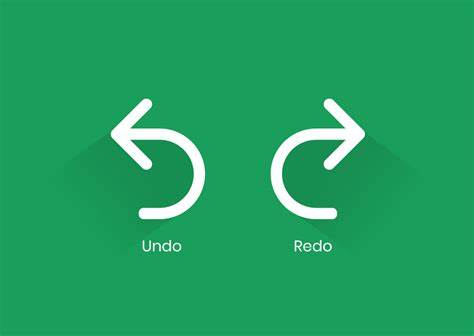
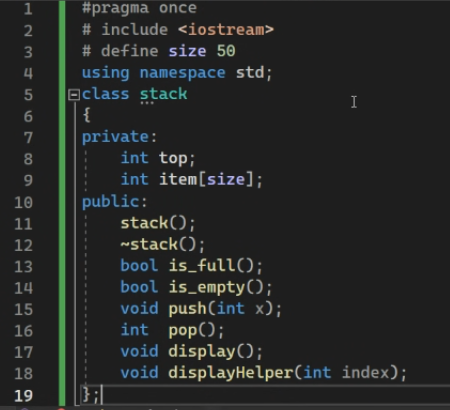
**Project Data Structures**

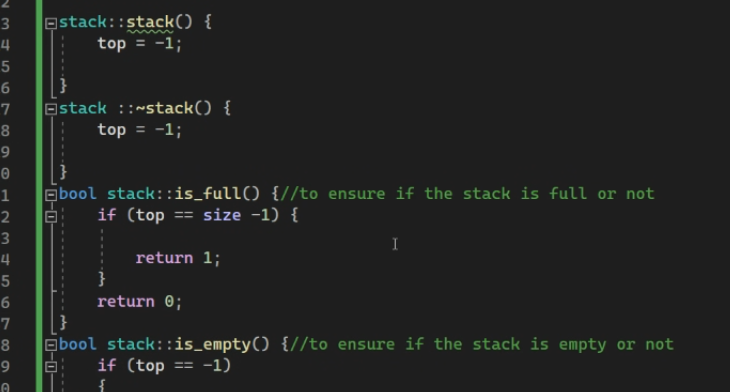
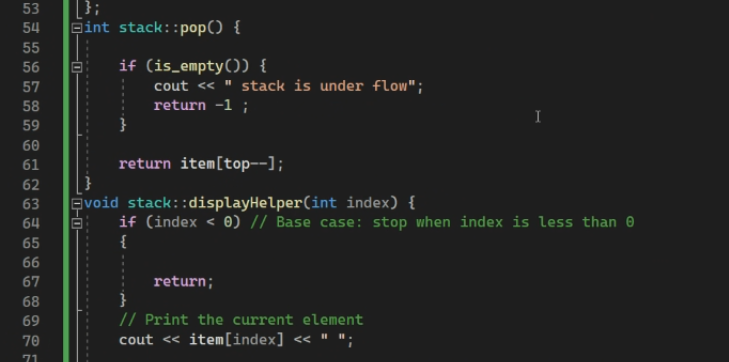
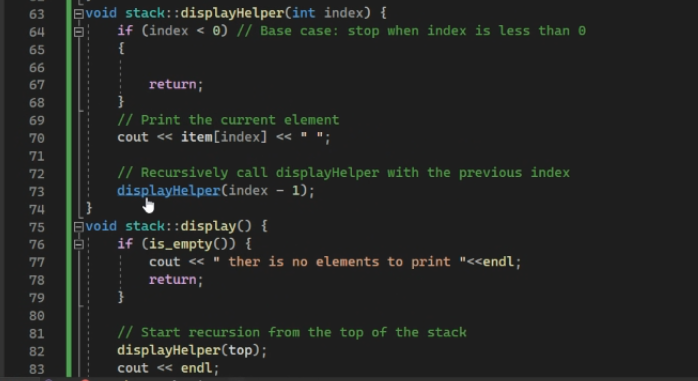
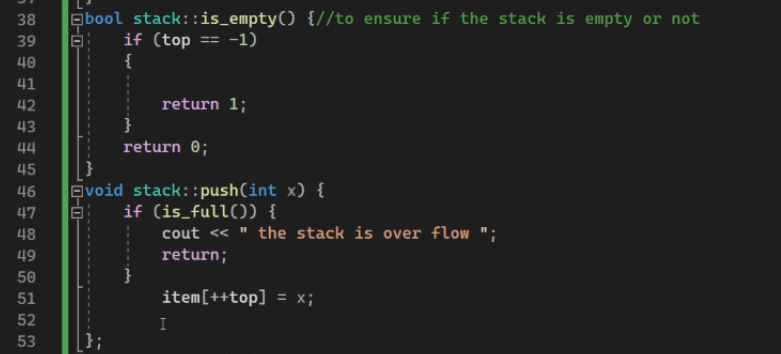
**Project Name : Undo/Redo Functionality**



**Header Files:**

* iostream: Provides input/output functionality (e.g., cin, cout)
* stack.h: Contains the definition of the stack class for managing the list of numbers
* 

**stack.h:**

* **Class stack:**
  + Private members:
    - top: An integer variable to keep track of the top index of the stack (initially -1)
    - item: An array of size 50 to store the stack elements
  + Public member functions:
    - Constructors and destructor:
      * stack(): Initializes top to -1.
      * ~stack(): (Empty destructor, typically used for cleanup)
    - Helper functions to check stack status:
      * is\_full(): Returns true if the stack is full (top == size - 1), false otherwise.
      * is\_empty(): Returns true if the stack is empty (top == -1), false otherwise.
    - Operations on the stack: 
      * push(int x): Adds an element x to the top of the stack (if not full).
      * int pop(): Removes and returns the top element from the stack (if not empty). Returns -1 on underflow.
      * void display(): Prints all elements of the stack in reverse order (top to bottom) using a recursive helper function displayHelper.
        + void displayHelper(int index): Recursive function that prints the element at the given index and then calls itself with the previous index until reaching the bottom (index < 0).
    - (Optional) Helper function for internal display logic:
      * void displayHelper(int index): Recursive function used by display() to print elements in reverse order.
      * 

**function.h:**

* **Function recurion():**
  + Declares two stack objects: s to store the main list of numbers and undo to store elements for undo operations.
  + Initializes variables:
    - x: Used to store user input (choice of operation).
    - y: Stores the number to be added.
    - f: Flag to control undo/redo behavior (1 for added, 0 for deleted).
    - g: Flag to handle potential underflow issues during stack pops (-1 indicates underflow).
    - k: Flag to control redo behavior (1 if undo has been used).
  + Loops until the user enters 0 (exit):
    - Prompts the user to choose an operation (add, delete, display, undo, redo, exit).
    - Performs the chosen operation based on x:
      * **Add (x == 1):**
        + Prompts the user for a number to add.
        + Pushes the number to the s stack.
        + Sets f to 1 to indicate an added element.
        + Sets g to 2 (avoiding potential underflow issues).
      * **Delete (x == 2):**
        + Pops an element from s (if not empty) and stores it in g.
        + Pushes the popped element to the undo stack for undo functionality.
        + Sets f to 0 to indicate a deleted element.
      * **Display (x == 3):**
        + Checks if s is empty. If empty, prints a message indicating no elements.
        + Otherwise, calls s.display() to print all elements in the stack.
      * **Undo (x == 4):**
        + Sets k to 1 to indicate undo has been used.
        + If f is 1 (added element):

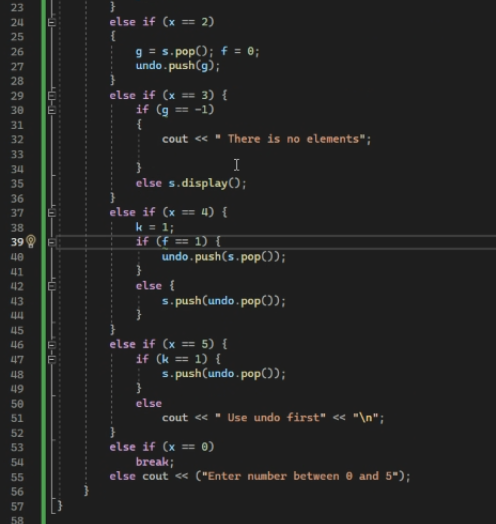
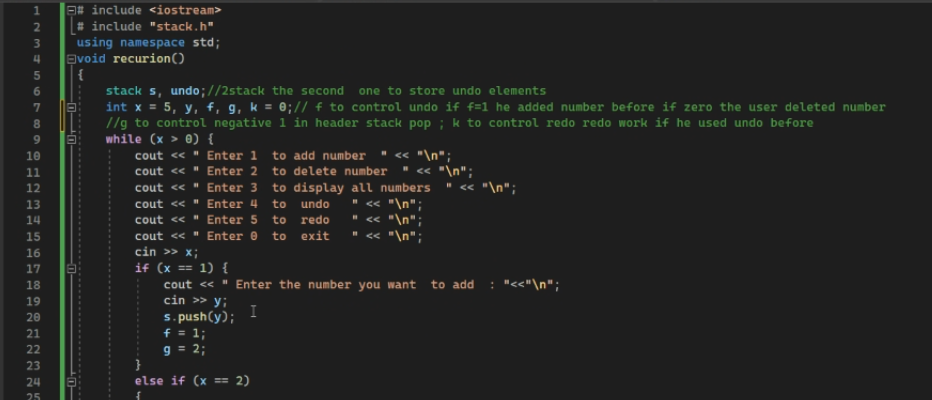
Pops an element from s (for consistency).

Pushes the popped element to undo (may not be strictly necessary for functionality).

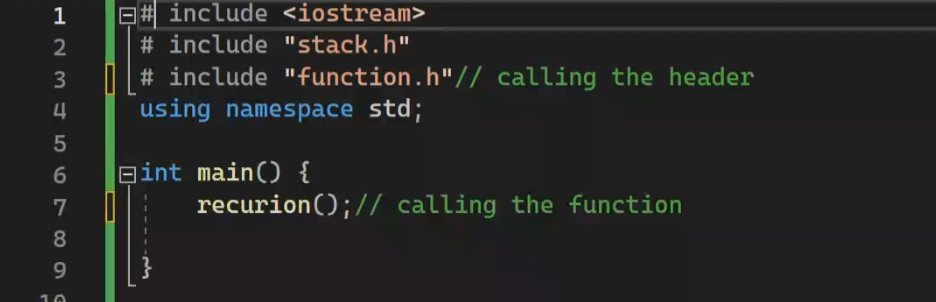
* + - * + Otherwise (deleted element):

Pops an element from undo and pushes it back to s.

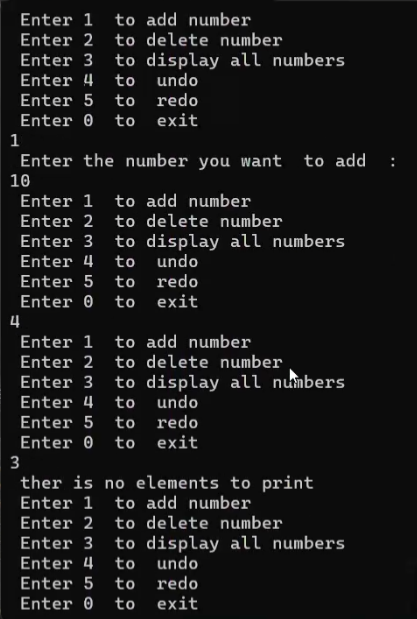
* + - * **Redo (x == 5):**

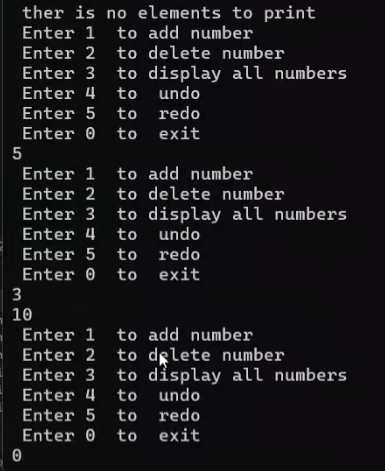
If k is 1 (undo has been used): 

* **Main .cppp**

****

**\*Outputs\***

****

****