**4.1** Escreva uma definição da função

```
void range(int vec[], unsigned size, int inicio, int incr)
```

que inicializa elementos de um vector `vec` com `size` valores inteiros `inicio`, `incio+incr`, `incio+2*incr`, etc. seguindo uma progressão aritmética. Exemplo:

```
int a[5];
range(a, 5, 3, 2); /* a[] passa a conter { 3, 5, 7, 9, 11 } */
```

Utilize um ciclo `for` para percorrer os índices válidos e atribuir valores sucessivos aos elementos do vector.

**4.2** Escreva uma função `int ocorre(int vec[], int size, int val)` que procura um valor `val` no elementos de um vector `vec`. Se o valor ocorre como um dos elementos do vector, o resultado deve ser 1; caso contrário deve ser 0.

**4.3** Escreva um programa que lê uma sequência de valores inteiros positivos terminada por -1 e no final escreve toda a sequência mas sem repetidos.

Sugestão: guarde os valores numa variável indexada à medida que são lidos; de cada vez que lê um novo valor pode verificar se já é repetido (e nesse caso não o guarde novamente). No final imprima todos os valores guardados. Pode usar a função do exercício 4.2 para testar se um valor lido ocorre entre os anteriores (i.e., se é repetido).

**4.4** Escreva uma função `int repetidos(int vec[], unsigned size)` que testa se há (pelo menos) dois valores iguais no vector `vec` com tamanho `size`; o resultado deve ser 1 em caso afirmativo e 0 em caso negativo. Exemplos:

```
int a[5] = { 2, -1, 0, 2, -1 };
int b[5] = { 3, 4, 1, 2, -1 };
printf("%d\n", repetidos(a, 5)); // imprime 1
printf("%d\n", repetidos(b, 5)); // imprime 0
```

Tenha atenção que a sua função não modifique os elementos do vector passado como argumento.

**4.5** Escreva uma função `int contar_maiores(int vec[], int size, int val)` cujos argumento são uma variável indexada `vec` com tamanho `size` e um valor `val` e cujo resultado deve ser a contagem do número de elementos de `vec` que são estritamente maiores do que `val`.

**4.6** Escreva uma função `int filtrar_positivos(int vec[], int size)` que remove os valores não positivos (isto é, negativos ou zero) de um vector `vec` com tamanho `size`.

A função deve modificar a variável indexada dada de forma a que os valores positivos fiquem num segmento inicial do vetor. O resultado deve ser o número de valores positivos (i.e. o comprimento do segmento final).

**4.7** Escreva uma função `int ordenada(int vec[], int size)` que testa se uma variável indexada de inteiros está por *ordem ascendente* em sentido lato, isto é, se $vec[i] \leq vec[i+1]$

para todos os índices $i$ de 0 a $size - 2$. O resultado deve ser 1 em caso afirmativo e 0 em caso negativo. A função não deve modificar os valores da variável indexada.

Exemplos: se `vec = {1, 3, 3, 5, 6}` então `ordenada(vec, 5)` deve retornar 1; se `vec = {1, 3, 2, 5, 6}` então `ordenada(vec, 5)` deve retornar 0.

**4.8** Escreva uma função `int desordem(int vec[], int size)` que conta quantos pares de valores numa variável indexada estão fora de ordem, isto é, $vec[i] > vec[i+1]$. Exemplo: se `vec = {3, 1, 2, 2, 4, 0}` e `size=6` então o resultado deve ser 2 (porque $3 \not\leq 1$ e $4 \not\leq 0$).

Note ainda que se a sequência estiver por ordem ascendente, então o resultado é 0 e se estiver por ordem descente, então o resultado é `size` $- 1$.

**4.9** Defina uma função `void sort(int vec[], int n)` que ordena um vetor de inteiros de comprimento $n$ por *ordem ascendente*. Por exemplo: se `vec={3,2,1,3,5}` e `n=5` então depois de executar `sort(vec, n)` devemos ter `vec={5,3,3,2,1}`. `vec={1,2,3,3,5}`.

Sugestão: implemente o algoritmo de ordenação (*seleção*), descrito, por exemplo, em: `https://en.wikipedia.org/wiki/Selection_sort`.