Activity No. 4	
Stacks	
Course Code: CPE010	**Program:** Computer Engineering
Course Title: Data Structures and Algorithms	**Date Performed:** October 4, 2024
Section: CPE21S4	**Date Submitted:** October 6, 2024
Name(s): Bona, Andrei Nycole So	**Instructor:** Prof. Maria Rizette Sayo

6. Output

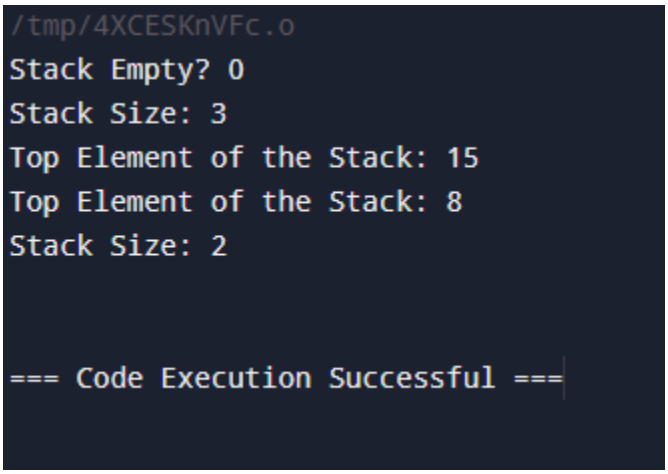

Table 4-1. Output of ILO A

```
/tmp/wcwvUvTtZL.o
Enter number of max elements for new stack: 3
Stack Operations:
1. PUSH, 2. POP, 3. TOP, 4. isEMPTY, 5. Display Stack
1
New Value:
10
Stack Operations:
1. PUSH, 2. POP, 3. TOP, 4. isEMPTY, 5. Display Stack
1
New Value:
20
Stack Operations:
1. PUSH, 2. POP, 3. TOP, 4. isEMPTY, 5. Display Stack
1
New Value:
30
Stack Operations:
1. PUSH, 2. POP, 3. TOP, 4. isEMPTY, 5. Display Stack
2
Popping: 30
Stack Operations:
1. PUSH, 2. POP, 3. TOP, 4. isEMPTY, 5. Display Stack
3
The element on the top of the stack is 20
Stack Operations:
1. PUSH, 2. POP, 3. TOP, 4. isEMPTY, 5. Display Stack
4
0
Stack Operations:
1. PUSH, 2. POP, 3. TOP, 4. isEMPTY, 5. Display Stack
5
Elements inside the stack:
20
10
```

```
void display()
{
    if(!isEmpty())
    {
        std::cout<<"Elements inside the stack:"<<std::endl;
        while (!isEmpty())
        {
            std::cout<<stack[top]<<std::endl;
            top--;
        }
    }
    else
    {
        std::cout<<"There is no element inside the stack"<<std::endl;
    }

}
```

- empty - Test whether container is empty (public member function)
- size - Return size (public member function)
- top - Access next element (public member function)
- push - Insert element (public member function)
- pop - Remove top element (public member function)

created function

- display - It will display the elements inside the stack

Table 4-2. Output of ILO B.1.

```
/tmp/QvmcXaVp2N.o
After the first PUSH top of stack is : Top of Stack: 1
After the second PUSH top of stack is : Top of Stack: 5

After the first POP operation, top of stack is: Top of Stack: 1
After the second POP operation, top of stack : Stack is Empty.

Stack Underflow.

Stack is Empty.

After the first POP operation, top of stack is: Top of Stack: 24
After the second POP operation, top of stack is: Top of Stack: 29

Elements inside the stack:
29
24


=== Code Execution Successful ===
```

- empty - Test whether container is empty (public member function)
- size - Return size (public member function)
- top - Access next element (public member function)
- push - Insert element (public member function)
- pop - Remove top element (public member function)

created function

- display - It will display the elements inside the stack

Table 4-3. Output of ILO B.2.

7. Supplementary Activity

Self-Checking:
For the following cases, complete the table using the code you created.

Expression Valid?	(Y/N)	Output (Console Screenshot)	Analysis
(A+B)+(C-D)	Y	Expression is balanced	All opening symbols match with corresponding closing ones.
((A+B)+(C-D)	N	Expression is not balanced	The stack has an extra opening parenthesis.
((A+B)+[C-D])	Y	Expression is balanced	All symbols are properly nested and balanced.
((A+B]+[C-D]}	N	Expression is not balanced	Mismatch between brackets and braces.

Tools Analysis

- Stack Implemented Using Arrays: The array implementation is simpler, tiny problems and fixed-size are efficient. Problem is it has a limit of MAX which makes it vulnerable to stack overflow. It is suitable when we know the maximum number of elements.

- Stack Implemented Using Linked Lists: In this implementation, we can create as many nodes elements without knowing the exact number of elements. Nonetheless, the overhead in terms of dynamic memory allocation and deallocation could impact performance (especially for large-scale problems).

- Stack Using the Standard Template Library: The Stack container is efficient and easy to use with built-in functionality that we can use directly. It's also dynamically sized, offering the advantages of a linked list heirachy without any need for manual memory management. Mezzanine is the obvious choice for most situations outside of customization.

8. Conclusion

With this laboratory activity, I gained knowledge about stack, which essentially operates based on a Last In First Out (LIFO) principle. Here, I implemented both stacks using arrays and linked lists, focusing on the most important functions like push (for adding elements), pop (deletes an element), empty (checks whether it is an empty stack), top (to print the top of the stack without deleting it), and display (all elements). This also strengthened my knowledge of stack operations and the necessity debugging.

In the supplementary activity, the task was to check for balanced symbols. The process was first initializing a stack and then walking through the input expression. In case of any opening symbol, I used the push function, and in closing symbols, I used pop to match the pairs while ensuring that the stack is not empty with the empty function. If the stack was empty at the end, it meant that it contained well-balanced symbols and thus validated the use of stacks in handling nested structures. I learned to link theoretical concepts with practical skills as I implemented critical stack operations in this activity.

This laboratory allowed me to improve and do better in enhancing C++ programming. The display function came in very handy while trying to display the contents inside the stack and debug. However, I still acknowledge that my knowledge is still very basic because there are still methods and techniques of working with stack that is quite advanced which I have not mastered yet. I use different resources online to answer this laboratory and to improve my understanding and programming capability.

9. Assessment Rubric