Delete Middle Element of Stack CodeStudio

This C++ program demonstrates two approaches for deleting the middle element from a stack using either recursion or iteration. The program uses the **stack** container from the C++ Standard Library.

Example:

Input: **13 89 2 44 7 34 22**

Output**: 13 89 2 7 34 22**

Approach 1: Function to delete the middle element from the stack using recursion

1. **solve (void solve(stack<int> &inputStack, int size, int count)):**
    - Recursive function to delete the middle element:
        - Base Case: If the count reaches the middle index (size / 2), the middle element is popped from the stack and the recursion terminates.
        - Recursive Case: The top element is removed and stored. The **solve** function is called recursively for the remaining elements.
        - After the recursion, the stored element is pushed back onto the stack.
2. **deleteMiddleElement (void deleteMiddleElement(stack<int> &inputStack, int size)):**
    - Calls the **solve** function to delete the middle element using recursion.

Time Complexity:

- **Deletion through recursion takes O(n) time complexity since each element is visited exactly once during the recursive process.**

Space Complexity:

- **The space complexity of the recursive approach is determined by the call stack. Each recursive call consumes space in the call stack. The maximum depth of recursion is the number of elements in the stack, resulting in O(n) space complexity.**

Approach 2: Function to remove the middle element from the stack using iteration

1. **removeMiddleElement (void removeMiddleElement(stack<int> &inputStack, int size)):**
    - Calculate the middle index.

  - Create a temporary stack (**tempStack**).
  - Move the first half of the elements from the input stack to **tempStack**.
  - Skip the middle element by popping it from the input stack.
  - Move the elements back from **tempStack** to the input stack.

Time Complexity:

- **The time complexity is O(n) since we iterate over the stack twice: once to move elements to the tempStack and once to move them back. Each iteration involves visiting each element once.**

Space Complexity:

- **The space complexity is O(n) due to the use of the tempStack, which can hold up to half of the elements from the input stack.** The other variables used in the function contribute to constant space overhead.