

Activity No. 4	
Stacks	
Course Code: CPE010	Program: Computer Engineering
Course Title: Data Structures and Algorithms	Date Performed: 04 - 10 - 2024
Section: CpE21 S4	Date Submitted: 04 - 10 - 2024
Name(s):Bonifacio, Nyko Adrein	Instructor:Prof. Sayo

6. Output

<div><div>main.cpp</div><div><div><div></div><div></div><div></div></div><div><div>Share</div><div>Run</div></div></div><pre>1 #include <iostream> 2 #include <stack> // Calling Stack from the STL 3 using namespace std; 4 int main() { 5 stack<int> newStack; 6 newStack.push(3); //Adds 3 to the stack 7 newStack.push(8); 8 newStack.push(15); 9 // returns a boolean response depending on if the stack is empty or not 10 cout << "Stack Empty? " << newStack.empty() << endl; 11 // returns the size of the stack itself 12 cout << "Stack Size: " << newStack.size() << endl; 13 // returns the topmost element of the stack 14 cout << "Top Element of the Stack: " << newStack.top() << endl; 15 // removes the topmost element of the stack 16 newStack.pop(); 17 cout << "Top Element of the Stack: " << newStack.top() << endl; 18 cout << "Stack Size: " << newStack.size() << endl; 19 return 0; 20 } 21</pre></div>	<div><div>Output</div><div><div>Clear</div></div><pre>/tmp/ZSbGJTu9p5.o Stack Empty? 0 Stack Size: 3 Top Element of the Stack: 15 Top Element of the Stack: 8 Stack Size: 2 --- Code Execution Successful ---</pre></div>
--	--

Table 4-1. Output of ILO A

Output

Clear

```
/tmp/2F4uQYTkqp.o
```

```
Enter number of max elements for new stack: 2
```

```
Stack Operations:
```

```
1. PUSH, 2. POP, 3. TOP, 4. isEmpty, 5. DISPLAY
```

```
2
```

```
Stack Underflow.
```

```
Stack Operations:
```

```
1. PUSH, 2. POP, 3. TOP, 4. isEmpty, 5. DISPLAY
```

```
1
```

```
New Value:
```

```
2
```

```
Stack Operations:
```

```
1. PUSH, 2. POP, 3. TOP, 4. isEmpty, 5. DISPLAY
```

```
1
```

```
New Value:
```

```
3
```

```
Stack Operations:
```

```
1. PUSH, 2. POP, 3. TOP, 4. isEmpty, 5. DISPLAY
```

```
4
```

```
Stack is not empty.
```

```
Stack Operations:
```

```
1. PUSH, 2. POP, 3. TOP, 4. isEmpty, 5. DISPLAY
```

```
3
```

```
The element on the top of the stack is 3
```

```
Stack Operations:
```

```
1. PUSH, 2. POP, 3. TOP, 4. isEmpty, 5. DISPLAY
```

```
5
```

```
Stack elements are: 3 2
```

```
Stack Operations:
```

```
1. PUSH, 2. POP, 3. TOP, 4. isEmpty, 5. DISPLAY
```

```
|
```

Table 4-2. Output of ILO B.1

<div> <div>Output</div> <div>Clear</div> </div> <pre> /tmp/fYor08m5L7.o After the first PUSH, top of stack is: Top of Stack: 1 Stack elements: 1 After the second PUSH, top of stack is: Top of Stack: 5 Stack elements: 5 1 After the first POP operation, top of stack is: Top of Stack: 1 Stack elements: 1 After the second POP operation, top of stack is: Stack is Empty. Stack is Empty. Stack Underflow. === Code Execution Successful === </pre>
--

Table 4-3. Output of ILO B.2

7. Supplementary Activity			
EXPRESSION	VALID (Y/N?)	OUTPUT (CONSOLE SCREEN SHOT)	ANALYSIS
(A+B)+(C-D)	Y	<div> <div>Output</div> <div> <pre> /tmp/0p949UyHqG.o Enter expression: (A+B)+(C-D) The expression is balanced. === Code Execution Successful === </pre> </div> </div>	Parentheses are properly balanced and in the correct order.
((A+B)+(C-D)	N	<div> <div>Output</div> <div> <pre> /tmp/bdWxDHQynK.o Enter expression: ((A+B)+(C-D) The expression is not balanced. === Code Execution Successful === </pre> </div> </div>	Nested parentheses are incorrectly matched and unbalanced.

((A+B)+[C-D])	Y	<pre> Output /tmp/uJRYGBvxJn.o Enter expression: ((A+B)+[C-D]) The expression is balanced. === Code Execution Successful === </pre>	Parentheses and square brackets are correctly matched and balanced.
((A+B)+[C-D])}	N	<pre> Output /tmp/AtBH4jZAwE.o Enter expression: ((A+B)+[C-D])} The expression is not balanced. === Code Execution Successful === </pre>	Mismatched parentheses and brackets make the expression invalid.

8. Conclusion

In this activity, I learned how to implement stacks using arrays, linked lists, and the C++ STL to check for balanced symbols in expressions. The array-based stack is simple but limited by fixed memory, while the linked list provides dynamic allocation, though it's more complex to manage. The STL stack simplifies the implementation by handling memory automatically. Testing different expressions helped solidify my understanding of handling nested symbols and edge cases. I performed well in applying these concepts, but I can improve by optimizing code, handling more complex edge cases, and providing better error messages.

9. Assessment Rubric