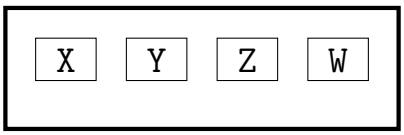
Fundamentos de programação

aula 02: Programando atividades repetitivas

1 Introdução

Na aula passada, nós consideramos o problema de descobrir o maior número armazenado nas posições `X`, `Y`, `Z`, `W` da memória

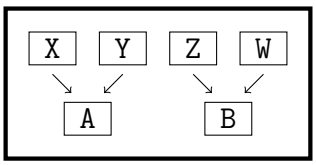


Nós resolvemos esse problema de duas maneiras.

A primeira delas consistia em fazer comparações entre esses elementos, e nos levava a considerar todo um monte de possibilidades

```
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!")
```

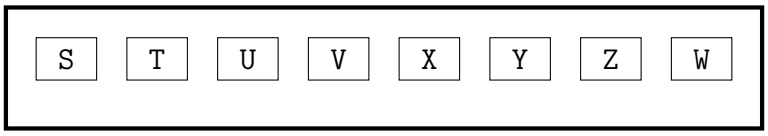
A segunda solução utilizava a seguinte esperteza



e nós dava um programa bem menor

```
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)
```

Agora, imagine que o problema consiste em descobrir o maior número armazenado em 8 posições de memória



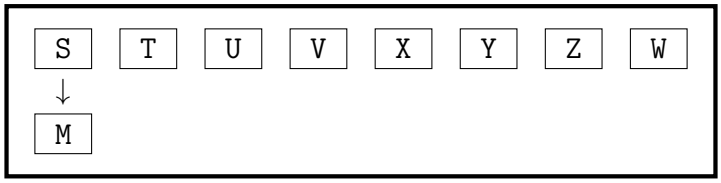
Já dá para imaginar que a primeira ideia acima ia nos dar um programa imenso.

E mesmo com a segunda ideia o programa ia ficar grande também.

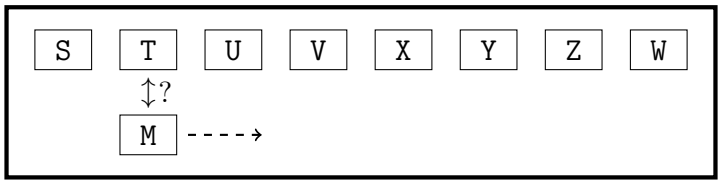
Agora, nós vamos ver uma outra esperteza que permite lidar com essa situação.

A esperteza consiste em ter um outro lugar `M` onde nós armazenamos o maior número que nós vimos até o momento.

Daí, no início, nós armazenamos o número da posição `S` lá



E depois, nós vamos comparando o valor `M` com todos os outrs números (um de cada vez), atualizando `M` sempre que necessário



O programa fica assim:

```
M  ←  S
Se ( T > M ) Então  M ← T
Se ( U > M ) Então  M ← U
Se ( V > M ) Então  M ← V
Se ( X > M ) Então  M ← X
Se ( Y > M ) Então  M ← Y
Se ( Z > M ) Então  M ← Z
Se ( W > M ) Então  M ← W
Print ("O maior número é:", M)
```

Legal, não é?

Abaixo nós temos a versão do programa na linguagem C (omitindo a declaração de bibliotecas e a declaração de variáveis, para ganhar espaço)

```
void main()
 {
     M = S;
     if ( T > M)  M = T;
     if ( U > M)  M = U;
     if ( V > M)  M = V;
     if ( X > M)  M = X;
     if ( Y > M)  M = Y;
     if ( Z > M)  M = Z;
     if ( W > M)  M = W;
     printf("O maior número é %d", M);
 }
```

2 Programando atividades repetitivas

Certo.

Mas, deveria ser possível fazer um programa ainda menor, não é?

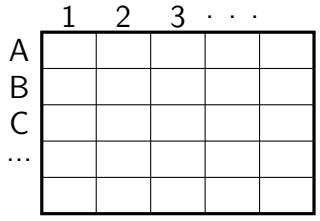
Quer dizer, quase todas as linhas fazem basicamente a mesma coisa.

Logo, deveria ser possível escrever essa linha apenas uma vez.

Sim, isso é possível!

Mas para isso, nós vamos precisar de mais uma esperteza.

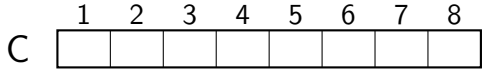
A esperteza consiste em fazer uso da organização da memória do computador



Quer dizer, no computador as posições de memória ficam todas uma ao lado da outra.

Daí que, nós podemos colocar os nossos números todos na mesma linha.

Por exemplo



Agora, o valor de `M` vai ser sempre comparado com um elemento da linha `C`, e o programa fica assim

```
M  ←  C[1]
Se ( T > C[2] ) Então  C[2] ← T
Se ( U > C[3] ) Então  C[3] ← U
Se ( V > C[4] ) Então  C[4] ← V
Se ( X > C[5] ) Então  C[5] ← X
Se ( Y > C[6] ) Então  C[6] ← Y
```

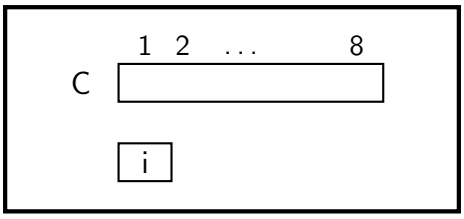
```
Se ( Z > C[7] ) Então  C[7] ← Z
Se ( W > C[8] ) Então  C[8] ← W
Print ("O maior número é:", M)
```

Legal.

Agora as linhas intermediárias estão quase iguais.

A única coisa que muda é o índice que indica a posição.

E a esperteza consiste em guardar esse índice na memória



e fazer o índice ser atualizado automaticamente

```
M  ←  C[1]
Para  i ← 2 Até 8 Faça
     Se ( C[i] > M )  Então  M ← C[i]
Print ("O maior número é:", M)
```

Esse é um dos truques das linguagens de programação para a realização de atividades repetitivas.

Quer dizer, o comando

```
Para  i <--  2  Até  8  Faça
 {
     ( . . . )
 }
```

atualiza automaticamente o índice `i`, começando no valor que você escolher (que nesse caso foi 2) e terminando no valor que você escolher (que nesse caso foi 8).

E para cada valor que o índice `i` assume, os comandos {(...)} são executados uma vez.

Por exemplo, nós podemos utilizar esse truque para escrever a mensagem "`to aqui!`" 10 vezes na tela

```
Para  i ← 1 Até 10 Faça
     Print ("To aqui!")
```

Mas, a coisa fica mais interessante quando os comandos que estão sendo executados dependem do índice `i`.

Por exemplo, o programa

```
Para  i ← 1 Até 10 Faça
     Print ("contando:", i)
```

imprime na tela

```
contando: 1
contando: 2
. . .
contando: 10
```

Essa também é a ideia do programa que encontra o maior número

```
M  ←  C[1]
Para  i ← 2  Até  8  Faça
     Se ( C[i] > M )  Então  M ← C[i]
Print ("O maior número é:", M)
```

Abaixo nós temos a versão desse programa na linguagem C

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

int C[8] = { 12, 21, 34, 43, 56, 65, 78, 87 };

int i, M;

int main()
 {
     M = C[0];

     for ( i=0 ; i=i+1 ; i<8 )

         if ( C[i] > M )   M = C[i];

     printf("O maior número é:%d", M);
 }
```

E aqui nós temos diversas observações a fazer

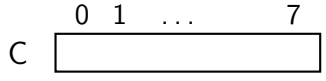
- A linha

  ```
  int C[8] = { 12, 21, 34, 43, 56, 65, 78, 87 };
  ```

  está dizendo basicamente o seguinte:

  > Esse programa vai utilizar uma porção de memória com o nome `C`,
  > que contém 8 posições onde serão armazenados números inteiros.
  > Além disso, aqui estão 8 números para serem armazenados lá.

- A linguagem C sempre começa a contar as posições a partir do 0

  Daí que, a primeira posição de `C` é `C[0]`, o que explica essa linha do programa

  ```
  M = C[0];
  ```

- Finalmente, o comando `for` da linguagem C é muito versátil.

  Quer dizer, ele tem 3 partes

$$\texttt{for (} \underbrace{\texttt{i = 1}}_{①} \texttt{ ; } \underbrace{\texttt{i < 8}}_{②} \texttt{ ; } \underbrace{\texttt{i = i + 1}}_{③} \texttt{)}$$

  - a parte ① é o lugar onde a gente dá o valor inicial do índice `i`.
  - na parte ②, a gente indica o valor final; mas a gente faz isso na forma de uma condição de parada

$$\texttt{i < 8}$$

    quer dizer, quando o indice `i` alcança o valor 8, essa condição se torna falsa, e daí a repetição pára.
  - finalmente, a parte ③ indica o tamanho do pulo

$$\texttt{i = i + 1}$$

    por exemplo, se a gente quisesse que o índice `i` fosse pulando de 2 em 2, a gente colocaria ali

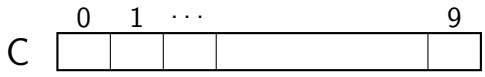
$$\texttt{i = i + 2}$$

  Mas, apesar de toda essa versatilidade, na prática a gente quase sempre usa essa versão padrão do comando `for`

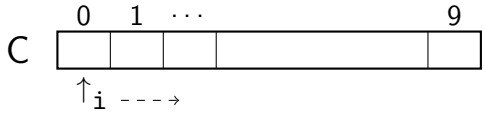
  - `for ( i=0; i<10; i=i+1 )`

3 Outros exemplos

Imagine agora que nós queremos encontrar não apenas o maior elemento, mas também o menor elemento de uma lista

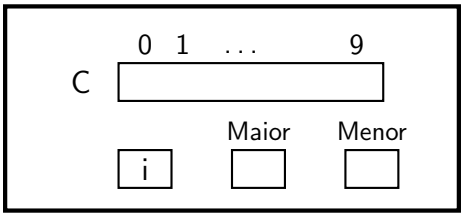


A ideia, claro, é ir percorrendo a lista da esquerda para a direita, examinando cada número para ver se ele é o menor ou o maior elemento que a gente viu até o momento



(Como a gente faria com uma lista escrita no papel, com o auxílio do nosso dedo.)

Para realizar essa tarefa, nós vamos precisar de 3 variáveis auxiliares



E a coisa fica assim (já na linguagem C)

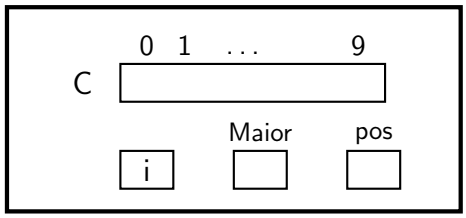
```
#include <stdlib.h>
#include <stdio.h>

int C[10] = { 12, 21, 34, 43, 56, 65, 78, 87, 89, 98 };
int i, Maior, Menor;

int main()
 {
    Maior = C[0];   Menor = C[0];

    for ( i=1; i<10; i=i+1 )
     {
        if ( C[i] > Maior )   Maior = C[i];
        if ( C[i] < Menor )   Menor = C[i];
     }
    printf ("O maior é %d, e o menor é %d", Maior, Menor);
 }
```

Mais exemplos

a. **Usando a esperteza outra vez**

Às vezes, nós queremos saber onde o maior elemento está, e não apenas o seu valor.

Para isso, nós podemos usar a mesma esperteza que foi aprendida na aula passada



Quer dizer, sempre que nós atualizamos a variável `Maior`, a gente anota a posição onde isso aconteceu.

A coisa fica assim

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

int C[10] = { 12, 21, 34, 43, 56, 65, 78, 87, 89, 98 };
int i, Maior, pos;
```

```
int main()
 {
    Maior = C[0];   pos = 0;

    for ( i=1; i<10; i=i+1 )
     {
        if ( C[i] > Maior )   {  Maior = C[i];   pos = i;  }
     }
    printf ("O maior número é %d, e ele fica na posição %d", Maior, pos);
 }
```

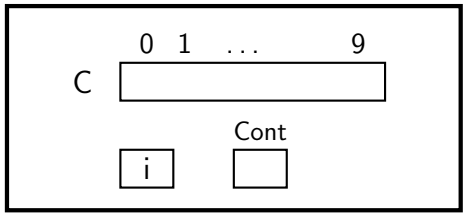
◇

b. **Contando as coisas**

Imagine agora que a lista pode ter números positivos e negativos.

E imagine que nós queremos saber quantos números negativos existem na lista.

Para isso, basta utilizar um contador



E percorrer a lista da esquerda para a direita, incrementando o contador sempre que a gente encontra um número negativo.

A coisa fica assim

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

int C[10] = { 12, 21, 34, 43, 56, 65, 78, 87, 89, 98 };
int i, Cont = 0;

int main()
 {
    for ( i=0; i<10; i=i+1 )
     {
        if ( C[i] < 0 )   Cont = Cont + 1;
     }
    printf ("O maior número é %d, e ele fica na posição %d", Maior, pos);
 }
```

Observação

- A instrução

  ```
  Cont   =   Cont + 1;
  ```

  pode parecer um pouco engraçada à primeira vista.

  Mas, basta lembrar que isso não é uma equação.

  Quer dizer, essa instrução está dizendo:

- pegue o valor armazenado na posição Cont
- incremente esse valor de 1 unidade
- e guarde o resultado na posição Cont outra vez

◇

c. **Imprimindo as coisas**

Agora imagine que nós queremos ver quais são os números negativos.

A melhor maneira de fazer isso consiste em imprimir o número negativo assim que a gente encontra ele.

E a coisa fica assim

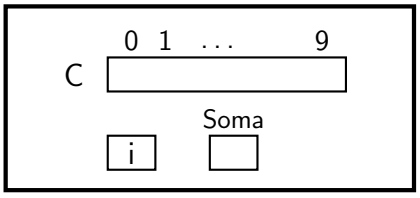
```
#include <stdlib.h>
#include <stdio.h>

int C[10] = { 12, -21, 34, -43, 56, 65, -78, 87, -89, -98 };
int i;

int main()
 {
    for ( i=0; i<10; i=i+1 )
     {
        if ( C[i] < 0 )
           printf ("%d", C[i]);
 }   }
```

◇

d. **Somando as coisas**

Agora imagine que nós queremos saber qual é a soma dos números positivos da lista.

Para fazer isso, nós precisamos guardar essa soma em algum lugar



E a coisa fica assim

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

int C[10] = { 12, 21, 34, 43, 56, 65, 78, 87, 89, 98 };
int i, Soma = 0;

int main()
 {
    for ( i=0; i<10; i=i+1 )
     {
        if ( C[i] > 0 )   Soma = Soma + C[i];
     }
    printf ("O maior número é %d, e ele fica na posição %d", Maior, pos);
 }
```

◇