveis, operadores, constantes, parênteses, funções, etc que após resolvido tem o seu resultado colocado em `nomvar` que pode ou não existir previamente. Veja os exemplos

```
int a, b, c;
float x, y, z;
char n,m;
a = 1;
a = a+1;    //também pode ser a++;
b = sin(a)+1;
c = (sqrt(a+2)*7)-3;
```

3.7 Erros

Um programa de computador nasce, vive e morre com erros. O processo formado por: busca, isolamento, correção e certificação é perene e ocupa mais de 80% do tempo de trabalho de qualquer programador. Acostume-se a aprender com os erros cometidos. Uma tentativa de classificar os erros poderia ser:

ignorância este é o erro mais grave na nossa lista. Eventualmente ele pode impedir a continuação do processo.

de compilação tipicamente é o erro de síntaxe. Geralmente este é o mais simples de corrigir, já que é sinalizado pelo compilador, por exemplo

- falta de ponto de vírgula ao final do comando
- falta de parênteses em uma condição
- escrever `inte` ao invés de `int`
- milhares de coisas parecidas...

Pode haver alguma sutileza aqui: nem sempre o compilador é "inteligente" para isolar o erro. Às vezes ele sinaliza uma linha errada e o erro cometido está antes dela. Os erros de compilação são assinalados " mais ou menos " pelo Codeblocks com um retângulo vermelho. Na janela de baixo, há uma mensagem explicativa.

de execução aqui começam os erros semânticos: Um bom exemplo é a divisão a/b. Se $b = 0$ tem-se um erro de execução. O compilador não tem como saber durante a compilação que este erro vai aparecer. E, pior ainda este erro pode ser intermitente. Outro exemplo é usar uma variável sem ter inicializado ela antes. Um eventual erro só vai aparecer se a área de memória alocada tiver um conteúdo inadequado. Se por azar essa área contiver zeros, tudo tende a funcionar.

de lógica ou processamento errado. Imagine que em um cálculo era para escrever $2 * a$ e o programador por engano escreveu $2/a$. É óbvio que há um erro aqui, mas às vezes este erro também é intermitente, pois este comando pode estar dentro de uma condição ou ele próprio pode envolver condições...

de loop infinito Imagine um ciclo formado por

```
A=5;
while (A<10){
   A=A-1;
}
```

Claramente este programa não vai acabar nunca. O sintoma é que o programa parece demorar demais. Cabe uma análise se esta demora era esperada (e quanto deve-se aguardar). O erro só vai aparecer quando você cancelar o programa (CTRL-C) e analisar o código.

3.7.1 Para treinar

Dada uma tarefa e um código, localize o erro e a correção em cada caso:

1. Achar a média de 6 números lidos

```
float b=0; int a,c;
for(a=1;a<6;a++){
     cin>>c;
 b=b+c; }
cout<<b/6;
```

Resposta: só está lendo 5 números.

2. Achar a soma de 10 números lidos

```
float b=0; int a,c;
for(a=0;a>10;a++){
     cin>>c;
 b=b+c; }
cout<<b;
```

Resposta: Não lê nenhum número.

3. Escrever uma variável

```
cout<A;
```

Resposta: `cout<<A;`

4. Escrever uma variável

   `cout>>A;`

   Resposta: `cout<<A;`

5. Verificar se o eleitor é obrigado a votar

   `if (18 < idade < 70)`

   Resposta: A condicional correta é `((18 <= idade) && (idade < 70))`

6. Hoje é o dia 23 ?

   `if (hoje = 23)`

   Resposta: Não se está testando e sim assinalando. O correto é `if (hoje == 23)`

3.8 Aritmética

O computador sabe e conhece todas (todas!) as regras da matemática que todos nós estudamos nos últimos 12 anos. Ainda bem, senão este curso de programação ia demorar anos. Veja a seguir uma tabela das operações que podem ser comandadas dentro de um programa C++

operação	descrição	exemplo
$A+B$	adição de A e B. Se um deles é float o resultado é float (*)	$4.1+3$ é 7.1
$A-B$	subtração $A-B$. Se A é menor que B o resultado é negativo (*)	$1-3$ é -2
$-A$	menos unário: troca o sinal de A	$-A$ é o valor de A com sinal trocado
$A*B$	multiplicação: ... (*)	$3*5$ é 15
A/B	divisão inteira: se A e B são inteiros é a divisão inteira	$10/3$ é 3
A/B	divisão real: se um (ou ambos) são float, a divisão é real	$10.0/3$ é 3.33333
$A\%B$	resto da divisão inteira de A por B. Se um deles é float tem-se um erro	$10\%7$ é 3
()	determina prioridade das operações	$(3+4)*5$ é 35 e $3+(4*5)$ é 23
$sqrt(A)$	retorna a raiz quadrada de A como um float, independente do tipo de A. Precisa do comando `#include<cmath>(**)`	$sqrt(10)$ é 3.16228
$pow(A,B)$	Calcula A^B. Tanto A quanto B podem ser inteiros ou float. (**) Obviamente a raiz quadrada pode ser obtida com $pow(A,0.5)$	$pow(2,3)$ é 8
$sin(A)$	Devolve o seno de A como um flutuante. A pode ser inteiro ou float, e SEMPRE é medido em radianos. As outras funções trigonométricas são `cos(A)`, `tan(A)`, `asin(A)`, `acos(A)`, `atan(A)` (**)	$sin(1)$ é 0.841471
$sinh(A)$	Toda a família hiperbólica: `cosh`, `tanh`, `asinh`, `acosh` e `atanh`	
$log(A)$	Logaritmo neperiano (base e) de A. A pode ser int ou float, mas o logaritmo é sempre float (**)	$log(10)$ é 2.30259 $\log_e b=c \Rightarrow e^c=b$
$log10(A)$	Logaritmo decimal (base 10) de A. A pode ser int ou float, mas o logaritmo é sempre float (**)	$log10(2)$ é 0.30103 $\log_{10} b=c \Rightarrow 10^c=b$
$exp(A)$	Devolve e^A como um float (**)	$exp(1)$ é 2.71828
$abs(A)$	Devolve o módulo (valor absoluto) do inteiro A	$abs(-3)$ é 3
$round(A)$	Devolve o valor arredondado de A como inteiro. A (**)	$round(2.6)$ é 3
$trunc(A)$	Elimina a parte fracionária do número	$trunc(3.1)$ é 3
$max(A,B)$	Retorna o maior dos 2. Ambos devem ter o mesmo tipo. Há também `min(A,B)`	$max(2,3)$ é 3
$floor(A)$	é o `chão` de A (**)	$floor(2.9)$ é 2
$ceil(A)$	é o `teto` de A (**)	$ceil(2.1)$ é 3

3.9 Estudando programação

Como já dizia Aristóteles (no livro Ética) *Tudo o que precisamos aprender, aprendemos fazendo....* Na programação não é diferente. Mas, aqui temos uma grande vantagem: um corretor justo, incansável, gratuito, paciente e sempre disponível. Trata-se do computador. Você pode produzir código e o computador vai lhe dizer se o seu código está certo ou não.

Listei abaixo 3 opções onde você pode estudar programação. A mais amigável é a OBI, mas nos últimos dias o site estava fora do ar. Deve voltar em breve, mas as outras duas; UVA e Euler também são lugares legais.

3.9.1 OBI

OBI é a Olimpíada Brasileira de Informática. É uma iniciativa sensacional para ajudar estudantes brasileiros a se desenvolverem na programação. Qualquer um pode se beneficiar do corretor automatizado de programas. Basta entrar no site da OBI (https://olimpiada.ic.unicamp.br/pratique/) e escolher uma definição à direita na tela. Há dezenas de definições bastante bem escritas e com exemplos de entradas e saídas – o que é uma excelente maneira de entender o que o programa deve fazer. Entendida a encomenda, você deve escrever seu programa (no caso particular de Python, nem programa precisa ser, basta que seja um script funcional. Em C++ basta mandar ver). Uma vez salvo em algum lugar conhecido por você (diretório e subdiretório do seu disco) e com um nome único, basta escolher na tela a linguagem que você usou e o programa que escreveu. Ao clicar em SUBMETER o seu programa será lido, compilado e executado um determinado número de vezes, com outros dados de entrada diferentes daqueles que estão na definição.

Após alguns segundos, a aplicação WEB retorna uma resposta que pode ser

```
Resultado da submissão
TAREFA: Álbum da copa
LINGUAGEM: Python3
Compilação correta
Fase de testes -- Tempo Limite para cada execução: 500 ms
Teste 1: ..... (20 pontos)
Teste 2: ..... (20 pontos)
Teste 3: ..... (20 pontos)
Teste 4: ..... (20 pontos)
Teste 5: ..... (20 pontos)
Total: 100 pontos (de 100 possíveis)
Legenda:
.: resultado correto              X: resultado incorreto
E: erro em tempo de execução      M: Referência a memória inválida
S: programa não gerou saída       T: tempo limite excedido
V: violação de recursos
```

3.9.2 UVA

Instalado na Universidad de Valladolid, Espanha (daí o nome UVA) trata-se de um engenho de correção de solução de problemas algoritmicos funcionando no modelo das maratonas. O site é `https://uva.onlinejudge.org/`.

O site foi criado em 1995 por Miguel Angel Revilla, um professor de algoritmos da U. Valladolid. O site entrou em ação para o público em geral em Abril de 1997. Em 2007 o sistema foi migrado (software e hardware) para melhores condições.

O interessado pega uma definição de problema de programação, (há mais de 5000 no site), escreve um programa (em C, Java, Python ou C++) que o resolve e submete ao site o programa fonte que escreveu.

Para auxiliá-lo nesta tarefa o site UVA entrega, para cada programa da lista, um conjunto preparado de dados de entrada e associados a cada um, o conjunto de dados de saída que o programa supostamente deveria produzir. Com essas 3 coisas: definição, dados de entrada e dados de saída esperados, o candidato a programador tem tudo o que precisa para escrever e enviar ao site o programa pedido.

O site, ao receber tal programa, compila-o, e o executa com uma instância nova (inédita) de dados.

Os resultados que o site entrega alguns segundos após a submissão são:

OK o programa que você submeteu foi rodado com dados inéditos e produziu exatamente o resultado esperado.

Limite tempo O programa não terminou no intervalo de tempo a ele alocado. Ou o programa está em loop devido a algum erro nas estruturas de repetição, ou o que é menos comum, é necessário um algoritmo mais eficiente para atender a este problema.

Erro de compilação O nome diz tudo, o site não conseguiu compilar o programa que você submeteu. Ou ocorreu um equívoco na linguagem selecionada ou a linguagem instalada no seu computador não é compatível com aquela usada pelo site. Ou ainda, o que é mais frequente, você cometeu um erro de escrita no seu código.

Resultado diferente O seu programa funcionou, mas produziu resultados distintos dos que o site esperava para aquele problema. Em outras palavras, sua solução é incorreta. A chave para resolver este tipo de erro é uma leitura cuidadosa da definição, sobretudo nos detalhes (chamadas condições de contorno).

Muito importante: na última instrução executada pelo programa em C ou C++, deve-se retornar um 0. Sem isso, o programa acusará erro.

3.9.3 Projeto Euler

Este é outro esquema muito interessante para aprender a programar e se desenvolver na nobre arte. Ao contrário da UVA, no projeto Euler não se envia um programa e sim um resultado numérico que geralmente é difícil de se obter. O site é http://projecteuler.net.

Aqui, há razões de ordem prática para se buscar algoritmos e programas eficientes. Alguns pedidos se programados sem cuidado podem demorar meses ou anos em um computador comum. Ninguém precisa esperar tanto e segundo os criadores do site, nenhum problema demanda mais do que 1 segundo de CPU.

A questão é que para muitos problemas, tal limite de tempo impõe severas restrições e especialização no algoritmo procurado.

Um caso real que aconteceu comigo. Ao programar um problema (o 104 na lista do Euler) que diz

A sequencia de Fibonacci é definida pela relação de recorrência $F_n = F_{n-1} + F_{n-2}$ *onde* $F_1 = 1$ *e* $F_2 = 1$. *Deve-se notar que* F_{541} *que contém 113 dígitos é o primeiro número de Fibonacci para o qual os últimos 9 dígitos são pandigitais 1-9 (ou seja contém todos os dígitos de 1 a 9 não necessariamente em ordem). E,* F_{2749} *que contém 575 dígitos é o primeiro número de Fibonacci para o qual os primeiros 9 dígitos são pandigitais 1-9. Dado* F_k *que é o primeiro número de Fibonacci para o qual os primeiros 9 dígitos E os últimos 9 dígitos são pandigitais 1-9, ache* k.

Minha primeira opção demorou 5 dias rodando para achar o resultado (certo). Essa demora se justificava pois lida-se com números de Fibonacci com mais de 5000 dígitos cada um. Daí, quebrei a cabeça para otimizar o programa e alguns dias depois cheguei a uma versão equivalente que demorou menos de 1 segundo de tempo de relógio.

Seja como for, como você só vai fornecer um resultado, não importa quanto vai demorar, nem qual linguagem/compilador ou ambiente operacional você vai usar. Em tese, poderia até usar lápis e papel apenas. Há usuários cadastrados usando mais de 100 linguagens diferentes na obtenção das soluções.

O site não dá nenhuma dica ou auxílio até você resolver certo o problema. Nessa hora você ganha acesso a uma lista de discussão de todas as pessoas que já resolveram o mesmo problema. Muita da otimização posterior vem das idéias postadas nessa lista.

Alguns problemas são muito difíceis. Pode-se dizer que são adequados apenas para poucas centenas de habitantes deste planeta. A coisa é, como se dizia quando eu era pequeno, *às veras*.

Para treinar, entre no site Euler e cadastre-se fornecendo nome e senha. Agora você tem uma conta no servidor e pode começar a submeter problemas. Escolha o problema 1, cuja definição é *Se listarmos todos os números naturais abaixo de 10 que são múltiplos de 3 ou 5, obteremos 3, 5, 6 e 9. A soma desses múltiplos é 23. Ache a soma de todos os múltiplos de 3 ou 5 abaixo de 1000.*

Tente escrever um programa de computador que ache a resposta.

Se não conseguir, pode oferecer a resposta certa que é 233168.

Pronto, você já é membro da coletividade mundial do projeto Euler.

Capítulo 4

Uma primeira prática para o aluno fazer

4.1 (IMC)

Escreva um programa que calcule o IMC (=índice de massa corpórea, informa se alguém é obeso). Descubra como ele é calculado e implemente o algoritmo. O programa deve receber peso e altura e imprimir o IMC. Os dados são o peso, medido em kilogramas e a altura medida em m. (descubra qual a fórmula).

```
#include<iostream>
using namespace std;
int main(){
    float peso;
    float altura;
    float imc;
    cout<<"peso = ";
    cin>>peso;
    cout<<"altura = ";
    cin>>altura;
    imc=peso/(altura*altura);
    cout<<"IMC: "<<imc;
    return 0;
}
```

4.2 (Média 3 notas)

Suponha que você acabou de saber sua nota da prova 3 desta disciplina. Sabe também as notas P1 e P2. Escreva um programa que receba as notas P1, P2 e P3 e imprima a média final, usando o seguinte algoritmo:

$$res = ((P1 * 2) + (P2 * 3) + (P3 * 4))/9.0$$

```
int main(){
    float p1,p2,p3,me;
    cin>>p1>>p2>>p3;
    me=((p1*2)+(p2*3)+(p3*4))/9.0;
    cout<<"media = "<<me;
    return 0;
}
```

4.3 (Medida de velocidade)

Escreva um programa que receba um número real indicando uma velocidade medida em km/h e imprima a velocidade equivalente em m/min e depois em cm/seg.

```
int main(){
    float vel,mm,cs;
    cin>>vel;
    mm=(vel*1000)/60;
    cs=(mm*100)/60;
```

```
    cout<<"em m/min = "<<mm<<"     em cm/seg = "<<cs;
    return 0;
}
```

4.4 (Concentração iônica)

Escreva um programa que receba a concentração de ions H^+em uma solução e imprima o pH dessa mesma solução. A entrada está referida em mol/L do composto dissolvido na solução. Lembre que o $pH = -\log_{10}(H^+)$. O logaritmo negativo indica simplesmente quantas vezes um número deve ser dividido por 10 até chegar em 1. Por exemplo, se uma concentração é de $1,05 \times 10^{-5}M$ escreva a equação como $pH = -\log_{10} 1.05 \times 10^{-5} = 4.9788$. Note que neste exemplo a concentração entra como 0.0000105.

```
#include<iostream>
#include<cmath>
using namespace std;
int main(){
   float conc,ph;
   cin>>conc;
   ph=-log10(conc);
   cout<<"pH = "<<ph;
   return 0;
   }
```

4.5 (imposto em wiskie)

Suponha que a receita federal mede os rendimentos em litros/uisque 12 anos. Neste exemplo, os primeiros 10 litros do rendimento são isentos. Os seguintes 10 litros pagam 5% de I.R. Os seguintes 10 litros pagam 10% de i.r. e finalmente, qualquer rendimento que ultrapasse 30 litros pagará 25% de imposto. Escreva um programa que receba um certo rendimento medido em litros e imprima o imposto devido também medido em litros (números reais). Alguns exemplos: 35 e 2.75; 14 e 0.2; 100 e 19; 1000 e 244; 10 e 0; 11 e 0.05

```
int main(){
   double gent,g1,g2,g3,g4,ir;
   cin>>gent;
   g1=min(10.0,gent);
   g2=max(0.0,min(10.0,gent-10));
   g3=max(0.0,min(10.0,gent-20));
   g4=max(0.0,gent-30);
   ir=(0.05*g2)+(0.1*g3)+(0.25*g4);
   cout<<g1<<"  "<<g2<<"  "<<g3<<"  "<<g4<<"  ir="<<ir<<endl;
   }
```

4.6 (Cosseno)

O cosseno de um ângulo pode ser calculado pela série

$$cos(x) = \sum_{n=0}^{\infty} \frac{(-1)^n}{(2n)!} x^{(2n)}$$

Como ainda não sabemos repetir genericamente na programação (vem logo, logo), vamos nos limitar a somar 3 parcelas. Então escreva um programa que receba o valor do ângulo x (medido em radianos), calcule o cosseno desse ângulo, usando 3 parcelas da soma acima e imprima o resultado. Lembre que esta fórmula vale para $-\pi \leq x \leq \pi$. Na verdade a fórmula é válida para $-\infty \leq x \leq \infty$, mas o período da função coseno é 2π, ou deslocando de π fica $-\pi \leq x \leq \pi$.

```
#include<iostream>
#include<cmath>
using namespace std;
int main(){
   float x,p1,p2,p3;
   cin>>x;
   p1=1;
   p2=(-1*pow(x,2))/2;
```

```
   p3=(pow(x,4))/24;
   cout<<"coseno="<<p1+p2+p3<<endl;
   cout<<"para comparar "<<cos(x);
}
```

4.7 (Compressão de BLOB)

O índice de compressão de um BLOB (binary large object, pode ser um arquivo PDF, uma imagem JPG ou uma música MP3, ou qualquer outro objeto binário) é dado por

$$i = 1 - \frac{depois}{antes}$$

depois que um algoritmo comprime um objeto que tinha tamanho *antes* e ficou com um tamanho *depois*. Escreva um programa que receba o tamanho antes e o tamanho depois (ambos medidos em bytes) e imprima o índice de compressão.

```
int main(){
    float an,de;
    cin>>an>>de;
    cout<<"Indice = "<<1-(de/an)<<endl;
}
```

4.8 (População de bactérias)

Uma certa população de bactérias diminui $t\%$ ao dia. Escreva um programa que receba P (população em milhares de indivíduo) e t em percentagem de mortes/dia. O programa deve imprimir qual a população remanescente após 4 dias, supondo que não houve nenhum "nascimento".

```
int main(){
   float pop,pmo;
   cin>>pop>>pmo;
   pop=pop-(pmo/100)*pop;
   cout<<"2="<<pop<<endl;
   pop=pop-(pmo/100)*pop;
   cout<<"3="<<pop<<endl;
   pop=pop-(pmo/100)*pop;
   cout<<"No quarto dia, a populacao e "<<pop;
}
```

4.9 (Salada de frutas)

Uma salada de frutas industrial é produzida com Laranja, Mamão, Abacaxí e Banana, na seguinte proporção: 1 kg de laranja : 1.5 kg de mamão : 0.5 kg de abacaxi : 2 Kg de banana, para produzir 5 kg de sala de fruta. Escreva um programa que receba as quantidades disponíveis de L, M, A e B e imprima a quantidade de salada que pode ser produzida. Todas as quantidades estão dadas em kilogramas.

```
int main(){
   float L,M,A,B;
   float tl,tm,ta,tb;
   cin>>L>>M>>A>>B;
   tl=L/1.0;
   tm=M/1.5;
   ta=A/0.5;
   tb=B/2.0;
   cout<<min(min(min(tl,tm),ta),tb)*5.0;
}
```

4.10 (Eclipse lunar)

Está previsto para o próximo dia 14 a ocorrência de um eclipse lunar. Em nossa região, ele deve começar em algum momento entre 21h00 e 23h45 do dia 14 e terminar entre 03h05 e 05h41 do dia 15. A variação de duração e de início e final depende do lugar onde você estará. Escreva um programa que receba a hora (e minuto) do início e também a hora (e minuto) do final. Seu programa deve imprimir a duração em minutos do eclipse nesse local.

```
int main(){
   int hi,mi,hf,mf;
   int p1,p2;
   cin>>hi>>mi>>hf>>mf;
   p1=(hi*60)+mi;
   p2=(24*60)+(hf*60)+mf;
   cout<<"Duracao "<<p2-p1<<endl;
}
```

4.11 (Calças jeans)

Uma calça jeans masculina de um certo tamanho usa $1.7m^2$de brim e pode ser vendida por M reais. Uma calça similar, mas feminina usa $1.58m^2$ e pode ser vendida por F reais. Dada uma certa peça (retalho) de jeans ela poderá ser usada para fazer qualquer número de um ou outro tipo de calça, mas não ambas no mesmo retalho. As sobras após o corte não têm valor. Suponha ainda que o mercado absorve tudo o que for produzido. Escreva um programa que receba a área de um retalhe de jeans, (medido em m^2) os valores M e F medidos em reais e imprima quanto será faturado na produção e venda de calças masculinas e quanto será faturado na produção e vendas de calças femininas.

```
int main(){
   float ret,fm,ff,pem,pef;
   cin>>ret>>fm>>ff;
   pem=ret/1.7;
   pef=ret/1.58;
   cout<<"Masculinas = "<<fm*pem<<endl;
   cout<<"Femininos  = "<<ff*pef;
}
```

Capítulo 5

Condicional

5.1 Expressões relacionais

As expressões relacionais envolvem comparações e são `==`, `!=`, >, <, `<=`, `>=` entre outras. Elas são muito importantes neste capítulo da programação pois determinam se um bloco de comandos será ou não executado. Uma expressão relacional sempre resulta um de dois valores: `VERDADEIRO` (ou `TRUE`) ou `FALSO` (ou `FALSE`). Valem aqui as 2 regras da lógica binária clássica:

Não contradição Uma condição é verdadeira ou falsa, n˜ão podendo ser as duas coisas ao mesmo tempo.

Terceiro excluído Se uma condição não for falsa ela será verdadeira e vice-versa.

Deve-se ressaltar que o C++ (ao contrário de outras linguagens como por exemplo Python, mas igualmente como outras linguagens como APL) não possui o tipo Booleano (de George Boole) para representar condições. Mas apresenta uma regra que é facilmente entendida, a saber:

> Uma expressão valendo 0 (zero) será entendida como FALSA enquanto uma expressão com qualquer valor diferente de zero será entendida como VERDADEIRA. A expressão pode ser inteira ou flutuante.

5.2 Expressões lógicas

Quando se necessitar uma relação composta (por exemplo: *gosto de sorvete e de amendoim* ou ainda *vou ao cinema quando chove ou quando o filme é bom*) precisa-se de conectores lógicos, que são em número de 2 ou 3, vá lá. São eles `and(&&)`, `or(||)` e `not`. Tanto as expressões relacionais como as lógicas devolvem valores numéricos que devem ser interpretados como 0=false e $\neq 0$=true. Acompanhe nos exemplos

```
int a=1, b=2, c=3;
float x=1.0, y=2.0, z=3.0;
cout<<(a>=b); // deu 0
cout<<(b>=a); // deu 1
cout<<(x>=a); // deu 1
cout<<((c>=3)&&(y<=1)); // deu 0
cout<<((5!=6)||(y>0));  // deu 1
```

5.2.1 Expressões lógicas

A resposta a expressões lógicas leva em consideração as seguintes tabelas verdade:

```
AND   0   1            OR    0   1       NOT
---|---------          ---|---------      ---|-------
 0 |  0   0            0 |  0   1         0 |   1
 1 |  0   1            1 |  1   1         1 |   0
```

Muita calma nessa hora. Uma expressão lógica conecta 2 expressões relacionais usando um dos conectivos vistos acima: `&&` (e), `||` (ou) ou `not` (não). Usam-se expressões lógicas quando se tem condições compostas. Por exemplo:

- Você estará aprovado ao final do curso se (e sse) tiver nota maior ou igual a 7 e frequência maior que 75%. Eis a expressão que indica isso em um programa C++: `(nota>=7) && (freq>75)`.
- Um eleitor no Brasil, pode votar se tiver `idade>=16`, mas é obrigado a votar se `(idade>=16) && (idade<70)`.

- O preço de um chope é razoável se `(chope>10) && (chope<20)`.
- Você é considerado prioritário se `(idade>=65) || (especial==1)`. Note que o valor 1 foi usado como sinônimo de `sim`.
- O mês tem 30 dias ?: `(mes==4) || (mes==6) || (mes==9) || (mes==11)`
- O ano é bissexto ?: `((ano%4)==0) && (((ano%100) != 0) || (ano%400 == 0))`. Perceba nesta última condição composta que elas podem rapidamente se tornar bem complexas. Perceba também que os parênteses que eram usados nas expressões aritméticas, podem e devem ser usados nas expressões relacionais e lógicas com o MESMO significado.

5.3 Condicional

Este é o comando que dá "inteligência" (?) ao computador. Ele estabelece que o programa deve realizar alguma coisa SE uma determinada condição for verdadeira. É uma situação que nós já conhecemos: na matemática real, uma equação do segundo grau só terá raízes reais SE o valor conhecido por Δ for positivo (lembram?) Pensem num alarme digital em uma caldeira. Ele deve soar SE a temperatura ultrapassar um determinado limiar. Pense num desconto em uma compra no mercado. Ele deve ser lançado SE o cliente comprar 3 ou mais unidades de um produto. Pense na cobrança do estacionamento de um shopping. O valor deve ser cobrado SE o tempo de permanência for superior a 15 minutos.

O comando condicional tem o formato (em C++):

```
if (condição) {
  bloco condicional
}
```

Conforme vimos, neste caso, a condição será avaliada: se verdadeira o bloco que está entre as chaves será executado. Se a condição for falsa, nada é executado. Note que a presença (e o local) das chaves é muito importante, já que elas sinalizam onde começa e acaba o bloco condicional.

Perceba também que a condição em C++ precisa estar dentro de um par de parênteses. Outra modalidade de comando condicional é aquele chamado composto, cujo formato é

```
if (condição) {
  bloco condicional
}
else {
  bloco para a condicional negada
}
```

Note que agora existem 2 blocos de comandos: o condicional e o condicional negado. Se a condição é verdadeira o primeiro bloco é executado (e o segundo não). Se a condicional é falsa, o segundo bloco é executado (e o primeiro não). Pode-se considerar os 2 formatos e finalmente constituir o formato único que é

```
if (condição) {
  bloco condicional
}
[else {
  bloco para a condicional negada
}]
```

onde os colchetes sinalizam que o seu interior é opcional. Veja alguns exemplos:

```
a = 1; b = 2; c=3;
if (a == 2){
   b++; }
else {
   c++;
}
cout<<a<<b<<c;  // 1,2,4
if (c<10){
   a++;
}
cout<<a<<b<<c;  // 2,2,4
```

Condicionais aninhadas

Como o bloco pode conter qualquer coisa, inclusive outros comandos condicionais, surge uma possível fonte de confusão: se houver 2 comandos IF e um único ELSE, a qual dos dois ele pertence ? Antes de pensar na resposta veja um exemplo

```
a=1;b=2;c=3;
if (a<3) {
   b++;
   if (c<2){
      a++;
   }
   else {
     a++;
   }
}
cout<<a<<b<<c;   // deu 2,3,3
}
```

Note que esta incerteza é resolvida pela presença das chaves. Daí a sua importância mesmo quando elas não forem obrigatórias: quando o bloco tem 1 só comando elas são opcionais. (Mas não deveriam ser...).

5.4 Erros

Os erros em programação podem ser de diversos tipos, a saber:

de compilação Erros de síntaxe que o compilador detecta facilmente. Impedem a compilação de ser bem sucedida e em geral são facilmente corrigidos, já que descritos pelo compilador. Por exemplo, terminar um comando sem o `;` ou não fechar todos os parênteses que abriu, ou escrever `coout`, ou esquecer de definir a função `main()` ou assim por diante.

de execução Mais graves que o anterior, até porque em geral mais difíceis de descobrir/consertar. São conhecidos como erros semânticos. Em geral denotam um algoritmo errado ou mal implantado. Para este tipo de erro o computador e o compilador lavam as mãos.

de ambiente Algum mau funcionamento do ambiente: Windows, C++, o próprio hardware. São extraordinariamente raros, só estão aqui para lembrar que eles existem, mas não devem ser considerados como candidatos na depuração de programas.

5.5 Exercícios

Faça um teste de mesa (chinês) nos exemplos a seguir e descubra qual valor vai ser impresso:

Exercício 1

```
int main() {
   int A,B,C,D,F,G;
   A=2,B=9,C=2,D=4,F=4,G=3;
   if(G>=5){
      A=B-C;
      G=F+A;
      A=F*D;
   }
   cout<<B+F+G-C;
}
```

Exercício 2

```
int main() {
   int A,B,C,D,F,G;
   A=4,B=9,C=6,D=4,F=2,G=3;
   if(C<=2){
      G=D*F;
      F=C*F;
   }
   else{
      C=C+A;
   }
   cout<<G+A+F-B;
}
```

Exercício 3

```
int main() {
   int A,B,C,D,F,G;
   A=6,B=2,C=6,D=6,F=6,G=2;
   if((D>4)&&(A!=8)){
      A=F+C;
      F=D+G;
   }
   else{
      C=B*B;
   }
   cout<<D+A+F-C;
}
```

Exercício 4

```
int main() {
   int A,B,C,D,F,G;
   A=4,B=5,C=8,D=8,F=4,G=8;
   if((C==9)||(A>=6)){
```

```
      B=B+F;
   }
   else{
      B=G-G;
      C=A*G;
   }
   cout<<A+F+D-B;
}
```

Exercício 5

```
int main() {
   int A,B,C,D,F,G;
   A=9,B=4,C=8,D=8,F=9,G=4;
   if((D<=7)&&(G>5)){
      D=B+C;
      if(A>=2){
         G=D*G;
      }
      else{
         G=C-A;
      }
   }
   else{
      A=A+D;
      D=G+B;
   }
   cout<<A+D+B+G;
}
```

Exercício 6

```
int main() {
   int A,B,C,D,F,G;
   A=4,B=4,C=8,D=7,F=2,G=4;
   if((F==5)||(C<9)){
      D=D*C;
      if(D!=6){
         B=B-F;
      }
      else{
         C=F+G;
      }
   }
   else{
      if(B==7){
         D=F*A;
      }
      B=A+B;
   }
   cout<<A+F+G-B;
}
```

Exercício 7

```
int main() {
   int A,B,C,D,F,G;
   A=4,B=9,C=5,D=6,F=7,G=4;
   if((F>4)||(B==5)){
      A=F*B;
      if(C>3){
         C=F+D;
      }
      else{
         if((C==5)&&(C<9)){
            C=F-C;
         }
      }
   }
   else{
      if(A>5){
         C=C*A;
      }
      D=D+B;
   }
   cout<<F+C+B-G;
}
```

Exercício 8

```
int main() {
   int A,B,C,D,F,G;
   A=6,B=6,C=9,D=7,F=4,G=9;
   if((!(G>3))&&(!(B!=6))){
      G=D*G;
      if(F<4){
         D=F+C;
      }
      else{
         if((D>2)||(!(C>=3))){
            F=D-F;
         }
      }
   }
   else{
      if(A>=9){
         B=A*F;
      }
      D=D+D;
   }
   cout<<B+G+A-D;
}
```

Exercício 9

```
int main() {
   int A,B,C,D,F,G;
   A=3,B=7,C=7,D=2,F=5,G=5;
   if((F!=4)||(!(G!=8))){
      G=F*C;
      if(G!=9){
         B=D*F;
      }
      else{
         F=G-F;
      }
   }
   else{
      B=D*B;
      F=C+F;
   }
   cout<<G+B+C-F;
}
```

Exercício 10

```
int main() {
   int A,B,C,D,F,G;
   A=6,B=6,C=2,D=9,F=8,G=5;
   if((!(B>=3))||(B!=7)){
      G=B-G;
      if(F!=6){
         C=G+F;
      }
      else{
         A=B+D;
      }
   }
   else{
      if(F>4){
         B=A+C;
      }
      F=A-C;
   }
   cout<<G+A+B-C;
}
```

Exercício 11

```
int main() {
   int A,B,C,D,F,G;
   A=8,B=5,C=2,D=7,F=7,G=2;
   if((D>=9)||(D!=4)){
      F=B+D;
      if(F>=7){
         G=B*C;
      }
      else{
         if((G!=6)||(D<=2)){
            F=F-A;
         }
      }
   }
   else{
      if(G<=2){
         G=B+G;
      }
      C=D-F;
   }
   cout<<C+D+A-F;
}
```

Exercício 12

```
int main() {
   int A,B,C,D,F,G;
   A=2,B=7,C=8,D=9,F=6,G=2;
   if((C!=7)||(B>9)){
      A=D+B;
      if(C<=3){
         C=G+D;
      }
      else{
         if((C>=7)||(G<3)){
            F=G*C;
         }
      }
   }
   else{
      if(D!=7){
         G=D+C;
      }
      C=C+A;
   }
   cout<<B+G+F+A;
}
```

Respostas

```
ex. 1=14
ex. 2=0
ex. 3=20
ex. 4=16
ex. 5=33
ex. 6=8
ex. 7=25
ex. 8=7
ex. 9=47
ex. 10=4
ex. 11=5
ex. 12=41
```

5.6 Mais exercícios, agora compostos

Exercício 1

```
int main() {
   int A,B,C,D,F,G;
   A=7,B=4,C=7,D=2,F=7,G=4;
   if(G>8){
      A=G+A;
      if((C!=8)||(G!=6)){
         D=F*D;
      }
      A=C-C;
      }
   else{
      F=D+A;
      if((C!=6)||(C>6)){
         F=D+A;
      }
      else{
         if(C<=6){
            C=D+D;
         }
         else{
            A=F+G;
         }
         G=B-F;
      }
      B=G*B;
      }
   C=B*G;
   cout<<G+F+A-C;
}
```

Exercício 2

```
int main() {
   int A,B,C,D,F,G;
   A=4,B=8,C=7,D=3,F=3,G=8;
   if(A>=7){
      if((G==5)&&(G==4)){
         B=A-D;
      }
      if(D>=9){
         F=F+G;
      }
      F=F*F;
   }
   else{
      B=F+G;
      if((C>6)||(!(B<2))){
         F=B+A;
         if(F<5){
            B=F*F;
         }
      }
      else{
         if((D>7)&&(F==6)){
            C=F*D;
         }
         else{
            F=G+C;
         }
         C=B+B;
      }
      F=D+G;
   }
   cout<<G+B+C-A;
}
```

Exercício 3

```
int main() {
   int A,B,C,D,F,G;
   A=9,B=6,C=2,D=7,F=3,G=2;
   if(D==7){
      F=D-D;
   }
   if((C<7)&&(B!=6)){
      if(D>3){
         C=F*C;
      }
   }
   A=D*F;
   if((D<5)||(C>8)){
      D=G*D;
      if(B>=2){
         B=F*B;
      }
      else{
         A=D*A;
      }
   }
   else{
      if((!(B==5))||(D<6)){
         F=A-F;
      }
      else{
         F=G-B;
      }
      B=A-D;
   }
   B=F*B;
   cout<<F+C+A+B;
}
```

Exercício 4

```
int main() {
   int A,B,C,D,F,G;
   A=6,B=4,C=5,D=7,F=9,G=6;
   if(D>=2){
      C=A*F;
   }
   if((D<=7)||(!(C==6))){
      if(F<2){
         B=G*A;
      }
      else{
         if((A>6)&&(F>=4)){
            D=D+F;
         }
         else{
            if(C!=2){
```

```
                B=B-F;
             }
          }
          A=A*B;
       }
       G=D-F;
    }
    D=D+D;
    if((A>5)&&(A!=3)){
       D=A*B;
       if(F<7){
          G=G-C;
       }
       else{
          D=A*D;
       }
    }
    else{
       if((A<3)&&(A!=5)){
          C=G+B;
       }
       else{
          G=G+C;
       }
       A=D-F;
    }
    D=G+D;
    cout<<A+D+B-C;
}
```

Exercício 5

```
int main() {
   int A,B,C,D,F,G;
   A=5,B=3,C=6,D=8,F=8,G=2;
   if(D!=9){
      if((G<4)||(F==7)){
         if(D>8){
            if((G>=7)&&(!(D>3))){
               F=F+C;
            }
            else{
               A=D-C;
            }
            F=F-B;
         }
         else{
            C=A*B;
         }
         A=F-F;
      }
      else{
         A=G-B;
      }
      F=A-F;
   }
   else{
      if(B==3){
         if((C<=7)||(D<=7)){
            if(A>4){
               F=B*G;
            }
            else{
               F=F+A;
            }
         }
         else{
            A=C*D;
         }
      G=G+B;
      }
   }
   C=F*G;
   cout<<C+F+B-G;
}
```

Exercício 6

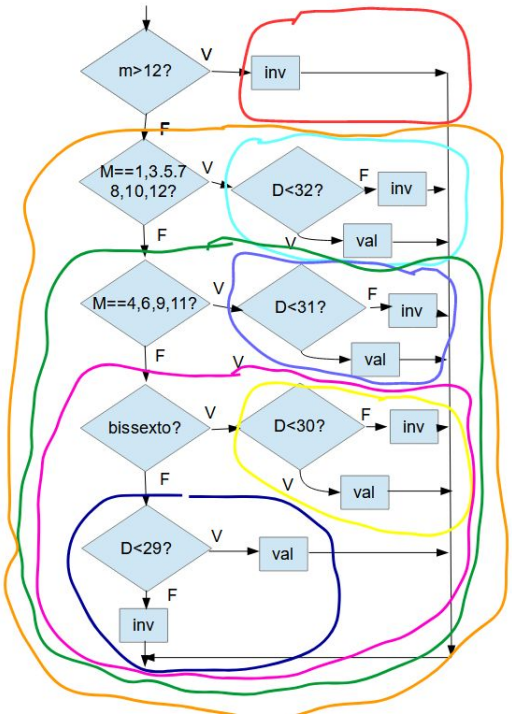
```
int main() {
   int A,B,C,D,F,G;
   A=5,B=4,C=9,D=6,F=6,G=3;
   if(B>=5){
      C=B+G;
      if((D>6)||(G==4)){
         B=B-B;
      }
      F=G-C;
   }
   else{
      A=C*A;
      if((A<9)||(A<5)){
         A=C*A;
      }
      else{
         if(D<4){
            B=B+F;
         }
         else{
            G=F*D;
         }
         B=D-A;
      }
      C=D-G;
   }
   D=A*G;
   cout<<B+C+D+F;
}
```

Respostas

```
ex. 1=-44
ex. 2= 22
ex. 3=  2
ex. 4= 19
ex. 5=-23
ex. 6=1557
```

5.7 um bom exemplo: data boa

Seja um programa C++ para ler uma data (dia, mês e ano) e decidir se ela é boa ou não. Exemplos de datas boas: 01/01/01, 25/08/62, 31/12/22, ... Exemplos de datas não boas: 01/50/22, 30/02/45, 32/07/70, ... Este exemplo é adequado para introduzir o conceito de IFs encadeados (um dentro do outro) e ele ressalta de maneira vigorosa a importância das chaves.




```
#include<iostream>
using namespace std;
int main() {
  int d,m,a,b,val;
  val=0;
  cout<<"Informe dia, mes e ano"<<endl;
  cin>>d>>m>>a;
  if ((a%4)==0) {
      b=1;}
   else {
      b=0;
   }
   if (m>12) {val=0;}
   else {
    if ((m==1)||(m==3)||(m==5)||(m==7)||(m==8)||(m==10)||(m==12)) {
        if (d<32){val=1;}
        else {val=0;}
    }
    else {
        if ((m==4)||(m==6)||(m==9)||(m==11)) {
            if (d<31){val=1;}
            else {val=0;}
            }
            else {
                if (b==1) {
                    if (d<30) {val=1;}
                    else {val=0;}
                }
                else {
                    if (d<29) {val=1;}
                    else {val=0;}
                }
            }
        }
    }
    if (val==1){ cout<<"valido"<<endl;}
    else {cout<<"invalido"<<endl;}
}
```

5.8 Mais exercícios

Tirados do planejamento on-line do curso, devem ser produzidos pelo aluno. A palavra entre parênteses é o nome do programa.

5.8.1 (achamaior)

Escreva um programa C++ que leia 3 números e imprima o maior deles. Lembrar que os 3 ou 2 deles podem ser iguais.

5.8.2 (somamaior)

... Que leia 3 números e imprima a soma dos dois maiores. Lembrar que os 3 ou 2 deles podem ser iguais.

5.8.3 (quadrante)

... que leia as coordenadas de um ponto (x, y) reais, e imprima a localização do ponto podendo ser: I, II, III ou IV quadrante, eixo X, eixo Y ou origem.

5.8.4 (triang)

... leia a,b,c,(reais) e verifique se eles podem formar um triângulo, Se sim, imprimir o perímetro e o tipo de triângulo que pode ser equilátero, isósceles ou escaleno.

5.8.5 (siseqlin)

... leia a,b,c,d,e,f e ache x e y com os seguintes valores $x = \frac{c.e-b.f}{.c-b.d}$ e $y = \frac{a.f-c.d}{a.e-b.d}$ se for possível. x e y devem ser impressos.

5.8.6 (notas)

... leia o código de um aluno e suas 3 notas. A maior nota tem peso 4 e as outras duas têm peso 3. Se a média for maior ou igual a 5,0 deve imprimir APROVADO e caso contrário imprimir REPROVADO. Imprima o código, as 3 notas, a média e a mensagem.

5.8.7 (idade)

... leia dia, mes, ano e determine a idade em 07/maio/2022.

5.9 (USP: triangular)

Dizemos que um número natural é triangular se ele é produto de 3 números naturais consecutivos. Exemplo, 120 é triangular pois $4 \times 5 \times 6 = 120$. Escreva um programa C++ que receba um número e imprima "TRIANGULAR"se ele o for.
Da teoria: $N = (a-1) \times a \times (a+1)$ e daqui $N = a^3 - a$. Daqui, se extrairmos a raiz cúbica de N truncando a resposta deve-se obter um valor possível para $a-1$. Agora basta multiplicar este valor pelo seu sucessor e pelo seu sucessor do sucessor e comparar a resposta com N. Se for igual, será triangular.

```
int main() {
  float N;
  int a;
  cin>>N;
  a = pow(N,0.33333333);
  if ((a*(a+1)*(a+2))==N) {cout<<"TRIANGULAR";}
}
```

5.10 Aprendendo a isolar condicionais

Uma habilidade importante (sobretudo ao olhar código feito por outros, ou por você mesmo em outros tempos) é isolar cada condicional com sua correspondente alternativa negada (o else), que lembrando é opcional. Para deixar o problema mais realista, eliminou-se completamente a indentação. Vão aí alguns exercícios:
Considere:

1. Todas as variáveis são inteiras.
2. O comando de atribuição precisa ser corrigido. Ele está como está para economizar espaço.
3. Ao final dos comandos é necessário incluir um ;
4. As condições precisam ser colocadas entre parênteses.

1. Calcule o valor correto que ser impresso pelo trecho a seguir

```
int main() {
A,B,C=2,8,7
if ((A!=3)&&(!(B!=7)))&&(!(C==7)){
if B == 9 {
if C < 8 {
if C < 7 {
if B < 5 {
A = A * 4
}
}else{
A = A + (B + 2)
}
B = C * 2
}else{
B = B + (A + 2)
}
A = B + 4
}else{
C = C + (B + 5)
}
```

```
}else{
C = C * 2
}
cout<< (A+B+C)
}
```

2. Calcule o valor correto que ser impresso pelo trecho a seguir

```
int main() {
A,B,C=2,8,7
if ((A!=3)&&(!(B!=7)))&&(!(C==7)){
if B == 9 {
if C < 8 {
if C < 7 {
if B < 5 {
A = A * 4
}
A = B + 4
}
}else{
A = A + (B + 2)
}
}else{
B = C * 2
B = B + (A + 2)
}
}else{
C = C + (B + 5)
}
C = C * 2
cout<< (A+B+C)
}
```

3. Calcule o valor correto que ser impresso pelo trecho a seguir

```
int main() {
A,B,C=2,9,8
if ((A==6)&&(!(B<=2)))&&(C>2){
if B < 9 {
if C >= 7 {
if C != 6 {
if B > 5 {
A = C * 2
}
B = B * 3
}else{
B = B + (A + 2)
}
}else{
C = B + 4
C = C + (A * 4)
}
}
B = C + 5
}else{
C = C + (A * 5)
}
cout<< (A+B+C)
}
```

4. Calcule o valor correto que ser impresso pelo trecho a seguir

```
int main() {
```

```
A,B,C=2,9,8
if ((A==6)&&(!(B<=2)))&&(C>2){
if B < 9 {
if C >= 7 {
if C != 6 {
if B > 5 {
A = C * 2
}else{
B = B + (A + 2)
}
B = C + 5
}
}
}else{
C = C + (A * 4)
B = B * 3
}
C = B + 4
}else{
C = C + (A * 5)
}
cout<< (A+B+C)
}
```

5. Calcule o valor correto que ser impresso pelo trecho a seguir

```
int main() {
A,B,C=3,7,9
if (!(A!=7))||(B==3){
if B < 7 {
if C > 7 {
if C >= 8 {
if C == 5 {
A = A * 2
}else{
C = C + (A * 2)
}
C = A - 4
B = B + 5
}else{
C = A - 5
A = A + (B + 5)
}
}else{
A = A + (B + 2)
}
}
}
cout<< (A+B+C)
}
```

6. Calcule o valor correto que ser impresso pelo trecho a seguir

```
int main() {
A,B,C=3,7,9
if (!(A!=7))||(B==3){
if B < 7 {
if C > 7 {
if C >= 8 {
if C == 5 {
A = A * 2
}
}else{
```

```
A = A + (B + 2)
C = C + (A * 2)
}
}
}else{
C = A - 4
A = A + (B + 5)
C = A - 5
}
B = B + 5
}
cout<< (A+B+C)
}
```

7. Calcule o valor correto que ser impresso pelo trecho a seguir

```
int main() {
A,B,C=3,8,4
if (!(A!=7))&&(!(B<4)){
if B <= 8 {
if C == 7 {
if A == 6 {
if C <= 6 {
B = B - 4
}else{
C = C + (B * 2)
}
}else{
B = B + (A - 2)
C = A + 2
C = C - 2
}
}else{
A = A + (B - 3)
}
}
}
B = C + 5
cout<< (A+B+C)
}
```

Respostas

```
ex. 1 = 24
ex. 2 = 50
ex. 3 = 29
ex. 4 = 29
ex. 5 = 19
ex. 6 = 19
ex. 7 = 16
```

Capítulo 6

Repetição

6.1 desvio

É o nosso próximo assunto.

Até agora, em nossos exemplos, os comandos começam a ser executados logo após a definição de `main()` e seguem sequencialmente até o final do programa, comando após comando. Esta ordem é a esperada, tanto é que nos exemplos acima ela foi seguida normalmente sem ter sido explicitada.

Entretanto, este jeito de funcionar não tem muito futuro. Imagine um programa de computador para calcular o salário de 1000 empregados. Não tem o menor sentido programar a mesma coisa 1000 vezes em sequência. Além do que agindo assim, perder-se-ia uma grande vantagem do computador que é garantir que todos tenham o mesmo tratamento. Neste caso, programa-se o cálculo para 1 funcionário e depois garante-se que os 1000 passem pelo mesmo processamento. Para fazer isto em algum ponto do código deve-se escrever `va para x`, onde x é um rótulo aposto em algum dos comandos provavelmente no início do processamento. Em termos esquemáticos

O desvio sozinho não tem muita lógica ou muito uso. Tanto é que em linguagens mais modernas (p.ex Python) ele nem existe mais. Mesmo em C++, embora exista, provavelmente você não vai usá-lo nunca.

Há 50 anos este comando era usado extensivamente, mas diversos pesquisadores (os italianos Conrado, Jacopini e sobretudo o holandes Dijkstra) defenderam a tese de que usar desvios em programas era o caminho mais curto para produzir software confuso e sobretudo incorrigível, ou usando o jargão indebugável. Explico: Imagine um código contendo dezenas de desvios. O programa pára no meio por causa de um erro. Como saber como estava o fluxo e sobretudo " de onde estava vindo ?"

Dessa constatação começou a nascer a "programação estruturada". Nela, o desvio é abolido e em seu lugar entram as estruturas de repetição. Nestas, há um desvio, mas ele é bem comportado, podendo acessar apenas o bloco onde ele se encontra. Então, não há desvio entre blocos, que eram a maior causa dos problemas acima apontados.

6.2 Repetição

Como se viu, a programação estruturada troca o desvio pela repetição. Então vamos a ela: Existem 3 blocos básicos de repetição:

- repetição indeterminada com teste no início (em C++ while (condição){...})
- repetição determinada (em C++ for(inicio;teste;incremento){...})
- repetição indeterminada com teste ao final (em C++ do {...} while(condição))

Como nosso curso é de programação (e não de C++), vai-se deixar o terceiro formato de lado.

A repetição entra em cena sempre que um determinado trecho de programa precisa ser repetido várias vezes. Acompanhe o exemplo.

6.3 X é primo ?

Escreva um programa C++ que receba um inteiro positivo N e informe *primo* se ele for primo e *composto* se ele não for. Definição: um número inteiro positivo é primo se ele for maior que 1 e tiver apenas dois divisores: a unidade e o próprio N. Desta definição, percebe-se que há necessidade de contar a quantidade de divisores do número dado e depois comparar esta quantidade com 2.

```
int main(){
   int N;
   int divisores, x;
   divisores=0;
   x=1;
   cin>>N;
   while (x<=N){
      if ((N%x)==0){
          divisores++;
      }
      x++;
   }
   if (divisores>2) {
       cout<<"composto"<<endl;
       }
   else {
       cout<<"primo"<<endl;
   }
}
```

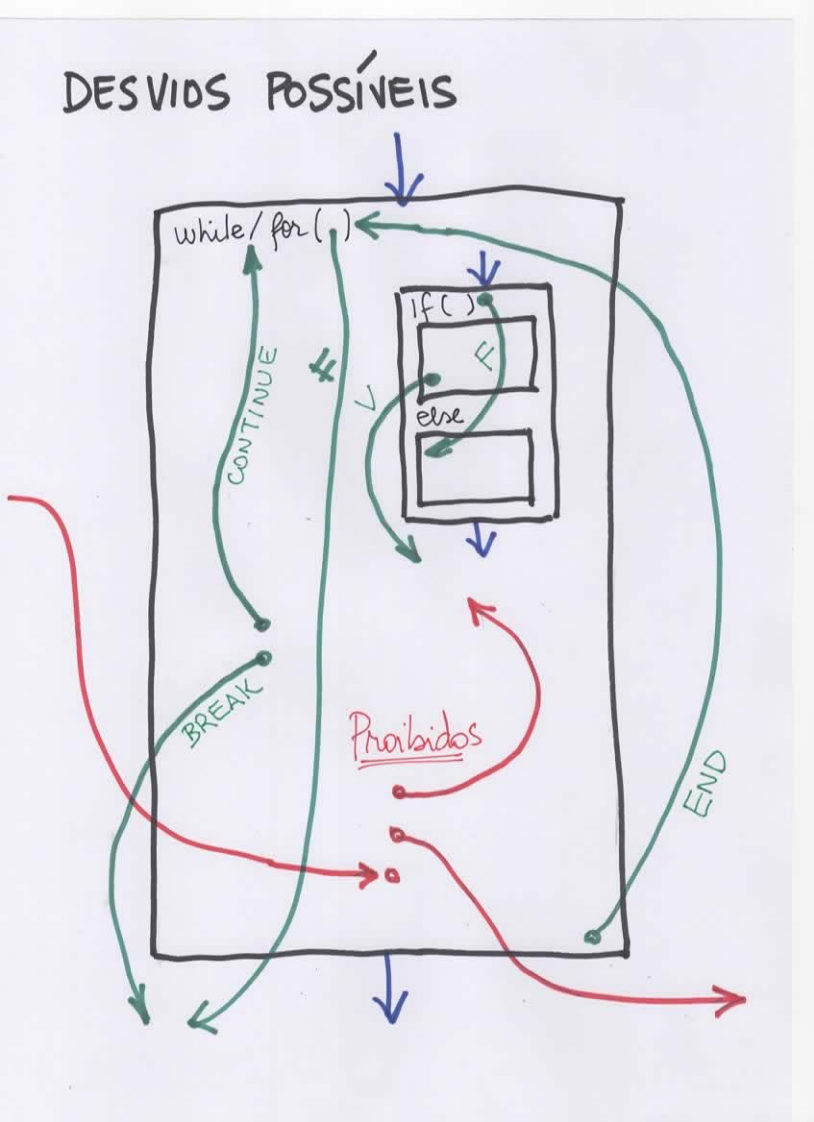
Inúmeras melhorias podem ser feitas sobre este código:

- deixar de fora o 1 e o N e comparar os divisores com > 0.
- Excluir os pares (fora o 2), e acelerar de 2 em 2.
- Ir só até $\sqrt{N}$, já que só é necessário achar 1 divisor (fora o 1 e o N).
- Ao achar um divisor, dá para cair fora.
- Dá para acelerar de 6 em 6, mas aí é preciso estudar um pouco os primos, já que todo primo tem a forma $P = 6k \pm 1$.

Este exemplo ilustra bem um ponto: a eficácia (achar a resposta certa) versus a eficiência (achar a resposta rápido). Na nossa vida profissional sempre iremos priorizar a eficácia. Só se houver importância é que daremos atenção à eficiência...

6.4 Em resumo

eis uma explicação gráfica (visual) do conceito de desvio



6.5 (Amplitude)

Escreva um programa C++ que leia n (indeterminado, condição de fim = 0) e imprima a amplitude dos valores fornecidos (amplitude = maior - menor)

```
int main() {
   int x,mai,men;
   mai=-999999;
   men=999999;
   cin>>x;
   while(x>0){
      if (x>mai) {mai=x;}
      if (x<men) {men=x;}
      cin>>x;
   }
   cout<<"Amplitude: "<<mai-men<<endl;
}
```

6.6 Treinamento em repetição

Exemplos

Seja o programa

```
int main() {
   int A,B,C,D;
   A=5,B=2,C=7;
   while(B<15){
      C=A+5;
      A=A+4;
      B=B+1;
   }
   cout<<C+A+B;
}
```

Cujo resultado é 130 e seja o seguinte código

```
int main() {
   int A,B,C,D;
   A=4,B=1,C=2;
   if(C<=12){
      while(B<=12){
         C=A+3;
         B=B+4;
      }
   }else{
      C=C-8;
      B=B+3;
   }
   cout<<C+A+B;
}
```

cujo resultado é 24. Agora é a sua vez:

Exercício 1

```
int main() {
   int A,B,C,D;
   A=6,B=6,C=7;
   while(C<=19){
      A=B+5;
      A=B+3;
      C=C+3;
   }
   cout<<A+B+C;
}
```

Exercício 2

```
int main() {
   int A,B,C,D;
   A=5,B=1,C=7;
   if(A<=19){
      while(B<19){
         A=A+3;
         B=B+4;
      }
   }
   else{
      C=A-7;
      B=B+4;
   }
   cout<<A+A+B;
}
```

Exercício 3

```
int main() {
   int A,B,C,D;
   A=5,B=5,C=4;
   if(B!=15){
      while(C<15){
         B=B-2;
         C=C+4;
      }
   }
   else{
      while(C<15){
         B=B+1;
         C=C+2;
      }
   }
   cout<<B+B+C;
}
```

Exercício 4

```
int main() {
   int A,B,C,D;
   A=3,B=7,C=4;
   D=6;
   D=D+4+7;
   while(D>=7){
      A=C-6;
      while(B<=14){
         C=A-2;
         B=B+3;
      }
      C=C+12;
      D=D-3;
   }
   cout<<C+A+B;
}
```

Exercício 5

```
int main() {
   int A,B,C,D;
   A=3,B=8,C=4;
   D=7;
   C=C+4+8;
   while(C>=7){
      A=B+5;
      if(A<8){
         while(D<16){
            B=B+8;
            D=D+3;
         }
         B=A-17;
      }
      else{
         A=A-8;
      }
      C=C-4;
   }
   cout<<D+D;
}
```

Exercício 6

```
int main() {
   int A,B,C,D;
   A=7,B=4,C=2;
   D=8;
   B=B+2+4;
   while(B>2){
      C=C+2;
      if(D>=5){
         while(A<18){
            A=A+6;
         }
         D=D-2;
      }
      else{
         C=D+15;
      }
      if(B<4){
         D=D+5;
      }
      else{
         D=C+9;
      }
      B=B-1;
   }
   cout<<D+C;
}
```

Exercício 7

```
int main() {
   int A,B,C,D;
   A=1,B=6,C=1;
   while(B<=16){
      C=A-5;
      C=C+8;
      B=B+1;
   }
   cout<<C+A+B;
}
```

Exercício 8

```
int main() {
   int A,B,C,D;
   A=7,B=3,C=7;
   if(C!=19){
      while(B<19){
         C=A-4;
         B=B+4;
      }
   }
   else{
      C=A+2;
      B=B+4;
   }
   cout<<C+A+B;
}
```

Exercício 9

```
int main() {
   int A,B,C,D;
   A=5,B=4,C=4;
   if(C!=19){
      while(A<19){
         C=B-6;
         A=A+3;
      }
   }
   else{
      while(A<17){
         B=C+4;
         A=A+1;
      }
   }
   cout<<1+C+B+A;
}
```

Exercício 10

```
int main() {
   int A,B,C,D;
   A=2,B=1,C=5;
   D=6;
   B=B+5+1;
   while(B>=1){
      A=B+2;
      if(C>10){
         while(D<16){
            C=C-2;
            D=D+3;
         }
         A=C+14;
      }
      else{
         A=C+5;
      }
      B=B-1;
   }
   cout<<B+C;
}
```

Respostas

```
ex. 1=37
ex. 2=61
ex. 3=14
ex. 4=86
ex. 5=14
ex. 6=46
ex. 7=22
ex. 8=29
ex. 9=23
ex.10= 5
```

6.7 for

O comando `for` é um ótimo aliado no processamento de laços com tamanho conhecido. Acompanhe o formato do comando

```
for (inicialização; saída; incremento){
   ...
}
```

A idéia é dividir o comando em 3 etapas:

- a inicialização que é feita ANTES de entrar no bloco.
- a saída que é uma condição que é testada (verificada) sempre no início do bloco. Se a condição é verdadeira, o bloco é executado. Se falta ocorre a saída do comando.
- O incremento é realizado ao final do bloco e logo após o incremento há um desvio ao início do comando for

Acompanhe um exemplo simples

```
for (i=0;i<4;i++){
   cout<<i;
}
```

O comando for acima vai imprimir os valores 0,1,2 e 3.

6.7.1 for e while

Certamente while é mais genérico que for (ou seja, tudo o que se faz com for pode ser feito com while, mas a recíproca nem sempre é verdadeira. A razão da existência do for é que ele reduz o tamanho do código. Grosso modo, pode orientar o for a quantidades fixas e o while para uma saída condicional. Mas, do exposto fica a regra: se você precisar ir para uma ilha deserta e só puder levar um repetidor, leve o while. Na definição do for, sabe-se existirem 3 blocos, a saber inicialização, saída e atualização. A novidade é que qualquer um dos blocos pode ser deixado vazio, e a compilação segue em frente. Também em relação ao while, pode-se comandar o bloco via uma tautologia, mas neste caso é preciso organizar uma saída de dentro do bloco via comando `break`. Acompanhe o exemplo

```
while (1==1){
   ...
   if (condição){
      break}
   ...
}
```

6.8 Condição composta e while

Suponha que você precisa ler um número e ele só será aceitável se estiver entre 10 e 50. Como ficaria o código ? A condição de correção é $a \geq 10$ e $a \leq 50$. Mas como eventualmente o número fornecido esteja errado, a sua captura deve estar dentro de um bloco de repetição, do qual só se sairá após o número estar correto. Então, perceba que no while, deve-se negar a condição de correção. É hora de usar o T. de Morgan, que diz $\overline{A \wedge B} = \overline{A} \vee \overline{B}$ e com isso o código fica:

```
a=0;
while ((a<10) || (a>50)){
    cout<<"Informe um número"<<endl;
    cin>>a;
}
```

6.9 (para comparar)

Seja a necessidade de descobrir a média entre 5 números, sendo implementada de 3 maneiras diferentes: Primeiro, usando 5 variáveis:

```
s=0;
cout<<"forneca 5 números"<<endl;
cin>>a>>b>>c>>d>>e;
s=(a+b+c+d+e)/5.0;
cout<<s;
```

Segundo, usando uma variável e o comando for

```
s=0;
cout<<"forneca 5 números"<<endl;
for (i=0;i<5;i++){
   cin>>a;
   s=s+a;
}
s=s/5.0;
cout<<s;
```

Terceiro, usando uma condição de saída (via while) o que permite adicionar qualquer quantidade de números

```
float s=0;
int qtd=0;
while (1==1){
   cout<<"Informe, (0 para terminar)"
   cin>>a;
   if (a==0){break;}
   s=s+a;
   qtd++;
}
s=s/qtd;
cout<<s;
```

6.10 (palito)

Imagine um video-game, no qual operador e programa disputam a vitória. É bem simples, a partir das regras abaixo

- O humano escolhe um número de palitos entre 20 e 30.
- O computador tira 1, 2 ou 3 palitos
- O humano faz o mesmo, alternadamente.
- Quem tirar o último palito, perde.

A questão é: como elaborar uma estratégia (quase) sempre vencedora ? A estratégia é jogar seu adversário com 1 palito, para que ele perca. Mas, para fazer isto precisa-se deixá-lo com 5. Neste ponto, se ele tirar 3, vc tira 1. Se ele tirar 2, você tira 2 e se ele tirar 1 você tira 3 e ele sempre acaba em 1. Então o novo objetivo é deixá-lo com 5. Mas para isso, você precisa deixá-lo com 9. E depois com 13, 17, 21, 25, 29. Se você conseguir encaixar seu adversário nesta sequência e jogar direitinho, vai ganhar o jogo. Se, ao contrário você cair nessa sequência e o seu adversário jogar certinho, você vai perder. Veja como isso foi implementado em C++.

```
int main() {
   int n,seq,j,tv,lixo;
   n=0;
   while ((n<20) || (n>30)) {
        cout<<"Com quanto comecamos ?"<<endl;
        cin>>n;
   }
   while(n>0){
      seq = n % 4;
      if (seq==1) {
          j=rand()%3+1;  // perdi, jogo 1 esperando o cara errar
      }
      if (seq==2) {
          j=1; // ganhei jogo 1 para deixar ele na sequencia}
      }
      if (seq==3) {
         j=2; // deixa ele na seq
         }
      if (seq==0) {
         j=3; // idem
         }
      cout<<"eu joguei "<<j<<endl;
      n=n-j;
      cout<<"existem "<<n<<" palitos \n"<<endl;
```

```
      if (n <= 0) {
         cout<<"Por incrivel que pareca, perdi"<<endl;
         cin>>lixo;
         break;
      }
      tv=0;
      while ((tv<1) || (tv>3) || ((n-tv)<0)) {
         cout<<"jogue"<<endl;
         cin>>tv;
      }
    n=n-tv;
    cout<<"existem "<<n<<" palitos "<<endl;
    if (n<=0) {
       cout<<"babaca, eu ganhei\n";
       cin>>lixo;
       break;
      }
   }
}
```

6.11 (qudradoecubo)

Escreva um programa C++ que imprima uma tabela de 1..20 com seus quadrados e cubos. Aproveite para olhar como se formatam

```
#include<iostream>
#include<cmath>
#include<iomanip>
using namespace std;
int main(){
  int i;
  for (i=1;i<=80;i++){
    cout<<setw(4);
    cout<<i;
    cout<<setw(7);
    cout<<pow(i,2);
    cout<<setw(10);
    cout<<pow(i,3);
    cout<<endl;
  }
}
```

6.12 (cpf)

Como se sabe, hoje em dia é muito importante confiar em códigos e identificadores. Imagine pagar um boleto de 20.000 reais e errar um numerinho daqueles. Se ninguém detectar o erro, lá se vão 20.000 reais. Para evitar este tipo de erro, existem diversas abordagens, e uma das mais simples (e mais usada) é a técnica dos dígitos verificadores. Trata-se de um (ou mais) dígito (s) que é calculado a partir do código original, e que é recalculado cada vez que o código completo é digitado. Se na hora do recálculo o resultado for diferente daquilo que foi digitado, é batata ! Tem erro na digitação.

Por exemplo, suponha que um certo código é 12345-6. Ou seja, o 6 é o resultado de uma certa função aplicado ao número 12345. Daí, na hora da digitação, alguém com a cabeça nas nuvens digita 12435-6. Note a inversão dos dígitos no número original. Antes de fazer qualquer coisa, o programa que recebeu este número aplica a função do D.V. ao código 12435 e certamente obterá um valor diferente de 6. Como o sujeito digitou 6, pode-se afirmar sem erro: cometeu-se um erro.

Os erros que podem ser cometidos são:

erro	código certo	código digitado errado
obliteração	123456	12456
repetição	123456	1223456
transposição	123456	123546
substituição	123456	128456
fraude	123456	987654

Um dos usos mais frequentes desta técnica está no cálculo do CPF que caminha para se tornar o número único de identificação em nosso país:

1. Divida os 9 digitos que compõe o código
2. multiplique cada dígito pela sequência 10,9,8,...,3,2
3. some tudo achando S
4. o primeiro dígito verificador é 11 - o resto da divisão de S por 11
5. se o resultado anterior for 10 ou 11 ele é trocado por 0
6. o D.V. recécm calculado é agregado ao código original e o ciclo recomeça com os pesos 11,10,9,...,3,2
7. obtem-se agora o segundo DV
8. o CPF completo tem os 9 dígitos originais, mais os 2 DVs calculados.

Veja-se a seguir 2 implementações: a L (longa) e a C (curta). Use a que lhe faça sentir melhor...

```
#include<iostream>
using namespace std;
void cpfl(){
  int d1,d2,d3,d4,d5,d6,d7,d8,d9,dv1,dv2;
  int s;
  cin>>d1>>d2>>d3>>d4>>d5>>d6>>d7>>d8>>d9;
  s=d9*2;
  s=s+d8*3;
  s=s+d7*4;
  s=s+d6*5;
  s=s+d5*6;
  s=s+d4*7;
  s=s+d3*8;
  s=s+d2*9;
  s=s+d1*10;
  s=s%11;
  dv1=11-s;
  if ((dv1==10) || (dv1==11)) {dv1=0;}
  cout<<"DV1="<<dv1<<endl;
  s=dv1*2;
  s=s+d9*3;
  s=s+d8*4;
  s=s+d7*5;
  s=s+d6*6;
  s=s+d5*7;
  s=s+d4*8;
  s=s+d3*9;
  s=s+d2*10;
  s=s+d1*11;
  s=s%11;
  dv2=11-s;
  if ((dv2==10) || (dv2==11)) {dv2=0;}
  cout<<"DV2="<<dv2<<endl;
}

void cpfc(){
    int ent,i,s1=0,s2=0;
    i=10;
    while (i>1){
        cin>>ent;
        s1=s1+(ent*i);
        s2=s2+(ent*(i+1));
        i--;
    }
    s1=11-(s1%11);
```

```
    if ((s1==10)||(s1==11)){s1=0;}
    s2=s2+(s1*2);
    s2=11-(s2%11);
    if ((s2==10)||(s2==11)){s2=0;}
    cout<<s1<<" "<<s2;
}
int main(){
  cpfc();
}
```

6.13 (USP: multiplos)

Dados n, i e j imprimir em ordem crescente os n primeiros naturais múltiplos de i, de j ou de ambos.

```
int main(){
   int i,j,n;
   int qtd=0,cand=0;
   cin>>n>>i>>j;
   while (qtd<n){
      if (((cand%i)==0) || ((cand%j)==0)){
           cout<<cand<<" ";
           qtd=qtd+1;
      }
      cand++;
   }
}
```

6.14 (USP: fatorial)

Dado um inteiro não negativo n determinar $n!$.

```
int main(){
  long int fa=1;
  int n;
  cin>>n;
  while (n>1){
    fa=fa*n;
    n--;
  }
  cout<<fa;
}
```

Embora não seja objetivo deste curso estudar **recursividade** ela é uma idéia poderosa e é uma bora hora para apresentar o conceito. Um programa é recursivo quando chama a sí mesmo. Veja o mesmo fatorial implementado rescursivamente

```
long long int fat(long long x){
  if (x<2){
     return 1;
  }
  else {
     return x*fat(x-1);
  }
}
int main(){
  long long n;
  cin>>n;
  cout<<fat(n);
}
```

Note que 3 coisas devem acontecer para uma função ser recursiva

- A função chamar a si própria
- Haver um caso básico, no qual não há recursão
- A cada chamada haver uma aproximação em direção ao caso básico

A humanidade dos programadores parece se dividir em 2 tribos irreconciliáveis: os amantes e os odiantes da recursividade. Mas, se você for programar, por exemplo, em LISP, daí não tem jeito: a estrutura de repetição praticamente única dessa linguagem é a recursividade.

6.15 (USP: mdc-euclides)

Dados dois números positivos, determinar o máximo divisor comum entre eles usando o algoritmo de Euclides, que é

```
int mdc(int a, int b){
   if (b==0){return a;}
   return mdc(b,(a%b));
}
int main(){
   int x,y;
   cin>>x>>y;
   cout<<mdc(x,y);
}
```

6.16 (USP: congruente)

Diz-se que um número i é congruente a j (módulo m) se e somente se $i\%m = j\%m$, por exemplo 35 é congruente a 39 módulo 4 pois $35\%4 = 39\%4 = 3$. Dados n, j e m imprimir os n primeiros naturais congruentes a j módulo m.

```
int main(){
   int n,j,m;
   cin>>n>>j>>m;
   int i=1,cuca;
   cuca=j%m;
   while (n>0){
      if ((i%m)==cuca){
        cout<<i<<" ";
        n--;
      }
      i++;
    }
}
```

6.17 (USP: ladostriangulo)

Dados 3 inteiros positivos, verificar se eles formam os lados de um triângulo retângulo.

```
int main(){
  int a,b,c;
  cin>>a>>b>>c;
  if (((a*a)+(b*b)==(c*c)) || ((a*a)+(c*c)==(b*b)) ||
    ((b*b)+(c*c)==(a*a))) {cout<<"SIM";}
}
```

6.18 (USP: dezenaseraiz)

(FIS 88) QUalquer número natural de 4 dígitos pode ser dividido em duas dezenas formadas pelos seus primeiros dois dígitos e pelos últimos dois dígitos. Por exemplo

- 1297: 12 e 97
- 5314: 53 e 14

Escreva um programa C++ que imprima todos os milhares (4 algarismos) cuja raiz quadrada seja a soma das dezenas formadas acima. Por exemplo, 9801 tem como raiz 99 que é a soma de 98 com 01. Logo 9801 é um dos números a serem impressos.

```
int main() {
   int ini=1000;
   int p1,p2;
   while (ini<10000){
      p1=ini/100;
      p2=ini%100;
      if(sqrt(ini)==p1+p2){
          cout<<ini<<endl;
      }
      ini++;
   }    // sao impressos: 2025, 3025 e 9801
}
```

6.19 (USP: segmento iguais)

Dados n e uma sequência de n números inteiros, determinar quantos segmentos de números iguais consecutivos compõe essa sequência. Por exemplo para $n = 10$ e a sequência 5,2,2,3,4,4,4,4,1,1 é formada por 5 segmentos de números iguais.

```
int main(){
   int n,i,qtd;
   int v[100];
   cin>>n;
   for (i=0;i<n;i++){
      cin>>v[i];
   }
   i=1;
   qtd=1;
   while(i<n){
      if (v[i]!=v[i-1]) {qtd++;}
      i++;
   }
   cout<<qtd;
}
```

Desafio: programe isso sem usar vetores...

6.20 (USP: segmentos crescentes)

Dados n e uma sequência de n números inteiros, determinar o comprimento do segmento crescente de comprimento máximo. Por exemplo para $n = 9$ e a sequência 5,10,3,2,4,7,9,8,5 o comprimento do maior segmento crescente (2,4,7,9) é 4. Na sequência 10,8,7,5,2 o comprimento do maior segmento crescente é 1.

```
int main(){
   int i,n,mai,tam;
   int v[100];
   cin>>n;
   for(i=0;i<n;i++){
      cin>>v[i];
   }
   mai=-99;
   i=1;
   tam=1;
   while (i<n){
     cout<<i<<" "<<tam<<" "<<mai<<" "<<v[i]<<endl;
     if (v[i]>=v[i-1]){
         tam++;
     }
     else {if (tam>mai){
              mai=tam;}
              tam=1;
          }
     i++;
     }
```

```
   if (mai==-99){cout<<tam;}
   else {cout<<mai;}
}
```

6.21 (USP: número palíndrome)

Diz-se que um número natural n com pelo menos 2 dígitos é plaíndromo se o primeiro algarismo é igual ao último, o segundo é igual ao penúltimo e assim por diante. Então, 567765 e 32423 são palíndromes e 567675 não é. Dado um inteiro $n > 9$ verificar se ele é palíndrome.

```
int main(){
  int n,a1,a2,enaoe,j;
  cin>>n;
  enaoe=1;
  j=floor(log10(n));
  while (n>0){
    a1=n%10;
    a2=n/pow(10,j);
    if (a1!=a2){enaoe=0;}
    n=n-a2*pow(10,j);
    n=n/10;
    j=j-2;
  }
  cout<<enaoe;
}
```

6.22 (Ordem crescente)

Escreva um programa C++ que leia uma série indeterminada de números (condição de fim=0) e imprima a mensagem "em ordem crescente"se eles estiverem. Uma série está em ordem crescente se $s_i \leq s_j \forall i < j$.

```
int main() {
   int x,esta,ant;
   ant=-99999;
   esta=1;
   cin>>x;
   while(x>0){
      if (x<ant) {esta=0;}
      ant=x;
      cin>>x;
   }
   if (esta==1){cout<<"em ordem crescente";}
}
```

6.23 (PA e PG)

Escreva um programa C++ que leia a_1, r, n e imprima a PA e a PG associados:

```
int main(){
  long int a0,a1,r,n,x;
  cin>>a0>>r>>n;
  x=1;
  a1=a0;
  while(x<=n){
    cout<<a1<<" ";
    a1=a1+r;
    x++;
  }
  cout<<endl;
  x=1;
  a1=a0;
  while(x<=n){
```

```
    cout<<a1<<" ";
    a1=a1*r;
    x++;
  }
}
```

6.24 (Fibonacci)

Dada a sequência de Fibonacci 1,2,3,5,8,13,21,34,55,... e considerando os termos menores que 4.000.000 ache a soma dos termos ímpares (este é o problema Euler=2)

```
int main() {
   long long int n1,n2,n3,soma;
   n1=1;
   n2=1;
   soma=n1+n2;
   n3=0;
   while (n3<4000000000) {
      n3=n1+n2;
      if (n3%2==1){
         soma=soma+n3;
      }
      n1=n2;
      n2=n3;
   }
   cout<<"a soma e "<<soma<<endl;
}
```

6.25 (Primos relativos)

Dados m e n inteiros e positivos, eles são relativamente primos se não tiverem fator comum (por exemplo, 14 e 25 são relativamente primos). Escreva um programa C++ que leia m e n e imprima "RELATIVAMENTE PRIMOS"se eles o forem.

```
int main() {
   char R='N';
   int d=2;
   int m,n;
   cin>>m>>n;
   while (d<=m){
      if ((m%d==0) && (n%d==0)){
         R='S';
      }
      d++;
   }
   if (R=='N'){
      cout<<m<<" e "<<n<<" sao relativamente primos"<<endl;
   }
   else {
    cout<<"nao sao"<<endl;
   }
}
```

6.26 (Média dos pares)

Escreva um programa C++ que leia uma série indeterminada de inteiros positivos (condição de fim=0) e imprima a média dos números lidos que foram pares.

```
... para você fazer ...
```

6.27 (Palíndromo)

Escreva um programa C++ que leia uma frase e informe se ela é (1) ou não é (0) um palíndromo. Exemplos de palíndromos

```
socorram me subi no onibus em marrocos; a droga da gorda; roma me tem amor
oto come mocoto; sairam o tio e oito marias; a diva da vida; luz azul; ato idiota
me ve se a panela da moca e de aco madalena paes e vem; o treco certo
a base do teto desaba; atila toledo mata modelo talita; anotaram a data da maratona
o romano acata amores a damas amadas e roma ataca o namoro; sa da tapas e sapatadas
seco de raiva coloco no colo caviar e doces; assim a aia ia a missa; o cid e medico
erro comum ocorre; a cera causa sua careca; morram apos a sopa marrom
solo di sol a los idolos (espanhol); esope reste ici et se repose (francês)
Eh ça va la vache (idem)
```

```
#include<iostream>
#include<string>
using namespace std;
int main(){
   string frase;
   int pali,ini,fin;
// cin>>frase; pega só até o primeiro branco
   getline(cin, frase);
   pali=1;
   fin=frase.length()-1;
   ini=0;
   while(fin>=ini){
      while(frase[ini]==' '){ini++;}
      while(frase[fin]==' '){fin--;}
      if (frase[ini]!=frase[fin]){pali=0;}
      fin--;
      ini++;
   }
   cout<<pali;
}
```

Observações:

- Aparece aqui o tipo string (linha de caracteres)
- Se você fizer `cin>>frase` a mensagem termina no primeiro branco.
- Para pegar toda a linha deve-se fazer `getline(cin, frase)`
- Para conhecer o tamanho da frase o comando é `frase.length()` Como a numeração começa em zero precisa tirar uma unidade.
- Note que não se usam acentos, cedilhas, hífens etc
- Aqueles dois whiles para tirar brancos poderiam ser ifs. Entretanto, se o usuário digitar 2 brancos separando palavras, o while funciona e o if não.
- Note que quando o comando é unico compensa (acho eu) escrevê-lo na mesma linha da condição.
- Mas, mesmo assim não se deixa de fora as chaves. Desnecessárias, mas MUITO importantes.

6.28 Soma 6 números

Escreva um programa que leia 6 números e depois mostre a soma deles

```
int main(){
  int x,ct,soma;
  ct=0;
  soma=0;
  while (ct<6){
    cin>>x;
    soma=soma+x;
```

```
    ct=ct+1;
  }
cout<<"a soma e "<<soma<<endl;
}
```

6.29 Soma entre N_1 e N_2

... que leia dois números N_1 e N_2 e some os inteiros existentes entre N_1 e N_2 imprimindo o resultado ao final

```
int main(){
   int n1,n2,soma;
   soma=0;
   cin>>n1>>n2;
   while (n1<=n2){
    soma=soma+n1;
    n1=n1+1;
   }
   cout<<soma;
}
```

6.30 Soma entre N1 e N2 divisíveis por 7

... somar os inteiros entre N_1 e N_2 que forem divisíveis por 7 e imprimir a soma ao final

```
int main(){
   int n1,n2,soma;
   soma=0;
   cin>>n1>>n2;
   while (n1<=n2){
        if ((n1%7)==0){
            soma=soma+n1;
        }
    n1=n1+1;
   }
   cout<<soma;
}
```

Neste exercício deve-se notar o reaproveitamento de código, já que se usou o programa anterior, alterado.

6.31 Outros exercícios

1. Idem anterior, mas mostrando também quantas parcelas foram somadas
2. ... leia 6 números e imprima o valor da sua multiplicação. (Nota: elemento neutro na multiplicação é 1)
3. ... leia N e imprima a soma $\sqrt{1}+\sqrt{2}+...+\sqrt{n}$
4. ... leia N e imprima a taboada de N ($n=5 \rightarrow 5\times 1=5,\ 5\times 2=10,...5\times 10=50$)
5. ... leia 10 pares e imrpima a média de cada par
6. ... leia 10 pares e imprima a soma das médias de cada par
7. ... leia a_1, considere $r=3$ e imprima a soma das primeiras 20 parcelas da PA (progressão aritmética
8. ... leia a e b inteiros e calcule b^a. Note que são válidos $a=0$, $a=1$ e também $a<0$.

6.32 Controle de parada baseado na entrada

Até agora, escrevemos programas onde a quantidade de entradas é determinada previamente. Nem sempre é o caso, já que o fim pode ser dado por uma condição lógica. Exemplo: Escreva um programa que leia um número indeterminado de inteiros e mostre a sua soma. A condição de fim é a leitura do número 0. Note que este número é conhecido como SENTINELA. Ele é necessário já que não se sabe quantos números entrarão.

```
int main(){
   int x,soma;
   soma=0;
   cin>>x;
   while (x!=0){
      soma=soma+x;
      cin>>x;
   }
   cout<<soma;
}
```

6.33 Ler série de pares

Mais um exemplo: Escreva um programa que leia uma série de pares e para cada par mostre a sua média. A série termina quando for lido o par 0,0. Antes de programar este exemplo, vale uma passada no Teorema de Morgan que diz:

$$\overline{a \vee b} = \overline{a} \wedge \overline{b} \quad e \quad \overline{a \wedge b} = \overline{a} \vee \overline{b}$$

Traduzindo este teorema em palavras: a negação de uma condição composta com E é igual às negações compostas com OU e vice versa. Um exemplo prático real: No final do semestre você estará aprovado se tiver média ≥ 7 E frequência > 75. Negando tudo, você estará reprovado se tiver média < 7 OU frequência ≤ 75. Voltando ao nosso exemplo, se a condição de parada é $a, b = (0, 0)$ a continuação (negação de parada) é $a \neq 0$ ou $b \neq 0$, então

```
int main(){
   int a,b;
   float media;
   cin>>a>>b;
   while ((a!=0)||(b!=0)){
      cout<<(a+b)/2.0;
      cin>>a>>b;
   }
}
```

6.34 Repetições aninhadas

Escreva um programa que leia N_1, N_2, ..., 0 e para cada valor lido, mostrar uma linha com n asteriscos. Este é o caso de duas repetições, uma dentro da outra. Acompanhe

```
int main() {
  int n,k;
  cin>>n;
  while (n!=0){
    k=0;
    while(k<n){
        cout<<"*";
        k=k+1;
    }
    cout<<endl;
    cin>>n;
  }
}
```

6.35 Taboada

Escreva um programa que mostre a taboada dos inteiros no intervalo $[a, b]$ (a e b lidos e garantidamente $a \leq b$).

```
int main() {
   int a,b,k;
   cin>>a>>b;
   while (a<=b){
      k=1;
      while(k<=10){
        cout<<a<<" x "<<k<<" = "<<a*k<<endl;
```

```
            k=k+1;
        }
        a=a+1;
    }
}
```

6.36 Séries e somatórios

Uma série é um conjunto de números que seguem uma lei de formação, na qual o valor individual é uma função da posição ocupada, ou seja, pode-se obter o valor do termo como uma função do índice i da posição ocupada. Exemplo:
X= $1/2^0$, $1/2^1$, $1/2^2$, ... $1/2^n$ Daqui, pode-se escrever que $X_i = \frac{1}{2^i}$ e a sua soma é

$$S = \sum_{i=0}^{n} \frac{1}{2^i}$$

O programa que lê n e soma os $X_i = \frac{1}{2^i}$ com i variando de 0 a n é:

```
int main() {
    float sum=0;
    int n;
    cin>>n;
    while (n>=0){
        sum=sum+(1/pow(2,n));
        n=n-1;
    }
    cout<<sum;
}
```

às vezes, surge uma sequência como essa, mas com alternância de sinais: $+ - + -$ .... Se for o caso, basta incluir na expressão $(-1)^i$ como fator. No exemplo acima ficaria$X_i = (-1)^i \times \frac{1}{2^i}$. Se a paridade é invertida, basta elevar a $i+1$ ou $X_i = (-1)^{i+1} \times \frac{1}{2^i}$

Treinando a manipulação de séries Para cada série numérica abaixo, escreva o termo genérico em função de i e depois escreva um programa C++ que calcule a soma dos 7 termos iniciais.

T_1	T_2	T_3	T_4	T_5	$\sum_1^7$	termo		
10/11	12/13	14/15	16/17	18/19	6.562948749	$exp = 2\times n/(2\times n)+1$	$n=5$	$inc=1$
2/1	10/9	26/25	50/49	82/81	8.198046576	$exp = (n*2)+1/n*2$	$n=1$	$inc=2$
24/25	48/49	80/81	120/121	168/169	6.905159883	$exp = (n*2)-1/n*2$	$n=5$	$inc=2$
2/3	8/9	14/15	20/21	26/27	6.348288748	$exp = 2\times n/(2\times n)+1$	$n=1$	$inc=3$
10/15	12/18	14/21	16/24	18/27	4.666666667	$exp = 2\times n/3\times n$	$n=5$	$inc=1$
10/9	37/36	82/81	145/144	226/225	7.16797745	$exp = (n*2)+1/n*2$	$n=3$	$inc=3$
2/3	5/5	10/7	17/9	26/11	13.52725053	$exp = (n*2)+1/(2\times n)+1$	$n=1$	$inc=1$
2/3	6/7	10/11	14/15	18/19	6.233086889	$exp = 2\times n/(2\times n)+1$	$n=1$	$inc=2$
26/15	37/18	50/21	65/24	82/27	18.978848	$exp = (n*2)+1/3\times n$	$n=5$	$inc=1$
2/3	10/7	26/11	50/15	82/19	23.70864139	$exp = (n*2)+1/(2\times n)+1$	$n=1$	$inc=2$
2/1	6/9	10/25	14/49	18/81	3.91026751	$exp = 2\times n/n*2$	$n=1$	$inc=2$
24/25	35/36	48/49	63/64	80/81	6.865578917	$exp = (n*2)-1/n*2$	$n=5$	$inc=1$
10/7	26/11	50/15	82/19	122/23	30.3322973	$exp = (n*2)+1/(2\times n)+1$	$n=3$	$inc=2$
15/16	24/25	35/36	48/49	63/64	6.81134338	$exp = (n*2)-1/n*2$	$n=4$	$inc=1$
2/3	6/9	10/15	14/21	18/27	4.666666667	$exp = 2\times n/3\times n$	$n=1$	$inc=2$
6/7	12/13	18/19	24/25	30/31	6.605047296	$exp = 2\times n/(2\times n)+1$	$n=3$	$inc=3$
26/25	50/49	82/81	122/121	170/169	7.094840117	$exp = (n*2)+1/n*2$	$n=5$	$inc=2$
3/5	24/11	63/17	120/23	195/29	36.38950852	$exp = (n*2)-1/(2\times n)+1$	$n=2$	$inc=3$
10/7	17/9	26/11	37/13	50/15	19.99990275	$exp = (n*2)+1/(2\times n)+1$	$n=3$	$inc=1$
2/3	5/6	10/9	17/12	26/15	10.19761905	$exp = (n*2)+1/3\times n$	$n=1$	$inc=1$
26/11	50/15	82/19	122/23	170/27	37.18944016	$exp = (n*2)+1/(2\times n)+1$	$n=5$	$inc=2$
17/9	50/15	101/21	170/27	257/33	44.17575018	$exp = (n*2)+1/(2\times n)+1$	$n=4$	$inc=3$
2/1	17/16	50/49	101/100	170/169	8.105501656	$exp = (n*2)+1/n*2$	$n=1$	$inc=3$
0/3	15/12	48/21	99/30	168/39	22.77169607	$exp = (n*2)-1/3\times n$	$n=1$	$inc=3$
8/9	12/13	16/17	20/21	24/25	6.600737446	$exp = 2\times n/(2\times n)+1$	$n=4$	$inc=2$
3/5	8/7	15/9	24/11	35/13	15.18953204	$exp = (n*2)-1/(2\times n)+1$	$n=2$	$inc=1$
17/12	37/18	65/24	101/30	145/36	23.61964286	$exp = (n*2)+1/3\times n$	$n=4$	$inc=2$

Para gerar mais: DyalogAPL:)load montaserie.dws, montaserie quantas vezes quiser

Uma possível solução (para UMA das linhas acima)

```
int main() {
  float sum=0,num,den;
  int i;
  float n;
  n=5;
  for(i=0;i<7;i++){
        num=2.0*n;
        den=(2*n)+1.0;
        cout<<num<<' '<<den<<endl;
    sum=sum+(num/den);
    n=n+1;
  }
  cout<<sum;
}
```

6.37 (achamaior)

Escreva um programa C++ que leia uma sequência de números positivos inteiros terminados por 0 e imprima o maior valor lido.

```
int main() {
   int n,mai;
   mai=-999999;
   cin>>n;
   while(n!=0){
       if (n>mai){
           mai=n;
       }
       cin>>n;
   }
   cout<<mai;
}
```

6.38 (invordem)

... que leia uma sequência de linhas cada uma contendo 3 reais. Escrever os 3 números lidos em ordem inversa. Acaba quando for lida a linha 0,0,0.

6.39 (sucessorpar)

... leia números inteiros até ler 0. Para cada número lido, imprimir o número lido e o seu sucessor par. Não processar o 0.

```
int main(){
   int n;
   cin>>n;
   while(n!=0){
      if ((n%2)==0){cout<<n<<" "<<n+2<<endl;}
      else {cout<<n<<" "<<n+1<<endl;}
      cin>>n;
   }
   return 0;
}
```

6.40 (ehprimo)

... que leia o valor positivo inteiro n e imprima "SIM"se n for primo e imprima "NAO"se n for 0, 1, ou composto.

```
int main() {
   int n,divs,valor;
   cin>>n;
   divs=0;
```

```
   valor=2;
   while(valor<n){
      if ((n%valor)==0){
           divs=divs+1;
      }
      valor=valor+1;
   }
   if (divs==0){
       cout<<"primo"<<endl;
   }
   else {
       cout<<"composto"<<endl;
   }
}
```

Capítulo 7

Funções

7.1 Funções

Há ótimas motivações para usar funções dentro de programas:

- reaproveitar código (escrever menos)
- evitar a repetição (evitar erro em uma das versões)
- legibilidade: coesão e acoplamento. Definições: Coesão: tudo o que resolve alguma coisa está junto e acoplamento: coisas que resolvem coisas diferentes não estão juntas. O ideal, para que um sistema seja flexível e compreensível, são módulos com alta coesão e baixo acoplamento.
- lidar com a complexidade crescente do software.[1]

De fato, uma das funções na indústria de software mais bem remuneradas é o arquiteto de software, cuja função principal é quebrar uma funcionalidade complexa em partes e sub-partes e sub-sub-partes que sejam manuseáveis e testáveis isoladamente.

Como acontece a execução? Quando a função main() recebe o controle ela começa na primeira linha. Acaba, como QUALQUER função quando se executa seu último comando OU quando se executa o comando `return`, o que acontecer antes. Quando ao executar o código, o processador acha uma CHAMADA a uma função, ele guarda o endereço do próximo comando na pilha de retorno e desvia para a primeira linha da função chamada. Isso ocorre quantas vezes for necessário sem que o processador se perca.

Obviamente, para que isto funcione, o compilador precisa ter recebido a definição da função ANTES que ela seja chamada. Se não, vai haver um erro de compilação. Este erro pode ser eliminado escrevendo-se um protótipo da função antes e adiando a sua definição. Como o uso de protótipos pode ser uma fonte desnecessária de erros, vamos evitar seu uso. Portanto, agora para nós vale a regra: **antes de usar uma função ela terá que ser definida**. O corolário aqui, é que a função `main()` deverá ser a última a ser definida.

Eis como fica o lay-out básico de programas que usam funções:

```
#includes
#defines
using namespace std;
<retorno> função-1(parâmetros) {
  ...
  return ...
}
int main() {
  ...
  usa-se a função-1
  [return 0;]
}
```

Uma função é um bloco autocontido dentro de um programa (ou de uma biblioteca) que pode operar na modalidade caixa preta (isto é, seu usuário, não precisa saber como ela funciona, apenas o que faz e qual é a sua interface). [2]

[1] Imagine um programa como o MS Word (presumidas 7.000.000 de linhas de código) sem usar funções...

[2] Interface é uma palavra omnipresente na ciência da computação e ela sugere como objetos de software se relacionam entre sí, isto é, o que eles cedem e recebem. A título de exemplo (meio pobre), pode-se dizer que a interface entre um carro e um motorista seriam a direção, os pedais, as alavancas bem como os marcadores e indicadores do painel

O conceito de função é chave para quebrar a complexidade crescente dos programas de computador em unidades autônomas e manejáveis. A última vez em que soube, o Microsoft Word andava na casa dos 4.5 milhões de linhas de código, há coisa de mais de 10 anos. Imagine escrever e sobretudo manter um programa deste tamanho se o mesmo for atacada em bloco, o que se chama de programação espagueti. Uma função é um pedaço de programa que:

- Tem um nome único e possivelmente auto-explicativo
- Realiza uma única computação
- Recebe um conjunto de dados de entrada necessários para fazer a computação acima
- Devolve um ou mais resultados a quem o chamou
- Pode ser construída, testada e mantida de maneira independente em relação aos programas que a usam
- Pode ser usada em diversos locais do mesmo programa, com isto evitando duplicar linhas de código
- Garante que todos os usuários usem o mesmo código, já que ele está escrito em um único local (a função)

Note que todos estamos acostumados com este conceito desde que tenhamos estudado matemática além da 6 a série do fundamental. Quando se fala lá na matemática na função raiz quadrada, está se falando de uma função única (calcular a raiz quadrada), que recebe um valor (aquele de quem se quer obter a raiz) e devolve um resultado (a raiz quadrada calculada). Tudo isto opera na modalidade de caixa preta, já que bem poucas pessoas conhecem como se calcula uma raiz. Ao escrever $\sqrt{x}$, todos reconhecem o que é isso e não perdem um segundo sequer ao estudar como essa função opera. Na programação, a coisa é parecida, mas mais flexível, já que se a lista de funções da matemática embora grande, é mais ou menos estável, na programação é completamente flexível e opera ao gosto do arquiteto de sistemas, ou em casos mais simples, do programador. Alías, quebrar um programa grande em um conjunto coerente e integrado de funções é habilidade importante e convenientemente valorizada pela indústria de software. Quem programa em C (ou C++) já usa e abusa do conceito de função. Todo programa C precisa ter uma função de nome main() e essa função é que recebe o controle quando o programa que a contém é chamado. Programas mais simples têm apenas a função main, mas nada impede que um programa mais complexo tenha dezenas ou centenas de funções umas chamando as outras, cedendo e recebendo dados. A regra acima entretanto permanece válida. Todo programa C tem que ter uma função principal chamada main. A anatomia de uma função é:

```
tipo-resposta nome-função(parâmetros) {
... computação realizada pela função ...
return variavel-resposta
}
```

Neste formato cabem as seguintes definições:

tipo-resposta indica qual o tipo do dado que a função deverá retornar quando for encerrada. Pode ser qualquer um dos tipos já estudados (como por exemplo int ou char...)

nome-função um nome qualquer, sujeito às regras de formação de nomes da linguagem. Espera-se que seja autoexplicativo.

() ao lado do nome indica que este nome é uma função. Este formalismo existe para impedir que este nome seja confundido com uma variável, já que a regra de formação de nomes é a mesma. Se os parênteses aparecem sem nada dentro, como em main() isto significa que esta função não recebe parâmetros.

parâmetros tipos e nomes dos diversos parâmetros que a função deverá receber quando for chamada. Se houver mais de um, deverão estar separados por uma vírgula.

...computação... tudo o que você já aprendeu e sabe sobre a linguagem C++ pode e deve ser usado aqui.

return variável-resposta Este comando faz várias coisas:

- Retorna o valor que existe na variável citada para quem chamou esta função
- Encerra esta função, mesmo que haja mais comandos abaixo do return que, portanto, jamais serão executados
- Libera todos os recursos anteriormente alocados à execução desta função.

O ciclo de vida de uma função sugere 2 eventos separados e imprescindíveis. O primeiro é a definição da função, através do formalismo visto aí encima. Mas, só definir uma função, não causa nenhum processamento. Este ocorre num segundo momento, quando a função é chamada dentro do programa (ou dentro de outra função), momento no qual finalmente a computação descrita dentro da função é executada. A regra aqui é que antes de executar ou chamar uma função, a mesma terá que ter sido definida. Vale, entretanto, a lembrança de que esta definição eventualmente poderá estar em outro módulo e até ser meio "invisível" para quem a está chamando. Não é exatamente o caso, mas temos algo parecido para dar como exemplo. Os "comandos" cout e cin não são funções, mas podem ser entendidos como se o fossem. Notem

que eles precisam estar definidos antes de serem usados, mas a definição não é feita por nós e sim apenas agregada ao nosso programa. Quem faz isso é o comando `#include<iostream>`. A esse respeito, se você usar um cout no seu programa sem colocar o `#include` no programa receberá um erro dizendo que cout não está definida. Depois que a função foi corretamente definida, ela pode ser usada quantas vezes você quiser, sempre realizando a mesma computação para a qual foi escrita operando a cada vez sobre os parâmetros passados a ela naquela execução.

7.1.1 Exemplo de pitágoras

Veja-se um exemplo do que se diz

```
#include<iostream>
#include<cmath>
using namespace std;
float pitagoras(float cateto1, float cateto2) {
  float a,b;
  a = cateto1 * cateto1;
  b = cateto2 * cateto2;
  return sqrt(a+b);
}
int main() {
  float c1,c2,hip;
  cout<<"Calculo de uma hipotenusa "<<endl;
  cin>>c1;
  cin>>c2;
  hip = pitagoras(c1,c2);
  cout<<"Hipotenusa "<<hip;
}
```

Quem é quem neste exemplo

pitagoras	é uma função definida pelo usuário que recebe 2 parâmetros float, calcula alguma coisa e devolve um float.
cateto1	é o primeiro parâmetro que deve ser um float.
cateto2	o segundo parâmetro, também float.
a,b	variáveis internas à função pitagoras só começam a existir no momento em que a função começa a operar e deixam de existir após a execução do return.
operações envolvendo os parâmetros	os parâmetros citados no cabeçalho são usados extensivamente no corpo da função. Deve-se notar que os parâmetros não existem fisicamente na memória da máquina. Só vão existir quando a função for chamada.
sqrt(a+b)	calcula o resultado final desta função
return	devolve o resultado final para quem chamou esta função e encerra a função. main a função principal deste programa
c1,c2 e hip	variáveis dentro do programa. Existem fisicamente na memória do computador cout e cin obtenção de dados externos e oferecimento dos mesmos
hip	a variável onde o resultado da função vai ser guardado
pitagoras	a função acima definida sendo chamada. É só agora que ela executa alguma computação
c1,c2	os valores que vão ser repassados à função. A partir deste ponto é como se cateto1 recebesse (ele recebe) uma cópia do conteúdo de c1. Igualmente cateto2 recebe uma cópia do conteúdo de c2

Observação: note que tanto na definição quanto na chamada, pitagoras não tem acento. É uma boa estratégia. Depois que foi definida, pitagoras funciona como se fosse uma função primitiva do C++, tal como sqrt.

7.2 Mais uma implementação do CPF

Agora usando uma função

```
#include<iostream>
using namespace std;
int cpf(int n){
   int n1,n2,n3,n4,n5,n6,n7,n8,n9,d1,d2,t1,t2,p1,p2;
   n9=n%10;
   n=n/10;
   n8=n%10;
```

```
   n=n/10;
   n7=n%10;
   n=n/10;
   n6=n%10;
   n=n/10;
   n5=n%10;
   n=n/10;
   n4=n%10;
   n=n/10;
   n3=n%10;
   n=n/10;
   n2=n%10;
   n=n/10;
   n1=n%10;
   p1=(n1*10)+(n2*9)+(n3*8)+(n4*7)+(n5*6)+(n6*5)+(n7*4)+(n8*3)+(n9*2);
   t1=p1%11;
   if ((t1==0)||(t1==1)){d1=0;}
      else {d1=11-t1;}
   p2=(n1*11)+(n2*10)+(n3*9)+(n4*8)+(n5*7)+(n6*6)+(n7*5)+(n8*4)+(n9*3)+(d1*2);
   t2=p2%11;
   if ((t2==0)||(t2==1)){
      d2=0;} else {d2=11-t2;}
   return (d1*10)+d2;
}
int main(){
   int a;
   cin>>a;
   cout<<cpf(a)<<endl;
   cout<<"acabou";}
```

7.3 egipcios

Os egipcios antigos (há mais de 5.000 anos) já conheciam o conceito de π – não com esse nome, claro, mas como a constante que iguala o comprimento de uma circunferência ao seu raio. Como eles não dominavam a aritmética real que temos hoje, π era representado por uma fração ordinária, nomeadamente 22/7.

Seu problema aqui é receber um número real qualquer com bastantes casas decimais e localizar qual a fração de números inteiros que chega mais perto do valor do número real original.

Numa primeira abordagem ambos – numerador e denominador – devem ser menores do que 100 (ou seja, com 2 dígitos) e no segundo caso devem ser menores do que 1000 (ou seja, com 3 dígitos).

Veja-se alguns exemplos

número	n	d
3.141592654 c/ 2 dígitos	22	7
3.141592654 c/ 3 dígitos	355	113
2.718281828 c/ 2 dígitos	87	32
2.718281828 c/ 3 dígitos	878	323
5.67891234 c/ 2 dígitos	91	16
5.67891234 c/ 3 dígitos	619	109

Escreva um programa em C++ que ache as frações inteiras para um determinado real fornecido, no primeiro caso com números de 2 dígitos e no segundo caso com números de 3 dígitos.

```
int egipcio(float nr, int lim){
   float minimo=+9999999;
   float ca,i,j,mini,minj;
   i=1.0;
   while (i<lim){
      j = 1.0;
      while (j<lim){
          ca=i/j;
          if ((abs(nr-ca))<minimo){
             minimo=abs(nr-ca);
             mini=i;
             minj=j;
```

```
            }
            j=j+1;
        }
        i=i+1;
    }
    cout<<mini<<' '<<minj<<endl;

}
int main(){
    cout<<"Primeiro "<<endl;
    egipcio(4.49573979,100);
    cout<<"Segundo "<<endl;
    egipcio(4.49573979,1000);
}
```

7.4 (Número Perfeito)

Escreva um programa C++ que imprima os números perfeitos menores que 10000. n é perfeito se $n = \sum$(divisores próprios de n). Os divisores próprios são os números que dividem n, à excessão do próprio n. Por exemplo, os divisores próprios de 6 são: 1, 2 e 3. Que somados são igual a 6, logo 6 é um número perfeito. Os divisores próprios de 8 são: 1, 2 e 4, que somados dão 7, logo 8 não é perfeito.

```
int divisores(int n){
  int x,som;
  som=1;
  x=2;
  while (x<n){
    if ((n%x)==0){som=som+x;}
    x++;
  }
  return som;
}
int main() {
  int sem=1;
  while (sem<100000){
    if (sem==divisores(sem)){cout<<sem<<" ";}
    sem++;
  }
}   // são eles: 1,6,28,496,8128
```

7.5 (Números Amigos)

Escreva um programa C++ que ache os números amigos menores que 10000 que são amigos. Dois números são amigos quando um deles for igual à soma dos divisores próprios do outro e vice-versa. É claro que um número não é amigo dele mesmo. Por exemplo 1184 é amigo de 1210 (divisores de 1184: 1 2 4 8 16 32 37 74 148 296 592. Divisores de 1210: 1 2 5 10 11 22 55 110 121 242 605). Note que a análise entre 1..10000 demorou 2.139 segundos (36 minutos) em uma CPU rápida.

```
int divisores(int n){
  int x,som;
  som=1;
  x=2;
  while (x<n){
    if ((n%x)==0){som=som+x;}
    x++;
  }
  return som;
}
int main(){
  int m,n;
  for (m=1;m<10000;m++){
    for (n=1;n<10000;n++){
```

```
       if (m==divisores(n)&&n==divisores(m)&&(m!=n)){
             cout<<m<<" amigo de "<< n<<endl;
       }
    }
  }
}
Os amigos achados são: 220,284; 1184,1210; 2620,2924; 5020,5564; 6232,6368
```

Dicas de melhorias para diminuir o tempo alto:

- Contar os divisores em pares ($a_1 \times a_2 = n$), com isso a busca pode acabar em $\sqrt{n}$ sem precisar chegar até n. Só que como não existe almoço grátis, há que se operar com os quadrados perfeitos.
- O segundo `for` não precisa ir até 10.000 podendo ir até o valor do for externo (ou no caso m). Só com esta mudança o tempo passou de 2139 para 712 segundos (de 36 para 12 minutos).

7.6 (Próximo primo)

Escreva um programa C++ que leia n e imprima o próximo primo $\geq n$.

```
 using namespace std;
    int primo4(long long x){
        int r,f;
      if (x==1){
        return 0;
      }
      if (x<4){
        return 1;
      }
      if (x%2==0){
        return 0;
      }
      if (x<9){
        return 1;
      }
      if (x%3==0){
        return 0;
      }
      r=sqrt(x);
      f=5;
      while (f<=r){
        if (x%f==0){
            return 0;
        }
        if ((x%(f+2))==0){
            return 0;
        }
        f=f+6;
      }
      return 1;
    }
int main(){
  long long qual;
  while (1==1){
    cout<<"Informe ";
    cin>>qual;
    if (qual==0){
        break;
    }
    while(1==1){
        if (primo4(qual)){
           cout<<"O proximo primo e "<<qual<<endl;
        break;
        }
```

```
        qual++;
    }
}
}
```

7.7 Notas da ginástica

Escreva um programa C++ que leia 5 notas em uma prova de ginástica olímpica. O programa deve calcular a média das 3 notas que sobram depois que se eliminam a maior e a menor notas dadas. Por exemplo:

n_1	n_2	n_3	n_4	n_5	média
10	10	9	9	8	$28 \div 3$
5	6	7	8	9	$21 \div 3$
10	10	10	10	10	$30 \div 3$
9	9	9	10	10	$28 \div 3$

```
#include<iostream>
using namespace std;
void selec (float v[], int tam) {
  int i,j,cor,inui;
  i=0;
  while (i<tam){
    cor=v[i];
    inui=i;
    j=i+1;
    while (j<tam){
      if (v[j]<cor){
        cor=v[j];
        inui=j;
      }
      j++;
    }
    swap(v[i],v[inui]);
    i++;
  }
}
int main(){
  float nota[5],med;
  cin>>nota[0]>>nota[1]>>nota[2]>>nota[3]>>nota[4];
  selec(nota,5);
  med=(nota[1]+nota[2]+nota[3])/3.0;
  cout<<"Media = "<<med;
}
```

Ou uma versão alternativa usando o sort do C++

```
#include<iostream>
#include<algorithm>
using namespace std;
int main(){
  float nota[5],med;
  cin>>nota[0]>>nota[1]>>nota[2]>>nota[3]>>nota[4];
  sort(nota,nota+5);
  med=(nota[1]+nota[2]+nota[3])/3.0;
  cout<<"Media = "<<med;
}
```

7.8 (Raiz quadrada nova)

Suponha que C++ não tem a função raiz-quadrada. Como seria possível construir uma ? Lembrando que dado um número real x sua raiz é y se e somente se $y^2 = x$. Reescrevendo $y = \frac{x}{y}$. Isto sugere um possível algoritmo:

- chute um valor g para y

- Calcule $\frac{x}{g}$
- Se $\frac{x}{g}$ é suficiente perto de g retorne-o, senão tente um chute melhor.

O chute inicial pode ser qualquer número, mas vamos padronizar como sendo 1. Eis como pode ficar

```
int perto(float a, float b){
   return b*0.001 > abs(a-b);
   }
float melhor(float a, float b){
   return (b+(a/b))/2.0;
   }
float testa(float a, float b){
   if (perto((a/b),b)){
      return b;}
   else {
      return testa(a, melhor(a,b));
   }
}
int main(){
   float a;
   cin>>a;
   cout<<testa(a,1);
}
```

7.9 Passagem por valor

Passagem por valor é quando uma variável é copiada em uma área temporária antes de ser passada para uma função como parâmetro. Isto significa que quaisquer alterações sobre a variável dentro da função não causarão qualquer efeito sobre a variável original. Acompanhe

```
int separa(int x){
  int x2;
  x2=x%10;
  x=x/10;
  return x2;
}
int main(){
  int a,b;
  cin>>a;
  b=separa(a);
  cout<<a;
}
```

Repare que na função `main()` se for oferecido o valor 93 para a função `separa`, ela devolverá 3 (a unidade em 93). Logo após executar a função `separa` a função `main` imprime o valor da variável a que continua sendo 93. Isto porque, dentro da função `separa` quando ela divide $x/10$ está se dividindo uma cópia de a (com o nome de x) e não a variável a original.

Este mecanismo existe para empacotar as funções e garantir que eventuais erros cometidos por elas não se propaguem a outras partes do código. O nome desta técnica é **passagem por valor** e ele é o padrão de chamadas de funções, não só em $C++$ como praticamente em todas as demais linguagens de programação. Acompanhe a seguir um outro exemplo.

Quando se escreve uma função em C++, normalmente os parâmetros são passados por valor e o retorno idem. Assim, em

```
int dobro(int N){
  return N*2;
}
int main(){
  int x;
  x=55;
  cout<<dobro(x);
}
```

No exemplo acima, a função dobro desconhece a variável x, enquanto que a função principal desconhece uma possível variável N. Assim, se a função for chamada com 55, ela vai retornar 110, como era esperado. Vamos elaborar mais um pouco este exemplo, como em

```
int dobro(int N){
  N=N+10;
  return N*2;
}
int main(){
  int x;
  x=55;
  cout<<dobro(x)<<endl;
  cout<<x<<endl;
}
```

Note que a função `dobro`, antes de dobrar o valor do parâmetro recebido, soma 10 unidades a ele. E retorna o dobro deste novo valor. Quando esta nova função for chamada, ela fará isso corretamente, mas o valor da variável original (aquela que estava definida na função chamadora, no caso, a função main, NÃO É ALTERADA. Chamando este trecho com 55, a função dobro vai somar 10 (obtendo 65) e depois vai retornar 130. A função principal vai imprimir 130, mas depois ao imprimir o valor de x, teremos 55, e não 65. A soma de 10 unidades ficou restrita à função "dobro" e não teve seu efeito propagado para a função chamadora. Esta é a melhor maneira de trabalhar minimizando os efeitos colaterais e garantindo que cada trecho de código tem encapsulados os seus dados.

7.10 Passagem por referência

Entretanto, eventualmente, pode ser necessário abrir uma exceção a esta regra (a que permite que funções alterem os dados usados para chamar a função). Esta ação pode ser necessária também quando mais do que um valor precisa ser retornado. Agora, ao invés de return, poderemos retornar os resultados diretamente sobre os dados passados (que agora podem ser alterados pela função chamada). A chave para fazer isto é PASSAGEM DE VALORES POR REFERÊNCIA.

Quando se usa a passagem por referência, o que vai para a função não é uma cópia das variáveis, mas sim o endereço real que elas ocupam na memória. Agora só há uma instância da variável, e portanto quailquer mudança que ela sofrer se refletirá em todos os lugares onde a variável é mencionada.

Se na definição de uma função, colocarmos um & entre o tipo e o nome do parâmetro, estamos avisando ao compilador que quando a função for chamada, devem ser passados a ela não os valores da chamada, mas sim o endereço real das variáveis na memória. Logo, neste caso, eventuais alterações feitas pela função chamada serão preservadas para a função chamadora. Veja-se o exemplo

```
int dobro(int & N){
  N=N+10;
  return N*2;
}
int main(){
  int x;
  x=55;
  cout<<dobro(x)<<endl;
  cout<<x<<endl;
}
```

Agora, chamando a função com 55, a variável x passa a valer 55. Quando a função dobro é chamada, é a variável x que é instanciada dentro da função dobro. Quando a variável N (na realidade x) recebe o valor original (55) mais 10 ela passa a valer 65. A função retorna o dobro disso, mas a variável x original passa a valer 65. Esta habilidade é frequentemente necessária quando uma função precisa retornar mais do que um valor (quando então usaria o comando return). Agora, se for necessário devolver 2 valores após a chamada de uma função, será preciso:

- passar essas 2 variáveis para a função por referência (usando &)
- garantir que a função altere essas duas variáveis corretamente
- Não é necessário devolver nada via return
- Garantir que a função chamadora busque os resultados da aplicação da função diretamente nas variáveis chamadas

7.11 Função de troco

Por exemplo, vamos escrever uma função que receba um valor em reais e informe quantas notas de 50, 20 e 10 reais devem ser usadas para compor o valor original. Se não for possível compor o valor com essas 3 notas, deve retornar -1 nas 3 notas. Veja no exemplo:

140 deve retornar 2,2,0; significando 2 notas de 50, 2 notas de 20 e nenhuma nota de 10 reais.

200 deve retornar 4,0,0; significando 4 notas de 50.

33 deve retornar -1,-1,-1; significando que com notas de 50,20 e 10 não é possível compor o valor 33.

Eis como ficaria

```
int troco(float valor,int & n50, int & n20, int & n10){
  n50=0;
  n20=0;
  n10=0;
  while (valor>=50){
    n50=n50+1;
    valor=valor-50;
  }
  while (valor>=20){
    n20=n20+1;
    valor=valor-20;
  }
  while (valor>=10){
    n10=n10+1;
    valor=valor-10;
  }
  if (valor !=0) {
    n50=-1;
    n20=-1;
    n10=-1;
  }
}
int main(){
  float t;
  int a,b,c;
  t=200.0;
  troco(t,a,b,c);
  cout<<t<<' '<<a<<' '<<b<<' '<<c<<endl;
}
```

Aqui a resposta será 200 4 0 0 mostrando que as variáveis a, b e c tiveram seus valores corretamente estabelecidos.

7.12 Função: terrenos retangulares

Mais um exemplo: Você tem inúmeros terrenos retangulares em uma determinada cidade, Curitiba, por exemplo. Cada terreno é descrito por 2 medidas, a largura e o comprimento. Você gostaria de gerar uma tabela mostrando para cada terreno sua área e seu perímetro. Como você aprendeu recentemente a usar funções, gostaria de criar uma função que achasse esses 2 parâmetros. Eis como ficaria o relatório

```
20,100 ------ area 2000 perimetro 240
100,100 ------ area 10000 perimetro 400
1,400 ------ area 400 perimetro 802
```

Eis como ficaria

```
int areaper(float & larg, float & compr, float & area, float & perim){
  area = larg * compr;
  perim = (2*larg)+(2*compr);
}
int main() {
  float a,b,c,d;
  a=20;
  b=100;
  areaper(a,b,c,d);
  cout<<a<<','<<b<<" ------ area "<<c<<" perimetro "<<d<<endl;
  a=100;
  b=100;
  areaper(a,b,c,d);
  cout<<a<<','<<b<<" ------ area "<<c<<" perimetro "<<d<<endl;
```

```
  a=1;
  b=400;
  areaper(a,b,c,d);
  cout<<a<<','<<b<<" ------ area "<<c<<" perimetro "<<d<<endl;
}
```

7.13 Função: Celsius e Farenheit

Mais um exemplo: Suponha que você ganhou um termômetro que mostra a temperatura ambiente em graus Celsius (centígrados). Você gostaria de disponibilizar outras 2 temperaturas: a absoluta que também é centígrada, mas cujo ponto inicial é diferente da temperatura Celsius e também a temperatura Farenheit. Para isso gostaria de criar uma função C++ que recebesse a temperatura Celsius e devolvesse a absoluta e a Farenheit. Use as seguintes fórmulas:

$$A = C + 273.15$$

$$F = (C * 1.8) + 32$$

Eis como ficaria:

```
int celfar(float celsius, float & absol, float & faren){
  absol = celsius+273.15;
  faren = (celsius*1.8)+32;
}
int main() {
  float a,f;
  celfar(100,a,f);
  cout<<"Ebulicao: 100 "<<a<<' '<<f<<endl;
  float c = 37;
  celfar(c,a,f);
  cout<<"Corpo humano: "<<c<<' '<<a<<' '<<f<<endl;
}
```

7.14 Imposto de consumo e de serviços

Um exemplo um pouco mais elaborado: Suponha que na sua cidade e no seu país há 2 impostos: o de consumo (federal) que corresponde a 17% do valor da venda, com isenção de valores de venda menores que ou iguais a 100,00 reais. Há também o imposto de serviços (municipal) que é calculado sempre como sendo de 10% ou de %20 do valor bruto da venda a depender do valor da venda: maiores que 1000 reais pagam 20% e menores do que isso pagam 10%. Para calcular o imposto municipal, considera-se o valor bruto da venda, desconsiderando neste caso o imposto federal. Por exemplo, uma venda de 1500 reais, pagará 17% de consumo (porque $1500 > 100$) ou 255,00 de imposto federal e também 300 reais de imposto municipal (já que $1500 > 1000$). Escreva uma função que receba o valor da venda e devolva os dois impostos: o federal e o municipal.

```
int fedemuni(float valor, float & fede, float & muni){
  fede=0;
  if (valor > 100) {
    fede = valor * 0.17;
  }
  if (valor > 1000) {
    muni = valor*0.2;
  }
  else {
    muni = valor*0.1;
  }
}
int main() {
  float v,f,m;
  v=1500;
  fedemuni(v,f,m);
  cout<<"Venda: "<<v<<" Federal: "<<f<<" municipal: "<<m<<endl;
```

7.15 O mesmo problema dos dois jeitos

Acompanhe a seguir o mesmo problema sendo resolvido por valor e por referência. Sempre que possível a primeira alternativa é a preferida por ser a mais segura. O problema é dado um número de 2 dígitos devolver os próximos primos acima e abaixo dele. Por exemplo se chamarmos a função com 75, ela deve devolver 73 e 79. Se chamarmos com 99, ela deve devolver 97 e 101 e assim por diante. A primeira solução é com passagem por referência, acompanhe

```
#include<iostream>
using namespace std;
int primo3(int n){int f,i,r,j;
// conforme definida anteriormente
}
int vizipri(int x, int & p1, int & p2){
    int salva;
    salva=x;
    while (! primo3(salva)){
        salva--;
    }
    p1=salva;
    salva=x;
    while (! primo3(salva)){
        salva++;
    }
    p2=salva;
}

int main(){
  int alfa,np1,np2;
  cin>>alfa;
  vizipri(alfa,np1,np2);
  cout<<np1<<" "<<np2<<endl;
}
```

Agora vai-se resolver o mesmo problema com a passagem por valor. Como só se pode devolver uma única variável, vai-se devolver um float, no qual antes do ponto (a parte inteira) será o primeiro primo (entre 1 e 99) e a parte fracionária do resultado será o primo procurado dividido por 1000 e somado com o primo anterior. Acompanhe

```
#include<iostream>
#include<cmath>
using namespace std;
int primo3(int n){int f,i,r,j;
// considera-se definido
}

float vizipri(int x){
    int salva;
    float p1,p2;
    salva=x;
    while (! primo3(salva)){
        salva--;
    }
    p1=salva;
    salva=x;
    while (! primo3(salva)){
        salva++;
    }
    p2=salva;
    cout<<"p2"<<p2<<endl;
    return p1+(p2/100000.0); //só 1000 pode dar erro de arredondamento
}

int main(){
  int alfa,np1,np2;
  float res;
```

```
  cin>>alfa;
  res=vizipri(alfa);
  np1=res;
  np2=(res-np1)*100000; // idem idem
  cout<<np1<<" "<<np2<<endl;
}
```

7.16 (primosnk)

Escreva um programa C++ que leia n e k ($n \geq k$, não precisa testar) e imprima a soma dos números primos existentes entre n e k inclusive.

```
int eprimo(int n) {
   int divs,valor;
   divs=0;
   valor=2;
   while(valor<n){
      if ((n%valor)==0){
           divs=divs+1;
      }
      valor=valor+1;
   }
   return divs==0;
}
int main(){
   int N, K;
   int soma=0;
   int xave;
   cin>>N>>K;
   xave=N;
   while (xave<=K){
      if (eprimo(xave)){
        soma=soma+xave;
      }
      xave=xave+1;
   }
   cout<<soma;
}
```

7.17 (neperiano)

Sabendo que

$$e = 1 + \frac{1}{1!} + \frac{1}{2!} + \frac{1}{3!} + ...$$

escreva um programa que leia n, inteiro e ≥ 2 e imprima o valor de e calculado com n parcelas.

```
int fatr(int x){ // solucao recursiva
  if (x<2){
      return 1;
  }
  else{
      return x*fatr(x-1);
  }
}
int main() {
  float nepe=0.0;
  float parc=1;
  int k=0;
  while (k<10){
    nepe=nepe+1.0/fatr(k);
    k=k+1;
  }
```

```
  cout.precision(20);
  cout<<nepe;
}
```

7.18 (neper2)

... que leia *eps* (*eps* <= 0.1) e calcule e imprima o valor de e até que a parcela individual seja menor do que *eps*. Imprima o valor de e e quantidade de parcelas usadas no seu cálculo. Se quiser ajudar na depuração, imprima também os valores das parcelas.

```
int fatr(int x){ // solucao recursiva
  if (x<2){
      return 1;
  }
  else{
      return x*fatr(x-1);
  }
}
int main() {
  float nepe=0.0;
  float parc=1;
  int k=0;
  while (1==1){
    parc = 1.0/fatr(k);
    if (parc<0.0001){
        break;
    }
    nepe=nepe+parc;
    // cout<<parc<<endl; em tempo de depuracao
    k=k+1;
  }
  cout.precision(20);
  cout<<nepe<<endl;
  cout<<"Parcelas: "<<k;
}
```

7.19 (coseno)

Sabendo que o coseno pode ser calculado como:

$$cos(x) = \sum_{i=0}^{n} (-1)^i \frac{x^{2\times i}}{(2 \times i)!}$$

escreva um programa que leia o valor do arco x ($-\pi \leq x \leq \pi$), calcule o seu coseno com 6 parcelas e imprima o valor do coseno.

```
int fatr(int x){ // solucao recursiva
  if (x<2){
      return 1;
  }
  else{
      return x*fatr(x-1);
  }
}
float pc(int i,float ang){
   return (pow(-1,i))*(pow(ang,(2*i))/fatr(2*i));
   }
int main(){
   float soma=0.0;
   int k=0;
   float a;
   cin>>a;
```

```
   while(k<8.0){
      soma=soma+pc(k,a);
      cout<<"p "<<k<<" ang "<<a<<" parcela "<<pc(k,a)<<endl;
      k=k+1;
   }
   cout<<"Resultado final: "<<soma;
}
```

7.20 (centena)

Escreva um programa C++ que leia uma sequência de inteiros (até chegar a sentinela 0) e para cada valor N lido imprima o seu dígito das centenas.

```
#include<iostream>
using namespace std;
int centena(int n){
   int x,y;  // se n é 12345
   y=n/100;  // y e 123 (n e y sao int)
   x=y%10;   // x e 3   (x e a resposta sao int)
   return x;
   }
int main() {
   int lido;
   cin>>lido;
   while (lido !=0){
       cout<<centena(lido)<<endl;
       cin>>lido;
   }
   return 0;
}
```

7.21 (maior3)

Escreva uma função que receba 3 valores e devolva o maior deles. Se forem iguais, deve retornar qualquer um. Depois escreva um programa que receba 9 inteiros e imprima o maior deles usando a função acima definida.

```
int maior3(int a, int b, int c){
   if ((a>=b)&&(a>=c)) {return a;}
   if ((b>=a)&&(b>=c)) {return b;}
   return c;
}
int main(){
  int x1,x2,x3,y1,y2,y3,z1,z2,z3;
  int r1,r2,r3;
  cin>>x1>>x2>>x3;
  cin>>y1>>y2>>y3;
  cin>>z1>>z2>>z3;
  r1=maior3(x1,x2,x3);
  r2=maior3(y1,y2,y3);
  r3=maior3(z1,z2,z3);
  cout<<maior3(r1,r2,r3);
}
```

7.22 (prnprimos)

Escreva uma função ehprimo(n) que retorna 1 se n é primo ou 0 senão é. Depois escreva um programa C++ que leia A e B e imprima os primos entre A e B inclusive. Escrever 10 valores por linha. Suponha $A \leq B$.

```
using namespace std;
int primoburro(int n){
   int qtd,diva;
   qtd=0;
```

```
   diva=2;
   if (n<2) {return 0;} // necessario por causa do 1
   while(diva<n){
       if ((n%diva)==0){
          qtd=qtd+1;
       }
       diva=diva+1;
   }
   return (qtd==0);
}

int primoesperto(int n){
   int r, f;
   if (n==1){return 0;}
   if (n<4) {return 1;}
   if ((n%2)==0){return 0;}
   if (n<9){return 1;}
   if ((n%3)==0){return 0;}
   r =floor(sqrt(n));
   f=5;    // todo primo é 6k+-1
   while (f<=r){
      if ((n%f)==0 || (n%(f+2))==0){return 0;}
      f=f+6;
   }
   return 1;
}

int main(){
    int a,b,qtd;
    qtd=0;
    cin>>a>>b;
    if (a>b){swap(a,b);} // se nao der para supor a>=b (programacao defensiva)
    while (a<=b){
        if (primoesperto(a)){
            cout<<a<<" ";
            qtd=qtd+1;
        }
        if (qtd>9){
            cout<<endl;  //peca 100 000    200 000 e compare as 2 versoes
            qtd=0;       // veja o tempo de execucao ao final
        }                //esperto=9.7 segundos   burro=79.2 seg
        a=a+1;
    }
}
```

7.23 (invertedig)

Escreva uma função de nome $invertido(x, y)$ que recebe x e y e devolve 1 se eles forem invertidos (por exemplo $4123 \rightarrow 3214$) e 0 senão. Escreva depois um programa em C++ que receba diversos pares (condição de fim: um deles é zero) e informe se o primeiro é o invertido do segundo.

```
int tamanho(int n){
   int q=0;
   while (n!=0){
    n=n/10;
    q=q+1;
   }
   return q;
}
int tamanhoalt(int n){  // nao definido para n==0
   return 1+floor(log10(n));
}
int invertido(int x, int y){
```

```
    int d1,e1,tamx,tamy;
    if (tamanho(x)!=tamanho(y)) {return 0;}   //tamx=pow(10,floor(log10(x)));
    tamx=pow(10,(tamanho(x)-1));              //floor(log10(x)) = tamanho(x)-1
    while (x!=0){
       d1=x%10;
       e1=y/tamx;
       if (d1!=e1){return 0;}
       x=x/10;
       y=y-e1*tamx;
       tamx=tamx/10;
     }
     return 1;
}
int main(){
   int a,b;
   cin>>a>>b;
   while ((a!=0)&&(b!=0)) {
       if (invertido(a,b)){
          cout<<a<<" "<<b<<" sao invertidos"<<endl;
       }
          else {
          cout<<a<<" "<<b<<" NAO sao invertidos"<<endl;
       }
       cin>>a>>b;
   }
   return 0;
}
```

7.24 Relembrando passagens por valor e referência

Passagem de parâmetros por valor e por referência. Os formatos:

- por valor: tipo função (tipo pa1, tipo pa2, ...) ...
- por referência: tipo função (tipo & pa1, tipo & pa2, ...)...

Na chamada, nada muda, ambas são chamadas igualmente (por referência ou por valor). Obviamente a chamada por referência é necessária quando a função precisa devolver mais de 1 valor de resposta. Um tipo que vale a pena aprender é `void` (vazio, nulo) que significa que a função não retorna nada.

7.25 (atribquad)

Defina uma função que receba o lado de um quadrado, calcule e devolva a área, o perímetro e a diagonal desse quadrado.

```
void atribquad(float lado, float & area, float & per, float & diag){
   area = lado * lado;
   per = 4 * lado;
   diag = lado * sqrt(2.0);
}
int main(){
   float x,a,p,d;
   cin>>x;
   atribquad(x,a,p,d);
   cout<<"Area:      "<<a<<endl;
   cout<<"Perimetro: "<<p<<endl;
   cout<<"Diagonal:  "<<d<<endl;
}
```

7.26 (tempojogo)

Escreva uma função que recebe 4 informações de hora (h_1, m_1, h_2, m_2) sendo que $h_1, m_1 \leq h_2, m_2$ e retorne 1 se a duração registrada for maior que 240 minutos (4 horas) e 0 senão.

Questões: 1. operações com hora (e com data): Levar todos para uma origem comum, com o uso da menor unidade citada. Exemplo: o UNIX volta a 1/jan/1970. 2. Possível necessidade de swap (não neste caso, já que $h_1, m_1 \leq h_2, m_2$). 3. Quando testar este tipo de programa, faça uma ou mais contas na mão, **antes**: 10,5,14,3 → 238. Dá 240 se você chamar errado: h_1, m_2, h_2, m_2. 4. Nomes iguais ou não, tanto faz.

```
int tempojogo(int h1, int m1, int h2, int m2, int & dur){
   int d1,d2;
   d1=(h1*60)+m1;
   d2=(h2*60)+m2;
//   cout<<d1<<"  "<<d2<<endl;
   if (d2<d1) {swap(d1,d2);}
   dur=d2-d1;
   if (dur>210){return 1;}
   return 0;
}
int main(){
   int h1,m1,h2,m2,dur,res;
   cin>>h1>>m1>>h2>>m2;
   res=tempojogo(h1,m1,h2,m2,dur);
   cout<<"Duracao "<<dur<<"    com res="<<res<<endl;
}
```

7.27 (maxpot)

Escreva uma função que receba x, y, dur e devolve x^y e 1 se $x^y < dur$ e 0 senão. Questões: 1. A resposta 0 ou 1 volta no retorno da função (por convenção). 2. Pode-se usar POW já que ele pode aceitar inteiro. (Aceita também float). 3. A palavra potência pode ser usada dentro de eventual literal, mas conforme o sistema operacional pode dar zica, imprimindo potÛncia.

```
#include<cmath>
using namespace std;
int maxpot(int x, int y, int maxpot, int & res){
    res=pow(x,y);
    if (res<maxpot){return 1;}
    return 0;
}
int main(){
    int x,y,m,r,resposta;
    cin>>x>>y>>m;
    resposta=maxpot(x,y,m,r);
    cout<<"A potência e: "<<r<<" com resposta="<<resposta<<endl;
}
```

7.28 (raizes2g)

Escreva uma função que receba a, b e c e devolva x_1 e x_2, além da resposta 0=raizes complexas (e neste caso não volta nada em x_1 e x_2) ou 1=raizes reais e daí as duas raízes voltam nos seus lugares.
Questões: 1. Chamar como (int a,b,c) ou (int & a, int & b, int & c) tanto faz. 2. Mecanismo usual para evitar um erro de execução (que às vezes aparece como naN (=not a number). 3. Solução de compromisso: chamar 2 vezes a função `sqrt` ou criar uma variável a mais.

```
#include<cmath>
using namespace std;
int eq2grau(float & a, float & b, float & c, float & x1, float & x2){
  float delta,nv;
  delta=(b*b)-4*a*c;
  if (delta<0){return 0;}
  nv=sqrt(delta);
  x1=((-b)+nv)/(2*a);
  x2=((-b)-nv)/(2*a);
  return 1;
}
int main(){
```

```
  float a,b,c,x1,x2;
  int r;
  cin>>a>>b>>c;
  while ((a!=0)||(b!=0)||(c!=0)){
      r=eq2grau(a,b,c,x1,x2);
      if (r==0){cout<<"sem raizes reais"<<endl;}
      else{cout<<x1<<" "<<x2;}
      cin>>a>>b>>c;
  }
}
```

7.29 (ordena)

Definir uma função chamada `troca(a,b)` que troca os 2 valores de lugar e com ela escrever um programa C++ que leia a, b e c e imprima-os em ordem crescente.
Questões: 1. cansaço ? vá lavar louça. (teoria do banheiro limpo) 2. Não sabe por onde começar ? Desenhe casos de teste. Assim: 3 números $\rightarrow$ 6 casos, a saber:
maior,a,b - por exemplo 5, 1, 3
maior,b,a - 5,3,1
a,maior,b - 1,5,3
b,maior,a - 3,5,1
a,b,maior - 1,3,5
b,a,maior - 3,1,5
Qual é a lógica? se $a > b$, troca(a,b) e depois se $b > c$, troca(b,c). Agora sabidamente c é o maior. Depois disso, precisa testar de novo se $a > b$ troca(a,b) (de novo!). Neste ponto os números estão em ordem.

```
void troca(int &a, int & b){
  int tmp;
  tmp=a;
  a=b;
  b=tmp;
}
int main(){
  int a,b,c;
  cin>>a>>b>>c;
  if (a>b) {troca (a,b);}
  if (b>c) {troca (b,c);}
  if (a>b) {troca (a,b);}
  cout<<a<<" "<<b<<" "<<c<<endl;
}
```

7.30 (multiplos)

Escreva uma função que receba m, n e devolva a quantidade de múltiplos de m entre 1 e n, além do maior deles.
Questões: 1. O que fazer com a variável maior se ela for indefinida ? (por exemplo se não houver nenhum múltiplo ? É uma questão de combinar (o combinado nunca sai caro), por exemplo fazendo maior valer 0.
2. Nova aplicação do teorema de Morgan
3. Neste caso pode-se retornar o tipo `void`.

```
void multiplos(int m, int n, int & t, int & maior){
   if ((m<0) || (n<0)){t=0; maior=0; return;}
   while ((n%m)!=0){
     n=n-1;
   }
   maior=n;
   t=n/m;
   return;
}
int main(){
    int m,n,t,maior;
    cin>>m>>n;
    multiplos(m,n,t,maior);
```

```
    cout<<t<<" "<<maior<<endl;
}
```

7.31 Sobrecarregando funções

Funções do usuário podem ser sobrecarregadas. Ou seja pode haver 2 ou mais funções com nomes iguais, desde que tenham parâmetros diferentes, serão coisas diferentes. Denomina-se o conjunto de nome+parâmetros de assinatura da função, e a assinatura identifica realmente uma função. Acompanhe no exemplo

```
#include<iostream>
using namespace std;
int funca1(int n){
return n*2;
}
int funca1(int n, int n1){
return n+n1;
}
int main(){
   cout<<funca1(10); // retorna 20
   cout<<funca1(10,200); // retorna 210
}
```

Capítulo 8

Vetores

Estrutura homogênea de repetição de variáveis: mesmo nome e identificação pelo índice. A maneira de definir um vetor em C++ é acrescentando o tamanho do vetor ao lado do nome entre colchetes. exemplo: as últimas 12 inflações mês a mês.

```
float infla[12];
```

A especificação de tamanho não pode ser uma variável, o compilador não aceita. Então não vale

```
int x=10;
float var[x];   //vai dar erro
```

Entretanto, é perfeitamente possível fazer isto via define, acompanhe

```
# define tamanho 10
...
float var[tamanho];
```

Após esta definição sempre que a variável citada aparecer no código ela deverá estar acompanhada do colchete que individualiza o elemento dentro do vetor. Sua omissão causará um erro de compilação [1] Agora vejamos como isto funciona. Seja a definição da inflação acima

```
infla[0] -> a primeira
infla[11] -> a última
infla[x] -> genérica em função do valor de x
```

Pode ser inicializada usando chaves. Por exemplo

```
int main() {
  float a[5]={1.1,2.2,3.3,4.4,5.5};
  cout<<a[3];   // vai mostrar 4.4
}
```

Outra possibilidade muito comum é inicializar o vetor todo com zeros. Aí basta escrever um único zero

```
int main() {
  float a[5]={0};  // também vale igual float a[5]={};
  cout<<a[3];   // vai mostrar 0
}
```

Na realidade se a inicialização for menor que o tamanho da variável (numérica) ela será inicializada com valores nulos. Finalmente na inicialização, o tamanho pode ser omitido e o compilador buscará o tamanho real na inicialização. Não acho que seja uma boa idéia, pois exige uma contagem manual mais à frente...Exemplo

```
int x[]={1,2,3,4,5};
```

[1] Nota: isto é característica de C++. Em outros ambientes como Python ou APL essa omissão é perfeitamente aceitável e significa: aplique esta operação a todos os elementos do vetor...

8.1 Um exemplo envolvendo aleatoriedade

- Inicializar um vetor com valores aleatórios entre 0 e 1. A função aqui

```
#include<cstdlib>
#include<time.h>
using namespace std;
int main() {
  int i;
  float a[10];
  srand(time(NULL));
  i=0;
  while (i<10){
      a[i]=1.0*rand()/RAND_MAX;
      i=i+1;
  }
}
```

Notas:

1. O comando `i=i+1;` é muito comum em C++, e recebeu a "alcunha" de `i++`. Isto acabou batizando a extensão da linguagem `C` original.
2. Para gerar um número aleatório (pseudo-aleatório!) usa-se a função `rand()` que gera um aleatório inteiro entre 0 e `RAND_MAX`.
3. `RAND_MAX` é uma constante do ambiente e ela sinaliza o maior valor possível na randomização.
4. Para evitar gerar sempre a mesma sequência (afinal é pseudo-randômico), deve-se inicializar a semente com algum valor esse sim aleatório. É o que faz `srand(valor);`.
5. Uma boa idéia para automatizar este processo é chamar com `srand(time(NULL));`. Este comando inicializa a semente com o número de segundos reais passados entre as 00h00 do dia 1. de janeiro de 1970 UTC, e o exato instante da chamada.
6. Tanto `rand()` como `RAND_MAX` são inteiros. Se você dividí-los `as is`, a resposta será 0. Para atender ao pedido (aleatório entre 0 e 1), você precisa multiplicar essa divisão por 1.0 antes da atribuição, para converter tudo de inteiro para real.
7. `rand()` e `RAND_MAX` estão definidos no módulo `cstdlib`, daí a necessidade do `#include`.
8. `time()` está definido no módulo `time.h`, daí a necessidade do `#include`.
9. Se você quiser gerar um aleatório entre 1 e 60, para apostar na mega sena, faça `v1=rand() % 60+1;`. Se quiser gerar um ano entre 1985 e 2019, faça `v1=rand()%35+1985;`. A chamada `v1=rand()%50` vai gerar um aleatório entre `0..49`.

8.2 O comando for

Embora seja perfeitamente possível operar vetores usando `while` pode-se tirar proveito do fato de que os vetores em C++ sempre tem um tamanho fixo e conhecido de antemão. Então pode-se usar o comando for que simplifica todo o processamento no vetor. O comando tem o formato

```
for(inicialização; condição de fim; incremento) {
   ...
}
```

Veja um exemplo: Seja inicializar um vetor de 10 inteiros com o dobro do seu índice, ou seja 0,2,4,...,18. Eis como ficaria

```
int vet[10];
int i;
for (i=0;i<10;i++){
  vet[i]=i*2;
}
```

Este trecho acima poderia ser feito com while, e daria mais trabalho, mas funcionaria igual, veja

```
int vet[10];
int i;
i=0;
while(i<10){
   vet[i]=i*2;
   i++;
}
```

Ou seja, 3 linhas de código viraram uma única linha...

8.3 Exercícios com o uso de for

Interprete os comandos abaixo e descreva o que cada um faz. Considere os vetores P e L de mesmo tamanho.

```
1.
j=0;
for(i=n-1;i>=0;i--){
   P[j]=L[i];j++;
}

2.
j=0;
for(i=0;i<n-1;i++){
   P[j]=L[i];j++;
}

3.
j=0;
for(i=0;i<n;i=i+2){
   P[j]=L[i];j++;
}

4.
j=0;
for(i=1;i<n;i=i+2){
   P[j]=L[i];
   j++;
}

5.
j=0;
for(i=1;i<n;i++){
   P[j]=L[i];
   j++;
}

6.
j=0;
for(i=0;i<n;i=i+3){
   P[j]=L[i];
   j++;
}

7.
j=0;
for(i=0;i<n/2;i++){
   P[j]=L[i];
   j++;
}

8.
j=0;
for(i=n/2-k/2;i<n/2+k/2;i++]{
   P[j]=L[i];j++
}

9.
#include<algorithm>
sort(&L[0],&L[n]);

10.
j=0;
for(i=0;i<n;i++){
   if(L[i]==k){
      i++;
   }
   P[j]=L[i];j++;
}

11.
j=0;
for(i=0;i<s;i++){
   P[j]=L[i];
   j++;
P[j]=k;
j++;
for(i=s;i<n;i++){
   P[j]=L[i];
   j++;
}

12.
for(i=0;i<n;i++){
   if(k==L[i]){
      return 1;
   }
}
return 0;

13.
c=0;
for(i=0;i<n;i++){
   if(k==L[i]){
      c++;
   }
}
cout<<c;

14.
j=0;
for(i=0;i<n;i++){
   P[j]=L[i];
   if(L[i]==t){
      P[j+1]=t;
      j++;
   }
   j++;
}
```

Respostas

1 Reverter a lista

2 Excluir o último

3 Obter apenas os elementos de ordem par

4 Apenas os de ordem ímpar

5 Excluir o primeiro

6 Obter o 1^o, 4^o, 7^o, ...

7 Obter só a primeira metade

8 Obter os k elementos centrais

9 Ordenar a lista

10 Excluir o elemento de valor k

11 Inserir o valor k antes do elemento de ordem s

12 Existe o elemento k em L ?

13 Quantos ocorrências do valor k

14 Insira o valor k depois da ocorrência do valor t

8.4 Erros comuns

A cada nova ferramenta, abre-se mais uma caixa de erros possíveis. Ao lidar com vetores surgem

esquecer de incrementar Se você cometer este erro, seu programa vai entrar em *loop*, acompanhe

```
int vet[10];
int i;
i=0;
while(i<10){
   vet[i]=i*2;
}
```

acessar além Este erro é mais perigoso, porque o programa vai exibir comportamento *errático*, não necessariamente errado.

```
int vet[10];
int i;
i=0;
while(i<11){
   vet[i]=i*2;
   i++;
}
```

A coisa é sutil, mas o vetor `vet` só vai até a posição [9]. Ao acessar a posição [10] pega-se o próximo endereço de memória e sabe-se lá o que tem nessa posição. Em outros ambientes (Python ou APL aparece um erro: "list index out of range" no primeiro e "INDEX ERROR" no segundo, mas no C++ não há erro: é uma opção deliberada de design da linguagem.

não entrar no laço É uma distração fatal, veja

```
int vet[10];
int i;
i=10;
while(i<10){
   vet[i]=i*2;
   i++;
}
```

Aqui ele não entra no laço, mas nenhum erro é sinalizado. Suponho que isto **é** um erro.

8.5 Os comandos break e continue

Às vezes, dentro de um laço (seja via for ou via while) surge a necessidade repentina de sair do laço. Esta facilidade é oferecida pelo comando `break;`. Quando ele é processado há uma saída incondicional do bloco ({...}) em que ele está. Todas as demais condições permanecem inalteradas, mas o processamento cai fora.

8.6 (prodescalar)

Escreva um programa C++ que leia o conjunto `A[N]` (onde `A` é flutuante e `N` está definido como constante no início do programa e `B[N]` idem, calcule e imprima o produto escalar `A.B`.

```
#include<iostream>
#define N 5
using namespace std;
int main(){
   float a[N];
   float b[N];
   float r=0;
   int i;
   cout<<"Informe valores de A: "<<"( "<<N<<" valores)"<<endl;
   for(i=0;i<N;i++){
      cin>>a[i];
   }
   cout<<"Informe valores de B: "<<"( "<<N<<" valores)"<<endl;

   for(i=0;i<N;i++){
      cin>>b[i];
   }
```

```
   for(i=0;i<N;i++){
      r=r+a[i]*b[i];
   }
   cout<<"O valor do produto escalar a.b = "<<r<<" "<<endl;
}
```

Já o comando continue é parecido, mas ligeiramente diferente. Ele implica na saida daquela interação (não do loop) caminhando-se para a próxima interação. Acompanhe no exemplo

```
for (i=0;i<10;i++){
   if ((i>5)&&(i<8)){continue;}
   cout<<i<<" ";
}
```

Os valores impressos acima serão 0 1 2 3 4 5 8 9.

8.7 (pertence)

Escreva um programa C++ que leia `V[N]`, até ler um negativo (inteiros). Contar e informar quantos números foram carregados. Daí ler entradas inteiras e informar se elas pertencem ao vetor. Encerrar ao ler um número negativo.

```
#define N 7
int main(){
   int a, i=0,j;
   int v[N];
   while(i<N){
      cout<<"Informe elemento "<<i+1<<" : ";
      cin>>a;
      cout<<endl;
      if (a<0) {break;}
      v[i]=a;
      i++;
   }
   cout<<"Foram carregados "<<i<<" elementos"<<endl;
   a=0;
   cin>>a;
   while (a>=0){
      for(j=0;j<i;j++){
        if (a==v[j]){
            cout<<a<<" pertence a V"<<endl;
            j=99999;
        }
      }
      cin>>a;
   }
}
```

8.8 (interseccao)

Escreva um programa C++ que leia `A[N]` e `B[N]`. Calcular e imprimir $A \cap B$.

```
#define N 7
int main(){
   int a[N], b[N], r[N];
   int i,j,k;
   cout<<"Informe "<<N<<" valores para A: "<<endl;
   for (i=0;i<N;i++){
      cin>>a[i];
   }
   cout<<"Informe "<<N<<" valores para B: "<<endl;
   for (i=0;i<N;i++){
      cin>>b[i];
   }
   k=0;
```

```
   for(i=0;i<N;i++){
      for(j=0;j<N;j++){
        if (a[i]==b[j]){
             r[k]=a[i];
             k++;
        }
      }
   }
   cout<<"O conjunto interseccao com tamanho "<<k<<" eh: "<<endl;
   for(i=0;i<k;i++){
     cout<<r[i]<<" ";
   }
}
```

8.9 (ocorrencia)

Escreva um programa C++ que leia `A[N]` e imprima para cada número entre 0..50 quantas vezes cada um destes números ocorre em `A`.

```
#include<iostream>
#define N 10
using namespace std;
int main(){
  int v[N];
  int i,q,j;
  cout<<"Informe "<<N<<" valores "<<endl;
  for (i=0;i<N;i++){
     cin>>v[i];
  }
  for (i=0;i<51;i++){
    q=0;
    for (j=0;j<N;j++){
        if (i==v[j]){q++;}
    }
    cout<<i<<" ocorre "<<q<<" vezes"<<endl;
  }
}
```

8.10 consumo de água 2021

Escreva um programa C++ que leia 12 consumos (flutuantes). Calcular e imprimir quanto foi pago no ano, de acordo com a regra

até 5 m^3 7 unidades de preço por m^3

de 6 a 10 m^3 7.35 unidades de preço por m^3

de 11 a 15 m^3 8.00 unidades de preço por m^3

de 16 a 20 m^3 8.36 unidades de preço por m^3

de 21 a 30 m^3 8.42 unidades de preço por m^3

acima de 30 m^3 10.56 unidades de preço por m^3

Em junho houve um reajuste de 15%. Informar o mês de maior consumo e o valor desse mês. Nesta comparação se houver mais de um mês com o mesmo consumo, informar a primeira ocorrência. Informar o consumo médio do ano. Informar o total arrecadado a mais após o reajuste. Dica: para mostrar dinheiro com 2 casas, incluir `#include<iomanip>` e antes do cout, fazer `cout<<fixed<<setprecision(2);`.

8.11 Vetores e funções

Quando o parâmetro de uma função é um vetor (ou matriz), ele sempre é passado via referência, mesmo sem a presença do & (que, a propósito, se colocado vai dar erro de compilação). Veja

```
int funca(int v[]){
   v[0]=1;
   v[1]=10;
   v[2]=100;
   return 0;
}
int main(){
  int a[3];
  funca(a);
  cout<<a[0]<<' '<<a[1]<<' '<<a[2]; // 1,10,100
}
```

Outra questão é que na definição da função, o tamanho do vetor pode ser adiado até a chamada, como foi feito ai encima. Note que o tamanho é estabelecido na definição da variável (já que na chamada não vai tamanho algum). veja-se que o retorno de funções pode vir na lista de parâmetros (com &) mas é mais comum ele vir no return.

8.12 (intercomfuncao)

No exercício a seguir, veja que a mesma função (`carga`) é usada para 2 vetores (a, b) distintos, veja a seguir. Outra questão é que a e b tem tamanho N, mas o resultado tem tamanho incerto, razão pela qual tem que voltar com o tamanho real dele.

```
#define N 7
void carga(int v[]){
   int i;
   cout<<"com "<<N<<" valores "<<endl;
   for (i=0;i<N;i++){
      cin>>v[i];
   }
}
int inter(int a[], int b[], int r[]){
   int i,j;
   int tamr=0;
   for (i=0;i<N;i++){
      for (j=0;j<N;j++){
         if (a[i]==b[j]){
             r[tamr]=a[i];
             tamr++;
         }
      }
   }
   return tamr;
}
int main(){
   int veta[N], vetb[N], vetr[N];
   int ta,i;
   cout<<"Informe a ";
   carga(veta);
   cout<<"Informe b ";
   carga(vetb);
   ta=inter(veta, vetb, vetr);
   cout<<"Vetor interseccao "<<endl;
   for (i=0;i<ta;i++){
      cout<<vetr[i]<<" ";
   }
}
```

Uma boa estratégia de estudo é comparar os exemplos feitos na semana 9 (usando funções) com os da semana 8 (programação espagueti).

Outros exemplos que podem ser comparados com os similares feitos em p. espagueti

8.13 (prodescalar2)

```
//---------------------------- produto escalar ---------------
```

```
#include<iostream>
using namespace std;
#define N 5
float prod_esc(float a[], float b[], int tam) {
   int i;
   float res=0;
   for (i=0;i<tam;i++){
     res=res+a[i]*b[i];
   }
   return res;
}
int main(){
   float a[N], b[N];
   int i;
   cout<<"Informe os "<<N<<" valores de a"<<endl;
   for (i=0;i<N;i++){
    cin>>a[i];
   }
   cout<<"Informe os "<<N<<" valores de b"<<endl;
   for (i=0;i<N;i++){
    cin>>b[i];
   }
   cout<<"Produto interno a.b "<<prod_esc(a,b,N)<<endl;
}
```

8.14 (pertence2)

```
//----------------- pertence ? --------------------
#include<iostream>
using namespace std;
#define N 6
int carga(int vet[]){ //o tamanho real volta no return
   int a,i;
   cin>>a;
   for (i=0;i<N;i++){
      if (a<0){return i;}
      vet[i]=a;
      cin>>a;
   }
   return N; // ou i, tanto faz
}
int pertence(int vet[],int treal,int quem){
   int i;
   for(i=0;i<treal;i++){
      if (vet[i]==quem){return 1;}
   }
   return 0;
}
int main(){
   int a,achou,quantos;
   int v[N];
   quantos=carga(v);
   cout<<"Foram carregados "<<quantos<<" elementos"<<endl;
   cin>>a;
   while (a>=0){
      achou=pertence(v,quantos,a);
      if (achou==1){cout<<"Existe"<<endl;}
      cin>>a;
   }
}
```

8.15 (buscaValor)

Escreva um programa C++ que leia uma matriz de 10×10 e um valor x. O programa deverá fazer uma busca na matriz por este valor e informar a linha e a coluna quando x for encontrado ou a mensagem **valor não encontrado** em caso contrário

```
//------------   busca valor -----------------
#include<iostream>
using namespace std;
#define T 5
void leitura(int mat[T][T]){
  int i,j;
  cout<<"Forneca "<<T<<" x "<<T<<" valores"<<endl;
  for (i=0;i<T;i++){
    for(j=0;j<T;j++){
        cin>>mat[i][j];
    }
  }
  return;
}
void busca(int mat[T][T], int x, int & lin, int & col){
  int i,j;
  lin=-1;
  for (i=0;i<T;i++){
    for(j=0;j<T;j++){
        if(mat[i][j]==x){
           lin=i;
           col=j;
        };
    }
  }
  return;
}

int main(){
  int val,vi,vj;
  int mat[T][T];
  leitura(mat);
  cout<<"Informe o valor a pesquisar "<<endl;
  cin>>val;
  busca(mat,val,vi,vj);
  if (vi==-1){cout<<"valor nao encontrado"<<endl;}
  else {cout<<"Linha: "<<vi<<"  coluna: "<<vj<<endl;}
}
```

8.16 (prndiagonais)

Escreva um programa C++ que leia uma matriz 10×10 e imprima em linhas distintas os elementos da diagonal principal e o da diagonal secundária da matriz lida.

```
//------------ prndiagonais ----------------
#include<iostream>
using namespace std;
#define T 5
void leitura(int mat[T][T]){
  int i,j;
  cout<<"Forneca "<<T<<" x "<<T<<" valores"<<endl;
  for (i=0;i<T;i++){
    for(j=0;j<T;j++){
        cin>>mat[i][j];
    }
  }
  return;
```

```
}
int main(){
  int i;
//  int mat[T][T];
//  leitura(mat);
  int mat[T][T]={{1,2,3,4,5},{6,7,8,9,10},{11,12,13,14,15},
  {16,17,18,19,20},{21,22,23,24,25}};
  cout<<"Diagonal principal"<<endl;
  for (i=0;i<T;i++){
    cout<<mat[i][i]<<" ";
  }
  cout<<endl;
  cout<<"Diagonal secundaria"<<endl;
  for (i=0;i<T;i++){
    cout<<mat[i][(T-1)-i]<<" ";
  }
  cout<<endl;
}
```

8.17 (trocaelemens)

Escreva um programa C++ que obtenha do usuário uma matriz quadrada (dimensão máxima de 10×10) e troque o maior elemento de cada linha com o elemento da diagonal principal.

```
//---------  trocaelems -----------------
#include<iostream>
using namespace std;
#define T 5
void leitura(int mat[T][T]){
  int i,j;
  cout<<"Forneca "<<T<<" x "<<T<<" valores"<<endl;
  for (i=0;i<T;i++){
    for(j=0;j<T;j++){
        cin>>mat[i][j];
    }
  }
  return;
}
int main(){
  int i,j,mai,salvaj,aux;
//  int mat[T][T];
//  leitura(mat);
  int mat[T][T]={{1,2,3,4,5},{6,7,8,9,10},{11,12,13,14,15},
  {16,17,18,19,20},{21,22,23,24,25}};
  for (i=0;i<T;i++){
     mai=-9999999;
     for (j=0;j<T;j++){
        if (mat[i][j]>mai){
            mai=mat[i][j];
            salvaj=j;
        }
     }
     aux=mat[i][i];
     mat[i][i]=mai;
     mat[i][salvaj]=aux;
  }
  for(i=0;i<T;i++){
    for(j=0;j<T;j++){
        cout<<mat[i][j]<<" ";
    }
    cout<<endl;
  }
}
```

8.18 (somamatrizes)

Escreva um programa C++ que some duas matrizes A e B com dimensão máxima 10×10. A função deve receber como argumentos as duas matrizes, as suas dimensões reais (dentro do intervalo 2..10) e o resultado deve ser colocado em outra matriz também passada como argumento.

```
//------------ somamatrizes -------------
#include<iostream>
#include<iomanip>
#define M 3
#define N 4
using namespace std;

void leitura(int mat[M][N]){
  int i,j;
  cout<<"Forneca "<<M<<" x "<<N<<" valores"<<endl;
  for (i=0;i<M;i++){
    for(j=0;j<N;j++){
        cin>>mat[i][j];
    }
  }
  return;
}
void somamat(int a[M][N], int b[M][N], int tm, int tn, int r[M][N]){
   int i,j;
   for (i=0;i<tm;i++){
     for (j=0;j<tn;j++){
        r[i][j]=a[i][j]+b[i][j];
     }
   }
}

int main(){
   int ma[M][N]={{1,2,3,4},{5,6,7,8},{9,10,11,12}};
   int mb[M][N]={{1,1,1,1},{2,2,2,2},{3,3,3,3}};
   int mr[M][N];
   int i,j;
   // leitura(ma);
   //leitura(mb);
   somamat(ma,mb,2,3,mr);   // M e N ??
   for(i=0;i<2;i++){    //igual acima
    for(j=0;j<3;j++){   // idem
      cout<<mr[i][j]<<" ";
     }
    cout<<endl;
    }
}
```

8.19 (somavizinhos)

Escreva um programa C++ que leia uma matriz A de inteiros de tamanho $M \times N$ (M e N estabelecidos via `#define` e gerar uma nova matriz B onde cada elemento B_{ij} é a soma dos vizinhos acima e abaixo do elemento A_{ij}. Cuidado com os limites da matriz: na primeira e última linha da matriz haverá apenas um vizinho para cada elemento.

```
//---------------------somavizinhos ------------------
#include<iostream>
using namespace std;
#define M 4
#define N 5
void leitura(int mat[M][N]){
  int i,j;
  cout<<"Forneca "<<M<<" x "<<N<<" valores"<<endl;
  for (i=0;i<M;i++){
```

```
    for(j=0;j<N;j++){
        cin>>mat[i][j];
    }
  }
  return;
}
int main(){
  int i,j,soma;
//  int mata[M][N];
//  leitura(mat);
  int mata[M][N]={{1,2,3,4,5},{6,7,8,9,10},{11,12,13,14,15},
  {16,17,18,19,20}};
  int matb[M][N];
  for (i=0;i<M;i++){
    for(j=0;j<N;j++){
        soma=0;
        if (i>0){soma=soma+mata[i-1][j];}
        if (i<M-1){soma=soma+mata[i+1][j];}
        matb[i][j]=soma;
    }
  }
  for(i=0;i<M;i++){
    for(j=0;j<N;j++){
        cout<<matb[i][j]<<" ";
    }
    cout<<endl;
  }
}
```

8.20 (mediavizinhos)

Escreva um programa C++ que leia uma matriz A de inteiros $M \times N$ (M e N via define) e aplica uma função para calcular uma matriz B onde cada B_{ij} é a média aritméticas dos 4 vizinhos. Cuidado com os elementos extremos.

```
// ----------------- media vizinhos ----------------
#include<iostream>
#include<iomanip>
using namespace std;
#define M 4
#define N 5
void leitura(int mat[M][N]){
  int i,j;
  cout<<"Forneca "<<M<<" x "<<N<<" valores"<<endl;
  for (i=0;i<M;i++){
    for(j=0;j<N;j++){
        cin>>mat[i][j];
    }
  }
  return;
}
int main(){
  int i,j,qtos;
  float soma;
//  int mata[M][N];
//  leitura(mat);
  int mata[M][N]={{1,2,3,4,5},{6,7,8,9,10},{11,12,13,14,15},
  {16,17,18,19,20}};
  float matb[M][N];
  for (i=0;i<M;i++){
    for(j=0;j<N;j++){
        soma=0.0;
        qtos=0;
        if (i>0){soma=soma+mata[i-1][j]; qtos++;}
```

```
        if (i<M-1){soma=soma+mata[i+1][j];qtos++;}
        if (j>0){soma=soma+mata[i][j-1];qtos++;}
        if (j<N-1){soma=soma+mata[i][j+1];qtos++;}
        matb[i][j]=soma/qtos;
    }
  }
  for(i=0;i<M;i++){
    for(j=0;j<N;j++){
        cout<<setw(10) << fixed <<matb[i][j]<<" ";
    }
    cout<<endl;
  }
}
```

8.21 (maxcols)

Faça um programa C++ que leia do usuário uma matriz $N \times M$ e preecha um vetor de M elementos tal que a posição i do vetor contenha o maior valor da coluna i da matriz. Ao final, imprimir o vetor

```
//---------maxcols-----------------
#include<iostream>
#include<iomanip>
using namespace std;
#define N 3
#define M 5
void leitura(int mat[N][M]){
  int i,j;
  cout<<"Informe "<<N<<" x "<<M<<" elementos "<<endl;
  for (i=0;i<N;i++){
    for (j=0;j<M;j++){
        cin>>mat[i][j];
    }
  }
}
void imprime(int mat[N][M]){
  int i,j;
  cout<<"Matriz original "<<endl;
  for (i=0;i<N;i++){
    for (j=0;j<M;j++){
        cout<<setw(5)<<mat[i][j];
    }
    cout<<endl;
  }
}
int main(){
  int mat[N][M];
  int vet[M];
  int i,j,k,maior;
  leitura(mat);
  for (i=0;i<M;i++){
    maior=-9999;
    for (j=0;j<N;j++){
        if (mat[j][i]>maior){maior=mat[j][i];}
    }
    vet[i]=maior;
  }
  imprime(mat);
  cout<<"---------------------"<<endl;
  for (i=0;i<M;i++){
    cout<<setw(5)<<vet[i];
  }
}
```

8.22 (flipvert)

Escreva um programa C++ que leia uma matriz $N \times M$ (N e M via define) e altere a matriz lida, invertendo a ordem dos elementos de cada coluna, imprimindo a matriz resultante na tela. A inversão deve ser feita na própria matriz lida, sem matrizes auxiliares.

```
//----------------flipvert----------------
#include<iostream>
#include<iomanip>
using namespace std;
#define N 3
#define M 5
void leitura(int mat[N][M]){
  int i,j;
  cout<<"Informe "<<N<<" x "<<M<<" elementos "<<endl;
  for (i=0;i<N;i++){
    for (j=0;j<M;j++){
        cin>>mat[i][j];
    }
  }
}
void imprime(int mat[N][M]){
  int i,j;
  cout<<"Matriz original "<<endl;
  for (i=0;i<N;i++){
    for (j=0;j<M;j++){
        cout<<setw(5)<<mat[i][j];
    }
    cout<<endl;
  }
}
int main(){
  int mat[N][M];
  int i,j,tempo;
  leitura(mat);
  imprime(mat);   //N=linhas, M=colunas
  cout<<"-----------------"<<endl;
  for (i=0;i<M;i++){
    for(j=0;j<N/2;j++){
        tempo=mat[j][i];
        mat[j][i]=mat[(N-1)-j][i];
        mat[(N-1)-j][i]=tempo;
    }
  }
  imprime(mat);
}
```

8.23 (segmento)

Crie uma função chamada `ehsegmento` que recebe como parâmetros os seguintes itens

- um vetor de inteiros a
- um vetor de inteiros b
- um valor inteiro n que representa o tamanho do vetor a
- um valor inteiro m que representa o tamanho de b
- um inteiro p que representa uma posição do vetor a

A função deve devolver 1 se o vetor b for um segmento do vetor a iniciado na posição p de a ou 0 em caso contrário. Por exemplo, considere $p = 2$, o vetor a (com $n = 5$)

```
6 5 4 3 8 9
```

Se o vetor b for (com $m = 3$) for

```
4 3 8
```

A resposta deverá ser 1. Por outro lado, se o vetor b (com $m = 3$) for

```
4 8 3
```

o retorno da função deve ser 0.

```
//----------------segmento--------------------
#include<iostream>
using namespace std;
int ehsegmento(int a[], int b[], int ta, int tb, int p){
   int resp,i;
   resp=1;
   i=0;
   while((p<ta-1)&&(i<tb-1)){
      if (a[p]!=b[i]){resp=0;}
      p++;
      i++;
   }
   if ((p>=ta-1)&&(i<tb-1)){resp=0;}
   return resp;
}
int main(){
  int va[100]={6,5,4,3,8,9};
  int vb[100]={8,9,33};
  cout<<ehsegmento(va,vb,6,3,4);
}
```

8.24 (matpermuta)

Dizemos que uma matriz inteira A $(n \times n)$ é uma matriz de permutação se em cada linha e em cada coluna houver $n-1$ elementos nulos e um único elemento igual a 1. Dada uma matriz inteira A verificar se ele é uma matriz de permutação. Exemplos

```
0 1 0 0
0 0 1 0
1 0 0 0
0 0 0 1
```

é de permutação enquanto que

```
0 1 0 0
0 0 1 0
1 0 0 0
0 0 0 2
```

não é.

```
//------------------matpermuta----------------
#include<iostream>
using namespace std;
#define T 4
void leitura(int mat[T][T]){
  int i,j;
  cout<<"Informe "<<T<<" x "<<T<<" elementos"<<endl;
  for (i=0;i<T;i++){
    for(j=0;j<T;j++){
        cin>>mat[i][j];
    }
  }
}
int analisa(int m[T][T]){
  int resp,somal,somac,i,j;
```

```
  resp=1;
  for (i=0;i<T;i++){
    somal=0;
    for (j=0;j<T;j++){
        somal=somal+m[i][j];
    }
    if (somal!=1){resp=0;}
  }
  for (j=0;j<T;j++){
    somac=0;
    for (i=0;i<T;i++){
        somac=somac+m[i][j];
    }
    if(somac!=1){resp=0;}
  }
  for(i=0;i<T;i++){
    for(j=0;j<T;j++){
        if ((m[i][j]!=0)&&(m[i][j]!=1)){resp=0;}
    }
  }
  return resp;
}
int main(){
    int m[T][T];
    leitura(m);
    cout<<analisa(m);
}
```

8.25 (lincolnulas)

Dada uma matriz A $(n \times m)$ imprimir o número de linhas e o número de colunas nulas da matriz. Exemplo

```
0 0 0 0
1 0 2 2
4 0 5 6
0 0 0 0
```

tem duas linhas nulas e uma coluna nulas.

```
//-----------------lincolnulas--------------------
#include<iostream>
using namespace std;
#define N 4
#define M 5
void leitura(int mat[N][M]){
  int i,j;
  cout<<"Informe "<<N<<" x "<<M<<" elementos"<<endl;
  for (i=0;i<N;i++){
    for(j=0;j<M;j++){
        cin>>mat[i][j];
    }
  }
}
int somcol(int m[N][M]){
  int i,j;
  int resp,qtos=0;
  for (j=0;j<M;j++){
    resp=1;
    for(i=0;i<N;i++){
        if (m[i][j]!=0){resp=0;}
    }
    if (resp==1){qtos++;}
  }
```

```
  return qtos;
}
int somlin(int m[N][M]){
  int i,j;
  int resp,qtos=0;
  for (i=0;i<N;i++){
    resp=1;
    for(j=0;j<M;j++){
        if (m[i][j]!=0){resp=0;}
    }
    if (resp==1){qtos++;}
  }
  return qtos;
}
int main(){
  int m[N][M];
  leitura(m);
  cout<<"Linhas nulas  "<<somlin(m)<<endl;
  cout<<"Colunas nulas "<<somcol(m)<<endl;
}
```

8.26 (custotransp)

Os elementos $M[i,j]$ de uma matriz M $(n \times n)$ representam os custos de transporte da cidade i para a cidade j. Dados n itinerários, cada um deles com k cidades, calcular o custo total de cada itinerário. Exemplo:

```
 4   1   2   3
 5   2   1 400
 2   1   3   8
 7   1   2   5
```

O custo do itinerário `1 4 2 4 4 3 2 1` é $M[1,4]+M[4,2]+M[4,4]+M[4,3]+M[3,2]+M[2,1] = 3+1+400+5+2+1+5 = 417$

```
//-----------------custotransp--------------------------
#include<iostream>
using namespace std;
#define T 4
void leitura(int m[T][T]){
  int i,j;
  cout<<"Informe "<<T<<" x "<<T<<" elementos"<<endl;
  for (i=0;i<T;i++){
    for(j=0;j<T;j++){
        cin>>m[i][j];
    }
  }
}
int main(){
  int i,j,loca,cus;
  int m[T][T]={{4,1,2,3},{5,2,1,400},{2,1,3,8},{7,1,2,5}};
  int iti[100]={1,4,2,4,4,3,2,1};
  leitura(m);
  cout<<"Informe o itinerario - encerre por -1"<<endl;
  i=0;
  cin>>loca;
  while (loca!=-1){
    iti[i]=loca;
    i++;
    cin>>loca;
  }
  i=8;
  cus=0;
  j=0;
```

```
  while(j<i-1){
    cus=cus+m[(-1)+iti[j]][(-1)+iti[j+1]];
    cout<<"Trecho "<<(-1)+iti[j]<<" ate "<<(-1)+iti[j+1]<<" e de "
       <<m[(-1)+iti[j]][(-1)+iti[j+1]]<<endl;
    j++;
  }
  cout<<"O custo total eh "<<cus<<endl;
  return 0;
}
```

8.27 (itinerario)

Considere N cidades numeradas de 1 a n que estão interligadas por uma série de estradas de mão única. As ligações entre as cidades são representadas pelos elementos de uma matriz quadrada L $(n \times n)$ cujos elementos L_{ij} assumem o valor 0 ou 1 conforme exista ou não exista estrada direta que saia da cidade i e chegue na cidade j. Assim es elementos da i-ésima linha indicam as estradas que saem da cidade i e os elementos da j-ésima coluna indicam as estradas que chegam na cidade j. Por convenção $L_{ii} = 1$. A figura abaixo ilustra um exemplo para $n = 4$.

```
1 1 1 0
0 1 1 0
1 0 1 1
0 0 1 1
```

a) Dado k determinar quantas estradas saem e quantas estradas chegam à cidade k

b) A qual das cidades chega o maior número de estradas ?

c) Dado k verificar se todas as ligações diretas entre a cidade k e as demais são de mão dupla.

d) Relacionar as cidades que possuem saídas diretas para a cidade k

e) Relacionar, se existirem:

- As cidades isoladas, isto é que não têm ligação com nenhuma outra
- As cidades das quais não há saída, apesar de haver entrada
- As cidades das quais há saídam sem haver entrada

f) Dada uma sequência de m inteiros cujos valores representam cidades e estão entre 1 e n verificar se é possível realizar o roteiro correspondente. No exemplo dado o roteiro representado pela sequência `3 4 3 2 1` com $m = 5$ é impossível.

```
//-------------itinerario-----------------------
#include<iostream>
using namespace std;
#define T 4
void leitura(int m[T][T]){
  int i,j;
  cout<<"Informe "<<T<<" x "<<T<<" elementos"<<endl;
  for (i=0;i<T;i++){
    for(j=0;j<T;j++){
        cin>>m[i][j];
    }
  }
}
int main(){
  int m[T][T]={{1,1,1,0},{0,1,1,0},{1,0,1,1},{0,0,1,1}};
  //leitura(m);
  int i,j,k,saem,chegam,maxi,salvac,entrada,saida,cida;
  int chega,iso,possi;
// dado k, chegam e saem
  cout<<"dado k, saem e chegam em k"<<endl;
  cin>>k;
  saem=0;
  for (i=0;i<T;i++){
    saem=saem+m[k][i];
```

```
  }
  cout<<"Saem "<<saem<<endl;
  chegam=0;
  for (i=0;i<T;i++){
    chegam=chegam+m[i][k];
  }
  cout<<"Chegam "<<chegam<<endl;
// campea de chegadas
  maxi=-9999;
  for (i=0;i<T;i++){
    chega=0;
    for (j=0;j<T;j++){
        chega=chega+m[j][i];
    }
    if (chega>maxi){
        maxi=chega;
        salvac=i;
    }
  }
  cout<<"A campea de chegadas eh "<<salvac<<endl;
// dado k, verificar se all <-> k sao de mao dupla
  for(i=0;i<T;i++){
    cout<<"De "<<i<<" com "<<k<<" ";
    if ((m[i][k]==1) && (m[k][i]==1)){cout<<" sim"<<endl;}
    else {cout<<" nao"<<endl;}
  }
// saidas diretas para k
  cout<<"Saidas diretas para "<<k<<endl;
  for(i=0;i<T;i++){
    if (m[i][k]==1){cout<<" de "<<i<<endl;}
  }
// cidades isoladas
  cout<<"Cidades isoladas ";
  for (i=0;i<T;i++){
    iso=0;
    for (j=0;j<T;j++){
        if (i!=j){iso=iso+m[i][j]+m[j][i];}
    }
    if (iso==0){cout<<i<<" ";}
  }
  cout<<endl;
//cidades sem saida apesar de haver entrada
  cout<<"Cidades com entrada sem saida ";
  for (k=0;k<T;k++){
    entrada=0;
    saida=0;
    for (i=0;i<T;i++){
        entrada=entrada+m[i][k];
        saida=saida+m[k][i];
    }
    if ((entrada>0)&&(saida==0)){cout<<k<<" ";}
  cout<<endl;
  }
//cidades sem entrada apesar de haver saida
  cout<<"Cidades com saida sem entrada ";
  for (k=0;k<T;k++){
    entrada=0;
    saida=0;
    for (i=0;i<T;i++){
        entrada=entrada+m[k][i];
        saida=saida+m[i][k];
    }
    if ((entrada==0)&&(saida>1)){cout<<k<<" ";}
```

```
  cout<<endl;
  }
// sequencia de visitacao
  cout<<"Informe sequencia de cidades, termine por -1";
  int seq[100];
  cin>>cida;
  i=0;
  while(cida!=-1){
    seq[i]=cida-1;
    i++;
    cin>>cida;
  }
  possi=1;
  j=0;
  while(j<i-1){
    possi=possi && m[seq[j]][seq[j+1]];
    j++;
  }
  if (possi==1){cout<<"trajeto possivel"<<endl;}
  else{cout<<"trajeto impossivel"<<endl;}
}
```

8.28 (matidentidade)

Escreva uma função que devolva uma matriz identidade de ordem 11

```
#include<iostream>
#include<iomanip>
using namespace std;
void mi11(int A[11][11]){
   int i,j;
   for (i=0;i<11;i++){
      for (j=0;j<11;j++){
          if (i==j){
             A[i][j]=1;
             }
             else {
                A[i][j]=0;
             }
      }
   }
}
int main(){
   int x[11][11];
   int i,j;
   mi11(x);
   for (i=0;i<11;i++){
      for (j=0;j<11;j++){
         cout<<setw(2)<<x[i][j];
      }
   cout<<endl;
   }
}
```

8.29 (matsimetrica)

Escreva uma função que receba uma matriz de ordem 9 e devolva 1 se a matriz é simétrica e 0 senão.

```
#include<iostream>
using namespace std;
int msim(int a[9][9]){
   int e=1;
```

```
   int i,j;
   for (i=0;i<9;i++){
      for (j=0;j<9;j++){
          if (a[i][j]!=a[j][i]) {
             e=0;
          }
      }
   }
   return (e);
}
int main() {
   int x[9][9];
   int mi,mj;
   for (mi=0;mi<9;mi++){
      for (mj=0;mj<9;mj++){
         x[mi][mj]=0;
      }
   }
   x[3][5]=99;
   cout<<msim(x)<<endl;
   x[3][5]=0;
   cout<<msim(x)<<endl;
}
```

8.30 (matdetermina)

Suponha o sistema:
$5x + 8y + 3z = 55$
$2x + 9y + z = 32$
$3x + 2y + 5z = 53$
Escreva um programa C++ que ache os valores de x, y e z usando determinantes.

```
#include<iostream>
using namespace std;

int det(int x[3][3]){
   int i,j;
   i=(x[0][0]*x[1][1]*x[2][2])+(x[0][1]*x[1][2]*x[2][0])+(x[1][0]*x[2][1]*x[0][2]);
   j=(x[2][0]*x[1][1]*x[0][2])+(x[0][0]*x[1][2]*x[2][1])+(x[1][0]*x[0][1]*x[2][2]);
   return (i-j);
}

int main(){
   int A[3][3]={{5,8,3},{2,9,1},{3,2,5}};
   int Dx[3][3]={{55,8,3},{32,9,1},{53,2,5}};
   int Dy[3][3]={{5,55,3},{2,32,1},{3,53,5}};
   int Dz[3][3]={{5,8,55},{2,9,32},{3,2,53}};
   int x,y,z;

   x=det(Dx)/det(A);
   y=det(Dy)/det(A);
   z=det(Dz)/det(A);
   cout<<x<<"  "<<y<<"  "<<z<<endl;
}
```

Solução alternativa

```
#include<iostream>
using namespace std;

int det(int x[3][3]){
   int i,j;
   i=(x[0][0]*x[1][1]*x[2][2])+(x[0][1]*x[1][2]*x[2][0])+(x[1][0]*x[2][1]*x[0][2]);
```

```
   j=(x[2][0]*x[1][1]*x[0][2])+(x[0][0]*x[1][2]*x[2][1])+(x[1][0]*x[0][1]*x[2][2]);
   return (i-j);
}

int main(){
   int A[3][3]={{5,8,3},{2,9,1},{3,2,5}};
   int te[3]={55,32,53};
   int DD[3][3]={{5,8,3},{2,9,1},{3,2,5}};
   int x,y,z;
   DD[0][0]=te[0];
   DD[1][0]=te[1];
   DD[2][0]=te[2];
   x=det(DD)/det(A);
   DD[0][0]=A[0][0];
   DD[1][0]=A[1][0];
   DD[2][0]=A[2][0];

   DD[0][1]=te[0];
   DD[1][1]=te[1];
   DD[2][1]=te[2];
   y=det(DD)/det(A);
   DD[0][1]=A[0][1];
   DD[1][1]=A[1][1];
   DD[2][1]=A[2][1];

   DD[0][2]=te[0];
   DD[1][2]=te[1];
   DD[2][2]=te[2];
   z=det(DD)/det(A);
   cout<<x<<"  "<<y<<"  "<<z<<endl;
}
```

8.31 (triangpascal)

Escreva um programa C++ que imprima o triângulo de pascal de ordem 12 (setw 6 é suficiente).

```
#include<iostream>
#include<iomanip>
#define tam 13
using namespace std;
int main(){
   int tp[tam][tam]={0}; // ou {};
   int i,j;
   for (i=0;i<tam;i++){
      for (j=0;j<tam;j++){
         if (j==0){
            tp[i][j]=1;
         }
      }
   }
   for (i=1;i<tam;i++){
      for (j=1;j<tam;j++){
         tp[i][j]=tp[i-1][j-1]+tp[i-1][j];
      }
   }
   for (i=0;i<tam;i++){
      for (j=0;j<tam;j++){
         cout<<setw(6)<<tp[i][j];
         }
         cout<<endl;
   }
}
```

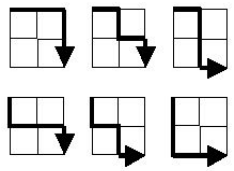
8.32 (mattransposta)

Escreva uma função que receba a matriz A (7×7) e calcule a sua transposta A^t.

```
#include<iostream>
#include<iomanip>
using namespace std;
int transp (int A[7][7], int T[7][7]){
   int i,j;
   for (i=0;i<7;i++){
      for (j=0;j<7;j++){
         T[i][j]=A[j][i];
      }
   }
}
int main(){
   int x[7][7];
   int y[7][7];
   int i,j;
   for (i=0;i<7;i++){
      for (j=0;j<7;j++){
         x[i][j]=(i*7)+j;
      }
   }
   for (i=0;i<7;i++){
      for (j=0;j<7;j++){
         cout<<setw(8)<<x[i][j];
         }
         cout<<endl;
   }
   transp(x,y);
   cout<<"----------------------------"<<endl;
   for (i=0;i<7;i++){
      for (j=0;j<7;j++){
         cout<<setw(8)<<y[i][j];
         }
         cout<<endl;
   }
}
```

8.33 (treliça-euler15)

Começando no canto esquerdo de uma treliça 2×2 e permitindo-se apenas movimentos para a direita e para baixo existem exatamente 6 rotas até o canto inferior da treliça, como se pode ver em



Escreva um programa C++ que calcule a quantidade de rotas em uma treliça 16×16 (a resposta é 601080390).

```
#include<iostream>
using namespace std;
int main(){
  long int x[17][17];
  int i,j;
  for (i=0;i<17;i++){
    x[0][i]=x[i][0]=1;}
  for (i=1;i<17;i++){
    for (j=1;j<17;j++){
        x[i][j]=x[i][j-1]+x[i-1][j];
    }
```

```
  }
  cout<<x[16][16];
}
 a matriz... (só as primeiras 7 linhas e colunas)
     1     1     1     1     1     1     1
     1     2     3     4     5     6     7
     1     3     6    10    15    21    28
     1     4    10    20    35    56    84
     1     5    15    35    70   126   210
     1     6    21    56   126   252   462
     1     7    28    84   210   462   924
```

Ordenações em C++

```
#include<iostream>
using namespace std;
void bolha(int v[], int tam){
  int i,j;
  i=tam-1;
  while (i>=1){
    j=1;
    while(j<=i){
      if (v[j-1]>v[j]){
         swap(v[j-1],v[j]);
      }
      j++;
    }
    i--;
  }
}
void bolhaest(int v[], int tam){
  int i,j,mudou;
  i=tam-1;
  mudou=1;
  while ((i>=1)&&mudou){
    j=1;
    mudou=0;
    while(j<=i){
      if (v[j-1]>v[j]){
        swap(v[j-1],v[j]);
        mudou=1;
      }
      j++;
    }
    i--;
  }
}
void inser(int v[], int n) {
  int i, aux, j;
  for (i = 1; i < n; i++) {
    aux = v[i];
    j = i-1;
    while (j >= 0 && v[j] > aux) {
      v[j+1] = v[j];
      j = j-1;
    }
    v[j+1] = aux;
  }
}
void selec (int v[], int tam) {
  int i,j,cor,inui;
  i=0;
```

```
  while (i<tam){
    cor=v[i];
    inui=i;
    j=i+1;
    while (j<tam){
      if (v[j]<cor){
        cor=v[j];
        inui=j;
      }
      j++;
    }
    swap(v[i],v[inui]);
    i++;
  }
}
void shell(int v[], int tam){
  int k,h,i,j,x,m;
  int decr[3]={5,3,1}; // se quiser, pode alterar os tamanhos
  int tam2;
  tam2=sizeof(decr)/sizeof(decr[0]);
  for (k=0;k<tam2;k++){
    h=decr[k];
    i=h;
    while (i<tam){
      x=v[i];
      j=i;
      while ((j>=h) && (v[j-h]>x)){
        v[j]=v[j-h];
        j=j-h;
      }
      v[j]=x;
      i++;
    }
  }
}
void criaheap(int v[], int n, int i) {
  int maior = i;
  int l = 2*i + 1;
  int r = 2*i + 2;
  if (l < n && v[l] > v[maior]) {
     maior = l;
  }
  if (r < n && v[r] > v[maior]) {
     maior = r;
  }
  if (maior != i) {
    swap(v[i], v[maior]);
    criaheap(v, n, maior);
  }
}
void heapsort(int v[], int n) {
  for (int i = n / 2 - 1; i >= 0; i--){
    criaheap(v, n, i);
  }
  for (int i=n-1; i>=0; i--) {
    swap(v[0], v[i]);
    criaheap(v, i, 0);
  }
}
void quick(int v[], int i, int f) {
  int ant, dep, mei;
  ant=i;
  dep=f;
```

```
  mei=v[(i+f)/2];
  do{
    while(v[ant]<mei){
      ant++;
    }
    while (v[dep]>mei){
      dep--;
    }
    if (ant<=dep) {
      swap(v[ant],v[dep]);
      ant++;
      dep--;
    }
  } while (ant<=dep);
    if (i<dep){
      quick(v, i,dep);
    }
  if (ant<f){
    quick(v,ant,f);
  }
}

int main(){
  int ve[]={7,3,1,9,11,23,6,6,19,10,33,44,55,23,24,25,26};
  int i,tam;
  tam = sizeof(ve)/sizeof(ve[0]);
  bolha(ve,tam);
  cout<<"bolha ";
  for (i=0;i<tam;i++){
    cout<<ve[i]<<' ';
  }
  int ve2[]={7,3,1,9,11,55,88,66,29,100,23,6,6,19,10};
  tam = sizeof(ve2)/sizeof(ve2[0]);
  bolhaest(ve2,tam);
  cout<<endl<<"bolhaestrela ";
  for (i=0;i<tam;i++){
    cout<<ve2[i]<<' ';
  }
  int ve3[]={5,7,1,3,2,4,8,6,9};
  tam = sizeof(ve3)/sizeof(ve3[0]);
  inser(ve3,tam);
  cout<<endl<<"insercao ";
  for (i=0;i<tam;i++){
    cout<<ve3[i]<<' ';
  }
  int ve4[]={22,23,7,3,1,9,11,23,6,6,19,10};
  tam = sizeof(ve4)/sizeof(ve4[0]);
  cout<<endl<<"selecao ";
  selec(ve4,tam);
  for (i=0;i<tam;i++){
    cout<<ve4[i]<<' ';
  }
  int ve5[]={7,3,1,9,11,23,6,6,19,10,8,88,888};
  tam = sizeof(ve5)/sizeof(ve5[0]);
  cout<<endl<<"shell ";
  shell(ve5,tam);
  for (i=0;i<tam;i++){
    cout<<ve5[i]<<' ';
  }
  int ve6[]={7,3,1,88,12,11,10,9,8,6,5,4,28,29,27,31,39,38,37,888};
  tam = sizeof(ve6)/sizeof(ve6[0]);
  cout<<endl<<"heap ";
  heapsort(ve6,tam);
```

```
  for (i=0;i<tam;i++){
    cout<<ve6[i]<<' ';
  }
  int ve7[]={2,22,26,29,7,3,91,1,9,11,23,6,6,14,19,10};
  tam = sizeof(ve7)/sizeof(ve7[0]);
  cout<<endl<<"quick ";
  quick(ve7,0,tam-1);
  for (i=0;i<tam;i++){
    cout<<ve7[i]<<' ';
  }
}
```

8.34 (crivo eratostenes)

Usando a mesma função sdiv, escreva um programa C++ que imprima os primeiros 10.000 números primos

```
int main(){
   int x=1,qt=1,lim=10000;
   while (qt<=lim){
       if (primo(x)==1){cout<<x<<" ";qt++;}
       x++;
   }
}
```

8.35 (Maior distância entre primos)

Escreva um programa C++ que ache a maior distância entre primos consecutivos, considerando

- primos abaixo de 100.000 (R=72, 31469 e 31397)
- primos abaixo de 1.000.000 (R=114, 492113 e 492227)
- primos abaixo de 10.000.000 (R=154, 4652353 e 4652507)

```
int main(){
   int x,ant,maior=-9999999,xm,xa;
   ant=1;
   x=1;
   while (x<100000){
      while (primo(x)==0){
         x++;
      }
      if ((x-ant)>maior){
        maior=x-ant;
        xm=x;
        xa=ant;
      }
      ant=x;
      x++;
   }
   cout<<maior<<" "<<xm<<" "<<xa;
}

Ambas as mains acima, usam:

int primo(int x){
   int r,f;
   if (x==1){return 0;}
   if (x<4){return 1;}
   if (x%2==0){return 0;}
   if (x<9){return 1;}
   if (x%3==0){return 0;}
   r=sqrt(x);
```

```
   f=5;
   while (f<=r){
    if ((x%f)==0){return 0;}
    if ((x%(f+2))==0){return 0;}
    f=f+6;
   }
   return 1;
}
```

8.36 (USP2.6 Fatores primos)

Dado um inteiro, determine sua composição em fatores primos, incluindo multiplicidades, assim:

```
60: 2,2,3,5
28: 2,2,7
1234567: 127 9721      (Obs: Usa-se a mesma função primo aí acima...)
int main(){
   int x,i;
   cin>>x;
   cout<<x<<": ";
   for (i=2;i<=x;i++){
      if (primo(i)==1){
          while (x%i==0){
              cout<<i<<" ";
              x=x/i;
          }
      }
   }
}
```

8.37 (USP2.3: maximoxy)

Dados dois inteiros m e n determinar dentre todos os pares de números inteiros x e y, tais que $0 \leq x \leq m$ e $0 \leq y \leq n$ um par para o qual o valor da expressão $xy - x^2 + y$ seja máximo. Mostrar o valor do máximo e de x e y.

```
int main(){
  int x,y,m,n,gy,gx,max=-99999;
  cin>>m>>n;
  for (x=0;x<=m;x++){
    for (y=0;y<=n;y++){
      if ((y+(x*y)-(x*x))>max){
         max=y+(x*y)-(x*x);
         gy=y;
         gx=x;
      }
    }
  }
  cout<<" O maximo e "<<max<<"  com "<<gx<<" e "<<gy;
}
```

8.38 (USP2.2: hipotenusas)

Dado um inteiro positivo n determinar todos os inteiros entre 1 e n que podem ser comprimentos de uma hipotenusa de um triângulo retângulo com catetos inteiros.

```
int main(){
   int n,i,j,k;
   cin>>n;
   for (i=5;i<=n;i++){
      for(j=1;j<=i;j++){
        for (k=1;k<=i;k++){
            if (((j*j)+(k*k))==(i*i)){
```

```
                cout<<i<<"       catetos: "<<j<<" e "<<k<<endl;
                k=99999;
                j=99999;
            }
        }
      }
   }
}
```

Relembrando, Vetor: array homogêneo de 1 dimensão. Coleção de variáveis com o mesmo nome e individualizadas pelo seu índice. É homogêneo porque todas elas têm que ter o mesmo tipo (em Python ou APL não é assim). A razão do C++ agir assim é pelo desempenho: todos os elementos ocupando o mesmo tamanho, fica fácil endereção qualquer elemento do vetor. Na memória elas residem com continuidade, e seu tamanho total é fixo: não pode aumentar.

O tamanho do vetor é escrito ao lado do nome dentro de colchetes. Por exemplo

```
float INF[12];
int data[3];
```

A numeração começa em zero. Então a primeira data é $data[0]$, a segunda $data[1]$ e a terceira $data[2]$. Não existe a $data[3]$, embora o compilador não dê erro. Fazer o teste com

```
int main(){
   int data[3];
   data[0]=10;
   data[1]=5;
   data[2]=2022;
   data[3]=90;
   int i;
   for (i=0;i<4;i++){cout<<data[i]<<" ";}}
```

Não se pode escrever `int x=10; int f[x];` mas é possível `#define T 10  int f[T]`. Aproveite para descrever o funcionamento do DEFINE.

depois que o vetor foi definido, $data = data + 1$ dá erro enquanto $cout << data$ imprime o ENDEREÇO de data. Para todos os efeitos é como se fosse um erro: O que vc vai fazer com o endereço ? Portanto, depois do vetor definido, seu nome SEMPRE deve vir acompanhado de colchetes. Novamente em Python ou APL, isto não é verdade. A referência à variável sem o colchete deve ser entendida como a TODA ELA.

$v[0]$ é sempre o primeiro e $v[t-1]$ é sempre o último. Claro que se pode escrever $v[x]$ sendo x uma variável INTEIRA e cujo valor vai dizer qual elemento de v sofre a operação.

Pode-se inicializar um vetor com chaves:

```
int x[4] = {1,2,3,4};
int x[4] = {1,2,3,4,5}; // dará erro
int x[4] = {1}; // funcional legal, valendo 1,0,0
```

Daí que para inicializar um vetor enorme com zeros, dá para fazer $v[1000] = 0$;

vetores parece que foram feitos para usar com for, vejamos

```
int i,x[10];
for (i=0,i<10;i++){x[i]=i;}
```

Quando passado como parâmetro para uma função um vetor vai por REFERÊNCIA, mesmo sem haver o &. Execute (lembre o significado de void=vazio)

```
void funcao(int v[3]){
  v[0]=1; v[1]=2; v[2]=100;  // se puser // volta 7,9,11 - teste !
  }
int main(){
  int x[3]={7,9,11};
  funcao(x);
  cout<<x[0]<<" "<<x[1]<<" "<<x[2]; }
```

Exercícios: o que faz:

```
int i,x[10]={1,2,3,4,5,6,7,8,9,10},s=0;
for (i=1;i<10;i=i+2){s=s+x[i];}
cout<<s;  // 30
```

```
for (i=0;i<7;i=i+3){x[i]=i;}
cout<<x[0]<<x[1]<<x[2]<<x[3]<<x[4];
cout<<x[5]<<x[6]<<x[7]<<x[8]<<x[9]; //02335668910

for (i=0;i<10;i++){x[i]=(2*i)+i;}
// 0369121518212427
```

Uma aplicação prática:
Suponha que a poupança (em milhares de reais) ao longo de 6 anos seja int i[6]=10,12,22,25,17,65. Como por isso em um gráfico ? Qual a média mensal ?

```
int i,j,p[6]={10,12,22,25,17,65};
int main(){
for (i=0;i<6;i++){
    cout<<p[i]<<": ";
    for (j=0;j<p[i];j++){
        cout<<"*";
    }
    cout<<endl;  } }
```

Mais uma: Vamos ver se um dado é viciado. Lançando-o 10.000 vezes:

```
#include<iostream>
#include<ctime>
using namespace std;
int main(){
   int i,res,freq[7]={0}; // nao usa o 0
   srand(time(0));  // semente
   for (i=0;i<10000;i++){
       res=1+rand()%6;
       freq[res]++;
   }
   for (i=1;i<7;i++){
       cout<<i<<" - "<<freq[i]<<endl;
   }
}
```

Mais um exemplo: Suponha uma loja com 50 vendedores que fizeram 1700 vendas no mês passado. Cada venda é conhecida como RG-vendedor, valor-venda. Escreva um programa C++ que leia os pares de dados (condição de fim -1, -1) e imprima o rg do vendedor campeão.
Considerações:

- ambiente de teste e de produção. Deploy. Vamos fazer o seguinte teste

  ```
  1450,100
  2430,1000
  19,530
  1450,1000
  2200,1200
  1450,300
  1200,2000
  1450,650
  19,666
  -1,-1
  ```

- Reduzir os vendedores a 10.
- Conectar 2 vetores: o vetor de vendedores e o vetor de vendas, ambos com o mesmo índice
- Ao chegar um vendedor, deve-se percorrer o vetor de vendedores até achar este RG ou até achar um RG=0. (Teorema de Morgan)

  ```
  int rg[V]={0};
  float val[V]={0.0};
  int main(){
     int i,quem,quanto,campe;
  ```

```
        float maxi=-999999;
        while (1==1){
           cin>>quem>>quanto;
           if ((quem==-1)&&(quanto==-1)){break;}
           i=0;
           while((rg[i]!=quem)&&(rg[i]!=0)){
                i++;
           }
           rg[i]=quem;
           val[i]=val[i]+quanto;
        }
        for (i=0;i<V;i++){
           cout<<rg[i]<<" - "<<val[i]<<endl;
        }
        for (i=0;i<V;i++){
           if (val[i]>maxi){maxi=val[i];campe=i;}
        }
        cout<<"O campeao é "<<rg[campe]; }
```

Sugestões de incremento (para próximas aulas)

- strings como vetores alfanuméricos
- calendário
- indireção
- pesquisa linear e binária. Busca com sentinela
- ordenação
- inclusão e exclusão
- heap ???

8.39 O problema fundamental da CC

O problema fundamental da Ciência da Computação (CC): dado um vetor e uma chave, encontrar a localização (índice) da chave no vetor ou a informação NÃO EXISTE.

Vetor aqui deve ser entendido em sentido amplo: um arquivo, um banco de dados, um relatório, um site na web, ...

8.40 Busca simples:

```
int x[1000]={devidamente inicializado};
int busca(int x[1000], int key){
   int i;
   i=0;
   while (i<1000)&&(x[i]!=key){
       i++;
   }
   if (i==1000){return -1}
   else {return i}
}
```

Note que como vetores precisam ser definidos com seu tamanho final (eles não podem crescer), é costume definir uma área MAIOR e sinalizar onde ela realmente acaba (em termos lógicos). Pode-se fazer isto de muitas maneiras, aqui vai-se colocar um contador na posição 0 do vetor. Assim:

```
int x[1001]; // espaço para 1000 ocorrências
x[0]=contador de quantas realmente estão em uso.
x[1]...x[x[0]] // as ocorrências reais
x[x[0]+1]...x[1000] // espaço para crescimento.
```

Neste tipo de vetor, a busca acima ficaria:

```
int x[1000]={devidamente inicializado};
int busca(int x[1001], int key){
   int i;
   i=0;
   while (i<x[0])&&(x[i+1]!=key){
       i++;
   }
   if (i==x[0]){return -1}
   else {return i}
}
```

Uma maneira de acelerar a busca é com uma sentinela. Trata-se de inserir no final do vetor o elemento que vai ser buscado. Depois disso, a busca sempre será bem sucedida. (Diminui a quantidade de testes em 50%). Achado o elemento buscado, se ele for a sentinela, isso sinaliza que o elemento não estava presente no vetor. Se não for a sentinela, é porque o elemento existe no vetor.

Acompanhe: Seja X[15]=10,1,6,3,8,12,9,5,21,17,10,0,0,0,0,0,0 O primeiro sinaliza o tamanho Quero pesquisar a chave 77. Na busca simples, eu preciso fazer 20 testes: i<10 (10 vezes) e 77==1, 6, 3, ...

Na busca com sentinela, se eu quero procurar o 77, eu insiro ele na posição 10 (x[0], mas sem incrementar x[0], ou seja o 77 sofreu uma inserção FAKE. O vetor agora fica X[15]=10,1,6,3,8,12,9,5,21,17,10,77,0,0,0,0,0 A busca por 77, só precisa testar se 77=1,6,3,...,77. Como se pode ver a busca sempre vai ser bem sucedida. Quando ela terminar, testa-se se o 77 achado está fora do vetor (o que sinaliza insucesso) ou dentro (sucesso).

Uma melhoria substancial é obtida na busca binária: imagine pesquisar em um dicionário de maneira sequencial (como fizemos até agora). A busca binária divide o universo em 2 e daí escolhe uma das duas metades e assim recursivamente até o final. A contrapartida é a exigência de ordenação (ordenador !) previa. Veja o algoritmo:

```
int x[20]={16,3,5,8,17,22,29,36,44,47,48,49,50, \
           60,70,81,99};
int buscab(int x[], int key){
   int ini,fim,met,lixo;
   ini=0; fim=x[0];
   while (1==1){
      met=(ini+fim)/2;
      if (x[met]==key) {return met;}
      if (ini>fim){return -1;}
      if (key < x[met]){
         fim=met-1;
      }
      else {
         ini=met+1;
      }
   }
}
int main(){
   cout<<buscab(x,99);
}
```

Só que para poder usar a busca binária o vetor tem que ser ordenado...

```
int x[20]={16,55,62,2,7,6,44,21,99,78,61,63,59,\
          35,39,5,91,0,0,0};
int ordena(int v[]){
   int i,j,maxi,imax;
   j=v[0];
   while (j>1){
      i=1;
      maxi=-999999;
      while (i<=j){
         if (v[i]>maxi){maxi=v[i];imax=i;}
         i++;
      }
      swap(v[j],v[imax]);
      j--;
   }
}
```

```
int main(){
  int i;
  ordena(x);
  for (i=0;i<20;i++){cout<<x[i]<<" ";} }
```

8.41 Inclusão

A inclusão no final (vetor desordenado é facil:

```
int x[20]={16,3,5,8,17,22,29,36,44,47,48,49,50, \
           60,70,81,99};
int inclui(int v[], int key){
   if (v[0]>19){return -1;}
   v[0]++;
   v[v[0]]=key;
   return v[0];
}
int main(){
   int i;
   inclui(x,201);
   for (i=0;i<20;i++){
    cout<<x[i]<<" ";
   }  }
```

A inclusão, mantendo o vetor em ordem é mais complicada, pois há que dar um "chega pra lá" nos elementos maiores do que a chave entrante...

```
int x[20]={16,3,5,8,17,22,29,36,44,47,48,49,50, \
           60,70,81,99,0,0,0};
int incluiord(int v[], int key){
   int i,j;
   if (v[0]>=19){return -1;}
   i=1;
   while ((v[i]<key)&&(i<=v[0])){i++;}
   j=v[0];
   while (j>=i){
        v[j+1]=v[j];
        j--;
    }
   v[i]=key;
   v[0]++;
   return i;
}
int main(){
   int i;
   incluiord(x,1000);
   for (i=0;i<20;i++){
    cout<<x[i]<<" ";
   } }
```

8.42 Exclusão

A exclusão pode ser deixando lixo (fácil, basta colocar um número identificável como vazio, por exemplo -0, ou -99999, ou 0 ou qualquer outra coisa inconfundível), mas daí a função de busca precisa levar em consideração este valor.

A função mais correta é a que traz os elementos uma posição à esquerda

```
int x[20]={16,3,5,8,17,22,29,36,44,47,48,49,50, \
           60,70,81,99,0,0,0};
int exclui(int v[], int key){
   int i,j;
   for (i=1;i<v[0];i++){
      if (v[i]==key){break;}
   }
```

```
   if (i==v[0]+1){return -1;}
   j=i;
   while (j<v[0]){
        v[j]=v[j+1];
        j++;
    }
    v[0]--;
}
int main(){
    int i;
   exclui(x,3);
   for (i=0;i<20;i++){
      cout<<x[i]<<" ";
   } }
```

Capítulo 9

Strings

9.1 Relembrando strings

O tipo string como um vetor de caracteres. Em C, o tipo elementar é *char* cujo comprimento é 1 byte. Suas constantes são escritas entre aspas simples, mas podem ser formadas com `\algumacoisa`. Esta barra invertida é chamada ESCAPE, e as principais são `\b`=backspace, `\0`=nulo, `\n`=new line, `\a`=alarme, entre outras.

Em C, é muito ruim manipular (mover, testar, preencher, ...) vetores de char.

> Como em C não há o string, o que existe é o array de char. É um saco, a qualquer momento arrisca-se uma *memory violation*.

```
#include<string.h>
char x[tamanho];
printf("%s\n",x); // para imprimir
strcpy(x,"conteudo para x"); // para assinalar
// é ilegal fazer x="conteudo para x";
strcmp(x,y); //x=destino, y=origem -- é ilegal x<"conteudo"
strcat(x,y); // para concatenar
strlen(x); // para saber o tamanho
```

Daí que em C++ surgiu um novo tipo primitivo chamado *string*. (Eventualmente pode ser necessário `#include<string>`, mas aparentemente não é). Veja um exemplo

```
string aa = "Ola mundo !";
string x("ola mundo");  // inicialização alternativa...
string y={"vamos ver"}; // idem...
```

Note as aspas duplas, para diferenciar do tipo char. (Em Python não é assim...)

Não aparece, mas TODO string termina por um caracter `\0`. É assim que o C (e o C++) sabem onde um string termina.

Registre-se que tecnicamente um string é um array de char, mas ele pode ser entendido como um novo tipo sendo na realidade uma classe (que vem a ser um “tipo” mais ”inteligente” por exemplo na comparação entre strings ou no seu redimensionamento).

Os operadores deste novo tipo são: = atribuição, + concatenação, [] acessar caracteres individualmente, além de << escrever e >> ler. Maior (>) e menor (<) com o significado de vem antes ou vem depois. Também existem as conversões de tipos. Veja exemplos

```
string a0="Ola"
string a1=" Mundo"
string aa=a0+a1;
aa[4]='Z';  // lembre que começa em zero...
cout<<aa;  // mostra ola Zundo
```

O tamanho de uma string é dado por

```
string x = "Ola mundo";
cout<<x.length();  // ou
cout<<x.size(); // ambos dão 9
string z;
z=x; // funciona perfeitamente
```

Na leitura depois de um `string z;` fazer `cin<<z;` tem um problema. A leitura é interrompida após o primeiro espaço em branco, tabulação ou < *enter* >. Para obter a digitação completa até o < *enter* >, supostamente o final da entrada, deve-se mudar o comando de entrada para

```
string z;
getline(cin,z);
```

Já o cout não precisa mudar e funciona igual ao que já se sabe.

Na comparação, dois strings podem ser iguais (mesmo tamanho e mesmo conteúdo). Veja os exemplos

```
   string x,y,z;
   x="ola mundo";
   y="ola";
   z="OLA MUNDO";
   cout<<(x>y);  // dá 1
   cout<<(z<x);  // dá 1
   if (msg1==msg2){cout<<"iguais";} else {cout<<"difs";}
```

O significado de > e < em strings é "vem depois (>), e vem antes (<) na sequência de códigos (ASCII, EBCDIC, UNICODE, ...) empregado. Este tipo de comparação é fundamental para por textos em ordem alfabética.

9.2 Arquivos

Este é um capítulo extenso da programação. Aqui, só alguns pitacos

```
#include<stream>  // arquivos com conteúdos string...
...
    ofstream arqs  // objeto arquivo de saída
    ifstream arqe // idem para arquivo entrada
    arqs.open("/p/ufpr/lixo1.dad");
    arqs<<"vamos ver se vai"<<endl;
    arqs.close();
```

Daí lá no disco...

```
C:\p\ufpr>type lixo1.dad
vamos ver se vai
```

Para fazer o contrário (ler dados do disco...)

```
char data[100];
arqe.open("...nome do arquivo...");
arqe>>data;
...
arqe.close()
```

Para encerrar, um ciclo completo de ler e gravar em arquivos:

```
#include<iostream>
#include<fstream>
using namespace std;
int main(){
   string line;
   ifstream meuarq("/p/ufpr/lixo1.dad");
   if (meuarq.is_open()){  // deu certo abrir ?
      while (!meuarq.eof()){
         getline(meuarq,line);
         cout<<line<<endl;
      }
      meuarq.close();
   }
   else {cout<<"deu xabu no arquivo"<<endl;}
   return 0; }
```

9.3 (USP6.6: frase e palavra)

COMP89 - Dados dois strings (um contendo uma frase e outro contendo uma palavra) determine o número de vezes que a palavra ocorre na frase. Por exemplo, para a frase "ANA E MARIANA GOSTAM DE BANANA"tem-se que a palavra "ANA"ocorre 4 vezes na frase.

```
#include<iostream>
#include<math.h>
#include<string>
using namespace std;
int main(){
  string fr,pa;
  int i,j,nao,qtd=0;
  getline(cin,fr);
  getline(cin,pa);
  // tamanho=fr.length()-1;
  qtd=0;
  for (i=0;i<fr.length()-pa.length()+1;i++){
      nao=0;
      for (j=i;j<pa.length()+i;j++){
        if (fr[j]!=pa[j-i]){nao=1;}
      }
      if (nao==0){qtd++;}
  }
  cout<<qtd;
}
```

9.4 cada palavra em uma linha

Escreva um programa C++ que leia uma frase de até 80 caracteres e imprima cada palavra em nova linha. O separador de palavras é um e só espaço em branco.

```
#include<iostream>
#include<string>
using namespace std;
int main(){
  string fra="ivo viu a uva e a banana";
  string pal,pal1;
 // getline(cin,fra);
  int i,j,k,tam;
  tam=fra.length();
  i=0;
  while (i<tam){
      j=i;
      while (fra[j]!=' '){
        j++;
      }
      cout<<fra.substr(i,j-i)<<"-"; // in,tam
      i=j+1;
  }
  cout<<"\b ";
}
```

9.5 Conversões

```
#include<iostream>
#include<string>
using namespace std;
int main(){
  string x,y,fra="12340";
  int n;
  float f;
```

```
  n=stoi(fra);   // - string to integer
  f=stof(fra);   // - string to float
  cout<<n<<endl;  // 12340
  cout<<f<<endl;  // 12340
  x=to_string(n); // o que for -- para string
  y=to_string(f); // idem -- idem
  cout<<x<<endl; // 12340
  cout<<y<<endl; } // 12340.000000
```

Lembram do palíndromo numérico ?

```
int main(){
  int n,a1,a2,enaoe,j;
  cin>>n;
  enaoe=1;
  j=floor(log10(n));
  while (n>0){
    a1=n%10;
    a2=n/pow(10,j);
    if (a1!=a2){enaoe=0;}
    n=n-a2*pow(10,j);
    n=n/10;
    j=j-2;
  }
  cout<<enaoe; }
```

Agora um outro jeito:

```
int main(){
   int n,i,j,simnao=1;
   string s;
   cin>>n;
   s=to_string(n);
   i=0;
   j=s.size()-1;
   while (i<=j){
      if (s[i]!=s[j]){simnao=0;}
      i++;
      j--;
   }
   cout<<simnao; }
```

Capítulo 10

Matrizes

Matrizes são arranjos homogêneos de 2 dimensões. (linhas e colunas). Usa a mesma definição já estudada para vetores (que eram em 1 dimensão) agora extendido a duas. Veja o exemplo

```
int x[10][30]; // x tem 10 linhas e 30 colunas
 x[0][0]=100; // o elemento da prim. linha e prim. coluna vale 100
```

Veja a seguir como uma matriz é definida e impressa

```
#include<iostream>
#include<iomanip>
using namespace std;
int x[3][4]={1,2,3,4,5,6,7};
main(){
   int i,j;
   for (i=0;i<3;i++){
      for(j=0;j<4;j++){
         cout<<setw(4)<<x[i][j];
      }
      cout<<endl;
   } }
```

Note o uso do $<$ *iomanip* $>$ e o consequente *setw* sinalizando o tamanho de cada campo. Sem ele, a matriz fica descaracterizada. Acompanhe

```
    int i,j;
    for (i=0;i<2;i++){
        for(j=0;j<4;j++){
            cout<<v[i][j]<<" ";
        }
        cout<<endl;   }
RESULTADO----------------------
1 200 1360 2
13000 33 2300 101

    for (i=0;i<2;i++){
        for(j=0;j<4;j++){
            cout<<setw(6)<<v[i][j];
        }
        cout<<endl;   }
RESULTADO----------------------
     1   200  1360     2
 13000    33  2300   101
```

10.1 Vendas da banquinha

Exercício suponha a seguinte matriz de vendas de uma banca de jornais. As linhas referem-se a sorvete, guloseimas, lanches e bebidas. As colunas referem-se aos dados de segunda à sexta feira. Então

```
303  787  726  813  530
511  385  826  572  294
168  890  305  623  929
942  592  304  922  923
```

Determine:

- O valor total vendido na semana naquela banca
- Os 5 valores totais de cada dia
- A média diária de sorvete, guloseima, lanche e bebida
- qual foi o melhor dia para o sorvete ?

```
#include<iostream>
#include<iomanip>
using namespace std;
int v[4][5]={303,787,726,813,530, \
             511,385,826,572,294, \
             168,890,305,623,929, \
             942,592,304,922,923};
main(){
    int vs[5]={0};
    float vp[4]={0};
    int ms=-9999999;
    int i,j,dia;
    int tot=0;
    for (i=0;i<4;i++){
        for(j=0;j<5;j++){
            vp[i]=vp[i]+v[i][j];
            vs[j]=vs[j]+v[i][j];
        }
        tot=tot+vp[i];
    }
    cout<<"Total da semana: "<<tot<<endl;
    cout<<"Totais de cada dia: ";
    for (j=0;j<5;j++){cout<<vs[j]<<" ";}
    cout<<endl<<"Medias de produto: ";
    for (j=0;j<4;j++){cout<<vp[j]/5.0<<" ";}
    for (j=0;j<5;j++){
          if (v[0][j]>ms){ms=v[0][j]; dia=j+1;}
    }
    cout<<endl<<"Melhor dia sorvete foi "<<dia;    }
```

10.2 USP7.4 - Existem repetidos na matriz ?

USP7.4 - Dados dois inteiros positivos m e n (ambos menores que 100) e uma matriz real $A_{m \times n}$ verificar se existem elementos repetidos em A.

```
#include<iostream>
using namespace std;
int main(){
int m,n,i,j,repe,ii,jj,ocara;
float M[100][100];
cin>>m>>n;
for (i=0;i<m;i++){
    for (j=0;j<n;j++){
        cin>>M[i][j];
    }
}
repe=0;
for (i=0;i<m;i++){
    for (j=0;j<n;j++){
        ocara=M[i][j];
```

```
        for (ii=0;ii<m;ii++){
            for (jj=0;jj<n;jj++){
                if (((ii!=i)&&(jj!=j)) && (M[i][j]==M[ii][jj])){repe=1;}
            }
        }
    }
}
if (repe==1){cout<<"Existem repetidos"<<endl;}
else {cout<<"Nao existem repetidos"<<endl;}  }
```

10.3 USP7.5 - matriz de permutação ?

USP7.5 - Dizemos que uma matriz inteira $A_{n\times n}$ é uma matriz de permutação se em cada linha e em cada coluna houver $n-1$ elementos nulos e um único elemento igual a 1. Por exemplo

$$\begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

é de permutação enquanto

$$\begin{pmatrix} 2 & -1 & 0 \\ -1 & 2 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

não é.
Dados um inteiro positivo $n \leq 10$ e uma matriz inteira $A_{n\times n}$ verifique se A é de permutação.

```
#include<iostream>
using namespace std;
int main(){
   int x[10][10];
   int n,i,j,e,sl,sc;
   cin>>n;
   for (i=0;i<n;i++){
     for (j=0;j<n;j++){
        cin>>x[i][j];
     }
   }
   e=1;

   for (i=0;i<n;i++){
     sl=0; sc=0;
     for (j=0;j<n;j++){
        if ((x[i][j]!=0)&&(x[i][j]!=1)){e=0;}
        sl=sl+x[i][j];
        sc=sc+x[j][i];
     }
     if (sl!=1){e=0;}
     if (sc!=1){e=0;}
   }
   if (e==1){cout<<"e permutacao";}
   else {cout<<"nao e permutacao";}  }
```

10.4 (USP1.9: multiplos)

Dados n, i e j imprimir em ordem crescente os n primeiros naturais múltiplos de i, de j ou de ambos.

```
int main(){
   int i,j,n;
   int qtd=0,cand=0;
   cin>>n>>i>>j;
   while (qtd<n){
      if (((cand%i)==0) || ((cand%j)==0)){
```

```
            cout<<cand<<" ";
            qtd=qtd+1;
        }
        cand++;
    }
}
```

10.5 (USP1.8: fatorial)

Dado um inteiro não negativo n determinar $n!$.

```
int main(){
  long int fa=1;
  int n;
  cin>>n;
  while (n>1){
    fa=fa*n;
    n--;
  }
  cout<<fa;
}
```

10.6 matriz inversa

Para acompanhar esta explicação projete o apres_mat_inversa.pdf.

Seja agora o problema clássico de achar a matriz inversa de uma matriz dada. Quem já resolveu este problema em matriz 3×3 sabe o trabalho que dá. Pense agora achar a inversa de matriz 50×50. Vamos lá.

Lembrando o processo, escreve-se a matriz de quem se quer obter a inversa e ao seu lado uma matriz identidade. Depois, mediante operações elementares (i. multiplicação por constante, ii. substituição de uma linha por 2 somadas e iii.troca de linhas) obtém-se a identidade no lugar da matriz original aplicando as mesmas operações naquela que começou como a identidade. Quando a matriz original virar a identidade, a que era a identidade virou a inversa. Vamos ao código

```
#include<iostream>
#include<iomanip>
#define T 4
using namespace std;
int inversa(float M[T][T], float I[T][T]){
   int i,j,k;
   float pivot,alvo;
   for (i=0;i<T;i++){
    for (j=0;j<T;j++){
        if (i==j){I[i][j]=1;}else{I[i][j]=0;}}}
   for (i=0;i<T;i++){
      for (j=0;j<T;j++){
        if (i!=j){
            pivot=-M[j][i]/M[i][i];
            for (k=0;k<T;k++){
                M[j][k]=M[j][k]+M[i][k]*pivot;
                I[j][k]=I[j][k]+I[i][k]*pivot;
            }
        }
      }
      alvo=M[i][i];
      for(k=0;k<T;k++){
        M[i][k]=M[i][k]/alvo;
        I[i][k]=I[i][k]/alvo;
      }
   }
   return 0;
}
int main(){
    int i,j;
```

```
    float m[T][T]={35,20,50,26,14,3,17,47, \
                   18,7,5,17,21,40,2,49};
    float in[T][T];
    inversa(m,in);
    cout.precision(2);
    for (i=0;i<T;i++){
        for (j=0;j<T;j++){
            cout<<setw(10)<<in[i][j];
        }
        cout<<endl;
    }
}

Deu:
   -0.0025    -0.017     0.089    -0.013
    0.0098    -0.022    -0.034     0.028
     0.022    0.0079    -0.044   -0.0038
   -0.0078     0.025   -0.0083    0.0035
```

10.7 (USP8.14: quadrado latino)

Diz-se que uma matriz $A_{n\times n}$ é um *quadrado latino* de ordem n se em cada linha e em cada coluna aparecem todos os inteiros $1, 2, 3, ...n$ ou seja cada linha e cada coluna é permutação dos inteiros $1, 2, 3, ...n$. Exemplo

$$\begin{pmatrix} 1 & 2 & 3 & 4 \\ 2 & 3 & 4 & 1 \\ 4 & 1 & 2 & 3 \\ 3 & 4 & 1 & 2 \end{pmatrix}$$

a matriz acima é um quadrado latino de ordem 4.

(a) Escreva uma função que receba como parâmetros um inteiro n e um vetor v com n inteiros e verifica se em v ocorrem todos os inteiros de 1 a n.

(b) Dados um inteiro positivo n e uma matriz inteira $A_{n\times n}$ usando a função acima, verifique se a matriz A é um quadrado latino de ordem n.

```
int vetlat(int n, int v[10]){
  int i,j,resp,qt=0;
  for (j=1;j<=n;j++){
    resp=0;
    for (i=0;i<n;i++){
        if (v[i]==j){resp=1;}
    }
    qt=qt+resp;
  }
  return qt==n;
}
int main(){
  int n,i,j,linhas=0,colunas=0,resposta;
  int a[10][10];
  int v[10];
  cin>>n;
  for (i=0;i<n;i++){
    for(j=0;j<n;j++){
        cin>>a[i][j];
    }
  }
  //linhas
  for(i=0;i<n;i++){
    linhas=linhas+vetlat(n,a[i]);
  }
  //colunas
  for (i=0;i<n;i++){
```

```
    for(j=0;j<n;j++){
        v[j]=a[j][i];
    }
    colunas=colunas+vetlat(n,v);
  }
  resposta=(linhas==n)&&(colunas==n);
  cout<<resposta<<endl;
}
```

10.8 (USP8.13: média na matriz)

FEA 88

(a) Escreva uma função que receba como parâmetro dois inteiros positivos m e n uma matriz real $A_{m\times n}$ e uma posição i,j da matriz, calcula a média aritmética dos quatro (ou três, ou dois) vizinhos do elemento (i, j) da matriz. Desconsidere os vizinhos que não pertencem a matriz. Por exemplo, os vizinhos de 0,0 são apenas o 1,0 e 0,1.

(b) Escreva uma função que recebe os valores m, n e $A_{m\times n}$ e devolve a matriz A^{media}, onde $A^{media}_{i,j}$ é a média aritmética dos vizinhos de i, j calculados pela função anterior.

(c) Escreva um programa que lê três inteiros, m, n e k e uma matriz $A_{m\times n}$ e utilizando a função anterior transforma a matriz k vezes, imprimindo a matriz inicial e a obtida após cada transformação.

```
#include<iostream>
#include<iomanip>
using namespace std;
float media(int m, int n, int i, int j, float mat[10][10]){
   int qt=0;
   float so=0.0;
   if (i>0){qt=qt+1; so=so+mat[i-1][j];}
   if (i<(m-1)){qt=qt+1;so=so+mat[i+1][j];}
   if (j>0){qt=qt+1; so=so+mat[i][j-1];}
   if (j<(n-1)){qt=qt+1;so=so+mat[i][j+1];}
   return so/qt;
}
void matmed(int m, int n, float mat[10][10], float med[10][10]){
   int i,j;
   for (i=0;i<m;i++){
    for(j=0;j<n;j++){
        med[i][j]=media(m,n,i,j,mat);
    }
   }
}
void print(int m, int n, float mat[10][10]){
  int i,j;
  for (i=0;i<m;i++){
    for(j=0;j<n;j++){
        cout<<setprecision(3);
        cout<<setw(9);
        cout<<mat[i][j]<<" ";
    }
    cout<<endl;
  }
}
void copia(int m, int n, float mat[10][10], float m2[10][10]){
  int i,j;
  for (i=0;i<m;i++){
    for(j=0;j<n;j++){
       m2[i][j]=mat[i][j];
    }
  }
}
int main(){
   int m,n,i,j,k;
```

```
   float mat[10][10], m2[10][10];
   cin>>m>>n>>k;
   for (i=0;i<m;i++){
    for(j=0;j<n;j++){
        cin>>mat[i][j];
    }
   }
   print(m,n,mat);
   for(;k>0;k--){
      matmed(m,n,mat,m2);
      cout<<"-----------------"<<endl;
      print(m,n,m2);
      copia(m,n,m2,mat);
   }
}
```

Nota de implementação: A função de cópia de uma matriz para a outra (função `copia` acima), poderia ser pensada de outra maneira, talvez para ser mais eficiente. Há uma discussão grande sobre isso, mas não há nenhuma *bala de prata*. Assim, para manter as coisas simples e 100% funcionais em qualquer ambiente, usa-se a solução óbvia: mover elemento a elemento. Funciona e é confiável.

10.9 USP7.6 Linhas e colunas nulas

USP7.6 Dados dois inteiros positivos $m(\leq 10)$ e $n(\leq 10)$ e uma matriz $A_{m\times n}$ imprimir o número de linhas e de colunas nulas da matriz. Exemplo: $m = 4$ e $n = 4$

$$\begin{pmatrix} 1 & 0 & 2 & 3 \\ 4 & 0 & 5 & 6 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

tem 2 linhas e 1 coluna nulas.

```
#include<iostream>
using namespace std;
int main(){
   int m,n,i,j,col,lin,qtd=0;
   float a[10][10];
   cin>>m>>n;
   for (i=0;i<m;i++){
      for (j=0;j<n;j++){
         cin>>a[i][j];
      }
   }
   for (i=0;i<m;i++){
      lin=0;
      for (j=0;j<n;j++){
         if (a[i][j]!=0){lin=1;}
      }
      if (lin==0){qtd++;}
   }
   for (i=0;i<n;i++){
      col=0;
      for (j=0;j<m;j++){
         if (a[j][i]!=0){col=1;}
      }
      if (col==0){qtd++;}
   }
   cout<<qtd;  }
```

10.10 USP7.7 - quadrado mágico

USP7.7 Dizemos que uma matriz quadrada é um quadrado mágico se a soma dos elementos de cada linha, a soma dos elementos de cada coluna e a soma dos elementos das diagonais principais e secundária são todos iguais. Por exemplo, a

matriz A

$$\begin{pmatrix} 8 & 0 & 7 \\ 4 & 5 & 6 \\ 3 & 10 & 2 \end{pmatrix}$$

é um quadrado mágico. Dado um inteiro n e uma matriz quadrada $n \times n$ verifique se A é um quadrado mágico.

```
#include<iostream>
using namespace std;
int main(){
   int n,i,j,quadr=1,somac=0,sl=0,sc=0;
   float a[10][10];
   cin>>n;
   for (i=0;i<n;i++){
      for (j=0;j<n;j++){
         cin>>a[i][j];
      }
   }
   for (i=0;i<n;i++){somac=somac+a[i][i];}
// calculo das linhas
   for (i=0;i<n;i++){
       sl=0;sc=0;
       for (j=0;j<n;j++){
           sl=sl+a[i][j];
           sc=sc+a[j][i];
       }
       if ((sl!=somac)||(sc!=somac)){quadr=0;}
   }
   sl=0;
   for (i=0;i<n;i++){
      sl=sl+a[i][(n-1)-i];
   }
   if (sl!=somac){quadr=0;}
   cout<<quadr; }
```

10.11 USP7.8 - Triângulo de pascal

USP7.8 Dado um inteiro positivo n imprimir as n primeiras linhas do Triângulo de Pascal.

```
 1
 1  1
 1  2  1
 1  3  3  1
 1  4  6  4  1
 1  5  10 10 5  1
```

```
#include<iostream>
#include<iomanip>
using namespace std;
int main(){
   int n,i,j;
   float p[20][20]={0};
   cin>>n;
   for (i=0;i<n;i++){
      p[i][i]=p[i][0]=1;
   }
   for (i=1;i<n;i++){
      for (j=1;j<i;j++){
        p[i][j]=p[i-1][j]+p[i-1][j-1];
      }
   }
   for (i=0;i<n;i++){
    for (j=0;j<n;j++){
        cout<<setw(5)<<p[i][j];
    }
```

```
    cout<<endl;
   }   }
```

O mesmo enunciado, usando apenas dois vetores

```
int main(){
   int n,i,j;
   int p[20]={0};
   int ant[20]={0};
   cin>>n;
   p[0]=1; ant[0]=1;
   for (i=1;i<n;i++){
      cout<<"   1";
      for(j=1;j<i;j++){
          p[j]=ant[j]+ant[j-1];
          cout<<setw(4)<<p[j];
      }
      cout<<endl;
      for (j=0;j<n;j++)
         ant[j]=p[j];
      }   }
```

10.12 USP8.8d - matriz identidade ?

USP8.8d - Faça uma função que verifica se uma matriz $m \times m$ é a matriz identidade

```
int ident(int a[m][m]){
  int i,j,e=1;
  for (i=0;i<m;i++){
    for (j=0;j<m;j++){
      if ((i!=j)&&(a[i][j]!=0)){e=0;}
      if ((i==j)&&(a[i][j]!=1)){e=0;}
    }
  }
  return e;
}
```

10.13 USP8.16c - multiplicação matricial

USP8.16c - Escreva uma função que receba duas matrizes quadradas A e B de ordem m e devolva (também via parâmetro) a multiplicação matricial de A e B.

```
#include<iostream>
#include<iomanip>
#define m 3
using namespace std;
void mm(int a[m][m], int b[m][m], int r[m][m]){
  int i,j,k,s;
  for (i=0;i<m;i++){
    for (j=0;j<m;j++){
      s=0;
      for (k=0;k<m;k++){
        s=s+a[i][k]*b[k][j];
      }
      r[i][j]=s;
    }
  }
}
int  main (){
  int a[3][3]={1,2,3,4,5,6,7,8,9};
  int b[3][3]={2,4,6,8,10,12,14,16,18};
  int r[3][3]={0};
  mm(a,b,r);
```

```
  int i,j;
  for (i=0;i<3;i++){
      for(j=0;j<3;j++){
        cout<<setw(4)<<r[i][j];
      }
      cout<<endl;
  }  }
----------------- resposta é ------------
 60   72   84
132  162  192
204  252  300
```

10.14 Lagrange

LAGRANGE - Existe um algoritmo de interpolação, denominado Lagrange, que recebendo dois vetores, denominados x e fx de mesmo comprimento e um novo valor chamado nx, calcula o polinômio interpolador (a partir de x e fx) e depois aplica o valor nx a este polinômio imprimindo o valor de $fx(nx)$.

$$P_n(x) = \sum_{k=0}^{n} y_k L_k(x)$$

e

$$L_k(x) = \frac{\prod_{\substack{j=0 \\ j\neq k}}^{n} (x - x_j)}{\prod_{\substack{j=0, \\ j\neq k}}^{n} (x_k - x_j)}$$

```
#include<iostream>
#include<iomanip>
using namespace std;
int main(){
  int n,i,j;
  float x[100];
  float fx[100];
  float nx;
  cin>>n;
  cout<<"----- valores de x ------"<<endl;
  for (i=0;i<n;i++){cin>>x[i];}
  cout<<"----- valores de fx ------"<<endl;
  for (i=0;i<n;i++){cin>>fx[i];}
  cout<<" Valor de nx ";
  cin>>nx;
  float yp=0.0;
  for (i=0;i<n;i++){
    float p=1.0;
    for (j=0;j<n;j++){
        if (i!=j){p=p*(nx-x[j])/(x[i]-x[j]);}
    }
    yp=yp+p*fx[i];
  }
  cout<<yp;   }
```

10.15 Mult. matr. é comutativa ?

Sabe-se que, em geral, a multiplicação matricial não é comutativa, isto é: A.B $\neq$ B.A, embora em alguns casos seja. Escreva uma função que receba 2 matrizes A e B ambas de ordem n ($n < 20$) e responda 1 se sua multiplicação matricial for comutativa e 0 senão.

```
#include<iostream>
#define m 4
using namespace std;
```

```
void mm(int a[m][m], int b[m][m], int r[m][m]){
  int i,j,k,s;
  for (i=0;i<m;i++){
    for (j=0;j<m;j++){
      s=0;
      for (k=0;k<m;k++){
        s=s+a[i][k]*b[k][j];
      }
      r[i][j]=s;
    }
  }
}
int main(){
   int n,i,j;
   int a[4][4]={1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16};
   int b[4][4]={0,0,1,2,3,4,5,6,0,0,7,8,9,0,0,0};
   // cin>>n;
   // for(i=0;i<n;i++){
   //   for(j=0;j<n;j++){
   //      cin>>a[i;j]>>b[i;j]; }}
   n=4;
   int rab[4][4];
   int rba[4][4];
   mm(a,b,rab);
   mm(b,a,rba);
   int e=1;
   for (i=0;i<n;i++){
      for(j=0;j<n;j++){
         if (rab[i][j]!=rba[i][j]){e=0;}
      }
   }
   cout<<e;
}
```

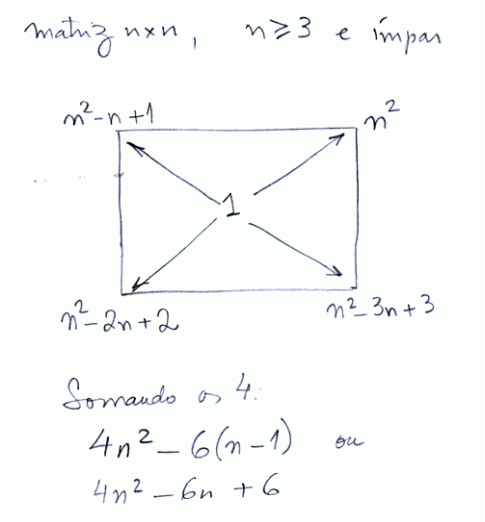
10.16 vivo035c - soma das diagonais crescentes

SOMA DAS DIAGONAIS (vivo035c) - EULER28 - Começando com o número 1 e movendo-se para a direita no sentido horário, uma espiral de 5 por 5 é formada da seguinte forma:

21	22	23	24	25
20	7	8	9	10
19	6	1	2	11
18	5	4	3	12
17	16	15	14	13

Pode-se verificar que a soma dos números nas diagonais é 101.

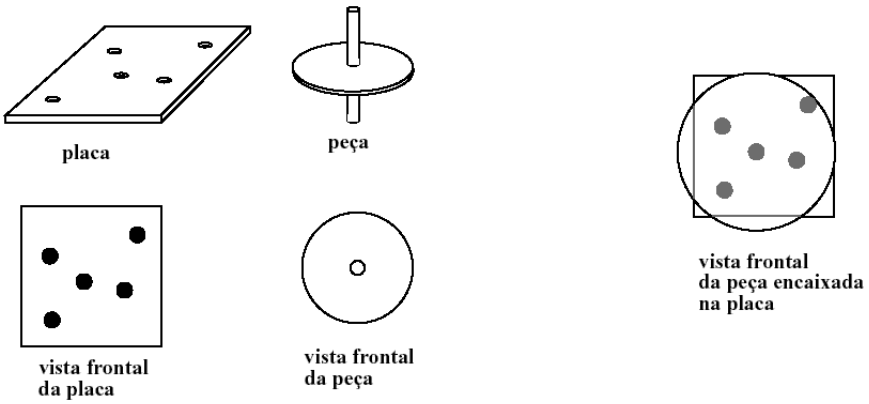
Qual é a soma dos números nas diagonais em uma espiral de 1001 por 1001 formada da mesma maneira?



```
#include<iostream>
using namespace std;
long long int eulr(long long int k) {
   if (k>2) {
      return ((4*k*k) - (6*(k-1)) + eulr(k-2));
   }
   else {
      return 1;
   }
}
long long int euli(long long int k) {
   int j, s=1;
   for (j=3;j<=k;j=j+2) {
      s=s+(4*j*j)-(6*(j-1));
   }
   return s;
}
main() {
   long long int n,s=0;
   cout<<"Informe tam matriz (impar e > 2)";
   cin>>n;
   cout<<eulr(n)<<endl; // recursivo
   cout<<euli(n)<<endl;  } // iterativo,
   //   ambos dao o mesmo valor
```

10.17 VIVOm34 - cubra os furos

Exercício de maratona: CUBRA OS FUROS Uma placa de aço retangular contém N furos circulares de 5 mm de diâmetro, localizados em pontos distintos, não sobrepostos – ou seja, o centro de cada furo está a uma distância maior ou igual a 5 mm do centro de todos os outros furos. Uma peça de forma circular, tendo em seu centro um eixo de 5 mm de diâmetro, deve ser colocada sobre a placa, de modo que o eixo encaixe-se em um de seus furos. Tarefa: Você deve escrever um programa para determinar o diâmetro mínimo que a peça deve ter de tal forma que, com seu eixo encaixado em um dos furos da placa, a parte circular cubra completamente todos os outros furos da placa. Deve informar também qual é o furo que receberá a placa.



Estratégia de solução:

- Ache a distância de cada furo para todos os outros
- Guarde a maior destas distâncias, já que com a chapa disposta neste furo que está sendo examinado, a maior distância é a única que cobre todos os demais.
- Com o vetor preenchido, escolha o menor valor. Seu índice é o furo que deve ser usado.
- Ao guardar a distância, não esquecer de somar 2.5 (o diâmetro do furo) e multiplicar por 2 para passar de raio a diâmetro.

```
#include<iostream>
#include<cmath>
using namespace std;
int main(){
```

```
   int n,i,j,isal;
   float pt[n][2];
   float md[n];
   float maxd,mind,d,p1,p2;
   cin>>n;
   for (i=0;i<n;i++){cin>>pt[i][0]>>pt[i][1];}
   for (i=0;i<n;i++){
        maxd=-99999;
      for (j=0;j<n;j++){
        if (i!=j){
                p1=pow((pt[i][0]-pt[j][0]),2);
                p2=pow((pt[i][1]-pt[j][1]),2);
                d=sqrt(p1+p2);
                if (d>maxd){maxd=d;}
        }
      }
      md[i]=(maxd+2.5)*2;
   }
   mind=9999999;
   for(i=0;i<n;i++){
        if (md[i]<mind){mind=md[i]; isal=i;}
   }
   cout<<mind<<" "<<isal;
}
```

Para testar: 6, 48,78; 65,18; 43,24; 17,71; 50,53 e 70,50. Resposta: 81.157, furo 4.
Mais um: 4, 54,47; 51,31; 61,22 e 28,27. Resposta: 51.690, furo 1.

10.18 vivom44 - Quantos fibonaccis ?

QUANTOS FIBONACCIS ? (vivom44) Relembre-se a definição dos números de Fibonacci
$f_1 = 1$
$f_2 = 1$
$f_n = f_{n-1} + f_{n-2}, (n \geq 3)$
Dados dois números a e b calcule quantos números de Fibonacci existem no intervalo $[a, b]$. Note que deve usar *long long* por que estes números crescem rapidamente.

```
#include<iostream>
#include<cmath>
using namespace std;
int main(){
   long long i,j;
   long long a,c,b,qtd=0;
   cin>>i>>j;
   a=1;
   b=1;
   c=2;
   while(c<=j){
      c=a+b;
      if ((c>=i)&&(c<=j)){qtd++;}
      a=b;
      b=c;
   }
   cout<<"deu: "<<qtd;  }
```

Para testar: 10,100, resposta 5
1234567890,9876543210, resposta 4

10.19 UVA136 e vivom54 - números feios

NÚMEROS FEIOS (vivom54) - Originalmente UVA136 -

Números feios-UVA:136 O exercício a seguir não é igual ao 136 da UVA. Está apenas baseado nele. Números feios são aqueles cujos únicos fatores primos são 3 primos previamente determinados.

Por exemplo, se escolhermos os primos 2, 3 e 5, os primeiro 11 números feios são: 1, 2, 3, 4, 5, 6, 8, 9, 10, 12, 15, ... Por convenção, o 1 é sempre o primeiro.

Se os primos forem 5, 7 e 11, os 11 primeiros serão: 1, 5, 7, 11, 25, 35, 49, 55, 77, 121, 125, ...

Se os primos forem 2, 13 e 19, os 11 primeiros serão: 1, 2, 4, 8, 13, 16, 19, 26, 32, 38, 52, ...

Você deve escrever um programa que receba os 3 primos e o número de ordem do feio a gerar e calcule o feio pedido.

```
#include<iostream>
#include<cmath>
using namespace std;
int main(){
   int p1,p2,p3,n,i=2,iaux,qt=1;
   cin>>p1>>p2>>p3>>n;
   cout<<"1 ";
   while (1==1){
      if (qt==n){break;}
      iaux=i;
      while (0==i%p1){i=i/p1;}
      while (0==i%p2){i=i/p2;}
      while (0==i%p3){i=i/p3;}
      if (i==1){cout<<iaux<<" ";qt++;}
      i=iaux+1;
   }   }
```

Para testar: 5 3 13 55 é 6591
2 3 19 44 é 456
3 5 19 52 é 7695

10.20 Desvio padrão

DESVIO PADRÃO: Escreva um programa que leia n e depois essa quantidade de números reais, e ao final imprima o desvio padrão da distribuição, cuja formulação é

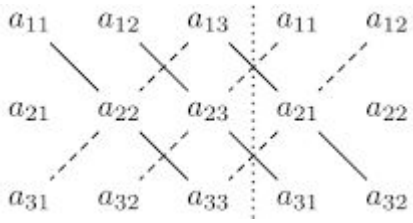
$$\sigma = \sqrt{\frac{\sum (x - \bar{x})^2}{n}}$$

```
#include<iostream>
#include<cmath>
using namespace std;
int main(){
   int n,i;
   float x[100];
   float md,su=0,aux;
   cin>>n;
   for (i=0;i<n;i++){
      cin>>x[i];
      su=su+x[i];
   }
   md=su/n;
   su=0;
   for (i=0;i<n;i++){
      su=su+((x[i]-md)*(x[i]-md));
   }
   su=sqrt(su/n);
   cout<<su;    }
```

Para testar: 5, 1 2 3 10 20. Resposta: 7.138 Mais: 6, 20 50 50 60 70 90. Resposta: 21.343

10.21 Sarrus

SARRUS - Escreva um programa C++ que receba uma matriz 3×3, calcule e imprima o determinante desta matriz usando a regra de Sarrus para efetuar o cálculo.




```
#include<iostream>
using namespace std;
float a[3][3]={{2,4,5},{3,7,0},{1,1,5}};
float sarrus(float x[3][3]){
   float a1,a2;
   a1=x[0][0]*x[1][1]*x[2][2];
   a1=a1+x[0][2]*x[1][0]*x[2][1];
   a1=a1+x[2][0]*x[0][1]*x[1][2];
   a2=x[0][2]*x[1][1]*x[2][0];
   a2=a2+x[0][1]*x[1][0]*x[2][2];
   a2=a2+x[0][0]*x[2][1]*x[1][2];
   return a1-a2;
}
main() {
   cout<<sarrus(a);
}
```

10.22 Operações com conjuntos

Operações com conjuntos. Suponha que um conjunto de números é representado como um vetor de até 21 posições, sendo a primeira usada para informar quantos elementos aquele conjunto possui (obviamente este número chega no máximo a 20). Você deve escrever um programa C++ que leia dois conjunto conforme acima descritos e a seguir imprimir:

- A união desses dois conjuntos
- A intersecção desses dois conjuntos
- A diferença entre o primeiro e o segundo conjuntos
- O produto cartesiano desses dois conjuntos

Antes de sair implementando, relembre-se duas características de conjuntos:

1. Um conjunto não pode (ou não deve) ter elementos repetidos.
2. A ordem dos elementos em um conjunto não é importante e não deve agregar nenhuma informação adicional sobre os elementos.

```
#include<iostream>
using namespace std;
  int a[21];
  int b[21];
  int r[41];
  int i,qta,qtb;
void uniao(int a[21], int b[21], int r[41]){
   int i,j,existe;
   r[0]=0;
   for (i=1;i<=a[0];i++){
      r[r[0]+1]=a[i];
      r[0]++;
   }
   for (j=1;j<=b[0];j++){
      existe=0;
      for (i=1;i<=a[0];i++){
        if (b[j]==a[i]){existe=1;}
      }
      if (existe!=1){
```

```
        r[r[0]+1]=b[j];
        r[0]++;
      }
   }
}
void inter(int a[21], int b[21], int r[41]){
   int i,j,existe;
   r[0]=0;
   for (i=1;i<=a[0];i++){
      existe=0;
      for (j=1;j<b[0];j++){
        if (a[i]==b[j]){existe=1;}
      }
      if (existe==1){
         r[r[0]+1]=a[i];
         r[0]++;
      }
   }
}
void diferenca(int a[21], int b[21], int r[41]){
   int i,j,existe;
   r[0]=0;
   for (i=1;i<=a[0];i++){
      existe=0;
      for (j=1;j<=b[0];j++){
         if (a[i]==b[j]){existe=1;}
      }
      if (existe==0){
         r[r[0]+1]=a[i];
         r[0]++;
      }
   }
}
int main(){
    int i,j;
  cin>>qta;
  a[0]=qta;
  for (i=1;i<=qta;i++){
    cin>>a[i];
  }
  cin>>qtb;
  b[0]=qtb;
  for (i=1;i<=qtb;i++){
    cin>>b[i];
  }
  cout<<"Uniao----------------"<<endl;
  uniao(a,b,r);
  for (i=1;i<=r[0];i++){cout<<r[i]<<" ";}
  cout<<endl<<"Inter-----------"<<endl;
  inter(a,b,r);
  for (i=1;i<=r[0];i++){cout<<r[i]<<" ";}
  cout<<endl<<"Diferenca------"<<endl;
  diferenca(a,b,r);
  for (i=1;i<=r[0];i++){cout<<r[i]<<" ";}
  cout<<endl<<"produto---------"<<endl;
  for (i=1;i<=a[0];i++){
      for(j=1;j<=b[0];j++){
        cout<<"("<<a[i]<<","<<b[j]<<") ";
      }
  }
}
```

10.23 Euler 19 - tripla pitagórica especial

Euler9 - Tripla pitagórica especial Um trio pitagórico é um conjunto de três números naturais, $a < b < c$, para os quais,

$$a^2 + b^2 = c^2$$

Por exemplo, $3^2 + 4^2 = 9 + 16 = 25 = 5^2$.

Existe exatamente uma trinca pitagórica para a qual $a + b + c = 1000$. Encontre o produto abc .

```
#include<iostream>
using namespace std;
int main() {
 int a,b,c;
 for (a=1; a<=1000; a++){
   for (b=1; b<=1000-a; b++){
     c=1000-(a+b);
     if ((a*a+b*b)==c*c) {
      if (a < b) {
            cout << "a= " << a << ", b = " << b << ", c= " << c << endl;
         cout << a*b*c << endl;
        }  }  }  }  }
```

A resposta é $200 \times 375 \times 425$ cujo produto é 31875000.

10.24 Calendário gregoriano

PASCOA Escreva um programa C++ que leia um ano e imprima as 4 datas do calendário móvel (Carnaval, Sexta Feira Santa, Páscoa e Corpus Christi) Gregoriano.

```
#include<iostream>
using namespace std;
// este algoritmo é devido a Aloysius Lilius e
// Cristopher Clavius. Escrito no séc XVI. Ano
// deve ser maior que 1587
int biss(int ano){
   int r4, r100, r400;
   r4=ano%4;
   r100=ano%100;
   r400=ano%400;
   if ((r4==0) && ((r100!=0)||(r400==0))){
       return 1;
   }
   return 0;
}
void pascoa(int ano) {
  int a,b,c,d,e,f,g,h,i,k,l,m,p,q,ordi,carna,ssant,corpc,soma;
  int dm[12]={31,28,31,30,31,30,31,31,30,31,30,31};
  int dp[12] = {0};
  a =ano % 19;
  b =(ano/100.0);
  c=ano % 100;
  d=b/4;
  e=b % 4;
  f=((b+8)/25);
  g=((1+b-f)/3);
  h=((19*a)+b+15-(d+g)) % 30;
  i=(c/4);
  k=c % 4;
  l=(32+(2*e)+(2*i)-(h+k)) % 7;
  m=((a+(11*h)+(22*l))/451);
  p=((h+l+114-(7*m))/31);
  q=(h+l+114-(7*m)) % 31;
  cout<<"pascoa "<<(q+1)<<"/"<<p<<endl;
  if (biss(ano)==1) {
```

```
    dm[1]=29;
  }
  soma=0;
  for (i=1;i<=11;i++){
     dp[i]=dp[i-1]+dm[i-1];
  }
  ordi=dp[p-1]+q+1;
  carna=ordi-47;
  ssant= ordi-2;
  corpc=ordi+60;
  i=0;
  while (dp[i]<carna) {
     i=i+1;
  }
  cout<<"terc carnav "<<(carna-dp[i-1])<<"/"<<i<<endl;
  i=0;
  while (dp[i]<ssant) {
    i=i+1;
  }
  cout<<"sexta santa "<<ssant-dp[i-1]<<"/"<<i<<endl;
  i=0;
  while (dp[i]<corpc) {
    i=i+1;
  }
  cout<<"Corpus Christi "<<corpc-dp[i-1]<<"/"<<i<<endl;
  }
main() {
  int ANO;
  cout<<"Informe ANO ";
  cin>>ANO;
  pascoa(ANO);
}
```

Para conferir, eis os feriados móveis em 2023 e 2024:
Sexta feira da paixão: 7/abr (23) e 29/mar (24). Terça feira de carnaval: 21/fev (23) e 13/fev (24). Quinta feira de Corpus Christi: 8 de jun (23) e 30/mai (24).

10.25 (USP1.24: subnúmero)

São dados dois números inteiros positivos p e q sendo que o número de dígitos de p é menor ou igual ao número de dígitos de q. Verificar se p é um subnúmero de q. Exemplos

- Se $p = 23$ e $q = 57238$, p é subnúmero de q
- Se $p = 23$ e $q = 258347$, então p não é subnúmero de q

```
int main(){   // p ocorre em q ?
  int p[100], q[100];
  int pp,qq,i,j,k,tamq,tamp,auxp,auxq;
  cin>>pp>>qq;
  tamp=0;
  tamq=0;
  for (i=0;pp!=0;i++){
    p[i]=pp%10;
    pp=pp/10;
    tamp++;
  }
  for (i=0;qq!=0;i++){
    q[i]=qq%10;
    qq=qq/10;
    tamq++;
  }
  tamq--;
```

```
  tamp--;
  while (tamq>=0){
    if (p[tamp]==q[tamq]){
            auxp=tamp;
            auxq=tamq;
        while ((auxp>=0)&&(p[auxp]==q[auxq])){
                auxq--;
                auxp--;
        }
        if (auxp<0){cout<<"sim"; break;}
        else {tamq--;}
    }
    else {tamq--;}
  }
}
```

10.26 (USP4.1: permutação)

Um número a é dito permutação de um número b se os dígitos de a formam uma permutação dos dígitos de b. Como por exemplo, 5412434 é uma permutação de 4321445 mas não é uma permutação de 4312455. Considere que o dígito 0 não aparece nos números.

(a) Escreva uma função de nome `contadigitos` que recebendo um inteiro n e um inteiro $d, 0 < d \leq 9$, devolve quantas vezes o dígito d aparece no número n.

(b) Usando a função do item anterior escreva um programa que lê dois números ae b e responda se a é permutação de b.

```
int contadigitos(int n, int d){
   int qtd=0;
   while (n>0){
      if ((n%10)==d){ qtd++;}
      n=n/10;
   }
   return qtd;
}
int main(){
    int a,b,i,nao=0;
    cin>>a>>b;
    for (i=1;i<10;i++){
        if (contadigitos(a,i)!=contadigitos(b,i)){nao=1;}
    }
    if (nao==0){cout<<"permutacao"<<endl;}
}
```

10.27 (USP6.8 duas sequências ordenadas)

Dados dois números naturais m e n e duas sequências ordenadas com m e n números inteiros respectivamente, obter uma única sequência ordenada contendo todos os elementos das sequencias originais sem repetição.

```
int main(){
  int m,n,i,j,k,chave;
  int sm[100],sn[100],r[200];
  cin>>m;
  for (i=0;i<m;i++){
    cin>>sm[i];
  }
  cin>>n;
  for(i=0;i<n;i++){
    cin>>sn[i];
  }
  i=j=k=chave=0;
  while(i<m && j<n){
    if (sm[i]<sn[j]){chave=1;}
```

```
    else {
        if (sn[j]<sm[i]){chave=2;}
        else{chave=3;}}
    if (chave==1){r[k]=sm[i];i++;k++;}
    if (chave==2){r[k]=sn[j];j++;k++;}
    if (chave==3){r[k]=sm[i];i++;j++;k++;}
    }
    if (i==m){for (;j<n;j++){r[k]=sn[j];j++;k++;}}
    if (j==n){for (;i<m;i++){r[k]=sm[i];i++;k++;}}
  cout<<" Impressao do resultado "<<endl;
  for (i=0;i<m+n-1;i++){
    cout<<r[i]<<" ";
  }
}
```

10.28 (USP6.9: soma de duas sequências)

Dados um inteiros n e duas sequências de dígitos com n dígitos entre 0 e 9 interpretadas como 2 inteiros, calcular a sequência de números que representa a soma dos dois inteiros.

```
int main(){
  int n,i,mx,i1,i2,vaium;
  int v1[30],v2[30],r[31]={0};
  cin>>n;
  for (i=0;i<n;i++){
    cin>>v1[i];
  }
  for (i=0;i<n;i++){
    cin>>v2[i];
  }
  mx=n;
  i1=n-1;
  vaium=(v1[i1]+v2[i1])/10;
  r[mx]=(v1[i1]+v2[i1])%10;
  i1--;
  mx--;
  while (mx>0){
     r[mx]=(vaium+v1[i1]+v2[i1])%10;
     vaium=(vaium+v1[i1]+v2[i1])/10;
     i1--;
     mx--;
     if (i1==-1){r[0]=vaium;break;}
  }
  cout<<"----------------------------"<<endl;
  for(i=0;i<n+1;i++){
      cout<<r[i]<<" ";
  }
}
```

Desafio: reprogramar isto com m e n (ou seja com tamanhos diferentes para as 2 sequências.

10.29 (USP8.7: a contido em b?)

(a) Escreva uma função que recebe como parâmetros dois inteiros positivos m e n, um vetor inteiro A com m números e um vetor B com n números inteiros, ambos representando conjuntos e verifica se A está contido em B ($A \subseteq B$).

(b) Use a função acima, para ver se os conjuntos A e B são iguais. ($A = B$ *se e somente se* $A \subseteq B \wedge B \subseteq A$)

```
int subconj(int m, int n, int A[100], int B[100]){
  int i,j,simnao,ctd=0;
  for (i=0;i<m;i++){
    simnao=0;
    for(j=0;j<n;j++){
```

```
        if (A[i]==B[j]){simnao=1;}
    }
    ctd=ctd+simnao;
  }
  if (ctd==m){return 1;}
  else {return 0;}
}
int main(){
  int X[100],Y[100];
  int m,n,i,r;
  cin>>m;
  for(i=0;i<m;i++){
    cin>>X[i];
  }
  cin>>n;
  for(i=0;i<n;i++){
    cin>>Y[i];
  }
  r=subconj(m,n,X,Y);
  if (r==1){cout<<"A e subconjunto de B"<<endl;}
  if (subconj(m,n,X,Y)&&subconj(n,m,Y,X)){cout<<"iguais"<<endl;}
}
```

10.30 Manipulação de matrizes

Para aprender a manipular matrizes, suponha uma matriz 5x5 construída com esta estrutura

```
int mxx(m[5][5]){
  int i=0, j=0;
  while (i<5){
    while (j<5){
      m[i][j]=...
      ...
      j=j+1;
   }
    i=i+1;
 }
}
```

1. Escreva o código que gere

```
[[ 1  2  3  4  5]
 [ 6  7  8  9 10]
 [11 12 13 14 15]
 [16 17 18 19 20]
 [21 22 23 24 25]]
```

2. Idem

```
[[ 1  0  2  0  3]
 [ 4  0  5  0  6]
 [ 7  0  8  0  9]
 [10  0 11  0 12]
 [13  0 14  0 15]]
```

3. Idem

```
[[1 2 2 2 2]
 [3 1 2 2 2]
 [3 3 1 2 2]
 [3 3 3 1 2]
 [3 3 3 3 1]]
```

4. Idem

```
[[25 20 15 10  5]
 [24 19 14  9  4]
 [23 18 13  8  3]
 [22 17 12  7  2]
 [21 16 11  6  1]]
```

5. Idem

```
[[ 2  3  4  5  1]
 [ 6  7  8  1 12]
 [ 9 10  1 13 14]
 [11  1 15 16 17]
 [ 1 18 19 20 21]]
```

6. Idem

```
[[ 1  2  3  4  5]
 [11 12 13 14 15]
 [21 22 23 24 25]
 [31 32 33 34 35]
 [41 42 43 44 45]]
```

7. Idem

```
[[1 2 2 3 3]
 [3 4 4 4 4]
 [5 5 5 5 5]
 [6 6 6 6 6]
 [6 7 7 7 7]]
```

8. Idem

```
[[0 1 2 3 4]
 [1 1 2 3 4]
 [2 2 2 3 4]
 [3 3 3 4 5]
 [4 4 4 5 5]]
```

Respostas

1.

```
int m1(int m[5][5]){
  int ct,i,j;
  ct=1;
  i=0;
  while (i<5){
    j=0;
    while (j<5){
      m[i][j]=ct;
      ct=ct+1;
      j=j+1;
   }
    i=i+1;
  }
  return 0;
}
```

2.

```
int m2(int m[5][5]){
  int ct,i,j;
  ct=1;
  i=0;
  while (i<5){
    j=0;
    while (j<5){
      m[i][j]=ct;
```

```
      ct=ct+1;
      j=j+2;
 }
    i=i+1;
  }
  return 0;
}
```

3.

```
int m3(int m[5][5]){
  int i,j;
  i=0;
  while (i<5){
    j=0;
    while (j<5){
      if (i==j){m[i][j]=1;}
      else {if (i<j){m[i][j]=2;}
             else {m[i][j]=3;} }
      j=j+1;
 }
    i=i+1;
  }
  return 0;
}
```

4.

```
int m4(int m[5][5]){
  int ct,i,j;
  ct=1;
  j=4;
  while (j>=0){
    i=4;
    while (i>=0){
      m[i][j]=ct;
      ct=ct+1;
      i=i-1;
 }
    j=j-1;
  }
  return 0;
}
```

5.

```
int m5(m[5][5]){
  int ct1, ct2, i,j;
  ct1=2;   ct2=12;   i=0;
  while (i<5){
    j=0;
    while (j<5){
      if (i+j<4) {m[i][j]=ct1;  ct1=ct1+1;}
      if (i+j==4) {m[i][j]=1;}
      if (i+j>4) {m[i][j]=ct2;  ct2=ct2+1;}
      j=j+1;
 }
    i=i+1;
  }
  return 0;
}
```

6.

```
int m6(m[5][5]){
  int ct,i,j;
```

```
  ct=1;
  i=0;
  while (i<5){
    j=0;
    while (j<5){
      m[i][j]=ct;
      ct=ct+1;
      j=j+1;
 }
    ct=ct+5;
    i=i+1;
  }
  return 0;
}
```

7.

```
int m7(m[5][5]){
  int ct,vezes,i,j;
  ct=1;
  vezes=0;
  i=0;
  while (i<5){
    j=0;
    while (j<5){
      m[i][j]=ct;
      vezes=vezes+1;
      if (vezes>=ct){ct=ct+1; vezes=0;}
      j=j+1;
 }
    i=i+1;
  }
  return 0;
}
```

8.

```
int m8(m[5][5]){
  int i,j,dd;
  i=0;
  while (i<5){
    j=0;
    while (j<5){
      dd=sqrt(i**2+j**2)
      if (m[i][j]==0){m[i][j]=dd;}
      j=j+1;
 }
    i=i+1;
  }
  return 0;
}
```

Capítulo 11

Estudo para a prova da UFPR

Estas provas a seguir são dos cursos universitários da nossa UFPR na área de exatas. Por escolha dessa universidade todos os cursos da área de exatas (todas as engenharias, matemática, física, química, arquitetura, geologia, etc etc) precisam fazer uma disciplina semestral de programação que estuda o ambiente C++ visto aqui. A seguir algumas provas dadas e resolvidas nesse ambiente.

11.1 Estudo para a prova 2 - UFPR

No material que segue, nos exemplos, os dados fornecidos pelo operador estão escritos em *itálico*, enquanto as respostas dadas pelo programa estão em **negrito**.

1. Escreva um programa que leia um número inteiro positivo N e imprima o próximo inteiro primo P tal que $P \geq N$. Um número primo é aquele que só tem 2 divisores que são a unidade e o próprio número. Assim, 13 é primo porque seus divisores são apenas 1 e 13, enquanto 12 não é primo, já que seus divisores são 1,2,3,4,6 e 12.
   Exemplos de execução
   12 **13**; ou *29* **29**; ou ainda *49* **53**

2. Escreva uma função de nome **termed** que receba 3 números reais e devolva aquele número que tiver o valor intermediário entre os 3. Assim, por exemplo, se a função receber 10,20 e 30, deve devolver o 20. Especial cuidado deve ser tomado se houver 2 números repetidos (ou até 3). Se houver 2 repetidos a função deve devolver o valor repetido e se houver 3, a mesma coisa. Cuidado com eventuais negativos, por exemplo se a função receber -2,5,1 deve devolver 1. Depois que a função estiver pronta, escreva o programa principal que lerá 3 valores do teclado e imprimirá o valor do meio (usando a função acima). Exemplos de execução:
   1,3,4 **3**; ou *3, 5, 5* **5**; ou *10, 10, 10* **10**

3. Escreva uma função de nome **media** que receba um inteiro n ($1 < n \leq 10$) e um vetor de até 10 flutuantes e devolva a média dos elementos do vetor. n sinaliza quantos dos elementos do vetor realmente estão sendo usados nesta chamada, sempre alinhados à esquerda do vetor. Por exemplo, se $n = 5$ e $v = 10, 20, 30, 40, 50, ?, ?, ?, ?, ?$ a função deverá devolver $10 + 20 + 30 + 40 + 50/5$ ou 30. Note que as posições indicadas por ? são desconhecidas e não alteram nada no cálculo.

4. Escreva uma função de nome **dp** que receba um inteiro n ($1 < n \leq 10$) e um vetor de até 10 flutuantes e devolva o desvio padrão dos elementos do vetor. n sinaliza quantos dos elementos do vetor realmente estão sendo usados nesta chamada, sempre alinhados à esquerda do vetor. Como neste cálculo é necessário conhecer a média aritmética, a função anterior deve obrigatoriamente ser usada. O desvio padrão tem como fórmula

   $$dp = \sqrt{\frac{\sum(x - \bar{x})^2}{n}}$$

   Por exemplo, se a função for chamada com 10,20,30,40,50, a média como vimos será 30, e o programa deverá calcular as diferenças: $10 - 30 = -20$, $20 - 30 = -10$, $30 - 30 = 0$, $40 - 30 = 10$ e $50 - 30 = 20$. Elevando estes números ao quadrado $-20^2 = 400$, $-10^2 = 100$, $0^2 = 0$, $10^2 = 100$ e $20^2 = 400$. Somando tudo, fica-se com $400 + 100 + 0 + 100 + 400 = 1000$ e dividindo por 5 fica $1000/5 = 200$. O desvio padrão é $\sqrt{200} = 14.14$

   Depois, usando as duas funções acima, escreva um programa que leia n (inteiro e positivo) e depois n valores reais, imprimindo ao final o desvio padrão dessa amostra.

5. Escreva um programa que leia um inteiro n e depois uma matriz quadrada de ordem n de inteiros ($n \leq 10$) e imprima os seguintes valores: 1 = se a matriz tem pela menos uma linha zerada, 2 = a matriz tem pelo menos uma coluna zerada, 3 = a matriz tem pelo menos uma linha e pelo menos uma coluna zerada. 0 =não há linha nem coluna zerada.

6. Escreva uma função de nome **terreno** que receba 2 números reais indicando as dimensões (largura e comprimento) de um terreno em metros. A função deve devolver 2 números: a área do terreno em m^2 e o perímetro do mesmo em m. Depois, escreva o programa principal que leia dois reais e imprima a área e o perímetro, usando a função recém definida.

7. Escreva uma função de nome **maidife** que receba um vetor de 6 números inteiros em ordem crescente (isto não precisa ser testado) e devolva a maior diferença entre qualquer número e seu vizinho no vetor. Depois escreva o programa principal que usando a função, leia os 6 números em ordem crescente e imprima a maior diferença. Por exemplo:
   1, 7, 8, 22, 50, 52 **28**
   1, 2, 3, 4, 5, 6 **1**
   10,10,10,10,10,10 **0**

8. Escreva uma função de nome **somacum** que receba o inteiro n ($n \leq 10$) e um vetor x de n inteiros e devolva outro vetor também de n inteiros onde o elemento i é igual a $\sum_{j=0}^{j=i} x_j$. Depois escreva a função principal que lê n e v e imprime o resultado da aplicação da função sobre o vetor. Por exemplo
   4, 3 5 11 1 **3 8 19 20**
   7, 1 2 3 4 5 6 7 **1 3 6 10 15 21 28**
   8, 1 0 1 0 1 0 1 0 **1 1 2 2 3 3 4 4**

9. Escreva uma função de nome **rotaciona** que receba um inteiro n (≤ 10), um vetor de n inteiros e um segundo inteiro j ($-n < j < n$) e devolva o mesmo vetor rotacionado em j posições. Valores negativos de j significam rotacionar à direita, enquanto valores positivos implicam em rotacionar à esquerda. Considere que o vetor é circular, como se tivesse as extremidades ligadas.

   Agora escreva um programa que leia n, j e o vetor e imprima o resultado da aplicação da função. Exemplos de uso
   6, 1, 0 1 2 3 4 5 **1 2 3 4 5 0**
   6, 2, 0 1 2 3 4 5 **2 3 4 5 0 1**
   6, -1, 0 1 2 3 4 5 **5 0 1 2 3 4**
   6, -2, 0 1 2 3 4 5 **4 5 0 1 2 3**
   5, -1, 0 1 2 3 4 **4 0 1 2 3**
   5, -4, 0 1 2 3 4 **1 2 3 4 0**

10. Escreva um programa que leia uma frase e imprima a quantidade de palavras que compõe a frase. Considere que o separador de palavras é apenas um espaço em branco. Por exemplo:
    ivo viu a uva **4**
    Ouviram do Ipiranga as margens plácidas **6**

Possíveis soluções

```
#include<iostream>
#include<iomanip>
#include<string>
#include<cmath>
using namespace std;
//--- imprime o próximo primo ----
int eprimo(int x){
   int i=2,d=0;
   while (i<x){
      if (x%i==0){d++;}
      i++;
   }
   return d==0;
}
int proxprim(int n){
   while (!eprimo(n)){
      n++;
   }
   return n;
}
//int main(){
//   int j;
//   cin>>j;
//   cout<<proxprim(j);
//}
//------- imprime o termo médio --------
//a versão anterior estava errada e foi descoberta pela Letícia. Eis a certa
float termed(float a, float b, float c){
   if ((a>=b)&&(b>=c)){return b;}
   if ((b>=a)&&(a>=c)){return a;}
   if ((a>=c)&&(c>=b)){return c;}
   if ((a<=b)&&(b<=c)){return b;}
   if ((b<=a)&&(a<=c)){return a;}
   if ((a<=c)&&(c<=b)){return c;}
}
//int main(){
//  float a,b,c;
//  cin>>a>>b>>c;
//  cout<<termed(a,b,c);
//}
//----------- média ------------
float media(int n, float v[10]){
  float s=0;
  int i;
  for(i=0;i<n;i++){
      s=s+v[i];
  }
  return s/n;
}
//------ desvio padrão ------
float dp(int n, float v[10]){
  float me;
  float dvs=0;
  int i;
  me=media(n,v);
  for (i=0;i<n;i++){
    dvs=dvs+((me-v[i])*(me-v[i]));
  }
  dvs=dvs/n;
  return sqrt(dvs);
}
```

```
/* int main(){
  int n,i;
  float v[10];
  cin>>n;
  for (i=0;i<n;i++){
      cin>>v[i];
  }
  cout<<dp(n,v);
} */
//- existem linhas e colunas zeradas ?-
int quadr(int n, int m[10][10]){
  int lin=0, col=0;
  int i,j,k,z;
  for (i=0;i<n;i++){
     z=0;
     for (k=0;k<n;k++){
        if (m[i][k]!=0){z=1;}
     }
     if (z==0){lin=1;}
     z=0;
     for (k=0;k<n;k++){
        if (m[k][i]!=0){z=1;}
     }
     if(z==0) {col=1;}
  }
  if ((lin==1)&&(col==1)){return 3;}
  if (lin==1){return 1;}
  if (col==1){return 2;}
  return 0;
}
/*
int main(){
  int n,i,j;
  int m[10][10];
  cin>>n;
  for(i=0;i<n;i++){
     for(j=0;j<n;j++){
        cin>>m[i][j];
     }
  }
  cout<<
  (quadr(n,m));
}   */
//--área e perímetro do terreno ----
void terreno(float a, float b,  \
         float & ar, float & pe){
   ar=a*b;
   pe=2*(a+b);
}
/*
int main(){
  float a,b,ar,pe;
  cin>>a>>b;
  terreno(a,b,ar,pe);
  cout<<ar<<" "<<pe;
}  */
//---- maior diferença do vetor --------
int maidife(int v[6]){
  int md=-999999;
  int i;
  for (i=1;i<6;i++){
      if ((v[i]-v[i-1])>md){md=v[i]-v[i-1];}
  }
```

```
  return md;
}  /*
int main(){
  int v[6];
  cin>>v[0]>>v[1]>>v[2]>>v[3]>>v[4]>>v[5];
  cout<<maidife(v);
}  */
//------ soma acumulada do vetor ---------
void somacum(int n, int ve[10], int vs[10]){
  int i,s,j;
  vs[0]=ve[0];
  for (i=1;i<n;i++){
     s=0;
     for(j=0;j<=i;j++){
        s=s+ve[j];
     }
     vs[i]=s;
  }
}
//   somacum alternativo com 1 unico for
void somacum_alt(int n, int ve[10], int vs[10]){
  int i,s,j;
  vs[0]=ve[0];
  for (i=1;i<n;i++){
     vs[i]=vs[i-1]+ve[i];
  }
}  /*
int main(){
  int ve[10], vs[10], n, i;
  cin>>n;
  for(i=0;i<n;i++){cin>>ve[i];}
//  somacum(n,ve,vs);
  somacum_alt(n,ve,vs);
  for(i=0;i<n;i++){cout<<vs[i]<<" ";}
}  */
//--------- rotaciona elementos do vetor -------
void rotaciona(int n, int v[10], int j){
    int aux[10],i,k;
    if (j<0){j=n+j;} // ja que -2 == 3 (para n=5)
    i=0;
    while (i<n){
       k=i+j;
       if(k>=n){k=k%n;}
       aux[i]=v[k];
       i++;
    }
    for (i=0;i<n;i++){
        v[i]=aux[i];
    }
}
int main(){
   int i,n,j,v[10];
   cin>>n>>j;
   for (i=0;i<n;i++){cin>>v[i];}
   rotaciona(n,v,j);
   for (i=0;i<n;i++){cout<<v[i]<<" ";}
}  /*
//---------- palavras na frase --------
int main(){
  string frase;
  int q=1,i;
  getline(cin,frase);
  for (i=0;i<frase.size();i++){
```

```
    if (frase[i]==' '){q++;}
}
cout<<q;   }    */
```

11.2 Estudo para a prova 2a - UFPR

No material que segue, nos exemplos, os dados fornecidos pelo operador estão escritos em *itálico*, enquanto as respostas dadas pelo programa estão em **negrito**.

1. Escreva uma função de nome **takedrop** que receba um inteiro n ($1 < n \leq 10$), uma chave k ($-n < k < n$) e um vetor de n inteiros e aplique a operação *take* se k for positivo e a função *drop* se k for negativo. Acompanhe nos exemplos:
   3 *takedrop* 1 2 3 4 5 6 é igual a 1 2 3
   -3 *takedrop* 1 2 3 4 5 6 é igual a 4 5 6
   1 *takedrop* 10 20 30 40 é igual a 10
   -1 *takedrop* 10 20 30 40 é igual a 20 30 40
   Depois, escreva um programa que leia n, a chave k e o vetor, aplique a função sobre o vetor e imprima o resultado. Veja os exemplos: *6, 3, 1 2 3 4 5 6* **1 2 3**
   6, -3, 1 2 3 4 5 6 **4 5 6**
   7, 2, 10 20 30 40 50 60 70 **10 20**
   5, 0, 1 2 3 4 5 **1 2 3 4 5**
2. Escreva uma função que receba um inteiro n ($n \leq 10$), uma base b ($1 < b \leq 9$) e um vetor de n inteiros. Os valores v do vetor serão $0 \leq v < b$. Este vetor representa um número expresso na base b. O objetivo da função é converter este número para a base decimal. Acompanhe nos exemplos:

   $2\ 2\ 4\ 1_3 = 1 \times 3^0 + 4 \times 3^1 + 2 \times 3^2 + 2 \times 3^3 = 1 \times 1 + 4 \times 3 + 2 \times 9 + 2 \times 27 = 1 + 12 + 18 + 54 = 85$

   $5\ 4\ 3_7 = 3 \times 7^0 + 4 \times 7^1 + 5 \times 7^2 = 3 \times 1 + 4 \times 7 + 5 \times 49 = 3 + 28 + 245 = 276$

   $3\ 4_5 = 4 \times 5^0 + 3 \times 5^1 = 4 \times 1 + 3 \times 5 = 4 + 15 = 19$. Depois escreva um programa que leia n, b e o vetor de números, chame a função e imprima o resultado. Exemplos de uso:
   4, 3, 2 2 0 1 **73**
   3, 7, 5 4 3 **276**
   2, 5, 3 4 **19**
3. Escreva uma função chamada **arredond** que receba n ($n \leq 10$) e um vetor de n reais e devolve um vetor de n inteiros que contém os valores do vetor de entrada devidamente arredondados. Use a regra tradicional para arredondar x, a saber: se a parte fracionária de x é igual ou maior a 0.5, o arredondamento é igual à parte inteira de $x+1$ enquanto se a parte fracionária é menor que 0.5, o arredondamento é igual à parte inteira de x. Acompanhe os exemplos
   $arredond(1.5, 2, 2.99, 30.01, 99) = 2, 2, 3, 30, 99$
   $arredond(1, 1.2, 1.4, 1.6, 1.8, 2) = 1, 1, 1, 2, 2, 2)$
   Depois escreva um programa que leia o vetor de reais e imprima o vetor de inteiros. Veja:
   1.5 2 2.99 30.01 99 **2 2 3 30 99**
   1 1.2 1.4 1.6 1.8 2 **1 1 1 2 2 2**
4. Escreva uma função de nome **emord** que receba um inteiro n ($n \leq 10$) e um vetor composto de n inteiros. A função deve analisar o vetor e responder: 1 = se o vetor estiver em ordem crescente, 2 = se estiver em ordem estritamente crescente e 0 = nenhuma das duas alternativas anteriores. Definição: o vetor v está em ordem crescente se e somente se $v[i] \geq v[j] \forall i > j$ e em ordem estritamente crescente se e somente se $v[i] > v[j] \forall i > j$. Depois escreva um programa que leia n e o vetor associado, chame a função e imprima o resultado devolvido pela função. Acompanhe os exemplos:
   6, 1 2 3 4 5 6 **2**
   6, 1 2 2 3 4 5 **1**
   6, 1 2 3 4 3 6 **0**
5. Escreva uma função de nome **areatri** que receba 6 valores reais, representando 3 pares de pontos, pela ordem x_1, y_1, x_2, y_2, x_3 e y_3 e devolva a área ocupada pelo triângulo determinado por esses 3 pontos. A área de um triângulo é determinada pela metade *do módulo* do determinante da seguinte matriz

$$\begin{vmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ x_3 & y_3 & 1 \end{vmatrix}$$

   Depois que a função estiver operacional, escreva um programa que leia 3 pontos (pelas suas coordenadas cartesianas) e devolva 1 se a origem (o ponto 0,0) estiver DENTRO do triângulo definido pelos 3 pontos e 0 senão. Para resolver este problema pense no seguinte algoritmo. Constróe-se 3 triângulos, tomando 2 pontos dos fornecidos e mais a origem. Se a soma dos 3 triângulos for igual à area do triângulo original, (0,0) está dentro do triângulo. Se a soma for maior que a área do triângulo original, então a origem está fora. Se não entendeu a explicação, faça os

dois desenhos num papel e medite sobre eles. Veja os exemplos
0 0, 0 1, 1 0 **1**
2 2, 3 3, 5 2 **0**

6. Escreva um programa C++ que leia n ($n \leq 10$) e a seguir uma matriz quadrada $A_{n\times n}$ de inteiros e depois imprima a diferença entre o maior valor na diagonal principal e o menor valor da diagonal secundária. Um elemento $v[i][j]$ pertence à diagonal principal se e somente se $i = j$. Já um elemento $v[i][j]$ pertence à diagonal secundária se e somente se $i + j = n - 1$. Exemplos:
3, 7 4 2 9 6 1 5 3 9 **7** (resultado de 9-2)
3, 0 1 1 3 4 5 6 7 5 **4** (resultado de 5-1)
7. Escreva um programa que leia um inteiro b e um real x ($x > 1$, para o logaritmo ser positivo) e imprima o logaritmo base b de x. Use o algoritmo das médias geométricas para chegar neste valor. Considere que se $a^b = c$ então $\log_a c = b$. Eis a descrição do algoritmo. Primeiro, há que se estabelecer limites para a atuação do algoritmo. Começa-se com $t = 1$ e daí segue-se enquanto $x > b^t, t++$. Ao final deste ciclo, o intervalo de busca é composto por $t - 1$ e t. Calcula-se a média m entre estes dois valores e se $b^m > x$ então a média substitui o segundo valor, senão o primeiro. Retorna-se daqui para o cálculo da média. Quando o erro cometido $|b^m - x| < 0.001$ o algoritmo imprime m e acaba.
8. Imagine uma matriz retangular de m ($3 \leq m \leq 10$) linhas por n colunas ($3 \leq n \leq 12$). Agora imagine os 8 vizinhos dos elementos que não estão na borda da matriz. Seu programa C++ deve ler m, n e os elementos de $A_{m\times n}$ e depois totalizar os valores do vizinhos e apontar qual a célula $(linha, coluna)$ cujo total de vizinhos é maior. Se houver empate mostre qualquer um dos empatantes.

```
#include<iostream>
#include<iomanip>
#include<string>
#include<cmath>
using namespace std;
//---------------------- takedrop -----------------------
void takedrop(int & n, int k, int v[]){
   int i;
   if (k<0){
      i=0;
      while(i+abs(k)<=n){
         v[i]=v[i+abs(k)];
         i++;
      }
      n=i-1;
   }
   else {
      n=k;
   }
}
/*
int main(){
   int n,i;
   int v[10]={1,2,3,4,5,6};
   n=6;
   takedrop(n,1,v);
   for (i=0;i<n;i++){cout<<v[i]<<" ";}
}  */
//---------------------- conversao de base ---------------------
int converte(int n, int b, int v[]){
   int res=0;
   int i;
   for(i=n-1;i>=0;i--){
      res=res+v[i]*pow(b,(n-(i+1)));
   }
   return res;
} /*
int main(){
   int i,x,n,b;
   int v[10];
   cin>>n>>b;
```

```
   for (i=0;i<n;i++){cin>>v[i];}
   x=converte(n,b,v);
   cout<<x;
} */
//--------------------- arredondamento -----------------
void arredond(int n, float va[], int vd[]){
   int i;
   for (i=0;i<n;i++){
      if (va[i]==ceil(va[i])){vd[i]=va[i];}
      else{
         vd[i]=trunc(va[i]+0.5);
      }
   }
} /*
int main(){
   float a[10]={0.4, 5, 1.9, 3, 5, 7.99, 8.01};
   int i,b[10];
   arredond(7,a,b);
   for (i=0;i<7;i++){cout<<b[i]<<" ";}
} */
//----- vetor em ordem crescente e estritamente crescente -----
int emord(int n, int v[]){
  int rec=2,rc=1,i;
  for(i=1;i<n;i++){
     if (v[i]<v[i-1]){rec=rc=0;}
     if (v[i]==v[i-1]){rec=0;}
  }
  if (rec!=0){return rec;}
  else{
     if(rc!=0){return rc;}
  }
  return 0;
} /*
int main(){
   int n,i,v[10];
   cin>>n;
   for (i=0;i<n;i++){cin>>v[i];}
   cout<<emord(n,v);
} */
//--------- triângulo inclue a origem ? -----------------
float areatri(float x1, float y1, float x2, \
              float y2, float x3, float y3){
   float a;
   a=(x1*y2)+(y1*x3)+(x2*y3);
   a=a-((x3*y2)+(x1*y3)+(x2*y1));
   a=a/2;
   return a;
} /*
int main(){
  float a,b,c,d,e,f;
  cin>>a>>b>>c>>d>>e>>f;
  float area1,area2;
  area1=abs(areatri(a,b,c,d,e,f));
  area2=abs(areatri(0,0,c,d,e,f));
  area2=area2+abs(areatri(a,b,0,0,e,f));
  area2=area2+abs(areatri(a,b,c,d,0,0));
  if (area1==area2){cout<<1;}
  else{cout<<0;}
} */
/*
//-------- maior na d.p. menos menor na d.s. -------------------
int main(){
  int n,i,j;
```

```
  int m[10][10];
  cin>>n;
  for(i=0;i<n;i++){for(j=0;j<n;j++){cin>>m[i][j];}}
  int maxdp=-999999;
  int minds=+999999;
  for (i=0;i<n;i++){
     if (m[i][i]>maxdp){maxdp=m[i][i];}
     if (m[i][(n-1)-i]<minds){minds=m[i][(n-1)-i];}
  }
  cout<<maxdp-minds;
}
*/  /*
//-------------- logaritmo na marra ----------------
int main(){
   int t,b,lixo;
   float m,x,z[2];
   cin>>b>>x;
   if (x<1){cout<<"erro"; return 1;}
   t=1;
   while(x>pow(b,t)){t++;}
   z[0]=t-1; z[1]=t;
   while (1==1){
     if (z[1]==z[0]){cout<<"nao foi possivel"; return 1;}
     m=(z[0]+z[1])/2.0;
     if(0.0001>abs((pow(b,m)-x))) {cout<<m;return 0;}
     if (pow(b,m)>x){z[1]=m;}
     else {z[0]=m;}
   }
}  */
//---------- maiores vizinhos ---------------------
int main(){
   int m,n,i,j,maxv=-99999,maxi,maxj,s;
   int a[10][12];
   cin>>m>>n;
   for(i=0;i<m;i++){for(j=0;j<n;j++){cin>>a[i][j];}}
   for(i=1;i<m-1;i++){
      for(j=1;j<n-1;j++){
        s=a[i-1][j-1]+a[i-1][j]+a[i-1][j+1];
        s=s+a[i][j-1]+a[i][j+1];
        s=s+a[i+1][j-1]+a[i+1][j]+a[i+1][j+1];
        if (s>maxv){
            maxv=s;
            maxi=i;
            maxj=j;
        }
      }
   }
   cout<<maxi<<" "<<maxj;
}
```

Capítulo 12

Provas de 2022 - UFPR

CI208 - Programação de Computadores — 2ª Chamada

Nome: _____ 1º semestre 2022

Instruções para a prova.

- **Duração da prova:** 1 hora e 40 minutos;
- A prova é SEM CONSULTA;
- Esta folha de enunciados deverá ser entregue ao professor junto com a folha de respostas;
- Nos exemplos de execução de programas, a saída para a tela emitida pelo programa está em *itálico* e a entrada do usuário está representada em **negrito**.
- **Os programas devem se comportar como indicado no enunciado e nos exemplos, independentemente dos dados digitados, obtendo entradas e mostrando mensagens e resultados NA MESMA SEQUÊNCIA E DA MESMA FORMA (incluindo os textos das mensagens).**

Questão 1 (50 pontos)

Faça um programa em C++ que leia dois inteiros h e m tais que $0 \leqslant h < 24$ e $0 \leqslant m < 60$ que representam o horário (h horas e m minutos) do início de uma determinada tarefa a ser realizada. Em seguida, seu programa deve solicitar ao usuário um número inteiro n que representa a quantidade de minutos necessários para a realização dessa tarefa. Por fim, o programa deve imprimir na tela qual o horário de término da tarefa realizada e uma mensagem indicando se a tarefa foi curta (duração de menos de 2 horas) ou longa.

Exemplo de execução:

```
horario de inicio:  09 32
duracao da tarefa:  45
horario de termino: 10:17 hrs
tarefa curta
```

Outro exemplo de execução:

```
horario de início:  21 0
duracao da tarefa:  350
horario de termino: 02:50 hrs
tarefa longa
```

Questão 2 (50 pontos)

Você está visitando os pontos turísticos de Manhattan (Nova Iorque), onde as ruas são numeradas de 1 a 155 no sentido leste/oeste (eixo x), e 1 a 16 no sentido norte sul (eixo y). Por conveniência, os endereços das atratividades são dados pelos cruzamentos das ruas (exemplo: o Empire State Building fica no cruzamento da rua 5 com a rua 33). Crie um programa que leia um cruzamento de partida no formato $eixoX_1 \quad eixoY_1$, e depois leia sucessivos cruzamentos de ruas em que você fez paradas para visitar algum ponto turístico, também no formato $eixoX_2 \quad eixoY_2$. A cada cruzamento de parada exiba a distância percorrida desde a última parada, e também a distância total percorrida até o momento (ambas medidas em quadras). Em Manhattan, a distância entre dois cruzamentos de rua é dada por $|eixoX_2 - eixoX_1| + |eixoY_2 - eixoY_1|$ onde esta distância é medida em quadras. Não se esqueça de validar se o usuário digitou uma coordenada de rua válida. O programa deve terminar quando o usuário digitar -1 -1 para as coordenadas de rua.

Exemplo de execução:

```
Informe coordenadas:
- cruzamento de partida:  5 4
- proximo cruzamento:  5 5
Deslocamento: 1 quadras
Deslocamento total: 1 quadras
- proximo cruzamento:  10 8
Deslocamento: 8 quadras
Deslocamento total: 9 quadras
- proximo cruzamento:  9 7
Deslocamento: 2 quadras
Deslocamento total: 11 quadras
- proximo cruzamento:  156 3
Cruzamento invalido!
- proximo cruzamento:  155 3
Deslocamento: 150 quadras
Deslocamento total: 161 quadras
- proximo cruzamento:  -1 -1
Fim!
```

```
int main(){
  int h,m,d,duracao=0;
  cin>>h>>m>>d;
  cout<<"Horario de inicio "<<h<<" "<<m<<endl;
  cout<<"Duracao "<<d<<endl;
  if (d>=120){duracao=1;}
  while (d>0){
    if(d>59){h++;d=d-60;}
    else{m=m+d;d=0;}
  }
  if (m>60){h=h+1;m=m-60;}
  if (h>23){h=h-24;}
  cout<<"Horario de fim "<<h<<":"<<m<<" hrs"<<endl;
  if (duracao==1){cout<<"Tarefa longa";}
  else{cout<<"Tarefa curta";}
}
```

```
int main(){
   int xi,yi,xa,ya,dp,dt=0;
   cout<<"Informe coordenadas:"<<endl;
   cout<<" - cruzamento de partida ";
   cin>>xi>>yi;
   while (1==1){
      cout<<"proximo cruzamento ";
      cin>>xa>>ya;
      if ((xa>155)||(ya>16)){
         cout<<"cruzamento invalido!"<<endl;
         continue;
      }
      if ((xa==-1)&&(ya==-1)){cout<<"fim"<<endl; break;}
      dp=abs(xa-xi)+abs(ya-yi);
      cout<<"Deslocamento "<<dp<<endl;;
      dt=dt+dp;
      cout<<"Deslocamento total "<<dt<<endl;;
      xi=xa;
      yi=ya;
   }
}
```

CI208 - Programação de Computadores — 2ª Chamada

Nome: _____ 1º semestre 2022

Instruções para a prova.

- **Duração da prova:** 1 hora e 40 minutos;
- A prova é SEM CONSULTA;
- Esta folha de enunciados deverá ser entregue ao professor junto com a folha de respostas;
- Nos exemplos de execução de programas, a saída para a tela emitida pelo programa está em *itálico* e a entrada do usuário está representada em **negrito**.
- **Os programas devem se comportar como indicado no enunciado e nos exemplos, independentemente dos dados digitados, obtendo entradas e mostrando mensagens e resultados NA MESMA SEQUÊNCIA E DA MESMA FORMA (incluindo os textos das mensagens).**

Questão 1 (50 pontos)

Escrever um programa em C++ que leia do teclado um vetor de N elementos inteiros (N estabelecido via `#define`), calcule a média aritmética inteira truncada simples entre os elementos com o menor e maior conteúdo no vetor e altere os elementos com valores inferiores à media obtida para -1 e os acima daquela para 1. O vetor alterado deverá ser mostrado na tela.

OBS.: Para a mostra do vetor no monitor de vídeo, seu programa deverá utilizar a função de protótipo `void mostrar_vetor(int [])`, que não necessita ser codificada, apenas chamada.

Exemplo de execução (com N=11):

```
Vetor (11 elementos):
0 0 11 0 0 0 0 2 0 0 0
Média (min, max) = (0 + 11) / 2 = 5
Vetor alterado:
-1 -1 1 -1 -1 -1 -1 -1 -1 -1 -1
```

Exemplo de execução (com N=5):

```
Vetor (5 elementos):
5 1 2 3 4
Média (min, max) = (1 + 5) / 2 = 3
Vetor alterado:
1 -1 -1 3 1
```

Questão 2 (50 pontos)

Faça um programa em C++ que lê uma matriz quadrada de ordem N (N fixo, estabelecido via `#define`) de valores inteiros. Em seguida o programa deve chamar uma função de nome `mediaDiag()`, que recebe uma matriz quadrada de ordem N e calcula a média aritmética dos elementos das diagonais. A função deve retornar este valor ao programa principal que a sua vez deverá imprimir este valor.

OBS.: Para a leitura de uma matriz, considere que já existe a função `void ler_matriz (int matriz[][N])`. Sua solução deve apenas CHAMAR esta função.

Exemplo de execução (com N=4):

```
Informe matriz:
1 3 5 8
3 8 9 7
4 4 5 6
2 9 1 3
Media = 5.0
```

(Calculo: ((1+8+5+3) + (8+9+4+2))/8, não mostrado ao usuário)

```
int main(){
   int v[N];
   cout<<"Vetor "<<N<<" elementos:"<<endl;
   int i,j,med=0,maior=-99999,menor=99999;
   for (i=0;i<N;i++){cin>>v[i];}
   for (i=0;i<N;i++){
      if (v[i]>maior){maior=v[i];}
      if (v[i]<menor){menor=v[i];}
   }
   med=(maior+menor)/2;
   cout<<"Media (min,max) = ("<<menor<<"+"<<maior<<")/2="<<med<<endl;
   for (i=0;i<N;i++){
      if (v[i]<med){v[i]=-1;}
      if (v[i]>med){v[i]=1;}
   }
   cout<<"Vetor alterado "<<endl;
   for (i=0;i<N;i++){
       cout<<v[i]<<" ";
   }
}
*/
float mediadiag(int mat[N][N]){
   int i,j;
   float med=0;
   for (i=0;i<N;i++){
      med=med+mat[i][i];
      med=med+mat[i][(N-1)-i];
   }
   return med/(N*2);
}
int main(){
   int mat[N][N]={1,3,5,8,3,8,9,7,4,4,5,6,2,9,1,3};
   cout<<"Media "<<mediadiag(mat);
}
```

CI208 - Programação de Computadores — 2ª Chamada

Nome: _____ 1º semestre 2022

Instruções para a prova.

- **Duração da prova:** 1 hora e 40 minutos;
- A prova é SEM CONSULTA;
- Esta folha de enunciados deverá ser entregue ao professor junto com a folha de respostas;
- Nos exemplos de execução de programas, a saída para a tela emitida pelo programa está em *itálico* e a entrada do usuário está representada em **negrito**.
- **Os programas devem se comportar como indicado no enunciado e nos exemplos, independentemente dos dados digitados, obtendo entradas e mostrando mensagens e resultados NA MESMA SEQUÊNCIA E DA MESMA FORMA (incluindo os textos das mensagens).**

Questão 1 (50 pontos)

Faça um programa em C++ para ler 2 números naturais de 2 algarismos e verificar se eles são similares. Para serem similares estes números precisam ter os mesmos algarismos na sua formação.

Exemplo de execução:

```
Digite 2 números:  23 32
Números similares
```

Outro exemplo de execução:

```
Digite 2 números:  45 35
Números não similares
```

Outro exemplo de execução:

```
Digite 2 números:  67 67
Números similares
```

Questão 2 (50 pontos)

Faça um programa em C++ que receba do usuário, via teclado, um intervalo de números definido por dois limites, um inferior e outro superior, valide-o (o limite inferior não pode ser maior que o superior) e, a seguir, uma lista de números de quantidade desconhecida, encerrada pela informação do valor zero pelo usuário; este valor não faz parte da lista, apenas serve para indicar seu fim. O programa deverá ler toda a lista e, ao seu final, mostrar quantos números digitados foram menores que o limite inferior, quantos ficaram entre os limites e quantos acima do limite superior definido.

Exemplo de execução:

```
Limites (inferior e superior):  20 10
Limites invalidos!
```

Outro exemplo de execução:

```
Limites (inferior e superior):  -3.14159 12
Lista de números ('0' = Fim):
[1]:  25
[2]:  3.14159
[3]:  -3
[4]:  12
[5]:  -10
[6]:  0
Nos. abaixo de -3.14159: 1
Nos. de -3.14159 a 12: 3
Nos. acima de 12: 1
```

```
int main(){
   int n,m,d1n,d2n,d1m,d2m;
   cin>>n>>m;
   d1n=n/10;
   d2n=n%10;
   d1m=m/10;
   d2m=m%10;
   if (((d1n==d1m)||(d1n==d2m))&&((d2n==d1m)||(d2n==d2m))){
      cout<<"numeros similares";
   }
   else {
      cout<<"numeros nao similares";
   }
}
```

```
int main(){
   float li,ls,n;
   int qi=0,qm=0,qs=0;
   cin>>li>>ls;
   if (li>ls){cout<<"Limites invalidos";return 0;}
   cin>>n;
   while (n!=0.0){
      if (n<li){qi++;}
      if ((n>=li)&&(n<=ls)){qm++;}
      if (n>ls){qs++;}
      cin>>n;
   }
   cout<<"Numeros abaixo de "<<li<<" "<<qi<<endl;
   cout<<"Numeros entre "<<li<<" e "<<ls<<" "<<qm<<endl;
   cout<<"Numeros acima de "<<ls<<" "<<qs<<endl;
   }
```

CI208 - Programação de Computadores 2ª Chamada

Nome: 1º semestre 2022

Instruções para a prova.

- **Duração da prova:** 1 hora e 40 minutos;
- A prova é SEM CONSULTA;
- Esta folha de enunciados deverá ser entregue ao professor junto com a folha de respostas;
- Nos exemplos de execução de programas, a saída para a tela emitida pelo programa está em *itálico* e a entrada do usuário está representada em **negrito**.
- **Os programas devem se comportar como indicado no enunciado e nos exemplos, independentemente dos dados digitados, obtendo entradas e mostrando mensagens e resultados NA MESMA SEQUÊNCIA E DA MESMA FORMA (incluindo os textos das mensagens).**

Questão 1 (50 pontos)

Faça um programa que leia um vetor V de inteiros com um número fixo N de elementos indicado via `#define`, e rotacione seus elementos para a direita, fazendo com que o valor de V_{i+1} receba o valor de V_i . Cuidado com as extremidades, o primeiro elemento deve receber o último elemento do vetor. Além disso, o programa deve imprimir o somatório de todos os elementos rotacionados.

OBS.: Considere que a função `void imprime_vetor(int vet[])` para mostrar o conteúdo de um vetor JÁ EXISTE. Sua solução deverá apenas CHAMAR esta função onde necessário.

Exemplo de execução (com N=6):

```
Entre vetor:
7  2  3  -1  9  0
Vetor rotacionado:
0  7  2  3  -1  9
Soma dos elementos: 20
```

Questão 2 (50 pontos)

Escrever um programa em C++ que leia uma matriz de número reais de ordem $M \times N$ (M e N estabelecidos via *#define*, sendo $N > 2$), verifique e mostre as linhas nas quais os dois primeiros valores da linha e qualquer dos valores restantes da mesma linha podem formar um triângulo. Indicar os casos de linhas que não permitam a formação de triângulo algum. Para a fomação de um triângulo, a soma de dois lados quaisquer deve ser maior que o terceiro.

OBS.: Para a leitura de uma matriz, considere que já existe a função `void ler_matriz (float matriz[][N])`. Sua solução deve apenas CHAMAR esta função.

Exemplo de execução (com M=5, N=4):

```
Indique a matriz:
3 4 5 8
0 0 0 0
1 2 3 4
4 3 2 3
1 1 1 3
Resultado:
Linha 0 colunas 0 e 1 com 2
Linha 1 não permite formar triângulos
Linha 2 não permite formar triângulos
Linha 3 colunas 0 e 1 com 2 e 3
Linha 4 colunas 0 e 1 com 2
```

```
int main(){
   int v[N];
   int i,j;
   for (i=0;i<N;i++){
      cin>>v[i];
   }
   j=v[N-1];
   for (i=N-2;i>=0;i--){
      v[i+1]=v[i];
   }
   v[0]=j;
   for (i=0;i<N;i++){
      cout<<v[i]<<" ";
      }
}

int tria(int a, int b, int c){
   if ((a<b+c)&&(b<a+c)&&(c<a+b)){return 1;}
   else {return 0;}
}
int main(){
   int mat[M][N]={3,4,5,8,0,0,0,0,1,2,3,4,4,3,2,3,1,1,1,3};
   int i,j,naopode;
   for (i=0;i<M;i++){
      for(j=0;j<N;j++){
         cout<<mat[i][j];
     }
     cout<<endl;
   }
   for (i=0;i<M;i++){
      naopode=1;
      for (j=2;j<N;j++){
         if (tria(mat[i][j],mat[i][0],mat[i][1])){naopode=0;}
      }
      if (naopode==1){cout<<"linha "<<i<<" nao permite"<<endl;}
      else {
         cout<<"linha "<<i<<" colunas 0 e 1 com ";
         for (j=2;j<N;j++){
            if (tria(mat[i][j],mat[i][0],mat[i][1])){cout<<" "<<j<<" e";}
         }
         cout<<"\b \b \n";
      }
    }
}
```

Capítulo 13

Questões baseadas no Euler

13.1 Euler11: Maior multiplicação em grade

Dada uma matriz de números, e dado um número qualquer dentro dela, definem-se 8 direções de vizinhanças: norte, nordeste, este, sudeste, sul, sudoeste, oeste, noroeste. Para cada número define-se 3 vizinhos que formam até 8 conjuntos de 4 números cada. O próprio número mais os 3 vizinhos. Define-se ainda o produto destes 4 números. Pede-se achar o maior produto na grade mostrada

```
#include<iostream>
#include<iomanip>
#include<math.h>
using namespace std;
void p1(){
  int m[10][10]={32, 54, 47, 6, 23, 72, 36, 30, 46, 96,
  53, 52, 39, 72, 55, 84, 57, 29, 77, 51,
  35, 62, 53, 11, 75, 21, 31, 57, 35, 71,
  21, 40, 30, 6, 67, 38, 2, 92, 2, 20,
  53, 37, 33, 92, 31,4, 70, 18, 27, 54,
  43, 68, 43, 40, 43, 17, 75, 59, 58, 58,
  24, 68, 28, 34, 65, 92, 39, 78, 38, 91,
  13, 35, 2, 79, 70, 56, 95, 28, 15, 49,
  11, 34, 80, 40, 26, 31, 37, 18, 29, 86,
  53, 8,  34, 75, 68, 97, 71, 76, 15, 45};
  int i,j,k;
  int maxi,entao=-99999;
  int n[16][16];
  for (i=0;i<16;i++){
    for (j=0;j<16;j++){
      n[i][j]=1;
    }
  }
  for (i=3;i<13;i++){
    for (j=3;j<13;j++){
      n[i][j]=m[i-3][j-3];
    }
  }
  int p[8];
  for (i=3;i<13;i++){
    for (j=3;j<13;j++){
      p[0]=n[i][j]*n[i-1][j]*n[i-2][j]*n[i-3][j]; //norte
      p[1]=n[i][j]*n[i-1][j+1]*n[i-2][j+2]*n[i-3][j+3]; //nordeste
      p[2]=n[i][j]*n[i][j+1]*n[i][j+2]*n[i][j+3]; //leste
      p[3]=n[i][j]*n[i+1][j+1]*n[i+2][j+2]*n[i+3][j+3]; //sudeste
      p[4]=n[i][j]*n[i+1][j]*n[i+2][j]*n[i+3][j]; //sul
      p[5]=n[i][j]*n[i+1][j-1]*n[i+2][j-2]*n[i+3][j-3]; //sudoeste
      p[6]=n[i][j]*n[i][j-1]*n[i][j-2]*n[i][j-3]; //oeste
      p[7]=n[i][j]*n[i-1][j-1]*n[i-2][j-2]*n[i-3][j-3]; //noroeste
      maxi=-99999;
```

```
      for (k=0;k<8;k++){
        if (p[k]>maxi){
          maxi=p[k];
        }
      }
      if (maxi>entao){
        entao=maxi;
      }
    }
  }
  cout<<entao;
}
int main(){
  p1();     // a resposta correta é 35592016
            // resultado de 68*97*71*76 na última linha
}
```

13.2 Euler 12: Número triangular divisível

A sequência de números triangulares é gerada pela adição de números naturais. Assim, o sétimo número triangular é $1 + 2 + 3 + 4 + 5 + 6 + 7 = 28$. Os primeiros 10 números triangulares são $1, 3, 6, 10, 15, 21, 28, 36, 45, 55, ...$ Vão-se listar agora os fatores dos primeiros 7 números triangulares:

```
1:1
3:1,3
6:1,2,3,6
10:1,2,5,10
15:1,3,5,15
21:1,3,7,21
28:1,2,4,7,14,28
```

Percebe-se nesta lista que 28 é o primeiro triangular a ter mais do que 5 divisores. Qual o valor do primeiro número triangular a ter mais do que 350 fatores ?

```
int di(int x){
  int q=2,n=2;
  while (n<=sqrt(x)){
    if (x%n==0){q=q+2;}
    if (x*x==n){q--;}
    n++;
  }
  return q;
}
void p2(){
    int candi=1,tr=1;
    while(1==1){
      candi++;
      tr=tr+candi;
      int dd=di(tr);
      if(dd>350){
        cout<<tr;
        break;
      }
    }
}
int main(){
  p2();   # o resultado foi 17907120
}
```

13.3 Euler 14: Sequência mais longa

A seguinte sequência interativa é definida para todos os inteiros positivos: $n \to n/2$(quando n é par) e $n \to 3n + 1$ (quando n é impar). Usando esta regra e começando com 13, gera-se a seguinte sequência $13 \to 40 \to 20 \to 10 \to 5 \to$

$16 \to 8 \to 4 \to 2 \to 1$ Pode-se ver que esta sequência (começando em 13 e terminando em 1), contém 10 termos. Ainda que não tenha sido provado, espera-se que todas as sequências terminem em 1. Qual número inicial, abaixo de N, produz a maior sequência em comprimento? Note que depois que a cadeia começa, os termos podem passar de N. $N = 600000$

```
int cola(long long int x){
  int q=0;
    if (x%2==0){x=x/2;}
    else {x=(3*x)+1;}
    q++;
  }
  return q;
}
void p3(){
  long long int n=600000;
  int maxi=-999999;
  int quem,dd;
  while (n>1){
    dd=cola(n);
    if(dd>maxi){
        maxi=dd;
        quem=n;
    }
    n--;
  }
  cout<<quem;
}
int main(){
  p3();  // a resposta é 511935
}
```

13.4 Euler 18: Máximo caminho com somas

Começando no topo de um triângulo e movendo apenas para números adjacentes na linha de baixo, o valor máximo da soma é 23 ($3 + 7 + 4 + 9 = 23$)

```
   3
  7 4
 2 4 6
8 5 9 3
```

Ache o máximo da soma começando no topo e terminando na última linha

```
  32
  72 35
  27 97 70
  10 28 55 39
  99 17 20 81 31
  81  5 81 23 44 11
  43 88 27 13 11 41  5
  14 78 99 75 30  4 71 53
  40 21 11 31 15 72 10 46  6
  56 29 68 11 80 54 23 91 85 47
  94 66 72 71 69 88 72  9 66 26 58
  47 69 97 77 96 42 53 40 61 93 82 40
  97 66 47 21 76 21 87 54 67 44 92 99 26

  int maxizao;
  int pi[13][13]={32, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
             72,35, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
             27,97,70, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
             10,28,55,39, 0, 0, 0, 0, 0, 0, 0, 0, 0,
             99,17,20,81,31, 0, 0, 0, 0, 0, 0, 0, 0,
             81, 5,81,23,44,11, 0, 0, 0, 0, 0, 0, 0,
             43,88,27,13,11,41, 5, 0, 0, 0, 0, 0, 0,
```

```
             14,78,99,75,30, 4,71,53, 0, 0, 0, 0, 0,
             40,21,11,31,15,72,10,46, 6, 0, 0, 0, 0,
             56,29,68,11,80,54,23,91,85,47, 0, 0, 0,
             94,66,72,71,69,88,72, 9,66,26,58, 0, 0,
             47,69,97,77,96,42,53,40,61,93,82,40, 0,
             97,66,47,21,76,21,87,54,67,44,92,99,26};
void eure18(int a, int l, int c){
   if(l==13){
      if(a>maxizao){maxizao=a;}
      return;
   }
   eure18(a+pi[l][c],l+1,c);
   eure18(a+pi[l][c],l+1,c+1);
   return;
}
int main(){
  maxizao=-99999;
  eure18(0,0,0);
  cout<<maxizao;   // solucao recursiva, resposta = 881
}
```

Uma curiosidade: estas 4 questões foram programadas em C++ e em Python. O programa C++ demorou 0.482 segundos. O mesmo programa Python demorou 34.689 segundos.

Cartela de C++

Um programa C++ (baskara)

```
#include<iostream>
#include<cmath>
using namespace std;
int main(){
  float a,b,c,delta,x1,x2;
  cout<<"Informe coeficientes ";
  cin>>a>>b>>c;
  delta=(b*b)-4*a*c;
  if (delta>=0) {
    x1=((-b)+sqrt(delta))/(2*a);
    x2=((-b)+sqrt(delta))/(2*a);
    cout<<" x1="<<x1<<;
    cout<<"     x2="<<x2<<endl;
  }
  else {
    cout<<"imaginarias "<<endl;
  }
}
```

Processo

1. Escrever o código fonte (texto plano) usando um editor de texto. Sugestão: editor Codeblocks. Seu nome: aaaa.cpp
2. Compilar o código fonte.
3. Se houver erros de síntaxe, o compilador avisará. Corrigir e retornar.
4. Se a compilação funcionar, o compilador gerará o módulo aaaa.exe
5. Executar o aaaa.exe. CRITICAR o resultado. Havendo erros, voltar.
6. sucesso !

Comandos de pré-compilador

Todos começam com o caracter `#` e aparecem na cor verde.

`#include<iostream>` Carrega o módulo que contém os comandos `cin` e `cout`.

`#include<cmath>` Carrega o módulo matemático.

`#define` Permite definir constantes no programa. Por exemplo

```
#define PI 3.1415
```

Note que o comando `#define` cria uma equivalência que é integralmente substituída ANTES de começar a compilação. Assim, se você definir `#define EULER 2.7182`, cada vez que mencionar `EULER` no código, esta menção será substituída pelo valor `2.7182` antes de começar a compilação. A palavra EULER vai desaparecer do código compilado sem deixar rastro.

Cabeçalho

```
#include{iostream}
using namespace std;
```

a declaração using namespace std significa que os nomes de `cin` e `cout` não serão compostos. É uma maneira de deixar o código "limpo".

Comentário

Textos livres colocados dentro do código para serem vistos por olhos humanos.

```
\\ autor P. Kantek jul/87
/* programa com características
que fogem da trivialidade   */
int main() {  // função principal
```

Note que na última linha acima, há uma parte que não é comentário (`int main() {`) e depois há um comentário (`// função principal`).

Entrada

O comando `cin` faz diversas coisas:

1. interrompe o programa
2. espera o operador digitar algum conteúdo (se encerra por enter ou por um espaço em branco)
3. transfere o que foi digitado para a variável citada ao lado do `cin`
4. prossegue o programa na próxima instrução após o `cin`.

Note que a operação será conduzida na entrada padrão do programa, por definição o teclado do computador onde o programa está rodando.

Veja-se o exemplo

```
int a,b;
cin>>a>>b;
```

As variáveis `a` e `b` serão preenchidas com o que for digitado (nessa ordem).

Leitura de caracteres

Para ler uma frase (mais de uma palavra), você precisa

```
#include<iostream>
#include<string>
   int fim;
... string frase;
// cin>>frase;
// para no primeiro espaço
  getline(cin, frase);
  fim=frase.length()-1;
```

Saída

A saída ocorre quando se usa o comando `cout`. Note a inversão das flechas em relação ao `cin`. Aqui não há interrupção do programa. Note que se for a última instrução do programa, deve-se providenciar uma interrupção proposital a fim de que o operador do programa possa ler o que foi impresso. Para isso inclua um `system ("Pause");` antes do return 0;. Providencie um `#include <cstdlib>` no início do código.

A saída é feita na saida padrão que usualmente é o monitor de vídeo principal do computador onde o programa está rodando. Veja-se um exemplo

```
int x,y=0;
x=99;
cout<<"O valor eh "<<x;
```

Durante a operação de `cout`, o cursor fica parado ao final do conteúdo exibido. Para comandar um salto de linha, deve-se associar ao conteúdo a constante `endl`. Veja no exemplo:

```
cout<<a<<" "<<b;
cout<<a<<" "<<b<<endl;
```

No primeiro caso, o cursor para ao lado de `b`. No segundo, na linha de baixo. Quando duas variáveis são impressas (como em `cout<<a<<b`) seus dois valores são apresentados sem nenhuma separação entre eles. Então, no exemplo se vale 23 e b vale 7, a apresentação será 237. Para evitar este problema acostume-se a imprimir espaços entre as variáveis, como em `cout<<a<<" "<<b`.

Definição de variáveis

Toda variável precisa ser definida ANTES de ser mencionada pelo programa. Por essa razão costuma-se começar um programa pelas definições de variáveis que serão usadas. Toda variável tem um nome, um tipo e eventualmente um tamanho (que muitas vezes é pré-determinado).

Nomes:

*. única palavra (espaços em branco não são permitidos)
*. letras, números e o caracter sublinha (_)
*. começa obrigatoriamente por uma letra
*. a caixa faz diferença, assim Oba ≠ OBA ≠ oba ≠ oBa ...
*. não se devem usar caracteres acentuados
*. o tamanho do nome fica a critério do programador

Tipos

Alguns tipos possíveis:

int variáveis inteiras, com sinal e com a característica da contabilidade.

float variáveis reais. São armazenadas na forma $\pm mantissa \times 2^{\pm expoente}$. A mantissa sempre descreve um número entre 0 e 1, enquanto o expoente desloca livremente o ponto decimal por qualquer parte do número. Deve-se notar que nas proximidades de 0, os saltos que a variável admite são pequenos, enquanto que nos limites superior e inferior do expoente, o salto é muito maior.

char um único caracter cujo valor deve ser escrito entre aspas simples

string um conjunto de caracteres cujo valor deve estar escrito entre aspas duplas

bool um valor booleano. Na prática é tratado como uma variável inteira e significa FALSO quando ela vale zero e VERDADEIRO quando vale qualquer valor diferente de zero.

tipo	tam.	faixa
int	4	±2.147.483.647
float	4	±1.5E-45 a 3.4E38
double	8	±5E-324 a 1.7E308
char	1	-128 a 127

long e double

São modificadores do tamanho para variáveis numéricas. long é para aumentar o tipo int. Nas versões mais modernas, usa-se long long. double é para aumentar as variáveis tipo float. Acompanhe alguns exemplos

```
int a, alfa, residuo;
float x1, raiz;
char a = 'v';
string nome = "Curitiba / Paraná";
```

Uma variável PRECISA ser inicializada ANTES de ser usada. Você não pode supor que uma variável numérica recém definida conterá, por exemplo, zero. Ela pode conter qualquer coisa. Por essa razão, costuma-se inicializar todas as variáveis de uma de duas maneiras: pela sua leitura ou pela colocação do elemento neutro na operação fundamental que a variável vai sofrer, via operação de atribuição. Então um contador recebe zero, um produto recebe 1, um avaliador de maior recebe $-\infty$, um avaliador de menor recebe $+\infty$ e assim por diante.

Operações aritméticas

`+` adição. Por exemplo `a=b+c;`

`-` subtração entre duas variáveis, p. ex: `a=b-c;`. Quando usado junto a uma única variável, este símbolo significa oposto, comandando a troca do sinal do operando. P. ex: `x=-y;`

`*` multiplicação, por exemplo `x=x*y;`.

`/` divisão inteira quando os dois operandos e o resultado são definidos como inteiros.

`/` divisão real quando o resultado é definido como float. Se for uma expressão solta, o resultado dependerá do tipo dos operandos envolvidos.

`%` resto da divisão, disponível quando todos os envolvidos forem inteiros. Se não forem ocorre um erro.

Exemplos:

```
2/3;  // resultado é 0
2.0/3;  // resultado é 0.666666
int a=2, b=3;
a/b;   //  vale 0
float a=2, b=3;
a/b;   //  vale 0.666666
int f=10, g=7;
f%g;   // vale 3
float j=10, k=7;
j%k;   // erro sintático
```

Note que a linguagem C++ (assim como todas as demais linguagens de programação) tem uma tabela ou uma regra de prioridade na interpretação de operações aritméticas. Por exemplo `2*1+3` em C++ é 5, mas em APL é 8. Em resumo, a tal precedência em C++ é: 1. parênteses, 2. oposto (menos unário), 3. multiplicação, divisão e resto e 4. adição e subtração. Note que a maior precedência é dada pelos parênteses (em TODAS as linguagens), então uma regra bem simples, é: na dúvida use e abuse dos parênteses.

Conversões

Quando tipos diferentes estão envolvidos em uma expressão o compilador tenta adaptar o resultado antes de efetuar a atribuição, acompanhe

int =	float	o resultado é truncado no ponto decimal
float =	int	o resultado recebe .0
qualquer =	mistura de tipos	o tipo do resultado é o de maior "prioridade": double → float → int → char

Eventualmente, para poder realizar operações aritméticas como `sqrt(x)`, `sin(x)`, `log(x)`, `exp(x)` `log10(x)`, `round(x)`, `pow(x,y)`, `abs(x)` entre muitas outras será necessário incluir a biblioteca `cmath`, acompanhe

```
#include<cmath>
```

Operadores relacionais

`==` Igualdade. Se `a` e `b` tiverem o mesmo valor, `a==b` devolve 1 (verdadeiro).

`!=` Desigualdade. Se `a` e `b` tiverem o mesmo valor, `a!=b` devolve 0 (falso).

`>` Maior. Se `a=2` e `b=3`, `b>a` retorna 1 (verdadeiro)

`<` Menor. Se `a=2` e `b=3`, `a<b` retorna 1 (verdadeiro)

`>=` Maior ou igual.

`<=` Menor ou igual.

Conectivos lógicos

Conectam expressões relacionais. São eles

`&&` Conhecido como AND, e na matemática com o símbolo ∧ conecta duas expressões relacionais, e devolve verdadeiro se e somente se as duas expressões envolvidas forem verdadeiras.

`||` Conhecido como OR, e na matemática com o símbolo ∨ conecta duas expressões relacionais e devolve verdadeiro se qualquer uma das expressões (ou as duas) forem verdadeiras.

not Único conectivo unário, só se aplica a uma expressão lógica negando seu valor.

Veja alguns exemplos

Suponha a condição de aprovação nesta disciplina: Você estará aprovado ao final do semestre se tiver nota maior ou igual a 5 e tiver frequencia maior que 75%. Um programa testaria isso assim

```
float nota, freq;
if ((nota>=5) && (freq>75)) { ...
```

Neste caso, se houver a necessidade de negar a condição de APROVADO para REPROVADO o comando seria

```
if ((nota<5) || (freq<=75)) { ...
```

Note a negação de cada condição, e a mudança do conectivo de AND (`&&`) para OR (`||`). Isto se deve ao teorema de De Morgan.

Condicional

É o comando que permite estabelecer caminhos alternativos a depender das condições lógicas. O comando é

```
if (condição) {
   ... bloco1 ...
}
[else {
   ... bloco2 ...
}]
```

A condição (simples ou composta, tanto faz) é avaliada. Note que obrigatoriamente ela é escrita dentro de parênteses. Se a condição for verdadeira, o bloco imediatamente a seguir (no exemplo, o bloco1) é executado. A condição negada (a razão da palavra `else`) não é obrigatória, razão pela qual está escrita entre colchetes. Mas, se presente

o bloco2 só é executado se a condição original for falsa.

Em resumo no exemplo

```
if (a>7) {
   x=x+1;
}
else {
   z=z+1;
}
```

Se $a > 7$ o comando $x = x + 1$ será executado e $z = z + 1$ será ignorado. Se $a \leq 7$, $x = x + 1$ não será executado e $z = z + 1$ será executado.

Condicionais compostas Nada impede que dentro de uma condicional haja outra condicional. Veja

```
if (a<7) {
   if (b>5) {
      h++;   // se a<7 && b>5 }
   j++;  // se a<7 }
else {
   k++;   // se a>=7 }
```

Perceba a importância da indentação e sobretudo da correta disposição das chaves dos blocos.

repetição Este comando serve para repetir blocos de processamento. Seu formato

```
while (condição) {
   ... bloco1 ...
}
```

Se a condição for verdadeira o bloco1 é executado. Até aqui é igual ao comando `if (condição)`. A diferença vem agora. Quando o bloco1 acabar, ocorre um desvio até a palavra `while` e a condição é reavaliada. Veja o exemplo

```
int a=0;
while (a<10) {
   cout<<a;
   a=a+3;
}
cout<<"acabou";
```

O bloco será executado 4 vezes (serão impressos os valores 0, 3, 6 e 9) e depois a mensagem "acabou"será impressa.

Um especial cuidado deve ser tomado de maneira a garantir que dentro do bloco haja situações em que a condição deixa de ser verdadeira. Caso contrário pode-se ter um laço infinito e o programa nunca deixará de executar o bloco. Veja um contraexemplo

```
int b = 8;
while (b<20) {
   b=b-2;  // note que NUNCA b>20...
}
```

break Um comando especial para sair incondicionalmente do bloco onde ele é emitido. Veja o exemplo

```
while (1==1){
   a=a+1;
   if (a>10) {
      break;
 }  }
```

Se dependesse apenas da condição `1==1` o bloco nunca deixaria de ser executado. Mas perceba que dentro do bloco há uma condição (`a>10`) e quando ela for verdadeira o bloco deixará de ser executado via o comando `break`.

Perceba que se a condição original do `while` for falsa já na entrada do bloco ele não será executado NENHUMA vez.

do while Nos casos em que o bloco deve ser executado pelo menos uma vez, usa-se um comando parecido, mas ligeiramente diferente

```
do {
   ... bloco1 ...
} while (condição)
```

Agora ao entrar no comando `do` o bloco1 é executado. Ao final do bloco, a condição é avaliada. Se ela for verdadeira há um retorno ao início do bloco. Se falsa, ocorre a saída.

for Embora tudo o que for necessário repetir poderá ser feito com `while`, há um outro comando muito usado, chamado `for`. Seu formato

```
for(inicialização;saída;alteração){
   ... bloco1 ...
}
```

Antes de entrar no bloco1, a inicialização é executada. A seguir a condição de saída é verificada. Se ela for verdadeira, o bloco é executado. (Se a condição for falsa, o comando todo é abandonado). Ao final do bloco a condição alteração é executada e há um desvio para o início do bloco1 (antes a condição de saída é verificada novamente). Veja o exemplo:

```
int a;
for (a=0;a<8;a=a+3){
  cout<<a<<" ";  // impressos 0,3,6
}
```

Note-se que qualquer uma das 3 especificações do for pode ser omitida sem que ocorra erro sintático. Quanto ao erro semântico fica por conta do programador...

Funções Um programa fonte em C++ é um conjunto de uma ou mais funções. A única obrigatória é a função de nome `main()` que recebe o controle quando o programa que a contém é chamado. A função é reconhecida em um programa C++ pela presença do abre e fecha parêntesis ao lado do nome.

O formato da função main é

```
int main() {
... função principal...
   return [valor]
}
```

O comando `return` indica o final da função e o seu parâmetro, se presente sinaliza qual o valor será retornado a quem chamou este programa.

return Por convenção, aqui, se tem: retorno de 0 (zero) significa um final bem sucedido, enquanto qualquer retorno diferente de zero indicará algum tipo de problema e o valor retornado pode ser usado para sinalizar qual o tipo de problema que ocorreu.

Você pode encerrar um programa (ou a sua função principal) simplesmente emitindo o comando `return` sem nenhum parâmetro. Neste caso, o cabeçalho da função main deve ser

```
void main() {...
```

sendo que aqui, a palavra `void` sinaliza nehum retorno.

Outra maneira de encerrar uma função é simplesmente esgotar os comandos que devem ser executados. Quando a } do bloco principal for encontrado, a função se encerra.

outras funções Além da principal, o autor pode definir e criar tantas funções quantas desejar. A única regra aqui é que a função deve ser definida antes de ser executada, para que o compilador conheça os parâmetros e o retorno dela.

Acompanhe a definição de uma função chamada `dobro` que retorna o dobro do valor passado a ela.

```
int dobro(int x){
   return x*2;
}
```

Quando a função for chamada com 10, ela retornará 20. Quando chamada com -1, retornará -2, e assim por diante.

A partir do momento em que a função foi definida, ela pode ser usada em qualquer ponto do programa, por exemplo, dentro de expressões aritméticas.

No exemplo acima, a variável `x` não existe exatamente, fora da função. Ela apenas sinaliza o que fazer com o parâmetro da função quando ela for chamada. Assim por exemplo

```
int a=8;
cout<<dobro(a);
```

Quando a função `dobro` é chamada, o `x` do seu cabeçalho tem o valor de `a` (neste caso 8) atribuído a ele e a devolução é de `x*2` ou no caso `8*2` que é 16. Este é o valor impresso.

passagem por valor Quando uma função é chamada por valor (como exemplificado acima) é uma cópia do valor do parâmetro que é passada à função. Assim, não importa o que a função fizer com esse parâmetro, ele manterá seu valor intocado na função chamada. Acompanhe

```
int triplo(int x){
   x=x*3;
   return x;
}
int a=10;
cout<<triplo(a);
cout<<a;
}``
```

perceba que dentro da função `triplo` o parâmetro `x` tem seu valor alterado. Mas, independente disso, na primeira impressão obtem-se 30 (o triplo de 10), mas na segunda impressão obtem-se 10 e não 30, pois a variável `a` foi preservada.

passagem por referência Pode-se alterar a passagem de parâmetro, indicando a passagem por referência. Neste caso não é uma cópia e sim a própria variável que é passada para a função. Acompanhe

```
int triplo(int & x){
   x=x*3;
   return x;
}
int a=10;
cout<<triplo(a);
cout<<a;
}``
```

Agora, as duas impressões são do valor 30. Pois, dentro da função triplo, é a própria variável `a` que está sendo manuseada, embora com o nome de `x`.

A passagem por referência é indicada pela presença de um & na definição dos parâmetros.

Vetor Um arranjo homogêneo de variáveis múltiplas que serão acessadas pelo mesmo nome e por um índice de referência.

Suponha precisar as taxas de inflação dos últimos 10 anos. Você poderá fazer

```
float infl[10];
```

Perceba um contador de ocorrências escrito ao lado do nome da variável, sempre entre colchetes.

O acesso pode ser feito com valores numéricos dentro do índice (dentro dos colchetes). Assim `infl[5]` corresponderá ao SEXTO valor do vetor `infl`. Um exemplo

```
int x[8];
int i;
for (i=0;i<8;i++){
   x[i]=i;  // valores 0,1,2,3,4,5,6,7
}
```

Matrizes É a extensão do conceito de vetor (que tem 1 dimensão) para um arranjo contendo duas dimensões. Então, se um vetor tem um comprimento, uma matriz tem largura e altura. Veja-se um exemplo

```
#include<iostream>
#include<iomanip>
using namespace std;
int main(){
   int mm[5][7];
   int i,j;
   for (i=0;i<5;i++){
      for(j=0;j<7;j++){
         mm[i][j]=(i*7)+j;
      }
   }
   for (i=0;i<5;i++){
      for(j=0;j<7;j++){
         cout<<setw(4)<<mm[i][j];
      }
      cout<<endl;
   }
}
```

O resultado aqui foi

```
   0   1   2   3   4   5   6
   7   8   9  10  11  12  13
...
```

Perceba que na impressão da matriz há que se estabelecer um tamanho para as colunas a fim que os números sejam apresentados em linhas verticais independente de seus comprimentos individuais. Isso foi feito com

```
#include<iomanip>
...
cout<<setw(4)<<mm...
```

Quando um vetor ou matriz (ou array) é passado como parâmetro para uma função, ele sempre é passado por REFERÊNCIA, e neste caso NÃO é colocado o caractere & na chamada. Assim, guarde esta regra e lembre-se dela ao criar este tipo de parâmetro em funções.

Protótipos Há uma regra já enunciada que diz que antes de usar uma função ela tem que ser definida. Mas, agora vamos supor um caso possivelmente real: Suponha a função `a(...)` que usa dentro dela a função `b(...)`. E, mais ainda, suponha que a função `b()` usa a função `a()` dentro dela. Está criado o impasse (aqui chamado de *deadlock*). A maneira de fugir deste dilema, é pela utilização de protótipos. Eles são uma descrição formal de como uma função será chamada e quais resultados devolverá. Ao usar um protótipo, você ganha o direito de "adiar" a definição da função até isto ser mais conveniente a você.

O formato do protótipo é exatamente igual ao cabeçalho da função, só que terminado por ;

```
int funcaoa(int a, float n);
void funcaob();
```

C++ O nome C++ vem da abreviatura muito comum em C de trocar o comando `C=C+1` que é muito comum, pela expressão `C++` que tem exatamente o mesmo significado.

Exemplos

```
int primo(int x){
  int ate,diva,qtd;
  ate=1+sqrt(x);
  diva=2;
  if (x==2){return 1;}
  while(diva<=ate){
      if ((x%diva)==0){
        return 0; }
      diva++;  }
  return 1;
}
```

Seja a multiplicação matricial $C[i][m] = \sum_{x=1}^{j} A[i][x] \times B[x][m]$

```
int A[10][20]; int B[20][50]; int C[10][50];
   //... obtém A e B...
for (i=0;i<10;i++){
   for (m=0;m<50;m++){
      som=0;
      for (x=0;x<20;x++){
         som=som+A[i][x]*B[x][m];  }
      C[i][m]=som; }
}        // ...C está calculado
```