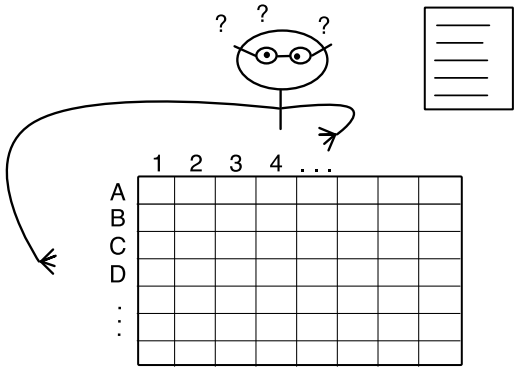
Fundamentos de programação

aula 01: Programando computadores

1 Introdução

Nada melhor do que uma caricatura para explicar o que uma coisa é.

Abaixo nós temos a caricatura de um computador



Os 3 elementos dessa figura são: o processador, a memória e o programa.

E o mais importante deles, claro, é o processador.

É ele quem faz as coisas acontecerem.

Mas, o pobre ...

Ele não é, digamos assim, lá muito esperto.

E para piorar as coisas, ele é desmemoriado, não tem a vista boa, e só tem um braço comprido, porque o outro é bem curtinho.

Por exemplo, imagine que um mostro 3 números pra você

```
17,   51,   42
```

Então, só de bater o olho, você já sabe que o `51` é o maior de todos.

Mas, o processador não consegue fazer isso.

Quer dizer, ele vai primeiro pegar o `17`, e depois o `51`.

Daí, comparando os dois ele descobre que o `51` é o maior dos dois.

Em seguida, ele pega o `42`.

E com mais uma comparação ele descobre que o `51` é o maior de todos.

Mas, mesmo isso o processador não consegue fazer sozinho.

Quer dizer, ele só consegue realizar essa tarefa se houver um programa que descreve para ele esse passo a passo.

A seguir, nós vamos ver outros exemplo de programas de computador.

2 Programas de computador

O que nós acabamos de ver, claro, é apenas uma caricatura.

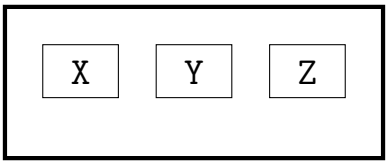
Agora, nós vamos começar a ver como as coisas realmente são.

Na figura da primeira página, nós apresentamos a memória como um grande quadriculado, onde cada posição é identificada pela sua linha e coluna (e.g., A3, B8, C1, etc.).

De fato, as coisas são assim mesmo: cada posição da memória tem o seu endereço.

Mas, na prática a gente não trabalha assim.

Quer dizer, na prática a gente dá nomes a algumas posições e trabalha com esses nomes



Imagine que existem 3 números nas posições `X`, `Y`, `Z`, e que nós precisamos descobrir qual é o maior deles.

(*Agora nós estamos na posição do processador: a gente não consegue ver que números são esses.*)

Então, nós podemos raciocinar da seguinte maneira:

- Primeiro, nós comparamos `X` e `Y`.

  Daí, se `X` for maior que `Y`, então nós comparamos `X` e `Z`.

  E daí, se `X` for o maior de novo, nós descobrimos a resposta.

Essa lógica corresponde ao seguinte programa de computador:

```
Se ( X > Y )                              <= 1a comparação
Então Se ( X > Z )                        <= 2a comparação
      Então Print ("X é o maior!")        <= resposta
```

Fácil, não é?

Mas, ainda não está completo.

Quer dizer, ninguém garante que `X` é maior que `Y`.

Mas se não for, então o `Y` é o maior dos dois.

E daí é ele quem a gente compara com o `Z`

```
Se ( X > Y )                              <= 1a comparação
Então Se ( X > Z )                        <= 2a comparação (caso 1)
      Então Print ("X é o maior!")
Senão Se ( Y > Z )                        <= 2a comparação (caso 2)
      Então Print ("Y é o maior!")
```

Certo.

Mas ainda falta uma coisa: considerar os casos em que `Z` é o maior de todos.

E para isso, a gente pode raciocinar da seguinte maneira

- Imagine que `X` é maior que `Y`.
  Daí, se a gente descobrir que `Z` é maior que `X`
  Então a gente vai saber que `Z` também é maior que `Y`.
  Logo, `Z` é o maior de todos.
  (O outro caso é análogo.)

Para colocar essa lógica dentro do nosso programa, basta fazer o seguinte

```
Se ( X > Y )
Então Se ( X > Z )
      Então Print ("X é o maior!")
      Senão Print ("Z é o maior!")
Senão Se ( Y > Z )
      Então Print ("Y é o maior!")
      Senão Print ("Z é o maior!")
```

Não é legal?

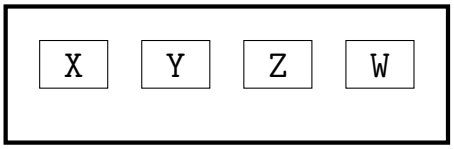
(Você viu que nós montamos esse programa com 3 bloquinhos `Se..Então..Senão..` ?)

Agora nós vamos ver outros exemplos.

Exemplos

a. **Complicando um pouquinho as coisas**

Se a gente quiser complicar um pouquinho as coisas, basta colocar mais um número na jogada.



E se prepare, porque a coisa vai ficar grande.
Mas não vai aparecer nada de novo.
Quer dizer, a lógica continua a mesma

- A gente vai comparando os números dois a dois
  e sempre compara o maior número que a gente já viu
  com o próximo número que a gente ainda não viu

O caso em que `X` é o maior de todos fica assim

```
Se ( X > Y )
Então Se ( X > Z )
      Então Se ( X > W )
            Então Print ("X é o maior!")
```

Quer dizer, dessa vez a gente precisa de 3 comparações em sequência.

Abaixo nós temos o programa completo

```
Se ( X > Y )
Então Se ( X > Z )
      Então Se ( X > W )
            Então Print ("X é o maior!")
            Senão Print ("W é o maior!")
      Senão Se ( Z > W )
            Então Print ("Z é o maior!")
            Senão Print ("W é o maior!")
Então Se ( Y > Z )
      Então Se ( Y > W )
            Então Print ("Y é o maior!")
            Senão Print ("W é o maior!")
      Senão Se ( Z > W )
            Então Print ("Z é o maior!")
            Senão Print ("W é o maior!")
```

◇

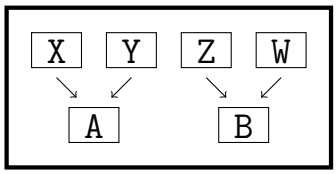
b. **Uma pequena esperteza**

O programa do exemplo anterior ficou realmente grande, não é?

Quer dizer, nós usamos 6 bloquinhos `Se-Então-Senão`.

Mas, uma pequena esperteza permite a gente reduzir as coisas.

A esperteza é a seguinte



Quer dizer,

- Quando a gente compara `X` e `Y`,

  a gente guarda o maior deles em `A`.

  E quando a gente compara `W` e `Z`,

  a gente guarda o maior deles em `B`.

  Daí, para chegar na resposta,

  basta comparar `A` e `B`.

O programa que implementa essa ideia é o seguinte

```
Se ( X > Y )
Então   A ← X
Senão   A ← Y

Se ( Z > W )
Então   B ← Z
Senão   B ← W

Se ( A > B )
Então   Print ("O maior é:", A)
Senão   Print ("O maior é:", B)
```

Legal! — agora só tem 3 bloquinhos outra vez.?

Esse programa também tem duas novidades:

- A primeira delas é o comando que escreve coisas na memória

  ```
  A   ←   X
  ```

- E a segunda é essa versão do `Print`

  ```
  Print ("O maior é:", A)
  ```

  que além de imprimir a mensagem, também imprime o valor de `A`.

◇

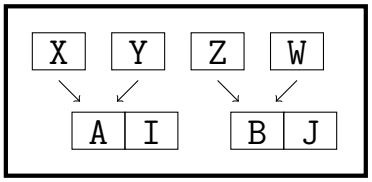
c. **Outra pequena esperteza**

Mas o programa que a gente acabou de ver ainda tem um problema.

Quer dizer, ele imprime o maior número na tela.

Mas a gente fica sem saber quem ele é: `X`, `Y`, `Z` ou `W`.

A solução para isso é uma outra esperteza



Quer dizer, agora, quando a gente compara `X` com `Y`, a gente guarda o maior deles em `A`, e coloca uma indicação de quem ele é em `I`.

(E faz a mesma coisa para `Z` e `W`.)

Essa indicação pode ser qualquer coisa.

Por exemplo, os números 1 e 2.

Isso não importa.

Só o que importa é que a gente saiba o que pode estar lá, para poder testar depois.

Abaixo nós temos o programa que implementa essa ideia.

```
Se ( X > Y )
Então  {  A ← X;  I ← 1  }
Senão  {  A ← Y;  I ← 2  }

Se ( Z > W )
Então  {  B ← Z;  I ← 1  }
Senão  {  B ← W;  I ← 2  }

Se ( A > B )
Então Se ( I = 1 )
      Então Print ("O maior é X")
      Senão Print ("O maior é Y")
Senão Se ( J = 1 )
      Então Print ("O maior é Z")
      Senão Print ("O maior é W")
```

Esse programa tem mais uma novidade:

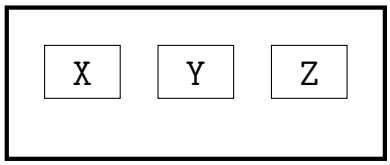
- Quando nós descobrimos que `X` é maior do que `Y` nós precisamos fazer duas coisas, e daí nós escrevemos assim

  ```
  {  A ← X;  I ← 1  }
  ```

  quer dizer, nós colocamos as instruções entre chaves, e colocamos um ponto e vírgula entre elas, para indicar que nós temos uma lista de instruções.

d. **Colocando as coisas em ordem**

Imagine outra vez que nós temos apenas 3 números



Mas dessa vez a tarefa é colocar os números em ordem:

- colocar o menor número em `X`
- colocar o número do meio em `Y`
- colocar o maior número em `W`

Ideia #1:

Depois que a gente já sabe quem é o maior de todos, é fácil descobrir quem é o do meio.

Daí, a gente também vai saber quem é o menor.

E uma vez que a gente já sabe isso, basta colocar as coisas em ordem.

Mas, aqui surge uma pequena dificuldade.

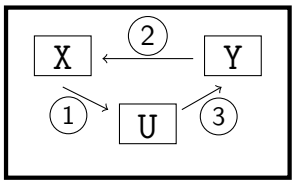
Quer dizer, *como é que a gente troca dois números de lugar?*

Porque, se a gente coloca o valor de `Y` em `X`

```
X  ←  Y
```

a gente perde o valor de `X` (e vice-versa).

Logo, a gente precisa fazer as coisas em 3 passos, usando um terceiro lugar



o que corresponde à seguinte sequência de instruções:

```
{  U ← X;  X ← Y;  Y ← U  }
```

Agora nós já temos tudo o que é preciso para implementar a primeira ideia.

Abaixo nós temos o trecho de programa que corresponde à ordem `X` > `Z` > `Y`:

```
Se ( X > Y )
Então Se ( X > Z)
      Então Se ( Y > Z )
            Então ( .  .  .  )
            Senão {
                    U ← X;  X ← Y;  Y ← U;
                    U ← Y;  Y ← Z;  Z ← U;
                  }
```

Mas, se a gente for fazer isso para todos os casos, a coisa vai ficar muito grande.

E isso nos leva para a nossa segunda ideia.

Ideia #2:

A ideia aqui é começar a mover os números logo após a primeira comparação (se necessário)

Por exemplo, se a gente descobrir que `X` é maior que `Y`, a gente já troca os dois números de lugar.

Porque no final, o número em `X` deverá ser menor que o número em `Y`.

Depois, a gente faz a mesma coisa com `X` e `Z` — isto é, compara e troca (se necessário).

Nesse ponto, a gente já tem certeza que o menor de todos está em `X`.

E daí, é só fazer a mesma coisa com `Y` e `Z`.

O programa abaixo implementa essa ideia

```
Se ( X > Y )
Então {  U ← X;  X ← Y;  Y ← U  }
Se ( X > Z )
Então {  U ← X;  X ← Z;  Z ← U  }
Se ( Y > Z )
Então {  U ← Y;  Y ← Z;  Z ← U  }
```

3 Programando na linguagem C

Agora, nós vamos escrever todos os programas da seção anterior na linguagem C.

Para isso, nós observamos que um programa em C tipicamente tem 3 partes:

```
#include <stdlib.h>            |   Parte 1
#include <stdio.h>             |

int  X = 17;                   |
int  Y = 51;                   |   Parte 2
int  Z = 42;                   |

int main {                     |
                               |   Parte 3
       ( . . . )               |
    }                          |
```

Como a linguagem C é muito grande, os seus comandos são organizados em *bibliotecas*.

Na Parte 1 nós dizemos que bibliotecas nós queremos usar.

No exemplo acima, nós temos duas bibliotecas:

- `stdlib.h`: biblioteca padrão
- `stdio.h`: biblioteca padrão para comandos de entrada e saída
  (comandos que escrevem na tela e leem coisas do teclado)

Outras bibliotecas bastante utilizadas são: `math.h` (funções matemáticas), `string.h` (comandos para a manipulação de sequências de caracteres).

Na prática, a gente está sempre utilizando as mesmas bibliotecas na maior parte do tempo, e apenas copia a Parte 1 de um programa para o outro.

A Parte 2 é o lugar onde a gente indica as posições da memória que o programa vai utilizar (i.e., o seu nome e o seu tipo), e coloca algum valor lá (opcionalmente).

Finalmente, a Parte 3 é o lugar onde ficam as instruções para o processador.

Abaixo nós temos o código C do primeiro programa que nós fizemos na Seção 2.

```
#include <stdlib.h>
#include <stdio.h>

int X = 17;
int Y = 51;
int Z = 42;

int main()
  {
    if ( X > Y )
       if ( X > Z )
```

```
          printf("X é o maior");
      else
          printf("Z é o maior");
   else
      if ( Y > Z )
         printf("Y é o maior");
      else
         printf("Z é o maior");
}
```

Algumas observações:

- Como muitos devem saber, `if` e `else` são as palavras em inglêspara `Se` e `Senão`.

  Agora, a curiosidade é que a linguagem C não usa a palavra `then` (`Então`).

  Quer dizer, aquilo que vem depois da condição do `if` já é subentendido como o `Então`.

- A outra observação é que em C o comando `Print` chama `printf`.

Agora vejamos os outros exemplos.

a. **Complicando um pouquinho as coisas**

```
#include <stdlib.h>
#include <stdio.h>

int X = 17, Y = 51, Z = 42, W = 88;

int main()
  {
    if ( X > Y )
       if ( X > Z )

          if ( X > W ) printf("X é o maior");
          else         printf("W é o maior");

       else if ( X > W ) printf("Z é o maior");
            else         printf("W é o maior");

    else if ( Y > Z )

          if ( Y > W ) printf("Y é o maior");
          else         printf("W é o maior");

       else if ( Z > W ) printf("Z é o maior");
            else         printf("W é o maior");
  }
```

Aqui nós fizemos algumas espertezas para tornar o programa mais compacto:

- as variáveis `X, Y, Z, W` foram declaradas todas na mesma linha
- os comandos `printf` foram colocados na mesma linha do `if` e `else`

◇

b. **Uma pequena esperteza**

```
#include <stdlib.h>
#include <stdio.h>

int X = 17, Y = 51, Z = 42, W = 88;

int A,B;

int main()
  {
    if ( X > Y )  A = X;
    else          A = Y;

    if ( Z > W )  B = Z;
    else          B = W;

    if ( A > B ) printf("O maior é: %d", A);
    else         printf("O maior é: %d", B);
  }
```

Nesse programa,

- Nós não demos um valor inicial para as variáveis `A` e `B`.

  Elas recebem o seu valor em instruções como

  ```
  A = X
  ```

  E aqui nós vemos o formato da instrução `A` $\leftarrow$ `X` na linguagem C.

- No final, nós também vemos a versão do `printf` que imprime o valor de uma variável

  ```
  printf("O maior é: %d", A)
  ```

  Aqui, o `%d` indica o local dentro da mensagem onde o valor deve aparecer.

  Abaixo nós temos um outro exemplo

  ```
  printf("Então, %d é o maior número", A)
  ```

$\diamond$

c. **Outra pequena esperteza**

```
#include <stdlib.h>
#include <stdio.h>

int X = 17, Y = 51, Z = 42, W = 88;

int A,I,B,J;

int main()
  {
    if ( X > Y )  { A = X; I = 1; }
    else          { A = Y; I = 2; }
```

```
  if ( Z > W )  { B = Z; J = 1; }
  else          { B = W; J = 2; }

  if ( A > B )
     if ( I == 1 ) printf("X é o maior");
     else          printf("Y é o maior");
  else
     if ( J == 1 ) printf("Z é o maior");
     else          printf("W é o maior");
}
```

Nesse programa nós vemos que

- Para testar se dois números são iguais, nós utilizamos o símbolo ==

```
if ( I == 1 )
```

◇

d. **Colocando as coisas em ordem**

Finalmente, aqui está o programa que implementa a segunda ideia do último exemplo

```
#include <stdlib.h>
#include <stdio.h>

int X = 17, Y = 51, Z = 42;

int U;

int main()
  {
    if ( X > Y )  {  U = X;  X = Y;  Y = U;  }

    if ( X > Z )  {  U = X;  X = Z;  Z = U;  }

    if ( Y > Z )  {  U = Y;  Y = Z;  Z = U;  }
  }
```

◇