Homework 2

100 Points

STACK ADT

Project: One of the stack applications is to backtrack. As an example take a list of numbers. Each time we read 0, display the number of elements in the stack, each time we read 1, display the element at the top of the stack, each time we read a positive number we push it onto the stack, and each time we read a negative number, we pop and print as many values from the stack as the absolute value of the negative number. If there are fewer numbers in the stack, print a message and nothing else. When the end of the file is detected, print a message and the items left in the stack. See two examples on the next page.

In main() define and initialize an array of strings containing the names of the data files to be used for testing the program: numbers1.txt, numbers2.txt, numbers3.txt followed by QUIT, to terminate testing. Write a loop to process these files using a stack, as described above.

Run the program once and save the output at the end of the source file as a comment. Compress the source and header files, input and output files (if any), and upload the compressed file: 22C_LastName_FirstName_H2.zip

See the following examples in Stack Demo: 1.Build and process a stack of integers, 2.Build and process a stack of strings (templates), and 3.The Eight Queens Problem.

The stack library is incomplete: you have to implement the **`getTop`** function that passes back the data at the top of the stack, without changing the stack, and the **`getCount`** function that returns the number of elements in the stack.

> **Note:** *Class templates are used to create generic classes and abstract types. They enable you to create one general version of a class without replicating code to handle multiple data types. Where to start when defining template? It is easier to convert code that works into templates rather then to write templates from scratch. For instance, add the two required functions to the stack of integers project, without using template, debug and test it, then update the stack templates.*

Grading:

Creating and processing the array of strings	- 10
Read numbers from file and process stack	- 50
Implement the **`getTop`** function	- 10
Implement the **`getCount`** function	- 10
The main() function	- 20

Example 1:

```
10 20 30 1 40 0 50 -2 15 25 -3 60 70
```

Read **10**, Push **10**
Read **20**, Push **20**
Read **30**, Push **30**
Read **1**, Display the element at the top of the stack: **30**
Read **40**, Push **40**
Read **0**, Display the number of elements in the stack: the stack has 4 numbers
Read **50**, Push **50**
Read **-2**, Pop and print **2** numbers: **50** and **40**
Read **15**, Push **15**
Read **25**, Push **25**
Read **-3**, Pop and print **3** numbers: **25**, **15** and **30**
Read **60**, Push **60**
Read **70**, Push **70**
END OF FILE, Display the stack from top to bottom: **70**, **60**, **20** and **10**

Example 2:

```
10 20 30 -6 40 50 -5
```

Read **10**, Push **10**
Read **20**, Push **20**
Read **30**, Push **30**
Read **-6**, Display an error message such as:
"The stack has less than 6 items" and nothing else!
Read **40**, Push **40**
Read **50**, Push **50**
Read **-5**: Pop and print **5** numbers **50**, **40**, **30**, **20** and **10**
END OF FILE, Display a message such as "Empty stack!"

Create the following 3 input files:
numbers1.txt
10 20 30 1 40 0 50 -2 15 25 -3 60 80

numbers2.txt
10 20 30 -6 40 50 -5

numbers3.txt
15 25 35 45 0 1 0 55 65 75 0 1 0 -50 0 1 0 -5 10 40 50 60 70 80 90 0 1 0 -3 13 23 33 43 54 -5 0 1 0 17 27 7
47 57 67 77 87 97 19 29 39 49 59 69 79 89 99 0 1 -6 0 1 14 24 34 44 0 -150 54 64 0