John Carrabino
carrabij@oregonstate.edu
August 12th, 2016

Lab Fb

TESTING THE STACK FILES:

In order to test the Stack files I received from my TA I modified the main file from my lab Fa submission and ran my tests from that lab in order to demonstrate the last in first out(LIFO) behavior of the stack class. This means that when an item is popped and displayed from the stack, it should be the most recent item entered.

Test Case	Input	Test Function	Expected Outcomes	Observed Outcomes
Add positive and negative numbers to the stack	What number(s) would you like to add? 11 -22 33	switch(choice){ case 1: Add an integer to the stack	Stack: 33 -22 11	Stack: 33 -22 11
Adding non-numeric, non-alphabetic characters to the stack	What number(s) would you like to add? 1+1 -2 2 3-3	switch(choice){ case 1: Add an integer to the stack	Stack: -3 3 2 -2 1 1	Stack: -3 3 2 -2 1 1
Remove an item from the stack **test input : "11 22 33"**	Choose one of the following options: 2 - Remove and display a number from the Stack	switch(choice){ case 2: Display and remove an integer form the stack	Stack: 33	Stack: 33
Remove all items from the stack **test input : "11 22 33"**	Choose one of the following options: 3 - Remove and Display all numbers in the Stack	switch(choice){ case 3:Display and remove all integers from the stack	Stack: 33 22 11	Stack: 33 22 11
Remove (N+1)th Item from list (demonstrated using empty list)	Choose one of the following options: 2 - Remove and display a number from the Stack	switch(choice){ case 2: Display and remove an integer from an empty stack	"The stack is empty"	"The stack is empty"

After I conducted my tests on the new Stack files I began to develop my Reverse Polish Notation (RPN) Calculator class. After brainstorming how to implement this class using the new Stack class files I realized there were a few things in the Stack class that I needed to change. First off, the StackNode class was declared as a private member of the Stack class, which I had to change to protected in order to access its data member from the RPN class. I did this because I wanted the RPN class to become a derived class of the new Stack class so that I could use all of the functions defined in both classes using polymorphism. I also added a virtual destructor in my Stack class in order to make it an abstract base class, so that I can assign a Stack pointer to an RPN object and work with that in my main function. Other than the changes named abve, all other functions in the Stack class remain unchanged.

When I was done drafting the operator functions for the RPN class I went online to look at examples of how people implemented RPN calculators and found that is I used an istringstream object in order to accept input and convert it to string or integer variables as needed. After that I finally finished my main function which prompts the user for an RPN expression and has them input an expression to be evaluated. As a couple of safeguards, I have my code report if there are too few operands or if a division error occurs. Here is a chart of some of the test I performed to make sure all of my RPN class functions worked.

*Note: In order to come up with good test equations I used the Infix/Postfix converter from the following website, which allowed me to convert expressions from infix to post fix and vice versa; http://www.mathblog.dk/tools/infix-postfix-converter/

Test Case	Input (input = postfix notation)	Test Function	Expected Outcomes	Observed Outcomes
Test the addition operator	infix: 1+2 + (3 + 4) + 5 postfix: 1 2 + 3 4 + + 5 +	addition();	Solution = 15	Expression: 1 2 + 3 4 + + 5 + q Solution: 15
Test the subtraction operator	infix: 1 - 2 - (3 - 4) - 5 postfix: 1 2 - 3 4 - - 5 -	subtraction();	Solution = -5	Expression: 1 2 - 3 4 - - 5 - q Solution: -5
Test the multiplication operator	infix: (10 * 2) * 3 postfix: 10 2 * 3 *	multiplication();	Solution = 60	Expression: 10 2 * 3 * q Solution: 60
Test the division operator	infix: (100 / 20) / 5 postfix: 100 20 / 5 /	division();	Solution = 1	Expression: 100 20 / 5 / q Solution: 1

Test all operators in a single expression.	infix: (100 / 5) * 2 + (7 - 6)
	
	postfix: 100 5 / 2 * 7 6 - +
		addition();
		subtraction();
		RPN::multiplication();
		division();
			Solution = 41	Expression: 100 5 / 2 * 7 6 - + q
				
				Solution: 41

Test division by 0	infix: postfix:	division();	An error will inform the user they have attempted to divide by zero, it will then return the current contents of the stack	Expression: 7 6 * 0 / q ERROR: Cannot divide by 0 The … contents of your current Stack are listed below. Stack: 0 42
Test invalid input	Input: 1 2 + 1000000000000000 50 2 – 10 – 2 * <ENTER>	RPN::wasError();	An error message will prompt the user that any input after an invalid entry will not be evaluated. 1 2 + is a valid expression so It would return, solution = 3	Please note that if you enter a number with more than ten digits then the program will terminate and evaluate your expression based on all input recieved prior to the bad input Expression: 1 2 + 100000000000000 50 2 - 10 2 * q Solution: 3
Test only empty input	Input: x <ENTER>	RPN::isEmpty();	No valid integers were entered, so no calculations could be performed.	Expression: FREEDOM☺!!! No valid integers were entered, so no calculations could be performed.