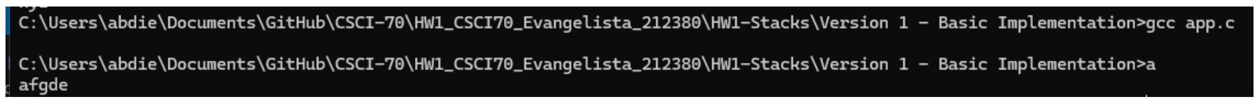
Abdiel Evangelista
CSCI 70-B

Version 1 - Basic Implementation:

1. Instead of printing the recently popped character z, the print statement is called on the array of characters, Store instead. In C, when a print statement is called on an array of characters, what is printed is a string of characters where the elements of the character array are appended to each other with the first elements to be added to the array being added first, thus the string "xyz" becomes the output of the code.

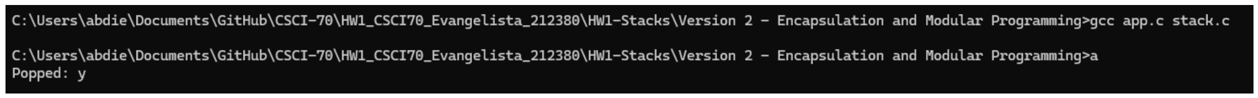
2. 

```
C:\Users\abdie\Documents\GitHub\CSCI-70\HW1_CSCI70_Evangelista_212380\HW1-Stacks\Version 1 - Basic Implementation>gcc app.c

C:\Users\abdie\Documents\GitHub\CSCI-70\HW1_CSCI70_Evangelista_212380\HW1-Stacks\Version 1 - Basic Implementation>a
afgde
```

First, the characters 'a', 'b', 'c', 'd' and 'e' are pushed to the stack. The Store array is now { 'a', 'b', 'c', 'd', 'e' }. Then, pop is called four times. However, the implementation of the pop function in the code only returns the most recently pushed character in the stack and decrements the top counter by 1. It doesn't actually remove any elements from the stack. Therefore, the store array is still { 'a', 'b', 'c', 'd', 'e' } and the top counter is now at 1, the index pointing at element 'b' in the stack. Then, characters 'f' and 'g' are pushed to the stack. With the implementation of the push function in the code, character 'f' replaces 'b' and character 'g' replaces 'c'. Now, the Store array is { 'a', 'f', 'g', 'd', 'e' } and the print statement is called on it, outputting 'afgde'.

Version 2 - Encapsulation and Modular Programming

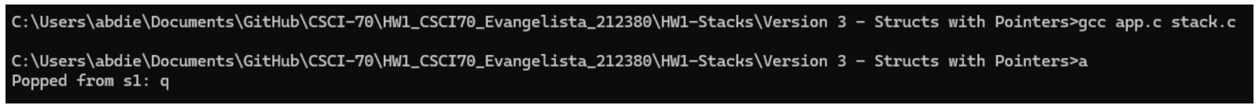
1. 

```
C:\Users\abdie\Documents\GitHub\CSCI-70\HW1_CSCI70_Evangelista_212380\HW1-Stacks\Version 2 - Encapsulation and Modular Programming>gcc app.c stack.c

C:\Users\abdie\Documents\GitHub\CSCI-70\HW1_CSCI70_Evangelista_212380\HW1-Stacks\Version 2 - Encapsulation and Modular Programming>a
Popped: y
```

First, the static is removed from Store and top, making them accessible from other files. Then, the top variable is also declared with extern from app.c, making the same top variable as the one on stack.c, accessible by app.c. Before the pop function is called, the top counter is set to 2, making it point to element 'y' instead of 'z', which is supposedly the most recently pushed element in the stack. When the pop function is called, 'y' is outputted instead of 'z'.

Version 3 - Structs with Pointers

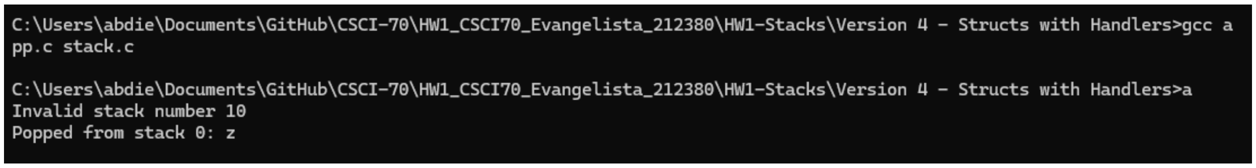
1. 

```
C:\Users\abdie\Documents\GitHub\CSCI-70\HW1_CSCI70_Evangelista_212380\HW1-Stacks\Version 3 - Structs with Pointers>gcc app.c stack.c

C:\Users\abdie\Documents\GitHub\CSCI-70\HW1_CSCI70_Evangelista_212380\HW1-Stacks\Version 3 - Structs with Pointers>a
Popped from s1: q
```

A weakness of this approach is that app.c can easily access the stack since this array is not protected in any way. As demonstrated in this output, app.c can easily change any of the created struct's stack's elements to anything. In this case, element 'z' in struct 1's stack was changed to element 'q' in the application() function, which is why 'q' is popped instead of 'z'.

Version 4 - Structs with Handlers

1. 

```
C:\Users\abdie\Documents\GitHub\CSCI-70\HW1_CSCI70_Evangelista_212380\HW1-Stacks\Version 4 - Structs with Handlers>gcc a
pp.c stack.c

C:\Users\abdie\Documents\GitHub\CSCI-70\HW1_CSCI70_Evangelista_212380\HW1-Stacks\Version 4 - Structs with Handlers>a
Invalid stack number 10
Popped from stack 0: z
```

   A weakness of this approach is that there is only a limited number of structs that can be made depending on the maximum number that is set. This weakness can go both ways in that either too few structs might be instantiated for the purpose of the user or too many structs are made even when the user only needs a few, compromising the performance and exerting unnecessary effort on the machine. In this case, since there is only a maximum of 10 structs, an 11th one wouldn't be possible even if the user needs this 11th struct. The number of structs that are instantiated at the start are also equal to the maximum so some structs already exist even though they aren't needed yet.

Version 5 - Object Oriented Programming

1. Reason 1 - A programmer needs only one stack. In this case, Version 2 of the program will suffice since the stack will be protected and won't be modifiable by the application apart from the push and pop functions. In using this program when the programmer's purpose only needs one stack, there won't be any confusion because there will only be one stack no matter what unlike with the OOP approach where multiple instances of the stack class can be created to make a stack.
2. Reason 2 - A programmer knows exactly how many stacks he wants to use. In this case, Version 4 of the program will suffice since the stack will also already be protected by this approach and he can just set the maximum number of stacks to the exact number he needs. The advantage this approach also has over the OOP approach is that the programmer wouldn't need to instantiate the stacks in the application function anymore and can instead just focus on the push, pop and prints.