

Data Structures and Algorithms Lab

Lab 06

Marks 05

Instructions

Work in this lab individually. You can use your books, notes, handouts etc. but you are not allowed to borrow anything from your peer student.

Marking Criteria

Show your work to the instructor before leaving the lab to get some or full credit.

What you must do

Program the following task in your C++ compiler and then compile and execute them. *Write the main function first and keep on testing the functionality of each function once created.*

Task 01

Write the implementation of following generic **Stack** class; it should provide the standard stack structure of *LIFO (Last-in First-out)* as discussed in the class.

```
template <class T>
class Stack
{
public:
    //constructor
    Stack(const int MAX_SIZE = 5);

    //destructor
    ~Stack();

    //stack manipulation operations
    void push(const T newItem);    //push a new item
    void pop();                    //pop an item
    void clear();                  //clear the stack

    //stack accessor
    T getTop() const;              //return item at the top

    //stack status operations
    bool isEmpty() const;          //is stack empty?
    bool isFull() const;           //is stack full?

    //outputs the data in stack. If the stack is empty, outputs "Empty Stack".
    void showStructure() const;

private:
    //Data members
    T* data;                       //array of items to be allocated dynamically as per MAX_SIZE
    int top;                       //top of the stack
    const int MAX_SIZE;            //maximum capacity of the stack
};
```

Write **main** function and create some objects of **Stack** for various data types (e.g., int, float, string) and test all the implemented functions. The **showStructure** function must display the **stack status** with its **top** pointing to the correct location on the console.

Sample run:

```
stack.Push(5.0);
stack.Push(6.5);
stack.showStructure();

stack.Push(-3.0);
stack.Push(-8.0);
stack.showStructure();

stack.Pop();
stack.Pop();
stack.showStructure();
```

```
top -->      6.5
           5
-----

top -->      -8
           -3
           6.5
           5
-----

top -->      6.5
           5
-----
```

Note: Ensure that you handle errors gracefully and provide meaningful error messages when necessary.

Task 02

Write implementation of following generic **Queue** class; it should provide the standard queue structure of *FIFO (First-in first-out)* as discussed in the class.

```
template <class T>
class Queue
{
public:
    //constructor
    Queue(const int MAX_SIZE = 5);

    //destructor
    ~Queue();

    //queue manipulation operations
    void enqueue(const T newItem); //add an element to the rear of queue
    void dequeue();                //delete element at the front of queue
    void clear();                  //clear the queue

    //queue accessors
    T getFront() const;           //return element at the front
    T getRear() const;           //return element at the rear

    //queue status operations
    bool isEmpty() const;        //is queue empty?
    bool isFull() const;         //is queue full?

    //outputs the data in queue. If the queue is empty, outputs "Empty Queue".
    void showStructure() const;

private:
    //data members
    T* data;                      //array of items to be allocated dynamically as per MAX_SIZE
    int front;                    //front index
    int rear;                     //rear index
    const int MAX_SIZE;          //maximum capacity of queue
};
```

Write **main** function and create some objects of **Queue** for various data types (e.g., int, float, string), test all the implemented functions. The **showStructure** function must display the **queue status** with its *front* and *rear* pointing to the correct locations on the console.

Sample run:

```
queue.enqueue(5.0);
queue.enqueue(6.5);
queue.showStructure();
```

```
front -->      5
              6.5      <-- rear
```

```
queue.enqueue(-3.0);
queue.enqueue(-8.0);
queue.showStructure();
```

```
front -->      5
              6.5
              -3
              -8      <-- rear
```

```
queue.dequeue();
queue.dequeue();
queue.showStructure();
```

```
front -->      -3
              -8      <-- rear
```

Note: Ensure that you handle errors gracefully and provide meaningful error messages when necessary.

😊😊😊 **BEST OF LUCK** 😊😊😊
