**CSE 212 – Programming with Data Structures**

**W03 Prove – Response Document**

|  |  |
| --- | --- |
| **Name:** | Kelsie Garcia |
| **Date:** | 10/3/2025 |
| **Teacher:** | Brother Alvey |

*It is a violation of BYU-Idaho Honor Code to post or share this document with others or to post it online. Storage into a personal and private repository (e.g. private GitHub repository, unshared Google Drive folder) is acceptable.*

**Question 1: From Part 1, describe what the Mystery Stack 1 code does and how the use of a stack helps in the implementation.**

This stack and method takes an input string, char by char with it’s foreach. Each char is pushed onto the stack and the letters are pushed into place with the order they are inputted by the string. The last letter is at the top of the stack.  
  
The while loop continues while the stack is not empty. Through each iteration, the top latter of the stack is popped using stack.Pop(). Because of the LIFO nature the chars are retrieved in the reverse order of how they were pushed. Then the result is returned with the string of chars of the taken input string now in reverse order.

This is a natural and easy way to implement a string reversal with LIFO and the data structure of a stack. No need for any temporary data structures like arrays. LIFO achieves the reversal order.

**Question 2: From Part 1, what are the three outputs from the Mystery Stack 1 code for the following three different inputs?**

* **racecar**
  + racecar
* **stressed**
  + desserts
* **a nut for a jar of tuna**
  + anut of raj a rof tun a

**Question 3: From Part 2, describe what the Mystery Stack 2 code does and how the use of a stack helps in the implementation.**

First there is has a Boolean method that checks if the input is a number that should be pushed into the stack and checks for a valid float. The new stack will hold the numbers and operands during the calculations. The foreach will process the input and it will split the string where it finds a space char, leaving letter/operators and will process them one at a time.

The if conditions will check for four operands and if the stack count is less than two then through an error with a string explaining why.

Then it creates two variables that it will pop off the stack. It creates float variable for the result.

Next, is an if that checks for which operand and tells it how to calculate depending on the operand setting the result variable after the calculation. If the op2 variable is 0 then it throws an invalid op and it also does the division as part of the else statement.

Then it needs to push the value of result onto the stack passing the res variable into it.

Back in the first if statement, this else if processes whether it a float item and pushes the item and parses it for float into the stack. Another else if, to ignore extra spaces, and else to throw another error if it is not anything defined in the conditions above.

An if statement check that in the count does not equal 1 to throw an error saying invalid case 4. Then finally return the single result by popping it from the stack.

The use of the stack helps implement storing the numbers and reading left to right. If an operant is processed, then the operator is used on the two most recently inputted numbers. The result is then pushed back on the stack, and it replaces the two number processed. It’ll process all the numbers until a single result is left, and it will return.

**Question 4: From Part 2, answer the following regarding what the Mystery Stack 2 code does:**

* **What will the result be if the input parameter is: 5 3 7 + \***
  + **50.0**
* **What will the result be if the input parameter is: 6 2 + 5 3 - /**
  + **4.0**
* **What input would result in the display of “Invalid Case 1!”**
  + **3 \***
* **What input would result in the display of “Invalid Case 2!”**
  + **3 0 /**
* **What input would result in the display of “Invalid Case 3!”**
  + **>**
* **What input would result in the display of “Invalid Case 4!”**
  + **1 3**