Hands-on Activity 4.1	
Stacks	
Course Code: CPE010	**Program:** Computer Engineering
Course Title: Data Structures and Algorithms	**Date Performed: Aug.25,2926**
Section: CPE21S4	**Date Submitted: Aug.25,2026**
Name(s): Bautista,Mariela S.	**Instructor: Engr. Quejado**

1. Objective(s)

To implement the stack ADT in C++
To create an implementation of stack with different internal representations

2. Intended Learning Outcomes (ILOs)

After this activity, the student should be able to:
a. Create a stack using the C++ STL
b. Develop C++ code that uses both arrays and linked lists to create a stack
c. Solve problems using an implementation of stack

3. Discussion

Stack is a linear data structure which follows a particular order in which the operations are performed. The order may be LIFO(Last In First Out) or FILO(First In Last Out).
Image Source: Wikipedia.org/wiki/Stack_(ADT)

The following four basic operations can be performed in the stack:

- **Push:** Adds an item in the stack. If the stack is full, then it is said to be an Overflow condition.
- **Pop:** Removes an item from the stack. The items are popped in the reversed order in which they are pushed. If the stack is empty, then it is said to be an Underflow condition.
- **Peek or Top:** Returns top element of stack.
- **isEmpty:** Returns true if stack is empty, else false.

There are many real-life examples of a stack. Consider an example of plates stacked in the cupboard. The plate which is at the top is the first one to be removed, i.e. the plate which has been placed at the bottom most position remains in the stack for the longest period of time. So, it can be simply seen to follow LIFO(Last In First Out) / FILO(First In Last Out) order.

There are two ways to implement a stack:

- Using array
  - o Pros: Easy to implement. Memory is saved as pointers are not involved.
  - o Cons: It is not dynamic. It doesn't grow and shrink depending on needs at runtime.
- Using linked list
  - o Pros: The linked list implementation of stack can grow and shrink according to the needs at runtime.
  - o Cons: Requires extra memory due to involvement of pointers.

We will look at using a Linked list to implement a Stack. Most of what we need to know we have already covered in our discussion of Linked Lists - stacks only need a 'push' to build the stack. However, there are a couple of pieces that we need to add. In the "stack terminology" we have a 'pop' capability, which is just like the delete in our linked list. We also need to implement the 'peek' functionality - this simply returns the value that is sitting on the top of the stack but does not alter the stack in any way. Lastly, we will have to be able to determine if the stack is empty.

Attribution: " Stack Data Structure" by Patrick McClanahan, LibreTexts is licensed under CC BY-SA .

4. Materials and Equipment

Personal Computer with C++ IDE
Recommended IDE:

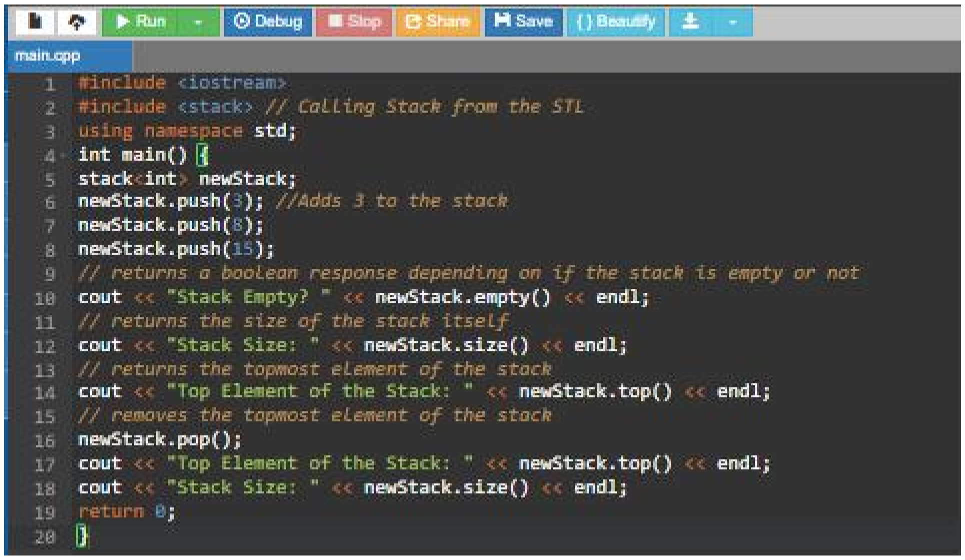
- ◆ CLion (must use TIP email to download)
- ◆ DevC++ (use the embarcadero fork or configure to C++17)

5. Procedure

ILO A: Create a stack using the C++ STL
Definition from the official CPP documentation for the stack STL.

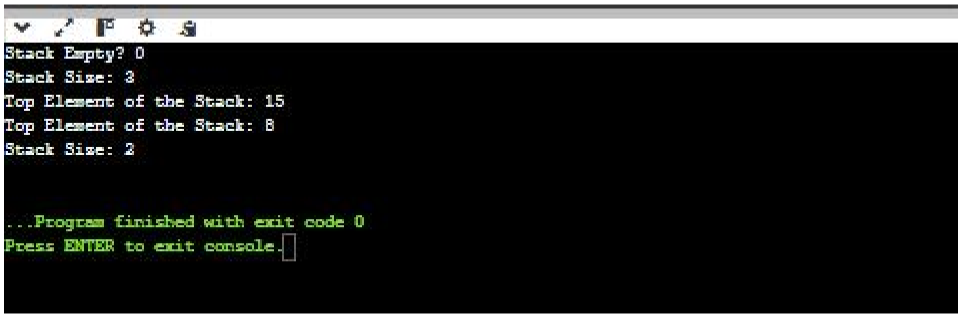
6. Output

ILO A: Create a stack using the C++ STL



```cpp
#include <iostream>
#include <stack> // Calling Stack from the STL
using namespace std;
int main() {
stack<int> newStack;
newStack.push(3); //Adds 3 to the stack
newStack.push(8);
newStack.push(15);
// returns a boolean response depending on if the stack is empty or not
cout << "Stack Empty? " << newStack.empty() << endl;
// returns the size of the stack itself
cout << "Stack Size: " << newStack.size() << endl;
// returns the topmost element of the stack
cout << "Top Element of the Stack: " << newStack.top() << endl;
// removes the topmost element of the stack
newStack.pop();
cout << "Top Element of the Stack: " << newStack.top() << endl;
cout << "Stack Size: " << newStack.size() << endl;
return 0;
}
```

Output:



```
Stack Empty? 0
Stack Size: 3
Top Element of the Stack: 15
Top Element of the Stack: 8
Stack Size: 2


...Program finished with exit code 0
Press ENTER to exit console.
```

Observation:

<u>After push(3), push(8), push(15)</u>

Stack Empty? 0

Stack Size: 3

Top Element of the Stack: 15

empty() returns 0 (false) → the stack is **not empty**.

size() is 3 → because we pushed 3 items.

top() is 15 → last element inserted is always at the top.

<u>After pop()</u>

```
Top Element of the Stack: 8
Stack Size: 2
```

pop() removed the top element (15).

Now the new top() is 8.

Size decreases from 3 → 2.

<u>Overall Remarks</u>

The STL stack makes stack operations clean and efficient without manually handling arrays or indexes.

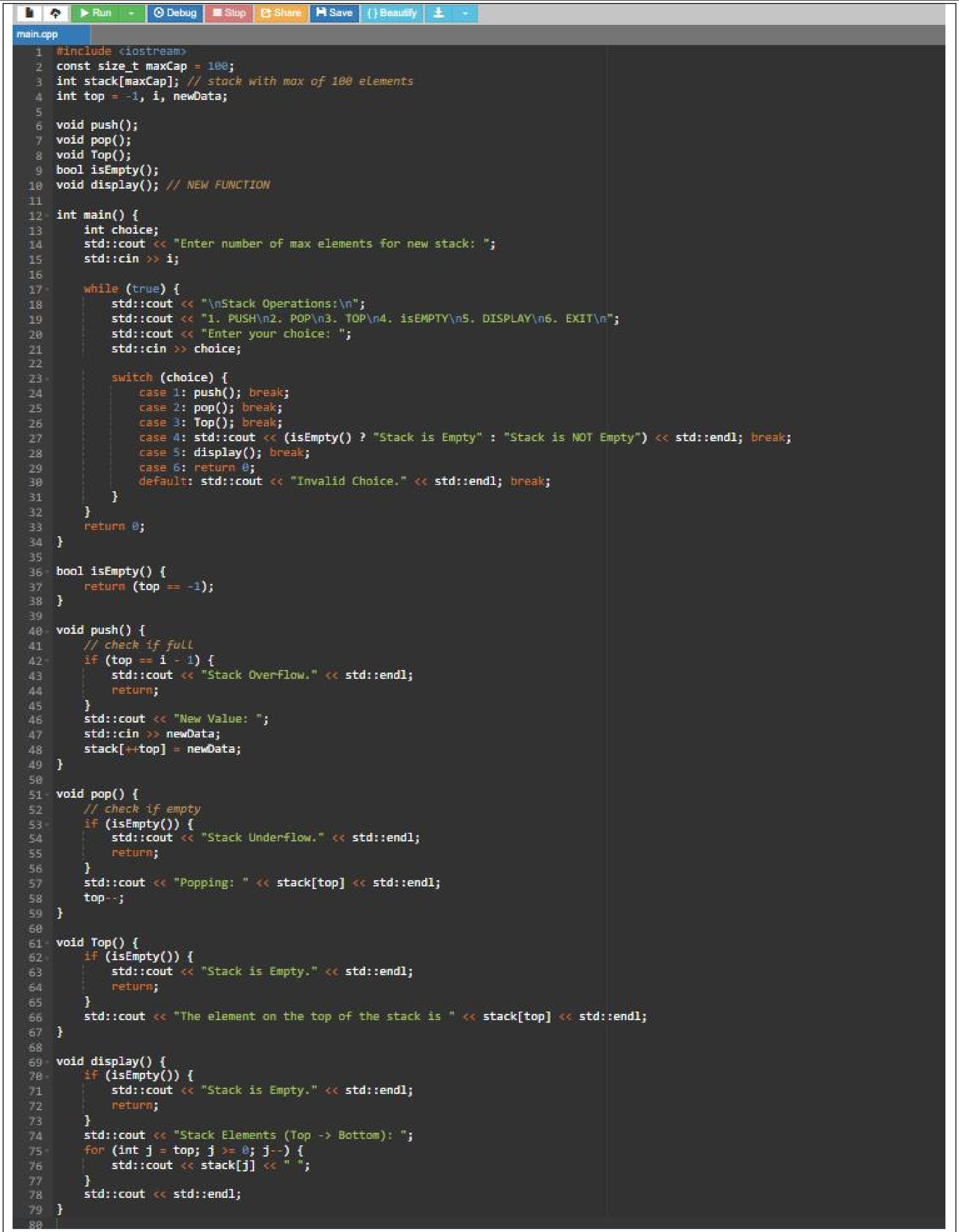
push() always inserts at the top.

pop() removes the top but doesn't return it (you check top() separately).

empty() gives a quick boolean check (0 = false, 1 = true).

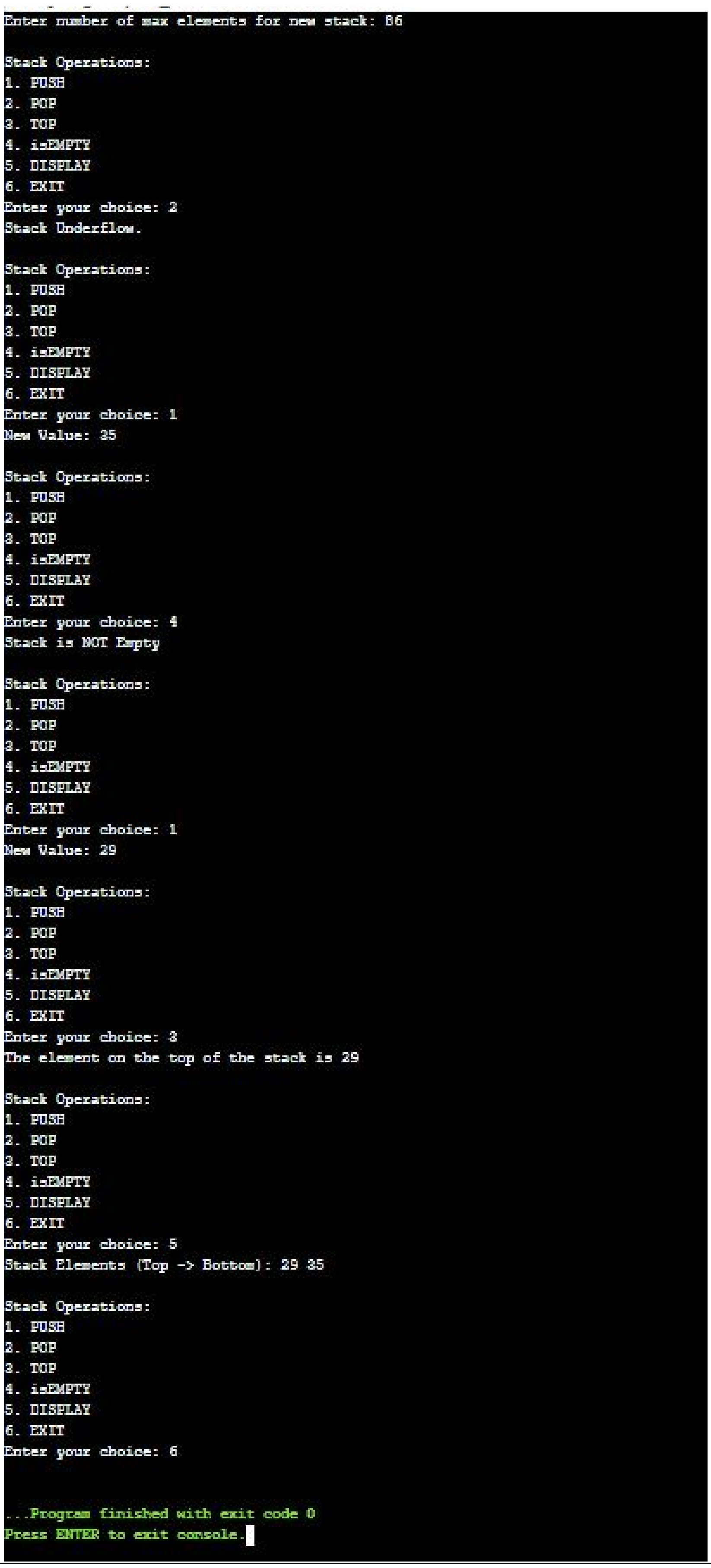
size() tracks the current number of elements in the stack.

Table 4-1. Output of ILO A



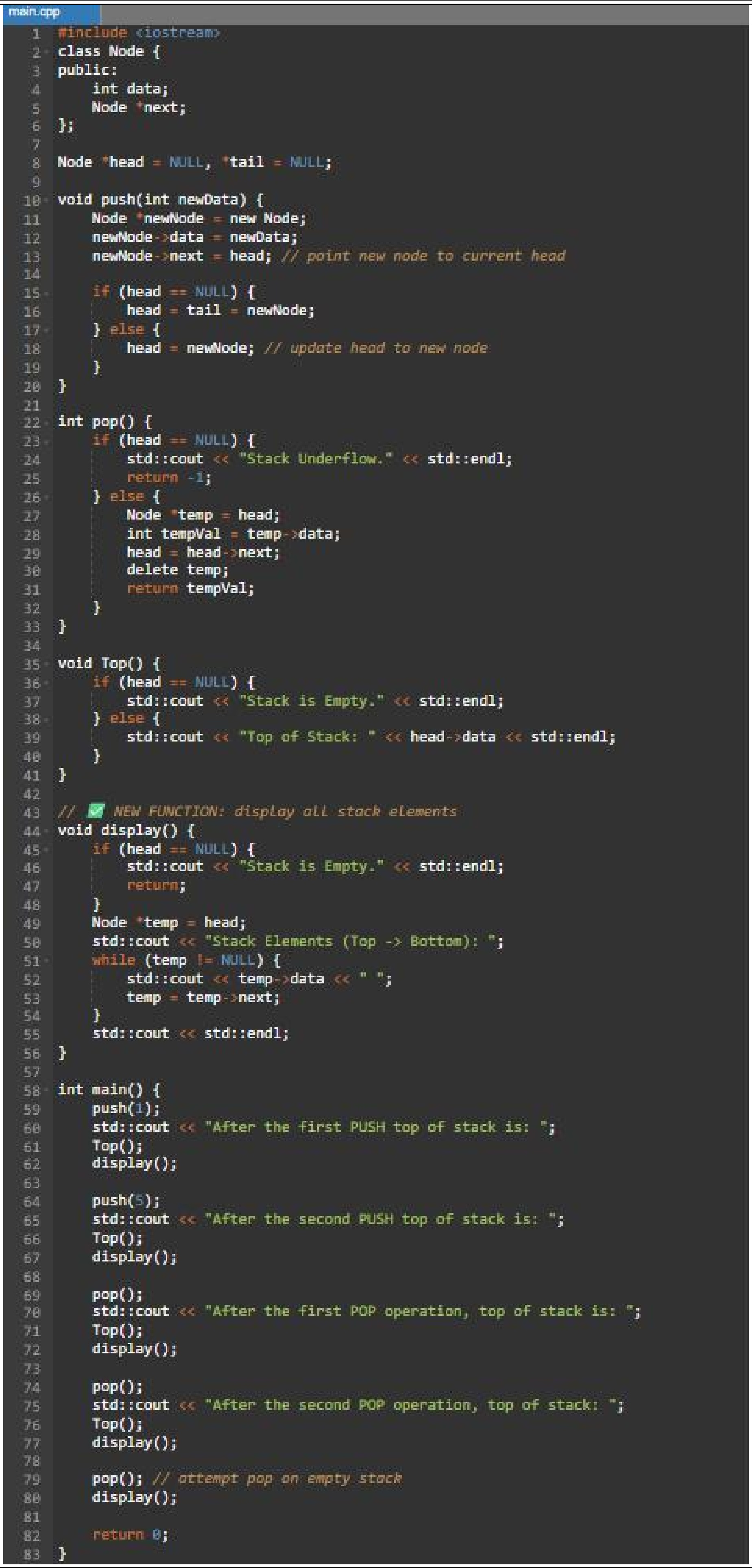
```cpp
#include <iostream>
const size_t maxCap = 100;
int stack[maxCap]; // stack with max of 100 elements
int top = -1, i, newData;

void push();
void pop();
void Top();
bool isEmpty();
void display(); // NEW FUNCTION

int main() {
    int choice;
    std::cout << "Enter number of max elements for new stack: ";
    std::cin >> i;

    while (true) {
        std::cout << "\nStack Operations:\n";
        std::cout << "1. PUSH\n2. POP\n3. TOP\n4. isEMPTY\n5. DISPLAY\n6. EXIT\n";
        std::cout << "Enter your choice: ";
        std::cin >> choice;

        switch (choice) {
            case 1: push(); break;
            case 2: pop(); break;
            case 3: Top(); break;
            case 4: std::cout << (isEmpty() ? "Stack is Empty" : "Stack is NOT Empty") << std::endl; break;
            case 5: display(); break;
            case 6: return 0;
            default: std::cout << "Invalid Choice." << std::endl; break;
        }
    }
    return 0;
}

bool isEmpty() {
    return (top == -1);
}

void push() {
    // check if full
    if (top == i - 1) {
        std::cout << "Stack Overflow." << std::endl;
        return;
    }
    std::cout << "New Value: ";
    std::cin >> newData;
    stack[++top] = newData;
}

void pop() {
    // check if empty
    if (isEmpty()) {
        std::cout << "Stack Underflow." << std::endl;
        return;
    }
    std::cout << "Popping: " << stack[top] << std::endl;
    top--;
}

void Top() {
    if (isEmpty()) {
        std::cout << "Stack is Empty." << std::endl;
        return;
    }
    std::cout << "The element on the top of the stack is " << stack[top] << std::endl;
}

void display() {
    if (isEmpty()) {
        std::cout << "Stack is Empty." << std::endl;
        return;
    }
    std::cout << "Stack Elements (Top -> Bottom): ";
    for (int j = top; j >= 0; j--) {
        std::cout << stack[j] << " ";
    }
    std::cout << std::endl;
}
```

OUTPUT:



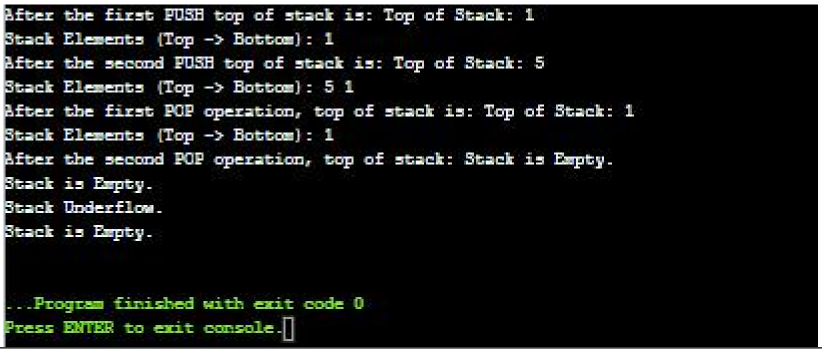
```
Enter number of max elements for new stack: 86

Stack Operations:
1. PUSH
2. POP
3. TOP
4. isEMPTY
5. DISPLAY
6. EXIT
Enter your choice: 2
Stack Underflow.

Stack Operations:
1. PUSH
2. POP
3. TOP
4. isEMPTY
5. DISPLAY
6. EXIT
Enter your choice: 1
New Value: 35

Stack Operations:
1. PUSH
2. POP
3. TOP
4. isEMPTY
5. DISPLAY
6. EXIT
Enter your choice: 4
Stack is NOT Empty

Stack Operations:
1. PUSH
2. POP
3. TOP
4. isEMPTY
5. DISPLAY
6. EXIT
Enter your choice: 1
New Value: 29

Stack Operations:
1. PUSH
2. POP
3. TOP
4. isEMPTY
5. DISPLAY
6. EXIT
Enter your choice: 3
The element on the top of the stack is 29

Stack Operations:
1. PUSH
2. POP
3. TOP
4. isEMPTY
5. DISPLAY
6. EXIT
Enter your choice: 5
Stack Elements (Top -> Bottom): 29 35

Stack Operations:
1. PUSH
2. POP
3. TOP
4. isEMPTY
5. DISPLAY
6. EXIT
Enter your choice: 6


...Program finished with exit code 0
Press ENTER to exit console.
```

Table 4-2. Ouput of ILO B.1



```cpp
#include <iostream>
class Node {
public:
    int data;
    Node *next;
};

Node *head = NULL, *tail = NULL;

void push(int newData) {
    Node *newNode = new Node;
    newNode->data = newData;
    newNode->next = head; // point new node to current head

    if (head == NULL) {
        head = tail = newNode;
    } else {
        head = newNode; // update head to new node
    }
}

int pop() {
    if (head == NULL) {
        std::cout << "Stack Underflow." << std::endl;
        return -1;
    } else {
        Node *temp = head;
        int tempVal = temp->data;
        head = head->next;
        delete temp;
        return tempVal;
    }
}

void Top() {
    if (head == NULL) {
        std::cout << "Stack is Empty." << std::endl;
    } else {
        std::cout << "Top of Stack: " << head->data << std::endl;
    }
}

// ✅ NEW FUNCTION: display all stack elements
void display() {
    if (head == NULL) {
        std::cout << "Stack is Empty." << std::endl;
        return;
    }
    Node *temp = head;
    std::cout << "Stack Elements (Top -> Bottom): ";
    while (temp != NULL) {
        std::cout << temp->data << " ";
        temp = temp->next;
    }
    std::cout << std::endl;
}

int main() {
    push(1);
    std::cout << "After the first PUSH top of stack is: ";
    Top();
    display();

    push(5);
    std::cout << "After the second PUSH top of stack is: ";
    Top();
    display();

    pop();
    std::cout << "After the first POP operation, top of stack is: ";
    Top();
    display();

    pop();
    std::cout << "After the second POP operation, top of stack: ";
    Top();
    display();

    pop(); // attempt pop on empty stack
    display();

    return 0;
}
```

OUTPUT:



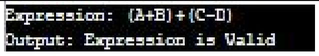
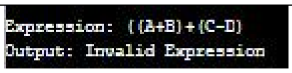
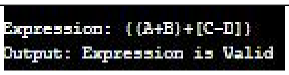
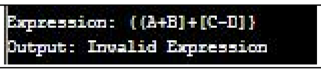
```
After the first PUSH top of stack is: Top of Stack: 1
Stack Elements (Top -> Bottom): 1
After the second PUSH top of stack is: Top of Stack: 5
Stack Elements (Top -> Bottom): 5 1
After the first POP operation, top of stack is: Top of Stack: 1
Stack Elements (Top -> Bottom): 1
After the second POP operation, top of stack: Stack is Empty.
Stack is Empty.
Stack Underflow.
Stack is Empty.


...Program finished with exit code 0
Press ENTER to exit console.
```

Table 4-3. Ouput of ILO B.2

7. Supplementary Activity

ILO C: Solve problems using an implementation of stack:

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)	Yes	 Expression: (A+B)+(C-D) Output: Expression is Valid	Because all symbols are properly matched.
((A+B)+(C-D)	No	 Expression: ((A+B)+(C-D) Output: Invalid Expression	There is a missing closing parenthesis.
((A+B)+[C-D])	Yes	 Expression: ((A+B)+[C-D]) Output: Expression is Valid	Nested symbols are properly matched.
((A+B]+[C-D]}	No	 Expression: ((A+B]+[C-D]} Output: Invalid Expression	Closing symbol mismatch.

Tools Analysis:

How do the different internal representations affect the implementation and usage of the stack?

- **Array-based Stack:** This is a fast and simple but also limited by fixed size.
- **Linked List Stack:**This is a Flexible and more dynamic but it requires more memory.
- **STL Stack:** Very easy to use since it's built-in to C++, less chance of errors, but you can't control how it works inside.

8. Conclusion

Summary of Lessons Learned:
What I have learned n this activity, is how stack operations (push, pop, top, empty) are applied using different implementations: array-based, linked list, and STL stack. I also practiced checking the validity of expressions using stack logic to match parentheses and other symbols.

Analysis of the Procedure:
The step-by-step coding process showed how each stack type works. Using arrays was straightforward but restrictive, while linked lists gave more flexibility. The STL stack simplified the process the most, making implementation faster and less prone to bugs.

Analysis of the Supplementary Activity:
Validating expressions reinforced the importance of the LIFO which stands by “Last In, First Out” principle. The stack correctly identifies mismatched or missing symbols, proving its real-world usefulness in parsing and compilers.

Concluding Statement / Feedback:
I think I did well in understanding the fundamental concept of stacks and applying them in code. However, I need to improve on debugging skills (like spotting typos or missing characters) and practice writing cleaner, more efficient code. Overall, this activity helped me strengthen both my coding ability and my logical reasoning. And fixing the bugs as much faster.

9. Assessment Rubric