11ª aula prática – Filas de Prioridade

Faça download do ficheiro ***aeda1819_fp11.zip*** da página da disciplina e descomprima-o (contém os ficheiros ***Caixa.h, Caixa.cpp, MaquinaEmpacotar.h, MaquinaEmpacotar.cpp, Test.cpp***)
Deverá realizar esta ficha respeitando a ordem das alíneas. Poderá executar o projeto como CUTE Test quando quiser saber se a implementação que fez é suficiente para passar no teste correspondente.

Enunciado

Nesta quadra natalícia, uma loja decidiu inovar para tornar mais eficiente o envio dos seus produtos. Para isso, comprou um empacotador automático que coloca os objetos em caixas segundo o peso de cada objeto e a capacidade remanescente de cada caixa. Suponha, portanto, a existência de um conjunto de caixas de capacidade de carga (peso máximo que suporta) C, e objetos o_1, o_2, …, o_n com peso p_1, p_2, …, p_n, respetivamente. O objetivo da máquina é empacotar todos os objetos sem ultrapassar a capacidade de carga de cada caixa, usando o menor número possível de caixas. Implemente um programa para resolver este problema, de acordo com a seguinte estratégia:

- Comece por colocar primeiro os objetos mais pesados;
- Coloque o objeto na caixa mais pesada, que ainda possua carga livre para conter este objeto.

Guarde os objetos a serem empacotados numa fila de prioridade (priority_queue<Objeto>), ordenada por peso do objeto. Guarde as caixas numa fila de prioridade (priority_queue<Caixa>), ordenada pela menor carga ainda disponível na caixa.

As classes **Objeto**, **Caixa**, e **MaquinaEmpacotar** estão parcialmente definidas no ficheiro ***Empacotador.h***, como indicado:

```
class Objeto {
      unsigned id;
      unsigned peso;
public:
      Objeto(unsigned idObj, unsigned pesoObj);
      unsigned getID() const;
      unsigned getPeso() const;

      bool operator < (const Objeto& o1) const;
      friend ostream& operator<<(ostream& os, Objecto obj);
};
```

```
typedef stack<Objeto> STACK_OBJS;

class Caixa {
    STACK_OBJS objetos;
    unsigned id;
    unsigned capacidadeCarga;
    unsigned cargaLivre;
    static unsigned ultimoId;
public:
    Caixa(unsigned cap=10);
    unsigned getID() const;
    unsigned getCargaLivre() const;
    void addObjeto(Objeto& obj);

    bool operator < (const Caixa& c1) const;
    string imprimeConteudo() const;
};


typedef priority_queue<Objeto> HEAP_OBJS;
typedef priority_queue<Caixa> HEAP_CAIXAS;

class MaquinaEmpacotar {
      HEAP_OBJS objetos;
      HEAP_CAIXAS caixas;
      unsigned capacidadeCaixas;
public:
      MaquinaEmpacotar(int capCaixas = 10);
      unsigned numCaixasUsadas();
      unsigned addCaixa(Caixa& cx);
      HEAP_OBJS getObjetos() const;
      HEAP_CAIXAS getCaixas() const;

      unsigned carregaPaletaObjetos(vector<Objeto> &objs);
      Caixa procuraCaixa(Objeto& obj);
      unsigned empacotaObjetos();
      string imprimeObjetosPorEmpacotar() const;
      Caixa caixaMaisObjetos() const;
};
```

a) Implemente o membro-função:

```
unsigned MaquinaEmpacotar::carregaPaletaObjetos(vector<Objeto> &objs)
```

que lê de um vetor fornecido os objetos a serem empacotados. Apenas os objetos com peso igual ou inferior à capacidade das caixas são carregados na máquina. A função retorna o número de objetos efetivamente carregados na máquina, sendo a paleta (vetor **objs**) atualizada com a retirada dos objetos que são carregados. Guarde os objetos na fila de prioridade **objetos**, ordenando-os por peso (o primeiro elemento da fila de prioridade é o objeto mais pesado).

b) Implemente o membro-função:

```
Caixa MaquinaEmpacotar::procuraCaixa(Objeto& obj)
```

Esta função procura na fila de prioridade **caixas** a próxima caixa com carga remanescente suficiente para guardar o objeto **obj**. Se essa caixa existir, retira-a da fila de prioridade e retorna-a. Caso não exista uma caixa com carga livre suficiente para alojar **obj**, a máquina usa uma nova caixa para o objeto, retornando-a.

c) Implemente o membro-função:

```
unsigned MaquinaEmpacotar::empacotaObjetos()
```

que guarda os objetos da paleta carregada no menor número possível de caixas, e retorna o número de caixas utilizadas. Considere que inicialmente nenhuma caixa está a ser usada.

d) Implemente o membro-função:

```
string MaquinaEmpacotar::imprimeObjetosPorEmpacotar() const
```

que retorna uma *string* contendo o ID e respetivo peso dos objetos por empacotar, que estão na fila **objetos** (a informação de cada objeto é separada por \n). Consulte o teste para verificar a formatação da `string`. Note que o operador << já está implementado na classe `Objeto`. Caso não existam objetos para empacotar, a função retorna a string "`Nao ha objetos!`"

e) Implemente o membro-função:

```
string Caixa::imprimeConteudo() const
```

que retorna uma *string* contendo o ID da caixa, assim como os respetivos ID e peso dos objetos que a caixa contém. A `string` a retornar deve ser da forma "C[<ID> <InfoObj1> <InfoObj2> …]". Consulte o teste para verificar a formatação da `string`. Note que o operador << já está implementado na classe `Objeto`. Caso não existam objetos na caixa, a função retorna a string "`Caixa <ID> vazia!`"

f) Implemente o membro-função:

```
Caixa MaquinaEmpacotar::caixaMaisObjetos() const
```

que encontra na fila de prioridade **caixas** aquela que contém o maior número de objetos, retornando-a. Se não houver caixas na lista, a função lança uma exceção do tipo **MaquinaSemCaixas** (implementada na classe `MaquinaEmpacotar`).