4ªaula prática - Pesquisa e Ordenação de Vetores

Faça download do ficheiro ***aeda2021_p04.zip*** da página da disciplina e descomprima-o (contém a pasta ***lib,*** a pasta ***Tests*** com os ficheiros ***carPark.h, carPark.cpp***, ***sequentialSearch.h, insertionSort.h*** e ***tests.cpp***, e os ficheiros ***CMakeLists*** e ***main.cpp***)

- Note que alguns testes unitários deste projecto (alíneas b e f) estão comentados. Retire os comentários quando implementar os testes.
- Deverá realizar a ficha respeitando a ordem das alíneas.

Enunciado

1. Considere o problema de gestão de um parque de estacionamento, já enunciado na aula 1. As classes **InfoCard** e **CarPark** são apresentadas a seguir, possuindo agora novos membros:

```
class InfoCard {
public:
   string name;
   int frequency;  //new data member
   bool present;
};

class CarPark {
  unsigned freePlaces;
   const unsigned capacity;
   vector<InfoCard> clients;
   const unsigned numMaxClients;
public:
   CarPark(unsigned cap, unsigned nMaxCli);
   unsigned getNumPlaces() const;
   unsigned getNumMaxClients() const;
   unsigned getNumOccupiedPlaces() const;
   unsigned getNumClients() const;
   vector<InfoCard> getClients() const;
   bool addClient(const string & name);
   bool removeClient(const string & name);
   bool enter(const string & name);
   bool leave(const string & name);
   int clientPosition(const string & name) const;
   int getFrequency(const string &name) const;
   InfoCard getClientAtPos(unsigned p) const;
   void sortClientsByFrequency();
   void sortClientsByName();
   vector<string> clientsBetween(unsigned f1, unsigned f2);
   friend ostream & operator<<(ostream & os, const CarPark & cp);
};
```

a) Reimplemente o membro função:

```
int CarPark::clientPosition(const string &name) const
```

que retorna o índice do cliente de nome *name* no vetor clientes. Se o cliente não existir, retorna -1. Para efetuar a pesquisa no vetor clientes, use o método de **pesquisa sequencial** estudado nas aulas (código em *sequentialSearch.h*).

b) Na classe **InfoCard** o novo membro dado *frequency* guarda o número de vezes que o cliente usou o parque. Altere as funções já implementadas de forma a atualizar convenientemente este membro dado.
Implemente também o membro função:

```
int CarPark::getFrequency(const string &name) const
```

que retorna o número de vezes que o cliente de nome *name* utilizou o parque. Se o cliente não existir, lança uma exceção do tipo *ClientDoesNotExist*.
Implemente a classe *ClientDoesNotExist* e **note que** o tratamento desta exceção efetua uma chamada ao membro-função *getName()* que deve retornar o nome do cliente que não existe e originou a exceção.

c) Implemente o membro função:

```
void CarPark:: sortClientsByFrequency()
```

que ordena o vetor clientes por ordem decrescente de frequência de utilização do parque, desambiguando (clientes com mesmo número de utilizações) por ordem crescente do nome. Use o método de **ordenação por inserção** estudado nas aulas (código em *insertionSort.h*).

d) Implemente o membro-função

```
vector<string> CarPark::clientsBetween(unsigned f1, unsigned f2)
```

que retorna um vetor com o nome de todos os clientes que utilizaram o parque >= *f1* vezes e <= *f2* vezes (*nota:* o vetor a retornar deve estar ordenado de acordo com o critério enunciando na alínea anterior).

e) Por vezes é útil obter informação dos clientes ordenados por nome. Usando um algoritmo de ordenação à sua escolha, implemente o membro-função:

```
void CarPark::sortClientsByName()
```

que ordena o vetor clientes por ordem crescente de nome.

f) Implemente o operador << :

```
ostream & operator << (ostream &os, const CarPark &cp)
```

Deve imprimir no monitor informação sobre todos os clientes registados, mostrando o nome do cliente, se está presente ou não no parque e o número de vezes que utilizou o parque.
Implemente também o membro-função:

```
InfoCard CarPark::getClientAtPos(unsigned p) const
```

que retorna o cliente (*InfoCard*) existente no índice *p* do vetor *clients*. Se não existir tal cliente, lança uma exceção do tipo *PositionDoesNotExist*.
Implemente a classe *PositionDoesNotExist* e **note que** o tratamento desta exceção efetua uma chamada ao membro-função *getPosition()* que deve retornar a posição do vetor inválida que originou a exceção.

.

2. Analise e calcule a complexidade dos seguintes excertos de código:

a)
```
void imprime_matriz(int largura, int altura, int ntabs) {
    num = 1;
    for(int a = 1 ; a <= altura ; a++) {
        cout << "[";
        for(int l = 1 ; l <= largura ; l++) {
            cout << num++;
            for(int t = 1 ; t <= ntabs ; t++) cout << "\t";
        }
       cout << "]" << endl;
    }
    cout << endl;
}
```

b)
```
int pesquisa (int v[], int size, int x) {
    int left = 0;
    int right = size-1;
    while (left <= right) {
        int middle = (left + right) / 2;
        if (x == v[middle])
            return middle; // encontrou
        else if (x > v[middle])
            left = middle + 1;
        else
            right = middle -1;
    }
    return -1;
}
```