Practical 6 – Stacks and Queues

Instructions

- Download the ***aeda2021_p06.zip*** file from the moodle page and unzip it (contains the ***lib*** folder, the ***Tests*** folder with files ***stackExt.h, counter.h***, ***counter.cpp, exceptions.h*** and ***tests.cpp*** as well as some test support – *cycle.h* and *performance.h*, and the ***CMakeLists*** and ***main.cpp*** files)
- You must perform the exercise respecting the order of the questions

Exercise

1. Write the **StackExt** class, which implements a stack structure with a new method: `findMin`. The `findMin` method returns the value of the smallest element in the stack and must be performed in constant time, O (1). You should implement the following methods:

```
– bool empty() const     //checks if the stack is empty
– T &top()                //returns the element at the top of the stack
– void pop()              //removes element
– void push(const T & val)     //adds element
– T &findMin()                  //returns the smallest element
```

Suggestion: Use the *stack* class from the STL library. Use two stacks: one to store all the values, and another to store the minimum values as they appear.
Note: unit tests can be time consuming.

2. We want to develop a simulator for a wrapping counter. The counter includes a queue of clients to be attended (*clients*). Clients arrive randomly at certain times and wait in queue to be attended. A client's service time varies with the number of gifts he has to wrap. The member data *actualTime* indicates the current instant (clock) of the simulation. The member data *nextEnter* indicates the instant when the next client arrives at the queue (arrivals are generated randomly). The member data *nextLeave* indicates the time when the first client in the queue will be served.

Create a **Client** class with the following members (empty constructor must randomly generate a number of gifts between 1 and 5).

```
class Client {
    unsigned numGifts;
public:
    Client();
    unsigned getNumGifts() const;
};
```

Create another **Counter** class that simulates the progress of a client queue.

```
class Counter {
   queue<Client> clients;
   const unsigned wrappingTime;   // time it takes to wrap a gift
   unsigned nextEnter, nextLeave; // time when the next arrival/departure
                                  // client takes place
   unsigned actualTime;           // actual simulation time
   unsigned numAttendedClients;
public:
    Counter(unsigned wt=2);          // wt = wrapping time
    unsigned getActualTime() const;
    unsigned getNextEnter() const;
    unsigned getNumAttendedClients() const;
    unsigned getWrappingTime() const;
    unsigned getNextLeave() const;
    Client & getNextClient();
    void enter();
    void leave();
    void nextEvent();
    friend ostream & operator << (ostream & out, const Counter & c1);
};
```

a) Implement the empty constructor of the **Client** class that should randomly generate a number of gifts between 1 and 5. Implement the member function `getNumGifts()` which returns the number of gifts of the customer (*numGifts*).

b) Implement the constructor of the **Counter** class. It must initialize the simulation time to zero (*actualTime*), randomly generate the arrival time of the next client (*nextEnter*) with a value between 1 and 20 and the next leaving (*nextLeave*) to zero. Implement the member functions of the **Counter** class:

- `unsigned getActualTime() const //returns the simulation actual time`
- `unsigned getNextEnter() const //returns arrival time of the next client`
- `unsigned getNumAttendedClients() const //number of attended clients`
- `unsigned getWrappingTime() const //returns` *`wrappingTime`*
- `unsigned getNextLeave() const //returns` *`nextLeave`*
- `Cliente & getNextClient() //returns the next cliente of the queue. If the queue is empty, should throw the` **`EmptyQueue`** `exception. This exception contains the method` *`string getMsg() const`*`, which returns the string "Empty Queue"`

c) Implement the member function:

```
void Counter::enter()
```

This function simulates the arrival of a new client to the queue and must:

- Create a new cliente and insert it in the queue.
- Randomly generate the arrival time of the next client (*nextEnter*) with a value between 1 and 20.
- Update the departure time of the next client (*nextLeave*) if necessary (that is, if the quele is empty before this arrival). The next output is equal to *actualTime* + *numGifts* of the new client created * *wrappingTime*
- Write on the monitor the actual time and the information about the cliente who arrived in the quele (see unit tests to know the content and format of the information)

d) Implement the member function:

```
void Counter::leave()
```

This function simulates the leaving of a client from the queue and must:

- Handle the exception conveniently, in case the queue is empty (suggestion: use the *getNextClient()* function to get the client to leave)
- Remove the first client from the queue
- Update the leaving time of the next client (*nextLeave*)
- Write on the monitor the actual time and the information about the client who left the queue (see unit tests to know the format and content of the information)

e) Implement the member function:

- `void Counter::nextEvent()`

This function invokes the next event (enter or leave), according to the values of *nextEnter* and *nextLeave*. It also updates the *actualTime* value accordingly

Note: the unit test does not fail, check the console for the values.

f) Implement the write operator (`operator <<`). This function should print on the monitor the number of attended clients and the number of clients waiting.

Note: the unit test does not fail, check the console for the values.