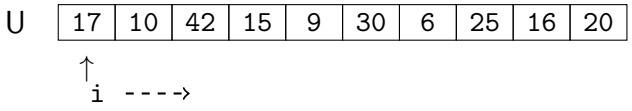
Trabalhando com dedo manualmente
Professor Wladimir

Contando a quantidade de números ímpares

Imagine que você tem uma lista U. E imagine que a tarefa é a seguinte:
Tarefa: Conte a quantidade de números ímpares




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

#define  N  10

int  L[N] = { 17, 10, 42, 15, 9, 30, 6, 25, 16, 20 };

int main ()
{
    int  i;
    int cont;

    i = 0;
    cont = 0;
    while(i < N){
        if(L[i] %2 == 1) cont++:
        i++;
    }

}
```

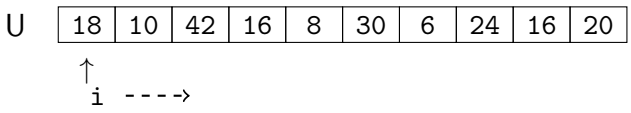
Esse mesmo problema pode ser feito da seguinte maneira:

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

#define  N  10

int  L[N] = { 17, 10, 42, 15, 9, 30, 6, 25, 16, 20 };

int main ()
{
    int  i;
    int cont;

    i = 0;
    cont = 0;
    while(i < N){
        while( i < N && L[i] %2 == 0) i++; //pulando os pares
        if(i < N) // se terminar no vetor
            cont++;
        i++;
    }

}
```

Note que os dois laços while no código são independentes, ou seja, cada um avalia apenas suas próprias condições, sem dependência direta entre eles. Isso implica que o segundo while — responsável por pular os elementos pares — deve verificar explicitamente se o índice i ainda está dentro dos limites do vetor.

Se não incluirmos a condição $i < N$ no segundo while, corremos o risco de acessar uma posição inválida da memória. Por exemplo, imagine que todos os elementos restantes do vetor a partir da posição atual sejam pares. Nesse caso, o segundo while continuará incrementando i até que ele ultrapasse o último índice válido, tentando acessar $L[i]$ mesmo quando $i == N$, o que causa um erro de execução.

Portanto, é fundamental garantir que a condição $i < N$ seja verificada antes de acessar o vetor. Essa verificação evita que o programa acesse posições fora dos limites válidos e garante a segurança na execução.



Colocando um sentinela

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

#define  N  10

int  L[N] = { 17, 10, 42, 15, 9, 30, 6, 25, 16, 20 };

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

    if(L[N-1]%2==0){
        cont = -1;
        L[N-1] = 1;
    }

    while(i < N){
        while( L[i] %2 == 0) i++; //pulando os pares
        if(i < N) // se terminar no vetor
            cont++;
        i++;
    }

}
```

Um sentinela é um valor especial inserido no vetor para marcar uma condição de parada, geralmente usada para:

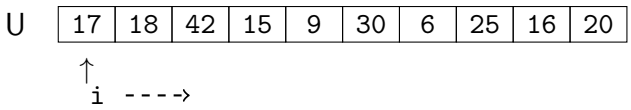
- evitar verificações explícitas de fim de vetor em cada iteração,
- garantir que certos testes lógicos sempre terminem corretamente.

```c
if(L[N-1]%2==0){
    cont = -1;
    L[N-1] = 1; // sentinela: garante que o último elemento é ímpar
}
```

Se o último valor do vetor for par, ele é substituído por 1 (um número ímpar). Esse 1 funciona como um sentinela que garante que o laço while(L[i] % 2 == 0) eventualmente termine, mesmo se os elementos finais forem todos pares.

O cont = -1 é um ajuste para compensar essa modificação artificial (já que esse ímpar foi forçado e não faz parte de um grupo real).

Encontrar o tamanho da maior sequência ordenada



E imagine que a tarefa é a seguinte

Tarefa: Encontrar o tamanho da maior sequência ordenada

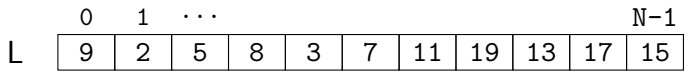
```c
#include <stdio.h>
#include <stdlib.h>

#define  N  10

int  L[N] = { 17, 18, 42, 15, 9, 30, 6, 25, 16, 20 };

int main ()
 {

    int i, seq, max;

    i = 0;
    seq = 0; // tamanho da sequencia ordenada atual
    max = 0; // tamanho da maior sequencia ordenada


    while(i<N){
        //iniciando uma nova sequência ordenada
        seq=1;
        i++;
        while(i<N && L[i] > L[i-1]){
            i++;
            seq++;

        }
        // termina quando saímos do vetor ou quando a sequencia ordenada quebra
        if(seq > max){
            max = seq;
            }

    }
 }

```

Ou assim,

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

#define  N  10

int  L[N] = { 17, 18, 42, 15, 9, 30, 6, 10, 16, 20 };

int main ()
 {

    int i, seq, max;

    seq = 1; // tamanho da sequencia ordenada atual
    max = 1; // tamanho da maior sequencia ordenada


    for(i = 1; i < N; i++){
        if(L[i] > L[i-1]) seq++;
        else seq = 1;
        if(seq > max) max = seq;

    }

    printf("max %d\n", max);
}
```

Lista particionada

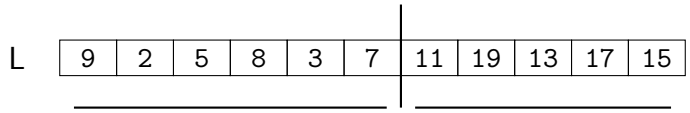
Imagine que nós temos uma lista que armazena números inteiros.
Por exemplo,



A tarefa consiste em

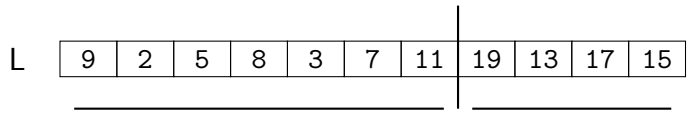
Dado um número k, *verificar se todos os números* $\leq$ k *aparecem antes de todos os números* $>$ k

No exemplo acima, para k $= 10$, *isso nos daria*



=> Sim

Mas, para k $= 18$, isso nos daria



=> Não

A ideia consiste em

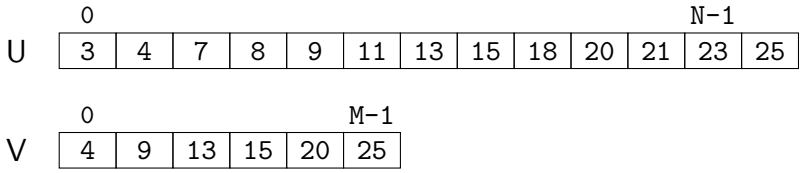
1. percorrer a lista até encontrar o primeiro elemento $>$ k
2. e daí, verificar se todos os elementos a partir daí também são $>$ k

A coisa fica assim

```
1   ( . . . )
2
3   int main()
4    {
5       int     i;              // um dedo
6
7       // 1a Etapa: encontra o 1o elemento > k
8       i = 0;
9       while ( i < N  &&  L[i] <= k )
10         i++;                                // desliza o dedo
11
12      // 2a Etapa: verifica os restantes
13      i++;
14      while ( i < N  &&  L[i] > k )
15      {
16         i++;                                // desliza o dedo
17      }
18
19      if ( i == N )       resp = 1;
20      else                resp = 0;
21   }
```

Lista dentro da lista (ordenada)

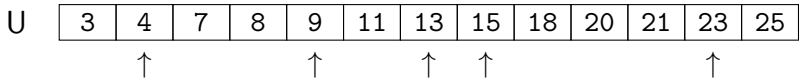
Imagine que nós temos duas listas ordenadas U e V.
Por exemplo,



A tarefa consiste em

Verificar se todos os elementos da lista V *aparecem na lista* U

No exemplo acima, isso nos daria



=> Sim

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

#define  N  12
#define  M  6

int U[N] = { 3, 4, 7, 8, 9, 11, 13, 15, 18, 20, 21, 23, 25 };
int V[M] = { 4, 9, 13, 15, 20, 25 };

int main()
{
 int  i, j;              // dois dedos

 j = 0;                  // posiciona dedo j no início de U
 i = 0;                  // posiciona dedo i no início de V

 while(i < M)
 {
     if(U[j] == V[i]) {
        i++;
        j++;
     }else if(U[j] < V[i]){
        j++;
     }else{
        break;
     }
 }

 if ( i == M )     resp = 1;
 else              resp = 0;
}
```

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

#define  N  12
#define  M  6

int U[N] = { 3, 4, 7, 8, 9, 11, 13, 15, 18, 20, 21, 23, 25 };
int V[M] = { 4, 9, 13, 15, 20, 25 };

int main()
{
 int  i, j;              // dois dedos

 j = 0;                  // posiciona dedo j no início de U
 i = 0;                  // posiciona dedo i no início de V

 while(i < M)
 {
     while( j < N && U[j] != V[i]) j++;

     if(j<N){
        i++;
     }else{
        break;
     }

 }

 if ( i == M )     resp = 1;
 else              resp = 0;
}
```

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

#define  N  12
#define  M  6

int U[N] = { 3, 4, 7, 8, 9, 11, 13, 15, 18, 20, 21, 23, 25 };
int V[M] = { 4, 9, 13, 15, 20, 25 };

int main()
{
 int  i, j;              // dois dedos

 i = 0;                  // posiciona dedo i no início de V
 j = 0;                  // posiciona dedo j no início de U

 while(i < M)
 {
     while( j < N && U[j] < V[i]) j++;

     if(j<N && U[j] == V[i]){
        i++;
     }else{
        break;
     }

 }

 if ( i == M )     resp = 1;
 else              resp = 0;
}
```