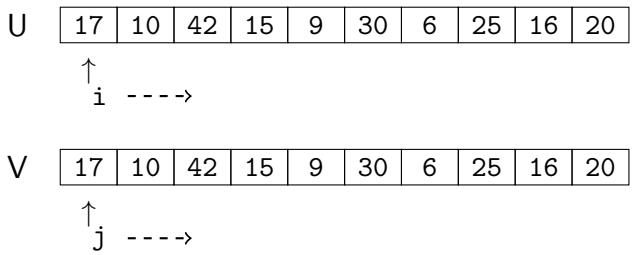
Trabalhando com dois dedos
Professor Wladimir

Listas iguais

Imagine que você tem duas listas U e V do mesmo tamanho.
E imagine que a tarefa é a seguinte:
Tarefa: Verificar se as duas listas contém exatamente os mesmos elementos (na mesma ordem)
Como é que a gente faz isso?
Bom, a gente faz assim

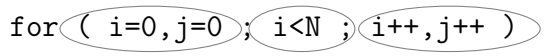
- a gente coloca um dedo no começo da lista U
- e coloca outro dedo no começo da lista V
- daí a gente vai deslizando os dedos até o fim das listas
- e vai comparando o elemento de U com o elemento de V



A coisa fica assim

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

#define  N  10

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

int main ()
{
 int  i, j; // dois dedos
 int dif = 0;  // inicialmente não temos diferença

 for ( i=0,j=0 ; i<N ; i++,j++ )     // os dois dedos andam juntos
  {
     if ( U[i]  !=  V[j] )
      {
          dif = 1;   break;          // encontrou uma diferença
  }   }

 if ( dif == 0 )         printf ("Sim, as listas são iguais");
 else                    printf ("Não, as listas não são iguais");
}

```

A novidade desse programa é o comando `for` com duas variáveis



Mas na prática, dá para escrever o programa com uma variável só
— (*porque* `i` *e* `j` *sempre tem o mesmo valor*)

```
( . . . )

   int  i, dif = 0;                    // apenas um dedo, e uma variável auxiliar

   for ( i=0 ; i<N ; i++ )
    {
       if ( U[i]  !=  V[i] )
        {
            dif = 1;   break;       // encontrou uma diferença
        }
    }

( . . . )
```

Lista U nas posições pares da lista V

Agora imagine que a lista V tem o dobro do tamanho da lista U.
Por exemplo,

U	17	10	42	15	9

V	17	30	10	6	42	25	15	16	9	20

E imagine que a tarefa é a seguinte
Tarefa: Verificar se os elementos da lista U aparecem nas posições pares da lista V
Quer dizer, agora a gente vai ter que

- percorrer a lista U pulando de 1 em 1
- e percorrer a lista V pulando de 2 em 2

Mas como é que a gente faz isso?
Bom, a gente faz isso usando dois dedos.
A coisa fica assim

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

#define  N  10

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

int main ()
 {
    int  i, j, dif = 0;               // dois dedos, e uma variável auxiliar

    for ( i=0,j=0; i<N; i++, j=j+2 )
     {
        if ( U[i] != V[j] )
         {
             dif = 1;    break;
         }
     }

    if ( dif == 0 )    printf ("Sim!, a lista U aparece dentro de V");
    else               printf ("Não!, a lista U não aparece dentro de V");
 }

```

A gente também podia ter escrito o programa assim

```c
 j = 0;
 for ( i=0; i<N; i++ )
  {
     if ( U[i] != V[j] )
      {
          . . . . . .
      }
     j  =  j + 2;           // controle manual da variável j
  }
```

Ou assim,

```
for ( i=0; i<N; i++ )
 {
    j  =  i * 2;              // calculando o valor da variável j

    if ( U[i] != V[j] )
     {
         . . . . . .
     }
 }
```

Ou assim,

```
        for ( i=0; i<N; i++ )
         {
            if ( U[i] != V[2*i] )     // calculando o valor do índice
             {
                 . . . . . .
             }
         }
```

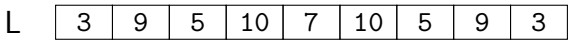
Lista palindromo

Todo mundo sabe que um palíndromo é uma palavra ou frase que é a mesma coisa quando lida nas duas direções.

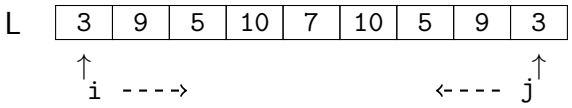
Por exemplo,

`ovo,  arara,  luz azul,  A cara rajada da jararaca`

Ora, uma *lista palíndromo* é uma lista que é a mesma coisa quando lida nas duas direções.
Por exemplo,



Agora, a tarefa é a seguinte
Tarefa: Verificar se L é uma lista palíndromo
Bom, a ideia mais uma vez é usar dois dedos para resolver o problema.
Quer dizer, um dedo vai percorrer a lista da esquerda para a direita.
E o outro dedo vai percorrer a lista da direita para a esquerda



A ideia é que, a cada passo, a gente compara os elementos indicados por `i` e `j`.
E se eles forem sempre iguais, nós temos uma lista palíndromo.
A coisa fica assim

```
( . . . )

#define  N  10

int  L[N] = { 3, 9, 5, 10, 7, 10, 5, 9, 3 };

int main ()
 {
    int  i, j, dif = 0;           // dois dedos, e uma variável auxiliar

    for ( i=0,j=N-1 ; i<N ; i++,j-- )
     {
        if ( L[i] != L[j] )
         {
            dif = 1;    break;
         }
     }
    if ( dif == 0 )    printf ("Sim, L é uma lista palíndromo");
    else               printf ("Não, L não é uma lista palíndromo");
 }
```

A gente pode fazer a esperteza de interromper o laço quando `j` $\leq$ `i`(*porque?*)

```
        int  i,j;       // dois dedos

        for ( i=0,j=N-1 ; i<N ; i++,j-- )
         {
            if ( j <= i )  break;

            . . . . . .
         }
```

A gente pode fazer a esperteza de ir até a metade:

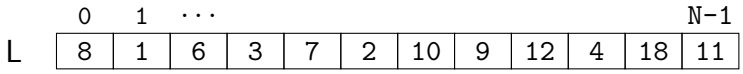
```
        int  i,j;       // dois dedos

        for ( i=0,j=N-1 ; i<N/2 ; i++,j-- )
         {
            if ( j <= i )  break;

            . . . . . .
         }
```

A gente pode fazer a esperteza de continuar testando se $i < j$

```
        int  i,j;       // dois dedos

        for ( i=0,j=N-1 ; i<j ; i++,j-- )
         {
            if ( j <= i )  break;

            . . . . . .
         }
```

Ímpares em ordem crescente

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



Tarefa: A tarefa consiste em Verificar se os elementos ímpares da lista aparecem em ordem crescente

Em outras palavras, nós precisamos verificar se cada elemento ímpar é maior do que o ímpar anterior.

E a gente pode fazer isso deslizando o dedo até o próximo ímapar — (*deixando um dedo no ímpar anterior*)

A coisa fica assim

```c
int main(){
    int  i, j; // dois dedos
    int ok = 1;        //inicialmente

    j = -1;            // posicao do impar anterior

    for(i = 0; i < N; i++){
        if(L[i]%2==1){
            if(j == -1){
                j = i;
            }else if( L[i] > L[j] ){
                j = i;
            }else{
                ok = 0;
                break;
            }
        }
    }
}
```

O código acima pode ser feito assim também:

```c
int main(){
    int  i, j; // dois dedos
    int ok = 1;        //inicialmente

    j = -1;            // posicao do impar anterior

    for(i = 0; i < N; i++){
        if(L[i]%2==1){
            if(j == -1 || L[i] > L[j]){
                j = i;
            }else{
                ok = 0;
                break;
            }
        }
    }
}
```

Mas se quiser transformar no problema de testar se um vetor está ordenado. A gente pode fazer o seguinte:

1. Contar a quantidade de números ímpares
2. Criar um vetor auxiliar
3. Transferir os ímpares para o vetor auxiliar
4. Checar se o vetor auxiliar está ordenado

```
int main(){
    int  i, j; // dois dedos
    int ok;
    int M = 0;

    for(i = 0; i < N; i++) if(L[i] %2 == 1) M++;
    int A[M];
    int pos = 0;
    for(i = 0; i < N; i++){
        if(L[i] %2 == 1) {
            A[pos] = L[i];
            pos++;
        }
    }

    ok = 1;
    for(int i = 1; i < M; i++)
        if(L[i-1] > L[i]){
            ok = 0;
            break;
        }


}
```