Activity No. 4.1	
Stacks	
Course Code: CPE010	**Program:** Computer Engineering
Course Title: Data Structures and Algorithms	**Date Performed:** 10/04/24
Section: Cpe21s4	**Date Submitted:** 10/06/24
Name(s): Anna Marie Zolina	**Instructor:** Mrs. Maria Rizette Sayo

6. Output

main.cpp

```cpp
//Tests the push, empty, size, pop, and top methods of the stack library.
#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

```
/tmp/lRYtIs8zQV.o
Stack Empty? 0
Stack Size: 3
Top Element of the Stack: 15
Top Element of the Stack: 8
Stack Size: 2


=== Code Execution Successful ===
```

Observation:

1. **Push (push())**
   - This function adds elements to the stack in LIFO (Last In First Out) by adding the first elements to the bottom and the previous ones to the top. In this code by adding the 3 first it comes to the bottom of the stack, and the last element of 15 being on the top.

2. **Empty (empty())**
   - This Function indicates whether the stack contains any element or not.

3. **Size of Stack (size())**
   - This function indicates how many elements are pushed inside of the stack.

4. **Top Element (top())**
   - This Function puts the last element pushed from the stack to the top. In this code 15 is the last one to be pushed, therefore 15 is the top element of this stack.

5. **Pop (push())**
   - This Function removes the element based on the type of data structures are in use. In this code the function removes the topmost element because it is indicated there that the manner that has to be followed is the LIFO (Last In First OUT).

Table 4-1. Output of ILO A

MODIFIED CODE:

```
        break;
case 5: displayStack(); // Call to displayStack
        break;
default: cout << "Invalid Choice " << endl;
```

```
void display() {
    if (isEmpty()) {
        cout << "Stack is Empty." << endl;
        return;
    }

    cout << "Elements in the stack: ";
    int s = 0; // Use 's' as the loop counter
    while (s <= top) {
        cout << stack[s] << " ";
        s++;
    }
    cout << endl;
}
```

```
Output
/tmp/iteyZmkenx.o
Enter number of max elements for new stack: 3
Stack Operations:
1. PUSH, 2. POP, 3. TOP, 4. isEMPTY, 5. DISPLAY
1
New Value:
10
Stack Operations:
1. PUSH, 2. POP, 3. TOP, 4. isEMPTY, 5. DISPLAY
1
New Value:
20
Stack Operations:
1. PUSH, 2. POP, 3. TOP, 4. isEMPTY, 5. DISPLAY
1
New Value:
30
Stack Operations:
1. PUSH, 2. POP, 3. TOP, 4. isEMPTY, 5. DISPLAY
5
Elements in the stack: 10 20 30
Stack Operations:
1. PUSH, 2. POP, 3. TOP, 4. isEMPTY, 5. DISPLAY

=== Session Ended. Please Run the code again ===
```

OPERATIONS DESCRIPTIONS:

1. **PUSH:**
   - This operation is the one that adds a new item on the top of the stack.

2. **POP:**
   - This operation is the one that removes the item from the top of the stack.

3. **TOP:**
- This operation is the one that displays the item that is currently at the top of the stack.

4. **isEMPTY:**
- This operation is the one that checks whether the stack is already empty. if true it displays empty, otherwise it is false.

5. **DISPLAY:**
- This operation is the one that shows all the items that are within the stack from the bottom or the first one to be pushed, to the top.

Table 4-2. Output of ILO B.1.

MODIFIED CODE:

```
void Display() {
    if (head == NULL) {
        cout << "Stack is Empty." << endl; // Inform if empty
        return;
    }

    Node *current = head; // Start from head
    cout << "Stack Elements: ";
    while (current != NULL) {
        cout << current->data << " "; // Print each element
        current = current->next; // Move to next node
    }
    cout << endl; // New line after displaying all elements
}
```

```
int main() {
    push(1);
    cout << "After the first PUSH, top of stack is: ";
    Top();
    Display(); // Call to display the elements

    push(5);
    cout << "After the second PUSH, top of stack is: ";
    Top();
    Display(); // Call to display the elements

    pop();
    cout << "After the first POP operation, top of stack is: ";
    Top();
    Display(); // Call to display the elements

    pop();
    cout << "After the second POP operation, top of stack is: ";
    Top();
    Display(); // Call to display the elements

    pop(); // Attempt to pop from an empty stack
    return 0;
}
```

```
Output
/tmp/Wm1uhC5gId.o
After the first PUSH, top of stack is: Top of Stack: 1
Stack Elements: 1
After the second PUSH, top of stack is: Top of Stack: 5
Stack Elements: 5 1
After the first POP operation, top of stack is: Top of Stack: 1
Stack Elements: 1
After the second POP operation, top of stack is: Stack is Empty.
Stack is Empty.
Stack Underflow.


=== Code Execution Successful ===
```

OPERATION DESCRIPTIONS:

1. **PUSH:**
- The one that adds the item to the top.

2. **POP:**
- The one that removes the item from the top of the stack, if empty the stack will display “Stack Underflow”.

3. **TOP:**
- The one that shows the top of the stack.

4. **isEMPTY:**
- The one that indicates whether the stack is empty.

5. **DISPLAY:**
- The one to display all the elements from top to bottom.

Table 4-3. Output of ILO B.2.

7. Supplementary Activity

Array-Based Stack

```
#include <iostream>
#include <string>

class ArrayStack {
    static const int MAX = 100; // Maximum stack size
    char arr[MAX];
    int top;

public:
    ArrayStack() : top(-1) {}

    void push(char c) {
        if (top < MAX - 1) arr[++top] = c; // Only push if there's space
```

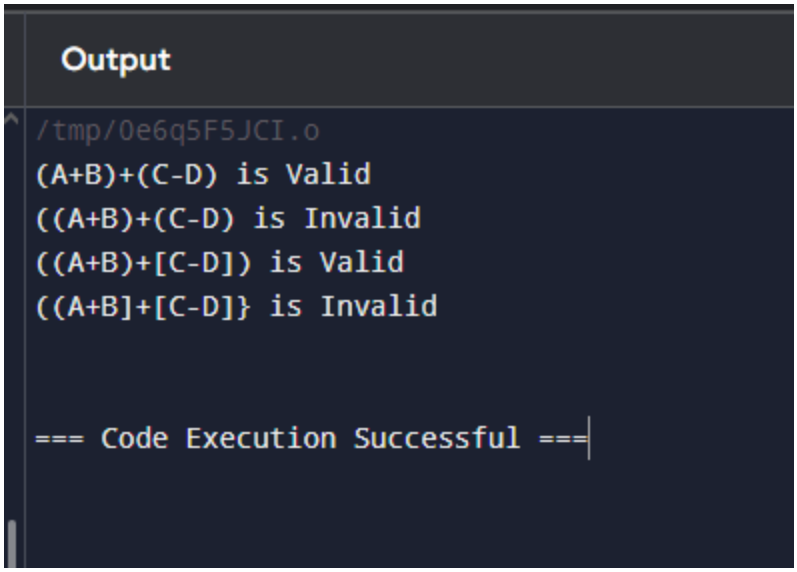
```
    }

    char pop() {
        return (top >= 0) ? arr[top--] : '\0'; // Pop or return '\0' if stack is empty
    }

    bool isEmpty() {
        return top == -1; // Check if stack is empty
    }
};

bool isBalanced(const std::string &expr) {
    ArrayStack stack;
    for (char c : expr) {
        if (c == '(' || c == '{' || c == '[') stack.push(c); // Push opening brackets
        else if (c == ')' || c == '}' || c == ']') {
            char topChar = stack.pop();
            if ((c == ')' && topChar != '(') || (c == '}' && topChar != '{') || (c == ']' && topChar != '['))
                return false; // Return false for mismatches
        }
    }
    return stack.isEmpty(); // Ensure no unmatched opening symbols
}

int main() {
    std::string expressions[] = {"(A+B)+(C-D)", "((A+B)+(C-D)", "((A+B)+[C-D])", "((A+B]+[C-D]}"};
    for (const auto &expr : expressions) {
        std::cout << expr << " is " << (isBalanced(expr) ? "Valid" : "Invalid") << std::endl;
    }
    return 0;
}
```



B. Linked List-Based Stack

```
#include <iostream>
#include <string>

struct Node {
    char data;
    Node* next;
};

class LinkedListStack {
    Node* top;

public:
    LinkedListStack() : top(nullptr) {}

    void push(char c) {
        Node* newNode = new Node{c, top}; // Create a new node
        top = newNode; // Make it the top
    }

    char pop() {
        if (top == nullptr) return '\0'; // If empty, return '\0'
        char data = top->data;
        Node* temp = top;
        top = top->next;
        delete temp; // Free memory
        return data;
    }

    bool isEmpty() {
        return top == nullptr; // Check if stack is empty
    }
};

bool isBalancedLL(const std::string &expr) {
    LinkedListStack stack;
    for (char c : expr) {
        if (c == '(' || c == '{' || c == '[') stack.push(c); // Push opening brackets
        else if (c == ')' || c == '}' || c == ']') {
            char topChar = stack.pop();
            if ((c == ')' && topChar != '(') || (c == '}' && topChar != '{') || (c == ']' && topChar != '['))
                return false; // Return false for mismatches
        }
    }
    return stack.isEmpty(); // Ensure no unmatched opening symbols
}

int main() {
    std::string expressions[] = {"(A+B)+(C-D)", "((A+B)+(C-D)", "((A+B)+[C-D])", "((A+B]+[C-D])"};
    for (const auto &expr : expressions) {
```

```
        std::cout << expr << " is " << (isBalancedLL(expr) ? "Valid" : "Invalid") << std::endl;
    }
    return 0;
}
```

8. Conclusion

Provide the following:
• Summary of lessons learned
• Analysis of the procedure
• Analysis of the supplementary activity
• Concluding statement / Feedback: How well did you think you did in this activity? What are your areas for improvement?

- Through this activity, I gained knowledge about stack implementations and how compilers use them for processing expressions and verify that symbols are balanced. My experience with arrays, linked lists, and the C++ STL helped me weigh the advantages and disadvantages of each data structure. Although I think I performed fairly well overall, I still need to work on my comprehension of balanced symbols and knowing when to recognize them in expressions. I'm going to concentrate on this in order to improve my programming abilities.

9. Assessment Rubric