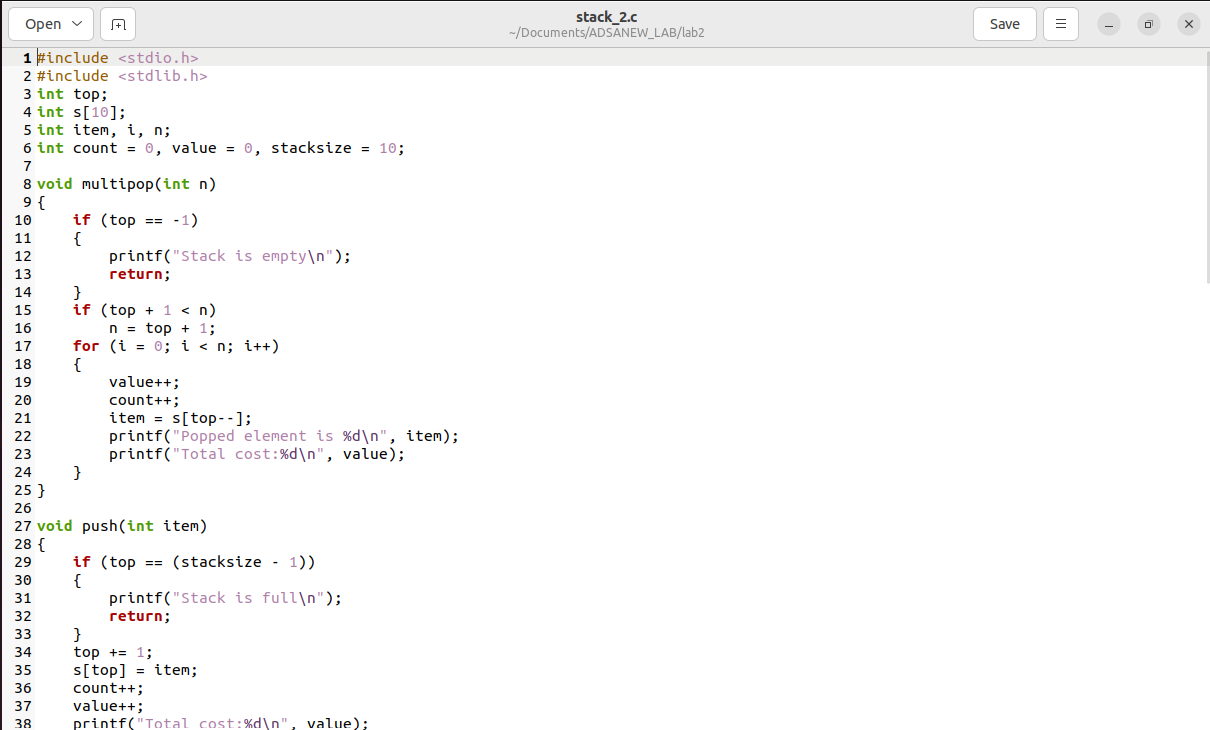
**Lab 2 :**

1. Write a program for implementing the following operations of stack S & also find the amortized cost if a sequence of n following operations are performed on a data structure.

(i) Push(S, x)

(ii) Pop(S)

(iii) Multipop(S,k)



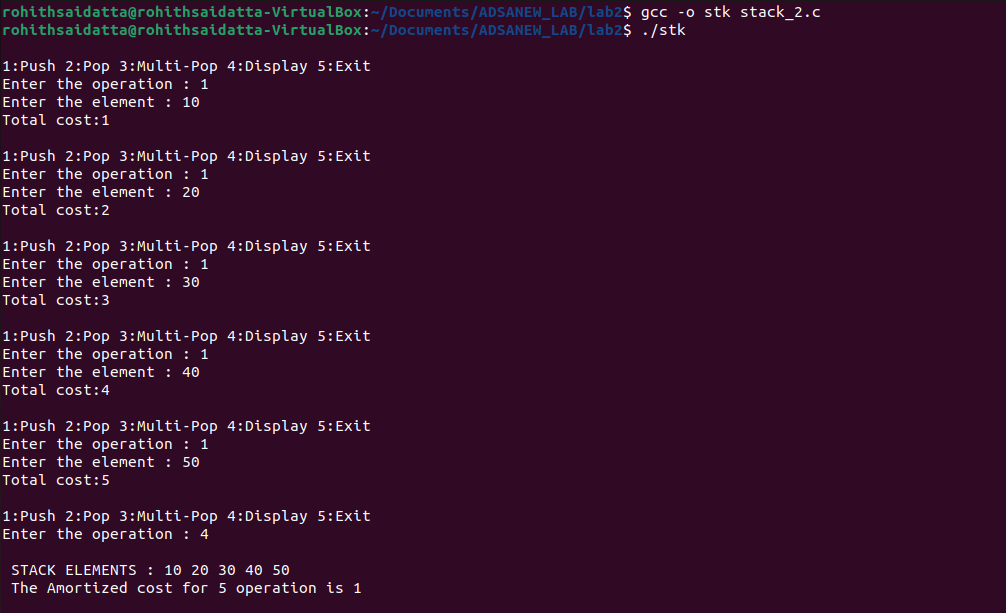
A computer screen shot of a white background

Description automatically generated

A screenshot of a computer program

Description automatically generated





A black and purple background

Description automatically generated with medium confidence

A black and purple background

Description automatically generated with medium confidence

Analysis

* Initially, added 5 elements (10, 20, 30, 40, 50) to the stack. This action increased the overall operation cost.
* Two elements (50 and 40) were then removed from the stack using pop operations. Each pop is added to the total operation cost.
* Later a multi-pop operation to remove 2 elements. Since only 3 elements remained, this cleared the stack, causing additional cost.
* Amortized cost per operation gives an average expense for each action. It's found by dividing the total cost by all operations performed.
* The actual amortized cost and stack content can differ based on how you carried out the steps.
* This analysis helps understand how costs change with various operations and their order, considering factors like stack size and sequence.

Time Complexity:

The time complexity of the stack operations in this code is generally constant (O(1)) for push and pop, as they involve single-element manipulation. Multi-pop and display operations take linear time (O(n)), where n is the number of elements being popped or displayed, respectively.

1. Push Operation:

Time Complexity: O(1)

Best Case: O(1) (when the stack has space for the new element)

Average Case: O(1) (constant time operation)

Worst Case: O(1) (when the stack is not full and there is available space)

1. Pop Operation:

Time Complexity: O(1)

Best Case: O(1) (when the stack is not empty)

Average Case: O(1) (constant time operation)

Worst Case: O(1) (when the stack is not empty)

1. Multi-Pop Operation:

Time Complexity: O(n) (where n is the number of elements to be popped)

Best Case: O(1) (when the number of elements to be popped is very small)

Average Case: O(n) (linear time operation based on the number of elements to be popped)

Worst Case: O(n) (when the number of elements to be popped is equal to the size of the stack)

2. Write a program to implement INCREMENT operation in a k-bit binary counter that counts upward from 0. What happens to the counter as it is incremented 16 times? Find the amortized cost of this operation if sequences of n increment operations are performed.

A screenshot of a computer

Description automatically generated

A close-up of a white background

Description automatically generated

A computer screen shot of a computer screen

Description automatically generated

Analysis:

If there's a 1 in the list at position i, change it to 0 and move to the next position (i = i + 1).

If there's a 0 in the list at position i, change it to 1, show the reversed list, and if i is greater than 0, reset i to 0.

After that, the code calculates and shows the total and average cost based on the number of actions performed (NOP).

Time complexity:

* Displaying the initial list takes O(n) time, where n is 4 (the length of the list).
* The loop continues as long as i is less than 4. In the worst case, it loops 4 times, each time doing a fixed number of actions.
* The actions within the loop are all quick, and the loop iterates 4 times no matter the input. This makes the entire code's time complexity O(1), meaning it's really fast.

Different Cases:

1. Best Case: When the list is all 0s initially, no actions are needed, and the time complexity is O(1).
2. Worst Case: When the list starts with non-zero values, all four actions happen (changing 1s to 0s and vice versa), and the loop repeats 4 times. Even then, the time complexity is O(1).
3. Average Case: Since the code's time complexity stays constant no matter the input, the average case time complexity is also O(1).