Practical 10 – Priority Queues

Download the ***aeda2021_p10.zip*** file from the moodle page and unzip it (contains the ***lib*** folder, the ***Tests*** folder with files ***box.h, box.cpp, packagingMachine.h*** and ***packagingMachine.cpp*** and ***tests.cpp*** and the ***CMakeLists*** and ***main.cpp*** files)

You must perform the exercise respecting the order of the questions.

Exercise

In this Christmas season, a store decided to innovate to make the delivering of its products more efficient. The store manager bought an automatic packaging machine that places the objects in boxes according to the weight of each object and the remaining capacity of each box. Suppose the existence of a set of boxes with load capacity (maximum weight supported) **boxCapacity**, and objects o_1, o_2, …, o_n of weight w_1, w_2, …, w_n, respectively. The purpose of the machine is to pack all objects without exceeding the capacity of each box, using as few boxes as possible. Implement a program to solve this problem, according to the following strategy:

- Start by placing the heaviest objects first;
- Place the object in the heaviest box, which still has free space to contain this object.

Store the objects to be packed in a priority queue (`priority_queue <Object>`), ordered by highest weight. Store the boxes in a priority queue (`priority_queue <Box>`), ordered by the lowest free space still available in the box

The classes **Object**, **Box**, and **PackagingMachine** are partially defined below:

```
typedef stack<Object> StackObj;
typedef priority_queue<Object> HeapObj;
typedef priority_queue<Box> HeapBox;

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);
};
```

```
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;
};

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) Implement in the **PackagingMachine** class the following member function:

```
unsigned PackagingMachine::loadObjects(vector<Object> &objs)
```

This function reads the objects to be packaged from the given vector **objs**, and stores them in the priority queue **objects**, ordered by weight (the first element in the priority queue is the heaviest object). Only objects weighting less than or equal to the capacity of the boxes are considered, and the vector **objs** is updated with the removal of these objects. The function returns the number of objects actually stored in the priority queue **objects.**

b) Implement in the **PackagingMachine** class the following member function:

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

This function searches the priority queue **boxes** for the next box with enough remaining space to store the **obj** object. If such a box exists, removes it from the priority queue and returns it. If there is not a box with sufficient free space to store **obj**, creates a new box returning it. Note: do not store **obj** in the box.

c) Implement in the **PackagingMachine** class the following member function:

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

This function stores the objects (that are in the queue **objects**) in the fewest possible boxes. Returns the number of boxes used. Consider that initially no box is being used.

d) Implement in the **PackagingMachine** class the following member function:

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

This function returns a *string* containing the ID and weight of the objects to be packed, which are in the queue **objects** (the information for each object is separated by \n). See the unit test to check the formatting of the string. Note that the `operator <<` is already implemented in the `Object` class. If there are no objects to pack, the function returns the string `"No objects!"`.

e) Implement in the **Box** class the following member function:

```
string Box::printContent() const
```

This function returns a string containing the box ID, as well as the respective ID and weight of the objects in the box. The string to be returned should be formatted as "Box <ID> [<InfoObj1> <InfoObj2>…]". See the unit test to check the formatting of the string. Note that the operator << is already implemented in the `Object` class.

If there are no objects in the box, the function returns the string `"Box <ID> empty!"`

f) Implement in the **PackagingMachine** class the following member function:

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

This function searches in the queue **boxes** the box that contains the largest number of objects, returning it. If there are no boxes in the list, the function throws an exception of type `MachineWithoutBoxes` (implemented in the `PackagingMachine` class).