4ªaula prática - Pesquisa e Ordenação de Vetores

Faça download do ficheiro ***aeda1920_fp04.zip*** da página da disciplina e descomprima-o (contém a pasta ***lib,*** a pasta ***Tests*** com os ficheiros ***parque.h, parque.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 programa *"Parque Estacionamento"*, para gestão de um parque de estacionamento, já enunciado na aula 1. As classes **InfoCartao** e **ParqueEstacionamento** são apresentadas a seguir, possuindo agora novos membros:

```
class InfoCartao {
public:
   string nome;
   int frequencia;  //novo membro
   bool presente;
};

class ParqueEstacionamento {
unsigned int vagas;
   const unsigned int lotacao;
   vector<InfoCartao> clientes;
   unsigned int numClientes;
   const unsigned int numMaximoClientes;
public:
   ParqueEstacionamento(unsigned int lot, unsigned int nMaxCli);
   ~ParqueEstacionamento();
   bool adicionaCliente(const string &nome);
   bool retiraCliente(const string &nome);
   bool entrar(const string &nome);
   bool sair(const string &nome);
   int getFrequencia(const string &nome) const;    // novo membro
   unsigned int getNumLugares() const;
   unsigned int getNumLugaresOcupados() const;
   int posicaoCliente(const string &nome) const;    // a alterar
  InfoCartao getClienteAtPos(int p) const;     // novo membro
   void ordenaClientesPorFrequencia();            // novo membro
  void ordenaClientesPorNome();            // novo membro
   vector<string> clientesGamaUso(int n1, int n2);   // novo membro
   friend ostream & operator << (ostream &os, const ParqueEstacionamento &pe);
};
```

a) Reimplemente o membro função:

```
int ParqueEstacionamento::posicaoCliente(const string &nome) const
```

que retorna o índice do cliente de nome *nome* 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.

b) Na classe **InfoCartao** o novo membro dado *frequencia* 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 ParqueEstacionamento::getFrequencia(const string &nome) const
```

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

c) Implemente o membro função:

```
void ParqueEstacionamento::ordenaClientesPorFrequencia()
```

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.

d) Implemente o membro-função

```
vector<string> ParqueEstacionamento::clientesGamaUso(int n1, int n2)
```

que retorna um vetor com o nome de todos os clientes que utilizaram o parque >= *n1* vezes e <= *n2* vezes (*nota:* o vector a retornar deve estar ordenado de acordo com o critério enunciando na alínea anterior).

e) Pretende-se manter uma ordenação dos clientes quer por frequência de utilização do parque (alínea c), quer por nome. Usando um algoritmo de ordenação à sua escolha, implemente o membro-função:

```
void ParqueEstacionamento::ordenaClientesPorNome()
```

que ordena o vetor clientes por ordem crescente de nome.

f) Implemente o operador << :

```
ostream & operator << (ostream &os, const ParqueEstacionamento &pe)
```

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:

```
InfoCartao ParqueEstacionamento::
        getClienteAtPos(vector<InfoCartao>::size_type p) const
```

que retorna o cliente (*InfoCartao*) existente no índice *p* do vetor *clientes*. Se não existir tal cliente, lança uma exceção do tipo *PosicaoNaoExistente*.
Implemente a classe *PosicaoNaoExistente* e **note que** o tratamento desta exceção efetua uma chamada ao membro-função *getValor()* 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;
}
```