# Flowcharts of Operation on Stacks

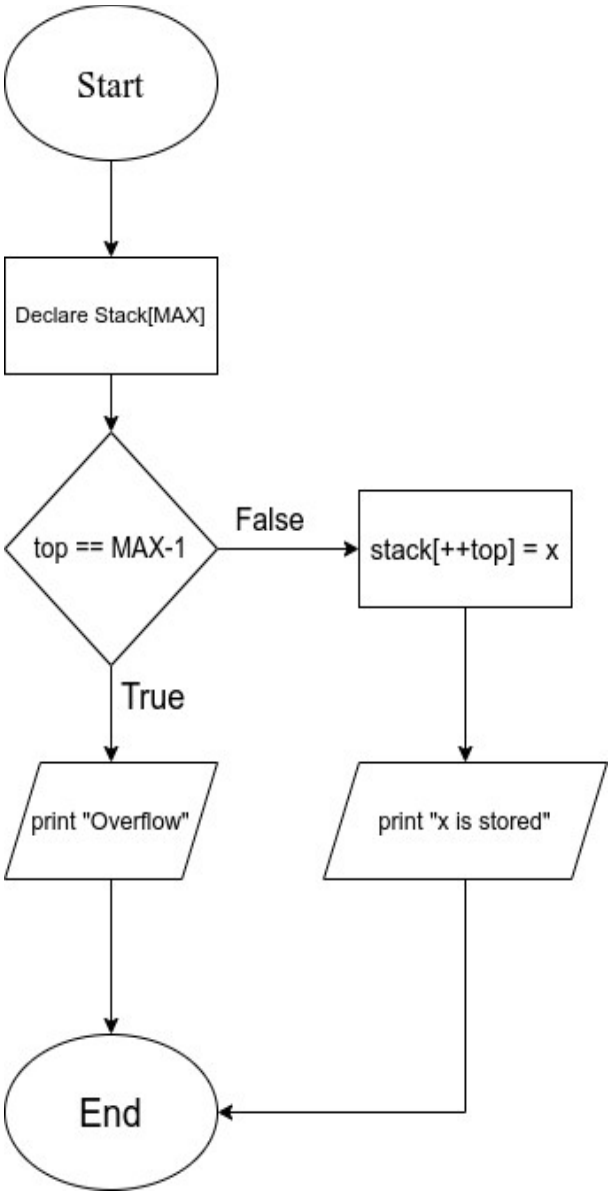


Fig: Push Operation

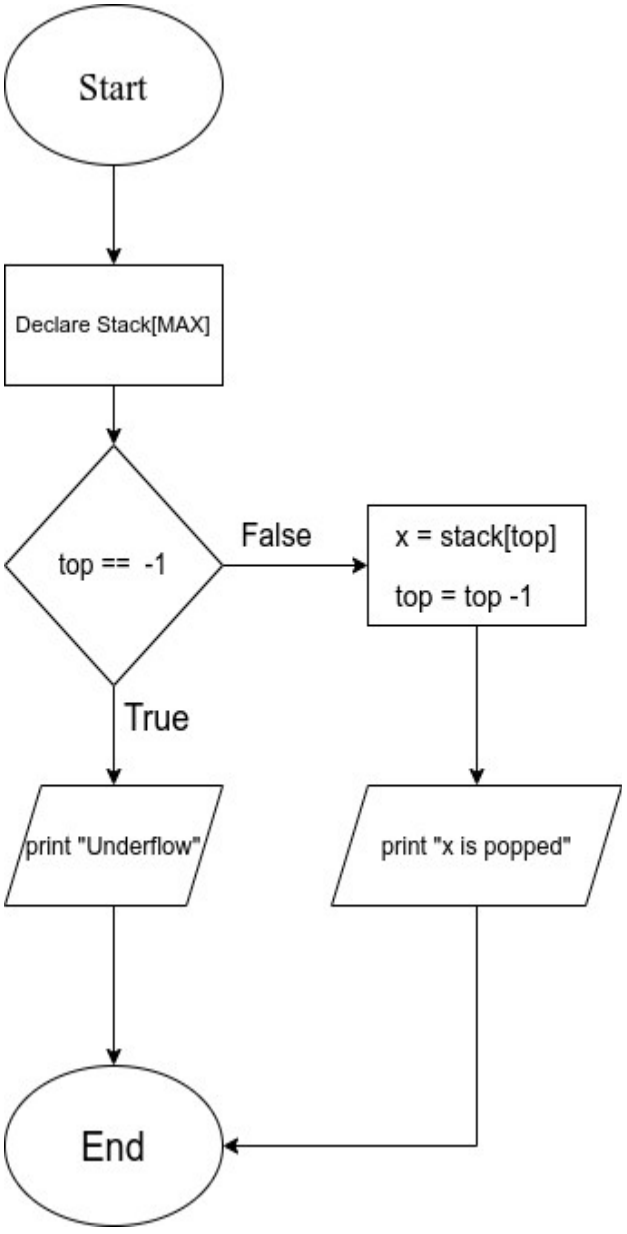


Fig: Pop Operation

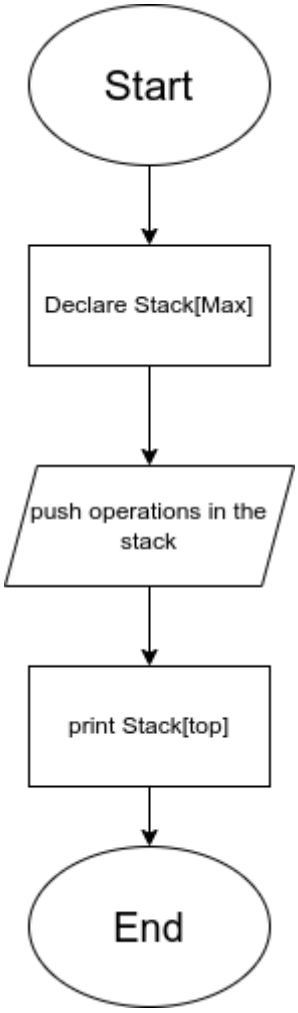


Fig:Peek Operation

Test Cases:

| Test Cases: | Input | Expected Result | Outcome |
|---|---|---|---|
| Case 1: Underflow condition | Choice=2,<br>top = -1,<br>pop() was invoked | There is no data in the stack. So, the stack should be empty or underflow | The stack was underflow or empty. (Pass) |
| Case 2: push operation | Top = 0,<br>Choice =1,<br>invokes the push(32) | "32" should be stored at stack[0]<br>&<br>top element will be "32" | 32 was added in the stack and it was the top element<br>(Pass) |
| The push(45) was called before the pop(). | | | |
| Case 3: pop operation | Choice = 2<br>top = 1<br>pop() was invoked | The top element in the stack was popped i.e. stack[top]=45 & top = top -1 | Top element was popped from the stack, i.e. 45 was popped &<br>top = top-1 |
| Again, push() function was used to push 76,333,756 | | | |
| Case 4: peek operation | Choice = 4<br>top = 3<br>peek() was invoked | The top element in the stack was printed out. | Top element in the stack was printed i.e 756 |
| Case 5: Overflow condition | Choice=1<br>top=3<br>push() was invoked | The overflow condition should trigger and stack should be full. | Overflow condition was triggered as stack is full. |

## Discussion:

In this lab session, we experimented with all the major stack operations and implemented it with C. The above test-cases shows the expected result and output of the experiments. We are able to visually represent the abstract nature of stack using the arrays in C.

## Conclusion:

In conclusion, we successfully understood and verified the operations and implementations of an abstract data type, specifically a Stack using an array.