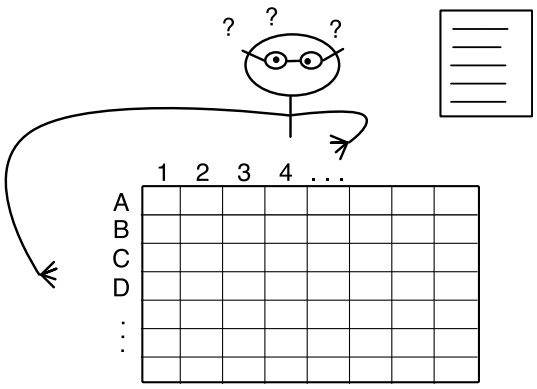
Fundamentos da Programação

aula 01: Programando computadores

1 Introdução

Abaixo nós temos um esquema da arquitetura de um computador



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

E claro, a coisa mais importante é o processador — (*o bonequinho ...*)

Quer dizer, é ele quem faz tudo acontecer.

E sem ele nada acontece.

Mas o pobre ...

Ele não é lá muito esperto.

Só entende instruções muito simples.

Não lembra onde ele coloca as coisas.

E não enxerga muito bem ...

Daí que, para as coisas funcionarem a gente precisa explicar tudo bem direitinho.

Quer dizer, a explicação que a gente dá é o *programa*.

Ali a gente diz:

```
- faça isso
- faça aquilo
- guarde isso aqui
- pergunte o valor do x
- faça essa conta e coloque o resultado acolá
- depois escreva isso na tela
```

e por aí vai ...

Vejamos como a coisa funciona.

2 Programando computadores

A primeira coisa que nós vamos fazer é um programa que imprime alguma coisa na tela.

Na linguagem C, a coisa é feita assim

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

3.   int main()
      {
4.       printf ("oi! to aqui ...");
      }
```

Quer dizer, `print` é o comando que imprime uma mensagem na tela.

Legal.

Mas esse programa ainda não faz nada de útil ...

A coisa útil mais simples que existe é uma pequena continha.

Então, o próximo programa vai fazer uma conta e imprimir o resultado na tela.

A coisa fica assim

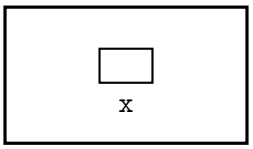
```
1.   #include <stdio.h>
2.   #include <stdlib.h>

3.   int main()
      {
4.       int  x;

5.       x  =  1 + 1;
6.       printf ("o resultado é: %d", x);
      }
```

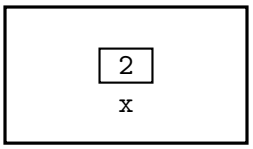
A primeira novidade aqui aparece na linha `4`.

Quer dizer, na linha `4` nós dizemos que o programa vai utilizar a porção de memória `x`, para armazenar números inteiros.



Falando tecnicamente, nós dizemos que `x` é uma *variável*.

E na linha `5`, nós armazenamos o resultado da conta na variável `x`



A segunda novidade aparece na linha 6, onde nós vemos como o comando `printf` imprime o valor de uma variável

```
printf ("o resultado é:  %d", x);
```

Quer dizer, a mensagem entre aspas continua lá.

Mas agora, nós colocamos o marcador %d para indicar o lugar da mensagem onde o valor de `x` deve aparecer.

Mas, a coisa também poderia ser assim

```
printf ("%d foi o resultado", x);
```

ou assim

```
printf ("deu %d a continha que eu fiz ...", x);
```

ou assim

```
printf ("%d mais %d é igual a %d", 1, 1, x);
```

A seguir, nós vamos ver mais exemplos de programas de computador.

Exemplos

a. Lendo coisas do teclado

O programa que nós acabamos de fazer ainda não tem muita graça.

Quer dizer, se a gente executar o programa 3 vezes, ele vai fazer a mesma coisa 3 vezes ...

Para que o programa faça coisas diferentes em execuções diferentes, ele pode perguntar alguma coisa pra gente.

Por exemplo, a coisa pode ficar assim

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

3.  int main()
     {
4.      int  x, y, soma;

5.      printf ("digite o primeiro número: ");
6.      scanf ("%d", &x);
7.      printf ("\n");
8.      printf ("digite o segundo número: ");
9.      scanf ("%d", &y);

10.     soma  =  x + y;
11.     printf ("a soma de %d e %d é igual a %d", x, y, soma);
     }
```

A novidade aqui é o comando `scanf`, que é semelhante ao `printf` mas funciona ao contrário

`scanf ("%d", &x);`

`printf ("%d", x);`

— (*comandos de entrada e saída ...*)

Quer dizer, o comando `scanf` indica que

- o programa vai ler um número inteiro do teclado: `%d`
- para ser colocado dentro da variável `x`: `&x`

Outra pequena novidade aparece na linha `2`

```
printf ("\n")
```

Quer dizer, essa é a maneira de indicar que nós queremos pular para a próxima linha, antes de imprimir a mensagem

```
printf ("digite o segundo número:  ")
```

◇

b. O comando if-else

Legal.

Agora está na hora de colocar um pouco de inteligência no nosso programa.

Quer dizer, inteligência é a capacidade de fazer coisas diferentes em situações diferentes.

E para detectar situações diferentes, nós precisamos testar se as coisas são assim ou assado.

Na linguagem C, isso é feito com o comando `if-else`.

A coisa fica assim

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

 3.   int main()
       {
 4.       float   n1, n2;
 5.       float   media;

 6.       printf ("digite a primeira nota: ");
 7.       scanf ("%d", &n1);
 8.       printf ("\n digite a segunda nota: ");
 9.       scanf ("%d", &n2);

10.       media  =  ( n1 + n2 ) / 2;
11.       if ( media >= 5 )
12.          printf ("média = %f  ==>  Aprovado", media);
13        else
14.          printf ("média = %f  ==>  Reprovado", media);
       }
```

A primeira novidade desse programa aparece nas linhas 4, 5 e 10

```
 4.  float n1, n2;
 5.  float media;
      ( .  .  .  )
10.  media = ( n1 + n2 ) / 2;
```

Quer dizer, nas linhas 4, 5 nós indicamos que as variáveis `n1`, `n2`, `media` vão armazenar números fracionários (ou do tipo `float`), porque uma nota não precisa ser um número inteiro.

E daí, para imprimir uma variável `float`, a gente escreve assim

```
printf ("média = %f", media);
```

A segunda novidade é o uso do comando `if-else` nas linhas $4 - 7$, que implementa a seguinte lógica

```
Se ( a média é maior ou igual a 5 )
   Imprime ("Aprovado")
Senão
   Imprime ("Reprovado")
```

◇

c. Múltiplos `if`'s

Mas, nem sempre um único comando `if` basta.

Quer dizer, imagine agora que os estudantes com média maior que 7 são aprovados direto.

E que os estudantes com média entre 4 e 7 devem fazer a prova final.

E que os estudantes com média menor que 4 são reprovados.

Então, a parte final do programa ficaria assim

```
10.       media  =  ( nota1 + nota2 ) / 2;
11.       if ( media >= 7 )
12.          printf ("média = %f  ==>  Aprovado", media);
13        else if ( media >= 4 )
14.          printf ("média = %f  ==>  Prova final", media);
15.       else
16.          printf ("média = %f  ==>  Reprovado", media);
      }
```

Quer dizer, essa é a maneira de testar múltiplas condições.

A ideia é que se a primeira condição não é verdadeira, então nós testamos a segunda condição.

Se a segunda condição não é verdadeira, então nós testamos a terceira.

E por aí vai ...

Quando uma condição verdadeira é encontrada, as demais não são mais testadas.

Mas, às vezes acontece de mais de uma condição poder ser verdadeira.

Nesse caso, a gente escreve a coisa assim

```
if ( < condição 1 > )
   < comando 1 >
if ( < condição 2 > )
   < comando 2 >
if ( < condição 3 > )
   < comando 3 >
...
```

◇

d. As raízes da equação quadrática

Considere a equação abaixo

$$x^2 \quad + \quad 2x \quad - \quad 3 \quad = \quad 0$$

Todo mundo sabe que a solução dessa equação é calculada assim

$$x \quad = \quad \frac{-2 \quad \pm \quad \sqrt{2^2 \quad - \quad 4 \cdot 1 \cdot (-3)}}{2} \qquad \Rightarrow \qquad x_1 \quad = \quad 1 \qquad \text{e} \qquad x_2 \quad = \quad -3$$

Agora, nós vamos programar o computador para que ele faça esse cálculo.

Quer dizer, a forma geral da equação quadrática é assim

$$ax^2 \quad + \quad bx \quad + \quad c \quad = \quad 0$$

e a fórmula que dá a solução é essa aqui

$$x \quad = \quad \frac{-b \quad \pm \quad \sqrt{b^2 \quad - \quad 4 \cdot a \cdot c}}{2}$$

Mas, essa fórmula só pode ser usada quando o valor dentro da raiz quadrada não é negativo.

Daí que, nós vamos fazer a coisa em duas etapas

- primeiro nós vamos calcular o valor dentro da raiz
- e depois nós testamos esse valor, e calculamos a solução quando isso for possível

A coisa fica assim

```
1.   #include <stdio.h>
2.   #include <stdlib.h>
3.   #include <math.h>

4.   int main()
      {
5.       float  a = 1, b = 2, c = -3;
6.       float  delta;
7.       float  x1, x2;
```

```
 8.       delta  =  b * b - 4 * a * c;

11.       if ( delta < 0 )
12.          printf ("a equação não tem solução");

13.       if ( delta == 0 )
           {
14.           x1  =  -b / ( 2 * a );
15.           printf ("a única solução da equação é: %f", x1);
           }

16.       if ( delta = 0 )
           {
17.           x1  =  ( -b + sqrt ( delta ) ) / ( 2 * a );
17.           x2  =  ( -b - sqrt ( delta ) ) / ( 2 * a );
19.           printf ("as soluções da equação são: %f e %f", x1, x2);
           }
      }
```

A primeira novidade desse programa aparece logo na linha 3

```
#include <math.h>
```

Quer dizer, aqui a gente está incluindo a biblioteca matemática para poder utilizar o comando `sqrt`() mais abaixo, que calcula a raiz quadrada.

A segunda novidade aparece na linha 5

```
float  a = 1, b = 2, c = -3
```

Quer dizer, aqui nós estamos atribuindo valores às variáveis `a`, `b`, `c` já no momento da sua declaração, para evitar ter que escrever um monte de `input`'s para obter os seus valores.

◇

e. Imprimindo em ordem crescente

Agora imagine que o nosso programa vai ler 3 números do teclado: `a`, `b`, `c`.

E a tarefa consiste em

→ *Imprimir os números em ordem crescente de valor*

Quer dizer, a ideia aqui é praticar a lógica da programação.

Raciocinando um pouquinho, a gente vê que existem 6 casos possíveis

- `a` < `b` < `c`
- `b` < `a` < `c`
- `c` < `a` < `b`
- `a` < `b` < `c`
- `b` < `c` < `a`
- `c` < `b` < `a`

Daí que, o programa pode ser feito assim

```
( . . . )
```

```
10.     if ( a < b  &&  b < c )
11.        printf ("%d < %d < %d", a, b, c);

12.     if ( a < c  &&  c < b )
13.        printf ("%d < %d < %d", a, c, b);

14.     if ( b < a  &&  a < c )
15.        printf ("%d < %d < %d", b, a, c);

16.     if ( b < c  &&  c < a )
17.        printf ("%d < %d < %d", b, c, a);

18.     if ( c < a  &&  a < b )
19.        printf ("%d < %d < %d", c, a, b);

20.     if ( c < b  &&  b < a )
21.        printf ("%d < %d < %d", c, b, a);
```

A novidade aqui são os comandos `if` que testam múltiplas condições, por meio do operador && — (*i.e., o E lógico*)

Mas, a gente pode ser um pouquinho mais esperto — (*para diminuir o número de comparações que o programa realiza*)

Quer dizer, note que as variáveis `a` e `b` são comparadas nas linhas `10`, `14`, `18` e `19`.

Mas, a gente só precisa fazer essa comparação 1 vez

```
if ( a < b )
 {
    //  raciocínio p/ o caso em que a < b
    ( . . . )
 }
else
 {
    //  raciocínio p/ o caso em que b < a
    ( . . . )
 }
```

Certo.

Imagine que a gente já sabe que `a` é menor que `b`.

Daí, se a gente descobrir que `c` é menor do que `a`, então a gente já vai ter a ordem completa: `c` $<$ `a` $<$ `b`

```
if ( a < b )
    if ( c < a )
       printf ("%d < %d < %d", c, a, b);
    ( . . . )
else
    //  raciocínio p/ o caso em que b < a
    ( . . . )
```

Por outro lado, se isso não é verdade (i.e., se `a` é menor do que `c`), então a gente descobriu que `a` é o menor de todos.

E agora, basta comparar os outros dois para saber a ordem completa

```
if ( a < b )
    if ( c < a )
       print ("%d < %d < %d" % (c, a, b))
    else
       if ( b < c )
          print ("%d < %d < %d" % (a, b, c))
       else
          print ("%d < %d < %d" % (a, c, b))
else
    //  raciocínio p/ o caso em que b < a
```

Daí, raciocinando da mesma maneira no outro caso, nós chegamos ao seguinte programa completo

```
( . . . )
```

```
1.  printf("digite o primeiro número:");
2.  scanf("%d",&a);
3.  printf("digite o segundo número:");
4.  scanf("%d",&b);
```

```
 5.   printf("digite o terceiro número:");
 6.   scanf("%d",&c);

 7.   if ( a < b )
 8.       if ( c < a )
 9.           print ("%d < %d < %d" % (c, a, b))
10.       else
11.           if ( b < c )
12.               print ("%d < %d < %d" % (a, b, c))
13.           else
14.               print ("%d < %d < %d" % (a, c, b))
15.   else
16.       if ( c < b )
17.            print ("%d < %d < %d" % (c, b, a))
18.       else
19.            if ( a < c )
20.                print ("%d < %d < %d"% (b, a, c))
21.             else
22.                print ("%d < %d < %d" % (b, c, a))
```

que realiza apenas 6 comparações.

Legal, não é?