Exercícios de exames anteriores

1) Tenha em consideração a biblioteca STL vector.

   a) Complete a função insert_several(vector<int>& v, int n) que recebe um vetor de inteiros v e um inteiro n. A função deverá inserir n números pares começados em 2 em incrementos de 2, no final do vetor. A função deverá devolver 0 em caso de sucesso ou -1 em caso de erro.

   int insert_several(vector<int>& v, int n)

   b) Complete a função insert_pos(vector<int>& v, int pos, int value) que recebe um vetor de inteiros v e dois inteiros, pos e value. A função deverá inserir o valor value na posição pos do vetor v. A função deverá devolver 0 em caso de sucesso ou -1 em caso de erro.

   int insert_pos(vector<int>& v, int pos, int value)

MT1 de 2023

2) Tenha em consideração a biblioteca STL ***vector***.

   a) Complete a função **range_of_values(**`vector<int>& v, int n1, int n2`**)** que recebe um vetor de inteiros **v** e dois inteiros **`n1 e n2`**. A função deverá retornar um vetor com os números do vetor v que estejam no intervalo dado por n1 e n2(inclusive). A função deverá devolver um vetor vazio em caso de v ser vazio ou em caso de n1 ser maior que n2.

   `vector<int> rangeOfValues(vector<int>& v, int n1, int n2)`

   Quando executado localmente, o ficheiro **ex1.cpp** deve apresentar o seguinte resultado ao testar a implementação da função mencionada.

```
Return Value:
12 24 27
Return Value: Ok, is empty
```

   b) Complete a função **insert_pos(**`vector<int> &v, int pos, int value, int rep`**)** que recebe um vetor de inteiros **v** e três inteiros, **pos**, **`value e rep`**. A função deverá inserir o valor **value** na posição **pos** do vetor **v,** `o numero de vezes indicado em` **rep**. A função deverá devolver 0 em caso de sucesso ou -1 em caso de erro.

   `int insert_pos(vector<int> &v, int pos, int value, int rep)`

   Quando executado localmente, o ficheiro **ex1.cpp** deve apresentar o seguinte resultado ao testar a implementação da função mencionada.

```
Return Value:
5 12 65 34 20 86 24 27 78
Return Value: 0

Return Value:
5 10 10 10 10 12 65 34 86 24 27 78
Return Value: -1
```

Recurso de 2023

3. Tendo em consideração o que aprendeu sobre classes, deverá definir a classe **Book**, incluindo a definição e implementação dos seus métodos.

Nota: Tenha em atenção que deverá construir o protótipo de cada método, ou seja, colocar o tipo de valor de retorno e os parâmetros dos mesmos.

a. Defina a classe **Book**, que deve ter os seguintes atributos e métodos:

Atributos (privados)	
`ISBN`	inteiro que representa o ISBN do livro
`quantity`	inteiro que representa o número de exemplares do livro
`available`	booleano que representa se há ou não exemplares disponíveis
`title`	*string* que representa o título do livro
`author`	*string* que representa o autor do livro
`price`	*float* que representa o preço do livro
`genres`	vetor de *strings* que representa o(s) género(s) do livro

Métodos (públicos)	
Construtor	recebe `ISBN`, `quantity`, `title`, `author` e `price`
`getISBN()`	método get que devolve o `ISBN`
`getQuantity()`	método get que devolve a **`quantity`**
`getAvailable()`	método get que devolve o número de exemplares existentes
Métodos (públicos)	
`getTitle()`	método get que devolve o valor de **`title`**
`getAuthor()`	método get que devolve o valor de **`author`**
`getPrice()`	método get que devolve o valor de **`price`**
`getGenres()`	método get que devolve o valor de **`genres`**
`setPrice()`	método set que recebe um *float* para atualizar **`price`**
`addGenres()`	método que recebe uma *string* com um novo género para adicionar a **`genres`**

Nota: Nesta alínea deverá apenas definir a classe e os métodos, não implementando os mesmos. Não se esqueça de incluir as bibliotecas necessárias para o funcionamento da classe.

b. Implemente o **construtor** da classe **Book** que recebe **ISBN**, **quantity**, **title**, **author** e **price**. Inicialize o objeto de acordo com os parâmetros recebidos verificando se a quantidade e o preço são válidos, ou seja, maiores ou iguais a 0 (se não forem, inicialize-os a zero). Para inicializar o parâmetro **available**, deverá verificar se a quantidade é maior que 0.

c. Implemente os métodos *get*: **getISBN()**, **getQuantity()**, **getAvailable()**, **getTitle()**, **getAuthor()**, **getPrice()**, **getGenres()**. Estes métodos devolvem, respetivamente, **ISBN**, **quantity**, **available**, **title**, **author**, **price** e **genres** de **Book**.

d. Implemente o método **setPrice()** que recebe um *float* para atualizar o **price** de **Book**.

e. Implemente o método **addGenres()** que adiciona a *string* que recebe com um novo género ao vetor **genres** de **Book, no final do vetor**.

MT1 de 2023

4) Tendo em consideração o que aprendeu sobre classes, deverá definir a classe **Car**, incluindo a definição e implementação dos seus métodos.

Nota: Tenha em atenção que deverá construir o protótipo de cada método, ou seja, colocar o tipo de valor de retorno e os parâmetros dos mesmos.

a. Defina a classe **Car**, que deve ter os seguintes atributos e métodos:

Atributos (privados)	
`brand`	*string* que representa a marca do carro
`model`	*string* que representa o modelo do carro
`price`	*float* que representa o preço do carro
`colors`	vetor de *strings* que representa as cores possiveis do carro

Métodos (públicos)	
Construtor	recebe `brand`, `model` e `price`
`getBrand()`	método get que devolveo valor de **brand**
`getModel()`	método get que devolve o valor de **model**
`getPrice()`	método get que devolve o valor de **price**
Métodos (públicos)	

`getColors()`	método get que devolve o valor de **`colors`**
`setPrice()`	método set que recebe um *float* para atualizar **`price`**
`addColors()`	método que recebe uma *string* com uma nova cor para adicionar a **`colors`**

Nota: Nesta alínea deverá apenas definir a classe e os métodos, não implementando os mesmos. Não se esqueça de incluir as bibliotecas necessárias para o funcionamento da classe.

b. Implemente o **construtor** da classe **`Car`** que recebe **`brand`**, **`model`** e **`price`**. Inicialize o objeto de acordo com os parâmetros recebidos verificando se o preço é válids, ou seja, maior ou igual a 0 (se não forem, inicialize-os a zero).

b. Implemente os métodos *get*: **`getBrand()`**, **`getModel()`**, **`getPrice()`**, **`getColors()`**. Estes métodos devolvem, respetivamente, **`brand`**, **`model`**, **`price`** e **`colors`** de **`Car`**.

b. Implemente o método **`setPrice()`** que recebe um *float* para atualizar o **`price`** de **`Car`**.

b. Implemente o método **`addColors()`** que adiciona a *string* que recebe com uma nova cor ao vetor **`colors`** de **`Car, no final do vetor`**.

Recurso de 2023

5. Tendo por base a biblioteca STL, implemente as seguintes alíneas associadas à manipulação de listas ligadas.

a) Implemente a função **`swapHeadAndTail()`** que efetua a troca, entre si, dos elementos na cabeça e na cauda de uma lista ligada.

```
int swapHeadAndTail(list<int> *list_name);
```

A função deverá retornar -1 em caso de erro (verificar se os argumentos são válidos) ou 0 em caso de sucesso.

Quando executado localmente, o ficheiro **`ex5.cpp`** deve apresentar o seguinte resultado ao testar a implementação da função mencionada.

De notar que os elementos da lista são gerados aleatoriamente e, portanto, a sua execução poderá não mostrar elementos iguais aos apresentados abaixo.

```
swapHeadAndTail(): OK
    Elementos da cabeça e da cauda nas posições corretas: OK
    Tamanho igual: OK
    Lista original: -15 -> 61 -> -92 -> 82 -> -79 -> -81 -> 65 -> -90
    Lista após troca: -90 -> 61 -> -92 -> 82 -> -79 -> -81 -> 65 -> -15

swapHeadAndTail(): OK
    Elementos da cabeça e da cauda nas posições corretas: OK
    Tamanho igual: OK
    Lista original: 61 -> 14 -> 73 -> -16 -> -45 -> 57 -> -10 -> 74 -> 90 -> -21
-> 56 -> 77 -> -98 -> -13 -> -42 -> 62 -> 19 -> 93 -> 66 -> 57
    Lista após troca: 57 -> 14 -> 73 -> -16 -> -45 -> 57 -> -10 -> 74 -> 90 -> -
21 -> 56 -> 77 -> -98 -> -13 -> -42 -> 62 -> 19 -> 93 -> 66 -> 61
```

```
swapHeadAndTail(): OK
    Elementos da cabeça e da cauda nas posições corretas: OK
    Tamanho igual: OK
    Lista original: -11 -> 72 -> 38 -> 74 -> -18
    Lista após troca: -18 -> 72 -> 38 -> 74 -> -11

swapHeadAndTail(): OK
    Elementos da cabeça e da cauda nas posições corretas: OK
    Tamanho igual: OK
    Lista original: 47
    Lista após troca: 47
```

b. Implemente a função **`reverseLastElements()`** que inverte os últimos **`k`** elementos de uma lista ligada (**`k`** representa um número inteiro não negativo).

`int reverseLastElements(list<int> *list_name, int k);`

A função deverá retornar -1 em caso de erro (verificar se os argumentos são válidos) ou 0 em caso de sucesso.

Quando executado localmente, o ficheiro **`ex5.cpp`** deve apresentar o seguinte resultado ao testar a implementação da função mencionada.

De notar que os elementos da lista são gerados aleatoriamente e, portanto, a sua execução poderá não mostrar elementos iguais aos apresentados abaixo.

```
reverseLastElements(): ERRO

reverseLastElements(): ERRO

reverseLastElements(): OK
    Tamanho igual: OK
    Lista original: 57 -> 14 -> 73 -> -16 -> -45 -> 57 -> -10 -> 74 -> 90 -> -21
-> 56 -> 77 -> -98 -> -13 -> -42 -> 62 -> 19 -> 93 -> 66 -> 61
    Lista após inversão: 57 -> 14 -> 73 -> -16 -> -45 -> 57 -> -10 -> 74 -> 90 -
> -21 -> 56 -> 77 -> -98 -> -13 -> 61 -> 66 -> 93 -> 19 -> 62 -> -42

reverseLastElements(): OK
    Tamanho igual: OK
    Lista original: -18 -> 72 -> 38 -> 74 -> -11
    Lista após inversão: -18 -> 72 -> 38 -> 74 -> -11
```

MT1 de 2023

6. Tendo por base a biblioteca STL, implemente a função **`removeElements()`** que remove de uma pilha os elementos que se encontram num vetor.

```
int removeElements(stack<string> &q, vector<string> &v);
```

A função deverá retornar -1 em caso de erro (verificar se os argumentos não são vazios) ou 0 em caso de sucesso.

Quando executado localmente, o ficheiro **`ex6.cpp`** deve apresentar o seguinte resultado ao testar a implementação da função mencionada.

Sugestão: utilize uma pilha auxiliar para percorrer a pilha.

```
Stack depois de removidos os elementos:
Filipe - Ana - Gustavo - Paulo - Teresa - Luis - Isabel - João
 O return da função: 0
```

7. Tendo por base a biblioteca STL, implemente as seguintes alíneas associadas à manipulação de listas ligadas.

Implemente a função **removeElements()** que remove de uma lista todos os elementos maiores que n.

```
int removeElements(list<int> *list1, int n);
```

A função deverá retornar -1 em caso de erro (verificar se os argumentos são válidos) ou 0 em caso de sucesso.

Quando executado localmente, o ficheiro **ex7.cpp** deve apresentar o seguinte resultado ao testar a implementação da função mencionada.

```
original list:
24 -> 14 -> 4 -> 22 -> 2 -> 30 -> 1
return value: 0
final list:
4 -> 2 -> 1
original list:
24 -> 14 -> 4 -> 22 -> 22 -> 3 -> 1
return value: 0
final list:
4 -> 3 -> 1
return value: -1
```

8) Tenha em consideração o que aprendeu sobre apontadores e referências e a estrutura:

```
struct Rectangle{
    int width;
    int height;
}
```

que representa um retângulo.

a. Complete a função **scale_rectangle_ptr(Rectangle *rectangle, int factor)** que recebe um apontador para uma estrutura **Rectangle** e um inteiro. A função deverá ampliar o retângulo num fator **factor** e deverá devolver 0 em caso de sucesso ou -1 em caso de erro.

```
int scale_rectangle_ptr(Rectangle *rectangle, int factor)
```

b. Complete a função **scale_rectangle_ref(Rectangle &rectangle, int *factor)** que recebe uma estrutura **Rectangle** por referência e um apontador para um inteiro. A função deverá ampliar o retângulo num fator **factor** e deverá devolver 0 em caso de sucesso ou -1 em caso de erro.

```
int scale_rectangle_ref(Rectangle &rectangle, int *factor)
```

c. Complete a função **copy_scaled_rectangle(Rectangle *rectangle, int factor)** que recebe um apontador para uma estrutura **Rectangle** e um inteiro. A função deverá criar uma cópia ampliada do retângulo num fator **factor** e deverá devolver o apontador para a cópia em caso de sucesso ou nullptr em caso de erro.

```
Rectangle *copy_scaled_rectangle(Rectangle *rectangle, int factor)
```

d. Complete a função **swap_dims_rectangles(Rectangle *rectangle1, Rectangle &rectangle2)** que recebe duas estruturas **Rectangle**: a primeira por apontador e a segunda por referência. A função deverá trocar as dimensões entre os retângulos (a largura e altura de um passam a ser a largura e altura do outro. Deverá devolver 0 em caso de sucesso ou -1 em caso de erro.

```
int swap_dims_rectangles(Rectangle *rectangle1, Rectangle &rectangle2)
```

MT1 de 2024

9) Considere que lhe foi atribuído o desenvolvimento de um programa em C++ que integra dados de um ficheiro CSV. Este representa dados de jogadores de basquetebol num determinado jogo. A estrutura Player encontra-se definida para representar cada jogador.

```
struct Player {
  int number;
  string name;
  double pointsPerGame;
}
```

a) Implemente a função **insertFromCSV**(), que lê um ficheiro de texto e armazena os dados num vetor de elementos do tipo Player. Por cada linha lida, deve ser imprimida uma mensagem do tipo "Player *player_number* was read.", em que player_number corresponde ao número do jogador lido.

Em cada linha, o ficheiro contém a seguinte informação sobre cada jogador: number,name,points_per_game.

Quando executado localmente, o ficheiro **team.cpp** deve apresentar o seguinte resultado ao testar a implementação da função mencionada.

```
Player 23 was read
Player 33 was read
Player 91 was read
Player 7 was read
Player 13 was read
```

b) Implemente a função **printContent**(), que deve imprimir o conteúdo de um vetor de elementos do tipo Player (um jogador por linha, imprimindo, sequencialmente, o número, nome e pontos do jogador). Por fim, deve ainda imprimir o número **total** de jogadores e a **média** de pontos marcados pela equipa.

Quando executado localmente, o ficheiro **team.cpp** deve apresentar o seguinte resultado ao testar a implementação da função mencionada.

```
23: Michael Jordan. Points = 30
33: Scottie Pippen. Points = 19
```

```
91: Dennis Rodman. Points = 7
7: Toni Kukoc. Points = 14
13: Luc Longley. Points = 9
Number of players: 5
Average points in the match: 15.8
```

MT1 de 2024