Ficha 6

Programação (L.EIC009)

Objectivos

- Introdução ao uso e definição de classes e objectos em C++.

Recursos

Slides das aulas teóricas: HTML PDF

1

Tenha em conta o uso de objectos da classe `std::string` para representar strings, e em particular as funções membro `length()` e `at()` para resolver os problemas a seguir.

1.1

Escreva um programa que lê um objecto `std::string` via `std::cin` e escreve em `std::cout` as posições e caracteres de uma string da forma exemplificada a seguir Exemplo de execução:

```
Enter string: abcde
0: a
1: b
2: c
3: d
4: e
```

Esqueleto parcial:

```
#include <iostream>
#include <string>

int main(void) {
  std::cout << "Enter string: ";
  std::string str;
```

```
  std::cin >> str;
  ...
  return 0;
}
```

1.2

Escreva uma função

```
bool is_palindrome(const std::string& s)
```

que retorne `true` se `s` é um palíndromo e `false` caso contrário.

1.3

Escreva uma função

```
int replace(std::string& s, char a, char b)
```

para substituir todas as ocorrências de `a` em `s` por `b` e retorne o nº de substituições feitas.

2

Tenha em conta o uso de `std::vector` na resolução dos seguintes problemas.

2.1

Escreva um programa que toma as strings passados como argumentos via `argc/argv` a um programa (excepto o nome do programa em `argv[0]`), e depois imprime as strings ordenadas alfabeticamente.

Use um objecto `std::vector<std:string>` em linha com o seguinte esqueleto e dicas dadas em comentários:

```
#include <algorithm>
#include <iostream>
#include <string>
#include <vector>

int main(int argc, char** argv) {
  std::vector<std::string> v;
```

```
  // Processa argc/argv colocando
  // argumentos do programa em v.
  // DICA: Use a função
  // membro push_back() por exemplo.
  ...

  // Ordena vector
  std::sort(v.begin(), v.end());

  // Imprime argumentos ordenados para std::cout
  // DICA: pode usar ciclo for-each
  ...

  return 0;
}
```

Exemplo de execução:

```
$ ./ex2_1 prog leic fcup feup
fcup
feup
leic
prog
```

2.2

Re-escreva o programa anterior para ler as strings de um ficheiro de texto cujo nome é passado como argumento ao programa.

Use um objecto `std::ifstream` para ler o ficheiro.

```
#include <algorithm>
#include <fstream>
#include <iostream>
#include <string>
#include <vector>

int main(int argc, char** argv) {
  std::ifstream in(argv[1]);  // Abre ficheiro de input
  std::vector<std::string> v;
  std::string s;
  while (getline(in, s)) {
    ...
  }
  ...
```

```
}
```

Para um ficheiro `x.txt` contendo (deverá considerar uma linha inteira em correspondência a uma string):

```
prog
leic
fcup
feup
```

então passando `x.txt` como argumento ao programa, deveremos obter output similar ao do exercício anterior.

2.3

Re-escreva o programa anterior para escrever as strings para um ficheiro de output em vez de `std::cout` em que o nome do ficheiro de output é passado como segundo argumento ao programa.

Use `std::ofstream` para escrever o ficheiro.

```
#include <algorithm>
#include <fstream>
#include <iostream>
#include <string>
#include <vector>

int main(int argc, char** argv) {
  std::ifstream in(argv[1]);  // Abre ficheiro de input
  std::vector<std::string> v;
  ...
  std::sort(v.begin(), v.end());
  std::ofstream out(argv[2]); // Abre ficheiro de output
  ...
}
```

3

Nota: semelhante a exercício da Ficha 5 mas envolvendo o uso de `std::vector` em vez de arrays.

Escreva o código das seguintes "templates" para funções `max_value` e `norm_values` tal que:

- `max_value(v)` devolve o valor máximo dos `n` elementos em `v` - pode assumir que `v` tem pelo menos um elemento;
- `norm_values(vec, n, min, max)` normaliza o valor dos `n` elementos em `v` mediante a conversão de valores inferiores a `min` em `min`, valores superiores a `max` em `max`, e não alterando outros valores compreendidos entre `min` e `max` - por ex. a normalização de `{ 2, -1, 2, 5, 1, 4}` para valores entre `0` e `3` deverá levar a `{ 2, 0, 2, 3, 1, 3}`).

```
template <typename T>
T max_value(const std::vector<T>& v) {
  ...
}
template <typename T>
void norm_values(std::vector<T>& v, T min, T max) {
 ...
}
```

Escreva um programa que teste o código usando um vector de valores de tipo `int` e outro vector de valores de tipo `double`, por ex. algo como:

```
std::vector<int> iv { 2, -1, 2, 5, 1, 4};
int imax = max_value(iv);
norm_values(iv, 0, 3);
...
std::vector<double>dv { -1.2, 0.5, 1.3, 3.2, -0.7, 1.1 };
double dmax = max_value(dv);
norm_values(dv, -1.0, 1.0);
...
```

4

No conjunto de problemas a seguir considera-se a definição de uma classe `coord3d` para coordenadas tri-dimensionais.

```
namespace leic {
  class coord3d {
    ...
  };
}
```

4.1

Define a classe tendo em conta os seguintes requisitos:

- A classe é parte do namespace `leic`.
- A classe tem campos membro `x`, `y` e `z` de tipo `int` para as coordenadas 3D.
- Os campos membro deverão ter visibilidade `private`, as restantes definições referidas abaixo deverão ter visibilidade `public`.
- A classe tem seguintes constructores:
  - `coord3d()` construtor por omissão, inicializa todas as coordenadas a `0`;
  - `coord3d(int vx, int vy, int vz)` em que os argumentos são usados para inicialisar os campos membro;
  - `coord3d(const coord3d& c)`: construtor por cópia, que inicialisa os campos membro por cópia de valores de `c`;
- A classe tem um destrutor `~coord3d()` com corpo vazio.
- Existem funções membro `int get_x() const` e `void set_x(int vx)` para respectivamente obter e modificar o valor da coordenada `x`, e funções membro análogas para `y` e `z`.

4.2

Defina o operator `<<` em associação a obectos `std::ostream&` e `coord3d` numa função **fora** da classe `coord3d`, por forma a que sejam impressas as coordenadas para o output `out` no formato `(x,y,z)`:

```
std::ostream& operator<<(std::ostream& out, const coord3d& c) {
  ...
  return out;
}
```

4.3

Defina os operadores `=` para atribuição de coordenadas e `+` e `+=` com o intuito de somar coordenadas como funções membro na classe `coord3d`. Consegue implementar `+` usando o construtor de cópia e o operador `+=` ?

```
class coord3d {
  coord3d& operator+=(const coord3d& c) {
    ...
```

```
    return *this;
  }
  coord3d operator+(const coord3d& c) const {
     ...
  }
```

5

5.1

Defina uma class `dmatrix` para representar matrizes de tipo `double` por form a que:

- internamente a classe deverá alocar dinamicamente (com `new`) espaço para guardar os valores bem como um array de apontadores para as linhas a matriz (ver abaixo);
- o espaço alocada deverá ser libertado com `delete` no destrutor da classe;
- as linhas e colunas da matriz devem ser obtidos via funções membro `lines()` e `cols()`
- as funções membro `at` devem permitir aceder a elementos da matriz para leitura ou escrita;
- as funções `fill` e `fill_diagonal` deverão permitir preencher um valor para, respectivamente, todas as entradas da matriz e apenas para a diagonal da matriz;
- a função `transpose` deve alterar a matriz por forma a obter-se a matriz transposta (dica: deverá precisar de alocar novos arrays para os dados e copiar os valores anteriores antes de libertar a memória que usam).

Esqueleto:

```
#include <cassert>
namespace leic {
  class dmatrix {
  private:
    double** values;
    ... // campos para nº de linhas e colunas
  public:
    dmatrix(int lines, int cols) {
      assert(lines > 0 && cols > 0);
      ...
      values = new double*[lines];
      double *data = new double[lines * cols];
      for (int i = 0; i < lines; i++) {
```

```
        values[i] = data + i * cols;
        for (int j = 0; j < cols; j++) {
          values[i][j] = 0.0;
        }
      }
    }
    ~dmatrix() {
      // Liberta memória com delete!
      ...
    }
    int lines() const {
      ...
    }
    int cols() const {
      ...
    }
    double at(int l, int c) const {
      ...
    }
    double& at(int l, int c) {
      ...
    }
    void fill(double v) {
      ...
    }
    void fill_diagonal(double v) {
      ...
    }
    void transpose() {
      ...
    }
  };
}
```

5.2

Escreva um programa que leia um inteiro `n` e imprima a matriz identidade de grau `n`. Use `fill_diagonal()`. Defina (**fora da classe `dmatrix`, não pode definir como função membro**) o operator `<<` para imprimir um objecto `dmatrix`.

```
std::ostream&
operator<<(std::ostream& out, const dmatrix& m) {
  ...
  return out;
}
```

Exemplo de execução e possível formato de impressão:

```
Enter n? 3
1 0 0
0 1 0
0 0 1
```

5.3

Defina um construtor de cópia e uma implementação para operador de atribuição (`=`) na classe `dmatrix` . Note que a implementação do operador `=` não deve assumir que `m` tem a mesma dimensão que `this` , e nesse caso deve libertar o espaço previamente ocupado por `this` .

```
namespace leic {
  class dmatrix {
     ...
     dmatrix(const dmatrix& m) {
       ...
     }
     ...
     dmatrix& operator=(const dmatrix& m) {
       ...
       return *this;
     }
};
```

Experimente o efeito do seu código, por ex. com um fragmento:

```
dmatrix a(3, 3);
a.fill_diagonal(1.0);
dmatrix b = a; // uso de construtor de cópia
dmatrix c(4,4);
c = a; // uso de operador de atribuição
std::cout << a << b << c;
```

6

Converta o código do exercício anterior de forma a ter uma classe "template" `matrix<T>` . As declarações das funções membro deverão ser análogas (substituindo o uso do tipo `double` pelo tipo genérico `T`) com excepção do

construtor que deve ter um parâmetro inicial `initial_value` designando o valor inicial para preencher todas as posições da matriz.

```
namespace leic {
  template typename T>
  class matrix {
  private:
    T** values;
    ... // campos para nº de linhas e colunas
  public:
    matrix(int lines, int cols, const T& initial_value) { ... }
    ...
    T at(int l, int c) const { ... }
    T& at(int l, int c) { ... }
    void fill(T v) { ... }
    void fill_diagonal(T v) { ... }
    ...
  };
}
```

7

Implemente a seguinte classe template para uma fila de duplo sentido ("double ended queue" ou "deque") que usa internamente uma lista duplamente ligada de elementos.

```
namespace leic {
    template <typename T>
    class deque {
    private:
        struct Node {
            T value;
            Node* prev;
            Node* next;
        };
        int l_size;
        Node* head;
        Node* tail;
    public:
        deque();
        ~deque();
        int size() const;
        bool empty() const;
        void add_first(const T& value);
        void add_last(const T& value);
        T remove_first();
```

```
        T remove_last();
    };
};
```