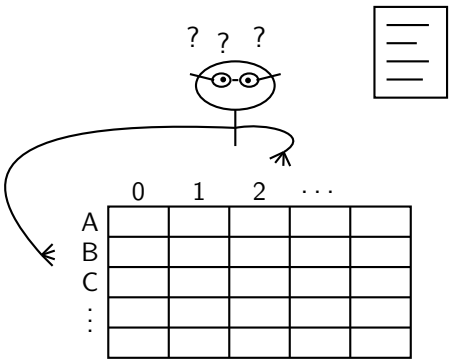
Fundamentos da Programação

aula 02: Inspeção de listas

1 Introdução

Considere mais uma vez o esquema da arquitetura de um computador

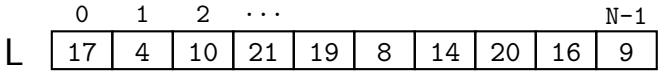


Na aula passada, nós concentramos a atenção no processador e no programa.

E na aula de hoje, nós vamos concentrar a atenção na memória.

Quer dizer, a gente não vai realmente prestar atenção na memória inteira.

A ideia é olhar para apenas uma linha



Quer dizer, isso é uma lista de números.

E a primeira tarefa da aula de hoje é

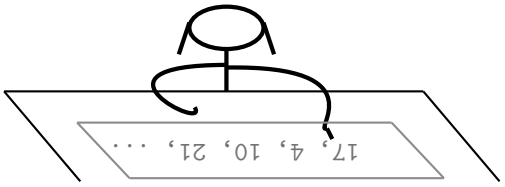
→ *Imprimir todos os números da lista*

Mas, como é que o computador faz isso?

Bom, o computador faz da mesma maneira que você.

E você faria isso assim

- `você põe o dedo sobre o primeiro elemento`
- `daí, você vai deslocando o dedo até o final da lista`
- `e, para cada elemento que você vê, você imprime esse elemento na tela — (ou diz ele em voz alta ...)`



Legal.

Mas como é o programa que faz isso?

Bom, a gente pode descrever essa lógica assim

```
Para  i  começando em  0  e indo até  N-1
 {
     Imprime o i-ésimo elemento da lista
 }
```

onde a variável `i` está fazendo o papel do dedo.

Certo.

Mas e o programa mesmo, como é?

Bom, na linguagem C a coisa fica assim

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

#define  N  10

int  L[N] = { 17, 4, 10, 21, 19, 8, 14, 20, 16, 9 };

int main ()
 {
     int  i;

     for ( i=0 ; i<N ; i++ )          // percorre a lista com o dedo i
      {
         printf (" %d ", L[i]);       //   imprime o i-ésimo elemento
 }    }
```

Nota: Você percebeu uma coisa engraçada?

Quer dizer, a gente conta os elementos da lista assim: `1, 2, 3, ..., N`.

Mas a linguagem C conta assim: `0, 1, 2, ..., N-1`.

Quer dizer,

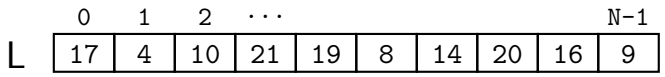
- o primeiro elemento da lista é o elemento zero: `L[0]`
- o segundo elemento da lista é o elemento um: `L[1]`
- o terceiro elemento da lista é o elemento dois: `L[2]`
- e por aí vai ...

E é por isso que o comando `for` fica assim

```
for ( i=0 ; i<N ; i++ )
```

2 Inspeção de listas

Considere mais uma vez a lista L



E imagine que a tarefa é a seguinte

→ *Calcular a soma de todos os elementos da lista*

Bom, a ideia aqui é fazer a mesma coisa.

Quer dizer, a gente percorre a lista da esquerda para a direita.

E vai somando os elementos um a um.

Daí, a coisa fica assim

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

#define   N  10

int  L[N] = { 17, 4, 10, 21, 19, 8, 14, 20, 16, 9 };

int main ()
 {
     int  i;
     int  soma = 0;

     for ( i=0 ; i<N ; i++ )          // percorre a lista
      {
         soma  =  soma  +  L[i];      //   somando os elementos
      }

     printf ("A soma dos elementos é: %d", soma);
 }
```

Vejamos outros exemplos

Exemplos

a. Somando os elementos ímpares

Agora imagine que a tarefa é a seguinte

→ *Calcular a soma dos elementos ímpares da lista*

Bom, a ideia aqui é fazer a mesma coisa outra vez.

Quer dizer, a gente vai percorrer a lista da esquerda para a direita.

Mas só vai somar os elementos que forem ímpares.

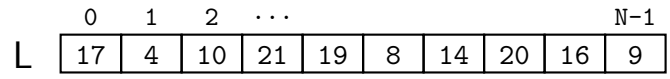
A coisa fica assim

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

#define  N  10

int  L[N] = { 17, 4, 10, 21, 19, 8, 14, 20, 16, 9 };


int main ()
 {
     int  i;
     int  soma = 0;

     for ( i=0 ; i<N ; i++ )
      {
         if ( L[i] % 2 == 1 )      // Se ( L[i] é ímpar )
          {
            soma  =  soma  +  L[i];
      }   }

     printf ("A soma dos elementos ímpares é: %d", soma);
 }
```

◇

b. Somando os elementos das posições ímpares

Considere mais uma vez a lista



E imagine que a tarefa é

→ *Calcular a soma dos elementos que se encontram nas posições ímpares da lista*

Bom, a estratégia é semelhante à que foi utilizada no exemplo anterior.

Só que dessa vez, ao invés de testar se o elemento é ímpar, nós testamos se a sua posição é ímpar.

A coisa fica assim

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

#define  N  10

int  L[N] = { 17, 4, 10, 21, 19, 8, 14, 20, 16, 9 };

int main ()
 {
     int  i, soma = 0;
```

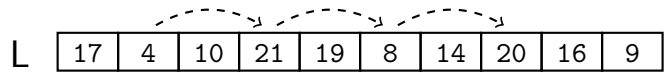
```
    for ( i=0 ; i<N ; i++ )
     {
        if ( i % 2 == 1 )          // Se ( i é ímpar )
         {
           soma  =  soma  +  L[i];
     }   }
    printf ("A soma dos elementos das posições ímpares é: %d", soma);
}
```

Legal.

Mas essa não é a única maneira de fazer as coisas.

Quer dizer, a linguagem C nos permite fazer uma pequena esperteza.

A esperteza consiste em percorrer a lista pulando de 2 em 2



Para fazer isso, a gente usa a terceira parte do comando `for`, e modifica isso

```
for ( i=1 ; i<N; i++ )
```

↑

pulando de 1 em 1

para isso aqui

```
for ( i=1 ; i<N; i=i+2 )
```

↑

pulando de 2 em 2

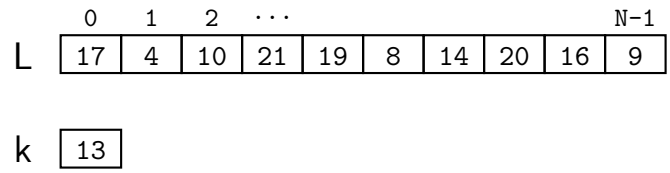
Abaixo nós temos a versão mais esperta do programa em C

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

#define  N  10

int  L[N] = { 17, 4, 10, 21, 19, 8, 14, 20, 16, 9 };

int main ()
 {
    int  i, soma = 0;

    for ( i=1 ; i<N ; i=i+2 )
     {
        soma  =  soma  +  L[i];
     }
    printf ("A soma dos elementos das posições ímpares é: %d", soma);
 }
```

◇

c. Procurando o número k

Agora, imagine que além da lista L nós também temos a variável `k` que armazena um número qualquer



A tarefa consiste em

→ *Verificar se o valor armazenado em* `k` *está na lista* L *ou não*

Bom, a ideia aqui é utilizar a nossa estratégia padrão.

Quer dizer,

- nós vamos percorrer a lista da esquerda para a direita
- e vamos comparar cada elemento com o número `k`

O único detalhe é que nós precisamos lembrar se o número `k` já foi encontrado ou não.

E para isso, nós utilizamos uma variável `aux`.

A coisa fica assim

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

#define  N  10

int  L[N] = { 17, 4, 10, 21, 19, 8, 14, 20, 16, 9 };

int  k = 13;

int main ()
 {
     int  i,  aux = 0;

     for ( i=0 ; i<N ; i++ )
      {
         if ( L[i] == k )
          {
             aux = 1;
      }   }

     if ( aux == 1 )    printf("Achei!");
     else               printf("Não está lá ...");
 }
```

Legal.

Agora nós vamos ver uma outra esperteza da linguagem C.

Quer dizer, uma vez que a gente encontra o número `k` na lista, não faz mais sentido continuar procurando por ele.

Daí que, nesse ponto, o comando `for` já pode ser interrompido.

Na linguagem C, a gente usa o comando `break` para fazer isso.

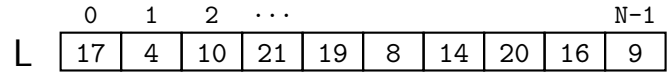
A coisa fica assim

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

#define  N  10

int  L[N] = { 17, 4, 10, 21, 19, 8, 14, 20, 16, 9 };

int  k = 13;

int main ()
 {
     int  i,  aux = 0;

     for ( i=0 ; i<N ; i++ )
      {
         if ( L[i] == k )
          {
             aux  =  1;   break;
      }   }

     if ( aux == 1 )    printf ("Achei!");
     else               printf ("Não está lá ...");
 }
```

◇

d. Encontrando o maior elemento da lista

Considere outra vez a lista L



E imagine que a tarefa é a seguinte

→ *Encontrar o maior elemento da lista*

Bom, a ideia mais uma vez é percorrer a lista da esquerda para a direita.

E a cada momento, a gente lembra qual foi o maior elemento que a gente viu até ali.

Quer dizer, a gente vai armazenar o maior valor na variável `M`.

E vai comparando cada elemento da lista com o valor de M.

A coisa fica assim

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

#define  N  10

int  L[N] = { 17, 4, 10, 21, 19, 8, 14, 20, 16, 9 };

int  k = 13;

int main ()
 {
     int  i, M;

     M = L[0];
     for ( i=1 ; i<N ; i++ )
      {
         if ( L[i] > M )
          {
             M = L[i];
      }   }
     printf ("O maior elemento é: %d", M);
 }
```

Mas ainda existe um pequeno problema.

Quer dizer, esse programa descobre quem é o maior elemento.

Mas ele não lembra aonde esse número está.

Só que isso é bem fácil de consertar.

Quer dizer, basta usar uma outra variável para lembrar aonde o maior valor está.

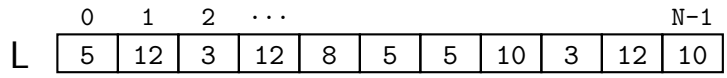
A coisa fica assim

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

#define  N  10

int  L[N] = { 17, 4, 10, 21, 19, 8, 14, 20, 16, 9 };

int main ()
 {
     int  i, M, posM;

     M  =  L[0];   posM  =  0;
     for ( i=1 ; i<N ; i++ )
      {
         if ( L[i] > M )
          {
             M  =  L[i];   posM  =  i;
      }   }
     printf ("O maior elemento da lista está na posição: %d", posM);
 }
```

◇

e. Contando as cópias do maior elemento

Agora imagine que a lista L pode conter elementos repetidos



E imagine que a tarefa é

→ *Contar o número de vezes que o maior elemento aparece na lista*

Bom, a ideia aqui é raciocinar em etapas

- primeiro, a gente descobre quem é o maior elemento
- e depois, a gente conta quantas vezes ele aparece

A coisa fica assim

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

#define  N  10

int  L[N] = { 5, 12, 3, 12, 8, 5, 10, 3, 12, 10 };

int main ()
 {
     int  i, M, cont = 0;


     // 1a ETAPA: encontra o maior

     M  =  L[0];
     for ( i=1 ; i<N ; i++ )
      {
         if ( L[i] > M )
          {
             M  =  L[i];
      }   }


     // 2a ETAPA: conta ocorrências do maior

     for ( i=1 ; i<N ; i++ )
      {
         if ( L[i] == M )
          {
             cont ++;
      }   }

     printf ("O número de cópias do maior elemento é: %d", cont);
 }
```

Certo.

Mas também é possível fazer a coisa em uma etapa só.

Quer dizer, a cada momento a gente lembra o maior número que a gente viu até ali.

E sempre que a gente encontra uma cópia dele, a gente incrementa o nosso contador.

Mas, quando a gente encontra um elemento maior do que `M`, a gente atualiza a variável `M` e volta o contador para 1.

A coisa fica assim

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

#define  N  10

int  L[N] = { 5, 12, 3, 12, 8, 5, 10, 3, 12, 10 };

int main ()
 {
     int  i, M, cont;

     M  =  L[0];  cont = 1;

     for ( i=1 ; i<N ; i++ )
      {
         if ( L[i] = M )
          {
             cont ++;
          }
         else if ( L[i] > M )
          {
             M  =  L[i];  cont = 1;
      }   }

     printf ("O número de cópias do maior elemento é: %d", cont);
 }
```

◇

f. Encontrando o maior ímpar

Imagine que nós temos uma lista de números armazenada na memória.

Por exemplo,

	0	1	···						N-1
L	18	5	8	11	28	3	6	14	7	20

A tarefa consiste em

→ *Encontrar o maior número ímpar da lista*

No exemplo acima, isso nos daria

```
=>  11
```

Certo.

A dificuldade aqui é que a lista não precisa ter nenhum número ímpar.

E se houver algum, ele não precisa estar na primeira posição.

Então, a gente vai raciocinar em etapas outra vez

- primeiro, a gente encontra o primeiro número ímpar da lista — (*se houver algum*)
- e depois, a gente encontra o maior número ímpar a partir dali

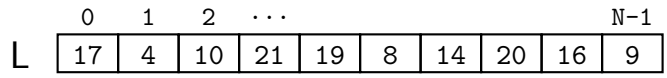
A coisa fica assim

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

#define  N  10

int  L[N] = { 5, 12, 3, 12, 8, 5, 10, 3, 12, 10 };

int main ()
 {
     int  i, Mimpar = 0;


     // 1a ETAPA: procura o primeiro ímpar

     for ( i=0 ; i<N ; i++ )
      {
         if ( L[i] % 2 == 1 )       // Se L[i] é ímpar
          {
             break;                 //   interrompe a procura nesse ponto
      }   }


     // 2a ETAPA: encontra maior ímpar --- (se encontrou o 1o ímpar)

     if ( i < N )                   // Se o dedo não passou do fim da lista
      {
         Mimpar = L[i];
         for ( ; i<N ; i++ )        // começa onde o dedo está
          {
             if ( L[i] % 2 == 1  &&  L[i] > Mimpar )
              {
                 Mimpar = L[i];
      }   }   }

     if ( Mimpar > 0 )
        printf ("O maior número ímpar da lista é: %d", Mimpar);
     else
        printf ("Não existem números ímpares na lista");
 }
```

◇

g. Encontrando o segundo maior

Imagine outra vez que todos os elementos da lista L são diferentes



E considere a seguinte tarefa

→ *Encontrar o segundo maior elemento da lista*

Bom, a idea é raciocinar em etapas outra vez.

- primeiro a gente encontra o maior número M
- e depois a gente encontra o maior elemento que é menor que M

A coisa fica assim

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

#define  N  10

int  L[N] = { 5, 12, 3, 12, 8, 5, 10, 3, 12, 10 };

int main ()
 {
    int  i, M, sM;

    // 1a ETAPA: encontra o maior

    M  =  L[0];
    for ( i=0; i<N; i++ )
     {
         if ( L[i] > M )
             M  =  L[i];
     }

    // 2a ETAPA: encontra o segundo maior

    if ( L[0] == M )    sM  =  L[1];
    else                sM  =  L[0];

    for ( i=0; i<N; i++ )
     {
        if ( L[i] < M  &&  L[i] > sM )
         {
            sM  =  L[i];
     }   }

    printf ("O segundo maior elemento da lista é: %d", sM);
 }
```

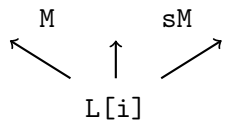
Certo.

Mas também é possível fazer a coisa em uma etapa só.

Quer dizer, a cada momento a gente lembra o maior e o segundo maior números que a gente já viu.

Daí, a gente compara cada elemento `L[i]` a esses dois números.

Existem 3 casos possíveis



E em cada caso, a gente atualiza `M` e `sM` da maneira adequada.

A coisa fica assim

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

#define  N  10

int  L[N] = { 17, 4, 10, 21, 19, 8, 14, 20, 16, 9 };

int main ()
 {
     int  i, M, sM;

     if ( L[0] > L[1] )
      {
         M  =  L[0];   sM  =  L[1];
      }
     else
      {
         M  =  L[1];   sM  =  L[0];
      }

     for ( i=2 ; i<N ; i++ )
      {
         if ( L[i]  >  M )          {  sM  =  M;    M  =  L[i];  }

         else if ( L[i]  >  sM )    {  sM  =  L[i];  }
      }

     printf ("O segundo maior elemento da lista é: %d", sM);
 }
```

◇