Ficha de Trabalho – 1

Objectivos: Construção de uma classe (**Ordenacao**) onde estão implementados os vários algoritmos de ordenação e daí tirar conclusões em relação ao número de trocas e ao tempo de execução.

Exercício 1:

Observação: *Todos os métodos devem ser implementados tendo em vista a ordenação de vetores de inteiros. No entanto, a implementação deve estar feita de modo a que facilmente seja generalizada.*

1. Pretende-se que o aluno implemente a classe Ordenacao, com vários métodos de ordenação de vectores. Todos os métodos de ordenação devem ser estáticos.

2. Pretende-se fazer no final uma comparação entre os vários métodos de ordenação de modo a concluir qual o mais eficiente. A comparação dos métodos deve ser feita em termos de trocas efectuadas entre os elementos e o tempo despendido. Essas métricas devem ser gravadas em ficheiro CSV, de modo a que no Excel se possa fazer os gráficos que permitem analisar a complexidade de cada algoritmo:

Exemplo do ficheiro "FicheiroOutput.csv"

Metodo	**N**	**NTrocas**	**Tempo**
Bubble	100	1234	0.1235
Bubble	1000	123400	0.8935
Etc.			

// ----------

Exemplo da class Ordenacao:

```
class Ordenacao
{
        public: static void Bubblesort(int *V, int N, string ficheiro_output);
        ….
}
```

Exemplo do uso da class num programa

```
void main()
{
        …….

        Ordenacao::Bubblesort(VectorDados, Tamanho, "FicheiroOutput.csv");

}
```

Exercício 2:

Pretende-se trabalhar com a class list do C++ - STL

(Usar o mais possível os métodos disponíveis na class list)
(Antes de implementar qualquer funcionalidade, verifique se a list já dispõem de métodos que o podem ajudar!)
(Faça uma pesquisa na net por "list in c++")

Considere uma lista de strings. Implemente as funcionalidades seguintes:

a) Declarar uma list de strings;
b) Implemente uma função para ler de um ficheiro de texto todas as palavras e colocar na lista;
c) Contar todas as palavras;
d) Listar as palavras que estão na lista;
e) Listar ao contrário o conteúdo da lista;
f) Eliminar uma palavra da lista (dada a palavra);
g) Eliminar uma palavra da lista (dada uma posição da lista);
h) Verificar se existem palavras repetidas;
i) Contar o numero de palavras de tamanho inferior a 5;
j) Passar todas as palavras da lista para maiúsculas;
k) Gravar num ficheiro de texto as palavras de tamanho superior a 10.

Exercício 3:

Pretende-se implementar as classes XMLWriter e XMLReader.
Estas classes disponibilizam funcionalidades, Métodos que permitem efectuar a gravação e a leitura de ficheiros XML.

Exemplo da classe XMLWriter.

```
class XMLWriter
{
…….
        public:
                XMLWriter() {…..}
                ~XMLWriter() {…..}
                void WriteStartDocument(string ficheiro) {…..}
                void WriteEndDocument(){…..}
                void WriteElementString(string el,  string valor) {…..}
                void WriteStartElement(string el) {…..}
                void WriteEndElement() {…..}
}
```

Exemplo da utilização da classe XMLWriter

```
void main()
{
        XMLWriter XX;

        XX.WriteStartDocument("FicheiroDados.xml");
        XX.WriteStartElement("DADOS");

        XX.WriteStartElement("PESSOA"); // Abre o Elemento "PESSOA"
        XX.WriteElementString("NOME","Jose Miguel");
        XX.WriteElementString("IDADE","18");
        XX.WriteEndElement(); // Fecha o Elemento "PESSOA"

        XX.WriteStartElement("PESSOA"); // Abre o Elemento "PESSOA"
        XX.WriteElementString("NOME","Pedrito De Portugal");
        XX.WriteElementString("IDADE","65");
        XX.WriteEndElement(); // Fecha o Elemento "PESSOA"

        XX.WriteEndElement(); // Fecha o Elemento "DADOS"
        XX.WriteEndDocument(); // Fecha o Elemento Documento
}
//Ficheiro "FicheiroDados.xml"
<DADOS>
        <PESSOA>
                <NOME>Jose Miguel</NOME>
                <IDADE>18</IDADE>
        </PESSOA>
        <PESSOA>
                <NOME>Pedrito De Portugal </NOME>
                <IDADE>65</IDADE>
        </PESSOA>
</DADOS>
```