Homework 5

100points

DUE DATE: ***Tuesday*** October 25th by 11:59pm

Warning: This homework is a programming assignment. Please remember that:

Your code must compile and run on Black server. If your code does not work on Black server as submitted the grade for that problem is 0 regardless of its quick fix(es). Always test your code on Black, even if it is incomplete, make sure to get your code to compile and run on our Servers.

Problem

In this homework you will be working with stacks and queues.

You will be implementing stacks using arrays and then you will be implementing a queue class.

There are 3 parts to your project:

Part 1: A stack will be implemented using an array. (This part is essential for Part 2 and Part 3)

Part 2: Using the stack implemented in in Part 1 you will then design what is called " a min stack" class that supports the following operations:

push,
pop,
gettop,
and get **minimum** element.

All these operations (including the get minimum value operation) should have run time efficiency of O(1).

Part 3: Using the stack class you implemented in Part 1, you will now design a queue that will support the following operations:

push,
pop,
gettop
and isEmpty operations.

There is no efficiency requirement for building this class but it would be wise to keep your algorithms to be mostly constant, no bigger than linear time.

If you are using any code segment that is inspired from your lecture notes please add a comment block at the top of your code segment and make reference to that particular lecture note.

For Example:

//following code segment is generated from LectureX slide X

or

//following code segment is generated from Class Book page XXX

void mystack<T>::doSomething(T const&elem)

Note that: Pre and post condition documentation is required for all the methods that is being asked for you to implement in this homework.

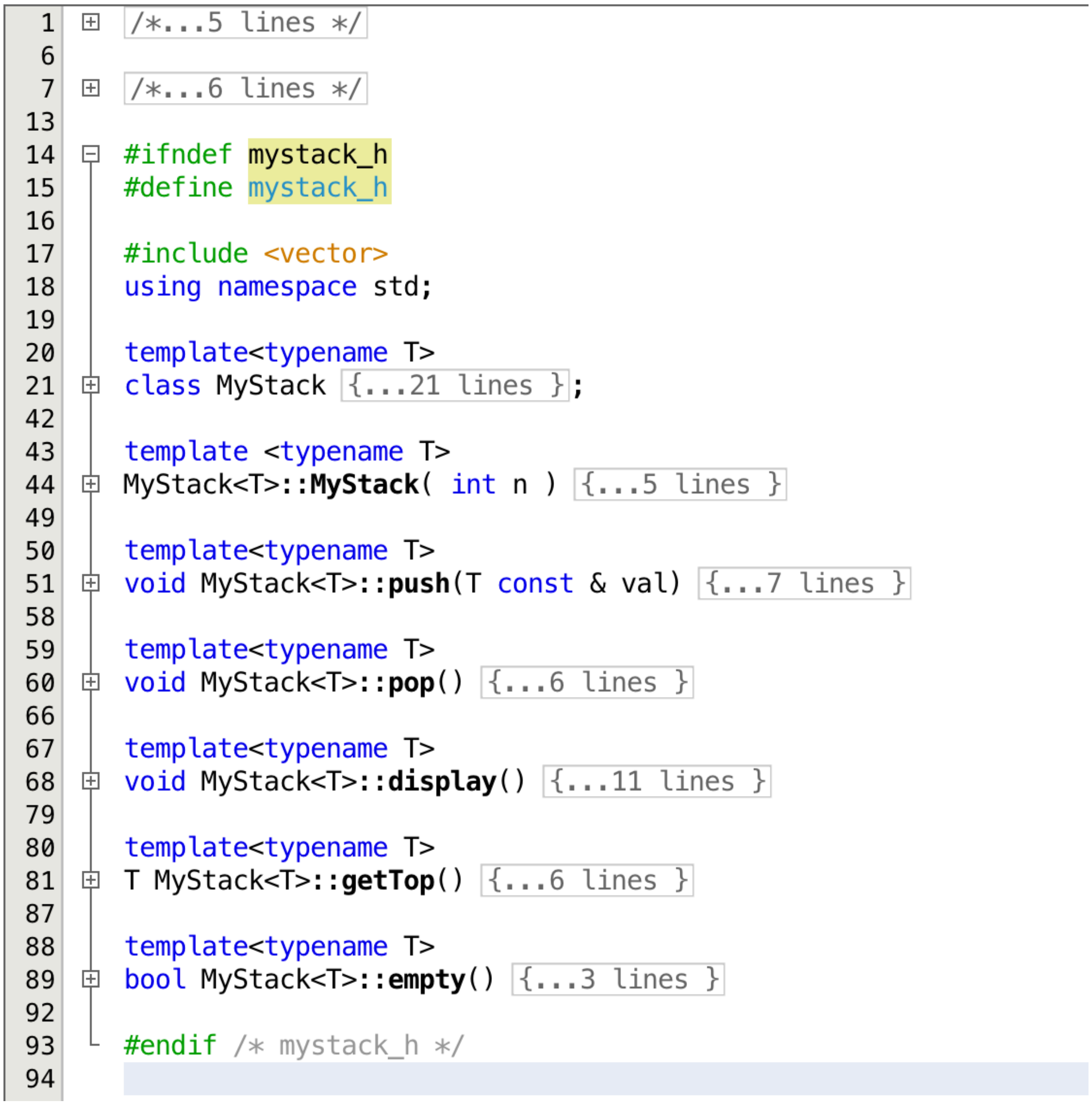
Part I:

Following is the list of required methods for your **Stack** class. It will be built as a **<u>template</u>**.

Please include **overflow** and **underflow exception classes** as a sub class in your Stack class and throw theses exceptions as you see it necessary (Hint: attempting to pop from an empty stack requires an exception being thrown)

- You will be responsible of implementing the following methods with given signatures:
    1. **push**: This method adds a new element to the top of the stack, throws overflow when necessary
    2. **pop**: This method deletes the top element of the stack throws underflow when necessary.
    3. **display**: prints stack of elements on console and lists one item per line (see sample run for formatting your output…)
    4. **empty**: This method tests to see if the stack is empty.
    5. **gettop**: This method simply returns the top element of the stack, it should not delete the element from stack.
    6. **<u>It is a requirement that you add pre and post conditions </u>(as comments)<u>to each method listed above, see lecture notes for guidance</u>.**

For more guidance for this class see screen shot below:



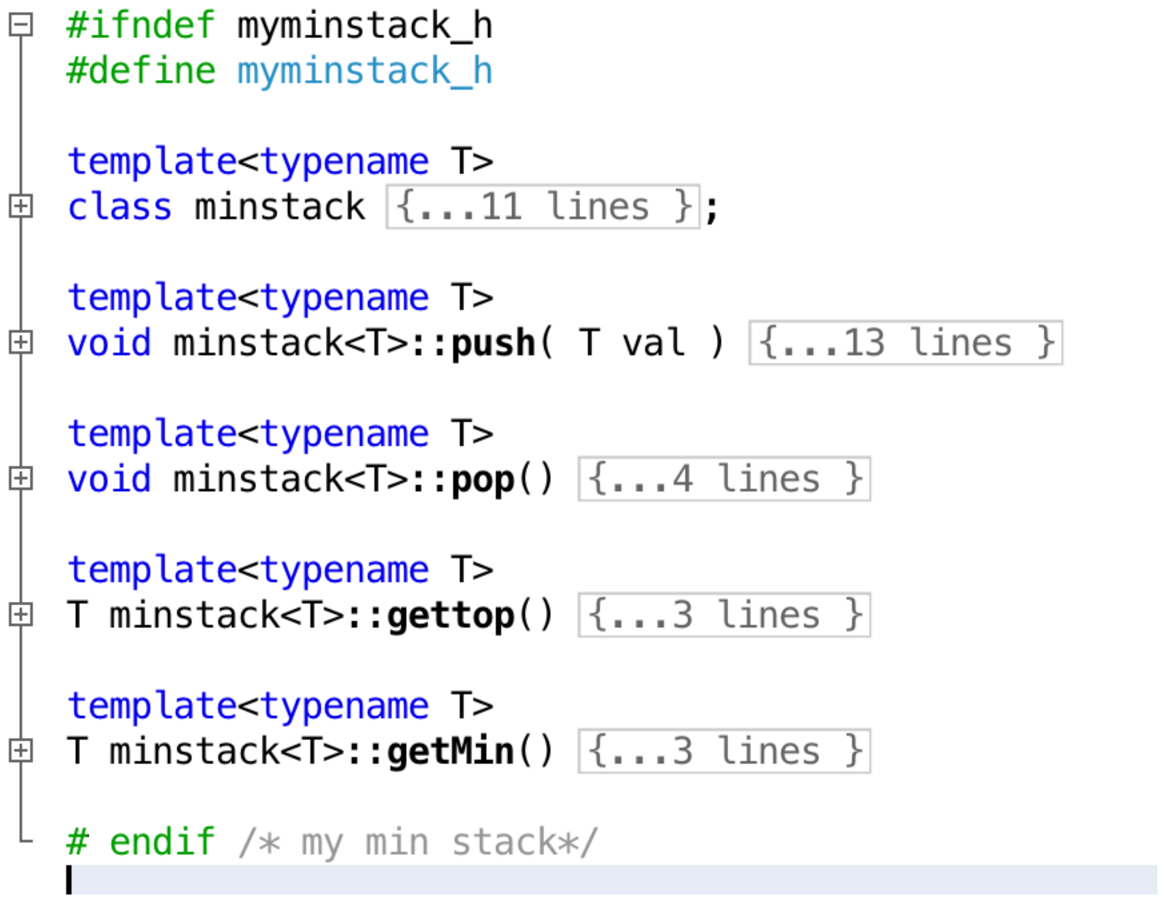
```
/*...5 lines */

/*...6 lines */

#ifndef mystack_h
#define mystack_h

#include <vector>
using namespace std;

template<typename T>
class MyStack {...21 lines };

template <typename T>
MyStack<T>::MyStack( int n ) {...5 lines }

template<typename T>
void MyStack<T>::push(T const & val) {...7 lines }

template<typename T>
void MyStack<T>::pop() {...6 lines }

template<typename T>
void MyStack<T>::display() {...11 lines }

template<typename T>
T MyStack<T>::getTop() {...6 lines }

template<typename T>
bool MyStack<T>::empty() {...3 lines }

#endif /* mystack_h */
```

Part II:

Using the Stack class that you built from Part 1. You are asked to build a min stack class. (Hint: In order to keep the methods to operate at constant time **O(1).** You might want to consider using an extra stack for some of the operations below.)

1. **push**: This method adds a new min element to the top of the min stack, (remember this is a **min** stack so the new element you are about to push must contain a lesser or equal value than what is already on top of the stack). Requirement: your method should have run time efficiency of **O(1).**
2. **pop**: This method deletes the top element of the min stack. Requirement: this method should have run time efficiency of **O(1)**.
3. **getmin**: This method retrieves the minimum element, this method should have run time efficiency of **O(1).**
4. **gettop**: This method returns the top element of the min stack, without removing it, this method should have run time efficiency of **O(1).**
5. **It is a requirement that you add pre and post conditions (as comments) to each method listed above, see lecture notes for guidance**.

This class does not have to be implemented as a template, however it must use mystack class to complete its required operations. See below screen shot for guidance:



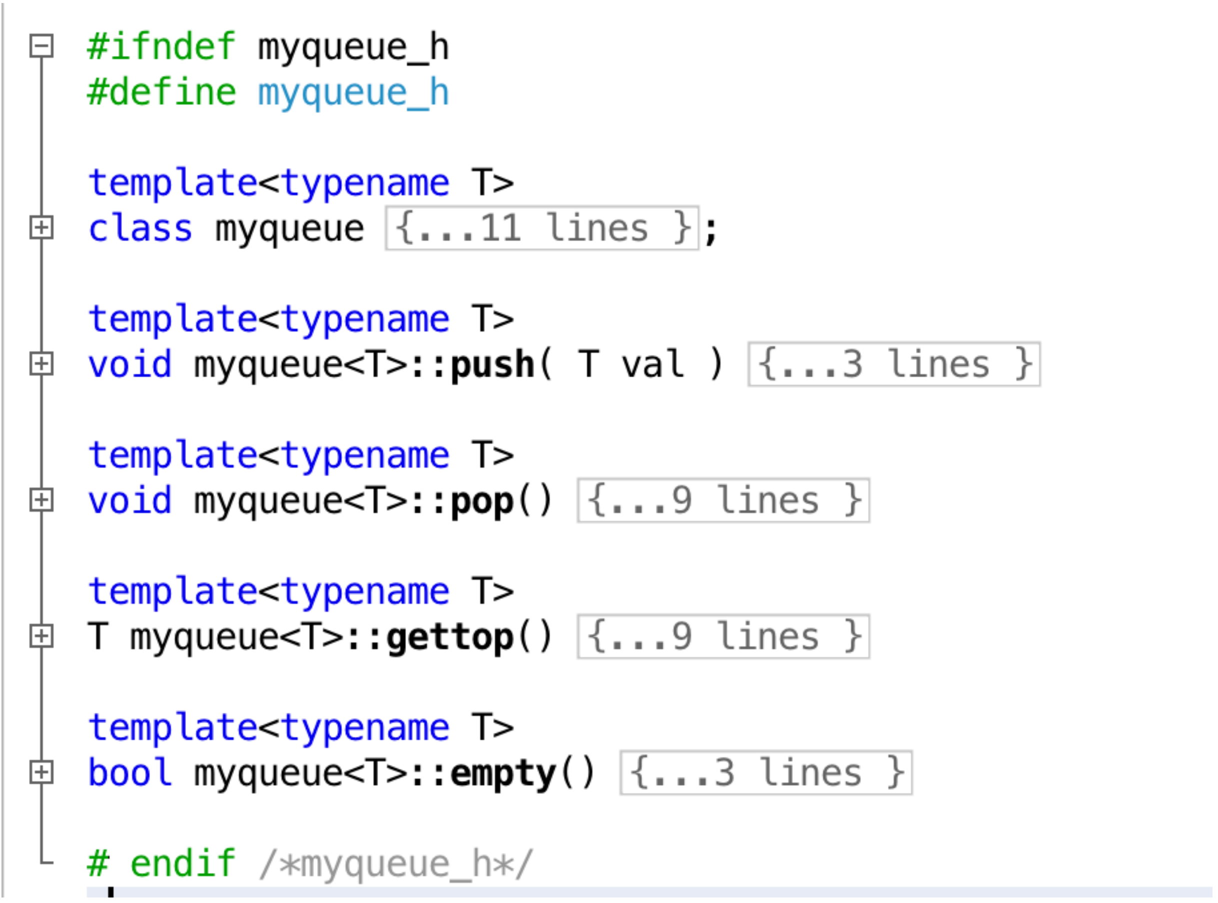
```
#ifndef myminstack_h
#define myminstack_h

template<typename T>
class minstack {...11 lines };

template<typename T>
void minstack<T>::push( T val ) {...13 lines }

template<typename T>
void minstack<T>::pop() {...4 lines }

template<typename T>
T minstack<T>::gettop() {...3 lines }

template<typename T>
T minstack<T>::getMin() {...3 lines }

# endif /* my min stack*/
```

Part III:

Using **mystack.h** class you will now implement your **Queue** class. The Queue class must be implemented as a template. For the implementation of the Queue class there is no requirement of implementing the methods in O(1) time. Your methods can grow linearly. There will be no point deduction for that. You will name your Queue class as myqueue class. You will be responsible of implementing the following methods:

1. **push**: This method adds a new element to the top of the queue.
2. **pop**:This method deletes the top element of the queue.
3. **empty**: This method tests to see if the queue is empty.
4. **gettop**: This method returns the top element of the queue.
5. **It is a requirement that you add pre and post conditions to each method listed above, see lecture notes for guidance**.

See screen shot below for guidance:



```
#ifndef myqueue_h
#define myqueue_h

template<typename T>
class myqueue {...11 lines };

template<typename T>
void myqueue<T>::push( T val ) {...3 lines }

template<typename T>
void myqueue<T>::pop() {...9 lines }

template<typename T>
T myqueue<T>::gettop() {...9 lines }

template<typename T>
bool myqueue<T>::empty() {...3 lines }

# endif /*myqueue_h*/
```

File Requirements

You are given the following files: makefile, main.cpp and mystack.h, myminstack.h, myqueue.h and 2 input files (for testing). Please do not use different names for naming your classes, testing will fail on our end.

- **main**.cpp: This file is for you to test your code. Make sure your implementations work with the main file provided, please name the classes and methods same as described above.
- **mystack**.h: You will implement this stack class as a header file. Name your class as **mystack.h**
- **myminstack**.h: You will implement the min stack class as a header file. Name your class as **myminstack.h**
- myqueue.h: You will implement the queue class as a header file. Name your class as **myqueue.h**

Project Deliverables

The following files with exactly the **same name** listed below must be submitted via Handin no later than **Tuesday October 25th 11:59pm**.

1. myminstack.h
2. mystack.h
3. myqueue.h

To test your code on black:

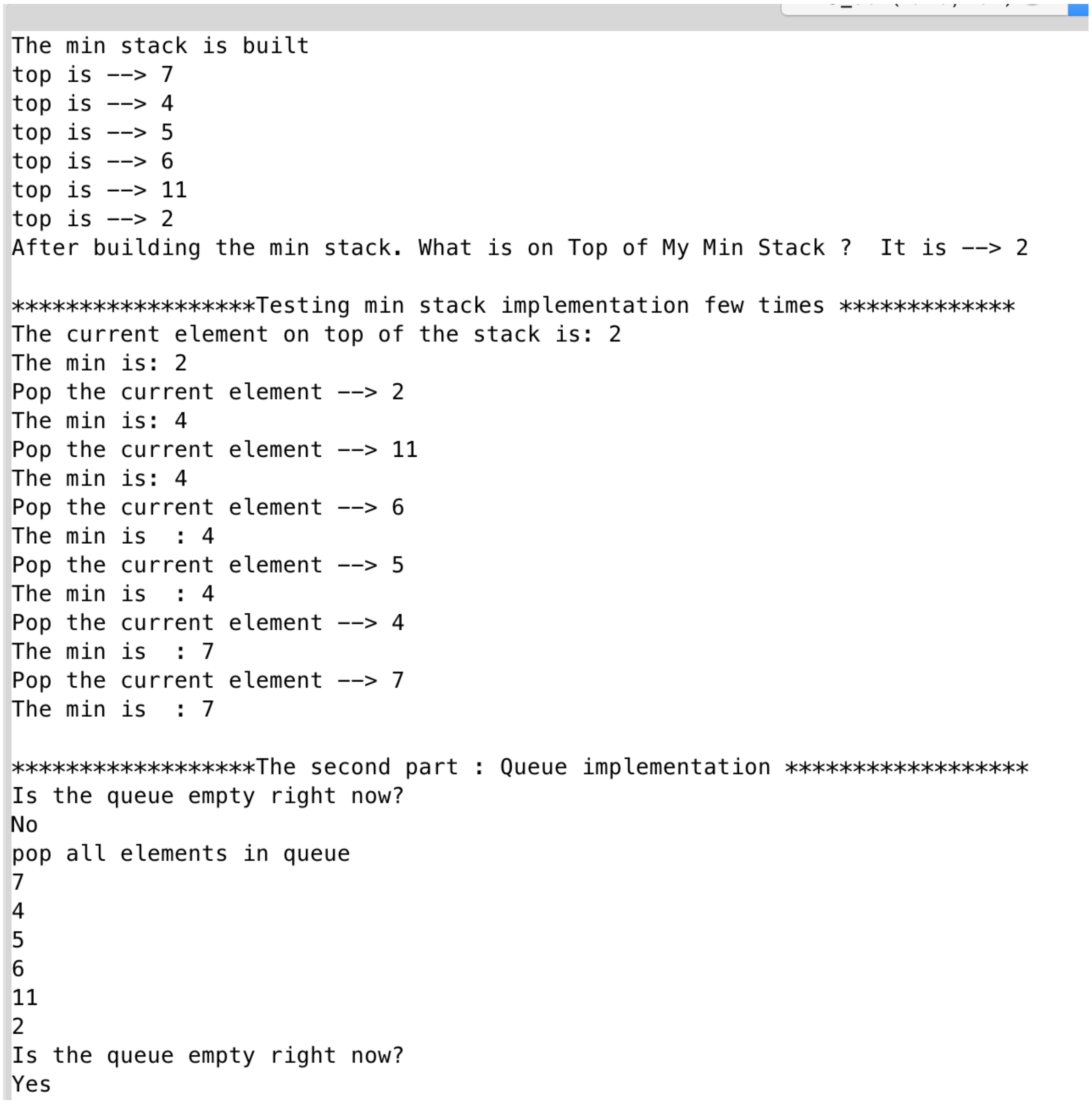
Follow the following steps:

1. Navigate to your folder where you put the main.cpp, makefile and mystack.h, myminstack.h, myqueue.h , inputfile, inputfile2.
2. Type "make" in the terminal.
3. Type "./run inputfile" and it should output as following with the given inputfile:

Sample Run(s)

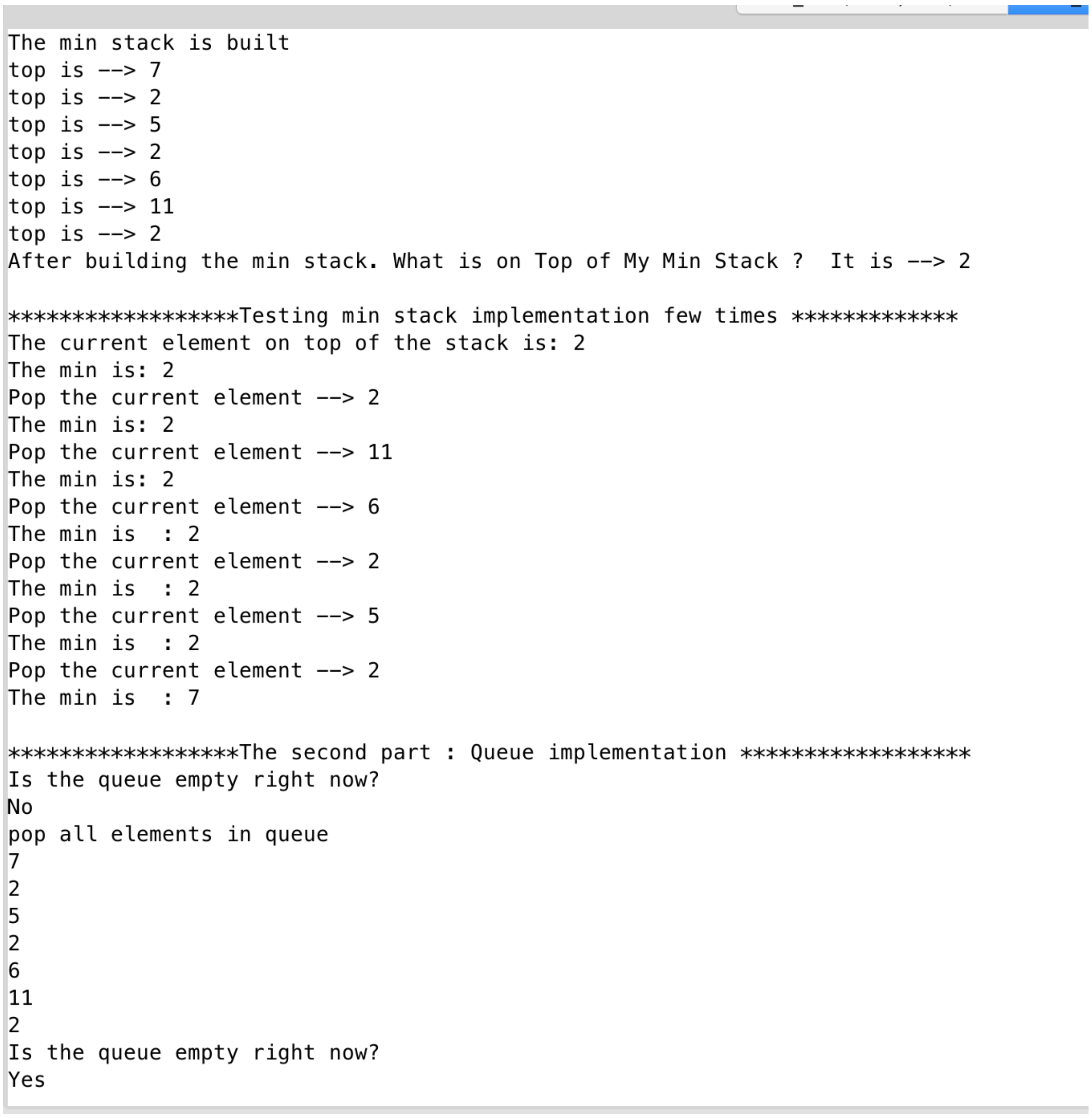
2 sample runs to help understand how the min stack works:

First run inputfile



```
The min stack is built
top is --> 7
top is --> 4
top is --> 5
top is --> 6
top is --> 11
top is --> 2
After building the min stack. What is on Top of My Min Stack ?  It is --> 2

******************Testing min stack implementation few times *************
The current element on top of the stack is: 2
The min is: 2
Pop the current element --> 2
The min is: 4
Pop the current element --> 11
The min is: 4
Pop the current element --> 6
The min is  : 4
Pop the current element --> 5
The min is  : 4
Pop the current element --> 4
The min is  : 7
Pop the current element --> 7
The min is  : 7

******************The second part : Queue implementation ******************
Is the queue empty right now?
No
pop all elements in queue
7
4
5
6
11
2
Is the queue empty right now?
Yes
```

Second run with inputfile2



```
The min stack is built
top is --> 7
top is --> 2
top is --> 5
top is --> 2
top is --> 6
top is --> 11
top is --> 2
After building the min stack. What is on Top of My Min Stack ?  It is --> 2

******************Testing min stack implementation few times *************
The current element on top of the stack is: 2
The min is: 2
Pop the current element --> 2
The min is: 2
Pop the current element --> 11
The min is: 2
Pop the current element --> 6
The min is  : 2
Pop the current element --> 2
The min is  : 2
Pop the current element --> 5
The min is  : 2
Pop the current element --> 2
The min is  : 7

******************The second part : Queue implementation ******************
Is the queue empty right now?
No
pop all elements in queue
7
2
5
2
6
11
2
Is the queue empty right now?
Yes
```