Modificando variáveis e checando propriedades

O objetivo dessa aula é exercitar:

1. a manipulação de duas variáveis durante o laço de repetição. (Contando quantas vezes o maior aparece) e (Colocando o maior por último)
2. Checar a propriedades do tipo para todo (Todos os pares).
3. Checar a propriedade do tipo existe um (Existe um par).
4. Checar propriedade entre elementos consecutivos (Lista Ordenada)

Contando quantas vezes o maior aparece

Considere o seguinte problema: Você recebe uma lista L de tamanho n, você quer saber quantas vezes o maior o elemento aparece em L. Por exemplo,

	0	1	2	3	4	5	6	7	8
L	1	10	15	12	30	12	15	30	28

Neste exemplo, o maior elemento é 30 e ele aparece 2 vezes.

A ideia mais direta seria percorrer a lista uma vez para descobrir o maior elemento. Em seguida, contar quantas vezes o maior elemento aparece em L.

```c
#include <stdio.h>

#define N 9
int L[] = {2,3,4,5,7,8,6,8,2};


int main()
{
    int i, maior, cont;
    maior = L[0];
    for(i = 1; i < N; i++){
        if(L[i] > maior){
            maior = L[i];
        }
    }
    cont = 0;
    for(i = 0; i < N; i++){
        if(L[i] == maior){
            cont++;
        }
    }
    printf("maior %d cont %d\n", maior, cont);
    return 0;
}
```

Essa tarefa pode ser realizada com apenas uma passagem na lista. A ideia é atualizar apropriadamente os valores de maior e cont durante a execução.

```c
int main()
{
    int i, maior, cont;
    maior = L[0];
    cont  = 1;
    for(i = 1; i < N; i++){
        if(L[i] > maior){
            maior = L[i];
            cont = 1;
        }else if(L[i]==maior){
            cont++;
        }
    }
    printf("maior %d cont %d\n", maior, cont);
    return 0;
}
```

A execução do algoritmo pode ser visualizada pela seguinte tabela:

i	L[i]	maior	cont
?		1	1
1	10	10	1
2	15	15	1
3	12	15	1
4	30	30	1
5	12	30	1
6	15	30	1
7	30	30	2
8	28	30	2

Durante a execução do programa, os novos valores das variáveis são obtidos pelos valores anteriores.

Colocando o maior na última posição

Considere o seguinte problema: Você recebe uma lista L de tamanho n, você quer que o último elemento da lista seja o maior.

	0	1	2	3	4	5	6	7	8
L	1	10	15	12	30	12	15	30	28

Uma maneira de realizar essa tarefa é encontrar a posição do maior e em seguida realizar a troca do último elemento com a posição do maior.

Isso pode ser realizado da seguinte maneira:

```
int main()
{
    int i, maior, posM;
    maior = L[0];
    posM  = 0;
    //Encontrando o maior elemento
    for(i = 1; i < N; i++){
        if(L[i] > maior){
            maior = L[i];
            posM = i;
        }
    }
    L[posM] = L[N-1];
    L[N-1] = maior;
    imprime(L, N);
    return 0;
}
```

Perguntas:

1. Por que o programa acima encontra a posição do primeiro maior?
2. Por que não precisamos realizar a troca dos valores de `L[posM]` e `L[N-1]` com o auxílio de uma variável auxiliar?

Ordenando colocando o maior por último

```c
#include <stdio.h>
#define N 9
int L[] = {1,4,3,8,2,5,7,6,9};
void imprime(int L[], int N);
int main()
{
    int i, maior, posM;
    for(int j = N; j > 1; j--){
        maior = L[0];
        posM  = 0;
        //Encontrando o maior elemento
        for(i = 1; i < j; i++){
            if(L[i] > maior){
                maior = L[i];
                posM = i;
            }
        }
        L[posM] = L[j-1];
        L[j-1] = maior;
        printf("Passo %d: ", j);
        imprime(L, N);
    }
    return 0;
}
```

A saída desse programa será a seguinte:

```
Passo 9: [ 1 4 3 8 2 5 7 6 9]
Passo 8: [ 1 4 3 6 2 5 7 8 9]
Passo 7: [ 1 4 3 6 2 5 7 8 9]
Passo 6: [ 1 4 3 5 2 6 7 8 9]
Passo 5: [ 1 4 3 2 5 6 7 8 9]
Passo 4: [ 1 2 3 4 5 6 7 8 9]
Passo 3: [ 1 2 3 4 5 6 7 8 9]
Passo 2: [ 1 2 3 4 5 6 7 8 9]
```

Todos os elementos são pares

Considere o seguinte problema: Você recebe uma lista L de tamanho n, você quer saber se todos os elementos de uma lista são pares.

	0	1	2	3	4	5	6	7	8
L	2	4	8	10	10	12	18	21	30

Para verificar se todos os números da lista `L` são pares, precisamos testar cada elemento de `L`. Se encontrarmos ao menos um número ímpar, concluímos que nem todos são pares. Utilizaremos uma variável booleana chamada `todosPares` para armazenar o resultado dessa verificação[1]. Inicialmente, `todosPares` será verdadeiro e permanecerá assim até encontrarmos algum número ímpar.

```
1  int main()
2  {
3      int i, todosPares;
4      todosPares = 1;
5      for(int i = 0; i < N; i++){
6          if(L[i] % 2 == 1){
7              todosPares = 0;
8              break;
9          }
10     }
11     return 0;
12 }
```

[1]Uma variável é booleana quando possui apenas dois valores possíveis: o valor 1 indica que a propriedade é verdadeira, enquanto o valor 0 indica que é falsa.

Existe algum número par

Considere o seguinte problema: Você recebe uma lista L de tamanho n, você quer saber se existe algum elemento par.

	0	1	2	3	4	5	6	7	8
L	2	4	8	10	10	12	18	21	30

Para verificar se existe algum número na lista `L` é par, precisamos testar cada elemento de `L`. Caso encontremos ao menos um número par, concluímos que existe pelo menos um número par. Para armazenar a verificação da propriedade, utilizaremos uma variável booleana chamada `existePar`. Inicialmente, `existePar` será falso e permanecerá assim até encontrarmos algum número par.

```
int main()
{
    int i, existePar;

    existePar = 0;

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

    return 0;
}
```

Checando se uma lista está ordenada

Considere o seguinte problema: Você recebe uma lista L de tamanho n, você quer saber se a lista está ordenada.

	0	1	2	3	4	5	6	7	8
L	2	5	7	9	10	12	15	13	17

Para verificar se a lista está ordenada, precisamos examinar todos os pares de elementos consecutivos. Se todos esses pares estiverem em ordem crescente, então a lista completa estará ordenada. Para realizar essa verificação, utilizaremos uma variável booleana chamada `ordenada`, que será inicialmente definida como verdadeira e permanecerá assim até que encontremos algum par de elementos consecutivos fora de ordem.

O processo de checagem dos pares de elementos consecutivos pode ser feito de duas maneiras:

Com a variável `i` começando em 0 e sendo incrementada enquanto `i < N-1` checando se `L[i] > L[i+1]`.

```c
int main()
{
    int i, ordenada;
    ordenada = 1;
    for(int i = 0; i < N-1; i++){
        if(L[i] > L[i+1]){ // par consecutivo fora de ordem
            ordenada = 0;
            break;
        }
    }

    if(ordenada) printf("ordenado\n");
    else printf("desordenado\n");

    return 0;
}
```

Com a variável `i` começando em 1 e sendo incrementada enquanto `i < N` checando se `L[i-1] > L[i]`.

```c
int main()
{

    int i, ordenada;

    ordenada = 1;

    for(int i = 1; i < N; i++){
        if(L[i-1] > L[i]){ // par consecutivo fora de ordem
            ordenada = 0;
            break;
        }
    }

    if(ordenada) printf("ordenado\n");
    else printf("desordenado\n");

    return 0;
}
```