## 10ª aula prática – Filas de Prioridade

Faça download do ficheiro ***aeda2021_p10.zip*** da página da disciplina e descomprima-o o (contém a pasta ***lib***, a pasta ***Tests*** com os ficheiros ***box.h, box.cpp***, ***packagingMachine.h, packagingMachine.cpp*** e ***tests.cpp***, e os ficheiros ***CMakeLists*** e ***main.cpp***)

Deverá realizar a ficha respeitando a ordem das alíneas.

## 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) ***boxCapacity***, e objetos $o_1$, $o_2$, …, $o_n$ com peso $w_1$, $w_2$, …, $w_n$, 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<Object>), ordenada por maior peso. Guarde as caixas numa fila de prioridade (priority_queue<Box>), ordenada pela menor carga ainda disponível na caixa.

As classes **Object**, **Box**, e **PackagingMachine** estão parcialmente definidas como indicado:

```
class Object {
      unsigned id;
      unsigned weight;
public:
      Object(unsigned i, unsigned w);
      unsigned getID() const;
      unsigned getWeight () const;

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

```
typedef stack<Object> StackObj;

class Box {
    StackObj objects;
    unsigned id;
    unsigned capacity;
    unsigned free;
    static unsigned lastId;
public:
    Box(unsigned cap=10);
    unsigned getID() const;
    unsigned getFree() const;
    void addObject(Objeto& obj);

    bool operator < (const Box& b1) const;
    string printContent() const;
};


typedef priority_queue<Object> HeapObj;
typedef priority_queue<Box> HeapBox;

class PackagingMachine {
      HeapObj objects;
      HeapBox boxes;
      unsigned boxCapacity;
public:
      PackagingMachine(int boxCap = 10);
      unsigned numberOfBoxes();
      unsigned addBox (Box& b1);
      HeapObj getObjects() const;
      HeapBox getBoxes() const;

      unsigned loadObjects(vector<Object> &objs);
      Box searchBox(Object& obj);
      unsigned packObjects();
      string printObjectsNotPacked() const;
      Box boxWithMoreObjects() const;
};
```

a) Implemente o membro-função:

```
unsigned PackagingMachine::loadObjects(vector<Object> &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 **objects**, ordenando-os por peso (o primeiro elemento da fila de prioridade é o objeto mais pesado).

b) Implemente o membro-função:

```
Box PackagingMachine::searchBox(Object& obj)
```

Esta função procura na fila de prioridade **boxes** 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**, cria uma nova caixa, retornando-a.Nota: não coloque **obj** na caixa.

c) Implemente o membro-função:

```
unsigned PackagingMachine::packObjects()
```

que guarda os objetos (presentes na fila **objects**) no menor número possível de caixas. Retorna o número de caixas utilizadas. Considere que inicialmente nenhuma caixa está a ser usada.

d) Implemente o membro-função:

```
string PackagingMachine::printObjectsNotPacked() const
```

que retorna uma *string* contendo o ID e respetivo peso dos objetos por empacotar, que estão na fila **objects** (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 `Object`. Caso não existam objetos para empacotar, a função retorna a string "`No objects!`"

e) Implemente o membro-função:

```
string Box::printContent() 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 "Box <ID> [ <InfoObj1> <InfoObj2> … ]". Consulte o teste para verificar a formatação da `string`. Note que o operador << já está implementado na classe `Object`.

Caso não existam objetos na caixa, a função retorna a string "`Box <ID> empty!`"

f) Implemente o membro-função:

```
Box PackagingMachine::boxWithMoreObjects() const
```

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