UFPR - UP - UTFPR - PUCPr – 24/04/2024 - 11:42:12.5
Prof Dr P Kantek (pkantek@gmail.com)
Aquecimento em programação VIVX001a, V: 2.06
75772 ANA JULIA SCHUENCK DOS SANTO T
24PRG101 - 1 _____ / _____ / _____

Programar é ... Já se disse com alguma razão que programar é emprestar um pouco de inteligência ao computador. Então, antes de começar, reconheça-se a nobreza do que se está a fazer. Os computadores (todos) descendem de duas propostas mais ou menos complementares: a primeira, descrita brilhantemente por A. Turing em 1936 e a segunda igualmente brilhante, por Von Neumann em começos da década de 50. Pela primeira, sabe-se que o computador é capaz de executar uma instrução após a outra. Tais instruções precisam ser simples e inequívocas. Pela segunda descobre-se que tais instruções são bastante limitadas: E/S, aritmética, movimentação, alternativa e desvio.

E/S Este tipo de instrução responde pela E=entrada e S=saída de dados do/para o computador. Suponha que você construiu uma máquina capaz de descobrir o seno de qualquer ângulo. Para pôr a máquina para funcionar, você precisa fornecer algum ângulo a ela e depois que ela fizer seu trabalho ela vai precisar informar a você o valor do seno. Estas duas etapas correspondem a operações de E/S. Do ponto de vista teórico da programação, são operações quase irrelevantes, embora do ponto de vista prático sejam fundamentais.

aritmética Esta família é muito importante, mas graças a Deus, já estudamos pelo menos 8 anos de matemática. Tudo o que se aprendeu lá (adição, produto, potência, paridade, regra de sinais, números inteiros e reais, positivos e negativos, propriedades associativa, comutativa, distributiva – prioridades e parêntesis, etc etc, etc, etc) continua válido.

movimentação Operação muito simples, pela qual, partes da memória, chamadas variáveis são movimentadas ou mais propriamente copiadas entre sí.

alternativa Aqui a instrução mais "nobre" da programação. É graças a ela que um computador se torna diferente de um liquidificador. Se uma condição for boa (verdadeira) executar-se-á alguma coisa. Se a condição for ruim (falsa) executar-se-á outra coisa, ou em certos casos não se executará nada. Vale aqui a lógica binária com seus 2 princípios:

- Não contradição: Uma condição é verdadeira ou falsa, não podendo ser as duas coisas ao mesmo tempo.
- Terceiro excluído: se não for falsa será verdadeira e vice-versa.

desvio Um programa de computador vai executar a primeira instrução, depois a segunda e assim por diante, até o final do programa, a menos que uma instrução seja de desvio: *vá para a instrução x* : Neste caso a seguir será executada a instrução x e não a próxima.

Na década de 70, alguns autores (Dijkstra, Corrado e Jacopini) notaram que o uso de desvios tendia a gerar programas difíceis de *debugar*: Este verbo novo (=depurar) é o processo de identificar, localizar e corrigir erros de programas. Na programação gasta-se 20% do tempo escrevendo código e 80% depurando-o. A programação usando desvios gerou a chamada programação *spaguetti*. Daí os autores sugeriram organizar os desvios: nascia a programação estruturada. Agora os comandos são agrupados em blocos, havendo 3 deles:

sequência 2 ou mais comandos em sequência

alternativa como acima descrito

repetição uma sequência que pode se repetir. Apenas aqui é permitido o desvio e sempre do fim do bloco para o começo.

Fora o caso acima (repetição) o comando de desvio é proibido. Isto é tão verdade que linguagens mais modernas não têm o desvio, como por exemplo a linguagem Python.

Outra característica bem importante da *arquitetura de Von Neumann* é a idéia de programa armazenado: instruções são armazenadas como dados, permitindo que programas processem programas. Esta ideia já havia sido posta por Turing. Nascia o conceito de linguagem de programação, pela qual as complexidades da programação de máquina são transferidas para um programa de computador: o compilador. Uma linguagem de programação é na verdade 2 coisas:

⋆ um conjunto de regras fonéticas, sintáticas e semânticas de como descrever um algoritmo e
⋆ um programa (chamado compilador ou interpretador) capaz de ler o algoritmo acima descrito e gerar o equivalente código de máquina capaz de ser executado em um computador real.

Nascia o $software$ em oposição ao $hardware$. Graças a este conceito, aprender a programar passou de um processo de meses/anos para um de horas/dias. Sem contar que agora o mesmo programa roda em um supercomputador de milhões de dólares e também em um celular android.

Algoritmo O conceito chave em programação é o de *algoritmo*. É o que está por trás de um programa. É a ideia, a receita, a lista dos procedimentos do programa. Se programar é arte e engenharia, o algoritmo responde integralmente pela parte "arte". O conceito nos acompanha faz tempo (o algoritmo de mdc, devido a Euclides tem mais de 2400 anos), mas sua definição precisa é devida ao matemático inglês Alan Turing. Para poder expressar um algoritmo, se necessita uma linguagem apropriada. Vamos descrever uma mini-linguagem agora: As peças elementares de informação serão chamadas variáveis. Cada uma terá um nome, uma palavra qualquer (como aliás já fizemos nas aulas de álgebra em qualquer dos mais de 8 anos de ensino fundamental de matemática).

As operações de E/S serão 2: leia(variável) ou imprima(variável), significando a entrada do valor de uma variável ou sua impressão.

Para designar um valor a uma variável usar-se-á o comando de atribuição, representando por $\leftarrow$, então, por exemplo $A \leftarrow 0$ significa colocar o valor zero na variável A.

As operações aritméticas, seguem 100% do aprendido nas aulas de matemática e são: +: adição, $*$: multiplicação, $-$: subtração, /: divisão real, //: divisão inteira, MOD: resto de, $**$: potência, além de prioridade determinada por parêntesis. Continua proibida a divisão por zero. As linguagens mais modernas já operam normalmente com números complexos, mas podemos deixar esta facilidade de lado (por enquanto).

Para o comando alternativo precisa-se definir uma condição lógica: uma expressão envolvendo condicionais ($>$, $<$, $\leq$, $\geq$, $=$, $\neq$, $\in$, ...) e cuja resposta será um de dois valores: VERDADEIRO ou FALSO. Estes, ainda podem ser combinados usando os conectivos: E ($\wedge$) e OU ($\vee$) além de NÃO ($\sim$). O comando alternativo tem o formato `se <condição> então bloco-A fim{se}` neste comando o bloco de comandos A só será executado se a condição citada for verdadeira. Um formato alternativo é `se <condição> então bloco-A senão bloco-B fim{se}` Agora, se a condição é verdadeira, A é executado (como acima), mas se é falso então o bloco-B é executado.

Finalmente, o desvio não existe, mas em seu lugar entram as estruturas de repetição. Só vamos ver 1 delas. Trata-se do comando `enquanto` que tem formato `enquanto <condição> bloco-A fim{enquanto}`. Note que ele é parecido com o comando alternativo, mas tem uma diferença fundamental: ao encontrar o comando `fim{enquanto}` há um desvio incondicional para o início do bloco ou o `enquanto`.

Um exemplo Vamos escrever um algoritmo para testar se o número X é primo ou não.

```
 1: algoritmo PRIMO
 2: leia(X)
 3: DIVISORES ← 0
 4: TESTADOR ← 2
 5: enquanto TESTADOR < X
 6:     se X MOD TESTADOR = 0
 7:         DIVISORES ← DIVISORES + 1
 8:     fim{se}
 9:     TESTADOR ← TESTADOR + 1
10: fim{enquanto}
11: se DIVISORES = 0
12:     escreva("PRIMO")
13: senão
14:     escreva("NÃO PRIMO")
15: fim{se}
```

Seguir um algoritmo, simulando suas operações é chamado teste de mesa (ou em inglês *walkthrough*), ou mais jocosamente *fazer um chinês*. É o feijão com arroz da programação, usado para entender, certificar e depurar algoritmos. Ao fazer isto com o algoritmo acima, perceba que ele funciona. Se chamado com 15 responderá NÃO PRIMO e se chamado com 13 responderá PRIMO. Programar é escrever estas "coisas". É como aprender um idioma novo. Mas antes de aprender a expressar idéias (algoritmos), há que se ter firmeza na fonética e na síntaxe. A semântica virá depois.

Mais um exemplo Alguns aqui devem conhecer a sequência de Fibonacci: 1,1,2,3,5,8,13,21,34,... Note que, nela, cada número é a soma dos dois anteriores. A sequência começa com 1,1 já que estes não tem dois anteriores. A partir do próximo (o 2), a regra genérica se aplica. Vamos estudar um algoritmo que leia um número n e imprima o n-ésimo número de Fibonacci.

```
 1: leia LIMITE
 2: A ← 1; B ← 1; N ← 3
 3: enquanto N ≤ LIMITE
 4:     R ← A + B
 5:     A ← B
 6:     B ← R
 7:     N ← N + 1
 8: fim{enquanto}
 9: escreva R
```

A seguir, alguns exercícios de síntaxe da linguagem:

☞ Para você fazer

Ex. 1

```
 1: X ← 10
 2: Y ← 37
 3: ALIN ← 3
 4: enquanto X < Y
 5:     X ← X + ALIN
 6:     se ((ALIN mod 3) ≠ 0)
 7:         ALIN ← ALIN + 2
 8:     fim{se}
 9:     ALIN ← ALIN + 1
10: fim{enquanto}
11: escreva X
```

Ex. 2

```
 1: A ← 3
 2: B ← 22
 3: enquanto A ≤ B
 4:     se ((B > 10) ∨ (A ≤ 10))
 5:         B ← B - 1
 6:     fim{se}
 7:     A ← A + 2
 8:     B ← B - 2
 9: fim{enquanto}
10: A ← A - 3
11: escreva (A + B)
```

Ex. 3

```
 1: X ← 5
 2: enquanto X < 11
 3:     Y ← 2
 4:     enquanto Y < 11
 5:         X ← X + 3
 6:         Y ← X + 1
 7:     fim{enquanto}
 8:     Y ← Y + 3
 9:     X ← X + 1
10: fim{enquanto}
11: escreva (X-Y)
```

Ex. 4

```
 1: X ← 1
 2: Y ← 4
 3: se  X < 5 ∧ Y > 3
 4:     X ← X - 3
 5: senão
 6:     Y ← Y + 2
 7:     se X mod 2 ≠ 0
 8:         Y ← Y + 1
 9:     fim{se}
10: fim{se}
11: escreva X+Y
```

Responda aqui os valores encontrados.

1	2	3	4

UFPR - UP - UTFPR - PUCPr – 24/04/2024 - 11:42:12.5
Prof Dr P Kantek (pkantek@gmail.com)
Aquecimento em programação VIVX001a, V: 2.06
75789 ARIEL GUSTAVO RODRIGUES DA S
24PRG101 - 2 _____ / _____ / _____

Programar é ... Já se disse com alguma razão que programar é emprestar um pouco de inteligência ao computador. Então, antes de começar, reconheça-se a nobreza do que se está a fazer. Os computadores (todos) descendem de duas propostas mais ou menos complementares: a primeira, descrita brilhantemente por A. Turing em 1936 e a segunda igualmente brilhante, por Von Neumann em começos da década de 50. Pela primeira, sabe-se que o computador é capaz de executar uma instrução após a outra. Tais instruções precisam ser simples e inequívocas. Pela segunda descobre-se que tais instruções são bastante limitadas: E/S, aritmética, movimentação, alternativa e desvio.

E/S Este tipo de instrução responde pela E=entrada e S=saída de dados do/para o computador. Suponha que você construiu uma máquina capaz de descobrir o seno de qualquer ângulo. Para pôr a máquina para funcionar, você precisa fornecer algum ângulo a ela e depois que ela fizer seu trabalho ela vai precisar informar a você o valor do seno. Estas duas etapas correspondem a operações de E/S. Do ponto de vista teórico da programação, são operações quase irrelevantes, embora do ponto de vista prático sejam fundamentais.

aritmética Esta família é muito importante, mas graças a Deus, já estudamos pelo menos 8 anos de matemática. Tudo o que se aprendeu lá (adição, produto, potência, paridade, regra de sinais, números inteiros e reais, positivos e negativos, propriedades associativa, comutativa, distributiva – prioridades e parêntesis, etc etc, etc, etc) continua válido.

movimentação Operação muito simples, pela qual, partes da memória, chamadas variáveis são movimentadas ou mais propriamente copiadas entre sí.

alternativa Aqui a instrução mais "nobre" da programação. É graças a ela que um computador se torna diferente de um liquidificador. Se uma condição for boa (verdadeira) executar-se-á alguma coisa. Se a condição for ruim (falsa) executar-se-á outra coisa, ou em certos casos não se executará nada. Vale aqui a lógica binária com seus 2 princípios:

- Não contradição: Uma condição é verdadeira ou falsa, não podendo ser as duas coisas ao mesmo tempo.
- Terceiro excluído: se não for falsa será verdadeira e vice-versa.

desvio Um programa de computador vai executar a primeira instrução, depois a segunda e assim por diante, até o final do programa, a menos que uma instrução seja de desvio: *vá para a instrução x* : Neste caso a seguir será executada a instrução x e não a próxima.

Na década de 70, alguns autores (Dijkstra, Corrado e Jacopini) notaram que o uso de desvios tendia a gerar programas difíceis de *debugar*: Este verbo novo (=depurar) é o processo de identificar, localizar e corrigir erros de programas. Na programação gasta-se 20% do tempo escrevendo código e 80% depurando-o. A programação usando desvios gerou a chamada programação *spaguetti*. Daí os autores sugeriram organizar os desvios: nascia a programação estruturada. Agora os comandos são agrupados em blocos, havendo 3 deles:

sequência 2 ou mais comandos em sequência

alternativa como acima descrito

repetição uma sequência que pode se repetir. Apenas aqui é permitido o desvio e sempre do fim do bloco para o começo.

Fora o caso acima (repetição) o comando de desvio é proibido. Isto é tão verdade que linguagens mais modernas não têm o desvio, como por exemplo a linguagem Python.

Outra característica bem importante da *arquitetura de Von Neumann* é a idéia de programa armazenado: instruções são armazenadas como dados, permitindo que programas processem programas. Esta ideia já havia sido posta por Turing. Nascia o conceito de linguagem de programação, pela qual as complexidades da programação de máquina são transferidas para um programa de computador: o compilador. Uma linguagem de programação é na verdade 2 coisas:

⋆ um conjunto de regras fonéticas, sintáticas e semânticas de como descrever um algoritmo e
⋆ um programa (chamado compilador ou interpretador) capaz de ler o algoritmo acima descrito e gerar o equivalente código de máquina capaz de ser executado em um computador real.

Nascia o $software$ em oposição ao $hardware$. Graças a este conceito, aprender a programar passou de um processo de meses/anos para um de horas/dias. Sem contar que agora o mesmo programa roda em um supercomputador de milhões de dólares e também em um celular android.

Algoritmo O conceito chave em programação é o de *algoritmo*. É o que está por trás de um programa. É a ideia, a receita, a lista dos procedimentos do programa. Se programar é arte e engenharia, o algoritmo responde integralmente pela parte "arte". O conceito nos acompanha faz tempo (o algoritmo de mdc, devido a Euclides tem mais de 2400 anos), mas sua definição precisa é devida ao matemático inglês Alan Turing. Para poder expressar um algoritmo, se necessita uma linguagem apropriada. Vamos descrever uma mini-linguagem agora: As peças elementares de informação serão chamadas variáveis. Cada uma terá um nome, uma palavra qualquer (como aliás já fizemos nas aulas de álgebra em qualquer dos mais de 8 anos de ensino fundamental de matemática).

As operações de E/S serão 2: leia(variável) ou imprima(variável), significando a entrada do valor de uma variável ou sua impressão.

Para designar um valor a uma variável usar-se-á o comando de atribuição, representando por $\leftarrow$, então, por exemplo $A \leftarrow 0$ significa colocar o valor zero na variável A.

As operações aritméticas, seguem 100% do aprendido nas aulas de matemática e são: $+$: adição, $*$: multiplicação, $-$: subtração, $/$: divisão real, $//$: divisão inteira, MOD: resto de, $**$: potência, além de prioridade determinada por parêntesis. Continua proibida a divisão por zero. As linguagens mais modernas já operam normalmente com números complexos, mas podemos deixar esta facilidade de lado (por enquanto).

Para o comando alternativo precisa-se definir uma condição lógica: uma expressão envolvendo condicionais ($>$, $<$, $\leq$, $\geq$, $=$, $\neq$, $\in$, ...) e cuja resposta será um de dois valores: VERDADEIRO ou FALSO. Estes, ainda podem ser combinados usando os conectivos: E ($\wedge$) e OU ($\vee$) além de NÃO ($\sim$). O comando alternativo tem o formato `se <condição> então bloco-A fim{se}` neste comando o bloco de comandos A só será executado se a condição citada for verdadeira. Um formato alternativo é `se <condição> então bloco-A senão bloco-B fim{se}` Agora, se a condição é verdadeira, A é executado (como acima), mas se é falso então o bloco-B é executado.

Finalmente, o desvio não existe, mas em seu lugar entram as estruturas de repetição. Só vamos ver 1 delas. Trata-se do comando `enquanto` que tem formato `enquanto <condição> bloco-A fim{enquanto}`. Note que ele é parecido com o comando alternativo, mas tem uma diferença fundamental: ao encontrar o comando `fim{enquanto}` há um desvio incondicional para o início do bloco ou o `enquanto`.

Um exemplo Vamos escrever um algoritmo para testar se o número X é primo ou não.

```
 1: algoritmo PRIMO
 2: leia(X)
 3: DIVISORES ← 0
 4: TESTADOR ← 2
 5: enquanto TESTADOR < X
 6:     se X MOD TESTADOR = 0
 7:         DIVISORES ← DIVISORES + 1
 8:     fim{se}
 9:     TESTADOR ← TESTADOR + 1
10: fim{enquanto}
11: se DIVISORES = 0
12:     escreva("PRIMO")
13: senão
14:     escreva("NÃO PRIMO")
15: fim{se}
```

Seguir um algoritmo, simulando suas operações é chamado teste de mesa (ou em inglês *walkthrough*), ou mais jocosamente *fazer um chinês*. É o feijão com arroz da programação, usado para entender, certificar e depurar algoritmos. Ao fazer isto com o algoritmo acima, perceba que ele funciona. Se chamado com 15 responderá NÃO PRIMO e se chamado com 13 responderá PRIMO. Programar é escrever estas "coisas". É como aprender um idioma novo. Mas antes de aprender a expressar idéias (algoritmos), há que se ter firmeza na fonética e na síntaxe. A semântica virá depois.

Mais um exemplo Alguns aqui devem conhecer a sequência de Fibonacci: 1,1,2,3,5,8,13,21,34,... Note que, nela, cada número é a soma dos dois anteriores. A sequência começa com 1,1 já que estes não tem dois anteriores. A partir do próximo (o 2), a regra genérica se aplica. Vamos estudar um algoritmo que leia um número n e imprima o n-ésimo número de Fibonacci.

```
1: leia LIMITE
2: A ← 1; B ← 1; N ← 3
3: enquanto N ≤ LIMITE
4:     R ← A + B
5:     A ← B
6:     B ← R
7:     N ← N + 1
8: fim{enquanto}
9: escreva R
```

A seguir, alguns exercícios de síntaxe da linguagem:

☞ Para você fazer

Ex. 1

```
 1: X ← 10
 2: Y ← 32
 3: CTDS ← 3
 4: enquanto X < Y
 5:     X ← X + CTDS
 6:     se ((CTDS mod 3) ≠ 0)
 7:         CTDS ← CTDS + 1
 8:     fim{se}
 9:     CTDS ← CTDS + 1
10: fim{enquanto}
11: escreva X
```

Ex. 2

```
 1: A ← 1
 2: B ← 23
 3: enquanto A ≤ B
 4:     se ((B > 8) ∨ (A ≤ 14))
 5:         B ← B - 2
 6:     fim{se}
 7:     A ← A + 3
 8:     B ← B - 1
 9: fim{enquanto}
10: A ← A - 2
11: escreva (A + B)
```

Ex. 3

```
 1: X ← 1
 2: enquanto X < 9
 3:     Y ← 4
 4:     enquanto Y < 13
 5:         X ← X + 1
 6:         Y ← X + 2
 7:     fim{enquanto}
 8:     Y ← Y + 2
 9:     X ← X + 1
10: fim{enquanto}
11: escreva (X-Y)
```

Ex. 4

```
 1: X ← 1
 2: Y ← 7
 3: se  X < 5 ∧ Y > 4
 4:     X ← X - 1
 5: senão
 6:     Y ← Y + 3
 7:     se X mod 2 ≠ 0
 8:         Y ← Y + 3
 9:     fim{se}
10: fim{se}
11: escreva X+Y
```

Responda aqui os valores encontrados.

1	2	3	4

UFPR - UP - UTFPR - PUCPr – 24/04/2024 - 11:42:12.5
Prof Dr P Kantek (pkantek@gmail.com)
Aquecimento em programação VIVX001a, V: 2.06
75796 CASSIANO MAGNO CHAGAS E SA
24PRG101 - 3 _____ / _____ / _____

Programar é ... Já se disse com alguma razão que programar é emprestar um pouco de inteligência ao computador. Então, antes de começar, reconheça-se a nobreza do que se está a fazer. Os computadores (todos) descendem de duas propostas mais ou menos complementares: a primeira, descrita brilhantemente por A. Turing em 1936 e a segunda igualmente brilhante, por Von Neumann em começos da década de 50. Pela primeira, sabe-se que o computador é capaz de executar uma instrução após a outra. Tais instruções precisam ser simples e inequívocas. Pela segunda descobre-se que tais instruções são bastante limitadas: E/S, aritmética, movimentação, alternativa e desvio.

E/S Este tipo de instrução responde pela E=entrada e S=saída de dados do/para o computador. Suponha que você construiu uma máquina capaz de descobrir o seno de qualquer ângulo. Para pôr a máquina para funcionar, você precisa fornecer algum ângulo a ela e depois que ela fizer seu trabalho ela vai precisar informar a você o valor do seno. Estas duas etapas correspondem a operações de E/S. Do ponto de vista teórico da programação, são operações quase irrelevantes, embora do ponto de vista prático sejam fundamentais.

aritmética Esta família é muito importante, mas graças a Deus, já estudamos pelo menos 8 anos de matemática. Tudo o que se aprendeu lá (adição, produto, potência, paridade, regra de sinais, números inteiros e reais, positivos e negativos, propriedades associativa, comutativa, distributiva – prioridades e parêntesis, etc etc, etc, etc) continua válido.

movimentação Operação muito simples, pela qual, partes da memória, chamadas variáveis são movimentadas ou mais propriamente copiadas entre sí.

alternativa Aqui a instrução mais "nobre" da programação. É graças a ela que um computador se torna diferente de um liquidificador. Se uma condição for boa (verdadeira) executar-se-á alguma coisa. Se a condição for ruim (falsa) executar-se-á outra coisa, ou em certos casos não se executará nada. Vale aqui a lógica binária com seus 2 princípios:

- Não contradição: Uma condição é verdadeira ou falsa, não podendo ser as duas coisas ao mesmo tempo.
- Terceiro excluído: se não for falsa será verdadeira e vice-versa.

desvio Um programa de computador vai executar a primeira instrução, depois a segunda e assim por diante, até o final do programa, a menos que uma instrução seja de desvio: *vá para a instrução x* : Neste caso a seguir será executada a instrução x e não a próxima.

Na década de 70, alguns autores (Dijkstra, Corrado e Jacopini) notaram que o uso de desvios tendia a gerar programas difíceis de *debugar*: Este verbo novo (=depurar) é o processo de identificar, localizar e corrigir erros de programas. Na programação gasta-se 20% do tempo escrevendo código e 80% depurando-o. A programação usando desvios gerou a chamada programação *spaguetti*. Daí os autores sugeriram organizar os desvios: nascia a programação estruturada. Agora os comandos são agrupados em blocos, havendo 3 deles:

sequência 2 ou mais comandos em sequência

alternativa como acima descrito

repetição uma sequência que pode se repetir. Apenas aqui é permitido o desvio e sempre do fim do bloco para o começo.

Fora o caso acima (repetição) o comando de desvio é proibido. Isto é tão verdade que linguagens mais modernas não têm o desvio, como por exemplo a linguagem Python.

Outra característica bem importante da *arquitetura de Von Neumann* é a idéia de programa armazenado: instruções são armazenadas como dados, permitindo que programas processem programas. Esta ideia já havia sido posta por Turing. Nascia o conceito de linguagem de programação, pela qual as complexidades da programação de máquina são transferidas para um programa de computador: o compilador. Uma linguagem de programação é na verdade 2 coisas:

⋆ um conjunto de regras fonéticas, sintáticas e semânticas de como descrever um algoritmo e
⋆ um programa (chamado compilador ou interpretador) capaz de ler o algoritmo acima descrito e gerar o equivalente código de máquina capaz de ser executado em um computador real.

Nascia o *software* em oposição ao *hardware*. Graças a este conceito, aprender a programar passou de um processo de meses/anos para um de horas/dias. Sem contar que agora o mesmo programa roda em um supercomputador de milhões de dólares e também em um celular android.

Algoritmo O conceito chave em programação é o de *algoritmo*. É o que está por trás de um programa. É a ideia, a receita, a lista dos procedimentos do programa. Se programar é arte e engenharia, o algoritmo responde integralmente pela parte "arte". O conceito nos acompanha faz tempo (o algoritmo de mdc, devido a Euclides tem mais de 2400 anos), mas sua definição precisa é devida ao matemático inglês Alan Turing. Para poder expressar um algoritmo, se necessita uma linguagem apropriada. Vamos descrever uma mini-linguagem agora: As peças elementares de informação serão chamadas variáveis. Cada uma terá um nome, uma palavra qualquer (como aliás já fizemos nas aulas de álgebra em qualquer dos mais de 8 anos de ensino fundamental de matemática).

As operações de E/S serão 2: leia(variável) ou imprima(variável), significando a entrada do valor de uma variável ou sua impressão.

Para designar um valor a uma variável usar-se-á o comando de atribuição, representando por $\leftarrow$, então, por exemplo $A \leftarrow 0$ significa colocar o valor zero na variável A.

As operações aritméticas, seguem 100% do aprendido nas aulas de matemática e são: +: adição, ∗: multiplicação, −: subtração, /: divisão real, //: divisão inteira, MOD: resto de, ∗∗: potência, além de prioridade determinada por parêntesis. Continua proibida a divisão por zero. As linguagens mais modernas já operam normalmente com números complexos, mas podemos deixar esta facilidade de lado (por enquanto).

Para o comando alternativo precisa-se definir uma condição lógica: uma expressão envolvendo condicionais ($>$, $<$, $\leq$, $\geq$, $=$, $\neq$, $\in$, ...) e cuja resposta será um de dois valores: VERDADEIRO ou FALSO. Estes, ainda podem ser combinados usando os conectivos: E ($\wedge$) e OU ($\vee$) além de NÃO ($\sim$). O comando alternativo tem o formato `se <condição> então bloco-A fim{se}` neste comando o bloco de comandos A só será executado se a condição citada for verdadeira. Um formato alternativo é `se <condição> então bloco-A senão bloco-B fim{se}` Agora, se a condição é verdadeira, A é executado (como acima), mas se é falso então o bloco-B é executado.

Finalmente, o desvio não existe, mas em seu lugar entram as estruturas de repetição. Só vamos ver 1 delas. Trata-se do comando `enquanto` que tem formato `enquanto <condição> bloco-A fim{enquanto}`. Note que ele é parecido com o comando alternativo, mas tem uma diferença fundamental: ao encontrar o comando `fim{enquanto}` há um desvio incondicional para o início do bloco ou o `enquanto`.

Um exemplo Vamos escrever um algoritmo para testar se o número X é primo ou não.

```
 1: algoritmo PRIMO
 2: leia(X)
 3: DIVISORES ← 0
 4: TESTADOR ← 2
 5: enquanto TESTADOR < X
 6:     se X MOD TESTADOR = 0
 7:         DIVISORES ← DIVISORES + 1
 8:     fim{se}
 9:     TESTADOR ← TESTADOR + 1
10: fim{enquanto}
11: se DIVISORES = 0
12:     escreva("PRIMO")
13: senão
14:     escreva("NÃO PRIMO")
15: fim{se}
```

Seguir um algoritmo, simulando suas operações é chamado teste de mesa (ou em inglês *walkthrough*), ou mais jocosamente *fazer um chinês*. É o feijão com arroz da programação, usado para entender, certificar e depurar algoritmos. Ao fazer isto com o algoritmo acima, perceba que ele funciona. Se chamado com 15 responderá NÃO PRIMO e se chamado com 13 responderá PRIMO. Programar é escrever estas "coisas". É como aprender um idioma novo. Mas antes de aprender a expressar idéias (algoritmos), há que se ter firmeza na fonética e na síntaxe. A semântica virá depois.

Mais um exemplo Alguns aqui devem conhecer a sequência de Fibonacci: 1,1,2,3,5,8,13,21,34,... Note que, nela, cada número é a soma dos dois anteriores. A sequência começa com 1,1 já que estes não tem dois anteriores. A partir do próximo (o 2), a regra genérica se aplica. Vamos estudar um algoritmo que leia um número n e imprima o n-ésimo número de Fibonacci.

```
1: leia LIMITE
2: A ← 1; B ← 1; N ← 3
3: enquanto N ≤ LIMITE
4:     R ← A + B
5:     A ← B
6:     B ← R
7:     N ← N + 1
8: fim{enquanto}
9: escreva R
```

A seguir, alguns exercícios de síntaxe da linguagem:

☞ Para você fazer

Ex. 1

```
 1: X ← 8
 2: Y ← 31
 3: BETA ← 3
 4: enquanto X < Y
 5:     X ← X + BETA
 6:     se ((BETA mod 2) ≠ 0)
 7:         BETA ← BETA + 2
 8:     fim{se}
 9:     BETA ← BETA + 1
10: fim{enquanto}
11: escreva X
```

Ex. 2

```
 1: A ← 2
 2: B ← 23
 3: enquanto A ≤ B
 4:     se ((B > 11) ∨ (A ≤ 9))
 5:         B ← B - 3
 6:     fim{se}
 7:     A ← A + 3
 8:     B ← B - 1
 9: fim{enquanto}
10: A ← A - 2
11: escreva (A + B)
```

Ex. 3

```
 1: X ← 2
 2: enquanto X < 8
 3:     Y ← 5
 4:     enquanto Y < 11
 5:         X ← X + 1
 6:         Y ← X + 2
 7:     fim{enquanto}
 8:     Y ← Y + 1
 9:     X ← X + 1
10: fim{enquanto}
11: escreva (X-Y)
```

Ex. 4

```
 1: X ← 6
 2: Y ← 1
 3: se  X < 4 ∧ Y > 5
 4:     X ← X - 1
 5: senão
 6:     Y ← Y + 3
 7:     se X mod 2 ≠ 0
 8:         Y ← Y + 1
 9:     fim{se}
10: fim{se}
11: escreva X+Y
```

Responda aqui os valores encontrados.

1	2	3	4

UFPR - UP - UTFPR - PUCPr – 24/04/2024 - 11:42:12.5
Prof Dr P Kantek (pkantek@gmail.com)
Aquecimento em programação VIVX001a, V: 2.06
75808 CHRISTOPHER BUCH FILIPAK
24PRG101 - 4 _____ / _____ / _____

Programar é ... Já se disse com alguma razão que programar é emprestar um pouco de inteligência ao computador. Então, antes de começar, reconheça-se a nobreza do que se está a fazer. Os computadores (todos) descendem de duas propostas mais ou menos complementares: a primeira, descrita brilhantemente por A. Turing em 1936 e a segunda igualmente brilhante, por Von Neumann em começos da década de 50. Pela primeira, sabe-se que o computador é capaz de executar uma instrução após a outra. Tais instruções precisam ser simples e inequívocas. Pela segunda descobre-se que tais instruções são bastante limitadas: E/S, aritmética, movimentação, alternativa e desvio.

E/S Este tipo de instrução responde pela E=entrada e S=saída de dados do/para o computador. Suponha que você construiu uma máquina capaz de descobrir o seno de qualquer ângulo. Para pôr a máquina para funcionar, você precisa fornecer algum ângulo a ela e depois que ela fizer seu trabalho ela vai precisar informar a você o valor do seno. Estas duas etapas correspondem a operações de E/S. Do ponto de vista teórico da programação, são operações quase irrelevantes, embora do ponto de vista prático sejam fundamentais.

aritmética Esta família é muito importante, mas graças a Deus, já estudamos pelo menos 8 anos de matemática. Tudo o que se aprendeu lá (adição, produto, potência, paridade, regra de sinais, números inteiros e reais, positivos e negativos, propriedades associativa, comutativa, distributiva – prioridades e parêntesis, etc etc, etc, etc) continua válido.

movimentação Operação muito simples, pela qual, partes da memória, chamadas variáveis são movimentadas ou mais propriamente copiadas entre sí.

alternativa Aqui a instrução mais "nobre" da programação. É graças a ela que um computador se torna diferente de um liquidificador. Se uma condição for boa (verdadeira) executar-se-á alguma coisa. Se a condição for ruim (falsa) executar-se-á outra coisa, ou em certos casos não se executará nada. Vale aqui a lógica binária com seus 2 princípios:

- Não contradição: Uma condição é verdadeira ou falsa, não podendo ser as duas coisas ao mesmo tempo.
- Terceiro excluído: se não for falsa será verdadeira e vice-versa.

desvio Um programa de computador vai executar a primeira instrução, depois a segunda e assim por diante, até o final do programa, a menos que uma instrução seja de desvio: *vá para a instrução x* : Neste caso a seguir será executada a instrução x e não a próxima.

Na década de 70, alguns autores (Dijkstra, Corrado e Jacopini) notaram que o uso de desvios tendia a gerar programas difíceis de *debugar*: Este verbo novo (=depurar) é o processo de identificar, localizar e corrigir erros de programas. Na programação gasta-se 20% do tempo escrevendo código e 80% depurando-o. A programação usando desvios gerou a chamada programação *spaguetti*. Daí os autores sugeriram organizar os desvios: nascia a programação estruturada. Agora os comandos são agrupados em blocos, havendo 3 deles:

sequência 2 ou mais comandos em sequência

alternativa como acima descrito

repetição uma sequência que pode se repetir. Apenas aqui é permitido o desvio e sempre do fim do bloco para o começo.

Fora o caso acima (repetição) o comando de desvio é proibido. Isto é tão verdade que linguagens mais modernas não têm o desvio, como por exemplo a linguagem Python.

Outra característica bem importante da *arquitetura de Von Neumann* é a idéia de programa armazenado: instruções são armazenadas como dados, permitindo que programas processem programas. Esta ideia já havia sido posta por Turing. Nascia o conceito de linguagem de programação, pela qual as complexidades da programação de máquina são transferidas para um programa de computador: o compilador. Uma linguagem de programação é na verdade 2 coisas:

⋆ um conjunto de regras fonéticas, sintáticas e semânticas de como descrever um algoritmo e
⋆ um programa (chamado compilador ou interpretador) capaz de ler o algoritmo acima descrito e gerar o equivalente código de máquina capaz de ser executado em um computador real.

Nascia o $software$ em oposição ao $hardware$. Graças a este conceito, aprender a programar passou de um processo de meses/anos para um de horas/dias. Sem contar que agora o mesmo programa roda em um supercomputador de milhões de dólares e também em um celular android.

Algoritmo O conceito chave em programação é o de *algoritmo*. É o que está por trás de um programa. É a ideia, a receita, a lista dos procedimentos do programa. Se programar é arte e engenharia, o algoritmo responde integralmente pela parte "arte". O conceito nos acompanha faz tempo (o algoritmo de mdc, devido a Euclides tem mais de 2400 anos), mas sua definição precisa é devida ao matemático inglês Alan Turing. Para poder expressar um algoritmo, se necessita uma linguagem apropriada. Vamos descrever uma mini-linguagem agora: As peças elementares de informação serão chamadas variáveis. Cada uma terá um nome, uma palavra qualquer (como aliás já fizemos nas aulas de álgebra em qualquer dos mais de 8 anos de ensino fundamental de matemática).

As operações de E/S serão 2: leia(variável) ou imprima(variável), significando a entrada do valor de uma variável ou sua impressão.

Para designar um valor a uma variável usar-se-á o comando de atribuição, representando por $\leftarrow$, então, por exemplo $A \leftarrow 0$ significa colocar o valor zero na variável A.

As operações aritméticas, seguem 100% do aprendido nas aulas de matemática e são: +: adição, $*$: multiplicação, $-$: subtração, /: divisão real, //: divisão inteira, MOD: resto de, $**$: potência, além de prioridade determinada por parêntesis. Continua proibida a divisão por zero. As linguagens mais modernas já operam normalmente com números complexos, mas podemos deixar esta facilidade de lado (por enquanto).

Para o comando alternativo precisa-se definir uma condição lógica: uma expressão envolvendo condicionais ($>$, $<$, $\leq$, $\geq$, $=$, $\neq$, $\in$, ...) e cuja resposta será um de dois valores: VERDADEIRO ou FALSO. Estes, ainda podem ser combinados usando os conectivos: E ($\wedge$) e OU ($\vee$) além de NÃO ($\sim$). O comando alternativo tem o formato `se <condição> então bloco-A fim{se}` neste comando o bloco de comandos A só será executado se a condição citada for verdadeira. Um formato alternativo é `se <condição> então bloco-A senão bloco-B fim{se}` Agora, se a condição é verdadeira, A é executado (como acima), mas se é falso então o bloco-B é executado.

Finalmente, o desvio não existe, mas em seu lugar entram as estruturas de repetição. Só vamos ver 1 delas. Trata-se do comando `enquanto` que tem formato `enquanto <condição> bloco-A fim{enquanto}`. Note que ele é parecido com o comando alternativo, mas tem uma diferença fundamental: ao encontrar o comando `fim{enquanto}` há um desvio incondicional para o início do bloco ou o `enquanto`.

Um exemplo Vamos escrever um algoritmo para testar se o número X é primo ou não.

```
 1: algoritmo PRIMO
 2: leia(X)
 3: DIVISORES ← 0
 4: TESTADOR ← 2
 5: enquanto TESTADOR < X
 6:     se X MOD TESTADOR = 0
 7:         DIVISORES ← DIVISORES + 1
 8:     fim{se}
 9:     TESTADOR ← TESTADOR + 1
10: fim{enquanto}
11: se DIVISORES = 0
12:     escreva("PRIMO")
13: senão
14:     escreva("NÃO PRIMO")
15: fim{se}
```

Seguir um algoritmo, simulando suas operações é chamado teste de mesa (ou em inglês *walkthrough*), ou mais jocosamente *fazer um chinês*. É o feijão com arroz da programação, usado para entender, certificar e depurar algoritmos. Ao fazer isto com o algoritmo acima, perceba que ele funciona. Se chamado com 15 responderá NÃO PRIMO e se chamado com 13 responderá PRIMO. Programar é escrever estas "coisas". É como aprender um idioma novo. Mas antes de aprender a expressar idéias (algoritmos), há que se ter firmeza na fonética e na síntaxe. A semântica virá depois.

Mais um exemplo Alguns aqui devem conhecer a sequência de Fibonacci: 1,1,2,3,5,8,13,21,34,... Note que, nela, cada número é a soma dos dois anteriores. A sequência começa com 1,1 já que estes não tem dois anteriores. A partir do próximo (o 2), a regra genérica se aplica. Vamos estudar um algoritmo que leia um número n e imprima o n-ésimo número de Fibonacci.

```
1: leia LIMITE
2: A ← 1; B ← 1; N ← 3
3: enquanto N ≤ LIMITE
4:     R ← A + B
5:     A ← B
6:     B ← R
7:     N ← N + 1
8: fim{enquanto}
9: escreva R
```

A seguir, alguns exercícios de síntaxe da linguagem:

☞ Para você fazer

Ex. 1

```
 1: X ← 8
 2: Y ← 38
 3: ALFA ← 4
 4: enquanto X < Y
 5:     X ← X + ALFA
 6:     se ((ALFA mod 2) ≠ 0)
 7:         ALFA ← ALFA + 2
 8:     fim{se}
 9:     ALFA ← ALFA + 1
10: fim{enquanto}
11: escreva X
```

Ex. 2

```
 1: A ← 3
 2: B ← 23
 3: enquanto A ≤ B
 4:     se ((B > 8) ∨ (A ≤ 14))
 5:         B ← B - 2
 6:     fim{se}
 7:     A ← A + 1
 8:     B ← B - 3
 9: fim{enquanto}
10: A ← A - 1
11: escreva (A + B)
```

Ex. 3

```
 1: X ← 1
 2: enquanto X < 6
 3:     Y ← 1
 4:     enquanto Y < 14
 5:         X ← X + 2
 6:         Y ← X + 2
 7:     fim{enquanto}
 8:     Y ← Y + 2
 9:     X ← X + 2
10: fim{enquanto}
11: escreva (X-Y)
```

Ex. 4

```
 1: X ← 6
 2: Y ← 7
 3: se  X < 6 ∧ Y > 5
 4:     X ← X - 1
 5: senão
 6:     Y ← Y + 2
 7:     se X mod 3 ≠ 0
 8:         Y ← Y + 2
 9:     fim{se}
10: fim{se}
11: escreva X+Y
```

Responda aqui os valores encontrados.

1	2	3	4

UFPR - UP - UTFPR - PUCPr – 24/04/2024 - 11:42:12.5
Prof Dr P Kantek (pkantek@gmail.com)
Aquecimento em programação VIVX001a, V: 2.06
75815 ERAN MARTINEZ RAMOS 24PRG101 - 5 / _____ / _____

Programar é ... Já se disse com alguma razão que programar é emprestar um pouco de inteligência ao computador. Então, antes de começar, reconheça-se a nobreza do que se está a fazer. Os computadores (todos) descendem de duas propostas mais ou menos complementares: a primeira, descrita brilhantemente por A. Turing em 1936 e a segunda igualmente brilhante, por Von Neumann em começos da década de 50. Pela primeira, sabe-se que o computador é capaz de executar uma instrução após a outra. Tais instruções precisam ser simples e inequívocas. Pela segunda descobre-se que tais instruções são bastante limitadas: E/S, aritmética, movimentação, alternativa e desvio.

E/S Este tipo de instrução responde pela E=entrada e S=saída de dados do/para o computador. Suponha que você construiu uma máquina capaz de descobrir o seno de qualquer ângulo. Para pôr a máquina para funcionar, você precisa fornecer algum ângulo a ela e depois que ela fizer seu trabalho ela vai precisar informar a você o valor do seno. Estas duas etapas correspondem a operações de E/S. Do ponto de vista teórico da programação, são operações quase irrelevantes, embora do ponto de vista prático sejam fundamentais.

aritmética Esta família é muito importante, mas graças a Deus, já estudamos pelo menos 8 anos de matemática. Tudo o que se aprendeu lá (adição, produto, potência, paridade, regra de sinais, números inteiros e reais, positivos e negativos, propriedades associativa, comutativa, distributiva – prioridades e parêntesis, etc etc, etc, etc) continua válido.

movimentação Operação muito simples, pela qual, partes da memória, chamadas variáveis são movimentadas ou mais propriamente copiadas entre sí.

alternativa Aqui a instrução mais "nobre" da programação. É graças a ela que um computador se torna diferente de um liquidificador. Se uma condição for boa (verdadeira) executar-se-á alguma coisa. Se a condição for ruim (falsa) executar-se-á outra coisa, ou em certos casos não se executará nada. Vale aqui a lógica binária com seus 2 princípios:

- Não contradição: Uma condição é verdadeira ou falsa, não podendo ser as duas coisas ao mesmo tempo.
- Terceiro excluído: se não for falsa será verdadeira e vice-versa.

desvio Um programa de computador vai executar a primeira instrução, depois a segunda e assim por diante, até o final do programa, a menos que uma instrução seja de desvio: *vá para a instrução x* : Neste caso a seguir será executada a instrução x e não a próxima.

Na década de 70, alguns autores (Dijkstra, Corrado e Jacopini) notaram que o uso de desvios tendia a gerar programas difíceis de *debugar*: Este verbo novo (=depurar) é o processo de identificar, localizar e corrigir erros de programas. Na programação gasta-se 20% do tempo escrevendo código e 80% depurando-o. A programação usando desvios gerou a chamada programação *spaguetti*. Daí os autores sugeriram organizar os desvios: nascia a programação estruturada. Agora os comandos são agrupados em blocos, havendo 3 deles:

sequência 2 ou mais comandos em sequência

alternativa como acima descrito

repetição uma sequência que pode se repetir. Apenas aqui é permitido o desvio e sempre do fim do bloco para o começo.

Fora o caso acima (repetição) o comando de desvio é proibido. Isto é tão verdade que linguagens mais modernas não têm o desvio, como por exemplo a linguagem Python.

Outra característica bem importante da *arquitetura de Von Neumann* é a idéia de programa armazenado: instruções são armazenadas como dados, permitindo que programas processem programas. Esta ideia já havia sido posta por Turing. Nascia o conceito de linguagem de programação, pela qual as complexidades da programação de máquina são transferidas para um programa de computador: o compilador. Uma linguagem de programação é na verdade 2 coisas:

⋆ um conjunto de regras fonéticas, sintáticas e semânticas de como descrever um algoritmo e
⋆ um programa (chamado compilador ou interpretador) capaz de ler o algoritmo acima descrito e gerar o equivalente código de máquina capaz de ser executado em um computador real.

Nascia o $software$ em oposição ao $hardware$. Graças a este conceito, aprender a programar passou de um processo de meses/anos para um de horas/dias. Sem contar que agora o mesmo programa roda em um supercomputador de milhões de dólares e também em um celular android.

Algoritmo O conceito chave em programação é o de *algoritmo*. É o que está por trás de um programa. É a ideia, a receita, a lista dos procedimentos do programa. Se programar é arte e engenharia, o algoritmo responde integralmente pela parte "arte". O conceito nos acompanha faz tempo (o algoritmo de mdc, devido a Euclides tem mais de 2400 anos), mas sua definição precisa é devida ao matemático inglês Alan Turing. Para poder expressar um algoritmo, se necessita uma linguagem apropriada. Vamos descrever uma mini-linguagem agora: As peças elementares de informação serão chamadas variáveis. Cada uma terá um nome, uma palavra qualquer (como aliás já fizemos nas aulas de álgebra em qualquer dos mais de 8 anos de ensino fundamental de matemática).

As operações de E/S serão 2: leia(variável) ou imprima(variável), significando a entrada do valor de uma variável ou sua impressão.

Para designar um valor a uma variável usar-se-á o comando de atribuição, representando por $\leftarrow$, então, por exemplo $A \leftarrow 0$ significa colocar o valor zero na variável A.

As operações aritméticas, seguem 100% do aprendido nas aulas de matemática e são: +: adição, $*$: multiplicação, $-$: subtração, /: divisão real, //: divisão inteira, MOD: resto de, $**$: potência, além de prioridade determinada por parêntesis. Continua proibida a divisão por zero. As linguagens mais modernas já operam normalmente com números complexos, mas podemos deixar esta facilidade de lado (por enquanto).

Para o comando alternativo precisa-se definir uma condição lógica: uma expressão envolvendo condicionais ($>$, $<$, $\leq$, $\geq$, $=$, $\neq$, $\in$, ...) e cuja resposta será um de dois valores: VERDADEIRO ou FALSO. Estes, ainda podem ser combinados usando os conectivos: E ($\wedge$) e OU ($\vee$) além de NÃO ($\sim$). O comando alternativo tem o formato `se <condição> então bloco-A fim{se}` neste comando o bloco de comandos A só será executado se a condição citada for verdadeira. Um formato alternativo é `se <condição> então bloco-A senão bloco-B fim{se}` Agora, se a condição é verdadeira, A é executado (como acima), mas se é falso então o bloco-B é executado.

Finalmente, o desvio não existe, mas em seu lugar entram as estruturas de repetição. Só vamos ver 1 delas. Trata-se do comando `enquanto` que tem formato `enquanto <condição> bloco-A fim{enquanto}`. Note que ele é parecido com o comando alternativo, mas tem uma diferença fundamental: ao encontrar o comando `fim{enquanto}` há um desvio incondicional para o início do bloco ou o `enquanto`.

Um exemplo Vamos escrever um algoritmo para testar se o número X é primo ou não.

```
 1: algoritmo PRIMO
 2: leia(X)
 3: DIVISORES ← 0
 4: TESTADOR ← 2
 5: enquanto TESTADOR < X
 6:     se X MOD TESTADOR = 0
 7:         DIVISORES ← DIVISORES + 1
 8:     fim{se}
 9:     TESTADOR ← TESTADOR + 1
10: fim{enquanto}
11: se DIVISORES = 0
12:     escreva("PRIMO")
13: senão
14:     escreva("NÃO PRIMO")
15: fim{se}
```

Seguir um algoritmo, simulando suas operações é chamado teste de mesa (ou em inglês *walkthrough*), ou mais jocosamente *fazer um chinês*. É o feijão com arroz da programação, usado para entender, certificar e depurar algoritmos. Ao fazer isto com o algoritmo acima, perceba que ele funciona. Se chamado com 15 responderá NÃO PRIMO e se chamado com 13 responderá PRIMO. Programar é escrever estas "coisas". É como aprender um idioma novo. Mas antes de aprender a expressar idéias (algoritmos), há que se ter firmeza na fonética e na síntaxe. A semântica virá depois.

Mais um exemplo Alguns aqui devem conhecer a sequência de Fibonacci: 1,1,2,3,5,8,13,21,34,... Note que, nela, cada número é a soma dos dois anteriores. A sequência começa com 1,1 já que estes não tem dois anteriores. A partir do próximo (o 2), a regra genérica se aplica. Vamos estudar um algoritmo que leia um número n e imprima o n-ésimo número de Fibonacci.

```
 1: leia LIMITE
 2: A ← 1; B ← 1; N ← 3
 3: enquanto N ≤ LIMITE
 4:     R ← A + B
 5:     A ← B
 6:     B ← R
 7:     N ← N + 1
 8: fim{enquanto}
 9: escreva R
```

A seguir, alguns exercícios de síntaxe da linguagem:

☞ Para você fazer

Ex. 1

```
 1: X ← 7
 2: Y ← 42
 3: GAMA ← 2
 4: enquanto X < Y
 5:     X ← X + GAMA
 6:     se ((GAMA mod 3) ≠ 0)
 7:         GAMA ← GAMA + 1
 8:     fim{se}
 9:     GAMA ← GAMA + 1
10: fim{enquanto}
11: escreva X
```

Ex. 2

```
 1: A ← 2
 2: B ← 21
 3: enquanto A ≤ B
 4:     se ((B > 6) ∨ (A ≤ 15))
 5:         B ← B - 1
 6:     fim{se}
 7:     A ← A + 3
 8:     B ← B - 3
 9: fim{enquanto}
10: A ← A - 2
11: escreva (A + B)
```

Ex. 3

```
 1: X ← 5
 2: enquanto X < 13
 3:     Y ← 7
 4:     enquanto Y < 14
 5:         X ← X + 3
 6:         Y ← X + 2
 7:     fim{enquanto}
 8:     Y ← Y + 2
 9:     X ← X + 2
10: fim{enquanto}
11: escreva (X-Y)
```

Ex. 4

```
 1: X ← 4
 2: Y ← 2
 3: se  X < 4 ∧ Y > 5
 4:     X ← X - 3
 5: senão
 6:     Y ← Y + 2
 7:     se X mod 2 ≠ 0
 8:         Y ← Y + 3
 9:     fim{se}
10: fim{se}
11: escreva X+Y
```

Responda aqui os valores encontrados.

1	2	3	4

101-75815 - /

UFPR - UP - UTFPR - PUCPr – 24/04/2024 - 11:42:12.5
Prof Dr P Kantek (pkantek@gmail.com)
Aquecimento em programação VIVX001a, V: 2.06
75822 HENRIQUE IVANKIO GUSSE 24PRG101 - 6 _____ / _____ / _____

Programar é ... Já se disse com alguma razão que programar é emprestar um pouco de inteligência ao computador. Então, antes de começar, reconheça-se a nobreza do que se está a fazer. Os computadores (todos) descendem de duas propostas mais ou menos complementares: a primeira, descrita brilhantemente por A. Turing em 1936 e a segunda igualmente brilhante, por Von Neumann em começos da década de 50. Pela primeira, sabe-se que o computador é capaz de executar uma instrução após a outra. Tais instruções precisam ser simples e inequívocas. Pela segunda descobre-se que tais instruções são bastante limitadas: E/S, aritmética, movimentação, alternativa e desvio.

E/S Este tipo de instrução responde pela E=entrada e S=saída de dados do/para o computador. Suponha que você construiu uma máquina capaz de descobrir o seno de qualquer ângulo. Para pôr a máquina para funcionar, você precisa fornecer algum ângulo a ela e depois que ela fizer seu trabalho ela vai precisar informar a você o valor do seno. Estas duas etapas correspondem a operações de E/S. Do ponto de vista teórico da programação, são operações quase irrelevantes, embora do ponto de vista prático sejam fundamentais.

aritmética Esta família é muito importante, mas graças a Deus, já estudamos pelo menos 8 anos de matemática. Tudo o que se aprendeu lá (adição, produto, potência, paridade, regra de sinais, números inteiros e reais, positivos e negativos, propriedades associativa, comutativa, distributiva – prioridades e parêntesis, etc etc, etc, etc) continua válido.

movimentação Operação muito simples, pela qual, partes da memória, chamadas variáveis são movimentadas ou mais propriamente copiadas entre sí.

alternativa Aqui a instrução mais "nobre" da programação. É graças a ela que um computador se torna diferente de um liquidificador. Se uma condição for boa (verdadeira) executar-se-á alguma coisa. Se a condição for ruim (falsa) executar-se-á outra coisa, ou em certos casos não se executará nada. Vale aqui a lógica binária com seus 2 princípios:

- Não contradição: Uma condição é verdadeira ou falsa, não podendo ser as duas coisas ao mesmo tempo.
- Terceiro excluído: se não for falsa será verdadeira e vice-versa.

desvio Um programa de computador vai executar a primeira instrução, depois a segunda e assim por diante, até o final do programa, a menos que uma instrução seja de desvio: *vá para a instrução x* : Neste caso a seguir será executada a instrução x e não a próxima.

Na década de 70, alguns autores (Dijkstra, Corrado e Jacopini) notaram que o uso de desvios tendia a gerar programas difíceis de *debugar*: Este verbo novo (=depurar) é o processo de identificar, localizar e corrigir erros de programas. Na programação gasta-se 20% do tempo escrevendo código e 80% depurando-o. A programação usando desvios gerou a chamada programação *spaguetti*. Daí os autores sugeriram organizar os desvios: nascia a programação estruturada. Agora os comandos são agrupados em blocos, havendo 3 deles:

sequência 2 ou mais comandos em sequência

alternativa como acima descrito

repetição uma sequência que pode se repetir. Apenas aqui é permitido o desvio e sempre do fim do bloco para o começo.

Fora o caso acima (repetição) o comando de desvio é proibido. Isto é tão verdade que linguagens mais modernas não têm o desvio, como por exemplo a linguagem Python.

Outra característica bem importante da *arquitetura de Von Neumann* é a idéia de programa armazenado: instruções são armazenadas como dados, permitindo que programas processem programas. Esta ideia já havia sido posta por Turing. Nascia o conceito de linguagem de programação, pela qual as complexidades da programação de máquina são transferidas para um programa de computador: o compilador. Uma linguagem de programação é na verdade 2 coisas:

⋆ um conjunto de regras fonéticas, sintáticas e semânticas de como descrever um algoritmo e
⋆ um programa (chamado compilador ou interpretador) capaz de ler o algoritmo acima descrito e gerar o equivalente código de máquina capaz de ser executado em um computador real.

Nascia o $software$ em oposição ao $hardware$. Graças a este conceito, aprender a programar passou de um processo de meses/anos para um de horas/dias. Sem contar que agora o mesmo programa roda em um supercomputador de milhões de dólares e também em um celular android.

Algoritmo O conceito chave em programação é o de *algoritmo*. É o que está por trás de um programa. É a ideia, a receita, a lista dos procedimentos do programa. Se programar é arte e engenharia, o algoritmo responde integralmente pela parte "arte". O conceito nos acompanha faz tempo (o algoritmo de mdc, devido a Euclides tem mais de 2400 anos), mas sua definição precisa é devida ao matemático inglês Alan Turing. Para poder expressar um algoritmo, se necessita uma linguagem apropriada. Vamos descrever uma mini-linguagem agora: As peças elementares de informação serão chamadas variáveis. Cada uma terá um nome, uma palavra qualquer (como aliás já fizemos nas aulas de álgebra em qualquer dos mais de 8 anos de ensino fundamental de matemática).

As operações de E/S serão 2: leia(variável) ou imprima(variável), significando a entrada do valor de uma variável ou sua impressão.

Para designar um valor a uma variável usar-se-á o comando de atribuição, representando por $\leftarrow$, então, por exemplo $A \leftarrow 0$ significa colocar o valor zero na variável A.

As operações aritméticas, seguem 100% do aprendido nas aulas de matemática e são: $+$: adição, $*$: multiplicação, $-$: subtração, $/$: divisão real, $//$: divisão inteira, MOD: resto de, $**$: potência, além de prioridade determinada por parêntesis. Continua proibida a divisão por zero. As linguagens mais modernas já operam normalmente com números complexos, mas podemos deixar esta facilidade de lado (por enquanto).

Para o comando alternativo precisa-se definir uma condição lógica: uma expressão envolvendo condicionais ($>$, $<$, $\leq$, $\geq$, $=$, $\neq$, $\in$, ...) e cuja resposta será um de dois valores: VERDADEIRO ou FALSO. Estes, ainda podem ser combinados usando os conectivos: E ($\wedge$) e OU ($\vee$) além de NÃO ($\sim$). O comando alternativo tem o formato `se <condição> então bloco-A fim{se}` neste comando o bloco de comandos A só será executado se a condição citada for verdadeira. Um formato alternativo é `se <condição> então bloco-A senão bloco-B fim{se}` Agora, se a condição é verdadeira, A é executado (como acima), mas se é falso então o bloco-B é executado.

Finalmente, o desvio não existe, mas em seu lugar entram as estruturas de repetição. Só vamos ver 1 delas. Trata-se do comando `enquanto` que tem formato `enquanto <condição> bloco-A fim{enquanto}`. Note que ele é parecido com o comando alternativo, mas tem uma diferença fundamental: ao encontrar o comando `fim{enquanto}` há um desvio incondicional para o início do bloco ou o `enquanto`.

Um exemplo Vamos escrever um algoritmo para testar se o número X é primo ou não.

```
 1: algoritmo PRIMO
 2: leia(X)
 3: DIVISORES ← 0
 4: TESTADOR ← 2
 5: enquanto TESTADOR < X
 6:     se X MOD TESTADOR = 0
 7:         DIVISORES ← DIVISORES + 1
 8:     fim{se}
 9:     TESTADOR ← TESTADOR + 1
10: fim{enquanto}
11: se DIVISORES = 0
12:     escreva("PRIMO")
13: senão
14:     escreva("NÃO PRIMO")
15: fim{se}
```

Seguir um algoritmo, simulando suas operações é chamado teste de mesa (ou em inglês *walkthrough*), ou mais jocosamente *fazer um chinês*. É o feijão com arroz da programação, usado para entender, certificar e depurar algoritmos. Ao fazer isto com o algoritmo acima, perceba que ele funciona. Se chamado com 15 responderá NÃO PRIMO e se chamado com 13 responderá PRIMO. Programar é escrever estas "coisas". É como aprender um idioma novo. Mas antes de aprender a expressar idéias (algoritmos), há que se ter firmeza na fonética e na síntaxe. A semântica virá depois.

Mais um exemplo Alguns aqui devem conhecer a sequência de Fibonacci: 1,1,2,3,5,8,13,21,34,... Note que, nela, cada número é a soma dos dois anteriores. A sequência começa com 1,1 já que estes não tem dois anteriores. A partir do próximo (o 2), a regra genérica se aplica. Vamos estudar um algoritmo que leia um número n e imprima o n-ésimo número de Fibonacci.

```
1: leia LIMITE
2: A ← 1; B ← 1; N ← 3
3: enquanto N ≤ LIMITE
4:     R ← A + B
5:     A ← B
6:     B ← R
7:     N ← N + 1
8: fim{enquanto}
9: escreva R
```

A seguir, alguns exercícios de síntaxe da linguagem:

☞ Para você fazer

Ex. 1

```
 1: X ← 8
 2: Y ← 31
 3: CTDS ← 2
 4: enquanto X < Y
 5:     X ← X + CTDS
 6:     se ((CTDS mod 2) ≠ 0)
 7:         CTDS ← CTDS + 2
 8:     fim{se}
 9:     CTDS ← CTDS + 1
10: fim{enquanto}
11: escreva X
```

Ex. 2

```
 1: A ← 2
 2: B ← 21
 3: enquanto A ≤ B
 4:     se ((B > 8) ∨ (A ≤ 15))
 5:         B ← B - 2
 6:     fim{se}
 7:     A ← A + 1
 8:     B ← B - 1
 9: fim{enquanto}
10: A ← A - 1
11: escreva (A + B)
```

Ex. 3

```
 1: X ← 5
 2: enquanto X < 11
 3:     Y ← 3
 4:     enquanto Y < 13
 5:         X ← X + 1
 6:         Y ← X + 1
 7:     fim{enquanto}
 8:     Y ← Y + 1
 9:     X ← X + 3
10: fim{enquanto}
11: escreva (X-Y)
```

Ex. 4

```
 1: X ← 6
 2: Y ← 6
 3: se  X < 5 ∧ Y > 5
 4:     X ← X - 2
 5: senão
 6:     Y ← Y + 1
 7:     se X mod 3 ≠ 0
 8:         Y ← Y + 3
 9:     fim{se}
10: fim{se}
11: escreva X+Y
```

Responda aqui os valores encontrados.

1	2	3	4

UFPR - UP - UTFPR - PUCPr – 24/04/2024 - 11:42:12.5
Prof Dr P Kantek (pkantek@gmail.com)
Aquecimento em programação VIVX001a, V: 2.06
75839 IAN DOLABELLA OHATA 24PRG101 - 7 / _____ / _____

Programar é ... Já se disse com alguma razão que programar é emprestar um pouco de inteligência ao computador. Então, antes de começar, reconheça-se a nobreza do que se está a fazer. Os computadores (todos) descendem de duas propostas mais ou menos complementares: a primeira, descrita brilhantemente por A. Turing em 1936 e a segunda igualmente brilhante, por Von Neumann em começos da década de 50. Pela primeira, sabe-se que o computador é capaz de executar uma instrução após a outra. Tais instruções precisam ser simples e inequívocas. Pela segunda descobre-se que tais instruções são bastante limitadas: E/S, aritmética, movimentação, alternativa e desvio.

E/S Este tipo de instrução responde pela E=entrada e S=saída de dados do/para o computador. Suponha que você construiu uma máquina capaz de descobrir o seno de qualquer ângulo. Para pôr a máquina para funcionar, você precisa fornecer algum ângulo a ela e depois que ela fizer seu trabalho ela vai precisar informar a você o valor do seno. Estas duas etapas correspondem a operações de E/S. Do ponto de vista teórico da programação, são operações quase irrelevantes, embora do ponto de vista prático sejam fundamentais.

aritmética Esta família é muito importante, mas graças a Deus, já estudamos pelo menos 8 anos de matemática. Tudo o que se aprendeu lá (adição, produto, potência, paridade, regra de sinais, números inteiros e reais, positivos e negativos, propriedades associativa, comutativa, distributiva – prioridades e parêntesis, etc etc, etc, etc) continua válido.

movimentação Operação muito simples, pela qual, partes da memória, chamadas variáveis são movimentadas ou mais propriamente copiadas entre sí.

alternativa Aqui a instrução mais "nobre" da programação. É graças a ela que um computador se torna diferente de um liquidificador. Se uma condição for boa (verdadeira) executar-se-á alguma coisa. Se a condição for ruim (falsa) executar-se-á outra coisa, ou em certos casos não se executará nada. Vale aqui a lógica binária com seus 2 princípios:

- Não contradição: Uma condição é verdadeira ou falsa, não podendo ser as duas coisas ao mesmo tempo.
- Terceiro excluído: se não for falsa será verdadeira e vice-versa.

desvio Um programa de computador vai executar a primeira instrução, depois a segunda e assim por diante, até o final do programa, a menos que uma instrução seja de desvio: *vá para a instrução x* : Neste caso a seguir será executada a instrução x e não a próxima.

Na década de 70, alguns autores (Dijkstra, Corrado e Jacopini) notaram que o uso de desvios tendia a gerar programas difíceis de *debugar*: Este verbo novo (=depurar) é o processo de identificar, localizar e corrigir erros de programas. Na programação gasta-se 20% do tempo escrevendo código e 80% depurando-o. A programação usando desvios gerou a chamada programação *spaguetti*. Daí os autores sugeriram organizar os desvios: nascia a programação estruturada. Agora os comandos são agrupados em blocos, havendo 3 deles:

sequência 2 ou mais comandos em sequência

alternativa como acima descrito

repetição uma sequência que pode se repetir. Apenas aqui é permitido o desvio e sempre do fim do bloco para o começo.

Fora o caso acima (repetição) o comando de desvio é proibido. Isto é tão verdade que linguagens mais modernas não têm o desvio, como por exemplo a linguagem Python.

Outra característica bem importante da *arquitetura de Von Neumann* é a idéia de programa armazenado: instruções são armazenadas como dados, permitindo que programas processem programas. Esta ideia já havia sido posta por Turing. Nascia o conceito de linguagem de programação, pela qual as complexidades da programação de máquina são transferidas para um programa de computador: o compilador. Uma linguagem de programação é na verdade 2 coisas:

⋆ um conjunto de regras fonéticas, sintáticas e semânticas de como descrever um algoritmo e
⋆ um programa (chamado compilador ou interpretador) capaz de ler o algoritmo acima descrito e gerar o equivalente código de máquina capaz de ser executado em um computador real.

Nascia o $software$ em oposição ao $hardware$. Graças a este conceito, aprender a programar passou de um processo de meses/anos para um de horas/dias. Sem contar que agora o mesmo programa roda em um supercomputador de milhões de dólares e também em um celular android.

Algoritmo O conceito chave em programação é o de *algoritmo*. É o que está por trás de um programa. É a ideia, a receita, a lista dos procedimentos do programa. Se programar é arte e engenharia, o algoritmo responde integralmente pela parte "arte". O conceito nos acompanha faz tempo (o algoritmo de mdc, devido a Euclides tem mais de 2400 anos), mas sua definição precisa é devida ao matemático inglês Alan Turing. Para poder expressar um algoritmo, se necessita uma linguagem apropriada. Vamos descrever uma mini-linguagem agora: As peças elementares de informação serão chamadas variáveis. Cada uma terá um nome, uma palavra qualquer (como aliás já fizemos nas aulas de álgebra em qualquer dos mais de 8 anos de ensino fundamental de matemática).

As operações de E/S serão 2: leia(variável) ou imprima(variável), significando a entrada do valor de uma variável ou sua impressão.

Para designar um valor a uma variável usar-se-á o comando de atribuição, representando por $\leftarrow$, então, por exemplo $A \leftarrow 0$ significa colocar o valor zero na variável A.

As operações aritméticas, seguem 100% do aprendido nas aulas de matemática e são: +: adição, $*$: multiplicação, $-$: subtração, /: divisão real, //: divisão inteira, MOD: resto de, $**$: potência, além de prioridade determinada por parêntesis. Continua proibida a divisão por zero. As linguagens mais modernas já operam normalmente com números complexos, mas podemos deixar esta facilidade de lado (por enquanto).

Para o comando alternativo precisa-se definir uma condição lógica: uma expressão envolvendo condicionais ($>$, $<$, $\leq$, $\geq$, $=$, $\neq$, $\in$, ...) e cuja resposta será um de dois valores: VERDADEIRO ou FALSO. Estes, ainda podem ser combinados usando os conectivos: E ($\wedge$) e OU ($\vee$) além de NÃO ($\sim$). O comando alternativo tem o formato `se <condição> então bloco-A fim{se}` neste comando o bloco de comandos A só será executado se a condição citada for verdadeira. Um formato alternativo é `se <condição> então bloco-A senão bloco-B fim{se}` Agora, se a condição é verdadeira, A é executado (como acima), mas se é falso então o bloco-B é executado.

Finalmente, o desvio não existe, mas em seu lugar entram as estruturas de repetição. Só vamos ver 1 delas. Trata-se do comando `enquanto` que tem formato `enquanto <condição> bloco-A fim{enquanto}`. Note que ele é parecido com o comando alternativo, mas tem uma diferença fundamental: ao encontrar o comando `fim{enquanto}` há um desvio incondicional para o início do bloco ou o `enquanto`.

Um exemplo Vamos escrever um algoritmo para testar se o número X é primo ou não.

```
 1: algoritmo PRIMO
 2: leia(X)
 3: DIVISORES ← 0
 4: TESTADOR ← 2
 5: enquanto TESTADOR < X
 6:     se X MOD TESTADOR = 0
 7:         DIVISORES ← DIVISORES + 1
 8:     fim{se}
 9:     TESTADOR ← TESTADOR + 1
10: fim{enquanto}
11: se DIVISORES = 0
12:     escreva("PRIMO")
13: senão
14:     escreva("NÃO PRIMO")
15: fim{se}
```

Seguir um algoritmo, simulando suas operações é chamado teste de mesa (ou em inglês *walkthrough*), ou mais jocosamente *fazer um chinês*. É o feijão com arroz da programação, usado para entender, certificar e depurar algoritmos. Ao fazer isto com o algoritmo acima, perceba que ele funciona. Se chamado com 15 responderá NÃO PRIMO e se chamado com 13 responderá PRIMO. Programar é escrever estas "coisas". É como aprender um idioma novo. Mas antes de aprender a expressar idéias (algoritmos), há que se ter firmeza na fonética e na síntaxe. A semântica virá depois.

Mais um exemplo Alguns aqui devem conhecer a sequência de Fibonacci: 1,1,2,3,5,8,13,21,34,... Note que, nela, cada número é a soma dos dois anteriores. A sequência começa com 1,1 já que estes não tem dois anteriores. A partir do próximo (o 2), a regra genérica se aplica. Vamos estudar um algoritmo que leia um número n e imprima o n-ésimo número de Fibonacci.

```
1: leia LIMITE
2: A ← 1; B ← 1; N ← 3
3: enquanto N ≤ LIMITE
4:     R ← A + B
5:     A ← B
6:     B ← R
7:     N ← N + 1
8: fim{enquanto}
9: escreva R
```

A seguir, alguns exercícios de síntaxe da linguagem:

☞ Para você fazer

Ex. 1

```
 1: X ← 7
 2: Y ← 35
 3: GAMA ← 4
 4: enquanto X < Y
 5:     X ← X + GAMA
 6:     se ((GAMA mod 3) ≠ 0)
 7:         GAMA ← GAMA + 1
 8:     fim{se}
 9:     GAMA ← GAMA + 1
10: fim{enquanto}
11: escreva X
```

Ex. 2

```
 1: A ← 2
 2: B ← 22
 3: enquanto A ≤ B
 4:     se ((B > 13) ∨ (A ≤ 6))
 5:         B ← B - 2
 6:     fim{se}
 7:     A ← A + 3
 8:     B ← B - 3
 9: fim{enquanto}
10: A ← A - 2
11: escreva (A + B)
```

Ex. 3

```
 1: X ← 4
 2: enquanto X < 9
 3:     Y ← 6
 4:     enquanto Y < 15
 5:         X ← X + 3
 6:         Y ← X + 1
 7:     fim{enquanto}
 8:     Y ← Y + 1
 9:     X ← X + 1
10: fim{enquanto}
11: escreva (X-Y)
```

Ex. 4

```
 1: X ← 3
 2: Y ← 3
 3: se  X < 4 ∧ Y > 5
 4:     X ← X - 1
 5: senão
 6:     Y ← Y + 3
 7:     se X mod 2 ≠ 0
 8:         Y ← Y + 1
 9:     fim{se}
10: fim{se}
11: escreva X+Y
```

Responda aqui os valores encontrados.

1	2	3	4

101-75839 - /

UFPR - UP - UTFPR - PUCPr – 24/04/2024 - 11:42:12.5
Prof Dr P Kantek (pkantek@gmail.com)
Aquecimento em programação VIVX001a, V: 2.06
75846 JOAO BONFIN LINO 24PRG101 - 8
/ _____ / _____

Programar é ... Já se disse com alguma razão que programar é emprestar um pouco de inteligência ao computador. Então, antes de começar, reconheça-se a nobreza do que se está a fazer. Os computadores (todos) descendem de duas propostas mais ou menos complementares: a primeira, descrita brilhantemente por A. Turing em 1936 e a segunda igualmente brilhante, por Von Neumann em começos da década de 50. Pela primeira, sabe-se que o computador é capaz de executar uma instrução após a outra. Tais instruções precisam ser simples e inequívocas. Pela segunda descobre-se que tais instruções são bastante limitadas: E/S, aritmética, movimentação, alternativa e desvio.

E/S Este tipo de instrução responde pela E=entrada e S=saída de dados do/para o computador. Suponha que você construiu uma máquina capaz de descobrir o seno de qualquer ângulo. Para pôr a máquina para funcionar, você precisa fornecer algum ângulo a ela e depois que ela fizer seu trabalho ela vai precisar informar a você o valor do seno. Estas duas etapas correspondem a operações de E/S. Do ponto de vista teórico da programação, são operações quase irrelevantes, embora do ponto de vista prático sejam fundamentais.

aritmética Esta família é muito importante, mas graças a Deus, já estudamos pelo menos 8 anos de matemática. Tudo o que se aprendeu lá (adição, produto, potência, paridade, regra de sinais, números inteiros e reais, positivos e negativos, propriedades associativa, comutativa, distributiva – prioridades e parêntesis, etc etc, etc, etc) continua válido.

movimentação Operação muito simples, pela qual, partes da memória, chamadas variáveis são movimentadas ou mais propriamente copiadas entre sí.

alternativa Aqui a instrução mais "nobre" da programação. É graças a ela que um computador se torna diferente de um liquidificador. Se uma condição for boa (verdadeira) executar-se-á alguma coisa. Se a condição for ruim (falsa) executar-se-á outra coisa, ou em certos casos não se executará nada. Vale aqui a lógica binária com seus 2 princípios:

- Não contradição: Uma condição é verdadeira ou falsa, não podendo ser as duas coisas ao mesmo tempo.
- Terceiro excluído: se não for falsa será verdadeira e vice-versa.

desvio Um programa de computador vai executar a primeira instrução, depois a segunda e assim por diante, até o final do programa, a menos que uma instrução seja de desvio: *vá para a instrução x* : Neste caso a seguir será executada a instrução x e não a próxima.

Na década de 70, alguns autores (Dijkstra, Corrado e Jacopini) notaram que o uso de desvios tendia a gerar programas difíceis de *debugar*: Este verbo novo (=depurar) é o processo de identificar, localizar e corrigir erros de programas. Na programação gasta-se 20% do tempo escrevendo código e 80% depurando-o. A programação usando desvios gerou a chamada programação *spaguetti*. Daí os autores sugeriram organizar os desvios: nascia a programação estruturada. Agora os comandos são agrupados em blocos, havendo 3 deles:

sequência 2 ou mais comandos em sequência
alternativa como acima descrito
repetição uma sequência que pode se repetir. Apenas aqui é permitido o desvio e sempre do fim do bloco para o começo.

Fora o caso acima (repetição) o comando de desvio é proibido. Isto é tão verdade que linguagens mais modernas não têm o desvio, como por exemplo a linguagem Python.

Outra característica bem importante da *arquitetura de Von Neumann* é a idéia de programa armazenado: instruções são armazenadas como dados, permitindo que programas processem programas. Esta ideia já havia sido posta por Turing. Nascia o conceito de linguagem de programação, pela qual as complexidades da programação de máquina são transferidas para um programa de computador: o compilador. Uma linguagem de programação é na verdade 2 coisas:

⋆ um conjunto de regras fonéticas, sintáticas e semânticas de como descrever um algoritmo e
⋆ um programa (chamado compilador ou interpretador) capaz de ler o algoritmo acima descrito e gerar o equivalente código de máquina capaz de ser executado em um computador real.

Nascia o $software$ em oposição ao $hardware$. Graças a este conceito, aprender a programar passou de um processo de meses/anos para um de horas/dias. Sem contar que agora o mesmo programa roda em um supercomputador de milhões de dólares e também em um celular android.

Algoritmo O conceito chave em programação é o de *algoritmo*. É o que está por trás de um programa. É a ideia, a receita, a lista dos procedimentos do programa. Se programar é arte e engenharia, o algoritmo responde integralmente pela parte "arte". O conceito nos acompanha faz tempo (o algoritmo de mdc, devido a Euclides tem mais de 2400 anos), mas sua definição precisa é devida ao matemático inglês Alan Turing. Para poder expressar um algoritmo, se necessita uma linguagem apropriada. Vamos descrever uma mini-linguagem agora: As peças elementares de informação serão chamadas variáveis. Cada uma terá um nome, uma palavra qualquer (como aliás já fizemos nas aulas de álgebra em qualquer dos mais de 8 anos de ensino fundamental de matemática).

As operações de E/S serão 2: leia(variável) ou imprima(variável), significando a entrada do valor de uma variável ou sua impressão.

Para designar um valor a uma variável usar-se-á o comando de atribuição, representando por $\leftarrow$, então, por exemplo $A \leftarrow 0$ significa colocar o valor zero na variável A.

As operações aritméticas, seguem 100% do aprendido nas aulas de matemática e são: +: adição, $*$: multiplicação, $-$: subtração, /: divisão real, //: divisão inteira, MOD: resto de, $**$: potência, além de prioridade determinada por parêntesis. Continua proibida a divisão por zero. As linguagens mais modernas já operam normalmente com números complexos, mas podemos deixar esta facilidade de lado (por enquanto).

Para o comando alternativo precisa-se definir uma condição lógica: uma expressão envolvendo condicionais ($>$, $<$, $\leq$, $\geq$, $=$, $\neq$, $\in$, ...) e cuja resposta será um de dois valores: VERDADEIRO ou FALSO. Estes, ainda podem ser combinados usando os conectivos: E ($\wedge$) e OU ($\vee$) além de NÃO ($\sim$). O comando alternativo tem o formato `se <condição> então bloco-A fim{se}` neste comando o bloco de comandos A só será executado se a condição citada for verdadeira. Um formato alternativo é `se <condição> então bloco-A senão bloco-B fim{se}` Agora, se a condição é verdadeira, A é executado (como acima), mas se é falso então o bloco-B é executado.

Finalmente, o desvio não existe, mas em seu lugar entram as estruturas de repetição. Só vamos ver 1 delas. Trata-se do comando `enquanto` que tem formato `enquanto <condição> bloco-A fim{enquanto}`. Note que ele é parecido com o comando alternativo, mas tem uma diferença fundamental: ao encontrar o comando `fim{enquanto}` há um desvio incondicional para o início do bloco ou o `enquanto`.

Um exemplo Vamos escrever um algoritmo para testar se o número X é primo ou não.

```
 1: algoritmo PRIMO
 2: leia(X)
 3: DIVISORES ← 0
 4: TESTADOR ← 2
 5: enquanto TESTADOR < X
 6:     se X MOD TESTADOR = 0
 7:         DIVISORES ← DIVISORES + 1
 8:     fim{se}
 9:     TESTADOR ← TESTADOR + 1
10: fim{enquanto}
11: se DIVISORES = 0
12:     escreva("PRIMO")
13: senão
14:     escreva("NÃO PRIMO")
15: fim{se}
```

Seguir um algoritmo, simulando suas operações é chamado teste de mesa (ou em inglês *walkthrough*), ou mais jocosamente *fazer um chinês*. É o feijão com arroz da programação, usado para entender, certificar e depurar algoritmos. Ao fazer isto com o algoritmo acima, perceba que ele funciona. Se chamado com 15 responderá NÃO PRIMO e se chamado com 13 responderá PRIMO. Programar é escrever estas "coisas". É como aprender um idioma novo. Mas antes de aprender a expressar idéias (algoritmos), há que se ter firmeza na fonética e na síntaxe. A semântica virá depois.

Mais um exemplo Alguns aqui devem conhecer a sequência de Fibonacci: 1,1,2,3,5,8,13,21,34,... Note que, nela, cada número é a soma dos dois anteriores. A sequência começa com 1,1 já que estes não tem dois anteriores. A partir do próximo (o 2), a regra genérica se aplica. Vamos estudar um algoritmo que leia um número n e imprima o n-ésimo número de Fibonacci.

```
1: leia LIMITE
2: A ← 1; B ← 1; N ← 3
3: enquanto N ≤ LIMITE
4:     R ← A + B
5:     A ← B
6:     B ← R
7:     N ← N + 1
8: fim{enquanto}
9: escreva R
```

A seguir, alguns exercícios de síntaxe da linguagem:

☞ Para você fazer

Ex. 1

```
 1: X ← 6
 2: Y ← 41
 3: BETA ← 2
 4: enquanto X < Y
 5:     X ← X + BETA
 6:     se ((BETA mod 3) ≠ 0)
 7:         BETA ← BETA + 2
 8:     fim{se}
 9:     BETA ← BETA + 1
10: fim{enquanto}
11: escreva X
```

Ex. 2

```
 1: A ← 1
 2: B ← 23
 3: enquanto A ≤ B
 4:     se ((B > 9) ∨ (A ≤ 6))
 5:         B ← B - 2
 6:     fim{se}
 7:     A ← A + 3
 8:     B ← B - 3
 9: fim{enquanto}
10: A ← A - 1
11: escreva (A + B)
```

Ex. 3

```
 1: X ← 3
 2: enquanto X < 13
 3:     Y ← 3
 4:     enquanto Y < 14
 5:         X ← X + 2
 6:         Y ← X + 1
 7:     fim{enquanto}
 8:     Y ← Y + 3
 9:     X ← X + 1
10: fim{enquanto}
11: escreva (X-Y)
```

Ex. 4

```
 1: X ← 6
 2: Y ← 1
 3: se  X < 6 ∧ Y > 3
 4:     X ← X - 2
 5: senão
 6:     Y ← Y + 1
 7:     se X mod 2 ≠ 0
 8:         Y ← Y + 1
 9:     fim{se}
10: fim{se}
11: escreva X+Y
```

Responda aqui os valores encontrados.

1	2	3	4

UFPR - UP - UTFPR - PUCPr – 24/04/2024 - 11:42:12.5
Prof Dr P Kantek (pkantek@gmail.com)
Aquecimento em programação VIVX001a, V: 2.06
75853 JOAO LUCA FERNANDES DE OLIVE
24PRG101 - 9 _____ / _____ / _____

Programar é ... Já se disse com alguma razão que programar é emprestar um pouco de inteligência ao computador. Então, antes de começar, reconheça-se a nobreza do que se está a fazer. Os computadores (todos) descendem de duas propostas mais ou menos complementares: a primeira, descrita brilhantemente por A. Turing em 1936 e a segunda igualmente brilhante, por Von Neumann em começos da década de 50. Pela primeira, sabe-se que o computador é capaz de executar uma instrução após a outra. Tais instruções precisam ser simples e inequívocas. Pela segunda descobre-se que tais instruções são bastante limitadas: E/S, aritmética, movimentação, alternativa e desvio.

E/S Este tipo de instrução responde pela E=entrada e S=saída de dados do/para o computador. Suponha que você construiu uma máquina capaz de descobrir o seno de qualquer ângulo. Para pôr a máquina para funcionar, você precisa fornecer algum ângulo a ela e depois que ela fizer seu trabalho ela vai precisar informar a você o valor do seno. Estas duas etapas correspondem a operações de E/S. Do ponto de vista teórico da programação, são operações quase irrelevantes, embora do ponto de vista prático sejam fundamentais.

aritmética Esta família é muito importante, mas graças a Deus, já estudamos pelo menos 8 anos de matemática. Tudo o que se aprendeu lá (adição, produto, potência, paridade, regra de sinais, números inteiros e reais, positivos e negativos, propriedades associativa, comutativa, distributiva – prioridades e parêntesis, etc etc, etc, etc) continua válido.

movimentação Operação muito simples, pela qual, partes da memória, chamadas variáveis são movimentadas ou mais propriamente copiadas entre sí.

alternativa Aqui a instrução mais "nobre" da programação. É graças a ela que um computador se torna diferente de um liquidificador. Se uma condição for boa (verdadeira) executar-se-á alguma coisa. Se a condição for ruim (falsa) executar-se-á outra coisa, ou em certos casos não se executará nada. Vale aqui a lógica binária com seus 2 princípios:

- Não contradição: Uma condição é verdadeira ou falsa, não podendo ser as duas coisas ao mesmo tempo.
- Terceiro excluído: se não for falsa será verdadeira e vice-versa.

desvio Um programa de computador vai executar a primeira instrução, depois a segunda e assim por diante, até o final do programa, a menos que uma instrução seja de desvio: *vá para a instrução x* : Neste caso a seguir será executada a instrução x e não a próxima.

Na década de 70, alguns autores (Dijkstra, Corrado e Jacopini) notaram que o uso de desvios tendia a gerar programas difíceis de *debugar*: Este verbo novo (=depurar) é o processo de identificar, localizar e corrigir erros de programas. Na programação gasta-se 20% do tempo escrevendo código e 80% depurando-o. A programação usando desvios gerou a chamada programação *spaguetti*. Daí os autores sugeriram organizar os desvios: nascia a programação estruturada. Agora os comandos são agrupados em blocos, havendo 3 deles:

sequência 2 ou mais comandos em sequência

alternativa como acima descrito

repetição uma sequência que pode se repetir. Apenas aqui é permitido o desvio e sempre do fim do bloco para o começo.

Fora o caso acima (repetição) o comando de desvio é proibido. Isto é tão verdade que linguagens mais modernas não têm o desvio, como por exemplo a linguagem Python.

Outra característica bem importante da *arquitetura de Von Neumann* é a idéia de programa armazenado: instruções são armazenadas como dados, permitindo que programas processem programas. Esta ideia já havia sido posta por Turing. Nascia o conceito de linguagem de programação, pela qual as complexidades da programação de máquina são transferidas para um programa de computador: o compilador. Uma linguagem de programação é na verdade 2 coisas:

⋆ um conjunto de regras fonéticas, sintáticas e semânticas de como descrever um algoritmo e
⋆ um programa (chamado compilador ou interpretador) capaz de ler o algoritmo acima descrito e gerar o equivalente código de máquina capaz de ser executado em um computador real.

Nascia o $software$ em oposição ao $hardware$. Graças a este conceito, aprender a programar passou de um processo de meses/anos para um de horas/dias. Sem contar que agora o mesmo programa roda em um supercomputador de milhões de dólares e também em um celular android.

Algoritmo O conceito chave em programação é o de *algoritmo*. É o que está por trás de um programa. É a ideia, a receita, a lista dos procedimentos do programa. Se programar é arte e engenharia, o algoritmo responde integralmente pela parte "arte". O conceito nos acompanha faz tempo (o algoritmo de mdc, devido a Euclides tem mais de 2400 anos), mas sua definição precisa é devida ao matemático inglês Alan Turing. Para poder expressar um algoritmo, se necessita uma linguagem apropriada. Vamos descrever uma mini-linguagem agora: As peças elementares de informação serão chamadas variáveis. Cada uma terá um nome, uma palavra qualquer (como aliás já fizemos nas aulas de álgebra em qualquer dos mais de 8 anos de ensino fundamental de matemática).

As operações de E/S serão 2: leia(variável) ou imprima(variável), significando a entrada do valor de uma variável ou sua impressão.

Para designar um valor a uma variável usar-se-á o comando de atribuição, representando por $\leftarrow$, então, por exemplo $A \leftarrow 0$ significa colocar o valor zero na variável A.

As operações aritméticas, seguem 100% do aprendido nas aulas de matemática e são: $+$: adição, $*$: multiplicação, $-$: subtração, $/$: divisão real, $//$: divisão inteira, MOD: resto de, $**$: potência, além de prioridade determinada por parêntesis. Continua proibida a divisão por zero. As linguagens mais modernas já operam normalmente com números complexos, mas podemos deixar esta facilidade de lado (por enquanto).

Para o comando alternativo precisa-se definir uma condição lógica: uma expressão envolvendo condicionais ($>$, $<$, $\leq$, $\geq$, $=$, $\neq$, $\in$, ...) e cuja resposta será um de dois valores: VERDADEIRO ou FALSO. Estes, ainda podem ser combinados usando os conectivos: E ($\wedge$) e OU ($\vee$) além de NÃO ($\sim$). O comando alternativo tem o formato `se <condição> então bloco-A fim{se}` neste comando o bloco de comandos A só será executado se a condição citada for verdadeira. Um formato alternativo é `se <condição> então bloco-A senão bloco-B fim{se}` Agora, se a condição é verdadeira, A é executado (como acima), mas se é falso então o bloco-B é executado.

Finalmente, o desvio não existe, mas em seu lugar entram as estruturas de repetição. Só vamos ver 1 delas. Trata-se do comando `enquanto` que tem formato `enquanto <condição> bloco-A fim{enquanto}`. Note que ele é parecido com o comando alternativo, mas tem uma diferença fundamental: ao encontrar o comando `fim{enquanto}` há um desvio incondicional para o início do bloco ou o `enquanto`.

Um exemplo Vamos escrever um algoritmo para testar se o número X é primo ou não.

```
 1: algoritmo PRIMO
 2: leia(X)
 3: DIVISORES ← 0
 4: TESTADOR ← 2
 5: enquanto TESTADOR < X
 6:     se X MOD TESTADOR = 0
 7:         DIVISORES ← DIVISORES + 1
 8:     fim{se}
 9:     TESTADOR ← TESTADOR + 1
10: fim{enquanto}
11: se DIVISORES = 0
12:     escreva("PRIMO")
13: senão
14:     escreva("NÃO PRIMO")
15: fim{se}
```

Seguir um algoritmo, simulando suas operações é chamado teste de mesa (ou em inglês *walkthrough*), ou mais jocosamente *fazer um chinês*. É o feijão com arroz da programação, usado para entender, certificar e depurar algoritmos. Ao fazer isto com o algoritmo acima, perceba que ele funciona. Se chamado com 15 responderá NÃO PRIMO e se chamado com 13 responderá PRIMO. Programar é escrever estas "coisas". É como aprender um idioma novo. Mas antes de aprender a expressar idéias (algoritmos), há que se ter firmeza na fonética e na síntaxe. A semântica virá depois.

Mais um exemplo Alguns aqui devem conhecer a sequência de Fibonacci: 1,1,2,3,5,8,13,21,34,... Note que, nela, cada número é a soma dos dois anteriores. A sequência começa com 1,1 já que estes não tem dois anteriores. A partir do próximo (o 2), a regra genérica se aplica. Vamos estudar um algoritmo que leia um número n e imprima o n-ésimo número de Fibonacci.

```
1: leia LIMITE
2: A ← 1; B ← 1; N ← 3
3: enquanto N ≤ LIMITE
4:     R ← A + B
5:     A ← B
6:     B ← R
7:     N ← N + 1
8: fim{enquanto}
9: escreva R
```

A seguir, alguns exercícios de síntaxe da linguagem:

☞ Para você fazer

Ex. 1

```
 1: X ← 6
 2: Y ← 37
 3: BETA ← 2
 4: enquanto X < Y
 5:     X ← X + BETA
 6:     se ((BETA mod 3) ≠ 0)
 7:         BETA ← BETA + 1
 8:     fim{se}
 9:     BETA ← BETA + 1
10: fim{enquanto}
11: escreva X
```

Ex. 2

```
 1: A ← 2
 2: B ← 22
 3: enquanto A ≤ B
 4:     se ((B > 11) ∨ (A ≤ 8))
 5:         B ← B - 2
 6:     fim{se}
 7:     A ← A + 1
 8:     B ← B - 2
 9: fim{enquanto}
10: A ← A - 1
11: escreva (A + B)
```

Ex. 3

```
 1: X ← 2
 2: enquanto X < 8
 3:     Y ← 4
 4:     enquanto Y < 12
 5:         X ← X + 3
 6:         Y ← X + 1
 7:     fim{enquanto}
 8:     Y ← Y + 2
 9:     X ← X + 2
10: fim{enquanto}
11: escreva (X-Y)
```

Ex. 4

```
 1: X ← 4
 2: Y ← 1
 3: se  X < 5 ∧ Y > 4
 4:     X ← X - 2
 5: senão
 6:     Y ← Y + 1
 7:     se X mod 4 ≠ 0
 8:         Y ← Y + 3
 9:     fim{se}
10: fim{se}
11: escreva X+Y
```

Responda aqui os valores encontrados.

1	2	3	4

UFPR - UP - UTFPR - PUCPr – 24/04/2024 - 11:42:12.5
Prof Dr P Kantek (pkantek@gmail.com)
Aquecimento em programação VIVX001a, V: 2.06
75860 JOAO PEDRO DE BRITO DUARTE
24PRG101 - 10 _____ / _____ / _____

Programar é ... Já se disse com alguma razão que programar é emprestar um pouco de inteligência ao computador. Então, antes de começar, reconheça-se a nobreza do que se está a fazer. Os computadores (todos) descendem de duas propostas mais ou menos complementares: a primeira, descrita brilhantemente por A. Turing em 1936 e a segunda igualmente brilhante, por Von Neumann em começos da década de 50. Pela primeira, sabe-se que o computador é capaz de executar uma instrução após a outra. Tais instruções precisam ser simples e inequívocas. Pela segunda descobre-se que tais instruções são bastante limitadas: E/S, aritmética, movimentação, alternativa e desvio.

E/S Este tipo de instrução responde pela E=entrada e S=saída de dados do/para o computador. Suponha que você construiu uma máquina capaz de descobrir o seno de qualquer ângulo. Para pôr a máquina para funcionar, você precisa fornecer algum ângulo a ela e depois que ela fizer seu trabalho ela vai precisar informar a você o valor do seno. Estas duas etapas correspondem a operações de E/S. Do ponto de vista teórico da programação, são operações quase irrelevantes, embora do ponto de vista prático sejam fundamentais.

aritmética Esta família é muito importante, mas graças a Deus, já estudamos pelo menos 8 anos de matemática. Tudo o que se aprendeu lá (adição, produto, potência, paridade, regra de sinais, números inteiros e reais, positivos e negativos, propriedades associativa, comutativa, distributiva – prioridades e parêntesis, etc etc, etc, etc) continua válido.

movimentação Operação muito simples, pela qual, partes da memória, chamadas variáveis são movimentadas ou mais propriamente copiadas entre sí.

alternativa Aqui a instrução mais "nobre" da programação. É graças a ela que um computador se torna diferente de um liquidificador. Se uma condição for boa (verdadeira) executar-se-á alguma coisa. Se a condição for ruim (falsa) executar-se-á outra coisa, ou em certos casos não se executará nada. Vale aqui a lógica binária com seus 2 princípios:

- Não contradição: Uma condição é verdadeira ou falsa, não podendo ser as duas coisas ao mesmo tempo.
- Terceiro excluído: se não for falsa será verdadeira e vice-versa.

desvio Um programa de computador vai executar a primeira instrução, depois a segunda e assim por diante, até o final do programa, a menos que uma instrução seja de desvio: *vá para a instrução x* : Neste caso a seguir será executada a instrução x e não a próxima.

Na década de 70, alguns autores (Dijkstra, Corrado e Jacopini) notaram que o uso de desvios tendia a gerar programas difíceis de *debugar*: Este verbo novo (=depurar) é o processo de identificar, localizar e corrigir erros de programas. Na programação gasta-se 20% do tempo escrevendo código e 80% depurando-o. A programação usando desvios gerou a chamada programação *spaguetti*. Daí os autores sugeriram organizar os desvios: nascia a programação estruturada. Agora os comandos são agrupados em blocos, havendo 3 deles:

sequência 2 ou mais comandos em sequência
alternativa como acima descrito
repetição uma sequência que pode se repetir. Apenas aqui é permitido o desvio e sempre do fim do bloco para o começo.

Fora o caso acima (repetição) o comando de desvio é proibido. Isto é tão verdade que linguagens mais modernas não têm o desvio, como por exemplo a linguagem Python.

Outra característica bem importante da *arquitetura de Von Neumann* é a idéia de programa armazenado: instruções são armazenadas como dados, permitindo que programas processem programas. Esta ideia já havia sido posta por Turing. Nascia o conceito de linguagem de programação, pela qual as complexidades da programação de máquina são transferidas para um programa de computador: o compilador. Uma linguagem de programação é na verdade 2 coisas:

⋆ um conjunto de regras fonéticas, sintáticas e semânticas de como descrever um algoritmo e
⋆ um programa (chamado compilador ou interpretador) capaz de ler o algoritmo acima descrito e gerar o equivalente código de máquina capaz de ser executado em um computador real.

Nascia o $software$ em oposição ao $hardware$. Graças a este conceito, aprender a programar passou de um processo de meses/anos para um de horas/dias. Sem contar que agora o mesmo programa roda em um supercomputador de milhões de dólares e também em um celular android.

Algoritmo O conceito chave em programação é o de *algoritmo*. É o que está por trás de um programa. É a ideia, a receita, a lista dos procedimentos do programa. Se programar é arte e engenharia, o algoritmo responde integralmente pela parte "arte". O conceito nos acompanha faz tempo (o algoritmo de mdc, devido a Euclides tem mais de 2400 anos), mas sua definição precisa é devida ao matemático inglês Alan Turing. Para poder expressar um algoritmo, se necessita uma linguagem apropriada. Vamos descrever uma mini-linguagem agora: As peças elementares de informação serão chamadas variáveis. Cada uma terá um nome, uma palavra qualquer (como aliás já fizemos nas aulas de álgebra em qualquer dos mais de 8 anos de ensino fundamental de matemática).

As operações de E/S serão 2: leia(variável) ou imprima(variável), significando a entrada do valor de uma variável ou sua impressão.

Para designar um valor a uma variável usar-se-á o comando de atribuição, representando por $\leftarrow$, então, por exemplo $A \leftarrow 0$ significa colocar o valor zero na variável A.

As operações aritméticas, seguem 100% do aprendido nas aulas de matemática e são: $+$: adição, $*$: multiplicação, $-$: subtração, $/$: divisão real, $//$: divisão inteira, MOD: resto de, $**$: potência, além de prioridade determinada por parêntesis. Continua proibida a divisão por zero. As linguagens mais modernas já operam normalmente com números complexos, mas podemos deixar esta facilidade de lado (por enquanto).

Para o comando alternativo precisa-se definir uma condição lógica: uma expressão envolvendo condicionais ($>$, $<$, $\leq$, $\geq$, $=$, $\neq$, $\in$, ...) e cuja resposta será um de dois valores: VERDADEIRO ou FALSO. Estes, ainda podem ser combinados usando os conectivos: E ($\wedge$) e OU ($\vee$) além de NÃO ($\sim$). O comando alternativo tem o formato `se <condição> então bloco-A fim{se}` neste comando o bloco de comandos A só será executado se a condição citada for verdadeira. Um formato alternativo é `se <condição> então bloco-A senão bloco-B fim{se}` Agora, se a condição é verdadeira, A é executado (como acima), mas se é falso então o bloco-B é executado.

Finalmente, o desvio não existe, mas em seu lugar entram as estruturas de repetição. Só vamos ver 1 delas. Trata-se do comando `enquanto` que tem formato `enquanto <condição> bloco-A fim{enquanto}`. Note que ele é parecido com o comando alternativo, mas tem uma diferença fundamental: ao encontrar o comando `fim{enquanto}` há um desvio incondicional para o início do bloco ou o `enquanto`.

Um exemplo Vamos escrever um algoritmo para testar se o número X é primo ou não.

```
 1: algoritmo PRIMO
 2: leia(X)
 3: DIVISORES ← 0
 4: TESTADOR ← 2
 5: enquanto TESTADOR < X
 6:     se X MOD TESTADOR = 0
 7:         DIVISORES ← DIVISORES + 1
 8:     fim{se}
 9:     TESTADOR ← TESTADOR + 1
10: fim{enquanto}
11: se DIVISORES = 0
12:     escreva("PRIMO")
13: senão
14:     escreva("NÃO PRIMO")
15: fim{se}
```

Seguir um algoritmo, simulando suas operações é chamado teste de mesa (ou em inglês *walkthrough*), ou mais jocosamente *fazer um chinês*. É o feijão com arroz da programação, usado para entender, certificar e depurar algoritmos. Ao fazer isto com o algoritmo acima, perceba que ele funciona. Se chamado com 15 responderá NÃO PRIMO e se chamado com 13 responderá PRIMO. Programar é escrever estas "coisas". É como aprender um idioma novo. Mas antes de aprender a expressar idéias (algoritmos), há que se ter firmeza na fonética e na síntaxe. A semântica virá depois.

Mais um exemplo Alguns aqui devem conhecer a sequência de Fibonacci: 1,1,2,3,5,8,13,21,34,... Note que, nela, cada número é a soma dos dois anteriores. A sequência começa com 1,1 já que estes não tem dois anteriores. A partir do próximo (o 2), a regra genérica se aplica. Vamos estudar um algoritmo que leia um número n e imprima o n-ésimo número de Fibonacci.

```
1: leia LIMITE
2: A ← 1; B ← 1; N ← 3
3: enquanto N ≤ LIMITE
4:     R ← A + B
5:     A ← B
6:     B ← R
7:     N ← N + 1
8: fim{enquanto}
9: escreva R
```

A seguir, alguns exercícios de síntaxe da linguagem:

☞ Para você fazer

Ex. 1

```
 1: X ← 7
 2: Y ← 32
 3: ALFA ← 3
 4: enquanto X < Y
 5:     X ← X + ALFA
 6:     se ((ALFA mod 3) ≠ 0)
 7:         ALFA ← ALFA + 1
 8:     fim{se}
 9:     ALFA ← ALFA + 1
10: fim{enquanto}
11: escreva X
```

Ex. 2

```
 1: A ← 3
 2: B ← 22
 3: enquanto A ≤ B
 4:     se ((B > 11) ∨ (A ≤ 8))
 5:         B ← B - 1
 6:     fim{se}
 7:     A ← A + 3
 8:     B ← B - 3
 9: fim{enquanto}
10: A ← A - 3
11: escreva (A + B)
```

Ex. 3

```
 1: X ← 3
 2: enquanto X < 11
 3:     Y ← 1
 4:     enquanto Y < 13
 5:         X ← X + 1
 6:         Y ← X + 2
 7:     fim{enquanto}
 8:     Y ← Y + 2
 9:     X ← X + 3
10: fim{enquanto}
11: escreva (X-Y)
```

Ex. 4

```
 1: X ← 3
 2: Y ← 9
 3: se  X < 5 ∧ Y > 5
 4:     X ← X - 1
 5: senão
 6:     Y ← Y + 2
 7:     se X mod 4 ≠ 0
 8:         Y ← Y + 1
 9:     fim{se}
10: fim{se}
11: escreva X+Y
```

Responda aqui os valores encontrados.

1	2	3	4

UFPR - UP - UTFPR - PUCPr – 24/04/2024 - 11:42:12.5
Prof Dr P Kantek (pkantek@gmail.com)
Aquecimento em programação VIVX001a, V: 2.06
75877 JOAO PEDRO MARIANO SUEKE
24PRG101 - 11 _____ / _____ / _____

Programar é ... Já se disse com alguma razão que programar é emprestar um pouco de inteligência ao computador. Então, antes de começar, reconheça-se a nobreza do que se está a fazer. Os computadores (todos) descendem de duas propostas mais ou menos complementares: a primeira, descrita brilhantemente por A. Turing em 1936 e a segunda igualmente brilhante, por Von Neumann em começos da década de 50. Pela primeira, sabe-se que o computador é capaz de executar uma instrução após a outra. Tais instruções precisam ser simples e inequívocas. Pela segunda descobre-se que tais instruções são bastante limitadas: E/S, aritmética, movimentação, alternativa e desvio.

E/S Este tipo de instrução responde pela E=entrada e S=saída de dados do/para o computador. Suponha que você construiu uma máquina capaz de descobrir o seno de qualquer ângulo. Para pôr a máquina para funcionar, você precisa fornecer algum ângulo a ela e depois que ela fizer seu trabalho ela vai precisar informar a você o valor do seno. Estas duas etapas correspondem a operações de E/S. Do ponto de vista teórico da programação, são operações quase irrelevantes, embora do ponto de vista prático sejam fundamentais.

aritmética Esta família é muito importante, mas graças a Deus, já estudamos pelo menos 8 anos de matemática. Tudo o que se aprendeu lá (adição, produto, potência, paridade, regra de sinais, números inteiros e reais, positivos e negativos, propriedades associativa, comutativa, distributiva – prioridades e parêntesis, etc etc, etc, etc) continua válido.

movimentação Operação muito simples, pela qual, partes da memória, chamadas variáveis são movimentadas ou mais propriamente copiadas entre sí.

alternativa Aqui a instrução mais "nobre" da programação. É graças a ela que um computador se torna diferente de um liquidificador. Se uma condição for boa (verdadeira) executar-se-á alguma coisa. Se a condição for ruim (falsa) executar-se-á outra coisa, ou em certos casos não se executará nada. Vale aqui a lógica binária com seus 2 princípios:

- Não contradição: Uma condição é verdadeira ou falsa, não podendo ser as duas coisas ao mesmo tempo.
- Terceiro excluído: se não for falsa será verdadeira e vice-versa.

desvio Um programa de computador vai executar a primeira instrução, depois a segunda e assim por diante, até o final do programa, a menos que uma instrução seja de desvio: *vá para a instrução x* : Neste caso a seguir será executada a instrução x e não a próxima.

Na década de 70, alguns autores (Dijkstra, Corrado e Jacopini) notaram que o uso de desvios tendia a gerar programas difíceis de *debugar*: Este verbo novo (=depurar) é o processo de identificar, localizar e corrigir erros de programas. Na programação gasta-se 20% do tempo escrevendo código e 80% depurando-o. A programação usando desvios gerou a chamada programação *spaguetti*. Daí os autores sugeriram organizar os desvios: nascia a programação estruturada. Agora os comandos são agrupados em blocos, havendo 3 deles:

sequência 2 ou mais comandos em sequência

alternativa como acima descrito

repetição uma sequência que pode se repetir. Apenas aqui é permitido o desvio e sempre do fim do bloco para o começo.

Fora o caso acima (repetição) o comando de desvio é proibido. Isto é tão verdade que linguagens mais modernas não têm o desvio, como por exemplo a linguagem Python.

Outra característica bem importante da *arquitetura de Von Neumann* é a idéia de programa armazenado: instruções são armazenadas como dados, permitindo que programas processem programas. Esta ideia já havia sido posta por Turing. Nascia o conceito de linguagem de programação, pela qual as complexidades da programação de máquina são transferidas para um programa de computador: o compilador. Uma linguagem de programação é na verdade 2 coisas:

⋆ um conjunto de regras fonéticas, sintáticas e semânticas de como descrever um algoritmo e
⋆ um programa (chamado compilador ou interpretador) capaz de ler o algoritmo acima descrito e gerar o equivalente código de máquina capaz de ser executado em um computador real.

Nascia o *software* em oposição ao *hardware*. Graças a este conceito, aprender a programar passou de um processo de meses/anos para um de horas/dias. Sem contar que agora o mesmo programa roda em um supercomputador de milhões de dólares e também em um celular android.

Algoritmo O conceito chave em programação é o de *algoritmo*. É o que está por trás de um programa. É a ideia, a receita, a lista dos procedimentos do programa. Se programar é arte e engenharia, o algoritmo responde integralmente pela parte "arte". O conceito nos acompanha faz tempo (o algoritmo de mdc, devido a Euclides tem mais de 2400 anos), mas sua definição precisa é devida ao matemático inglês Alan Turing. Para poder expressar um algoritmo, se necessita uma linguagem apropriada. Vamos descrever uma mini-linguagem agora: As peças elementares de informação serão chamadas variáveis. Cada uma terá um nome, uma palavra qualquer (como aliás já fizemos nas aulas de álgebra em qualquer dos mais de 8 anos de ensino fundamental de matemática).

As operações de E/S serão 2: leia(variável) ou imprima(variável), significando a entrada do valor de uma variável ou sua impressão.

Para designar um valor a uma variável usar-se-á o comando de atribuição, representando por $\leftarrow$, então, por exemplo $A \leftarrow 0$ significa colocar o valor zero na variável A.

As operações aritméticas, seguem 100% do aprendido nas aulas de matemática e são: +: adição, ∗: multiplicação, −: subtração, /: divisão real, //: divisão inteira, MOD: resto de, ∗∗: potência, além de prioridade determinada por parêntesis. Continua proibida a divisão por zero. As linguagens mais modernas já operam normalmente com números complexos, mas podemos deixar esta facilidade de lado (por enquanto).

Para o comando alternativo precisa-se definir uma condição lógica: uma expressão envolvendo condicionais ($>$, $<$, $\leq$, $\geq$, $=$, $\neq$, $\in$, ...) e cuja resposta será um de dois valores: VERDADEIRO ou FALSO. Estes, ainda podem ser combinados usando os conectivos: E ($\wedge$) e OU ($\vee$) além de NÃO ($\sim$). O comando alternativo tem o formato `se <condição> então bloco-A fim{se}` neste comando o bloco de comandos A só será executado se a condição citada for verdadeira. Um formato alternativo é `se <condição> então bloco-A senão bloco-B fim{se}` Agora, se a condição é verdadeira, A é executado (como acima), mas se é falso então o bloco-B é executado.

Finalmente, o desvio não existe, mas em seu lugar entram as estruturas de repetição. Só vamos ver 1 delas. Trata-se do comando `enquanto` que tem formato `enquanto <condição> bloco-A fim{enquanto}`. Note que ele é parecido com o comando alternativo, mas tem uma diferença fundamental: ao encontrar o comando `fim{enquanto}` há um desvio incondicional para o início do bloco ou o `enquanto`.

Um exemplo Vamos escrever um algoritmo para testar se o número X é primo ou não.

```
 1: algoritmo PRIMO
 2: leia(X)
 3: DIVISORES ← 0
 4: TESTADOR ← 2
 5: enquanto TESTADOR < X
 6:     se X MOD TESTADOR = 0
 7:         DIVISORES ← DIVISORES + 1
 8:     fim{se}
 9:     TESTADOR ← TESTADOR + 1
10: fim{enquanto}
11: se DIVISORES = 0
12:     escreva("PRIMO")
13: senão
14:     escreva("NÃO PRIMO")
15: fim{se}
```

Seguir um algoritmo, simulando suas operações é chamado teste de mesa (ou em inglês *walkthrough*), ou mais jocosamente *fazer um chinês*. É o feijão com arroz da programação, usado para entender, certificar e depurar algoritmos. Ao fazer isto com o algoritmo acima, perceba que ele funciona. Se chamado com 15 responderá NÃO PRIMO e se chamado com 13 responderá PRIMO. Programar é escrever estas "coisas". É como aprender um idioma novo. Mas antes de aprender a expressar idéias (algoritmos), há que se ter firmeza na fonética e na síntaxe. A semântica virá depois.

Mais um exemplo Alguns aqui devem conhecer a sequência de Fibonacci: 1,1,2,3,5,8,13,21,34,... Note que, nela, cada número é a soma dos dois anteriores. A sequência começa com 1,1 já que estes não tem dois anteriores. A partir do próximo (o 2), a regra genérica se aplica. Vamos estudar um algoritmo que leia um número n e imprima o n-ésimo número de Fibonacci.

```
1: leia LIMITE
2: A ← 1; B ← 1; N ← 3
3: enquanto N ≤ LIMITE
4:     R ← A + B
5:     A ← B
6:     B ← R
7:     N ← N + 1
8: fim{enquanto}
9: escreva R
```

A seguir, alguns exercícios de síntaxe da linguagem:

☞ Para você fazer

Ex. 1

```
 1: X ← 7
 2: Y ← 40
 3: GAMA ← 4
 4: enquanto X < Y
 5:     X ← X + GAMA
 6:     se ((GAMA mod 3) ≠ 0)
 7:         GAMA ← GAMA + 1
 8:     fim{se}
 9:     GAMA ← GAMA + 1
10: fim{enquanto}
11: escreva X
```

Ex. 2

```
 1: A ← 3
 2: B ← 23
 3: enquanto A ≤ B
 4:     se ((B > 6) ∨ (A ≤ 11))
 5:         B ← B - 1
 6:     fim{se}
 7:     A ← A + 2
 8:     B ← B - 3
 9: fim{enquanto}
10: A ← A - 3
11: escreva (A + B)
```

Ex. 3

```
 1: X ← 5
 2: enquanto X < 6
 3:     Y ← 2
 4:     enquanto Y < 15
 5:         X ← X + 2
 6:         Y ← X + 1
 7:     fim{enquanto}
 8:     Y ← Y + 3
 9:     X ← X + 1
10: fim{enquanto}
11: escreva (X-Y)
```

Ex. 4

```
 1: X ← 6
 2: Y ← 2
 3: se  X < 4 ∧ Y > 3
 4:     X ← X - 3
 5: senão
 6:     Y ← Y + 2
 7:     se X mod 4 ≠ 0
 8:         Y ← Y + 3
 9:     fim{se}
10: fim{se}
11: escreva X+Y
```

Responda aqui os valores encontrados.

1	2	3	4

101-75877 - /

UFPR - UP - UTFPR - PUCPr – 24/04/2024 - 11:42:12.5
Prof Dr P Kantek (pkantek@gmail.com)
Aquecimento em programação VIVX001a, V: 2.06
75884 JOAO VICTOR GOMES VILELA GER
24PRG101 - 12 _____ / _____ / _____

Programar é ... Já se disse com alguma razão que programar é emprestar um pouco de inteligência ao computador. Então, antes de começar, reconheça-se a nobreza do que se está a fazer. Os computadores (todos) descendem de duas propostas mais ou menos complementares: a primeira, descrita brilhantemente por A. Turing em 1936 e a segunda igualmente brilhante, por Von Neumann em começos da década de 50. Pela primeira, sabe-se que o computador é capaz de executar uma instrução após a outra. Tais instruções precisam ser simples e inequívocas. Pela segunda descobre-se que tais instruções são bastante limitadas: E/S, aritmética, movimentação, alternativa e desvio.

E/S Este tipo de instrução responde pela E=entrada e S=saída de dados do/para o computador. Suponha que você construiu uma máquina capaz de descobrir o seno de qualquer ângulo. Para pôr a máquina para funcionar, você precisa fornecer algum ângulo a ela e depois que ela fizer seu trabalho ela vai precisar informar a você o valor do seno. Estas duas etapas correspondem a operações de E/S. Do ponto de vista teórico da programação, são operações quase irrelevantes, embora do ponto de vista prático sejam fundamentais.

aritmética Esta família é muito importante, mas graças a Deus, já estudamos pelo menos 8 anos de matemática. Tudo o que se aprendeu lá (adição, produto, potência, paridade, regra de sinais, números inteiros e reais, positivos e negativos, propriedades associativa, comutativa, distributiva – prioridades e parêntesis, etc etc, etc, etc) continua válido.

movimentação Operação muito simples, pela qual, partes da memória, chamadas variáveis são movimentadas ou mais propriamente copiadas entre sí.

alternativa Aqui a instrução mais "nobre" da programação. É graças a ela que um computador se torna diferente de um liquidificador. Se uma condição for boa (verdadeira) executar-se-á alguma coisa. Se a condição for ruim (falsa) executar-se-á outra coisa, ou em certos casos não se executará nada. Vale aqui a lógica binária com seus 2 princípios:

- Não contradição: Uma condição é verdadeira ou falsa, não podendo ser as duas coisas ao mesmo tempo.
- Terceiro excluído: se não for falsa será verdadeira e vice-versa.

desvio Um programa de computador vai executar a primeira instrução, depois a segunda e assim por diante, até o final do programa, a menos que uma instrução seja de desvio: *vá para a instrução x* : Neste caso a seguir será executada a instrução x e não a próxima.

Na década de 70, alguns autores (Dijkstra, Corrado e Jacopini) notaram que o uso de desvios tendia a gerar programas difíceis de *debugar*: Este verbo novo (=depurar) é o processo de identificar, localizar e corrigir erros de programas. Na programação gasta-se 20% do tempo escrevendo código e 80% depurando-o. A programação usando desvios gerou a chamada programação *spaguetti*. Daí os autores sugeriram organizar os desvios: nascia a programação estruturada. Agora os comandos são agrupados em blocos, havendo 3 deles:

sequência 2 ou mais comandos em sequência

alternativa como acima descrito

repetição uma sequência que pode se repetir. Apenas aqui é permitido o desvio e sempre do fim do bloco para o começo.

Fora o caso acima (repetição) o comando de desvio é proibido. Isto é tão verdade que linguagens mais modernas não têm o desvio, como por exemplo a linguagem Python.

Outra característica bem importante da *arquitetura de Von Neumann* é a idéia de programa armazenado: instruções são armazenadas como dados, permitindo que programas processem programas. Esta ideia já havia sido posta por Turing. Nascia o conceito de linguagem de programação, pela qual as complexidades da programação de máquina são transferidas para um programa de computador: o compilador. Uma linguagem de programação é na verdade 2 coisas:

⋆ um conjunto de regras fonéticas, sintáticas e semânticas de como descrever um algoritmo e
⋆ um programa (chamado compilador ou interpretador) capaz de ler o algoritmo acima descrito e gerar o equivalente código de máquina capaz de ser executado em um computador real.

Nascia o $software$ em oposição ao $hardware$. Graças a este conceito, aprender a programar passou de um processo de meses/anos para um de horas/dias. Sem contar que agora o mesmo programa roda em um supercomputador de milhões de dólares e também em um celular android.

Algoritmo O conceito chave em programação é o de *algoritmo*. É o que está por trás de um programa. É a ideia, a receita, a lista dos procedimentos do programa. Se programar é arte e engenharia, o algoritmo responde integralmente pela parte "arte". O conceito nos acompanha faz tempo (o algoritmo de mdc, devido a Euclides tem mais de 2400 anos), mas sua definição precisa é devida ao matemático inglês Alan Turing. Para poder expressar um algoritmo, se necessita uma linguagem apropriada. Vamos descrever uma mini-linguagem agora: As peças elementares de informação serão chamadas variáveis. Cada uma terá um nome, uma palavra qualquer (como aliás já fizemos nas aulas de álgebra em qualquer dos mais de 8 anos de ensino fundamental de matemática).

As operações de E/S serão 2: leia(variável) ou imprima(variável), significando a entrada do valor de uma variável ou sua impressão.

Para designar um valor a uma variável usar-se-á o comando de atribuição, representando por $\leftarrow$, então, por exemplo $A \leftarrow 0$ significa colocar o valor zero na variável A.

As operações aritméticas, seguem 100% do aprendido nas aulas de matemática e são: +: adição, $*$: multiplicação, $-$: subtração, /: divisão real, //: divisão inteira, MOD: resto de, $**$: potência, além de prioridade determinada por parêntesis. Continua proibida a divisão por zero. As linguagens mais modernas já operam normalmente com números complexos, mas podemos deixar esta facilidade de lado (por enquanto).

Para o comando alternativo precisa-se definir uma condição lógica: uma expressão envolvendo condicionais ($>$, $<$, $\leq$, $\geq$, $=$, $\neq$, $\in$, ...) e cuja resposta será um de dois valores: VERDADEIRO ou FALSO. Estes, ainda podem ser combinados usando os conectivos: E ($\wedge$) e OU ($\vee$) além de NÃO ($\sim$). O comando alternativo tem o formato `se <condição> então bloco-A fim{se}` neste comando o bloco de comandos A só será executado se a condição citada for verdadeira. Um formato alternativo é `se <condição> então bloco-A senão bloco-B fim{se}` Agora, se a condição é verdadeira, A é executado (como acima), mas se é falso então o bloco-B é executado.

Finalmente, o desvio não existe, mas em seu lugar entram as estruturas de repetição. Só vamos ver 1 delas. Trata-se do comando `enquanto` que tem formato `enquanto <condição> bloco-A fim{enquanto}`. Note que ele é parecido com o comando alternativo, mas tem uma diferença fundamental: ao encontrar o comando `fim{enquanto}` há um desvio incondicional para o início do bloco ou o `enquanto`.

Um exemplo Vamos escrever um algoritmo para testar se o número X é primo ou não.

```
 1: algoritmo PRIMO
 2: leia(X)
 3: DIVISORES ← 0
 4: TESTADOR ← 2
 5: enquanto TESTADOR < X
 6:     se X MOD TESTADOR = 0
 7:         DIVISORES ← DIVISORES + 1
 8:     fim{se}
 9:     TESTADOR ← TESTADOR + 1
10: fim{enquanto}
11: se DIVISORES = 0
12:     escreva("PRIMO")
13: senão
14:     escreva("NÃO PRIMO")
15: fim{se}
```

Seguir um algoritmo, simulando suas operações é chamado teste de mesa (ou em inglês *walkthrough*), ou mais jocosamente *fazer um chinês*. É o feijão com arroz da programação, usado para entender, certificar e depurar algoritmos. Ao fazer isto com o algoritmo acima, perceba que ele funciona. Se chamado com 15 responderá NÃO PRIMO e se chamado com 13 responderá PRIMO. Programar é escrever estas "coisas". É como aprender um idioma novo. Mas antes de aprender a expressar idéias (algoritmos), há que se ter firmeza na fonética e na síntaxe. A semântica virá depois.

Mais um exemplo Alguns aqui devem conhecer a sequência de Fibonacci: 1,1,2,3,5,8,13,21,34,... Note que, nela, cada número é a soma dos dois anteriores. A sequência começa com 1,1 já que estes não tem dois anteriores. A partir do próximo (o 2), a regra genérica se aplica. Vamos estudar um algoritmo que leia um número n e imprima o n-ésimo número de Fibonacci.

```
1: leia LIMITE
2: A ← 1; B ← 1; N ← 3
3: enquanto N ≤ LIMITE
4:     R ← A + B
5:     A ← B
6:     B ← R
7:     N ← N + 1
8: fim{enquanto}
9: escreva R
```

A seguir, alguns exercícios de síntaxe da linguagem:

☞ Para você fazer

Ex. 1

```
 1: X ← 9
 2: Y ← 37
 3: ALIN ← 2
 4: enquanto X < Y
 5:     X ← X + ALIN
 6:     se ((ALIN mod 3) ≠ 0)
 7:         ALIN ← ALIN + 1
 8:     fim{se}
 9:     ALIN ← ALIN + 1
10: fim{enquanto}
11: escreva X
```

Ex. 2

```
 1: A ← 3
 2: B ← 22
 3: enquanto A ≤ B
 4:     se ((B > 12) ∨ (A ≤ 13))
 5:         B ← B - 2
 6:     fim{se}
 7:     A ← A + 2
 8:     B ← B - 1
 9: fim{enquanto}
10: A ← A - 2
11: escreva (A + B)
```

Ex. 3

```
 1: X ← 3
 2: enquanto X < 7
 3:     Y ← 2
 4:     enquanto Y < 12
 5:         X ← X + 1
 6:         Y ← X + 2
 7:     fim{enquanto}
 8:     Y ← Y + 1
 9:     X ← X + 2
10: fim{enquanto}
11: escreva (X-Y)
```

Ex. 4

```
 1: X ← 9
 2: Y ← 4
 3: se  X < 6 ∧ Y > 5
 4:     X ← X - 2
 5: senão
 6:     Y ← Y + 1
 7:     se X mod 4 ≠ 0
 8:         Y ← Y + 2
 9:     fim{se}
10: fim{se}
11: escreva X+Y
```

Responda aqui os valores encontrados.

1	2	3	4

101-75884 - /

UFPR - UP - UTFPR - PUCPr – 24/04/2024 - 11:42:12.5
Prof Dr P Kantek (pkantek@gmail.com)
Aquecimento em programação VIVX001a, V: 2.06
75989 LUCAS ANTOSZCZYSZEN 24PRG101 - 13 _____ / _____ / _____

Programar é ... Já se disse com alguma razão que programar é emprestar um pouco de inteligência ao computador. Então, antes de começar, reconheça-se a nobreza do que se está a fazer. Os computadores (todos) descendem de duas propostas mais ou menos complementares: a primeira, descrita brilhantemente por A. Turing em 1936 e a segunda igualmente brilhante, por Von Neumann em começos da década de 50. Pela primeira, sabe-se que o computador é capaz de executar uma instrução após a outra. Tais instruções precisam ser simples e inequívocas. Pela segunda descobre-se que tais instruções são bastante limitadas: E/S, aritmética, movimentação, alternativa e desvio.

E/S Este tipo de instrução responde pela E=entrada e S=saída de dados do/para o computador. Suponha que você construiu uma máquina capaz de descobrir o seno de qualquer ângulo. Para pôr a máquina para funcionar, você precisa fornecer algum ângulo a ela e depois que ela fizer seu trabalho ela vai precisar informar a você o valor do seno. Estas duas etapas correspondem a operações de E/S. Do ponto de vista teórico da programação, são operações quase irrelevantes, embora do ponto de vista prático sejam fundamentais.

aritmética Esta família é muito importante, mas graças a Deus, já estudamos pelo menos 8 anos de matemática. Tudo o que se aprendeu lá (adição, produto, potência, paridade, regra de sinais, números inteiros e reais, positivos e negativos, propriedades associativa, comutativa, distributiva – prioridades e parêntesis, etc etc, etc, etc) continua válido.

movimentação Operação muito simples, pela qual, partes da memória, chamadas variáveis são movimentadas ou mais propriamente copiadas entre sí.

alternativa Aqui a instrução mais "nobre" da programação. É graças a ela que um computador se torna diferente de um liquidificador. Se uma condição for boa (verdadeira) executar-se-á alguma coisa. Se a condição for ruim (falsa) executar-se-á outra coisa, ou em certos casos não se executará nada. Vale aqui a lógica binária com seus 2 princípios:

- Não contradição: Uma condição é verdadeira ou falsa, não podendo ser as duas coisas ao mesmo tempo.
- Terceiro excluído: se não for falsa será verdadeira e vice-versa.

desvio Um programa de computador vai executar a primeira instrução, depois a segunda e assim por diante, até o final do programa, a menos que uma instrução seja de desvio: *vá para a instrução x* : Neste caso a seguir será executada a instrução x e não a próxima.

Na década de 70, alguns autores (Dijkstra, Corrado e Jacopini) notaram que o uso de desvios tendia a gerar programas difíceis de *debugar*: Este verbo novo (=depurar) é o processo de identificar, localizar e corrigir erros de programas. Na programação gasta-se 20% do tempo escrevendo código e 80% depurando-o. A programação usando desvios gerou a chamada programação *spaguetti*. Daí os autores sugeriram organizar os desvios: nascia a programação estruturada. Agora os comandos são agrupados em blocos, havendo 3 deles:

sequência 2 ou mais comandos em sequência

alternativa como acima descrito

repetição uma sequência que pode se repetir. Apenas aqui é permitido o desvio e sempre do fim do bloco para o começo.

Fora o caso acima (repetição) o comando de desvio é proibido. Isto é tão verdade que linguagens mais modernas não têm o desvio, como por exemplo a linguagem Python.

Outra característica bem importante da *arquitetura de Von Neumann* é a idéia de programa armazenado: instruções são armazenadas como dados, permitindo que programas processem programas. Esta ideia já havia sido posta por Turing. Nascia o conceito de linguagem de programação, pela qual as complexidades da programação de máquina são transferidas para um programa de computador: o compilador. Uma linguagem de programação é na verdade 2 coisas:

⋆ um conjunto de regras fonéticas, sintáticas e semânticas de como descrever um algoritmo e
⋆ um programa (chamado compilador ou interpretador) capaz de ler o algoritmo acima descrito e gerar o equivalente código de máquina capaz de ser executado em um computador real.

Nascia o $software$ em oposição ao $hardware$. Graças a este conceito, aprender a programar passou de um processo de meses/anos para um de horas/dias. Sem contar que agora o mesmo programa roda em um supercomputador de milhões de dólares e também em um celular android.

Algoritmo O conceito chave em programação é o de *algoritmo*. É o que está por trás de um programa. É a ideia, a receita, a lista dos procedimentos do programa. Se programar é arte e engenharia, o algoritmo responde integralmente pela parte "arte". O conceito nos acompanha faz tempo (o algoritmo de mdc, devido a Euclides tem mais de 2400 anos), mas sua definição precisa é devida ao matemático inglês Alan Turing. Para poder expressar um algoritmo, se necessita uma linguagem apropriada. Vamos descrever uma mini-linguagem agora: As peças elementares de informação serão chamadas variáveis. Cada uma terá um nome, uma palavra qualquer (como aliás já fizemos nas aulas de álgebra em qualquer dos mais de 8 anos de ensino fundamental de matemática).

As operações de E/S serão 2: leia(variável) ou imprima(variável), significando a entrada do valor de uma variável ou sua impressão.

Para designar um valor a uma variável usar-se-á o comando de atribuição, representando por $\leftarrow$, então, por exemplo $A \leftarrow 0$ significa colocar o valor zero na variável A.

As operações aritméticas, seguem 100% do aprendido nas aulas de matemática e são: $+$: adição, $*$: multiplicação, $-$: subtração, $/$: divisão real, $//$: divisão inteira, MOD: resto de, $**$: potência, além de prioridade determinada por parêntesis. Continua proibida a divisão por zero. As linguagens mais modernas já operam normalmente com números complexos, mas podemos deixar esta facilidade de lado (por enquanto).

Para o comando alternativo precisa-se definir uma condição lógica: uma expressão envolvendo condicionais ($>$, $<$, $\leq$, $\geq$, $=$, $\neq$, $\in$, ...) e cuja resposta será um de dois valores: VERDADEIRO ou FALSO. Estes, ainda podem ser combinados usando os conectivos: E ($\wedge$) e OU ($\vee$) além de NÃO ($\sim$). O comando alternativo tem o formato `se <condição> então bloco-A fim{se}` neste comando o bloco de comandos A só será executado se a condição citada for verdadeira. Um formato alternativo é `se <condição> então bloco-A senão bloco-B fim{se}` Agora, se a condição é verdadeira, A é executado (como acima), mas se é falso então o bloco-B é executado.

Finalmente, o desvio não existe, mas em seu lugar entram as estruturas de repetição. Só vamos ver 1 delas. Trata-se do comando `enquanto` que tem formato `enquanto <condição> bloco-A fim{enquanto}`. Note que ele é parecido com o comando alternativo, mas tem uma diferença fundamental: ao encontrar o comando `fim{enquanto}` há um desvio incondicional para o início do bloco ou o `enquanto`.

Um exemplo Vamos escrever um algoritmo para testar se o número X é primo ou não.

```
 1: algoritmo PRIMO
 2: leia(X)
 3: DIVISORES ← 0
 4: TESTADOR ← 2
 5: enquanto TESTADOR < X
 6:     se X MOD TESTADOR = 0
 7:         DIVISORES ← DIVISORES + 1
 8:     fim{se}
 9:     TESTADOR ← TESTADOR + 1
10: fim{enquanto}
11: se DIVISORES = 0
12:     escreva("PRIMO")
13: senão
14:     escreva("NÃO PRIMO")
15: fim{se}
```

Seguir um algoritmo, simulando suas operações é chamado teste de mesa (ou em inglês *walkthrough*), ou mais jocosamente *fazer um chinês*. É o feijão com arroz da programação, usado para entender, certificar e depurar algoritmos. Ao fazer isto com o algoritmo acima, perceba que ele funciona. Se chamado com 15 responderá NÃO PRIMO e se chamado com 13 responderá PRIMO. Programar é escrever estas "coisas". É como aprender um idioma novo. Mas antes de aprender a expressar idéias (algoritmos), há que se ter firmeza na fonética e na síntaxe. A semântica virá depois.

Mais um exemplo Alguns aqui devem conhecer a sequência de Fibonacci: 1,1,2,3,5,8,13,21,34,... Note que, nela, cada número é a soma dos dois anteriores. A sequência começa com 1,1 já que estes não tem dois anteriores. A partir do próximo (o 2), a regra genérica se aplica. Vamos estudar um algoritmo que leia um número n e imprima o n-ésimo número de Fibonacci.

```
1: leia LIMITE
2: A ← 1; B ← 1; N ← 3
3: enquanto N ≤ LIMITE
4:     R ← A + B
5:     A ← B
6:     B ← R
7:     N ← N + 1
8: fim{enquanto}
9: escreva R
```

A seguir, alguns exercícios de síntaxe da linguagem:

☞ Para você fazer

Ex. 1

```
 1: X ← 8
 2: Y ← 42
 3: ALFA ← 3
 4: enquanto X < Y
 5:     X ← X + ALFA
 6:     se ((ALFA mod 3) ≠ 0)
 7:         ALFA ← ALFA + 1
 8:     fim{se}
 9:     ALFA ← ALFA + 1
10: fim{enquanto}
11: escreva X
```

Ex. 2

```
 1: A ← 2
 2: B ← 22
 3: enquanto A ≤ B
 4:     se ((B > 14) ∨ (A ≤ 13))
 5:         B ← B - 1
 6:     fim{se}
 7:     A ← A + 2
 8:     B ← B - 2
 9: fim{enquanto}
10: A ← A - 2
11: escreva (A + B)
```

Ex. 3

```
 1: X ← 1
 2: enquanto X < 13
 3:     Y ← 5
 4:     enquanto Y < 14
 5:         X ← X + 1
 6:         Y ← X + 2
 7:     fim{enquanto}
 8:     Y ← Y + 1
 9:     X ← X + 1
10: fim{enquanto}
11: escreva (X-Y)
```

Ex. 4

```
 1: X ← 3
 2: Y ← 6
 3: se  X < 5 ∧ Y > 4
 4:     X ← X - 1
 5: senão
 6:     Y ← Y + 2
 7:     se X mod 3 ≠ 0
 8:         Y ← Y + 1
 9:     fim{se}
10: fim{se}
11: escreva X+Y
```

Responda aqui os valores encontrados.

1	2	3	4

UFPR - UP - UTFPR - PUCPr – 24/04/2024 - 11:42:12.5
Prof Dr P Kantek (pkantek@gmail.com)
Aquecimento em programação VIVX001a, V: 2.06
75891 LUIZ FELIPE GUEDES BUCCHERI
24PRG101 - 14 _____ / _____ / _____

Programar é ... Já se disse com alguma razão que programar é emprestar um pouco de inteligência ao computador. Então, antes de começar, reconheça-se a nobreza do que se está a fazer. Os computadores (todos) descendem de duas propostas mais ou menos complementares: a primeira, descrita brilhantemente por A. Turing em 1936 e a segunda igualmente brilhante, por Von Neumann em começos da década de 50. Pela primeira, sabe-se que o computador é capaz de executar uma instrução após a outra. Tais instruções precisam ser simples e inequívocas. Pela segunda descobre-se que tais instruções são bastante limitadas: E/S, aritmética, movimentação, alternativa e desvio.

E/S Este tipo de instrução responde pela E=entrada e S=saída de dados do/para o computador. Suponha que você construiu uma máquina capaz de descobrir o seno de qualquer ângulo. Para pôr a máquina para funcionar, você precisa fornecer algum ângulo a ela e depois que ela fizer seu trabalho ela vai precisar informar a você o valor do seno. Estas duas etapas correspondem a operações de E/S. Do ponto de vista teórico da programação, são operações quase irrelevantes, embora do ponto de vista prático sejam fundamentais.

aritmética Esta família é muito importante, mas graças a Deus, já estudamos pelo menos 8 anos de matemática. Tudo o que se aprendeu lá (adição, produto, potência, paridade, regra de sinais, números inteiros e reais, positivos e negativos, propriedades associativa, comutativa, distributiva – prioridades e parêntesis, etc etc, etc, etc) continua válido.

movimentação Operação muito simples, pela qual, partes da memória, chamadas variáveis são movimentadas ou mais propriamente copiadas entre sí.

alternativa Aqui a instrução mais "nobre" da programação. É graças a ela que um computador se torna diferente de um liquidificador. Se uma condição for boa (verdadeira) executar-se-á alguma coisa. Se a condição for ruim (falsa) executar-se-á outra coisa, ou em certos casos não se executará nada. Vale aqui a lógica binária com seus 2 princípios:

- Não contradição: Uma condição é verdadeira ou falsa, não podendo ser as duas coisas ao mesmo tempo.
- Terceiro excluído: se não for falsa será verdadeira e vice-versa.

desvio Um programa de computador vai executar a primeira instrução, depois a segunda e assim por diante, até o final do programa, a menos que uma instrução seja de desvio: *vá para a instrução x* : Neste caso a seguir será executada a instrução x e não a próxima.

Na década de 70, alguns autores (Dijkstra, Corrado e Jacopini) notaram que o uso de desvios tendia a gerar programas difíceis de *debugar*: Este verbo novo (=depurar) é o processo de identificar, localizar e corrigir erros de programas. Na programação gasta-se 20% do tempo escrevendo código e 80% depurando-o. A programação usando desvios gerou a chamada programação *spaguetti*. Daí os autores sugeriram organizar os desvios: nascia a programação estruturada. Agora os comandos são agrupados em blocos, havendo 3 deles:

sequência 2 ou mais comandos em sequência

alternativa como acima descrito

repetição uma sequência que pode se repetir. Apenas aqui é permitido o desvio e sempre do fim do bloco para o começo.

Fora o caso acima (repetição) o comando de desvio é proibido. Isto é tão verdade que linguagens mais modernas não têm o desvio, como por exemplo a linguagem Python.

Outra característica bem importante da *arquitetura de Von Neumann* é a idéia de programa armazenado: instruções são armazenadas como dados, permitindo que programas processem programas. Esta ideia já havia sido posta por Turing. Nascia o conceito de linguagem de programação, pela qual as complexidades da programação de máquina são transferidas para um programa de computador: o compilador. Uma linguagem de programação é na verdade 2 coisas:

⋆ um conjunto de regras fonéticas, sintáticas e semânticas de como descrever um algoritmo e
⋆ um programa (chamado compilador ou interpretador) capaz de ler o algoritmo acima descrito e gerar o equivalente código de máquina capaz de ser executado em um computador real.

Nascia o *software* em oposição ao *hardware*. Graças a este conceito, aprender a programar passou de um processo de meses/anos para um de horas/dias. Sem contar que agora o mesmo programa roda em um supercomputador de milhões de dólares e também em um celular android.

Algoritmo O conceito chave em programação é o de *algoritmo*. É o que está por trás de um programa. É a ideia, a receita, a lista dos procedimentos do programa. Se programar é arte e engenharia, o algoritmo responde integralmente pela parte "arte". O conceito nos acompanha faz tempo (o algoritmo de mdc, devido a Euclides tem mais de 2400 anos), mas sua definição precisa é devida ao matemático inglês Alan Turing. Para poder expressar um algoritmo, se necessita uma linguagem apropriada. Vamos descrever uma mini-linguagem agora: As peças elementares de informação serão chamadas variáveis. Cada uma terá um nome, uma palavra qualquer (como aliás já fizemos nas aulas de álgebra em qualquer dos mais de 8 anos de ensino fundamental de matemática).

As operações de E/S serão 2: leia(variável) ou imprima(variável), significando a entrada do valor de uma variável ou sua impressão.

Para designar um valor a uma variável usar-se-á o comando de atribuição, representando por $\leftarrow$, então, por exemplo $A \leftarrow 0$ significa colocar o valor zero na variável A.

As operações aritméticas, seguem 100% do aprendido nas aulas de matemática e são: $+$: adição, $*$: multiplicação, $-$: subtração, $/$: divisão real, $//$: divisão inteira, MOD: resto de, $**$: potência, além de prioridade determinada por parêntesis. Continua proibida a divisão por zero. As linguagens mais modernas já operam normalmente com números complexos, mas podemos deixar esta facilidade de lado (por enquanto).

Para o comando alternativo precisa-se definir uma condição lógica: uma expressão envolvendo condicionais ($>$, $<$, $\leq$, $\geq$, $=$, $\neq$, $\in$, ...) e cuja resposta será um de dois valores: VERDADEIRO ou FALSO. Estes, ainda podem ser combinados usando os conectivos: E ($\wedge$) e OU ($\vee$) além de NÃO ($\sim$). O comando alternativo tem o formato `se <condição> então bloco-A fim{se}` neste comando o bloco de comandos A só será executado se a condição citada for verdadeira. Um formato alternativo é `se <condição> então bloco-A senão bloco-B fim{se}` Agora, se a condição é verdadeira, A é executado (como acima), mas se é falso então o bloco-B é executado.

Finalmente, o desvio não existe, mas em seu lugar entram as estruturas de repetição. Só vamos ver 1 delas. Trata-se do comando `enquanto` que tem formato `enquanto <condição> bloco-A fim{enquanto}`. Note que ele é parecido com o comando alternativo, mas tem uma diferença fundamental: ao encontrar o comando `fim{enquanto}` há um desvio incondicional para o início do bloco ou o `enquanto`.

Um exemplo Vamos escrever um algoritmo para testar se o número X é primo ou não.

```
 1: algoritmo PRIMO
 2: leia(X)
 3: DIVISORES ← 0
 4: TESTADOR ← 2
 5: enquanto TESTADOR < X
 6:     se X MOD TESTADOR = 0
 7:         DIVISORES ← DIVISORES + 1
 8:     fim{se}
 9:     TESTADOR ← TESTADOR + 1
10: fim{enquanto}
11: se DIVISORES = 0
12:     escreva("PRIMO")
13: senão
14:     escreva("NÃO PRIMO")
15: fim{se}
```

Seguir um algoritmo, simulando suas operações é chamado teste de mesa (ou em inglês *walkthrough*), ou mais jocosamente *fazer um chinês*. É o feijão com arroz da programação, usado para entender, certificar e depurar algoritmos. Ao fazer isto com o algoritmo acima, perceba que ele funciona. Se chamado com 15 responderá NÃO PRIMO e se chamado com 13 responderá PRIMO. Programar é escrever estas "coisas". É como aprender um idioma novo. Mas antes de aprender a expressar idéias (algoritmos), há que se ter firmeza na fonética e na síntaxe. A semântica virá depois.

Mais um exemplo Alguns aqui devem conhecer a sequência de Fibonacci: 1,1,2,3,5,8,13,21,34,... Note que, nela, cada número é a soma dos dois anteriores. A sequência começa com 1,1 já que estes não tem dois anteriores. A partir do próximo (o 2), a regra genérica se aplica. Vamos estudar um algoritmo que leia um número n e imprima o n-ésimo número de Fibonacci.

```
1: leia LIMITE
2: A ← 1; B ← 1; N ← 3
3: enquanto N ≤ LIMITE
4:     R ← A + B
5:     A ← B
6:     B ← R
7:     N ← N + 1
8: fim{enquanto}
9: escreva R
```

A seguir, alguns exercícios de síntaxe da linguagem:

☞ Para você fazer

Ex. 1

```
 1: X ← 8
 2: Y ← 34
 3: BETA ← 2
 4: enquanto X < Y
 5:     X ← X + BETA
 6:     se ((BETA mod 3) ≠ 0)
 7:         BETA ← BETA + 2
 8:     fim{se}
 9:     BETA ← BETA + 1
10: fim{enquanto}
11: escreva X
```

Ex. 2

```
 1: A ← 1
 2: B ← 21
 3: enquanto A ≤ B
 4:     se ((B > 13) ∨ (A ≤ 11))
 5:         B ← B - 1
 6:     fim{se}
 7:     A ← A + 2
 8:     B ← B - 2
 9: fim{enquanto}
10: A ← A - 2
11: escreva (A + B)
```

Ex. 3

```
 1: X ← 2
 2: enquanto X < 11
 3:     Y ← 1
 4:     enquanto Y < 12
 5:         X ← X + 1
 6:         Y ← X + 1
 7:     fim{enquanto}
 8:     Y ← Y + 1
 9:     X ← X + 2
10: fim{enquanto}
11: escreva (X-Y)
```

Ex. 4

```
 1: X ← 3
 2: Y ← 7
 3: se  X < 5 ∧ Y > 4
 4:     X ← X - 2
 5: senão
 6:     Y ← Y + 3
 7:     se X mod 3 ≠ 0
 8:         Y ← Y + 1
 9:     fim{se}
10: fim{se}
11: escreva X+Y
```

Responda aqui os valores encontrados.

1	2	3	4

UFPR - UP - UTFPR - PUCPr – 24/04/2024 - 11:42:12.5
Prof Dr P Kantek (pkantek@gmail.com)
Aquecimento em programação VIVX001a, V: 2.06
75903 MANUELLA LEAL DE MEIRELLES F
24PRG101 - 15 _____ / _____ / _____

Programar é ... Já se disse com alguma razão que programar é emprestar um pouco de inteligência ao computador. Então, antes de começar, reconheça-se a nobreza do que se está a fazer. Os computadores (todos) descendem de duas propostas mais ou menos complementares: a primeira, descrita brilhantemente por A. Turing em 1936 e a segunda igualmente brilhante, por Von Neumann em começos da década de 50. Pela primeira, sabe-se que o computador é capaz de executar uma instrução após a outra. Tais instruções precisam ser simples e inequívocas. Pela segunda descobre-se que tais instruções são bastante limitadas: E/S, aritmética, movimentação, alternativa e desvio.

E/S Este tipo de instrução responde pela E=entrada e S=saída de dados do/para o computador. Suponha que você construiu uma máquina capaz de descobrir o seno de qualquer ângulo. Para pôr a máquina para funcionar, você precisa fornecer algum ângulo a ela e depois que ela fizer seu trabalho ela vai precisar informar a você o valor do seno. Estas duas etapas correspondem a operações de E/S. Do ponto de vista teórico da programação, são operações quase irrelevantes, embora do ponto de vista prático sejam fundamentais.

aritmética Esta família é muito importante, mas graças a Deus, já estudamos pelo menos 8 anos de matemática. Tudo o que se aprendeu lá (adição, produto, potência, paridade, regra de sinais, números inteiros e reais, positivos e negativos, propriedades associativa, comutativa, distributiva – prioridades e parêntesis, etc etc, etc, etc) continua válido.

movimentação Operação muito simples, pela qual, partes da memória, chamadas variáveis são movimentadas ou mais propriamente copiadas entre sí.

alternativa Aqui a instrução mais "nobre" da programação. É graças a ela que um computador se torna diferente de um liquidificador. Se uma condição for boa (verdadeira) executar-se-á alguma coisa. Se a condição for ruim (falsa) executar-se-á outra coisa, ou em certos casos não se executará nada. Vale aqui a lógica binária com seus 2 princípios:

- Não contradição: Uma condição é verdadeira ou falsa, não podendo ser as duas coisas ao mesmo tempo.
- Terceiro excluído: se não for falsa será verdadeira e vice-versa.

desvio Um programa de computador vai executar a primeira instrução, depois a segunda e assim por diante, até o final do programa, a menos que uma instrução seja de desvio: *vá para a instrução x* : Neste caso a seguir será executada a instrução x e não a próxima.

Na década de 70, alguns autores (Dijkstra, Corrado e Jacopini) notaram que o uso de desvios tendia a gerar programas difíceis de *debugar*: Este verbo novo (=depurar) é o processo de identificar, localizar e corrigir erros de programas. Na programação gasta-se 20% do tempo escrevendo código e 80% depurando-o. A programação usando desvios gerou a chamada programação *spaguetti*. Daí os autores sugeriram organizar os desvios: nascia a programação estruturada. Agora os comandos são agrupados em blocos, havendo 3 deles:

sequência 2 ou mais comandos em sequência

alternativa como acima descrito

repetição uma sequência que pode se repetir. Apenas aqui é permitido o desvio e sempre do fim do bloco para o começo.

Fora o caso acima (repetição) o comando de desvio é proibido. Isto é tão verdade que linguagens mais modernas não têm o desvio, como por exemplo a linguagem Python.

Outra característica bem importante da *arquitetura de Von Neumann* é a idéia de programa armazenado: instruções são armazenadas como dados, permitindo que programas processem programas. Esta ideia já havia sido posta por Turing. Nascia o conceito de linguagem de programação, pela qual as complexidades da programação de máquina são transferidas para um programa de computador: o compilador. Uma linguagem de programação é na verdade 2 coisas:

⋆ um conjunto de regras fonéticas, sintáticas e semânticas de como descrever um algoritmo e
⋆ um programa (chamado compilador ou interpretador) capaz de ler o algoritmo acima descrito e gerar o equivalente código de máquina capaz de ser executado em um computador real.

Nascia o $software$ em oposição ao $hardware$. Graças a este conceito, aprender a programar passou de um processo de meses/anos para um de horas/dias. Sem contar que agora o mesmo programa roda em um supercomputador de milhões de dólares e também em um celular android.

Algoritmo O conceito chave em programação é o de *algoritmo*. É o que está por trás de um programa. É a ideia, a receita, a lista dos procedimentos do programa. Se programar é arte e engenharia, o algoritmo responde integralmente pela parte "arte". O conceito nos acompanha faz tempo (o algoritmo de mdc, devido a Euclides tem mais de 2400 anos), mas sua definição precisa é devida ao matemático inglês Alan Turing. Para poder expressar um algoritmo, se necessita uma linguagem apropriada. Vamos descrever uma mini-linguagem agora: As peças elementares de informação serão chamadas variáveis. Cada uma terá um nome, uma palavra qualquer (como aliás já fizemos nas aulas de álgebra em qualquer dos mais de 8 anos de ensino fundamental de matemática).

As operações de E/S serão 2: leia(variável) ou imprima(variável), significando a entrada do valor de uma variável ou sua impressão.

Para designar um valor a uma variável usar-se-á o comando de atribuição, representando por $\leftarrow$, então, por exemplo $A \leftarrow 0$ significa colocar o valor zero na variável A.

As operações aritméticas, seguem 100% do aprendido nas aulas de matemática e são: +: adição, $*$: multiplicação, $-$: subtração, /: divisão real, //: divisão inteira, MOD: resto de, $**$: potência, além de prioridade determinada por parêntesis. Continua proibida a divisão por zero. As linguagens mais modernas já operam normalmente com números complexos, mas podemos deixar esta facilidade de lado (por enquanto).

Para o comando alternativo precisa-se definir uma condição lógica: uma expressão envolvendo condicionais ($>$, $<$, $\leq$, $\geq$, $=$, $\neq$, $\in$, ...) e cuja resposta será um de dois valores: VERDADEIRO ou FALSO. Estes, ainda podem ser combinados usando os conectivos: E ($\wedge$) e OU ($\vee$) além de NÃO ($\sim$). O comando alternativo tem o formato `se <condição> então bloco-A fim{se}` neste comando o bloco de comandos A só será executado se a condição citada for verdadeira. Um formato alternativo é `se <condição> então bloco-A senão bloco-B fim{se}` Agora, se a condição é verdadeira, A é executado (como acima), mas se é falso então o bloco-B é executado.

Finalmente, o desvio não existe, mas em seu lugar entram as estruturas de repetição. Só vamos ver 1 delas. Trata-se do comando `enquanto` que tem formato `enquanto <condição> bloco-A fim{enquanto}`. Note que ele é parecido com o comando alternativo, mas tem uma diferença fundamental: ao encontrar o comando `fim{enquanto}` há um desvio incondicional para o início do bloco ou o `enquanto`.

Um exemplo Vamos escrever um algoritmo para testar se o número X é primo ou não.

```
 1: algoritmo PRIMO
 2: leia(X)
 3: DIVISORES ← 0
 4: TESTADOR ← 2
 5: enquanto TESTADOR < X
 6:     se X MOD TESTADOR = 0
 7:         DIVISORES ← DIVISORES + 1
 8:     fim{se}
 9:     TESTADOR ← TESTADOR + 1
10: fim{enquanto}
11: se DIVISORES = 0
12:     escreva("PRIMO")
13: senão
14:     escreva("NÃO PRIMO")
15: fim{se}
```

Seguir um algoritmo, simulando suas operações é chamado teste de mesa (ou em inglês *walkthrough*), ou mais jocosamente *fazer um chinês*. É o feijão com arroz da programação, usado para entender, certificar e depurar algoritmos. Ao fazer isto com o algoritmo acima, perceba que ele funciona. Se chamado com 15 responderá NÃO PRIMO e se chamado com 13 responderá PRIMO. Programar é escrever estas "coisas". É como aprender um idioma novo. Mas antes de aprender a expressar idéias (algoritmos), há que se ter firmeza na fonética e na síntaxe. A semântica virá depois.

Mais um exemplo Alguns aqui devem conhecer a sequência de Fibonacci: 1,1,2,3,5,8,13,21,34,... Note que, nela, cada número é a soma dos dois anteriores. A sequência começa com 1,1 já que estes não tem dois anteriores. A partir do próximo (o 2), a regra genérica se aplica. Vamos estudar um algoritmo que leia um número n e imprima o n-ésimo número de Fibonacci.

```
1: leia LIMITE
2: A ← 1; B ← 1; N ← 3
3: enquanto N ≤ LIMITE
4:     R ← A + B
5:     A ← B
6:     B ← R
7:     N ← N + 1
8: fim{enquanto}
9: escreva R
```

A seguir, alguns exercícios de síntaxe da linguagem:

☞ Para você fazer

Ex. 1

```
 1: X ← 7
 2: Y ← 40
 3: ALIN ← 3
 4: enquanto X < Y
 5:     X ← X + ALIN
 6:     se ((ALIN mod 2) ≠ 0)
 7:         ALIN ← ALIN + 1
 8:     fim{se}
 9:     ALIN ← ALIN + 1
10: fim{enquanto}
11: escreva X
```

Ex. 2

```
 1: A ← 1
 2: B ← 22
 3: enquanto A ≤ B
 4:     se ((B > 9) ∨ (A ≤ 12))
 5:         B ← B - 1
 6:     fim{se}
 7:     A ← A + 1
 8:     B ← B - 2
 9: fim{enquanto}
10: A ← A - 3
11: escreva (A + B)
```

Ex. 3

```
 1: X ← 5
 2: enquanto X < 13
 3:     Y ← 3
 4:     enquanto Y < 11
 5:         X ← X + 1
 6:         Y ← X + 2
 7:     fim{enquanto}
 8:     Y ← Y + 1
 9:     X ← X + 1
10: fim{enquanto}
11: escreva (X-Y)
```

Ex. 4

```
 1: X ← 6
 2: Y ← 7
 3: se  X < 6 ∧ Y > 4
 4:     X ← X - 3
 5: senão
 6:     Y ← Y + 1
 7:     se X mod 2 ≠ 0
 8:         Y ← Y + 1
 9:     fim{se}
10: fim{se}
11: escreva X+Y
```

Responda aqui os valores encontrados.

1	2	3	4

UFPR - UP - UTFPR - PUCPr – 24/04/2024 - 11:42:12.5
Prof Dr P Kantek (pkantek@gmail.com)
Aquecimento em programação VIVX001a, V: 2.06
75910 MARIA EDUARDA DA SILVA 24PRG101 - 16 _____ / _____ / _____

Programar é ... Já se disse com alguma razão que programar é emprestar um pouco de inteligência ao computador. Então, antes de começar, reconheça-se a nobreza do que se está a fazer. Os computadores (todos) descendem de duas propostas mais ou menos complementares: a primeira, descrita brilhantemente por A. Turing em 1936 e a segunda igualmente brilhante, por Von Neumann em começos da década de 50. Pela primeira, sabe-se que o computador é capaz de executar uma instrução após a outra. Tais instruções precisam ser simples e inequívocas. Pela segunda descobre-se que tais instruções são bastante limitadas: E/S, aritmética, movimentação, alternativa e desvio.

E/S Este tipo de instrução responde pela E=entrada e S=saída de dados do/para o computador. Suponha que você construiu uma máquina capaz de descobrir o seno de qualquer ângulo. Para pôr a máquina para funcionar, você precisa fornecer algum ângulo a ela e depois que ela fizer seu trabalho ela vai precisar informar a você o valor do seno. Estas duas etapas correspondem a operações de E/S. Do ponto de vista teórico da programação, são operações quase irrelevantes, embora do ponto de vista prático sejam fundamentais.

aritmética Esta família é muito importante, mas graças a Deus, já estudamos pelo menos 8 anos de matemática. Tudo o que se aprendeu lá (adição, produto, potência, paridade, regra de sinais, números inteiros e reais, positivos e negativos, propriedades associativa, comutativa, distributiva – prioridades e parêntesis, etc etc, etc, etc) continua válido.

movimentação Operação muito simples, pela qual, partes da memória, chamadas variáveis são movimentadas ou mais propriamente copiadas entre sí.

alternativa Aqui a instrução mais "nobre" da programação. É graças a ela que um computador se torna diferente de um liquidificador. Se uma condição for boa (verdadeira) executar-se-á alguma coisa. Se a condição for ruim (falsa) executar-se-á outra coisa, ou em certos casos não se executará nada. Vale aqui a lógica binária com seus 2 princípios:

- Não contradição: Uma condição é verdadeira ou falsa, não podendo ser as duas coisas ao mesmo tempo.
- Terceiro excluído: se não for falsa será verdadeira e vice-versa.

desvio Um programa de computador vai executar a primeira instrução, depois a segunda e assim por diante, até o final do programa, a menos que uma instrução seja de desvio: *vá para a instrução x* : Neste caso a seguir será executada a instrução x e não a próxima.

Na década de 70, alguns autores (Dijkstra, Corrado e Jacopini) notaram que o uso de desvios tendia a gerar programas difíceis de *debugar*: Este verbo novo (=depurar) é o processo de identificar, localizar e corrigir erros de programas. Na programação gasta-se 20% do tempo escrevendo código e 80% depurando-o. A programação usando desvios gerou a chamada programação *spaguetti*. Daí os autores sugeriram organizar os desvios: nascia a programação estruturada. Agora os comandos são agrupados em blocos, havendo 3 deles:

sequência 2 ou mais comandos em sequência

alternativa como acima descrito

repetição uma sequência que pode se repetir. Apenas aqui é permitido o desvio e sempre do fim do bloco para o começo.

Fora o caso acima (repetição) o comando de desvio é proibido. Isto é tão verdade que linguagens mais modernas não têm o desvio, como por exemplo a linguagem Python.

Outra característica bem importante da *arquitetura de Von Neumann* é a idéia de programa armazenado: instruções são armazenadas como dados, permitindo que programas processem programas. Esta ideia já havia sido posta por Turing. Nascia o conceito de linguagem de programação, pela qual as complexidades da programação de máquina são transferidas para um programa de computador: o compilador. Uma linguagem de programação é na verdade 2 coisas:

⋆ um conjunto de regras fonéticas, sintáticas e semânticas de como descrever um algoritmo e
⋆ um programa (chamado compilador ou interpretador) capaz de ler o algoritmo acima descrito e gerar o equivalente código de máquina capaz de ser executado em um computador real.

Nascia o $software$ em oposição ao $hardware$. Graças a este conceito, aprender a programar passou de um processo de meses/anos para um de horas/dias. Sem contar que agora o mesmo programa roda em um supercomputador de milhões de dólares e também em um celular android.

Algoritmo O conceito chave em programação é o de *algoritmo*. É o que está por trás de um programa. É a ideia, a receita, a lista dos procedimentos do programa. Se programar é arte e engenharia, o algoritmo responde integralmente pela parte "arte". O conceito nos acompanha faz tempo (o algoritmo de mdc, devido a Euclides tem mais de 2400 anos), mas sua definição precisa é devida ao matemático inglês Alan Turing. Para poder expressar um algoritmo, se necessita uma linguagem apropriada. Vamos descrever uma mini-linguagem agora: As peças elementares de informação serão chamadas variáveis. Cada uma terá um nome, uma palavra qualquer (como aliás já fizemos nas aulas de álgebra em qualquer dos mais de 8 anos de ensino fundamental de matemática).

As operações de E/S serão 2: leia(variável) ou imprima(variável), significando a entrada do valor de uma variável ou sua impressão.

Para designar um valor a uma variável usar-se-á o comando de atribuição, representando por $\leftarrow$, então, por exemplo $A \leftarrow 0$ significa colocar o valor zero na variável A.

As operações aritméticas, seguem 100% do aprendido nas aulas de matemática e são: $+$: adição, $*$: multiplicação, $-$: subtração, $/$: divisão real, $//$: divisão inteira, MOD: resto de, $**$: potência, além de prioridade determinada por parêntesis. Continua proibida a divisão por zero. As linguagens mais modernas já operam normalmente com números complexos, mas podemos deixar esta facilidade de lado (por enquanto).

Para o comando alternativo precisa-se definir uma condição lógica: uma expressão envolvendo condicionais ($>$, $<$, $\leq$, $\geq$, $=$, $\neq$, $\in$, ...) e cuja resposta será um de dois valores: VERDADEIRO ou FALSO. Estes, ainda podem ser combinados usando os conectivos: E ($\wedge$) e OU ($\vee$) além de NÃO ($\sim$). O comando alternativo tem o formato `se <condição> então bloco-A fim{se}` neste comando o bloco de comandos A só será executado se a condição citada for verdadeira. Um formato alternativo é `se <condição> então bloco-A senão bloco-B fim{se}` Agora, se a condição é verdadeira, A é executado (como acima), mas se é falso então o bloco-B é executado.

Finalmente, o desvio não existe, mas em seu lugar entram as estruturas de repetição. Só vamos ver 1 delas. Trata-se do comando `enquanto` que tem formato `enquanto <condição> bloco-A fim{enquanto}`. Note que ele é parecido com o comando alternativo, mas tem uma diferença fundamental: ao encontrar o comando `fim{enquanto}` há um desvio incondicional para o início do bloco ou o `enquanto`.

Um exemplo Vamos escrever um algoritmo para testar se o número X é primo ou não.

```
 1: algoritmo PRIMO
 2: leia(X)
 3: DIVISORES ← 0
 4: TESTADOR ← 2
 5: enquanto TESTADOR < X
 6:     se X MOD TESTADOR = 0
 7:         DIVISORES ← DIVISORES + 1
 8:     fim{se}
 9:     TESTADOR ← TESTADOR + 1
10: fim{enquanto}
11: se DIVISORES = 0
12:     escreva("PRIMO")
13: senão
14:     escreva("NÃO PRIMO")
15: fim{se}
```

Seguir um algoritmo, simulando suas operações é chamado teste de mesa (ou em inglês *walkthrough*), ou mais jocosamente *fazer um chinês*. É o feijão com arroz da programação, usado para entender, certificar e depurar algoritmos. Ao fazer isto com o algoritmo acima, perceba que ele funciona. Se chamado com 15 responderá NÃO PRIMO e se chamado com 13 responderá PRIMO. Programar é escrever estas "coisas". É como aprender um idioma novo. Mas antes de aprender a expressar idéias (algoritmos), há que se ter firmeza na fonética e na síntaxe. A semântica virá depois.

Mais um exemplo Alguns aqui devem conhecer a sequência de Fibonacci: 1,1,2,3,5,8,13,21,34,... Note que, nela, cada número é a soma dos dois anteriores. A sequência começa com 1,1 já que estes não tem dois anteriores. A partir do próximo (o 2), a regra genérica se aplica. Vamos estudar um algoritmo que leia um número n e imprima o n-ésimo número de Fibonacci.

```
1: leia LIMITE
2: A ← 1; B ← 1; N ← 3
3: enquanto N ≤ LIMITE
4:     R ← A + B
5:     A ← B
6:     B ← R
7:     N ← N + 1
8: fim{enquanto}
9: escreva R
```

A seguir, alguns exercícios de síntaxe da linguagem:

☞ Para você fazer

Ex. 1

```
 1: X ← 10
 2: Y ← 31
 3: ALFA ← 2
 4: enquanto X < Y
 5:     X ← X + ALFA
 6:     se ((ALFA mod 3) ≠ 0)
 7:         ALFA ← ALFA + 2
 8:     fim{se}
 9:     ALFA ← ALFA + 1
10: fim{enquanto}
11: escreva X
```

Ex. 2

```
 1: A ← 1
 2: B ← 21
 3: enquanto A ≤ B
 4:     se ((B > 11) ∨ (A ≤ 11))
 5:         B ← B - 3
 6:     fim{se}
 7:     A ← A + 1
 8:     B ← B - 1
 9: fim{enquanto}
10: A ← A - 1
11: escreva (A + B)
```

Ex. 3

```
 1: X ← 4
 2: enquanto X < 6
 3:     Y ← 5
 4:     enquanto Y < 15
 5:         X ← X + 1
 6:         Y ← X + 2
 7:     fim{enquanto}
 8:     Y ← Y + 3
 9:     X ← X + 2
10: fim{enquanto}
11: escreva (X-Y)
```

Ex. 4

```
 1: X ← 2
 2: Y ← 2
 3: se  X < 4 ∧ Y > 5
 4:     X ← X - 3
 5: senão
 6:     Y ← Y + 3
 7:     se X mod 3 ≠ 0
 8:         Y ← Y + 3
 9:     fim{se}
10: fim{se}
11: escreva X+Y
```

Responda aqui os valores encontrados.

1	2	3	4

UFPR - UP - UTFPR - PUCPr – 24/04/2024 - 11:42:12.5
Prof Dr P Kantek (pkantek@gmail.com)
Aquecimento em programação VIVX001a, V: 2.06
75927 MATHEUS AITA FABRICIO DE CAR
24PRG101 - 17 _____ / _____ / _____

Programar é ... Já se disse com alguma razão que programar é emprestar um pouco de inteligência ao computador. Então, antes de começar, reconheça-se a nobreza do que se está a fazer. Os computadores (todos) descendem de duas propostas mais ou menos complementares: a primeira, descrita brilhantemente por A. Turing em 1936 e a segunda igualmente brilhante, por Von Neumann em começos da década de 50. Pela primeira, sabe-se que o computador é capaz de executar uma instrução após a outra. Tais instruções precisam ser simples e inequívocas. Pela segunda descobre-se que tais instruções são bastante limitadas: E/S, aritmética, movimentação, alternativa e desvio.

E/S Este tipo de instrução responde pela E=entrada e S=saída de dados do/para o computador. Suponha que você construiu uma máquina capaz de descobrir o seno de qualquer ângulo. Para pôr a máquina para funcionar, você precisa fornecer algum ângulo a ela e depois que ela fizer seu trabalho ela vai precisar informar a você o valor do seno. Estas duas etapas correspondem a operações de E/S. Do ponto de vista teórico da programação, são operações quase irrelevantes, embora do ponto de vista prático sejam fundamentais.

aritmética Esta família é muito importante, mas graças a Deus, já estudamos pelo menos 8 anos de matemática. Tudo o que se aprendeu lá (adição, produto, potência, paridade, regra de sinais, números inteiros e reais, positivos e negativos, propriedades associativa, comutativa, distributiva – prioridades e parêntesis, etc etc, etc, etc) continua válido.

movimentação Operação muito simples, pela qual, partes da memória, chamadas variáveis são movimentadas ou mais propriamente copiadas entre sí.

alternativa Aqui a instrução mais "nobre" da programação. É graças a ela que um computador se torna diferente de um liquidificador. Se uma condição for boa (verdadeira) executar-se-á alguma coisa. Se a condição for ruim (falsa) executar-se-á outra coisa, ou em certos casos não se executará nada. Vale aqui a lógica binária com seus 2 princípios:

- Não contradição: Uma condição é verdadeira ou falsa, não podendo ser as duas coisas ao mesmo tempo.
- Terceiro excluído: se não for falsa será verdadeira e vice-versa.

desvio Um programa de computador vai executar a primeira instrução, depois a segunda e assim por diante, até o final do programa, a menos que uma instrução seja de desvio: *vá para a instrução x* : Neste caso a seguir será executada a instrução x e não a próxima.

Na década de 70, alguns autores (Dijkstra, Corrado e Jacopini) notaram que o uso de desvios tendia a gerar programas difíceis de *debugar*: Este verbo novo (=depurar) é o processo de identificar, localizar e corrigir erros de programas. Na programação gasta-se 20% do tempo escrevendo código e 80% depurando-o. A programação usando desvios gerou a chamada programação *spaguetti*. Daí os autores sugeriram organizar os desvios: nascia a programação estruturada. Agora os comandos são agrupados em blocos, havendo 3 deles:

sequência 2 ou mais comandos em sequência

alternativa como acima descrito

repetição uma sequência que pode se repetir. Apenas aqui é permitido o desvio e sempre do fim do bloco para o começo.

Fora o caso acima (repetição) o comando de desvio é proibido. Isto é tão verdade que linguagens mais modernas não têm o desvio, como por exemplo a linguagem Python.

Outra característica bem importante da *arquitetura de Von Neumann* é a idéia de programa armazenado: instruções são armazenadas como dados, permitindo que programas processem programas. Esta ideia já havia sido posta por Turing. Nascia o conceito de linguagem de programação, pela qual as complexidades da programação de máquina são transferidas para um programa de computador: o compilador. Uma linguagem de programação é na verdade 2 coisas:

⋆ um conjunto de regras fonéticas, sintáticas e semânticas de como descrever um algoritmo e
⋆ um programa (chamado compilador ou interpretador) capaz de ler o algoritmo acima descrito e gerar o equivalente código de máquina capaz de ser executado em um computador real.

Nascia o $software$ em oposição ao $hardware$. Graças a este conceito, aprender a programar passou de um processo de meses/anos para um de horas/dias. Sem contar que agora o mesmo programa roda em um supercomputador de milhões de dólares e também em um celular android.

Algoritmo O conceito chave em programação é o de *algoritmo*. É o que está por trás de um programa. É a ideia, a receita, a lista dos procedimentos do programa. Se programar é arte e engenharia, o algoritmo responde integralmente pela parte "arte". O conceito nos acompanha faz tempo (o algoritmo de mdc, devido a Euclides tem mais de 2400 anos), mas sua definição precisa é devida ao matemático inglês Alan Turing. Para poder expressar um algoritmo, se necessita uma linguagem apropriada. Vamos descrever uma mini-linguagem agora: As peças elementares de informação serão chamadas variáveis. Cada uma terá um nome, uma palavra qualquer (como aliás já fizemos nas aulas de álgebra em qualquer dos mais de 8 anos de ensino fundamental de matemática).

As operações de E/S serão 2: leia(variável) ou imprima(variável), significando a entrada do valor de uma variável ou sua impressão.

Para designar um valor a uma variável usar-se-á o comando de atribuição, representando por $\leftarrow$, então, por exemplo $A \leftarrow 0$ significa colocar o valor zero na variável A.

As operações aritméticas, seguem 100% do aprendido nas aulas de matemática e são: $+$: adição, $*$: multiplicação, $-$: subtração, $/$: divisão real, $//$: divisão inteira, MOD: resto de, $**$: potência, além de prioridade determinada por parêntesis. Continua proibida a divisão por zero. As linguagens mais modernas já operam normalmente com números complexos, mas podemos deixar esta facilidade de lado (por enquanto).

Para o comando alternativo precisa-se definir uma condição lógica: uma expressão envolvendo condicionais ($>$, $<$, $\leq$, $\geq$, $=$, $\neq$, $\in$, ...) e cuja resposta será um de dois valores: VERDADEIRO ou FALSO. Estes, ainda podem ser combinados usando os conectivos: E ($\wedge$) e OU ($\vee$) além de NÃO ($\sim$). O comando alternativo tem o formato `se <condição> então bloco-A fim{se}` neste comando o bloco de comandos A só será executado se a condição citada for verdadeira. Um formato alternativo é `se <condição> então bloco-A senão bloco-B fim{se}` Agora, se a condição é verdadeira, A é executado (como acima), mas se é falso então o bloco-B é executado.

Finalmente, o desvio não existe, mas em seu lugar entram as estruturas de repetição. Só vamos ver 1 delas. Trata-se do comando `enquanto` que tem formato `enquanto <condição> bloco-A fim{enquanto}`. Note que ele é parecido com o comando alternativo, mas tem uma diferença fundamental: ao encontrar o comando `fim{enquanto}` há um desvio incondicional para o início do bloco ou o `enquanto`.

Um exemplo Vamos escrever um algoritmo para testar se o número X é primo ou não.

```
 1: algoritmo PRIMO
 2: leia(X)
 3: DIVISORES ← 0
 4: TESTADOR ← 2
 5: enquanto TESTADOR < X
 6:     se X MOD TESTADOR = 0
 7:         DIVISORES ← DIVISORES + 1
 8:     fim{se}
 9:     TESTADOR ← TESTADOR + 1
10: fim{enquanto}
11: se DIVISORES = 0
12:     escreva("PRIMO")
13: senão
14:     escreva("NÃO PRIMO")
15: fim{se}
```

Seguir um algoritmo, simulando suas operações é chamado teste de mesa (ou em inglês *walkthrough*), ou mais jocosamente *fazer um chinês*. É o feijão com arroz da programação, usado para entender, certificar e depurar algoritmos. Ao fazer isto com o algoritmo acima, perceba que ele funciona. Se chamado com 15 responderá NÃO PRIMO e se chamado com 13 responderá PRIMO. Programar é escrever estas "coisas". É como aprender um idioma novo. Mas antes de aprender a expressar idéias (algoritmos), há que se ter firmeza na fonética e na síntaxe. A semântica virá depois.

Mais um exemplo Alguns aqui devem conhecer a sequência de Fibonacci: 1,1,2,3,5,8,13,21,34,... Note que, nela, cada número é a soma dos dois anteriores. A sequência começa com 1,1 já que estes não tem dois anteriores. A partir do próximo (o 2), a regra genérica se aplica. Vamos estudar um algoritmo que leia um número n e imprima o n-ésimo número de Fibonacci.

```
1: leia LIMITE
2: A ← 1; B ← 1; N ← 3
3: enquanto N ≤ LIMITE
4:     R ← A + B
5:     A ← B
6:     B ← R
7:     N ← N + 1
8: fim{enquanto}
9: escreva R
```

A seguir, alguns exercícios de síntaxe da linguagem:

☞ Para você fazer

Ex. 1

```
 1: X ← 10
 2: Y ← 40
 3: BETA ← 4
 4: enquanto X < Y
 5:     X ← X + BETA
 6:     se ((BETA mod 3) ≠ 0)
 7:         BETA ← BETA + 2
 8:     fim{se}
 9:     BETA ← BETA + 1
10: fim{enquanto}
11: escreva X
```

Ex. 2

```
 1: A ← 2
 2: B ← 22
 3: enquanto A ≤ B
 4:     se ((B > 7) ∨ (A ≤ 13))
 5:         B ← B - 1
 6:     fim{se}
 7:     A ← A + 3
 8:     B ← B - 2
 9: fim{enquanto}
10: A ← A - 1
11: escreva (A + B)
```

Ex. 3

```
 1: X ← 1
 2: enquanto X < 13
 3:     Y ← 6
 4:     enquanto Y < 13
 5:         X ← X + 1
 6:         Y ← X + 2
 7:     fim{enquanto}
 8:     Y ← Y + 1
 9:     X ← X + 1
10: fim{enquanto}
11: escreva (X-Y)
```

Ex. 4

```
 1: X ← 5
 2: Y ← 5
 3: se  X < 5 ∧ Y > 3
 4:     X ← X - 3
 5: senão
 6:     Y ← Y + 3
 7:     se X mod 2 ≠ 0
 8:         Y ← Y + 1
 9:     fim{se}
10: fim{se}
11: escreva X+Y
```

Responda aqui os valores encontrados.

1	2	3	4

UFPR - UP - UTFPR - PUCPr – 24/04/2024 - 11:42:12.5
Prof Dr P Kantek (pkantek@gmail.com)
Aquecimento em programação VIVX001a, V: 2.06
75934 MATHEUS LEANDRO DE BITENCOUR
24PRG101 - 18 _____ / _____ / _____

Programar é ... Já se disse com alguma razão que programar é emprestar um pouco de inteligência ao computador. Então, antes de começar, reconheça-se a nobreza do que se está a fazer. Os computadores (todos) descendem de duas propostas mais ou menos complementares: a primeira, descrita brilhantemente por A. Turing em 1936 e a segunda igualmente brilhante, por Von Neumann em começos da década de 50. Pela primeira, sabe-se que o computador é capaz de executar uma instrução após a outra. Tais instruções precisam ser simples e inequívocas. Pela segunda descobre-se que tais instruções são bastante limitadas: E/S, aritmética, movimentação, alternativa e desvio.

E/S Este tipo de instrução responde pela E=entrada e S=saída de dados do/para o computador. Suponha que você construiu uma máquina capaz de descobrir o seno de qualquer ângulo. Para pôr a máquina para funcionar, você precisa fornecer algum ângulo a ela e depois que ela fizer seu trabalho ela vai precisar informar a você o valor do seno. Estas duas etapas correspondem a operações de E/S. Do ponto de vista teórico da programação, são operações quase irrelevantes, embora do ponto de vista prático sejam fundamentais.

aritmética Esta família é muito importante, mas graças a Deus, já estudamos pelo menos 8 anos de matemática. Tudo o que se aprendeu lá (adição, produto, potência, paridade, regra de sinais, números inteiros e reais, positivos e negativos, propriedades associativa, comutativa, distributiva – prioridades e parêntesis, etc etc, etc, etc) continua válido.

movimentação Operação muito simples, pela qual, partes da memória, chamadas variáveis são movimentadas ou mais propriamente copiadas entre sí.

alternativa Aqui a instrução mais "nobre" da programação. É graças a ela que um computador se torna diferente de um liquidificador. Se uma condição for boa (verdadeira) executar-se-á alguma coisa. Se a condição for ruim (falsa) executar-se-á outra coisa, ou em certos casos não se executará nada. Vale aqui a lógica binária com seus 2 princípios:

- Não contradição: Uma condição é verdadeira ou falsa, não podendo ser as duas coisas ao mesmo tempo.
- Terceiro excluído: se não for falsa será verdadeira e vice-versa.

desvio Um programa de computador vai executar a primeira instrução, depois a segunda e assim por diante, até o final do programa, a menos que uma instrução seja de desvio: *vá para a instrução x* : Neste caso a seguir será executada a instrução x e não a próxima.

Na década de 70, alguns autores (Dijkstra, Corrado e Jacopini) notaram que o uso de desvios tendia a gerar programas difíceis de *debugar*: Este verbo novo (=depurar) é o processo de identificar, localizar e corrigir erros de programas. Na programação gasta-se 20% do tempo escrevendo código e 80% depurando-o. A programação usando desvios gerou a chamada programação *spaguetti*. Daí os autores sugeriram organizar os desvios: nascia a programação estruturada. Agora os comandos são agrupados em blocos, havendo 3 deles:

sequência 2 ou mais comandos em sequência

alternativa como acima descrito

repetição uma sequência que pode se repetir. Apenas aqui é permitido o desvio e sempre do fim do bloco para o começo.

Fora o caso acima (repetição) o comando de desvio é proibido. Isto é tão verdade que linguagens mais modernas não têm o desvio, como por exemplo a linguagem Python.

Outra característica bem importante da *arquitetura de Von Neumann* é a idéia de programa armazenado: instruções são armazenadas como dados, permitindo que programas processem programas. Esta ideia já havia sido posta por Turing. Nascia o conceito de linguagem de programação, pela qual as complexidades da programação de máquina são transferidas para um programa de computador: o compilador. Uma linguagem de programação é na verdade 2 coisas:

⋆ um conjunto de regras fonéticas, sintáticas e semânticas de como descrever um algoritmo e
⋆ um programa (chamado compilador ou interpretador) capaz de ler o algoritmo acima descrito e gerar o equivalente código de máquina capaz de ser executado em um computador real.

Nascia o $software$ em oposição ao $hardware$. Graças a este conceito, aprender a programar passou de um processo de meses/anos para um de horas/dias. Sem contar que agora o mesmo programa roda em um supercomputador de milhões de dólares e também em um celular android.

Algoritmo O conceito chave em programação é o de *algoritmo*. É o que está por trás de um programa. É a ideia, a receita, a lista dos procedimentos do programa. Se programar é arte e engenharia, o algoritmo responde integralmente pela parte "arte". O conceito nos acompanha faz tempo (o algoritmo de mdc, devido a Euclides tem mais de 2400 anos), mas sua definição precisa é devida ao matemático inglês Alan Turing. Para poder expressar um algoritmo, se necessita uma linguagem apropriada. Vamos descrever uma mini-linguagem agora: As peças elementares de informação serão chamadas variáveis. Cada uma terá um nome, uma palavra qualquer (como aliás já fizemos nas aulas de álgebra em qualquer dos mais de 8 anos de ensino fundamental de matemática).

As operações de E/S serão 2: leia(variável) ou imprima(variável), significando a entrada do valor de uma variável ou sua impressão.

Para designar um valor a uma variável usar-se-á o comando de atribuição, representando por $\leftarrow$, então, por exemplo $A \leftarrow 0$ significa colocar o valor zero na variável A.

As operações aritméticas, seguem 100% do aprendido nas aulas de matemática e são: $+$: adição, $*$: multiplicação, $-$: subtração, $/$: divisão real, $//$: divisão inteira, MOD: resto de, $**$: potência, além de prioridade determinada por parêntesis. Continua proibida a divisão por zero. As linguagens mais modernas já operam normalmente com números complexos, mas podemos deixar esta facilidade de lado (por enquanto).

Para o comando alternativo precisa-se definir uma condição lógica: uma expressão envolvendo condicionais ($>$, $<$, $\leq$, $\geq$, $=$, $\neq$, $\in$, ...) e cuja resposta será um de dois valores: VERDADEIRO ou FALSO. Estes, ainda podem ser combinados usando os conectivos: E ($\wedge$) e OU ($\vee$) além de NÃO ($\sim$). O comando alternativo tem o formato `se <condição> então bloco-A fim{se}` neste comando o bloco de comandos A só será executado se a condição citada for verdadeira. Um formato alternativo é `se <condição> então bloco-A senão bloco-B fim{se}` Agora, se a condição é verdadeira, A é executado (como acima), mas se é falso então o bloco-B é executado.

Finalmente, o desvio não existe, mas em seu lugar entram as estruturas de repetição. Só vamos ver 1 delas. Trata-se do comando `enquanto` que tem formato `enquanto <condição> bloco-A fim{enquanto}`. Note que ele é parecido com o comando alternativo, mas tem uma diferença fundamental: ao encontrar o comando `fim{enquanto}` há um desvio incondicional para o início do bloco ou o `enquanto`.

Um exemplo Vamos escrever um algoritmo para testar se o número X é primo ou não.

```
 1: algoritmo PRIMO
 2: leia(X)
 3: DIVISORES ← 0
 4: TESTADOR ← 2
 5: enquanto TESTADOR < X
 6:     se X MOD TESTADOR = 0
 7:         DIVISORES ← DIVISORES + 1
 8:     fim{se}
 9:     TESTADOR ← TESTADOR + 1
10: fim{enquanto}
11: se DIVISORES = 0
12:     escreva("PRIMO")
13: senão
14:     escreva("NÃO PRIMO")
15: fim{se}
```

Seguir um algoritmo, simulando suas operações é chamado teste de mesa (ou em inglês *walkthrough*), ou mais jocosamente *fazer um chinês*. É o feijão com arroz da programação, usado para entender, certificar e depurar algoritmos. Ao fazer isto com o algoritmo acima, perceba que ele funciona. Se chamado com 15 responderá NÃO PRIMO e se chamado com 13 responderá PRIMO. Programar é escrever estas "coisas". É como aprender um idioma novo. Mas antes de aprender a expressar idéias (algoritmos), há que se ter firmeza na fonética e na síntaxe. A semântica virá depois.

Mais um exemplo Alguns aqui devem conhecer a sequência de Fibonacci: 1,1,2,3,5,8,13,21,34,... Note que, nela, cada número é a soma dos dois anteriores. A sequência começa com 1,1 já que estes não tem dois anteriores. A partir do próximo (o 2), a regra genérica se aplica. Vamos estudar um algoritmo que leia um número n e imprima o n-ésimo número de Fibonacci.

```
 1: leia LIMITE
 2: A ← 1; B ← 1; N ← 3
 3: enquanto N ≤ LIMITE
 4:     R ← A + B
 5:     A ← B
 6:     B ← R
 7:     N ← N + 1
 8: fim{enquanto}
 9: escreva R
```

A seguir, alguns exercícios de síntaxe da linguagem:

☞ Para você fazer

Ex. 1

```
 1: X ← 9
 2: Y ← 42
 3: GAMA ← 3
 4: enquanto X < Y
 5:     X ← X + GAMA
 6:     se ((GAMA mod 3) ≠ 0)
 7:         GAMA ← GAMA + 2
 8:     fim{se}
 9:     GAMA ← GAMA + 1
10: fim{enquanto}
11: escreva X
```

Ex. 2

```
 1: A ← 3
 2: B ← 21
 3: enquanto A ≤ B
 4:     se ((B > 13) ∨ (A ≤ 12))
 5:         B ← B - 1
 6:     fim{se}
 7:     A ← A + 3
 8:     B ← B - 2
 9: fim{enquanto}
10: A ← A - 2
11: escreva (A + B)
```

Ex. 3

```
 1: X ← 2
 2: enquanto X < 7
 3:     Y ← 3
 4:     enquanto Y < 11
 5:         X ← X + 2
 6:         Y ← X + 1
 7:     fim{enquanto}
 8:     Y ← Y + 2
 9:     X ← X + 2
10: fim{enquanto}
11: escreva (X-Y)
```

Ex. 4

```
 1: X ← 8
 2: Y ← 2
 3: se  X < 4 ∧ Y > 5
 4:     X ← X - 3
 5: senão
 6:     Y ← Y + 3
 7:     se X mod 3 ≠ 0
 8:         Y ← Y + 2
 9:     fim{se}
10: fim{se}
11: escreva X+Y
```

Responda aqui os valores encontrados.

1	2	3	4

UFPR - UP - UTFPR - PUCPr – 24/04/2024 - 11:42:12.5
Prof Dr P Kantek (pkantek@gmail.com)
Aquecimento em programação VIVX001a, V: 2.06
75941 MAYCON TAVARES WOLPE 24PRG101 - 19 _____ / _____ / _____

Programar é ... Já se disse com alguma razão que programar é emprestar um pouco de inteligência ao computador. Então, antes de começar, reconheça-se a nobreza do que se está a fazer. Os computadores (todos) descendem de duas propostas mais ou menos complementares: a primeira, descrita brilhantemente por A. Turing em 1936 e a segunda igualmente brilhante, por Von Neumann em começos da década de 50. Pela primeira, sabe-se que o computador é capaz de executar uma instrução após a outra. Tais instruções precisam ser simples e inequívocas. Pela segunda descobre-se que tais instruções são bastante limitadas: E/S, aritmética, movimentação, alternativa e desvio.

E/S Este tipo de instrução responde pela E=entrada e S=saída de dados do/para o computador. Suponha que você construiu uma máquina capaz de descobrir o seno de qualquer ângulo. Para pôr a máquina para funcionar, você precisa fornecer algum ângulo a ela e depois que ela fizer seu trabalho ela vai precisar informar a você o valor do seno. Estas duas etapas correspondem a operações de E/S. Do ponto de vista teórico da programação, são operações quase irrelevantes, embora do ponto de vista prático sejam fundamentais.

aritmética Esta família é muito importante, mas graças a Deus, já estudamos pelo menos 8 anos de matemática. Tudo o que se aprendeu lá (adição, produto, potência, paridade, regra de sinais, números inteiros e reais, positivos e negativos, propriedades associativa, comutativa, distributiva – prioridades e parêntesis, etc etc, etc, etc) continua válido.

movimentação Operação muito simples, pela qual, partes da memória, chamadas variáveis são movimentadas ou mais propriamente copiadas entre sí.

alternativa Aqui a instrução mais "nobre" da programação. É graças a ela que um computador se torna diferente de um liquidificador. Se uma condição for boa (verdadeira) executar-se-á alguma coisa. Se a condição for ruim (falsa) executar-se-á outra coisa, ou em certos casos não se executará nada. Vale aqui a lógica binária com seus 2 princípios:

- Não contradição: Uma condição é verdadeira ou falsa, não podendo ser as duas coisas ao mesmo tempo.
- Terceiro excluído: se não for falsa será verdadeira e vice-versa.

desvio Um programa de computador vai executar a primeira instrução, depois a segunda e assim por diante, até o final do programa, a menos que uma instrução seja de desvio: *vá para a instrução x* : Neste caso a seguir será executada a instrução x e não a próxima.

Na década de 70, alguns autores (Dijkstra, Corrado e Jacopini) notaram que o uso de desvios tendia a gerar programas difíceis de *debugar*: Este verbo novo (=depurar) é o processo de identificar, localizar e corrigir erros de programas. Na programação gasta-se 20% do tempo escrevendo código e 80% depurando-o. A programação usando desvios gerou a chamada programação *spaguetti*. Daí os autores sugeriram organizar os desvios: nascia a programação estruturada. Agora os comandos são agrupados em blocos, havendo 3 deles:

sequência 2 ou mais comandos em sequência

alternativa como acima descrito

repetição uma sequência que pode se repetir. Apenas aqui é permitido o desvio e sempre do fim do bloco para o começo.

Fora o caso acima (repetição) o comando de desvio é proibido. Isto é tão verdade que linguagens mais modernas não têm o desvio, como por exemplo a linguagem Python.

Outra característica bem importante da *arquitetura de Von Neumann* é a idéia de programa armazenado: instruções são armazenadas como dados, permitindo que programas processem programas. Esta ideia já havia sido posta por Turing. Nascia o conceito de linguagem de programação, pela qual as complexidades da programação de máquina são transferidas para um programa de computador: o compilador. Uma linguagem de programação é na verdade 2 coisas:

⋆ um conjunto de regras fonéticas, sintáticas e semânticas de como descrever um algoritmo e
⋆ um programa (chamado compilador ou interpretador) capaz de ler o algoritmo acima descrito e gerar o equivalente código de máquina capaz de ser executado em um computador real.

Nascia o $software$ em oposição ao $hardware$. Graças a este conceito, aprender a programar passou de um processo de meses/anos para um de horas/dias. Sem contar que agora o mesmo programa roda em um supercomputador de milhões de dólares e também em um celular android.

Algoritmo O conceito chave em programação é o de *algoritmo*. É o que está por trás de um programa. É a ideia, a receita, a lista dos procedimentos do programa. Se programar é arte e engenharia, o algoritmo responde integralmente pela parte "arte". O conceito nos acompanha faz tempo (o algoritmo de mdc, devido a Euclides tem mais de 2400 anos), mas sua definição precisa é devida ao matemático inglês Alan Turing. Para poder expressar um algoritmo, se necessita uma linguagem apropriada. Vamos descrever uma mini-linguagem agora: As peças elementares de informação serão chamadas variáveis. Cada uma terá um nome, uma palavra qualquer (como aliás já fizemos nas aulas de álgebra em qualquer dos mais de 8 anos de ensino fundamental de matemática).

As operações de E/S serão 2: leia(variável) ou imprima(variável), significando a entrada do valor de uma variável ou sua impressão.

Para designar um valor a uma variável usar-se-á o comando de atribuição, representando por $\leftarrow$, então, por exemplo $A \leftarrow 0$ significa colocar o valor zero na variável A.

As operações aritméticas, seguem 100% do aprendido nas aulas de matemática e são: +: adição, $*$: multiplicação, $-$: subtração, /: divisão real, //: divisão inteira, MOD: resto de, $**$: potência, além de prioridade determinada por parêntesis. Continua proibida a divisão por zero. As linguagens mais modernas já operam normalmente com números complexos, mas podemos deixar esta facilidade de lado (por enquanto).

Para o comando alternativo precisa-se definir uma condição lógica: uma expressão envolvendo condicionais ($>$, $<$, $\leq$, $\geq$, $=$, $\neq$, $\in$, ...) e cuja resposta será um de dois valores: VERDADEIRO ou FALSO. Estes, ainda podem ser combinados usando os conectivos: E ($\wedge$) e OU ($\vee$) além de NÃO ($\sim$). O comando alternativo tem o formato `se <condição> então bloco-A fim{se}` neste comando o bloco de comandos A só será executado se a condição citada for verdadeira. Um formato alternativo é `se <condição> então bloco-A senão bloco-B fim{se}` Agora, se a condição é verdadeira, A é executado (como acima), mas se é falso então o bloco-B é executado.

Finalmente, o desvio não existe, mas em seu lugar entram as estruturas de repetição. Só vamos ver 1 delas. Trata-se do comando `enquanto` que tem formato `enquanto <condição> bloco-A fim{enquanto}`. Note que ele é parecido com o comando alternativo, mas tem uma diferença fundamental: ao encontrar o comando `fim{enquanto}` há um desvio incondicional para o início do bloco ou o `enquanto`.

Um exemplo Vamos escrever um algoritmo para testar se o número X é primo ou não.

```
 1: algoritmo PRIMO
 2: leia(X)
 3: DIVISORES ← 0
 4: TESTADOR ← 2
 5: enquanto TESTADOR < X
 6:     se X MOD TESTADOR = 0
 7:         DIVISORES ← DIVISORES + 1
 8:     fim{se}
 9:     TESTADOR ← TESTADOR + 1
10: fim{enquanto}
11: se DIVISORES = 0
12:     escreva("PRIMO")
13: senão
14:     escreva("NÃO PRIMO")
15: fim{se}
```

Seguir um algoritmo, simulando suas operações é chamado teste de mesa (ou em inglês *walkthrough*), ou mais jocosamente *fazer um chinês*. É o feijão com arroz da programação, usado para entender, certificar e depurar algoritmos. Ao fazer isto com o algoritmo acima, perceba que ele funciona. Se chamado com 15 responderá NÃO PRIMO e se chamado com 13 responderá PRIMO. Programar é escrever estas "coisas". É como aprender um idioma novo. Mas antes de aprender a expressar idéias (algoritmos), há que se ter firmeza na fonética e na síntaxe. A semântica virá depois.

Mais um exemplo Alguns aqui devem conhecer a sequência de Fibonacci: 1,1,2,3,5,8,13,21,34,... Note que, nela, cada número é a soma dos dois anteriores. A sequência começa com 1,1 já que estes não tem dois anteriores. A partir do próximo (o 2), a regra genérica se aplica. Vamos estudar um algoritmo que leia um número n e imprima o n-ésimo número de Fibonacci.

```
 1: leia LIMITE
 2: A ← 1; B ← 1; N ← 3
 3: enquanto N ≤ LIMITE
 4:     R ← A + B
 5:     A ← B
 6:     B ← R
 7:     N ← N + 1
 8: fim{enquanto}
 9: escreva R
```

A seguir, alguns exercícios de síntaxe da linguagem:

☞ Para você fazer

Ex. 1

```
 1: X ← 10
 2: Y ← 31
 3: CTDS ← 4
 4: enquanto X < Y
 5:     X ← X + CTDS
 6:     se ((CTDS mod 3) ≠ 0)
 7:         CTDS ← CTDS + 1
 8:     fim{se}
 9:     CTDS ← CTDS + 1
10: fim{enquanto}
11: escreva X
```

Ex. 2

```
 1: A ← 1
 2: B ← 22
 3: enquanto A ≤ B
 4:     se ((B > 11) ∨ (A ≤ 15))
 5:         B ← B - 2
 6:     fim{se}
 7:     A ← A + 3
 8:     B ← B - 2
 9: fim{enquanto}
10: A ← A - 3
11: escreva (A + B)
```

Ex. 3

```
 1: X ← 1
 2: enquanto X < 10
 3:     Y ← 5
 4:     enquanto Y < 15
 5:         X ← X + 3
 6:         Y ← X + 1
 7:     fim{enquanto}
 8:     Y ← Y + 3
 9:     X ← X + 2
10: fim{enquanto}
11: escreva (X-Y)
```

Ex. 4

```
 1: X ← 9
 2: Y ← 2
 3: se  X < 4 ∧ Y > 3
 4:     X ← X - 2
 5: senão
 6:     Y ← Y + 3
 7:     se X mod 4 ≠ 0
 8:         Y ← Y + 1
 9:     fim{se}
10: fim{se}
11: escreva X+Y
```

Responda aqui os valores encontrados.

1	2	3	4

UFPR - UP - UTFPR - PUCPr – 24/04/2024 - 11:42:12.5
Prof Dr P Kantek (pkantek@gmail.com)
Aquecimento em programação VIVX001a, V: 2.06
75958 PEDRO TOLEDO LEAL 24PRG101 - 20
/ _____ / _____

Programar é ... Já se disse com alguma razão que programar é emprestar um pouco de inteligência ao computador. Então, antes de começar, reconheça-se a nobreza do que se está a fazer. Os computadores (todos) descendem de duas propostas mais ou menos complementares: a primeira, descrita brilhantemente por A. Turing em 1936 e a segunda igualmente brilhante, por Von Neumann em começos da década de 50. Pela primeira, sabe-se que o computador é capaz de executar uma instrução após a outra. Tais instruções precisam ser simples e inequívocas. Pela segunda descobre-se que tais instruções são bastante limitadas: E/S, aritmética, movimentação, alternativa e desvio.

E/S Este tipo de instrução responde pela E=entrada e S=saída de dados do/para o computador. Suponha que você construiu uma máquina capaz de descobrir o seno de qualquer ângulo. Para pôr a máquina para funcionar, você precisa fornecer algum ângulo a ela e depois que ela fizer seu trabalho ela vai precisar informar a você o valor do seno. Estas duas etapas correspondem a operações de E/S. Do ponto de vista teórico da programação, são operações quase irrelevantes, embora do ponto de vista prático sejam fundamentais.

aritmética Esta família é muito importante, mas graças a Deus, já estudamos pelo menos 8 anos de matemática. Tudo o que se aprendeu lá (adição, produto, potência, paridade, regra de sinais, números inteiros e reais, positivos e negativos, propriedades associativa, comutativa, distributiva – prioridades e parêntesis, etc etc, etc, etc) continua válido.

movimentação Operação muito simples, pela qual, partes da memória, chamadas variáveis são movimentadas ou mais propriamente copiadas entre sí.

alternativa Aqui a instrução mais "nobre" da programação. É graças a ela que um computador se torna diferente de um liquidificador. Se uma condição for boa (verdadeira) executar-se-á alguma coisa. Se a condição for ruim (falsa) executar-se-á outra coisa, ou em certos casos não se executará nada. Vale aqui a lógica binária com seus 2 princípios:

- Não contradição: Uma condição é verdadeira ou falsa, não podendo ser as duas coisas ao mesmo tempo.
- Terceiro excluído: se não for falsa será verdadeira e vice-versa.

desvio Um programa de computador vai executar a primeira instrução, depois a segunda e assim por diante, até o final do programa, a menos que uma instrução seja de desvio: *vá para a instrução x* : Neste caso a seguir será executada a instrução x e não a próxima.

Na década de 70, alguns autores (Dijkstra, Corrado e Jacopini) notaram que o uso de desvios tendia a gerar programas difíceis de *debugar*: Este verbo novo (=depurar) é o processo de identificar, localizar e corrigir erros de programas. Na programação gasta-se 20% do tempo escrevendo código e 80% depurando-o. A programação usando desvios gerou a chamada programação *spaguetti*. Daí os autores sugeriram organizar os desvios: nascia a programação estruturada. Agora os comandos são agrupados em blocos, havendo 3 deles:

sequência 2 ou mais comandos em sequência

alternativa como acima descrito

repetição uma sequência que pode se repetir. Apenas aqui é permitido o desvio e sempre do fim do bloco para o começo.

Fora o caso acima (repetição) o comando de desvio é proibido. Isto é tão verdade que linguagens mais modernas não têm o desvio, como por exemplo a linguagem Python.

Outra característica bem importante da *arquitetura de Von Neumann* é a idéia de programa armazenado: instruções são armazenadas como dados, permitindo que programas processem programas. Esta ideia já havia sido posta por Turing. Nascia o conceito de linguagem de programação, pela qual as complexidades da programação de máquina são transferidas para um programa de computador: o compilador. Uma linguagem de programação é na verdade 2 coisas:

⋆ um conjunto de regras fonéticas, sintáticas e semânticas de como descrever um algoritmo e
⋆ um programa (chamado compilador ou interpretador) capaz de ler o algoritmo acima descrito e gerar o equivalente código de máquina capaz de ser executado em um computador real.

Nascia o *software* em oposição ao *hardware*. Graças a este conceito, aprender a programar passou de um processo de meses/anos para um de horas/dias. Sem contar que agora o mesmo programa roda em um supercomputador de milhões de dólares e também em um celular android.

Algoritmo O conceito chave em programação é o de *algoritmo*. É o que está por trás de um programa. É a ideia, a receita, a lista dos procedimentos do programa. Se programar é arte e engenharia, o algoritmo responde integralmente pela parte "arte". O conceito nos acompanha faz tempo (o algoritmo de mdc, devido a Euclides tem mais de 2400 anos), mas sua definição precisa é devida ao matemático inglês Alan Turing. Para poder expressar um algoritmo, se necessita uma linguagem apropriada. Vamos descrever uma mini-linguagem agora: As peças elementares de informação serão chamadas variáveis. Cada uma terá um nome, uma palavra qualquer (como aliás já fizemos nas aulas de álgebra em qualquer dos mais de 8 anos de ensino fundamental de matemática).

As operações de E/S serão 2: leia(variável) ou imprima(variável), significando a entrada do valor de uma variável ou sua impressão.

Para designar um valor a uma variável usar-se-á o comando de atribuição, representando por $\leftarrow$, então, por exemplo $A \leftarrow 0$ significa colocar o valor zero na variável A.

As operações aritméticas, seguem 100% do aprendido nas aulas de matemática e são: +: adição, $*$: multiplicação, $-$: subtração, /: divisão real, //: divisão inteira, MOD: resto de, $**$: potência, além de prioridade determinada por parêntesis. Continua proibida a divisão por zero. As linguagens mais modernas já operam normalmente com números complexos, mas podemos deixar esta facilidade de lado (por enquanto).

Para o comando alternativo precisa-se definir uma condição lógica: uma expressão envolvendo condicionais ($>$, $<$, $\leq$, $\geq$, $=$, $\neq$, $\in$, ...) e cuja resposta será um de dois valores: VERDADEIRO ou FALSO. Estes, ainda podem ser combinados usando os conectivos: E ($\wedge$) e OU ($\vee$) além de NÃO ($\sim$). O comando alternativo tem o formato `se <condição> então bloco-A fim{se}` neste comando o bloco de comandos A só será executado se a condição citada for verdadeira. Um formato alternativo é `se <condição> então bloco-A senão bloco-B fim{se}` Agora, se a condição é verdadeira, A é executado (como acima), mas se é falso então o bloco-B é executado.

Finalmente, o desvio não existe, mas em seu lugar entram as estruturas de repetição. Só vamos ver 1 delas. Trata-se do comando `enquanto` que tem formato `enquanto <condição> bloco-A fim{enquanto}`. Note que ele é parecido com o comando alternativo, mas tem uma diferença fundamental: ao encontrar o comando `fim{enquanto}` há um desvio incondicional para o início do bloco ou o `enquanto`.

Um exemplo Vamos escrever um algoritmo para testar se o número X é primo ou não.

```
 1: algoritmo PRIMO
 2: leia(X)
 3: DIVISORES ← 0
 4: TESTADOR ← 2
 5: enquanto TESTADOR < X
 6:     se X MOD TESTADOR = 0
 7:         DIVISORES ← DIVISORES + 1
 8:     fim{se}
 9:     TESTADOR ← TESTADOR + 1
10: fim{enquanto}
11: se DIVISORES = 0
12:     escreva("PRIMO")
13: senão
14:     escreva("NÃO PRIMO")
15: fim{se}
```

Seguir um algoritmo, simulando suas operações é chamado teste de mesa (ou em inglês *walkthrough*), ou mais jocosamente *fazer um chinês*. É o feijão com arroz da programação, usado para entender, certificar e depurar algoritmos. Ao fazer isto com o algoritmo acima, perceba que ele funciona. Se chamado com 15 responderá NÃO PRIMO e se chamado com 13 responderá PRIMO. Programar é escrever estas "coisas". É como aprender um idioma novo. Mas antes de aprender a expressar idéias (algoritmos), há que se ter firmeza na fonética e na síntaxe. A semântica virá depois.

Mais um exemplo Alguns aqui devem conhecer a sequência de Fibonacci: 1,1,2,3,5,8,13,21,34,... Note que, nela, cada número é a soma dos dois anteriores. A sequência começa com 1,1 já que estes não tem dois anteriores. A partir do próximo (o 2), a regra genérica se aplica. Vamos estudar um algoritmo que leia um número n e imprima o n-ésimo número de Fibonacci.

```
1: leia LIMITE
2: A ← 1; B ← 1; N ← 3
3: enquanto N ≤ LIMITE
4:     R ← A + B
5:     A ← B
6:     B ← R
7:     N ← N + 1
8: fim{enquanto}
9: escreva R
```

A seguir, alguns exercícios de síntaxe da linguagem:

☞ Para você fazer

Ex. 1

```
 1: X ← 10
 2: Y ← 39
 3: GAMA ← 2
 4: enquanto X < Y
 5:     X ← X + GAMA
 6:     se ((GAMA mod 2) ≠ 0)
 7:         GAMA ← GAMA + 1
 8:     fim{se}
 9:     GAMA ← GAMA + 1
10: fim{enquanto}
11: escreva X
```

Ex. 2

```
 1: A ← 3
 2: B ← 23
 3: enquanto A ≤ B
 4:     se ((B > 8) ∨ (A ≤ 15))
 5:         B ← B - 2
 6:     fim{se}
 7:     A ← A + 2
 8:     B ← B - 2
 9: fim{enquanto}
10: A ← A - 1
11: escreva (A + B)
```

Ex. 3

```
 1: X ← 3
 2: enquanto X < 11
 3:     Y ← 6
 4:     enquanto Y < 14
 5:         X ← X + 1
 6:         Y ← X + 1
 7:     fim{enquanto}
 8:     Y ← Y + 3
 9:     X ← X + 3
10: fim{enquanto}
11: escreva (X-Y)
```

Ex. 4

```
 1: X ← 3
 2: Y ← 7
 3: se  X < 5 ∧ Y > 3
 4:     X ← X - 3
 5: senão
 6:     Y ← Y + 1
 7:     se X mod 3 ≠ 0
 8:         Y ← Y + 2
 9:     fim{se}
10: fim{se}
11: escreva X+Y
```

Responda aqui os valores encontrados.

1	2	3	4

UFPR - UP - UTFPR - PUCPr – 24/04/2024 - 11:42:12.5
Prof Dr P Kantek (pkantek@gmail.com)
Aquecimento em programação VIVX001a, V: 2.06
75965 PHELIPE GABRIEL LIMA DA SILV
24PRG101 - 21 _____ / _____ / _____

Programar é ... Já se disse com alguma razão que programar é emprestar um pouco de inteligência ao computador. Então, antes de começar, reconheça-se a nobreza do que se está a fazer. Os computadores (todos) descendem de duas propostas mais ou menos complementares: a primeira, descrita brilhantemente por A. Turing em 1936 e a segunda igualmente brilhante, por Von Neumann em começos da década de 50. Pela primeira, sabe-se que o computador é capaz de executar uma instrução após a outra. Tais instruções precisam ser simples e inequívocas. Pela segunda descobre-se que tais instruções são bastante limitadas: E/S, aritmética, movimentação, alternativa e desvio.

E/S Este tipo de instrução responde pela E=entrada e S=saída de dados do/para o computador. Suponha que você construiu uma máquina capaz de descobrir o seno de qualquer ângulo. Para pôr a máquina para funcionar, você precisa fornecer algum ângulo a ela e depois que ela fizer seu trabalho ela vai precisar informar a você o valor do seno. Estas duas etapas correspondem a operações de E/S. Do ponto de vista teórico da programação, são operações quase irrelevantes, embora do ponto de vista prático sejam fundamentais.

aritmética Esta família é muito importante, mas graças a Deus, já estudamos pelo menos 8 anos de matemática. Tudo o que se aprendeu lá (adição, produto, potência, paridade, regra de sinais, números inteiros e reais, positivos e negativos, propriedades associativa, comutativa, distributiva – prioridades e parêntesis, etc etc, etc, etc) continua válido.

movimentação Operação muito simples, pela qual, partes da memória, chamadas variáveis são movimentadas ou mais propriamente copiadas entre sí.

alternativa Aqui a instrução mais "nobre" da programação. É graças a ela que um computador se torna diferente de um liquidificador. Se uma condição for boa (verdadeira) executar-se-á alguma coisa. Se a condição for ruim (falsa) executar-se-á outra coisa, ou em certos casos não se executará nada. Vale aqui a lógica binária com seus 2 princípios:

- Não contradição: Uma condição é verdadeira ou falsa, não podendo ser as duas coisas ao mesmo tempo.
- Terceiro excluído: se não for falsa será verdadeira e vice-versa.

desvio Um programa de computador vai executar a primeira instrução, depois a segunda e assim por diante, até o final do programa, a menos que uma instrução seja de desvio: *vá para a instrução x* : Neste caso a seguir será executada a instrução x e não a próxima.

Na década de 70, alguns autores (Dijkstra, Corrado e Jacopini) notaram que o uso de desvios tendia a gerar programas difíceis de *debugar*: Este verbo novo (=depurar) é o processo de identificar, localizar e corrigir erros de programas. Na programação gasta-se 20% do tempo escrevendo código e 80% depurando-o. A programação usando desvios gerou a chamada programação *spaguetti*. Daí os autores sugeriram organizar os desvios: nascia a programação estruturada. Agora os comandos são agrupados em blocos, havendo 3 deles:

sequência 2 ou mais comandos em sequência
alternativa como acima descrito
repetição uma sequência que pode se repetir. Apenas aqui é permitido o desvio e sempre do fim do bloco para o começo.

Fora o caso acima (repetição) o comando de desvio é proibido. Isto é tão verdade que linguagens mais modernas não têm o desvio, como por exemplo a linguagem Python.

Outra característica bem importante da *arquitetura de Von Neumann* é a idéia de programa armazenado: instruções são armazenadas como dados, permitindo que programas processem programas. Esta ideia já havia sido posta por Turing. Nascia o conceito de linguagem de programação, pela qual as complexidades da programação de máquina são transferidas para um programa de computador: o compilador. Uma linguagem de programação é na verdade 2 coisas:

⋆ um conjunto de regras fonéticas, sintáticas e semânticas de como descrever um algoritmo e
⋆ um programa (chamado compilador ou interpretador) capaz de ler o algoritmo acima descrito e gerar o equivalente código de máquina capaz de ser executado em um computador real.

Nascia o *software* em oposição ao *hardware*. Graças a este conceito, aprender a programar passou de um processo de meses/anos para um de horas/dias. Sem contar que agora o mesmo programa roda em um supercomputador de milhões de dólares e também em um celular android.

Algoritmo O conceito chave em programação é o de *algoritmo*. É o que está por trás de um programa. É a ideia, a receita, a lista dos procedimentos do programa. Se programar é arte e engenharia, o algoritmo responde integralmente pela parte "arte". O conceito nos acompanha faz tempo (o algoritmo de mdc, devido a Euclides tem mais de 2400 anos), mas sua definição precisa é devida ao matemático inglês Alan Turing. Para poder expressar um algoritmo, se necessita uma linguagem apropriada. Vamos descrever uma mini-linguagem agora: As peças elementares de informação serão chamadas variáveis. Cada uma terá um nome, uma palavra qualquer (como aliás já fizemos nas aulas de álgebra em qualquer dos mais de 8 anos de ensino fundamental de matemática).

As operações de E/S serão 2: leia(variável) ou imprima(variável), significando a entrada do valor de uma variável ou sua impressão.

Para designar um valor a uma variável usar-se-á o comando de atribuição, representando por $\leftarrow$, então, por exemplo $A \leftarrow 0$ significa colocar o valor zero na variável A.

As operações aritméticas, seguem 100% do aprendido nas aulas de matemática e são: $+$: adição, $*$: multiplicação, $-$: subtração, $/$: divisão real, $//$: divisão inteira, MOD: resto de, $**$: potência, além de prioridade determinada por parêntesis. Continua proibida a divisão por zero. As linguagens mais modernas já operam normalmente com números complexos, mas podemos deixar esta facilidade de lado (por enquanto).

Para o comando alternativo precisa-se definir uma condição lógica: uma expressão envolvendo condicionais ($>$, $<$, $\leq$, $\geq$, $=$, $\neq$, $\in$, ...) e cuja resposta será um de dois valores: VERDADEIRO ou FALSO. Estes, ainda podem ser combinados usando os conectivos: E ($\wedge$) e OU ($\vee$) além de NÃO ($\sim$). O comando alternativo tem o formato `se <condição> então bloco-A fim{se}` neste comando o bloco de comandos A só será executado se a condição citada for verdadeira. Um formato alternativo é `se <condição> então bloco-A senão bloco-B fim{se}` Agora, se a condição é verdadeira, A é executado (como acima), mas se é falso então o bloco-B é executado.

Finalmente, o desvio não existe, mas em seu lugar entram as estruturas de repetição. Só vamos ver 1 delas. Trata-se do comando `enquanto` que tem formato `enquanto <condição> bloco-A fim{enquanto}`. Note que ele é parecido com o comando alternativo, mas tem uma diferença fundamental: ao encontrar o comando `fim{enquanto}` há um desvio incondicional para o início do bloco ou o `enquanto`.

Um exemplo Vamos escrever um algoritmo para testar se o número X é primo ou não.

```
 1: algoritmo PRIMO
 2: leia(X)
 3: DIVISORES ← 0
 4: TESTADOR ← 2
 5: enquanto TESTADOR < X
 6:     se X MOD TESTADOR = 0
 7:         DIVISORES ← DIVISORES + 1
 8:     fim{se}
 9:     TESTADOR ← TESTADOR + 1
10: fim{enquanto}
11: se DIVISORES = 0
12:     escreva("PRIMO")
13: senão
14:     escreva("NÃO PRIMO")
15: fim{se}
```

Seguir um algoritmo, simulando suas operações é chamado teste de mesa (ou em inglês *walkthrough*), ou mais jocosamente *fazer um chinês*. É o feijão com arroz da programação, usado para entender, certificar e depurar algoritmos. Ao fazer isto com o algoritmo acima, perceba que ele funciona. Se chamado com 15 responderá NÃO PRIMO e se chamado com 13 responderá PRIMO. Programar é escrever estas "coisas". É como aprender um idioma novo. Mas antes de aprender a expressar idéias (algoritmos), há que se ter firmeza na fonética e na síntaxe. A semântica virá depois.

Mais um exemplo Alguns aqui devem conhecer a sequência de Fibonacci: 1,1,2,3,5,8,13,21,34,... Note que, nela, cada número é a soma dos dois anteriores. A sequência começa com 1,1 já que estes não tem dois anteriores. A partir do próximo (o 2), a regra genérica se aplica. Vamos estudar um algoritmo que leia um número n e imprima o n-ésimo número de Fibonacci.

```
 1: leia LIMITE
 2: A ← 1; B ← 1; N ← 3
 3: enquanto N ≤ LIMITE
 4:     R ← A + B
 5:     A ← B
 6:     B ← R
 7:     N ← N + 1
 8: fim{enquanto}
 9: escreva R
```

A seguir, alguns exercícios de síntaxe da linguagem:

☞ Para você fazer

Ex. 1

```
 1: X ← 9
 2: Y ← 42
 3: GAMA ← 4
 4: enquanto X < Y
 5:     X ← X + GAMA
 6:     se ((GAMA mod 3) ≠ 0)
 7:         GAMA ← GAMA + 2
 8:     fim{se}
 9:     GAMA ← GAMA + 1
10: fim{enquanto}
11: escreva X
```

Ex. 2

```
 1: A ← 3
 2: B ← 22
 3: enquanto A ≤ B
 4:     se ((B > 9) ∨ (A ≤ 13))
 5:         B ← B - 1
 6:     fim{se}
 7:     A ← A + 2
 8:     B ← B - 3
 9: fim{enquanto}
10: A ← A - 1
11: escreva (A + B)
```

Ex. 3

```
 1: X ← 4
 2: enquanto X < 12
 3:     Y ← 2
 4:     enquanto Y < 14
 5:         X ← X + 1
 6:         Y ← X + 1
 7:     fim{enquanto}
 8:     Y ← Y + 2
 9:     X ← X + 3
10: fim{enquanto}
11: escreva (X-Y)
```

Ex. 4

```
 1: X ← 9
 2: Y ← 1
 3: se  X < 6 ∧ Y > 4
 4:     X ← X - 2
 5: senão
 6:     Y ← Y + 1
 7:     se X mod 2 ≠ 0
 8:         Y ← Y + 3
 9:     fim{se}
10: fim{se}
11: escreva X+Y
```

Responda aqui os valores encontrados.

1	2	3	4

UFPR - UP - UTFPR - PUCPr – 24/04/2024 - 11:42:12.5
Prof Dr P Kantek (pkantek@gmail.com)
Aquecimento em programação VIVX001a, V: 2.06
75972 REBECA CABRAL DOS SANTOS
24PRG101 - 22 _____ / _____ / _____

Programar é ... Já se disse com alguma razão que programar é emprestar um pouco de inteligência ao computador. Então, antes de começar, reconheça-se a nobreza do que se está a fazer. Os computadores (todos) descendem de duas propostas mais ou menos complementares: a primeira, descrita brilhantemente por A. Turing em 1936 e a segunda igualmente brilhante, por Von Neumann em começos da década de 50. Pela primeira, sabe-se que o computador é capaz de executar uma instrução após a outra. Tais instruções precisam ser simples e inequívocas. Pela segunda descobre-se que tais instruções são bastante limitadas: E/S, aritmética, movimentação, alternativa e desvio.

E/S Este tipo de instrução responde pela E=entrada e S=saída de dados do/para o computador. Suponha que você construiu uma máquina capaz de descobrir o seno de qualquer ângulo. Para pôr a máquina para funcionar, você precisa fornecer algum ângulo a ela e depois que ela fizer seu trabalho ela vai precisar informar a você o valor do seno. Estas duas etapas correspondem a operações de E/S. Do ponto de vista teórico da programação, são operações quase irrelevantes, embora do ponto de vista prático sejam fundamentais.

aritmética Esta família é muito importante, mas graças a Deus, já estudamos pelo menos 8 anos de matemática. Tudo o que se aprendeu lá (adição, produto, potência, paridade, regra de sinais, números inteiros e reais, positivos e negativos, propriedades associativa, comutativa, distributiva – prioridades e parêntesis, etc etc, etc, etc) continua válido.

movimentação Operação muito simples, pela qual, partes da memória, chamadas variáveis são movimentadas ou mais propriamente copiadas entre sí.

alternativa Aqui a instrução mais "nobre" da programação. É graças a ela que um computador se torna diferente de um liquidificador. Se uma condição for boa (verdadeira) executar-se-á alguma coisa. Se a condição for ruim (falsa) executar-se-á outra coisa, ou em certos casos não se executará nada. Vale aqui a lógica binária com seus 2 princípios:

- Não contradição: Uma condição é verdadeira ou falsa, não podendo ser as duas coisas ao mesmo tempo.
- Terceiro excluído: se não for falsa será verdadeira e vice-versa.

desvio Um programa de computador vai executar a primeira instrução, depois a segunda e assim por diante, até o final do programa, a menos que uma instrução seja de desvio: *vá para a instrução x* : Neste caso a seguir será executada a instrução x e não a próxima.

Na década de 70, alguns autores (Dijkstra, Corrado e Jacopini) notaram que o uso de desvios tendia a gerar programas difíceis de *debugar*: Este verbo novo (=depurar) é o processo de identificar, localizar e corrigir erros de programas. Na programação gasta-se 20% do tempo escrevendo código e 80% depurando-o. A programação usando desvios gerou a chamada programação *spaguetti*. Daí os autores sugeriram organizar os desvios: nascia a programação estruturada. Agora os comandos são agrupados em blocos, havendo 3 deles:

sequência 2 ou mais comandos em sequência
alternativa como acima descrito
repetição uma sequência que pode se repetir. Apenas aqui é permitido o desvio e sempre do fim do bloco para o começo.

Fora o caso acima (repetição) o comando de desvio é proibido. Isto é tão verdade que linguagens mais modernas não têm o desvio, como por exemplo a linguagem Python.

Outra característica bem importante da *arquitetura de Von Neumann* é a idéia de programa armazenado: instruções são armazenadas como dados, permitindo que programas processem programas. Esta ideia já havia sido posta por Turing. Nascia o conceito de linguagem de programação, pela qual as complexidades da programação de máquina são transferidas para um programa de computador: o compilador. Uma linguagem de programação é na verdade 2 coisas:

⋆ um conjunto de regras fonéticas, sintáticas e semânticas de como descrever um algoritmo e
⋆ um programa (chamado compilador ou interpretador) capaz de ler o algoritmo acima descrito e gerar o equivalente código de máquina capaz de ser executado em um computador real.

Nascia o $software$ em oposição ao $hardware$. Graças a este conceito, aprender a programar passou de um processo de meses/anos para um de horas/dias. Sem contar que agora o mesmo programa roda em um supercomputador de milhões de dólares e também em um celular android.

Algoritmo O conceito chave em programação é o de *algoritmo*. É o que está por trás de um programa. É a ideia, a receita, a lista dos procedimentos do programa. Se programar é arte e engenharia, o algoritmo responde integralmente pela parte "arte". O conceito nos acompanha faz tempo (o algoritmo de mdc, devido a Euclides tem mais de 2400 anos), mas sua definição precisa é devida ao matemático inglês Alan Turing. Para poder expressar um algoritmo, se necessita uma linguagem apropriada. Vamos descrever uma mini-linguagem agora: As peças elementares de informação serão chamadas variáveis. Cada uma terá um nome, uma palavra qualquer (como aliás já fizemos nas aulas de álgebra em qualquer dos mais de 8 anos de ensino fundamental de matemática).

As operações de E/S serão 2: leia(variável) ou imprima(variável), significando a entrada do valor de uma variável ou sua impressão.

Para designar um valor a uma variável usar-se-á o comando de atribuição, representando por $\leftarrow$, então, por exemplo $A \leftarrow 0$ significa colocar o valor zero na variável A.

As operações aritméticas, seguem 100% do aprendido nas aulas de matemática e são: +: adição, ∗: multiplicação, −: subtração, /: divisão real, //: divisão inteira, MOD: resto de, ∗∗: potência, além de prioridade determinada por parêntesis. Continua proibida a divisão por zero. As linguagens mais modernas já operam normalmente com números complexos, mas podemos deixar esta facilidade de lado (por enquanto).

Para o comando alternativo precisa-se definir uma condição lógica: uma expressão envolvendo condicionais ($>$, $<$, $\leq$, $\geq$, $=$, $\neq$, $\in$, ...) e cuja resposta será um de dois valores: VERDADEIRO ou FALSO. Estes, ainda podem ser combinados usando os conectivos: E ($\wedge$) e OU ($\vee$) além de NÃO ($\sim$). O comando alternativo tem o formato `se <condição> então bloco-A fim{se}` neste comando o bloco de comandos A só será executado se a condição citada for verdadeira. Um formato alternativo é `se <condição> então bloco-A senão bloco-B fim{se}` Agora, se a condição é verdadeira, A é executado (como acima), mas se é falso então o bloco-B é executado.

Finalmente, o desvio não existe, mas em seu lugar entram as estruturas de repetição. Só vamos ver 1 delas. Trata-se do comando `enquanto` que tem formato `enquanto <condição> bloco-A fim{enquanto}`. Note que ele é parecido com o comando alternativo, mas tem uma diferença fundamental: ao encontrar o comando `fim{enquanto}` há um desvio incondicional para o início do bloco ou o `enquanto`.

Um exemplo Vamos escrever um algoritmo para testar se o número X é primo ou não.

```
 1: algoritmo PRIMO
 2: leia(X)
 3: DIVISORES ← 0
 4: TESTADOR ← 2
 5: enquanto TESTADOR < X
 6:     se X MOD TESTADOR = 0
 7:         DIVISORES ← DIVISORES + 1
 8:     fim{se}
 9:     TESTADOR ← TESTADOR + 1
10: fim{enquanto}
11: se DIVISORES = 0
12:     escreva("PRIMO")
13: senão
14:     escreva("NÃO PRIMO")
15: fim{se}
```

Seguir um algoritmo, simulando suas operações é chamado teste de mesa (ou em inglês *walkthrough*), ou mais jocosamente *fazer um chinês*. É o feijão com arroz da programação, usado para entender, certificar e depurar algoritmos. Ao fazer isto com o algoritmo acima, perceba que ele funciona. Se chamado com 15 responderá NÃO PRIMO e se chamado com 13 responderá PRIMO. Programar é escrever estas "coisas". É como aprender um idioma novo. Mas antes de aprender a expressar idéias (algoritmos), há que se ter firmeza na fonética e na síntaxe. A semântica virá depois.

Mais um exemplo Alguns aqui devem conhecer a sequência de Fibonacci: 1,1,2,3,5,8,13,21,34,... Note que, nela, cada número é a soma dos dois anteriores. A sequência começa com 1,1 já que estes não tem dois anteriores. A partir do próximo (o 2), a regra genérica se aplica. Vamos estudar um algoritmo que leia um número n e imprima o n-ésimo número de Fibonacci.

```
1: leia LIMITE
2: A ← 1; B ← 1; N ← 3
3: enquanto N ≤ LIMITE
4:     R ← A + B
5:     A ← B
6:     B ← R
7:     N ← N + 1
8: fim{enquanto}
9: escreva R
```

A seguir, alguns exercícios de síntaxe da linguagem:

☞ Para você fazer

Ex. 1

```
 1: X ← 6
 2: Y ← 40
 3: BETA ← 3
 4: enquanto X < Y
 5:     X ← X + BETA
 6:     se ((BETA mod 2) ≠ 0)
 7:         BETA ← BETA + 2
 8:     fim{se}
 9:     BETA ← BETA + 1
10: fim{enquanto}
11: escreva X
```

Ex. 2

```
 1: A ← 1
 2: B ← 22
 3: enquanto A ≤ B
 4:     se ((B > 11) ∨ (A ≤ 15))
 5:         B ← B - 3
 6:     fim{se}
 7:     A ← A + 2
 8:     B ← B - 3
 9: fim{enquanto}
10: A ← A - 3
11: escreva (A + B)
```

Ex. 3

```
 1: X ← 4
 2: enquanto X < 9
 3:     Y ← 3
 4:     enquanto Y < 11
 5:         X ← X + 2
 6:         Y ← X + 1
 7:     fim{enquanto}
 8:     Y ← Y + 3
 9:     X ← X + 3
10: fim{enquanto}
11: escreva (X-Y)
```

Ex. 4

```
 1: X ← 3
 2: Y ← 8
 3: se  X < 5 ∧ Y > 3
 4:     X ← X - 1
 5: senão
 6:     Y ← Y + 3
 7:     se X mod 2 ≠ 0
 8:         Y ← Y + 1
 9:     fim{se}
10: fim{se}
11: escreva X+Y
```

Responda aqui os valores encontrados.

1	2	3	4