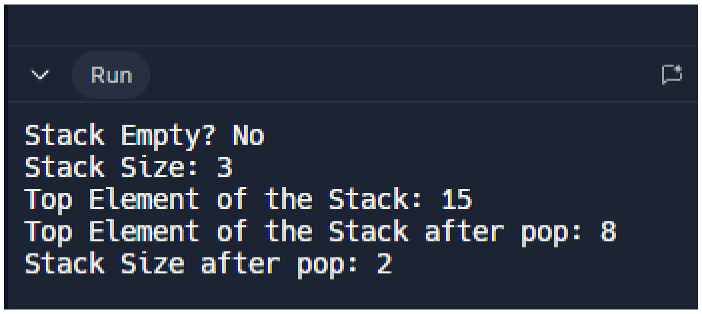
Activity No. 4	
STACKS	
Course Code: CPE010	**Program:** Computer Engineering
Course Title: Data Structures and Algorithms	**Date Performed:** Oct 4, 2024
Section: CPE21S4	**Date Submitted:** Oct 6, 2024
Name(s): Danica T. Guariño	**Instructor: Maria Rizette Sayo**
6. Output	

C/C++

```cpp
#include <iostream>
#include <stack>

using namespace std;

int main() {
    stack<int> newStack;


    newStack.push(3);
    newStack.push(8);
    newStack.push(15);


    cout << "Stack Empty? " << (newStack.empty() ? "Yes" : "No") << endl;


    cout << "Stack Size: " << newStack.size() << endl;


    cout << "Top Element of the Stack: " << newStack.top() << endl;


    newStack.pop();
    cout << "Top Element of the Stack after pop: " << newStack.top() << endl;
    cout << "Stack Size after pop: " << newStack.size() << endl;

    return 0;
}
```

Output:



```
Stack Empty? No
Stack Size: 3
Top Element of the Stack: 15
Top Element of the Stack after pop: 8
Stack Size after pop: 2
```

Table 4-1. Output of ILO A

C/C++

```cpp
#include <iostream>

const size_t maxCap = 100;
int stack[maxCap]; // stack with max of 100 elements
int top = -1, newData;

void push(int maxElements);
void pop();
void Top();
bool isEmpty();
void displayStack();

int main() {
    int choice, maxElements;
    std::cout << "Enter number of max elements for new stack (up to " << maxCap << "): ";
    std::cin >> maxElements;


    if (maxElements <= 0 || maxElements > maxCap) {
        std::cout << "Invalid input. Max elements must be between 1 and " << maxCap <<
std::endl;
        return 1;
    }

    while (true) {
        std::cout << "Stack Operations: " << std::endl;
        std::cout << "1. PUSH, 2. POP, 3. TOP, 4. isEMPTY, 5. DISPLAY, 6. EXIT" <<
std::endl;
        std::cin >> choice;
        switch (choice) {
        case 1:
            push(maxElements);
            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:
            displayStack();
            break;
        case 6:
                std::cout << "Exiting..." << std::endl;
                return 0;
            default:
                std::cout << "Invalid choice. Please try again." << std::endl;
            }
        }
    }

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

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

    void pop() {

        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 displayStack() {
        if (isEmpty()) {
            std::cout << "Stack is Empty." << std::endl;
            return;
        }
        std::cout << "Stack contents: ";
        for (int i = 0; i <= top; i++) {
            std::cout << stack[i] << " ";
        }
        std::cout << std::endl;
    }
```

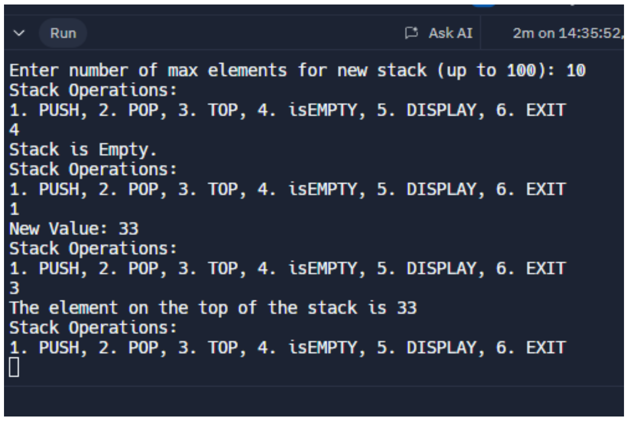
Output	
	```
	Enter number of max elements for new stack (up to 100): 10
	Stack Operations:
	1. PUSH, 2. POP, 3. TOP, 4. isEMPTY, 5. DISPLAY, 6. EXIT
	4
	Stack is Empty.
	Stack Operations:
	1. PUSH, 2. POP, 3. TOP, 4. isEMPTY, 5. DISPLAY, 6. EXIT
	1
	New Value: 33
	Stack Operations:
	1. PUSH, 2. POP, 3. TOP, 4. isEMPTY, 5. DISPLAY, 6. EXIT
	3
	The element on the top of the stack is 33
	Stack Operations:
	1. PUSH, 2. POP, 3. TOP, 4. isEMPTY, 5. DISPLAY, 6. EXIT
	```

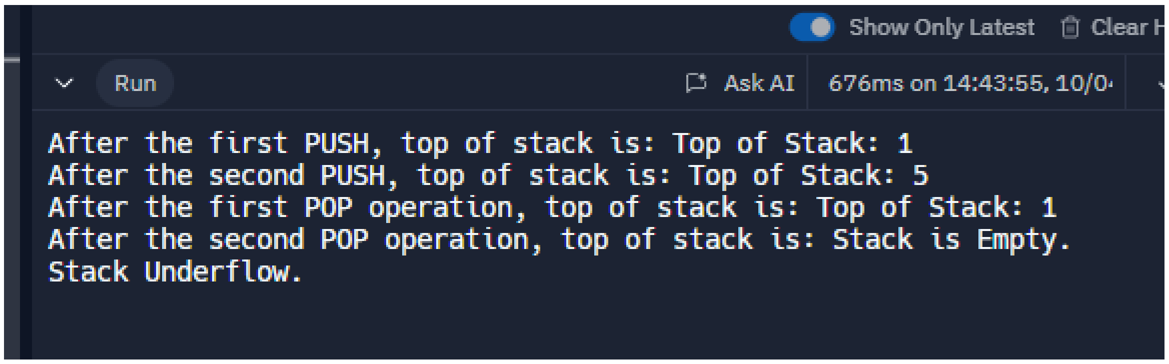
Table 4-2. Output of ILO B.1.

C/C++

```
#include <iostream>

class Node {
public:
    int data;
    Node* next;
};

Node* head = nullptr;

void push(int newData) {
    Node* newNode = new Node;
    newNode->data = newData;
    newNode->next = head;
    head = newNode;
}

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

void displayStack() {
    if (head == nullptr) {
        std::cout << "Stack is Empty." << std::endl;
        return;
    }
    Node* temp = head;
    std::cout << "Stack contents: ";
    while (temp != nullptr) {
        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();
    push(5);
    std::cout << "After the second PUSH, top of stack is: ";
    Top();
    pop();
    std::cout << "After the first POP operation, top of stack is: ";
    Top();
    pop();
    std::cout << "After the second POP operation, top of stack is: ";
    Top();
    pop(); // This will show stack underflow
    return 0;
}
```

Output



```
After the first PUSH, top of stack is: Top of Stack: 1
After the second PUSH, top of stack is: Top of Stack: 5
After the first POP operation, top of stack is: Top of Stack: 1
After the second POP operation, top of stack is: Stack is Empty.
Stack Underflow.
```

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

7. Supplementary Activity

ILO C: Solve problems using an implementation of stack:
The following problem definition and algorithm is provided for checking balancing of symbols. Create an implementation using stacks. Your output must include the following:
a. Stack using Arrays
b. Stack using Linked Lists
c. (Optional) Stack using C++ STL

Problem Definition:
Stacks can be used to check whether the given expression has balanced symbols. This algorithm is very useful in compilers. Each time the parser reads one character at a time. If the character is an opening delimiter such as (, {, or [- then it is written to the stack. When a closing delimiter is encountered like), }, or]-the stack is popped. The opening and closing delimiters are then compared. If they match, the parsing of the string continues. If they do not match, the parser indicates that there is an error on the line.

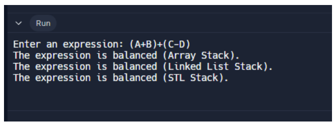
Steps:
1. Create a Stack.
2. While(end of input is not reached) {
a) If the character read is not a symbol to be balanced, ignore it.
b) If the character is an opening symbol, push it onto the stack.
c) If it is a closing symbol:
i) Report an error if the stack is empty.
ii) Otherwise, pop the stack.
d) If the symbol popped is not the corresponding opening symbol, report an error.
3. At the end of input, if the stack is not empty: report an error.

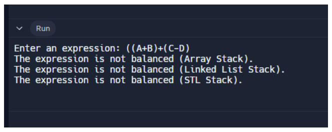
C/C++
```cpp
#include <iostream>
#include <stack> // For STL stack
#include <stdexcept>

using namespace std;


bool isOpeningDelimiter(char ch) {
    return ch == '(' || ch == '{' || ch == '[';
}

bool isClosingDelimiter(char ch) {
    return ch == ')' || ch == '}' || ch == ']';
}

bool isMatchingPair(char opening, char closing) {
    if (opening == '(' && closing == ')') return true;
    if (opening == '{' && closing == '}') return true;
    if (opening == '[' && closing == ']') return true;
    return false;
}
```

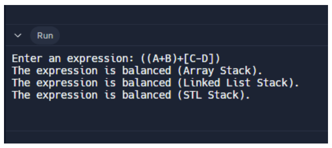
```
const int MAX_SIZE = 100;
class ArrayStack {
private:
    char data[MAX_SIZE];
    int top;

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

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

    bool isFull() { return top == MAX_SIZE - 1; }

    void push(char ch) {
        if (isFull()) {
            throw runtime_error("Stack Overflow");
        }
        data[++top] = ch;
    }

    char pop() {
        if (isEmpty()) {
            throw runtime_error("Stack Underflow");
        }
        return data[top--];
    }

    char peek() {
        if (isEmpty()) {
            throw runtime_error("Stack is Empty");
        }
        return data[top];
    }
};


bool checkBalancedArrayStack(const string& expr) {
    ArrayStack stack;
    for (char ch : expr) {
        if (isOpeningDelimiter(ch)) {
            stack.push(ch);
        } else if (isClosingDelimiter(ch)) {
            if (stack.isEmpty() || !isMatchingPair(stack.pop(), ch)) {
                return false; // Mismatch or empty stack
            }
        }
    }
    return stack.isEmpty(); // True if all delimiters match and stack is empty
}


class Node {
public:
    char data;
    Node* next;

    Node(char data) : data(data), next(nullptr) {}
};

class LinkedListStack {
private:
    Node* top;

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

    bool isEmpty() { return top == nullptr; }

    void push(char ch) {
        Node* newNode = new Node(ch);
        newNode->next = top;
        top = newNode;
    }

    char pop() {
        if (isEmpty()) {
            throw runtime_error("Stack Underflow");
        }
        char data = top->data;
        Node* temp = top;
        top = top->next;
        delete temp;
        return data;
    }

    char peek() {
        if (isEmpty()) {
            throw runtime_error("Stack is Empty");
        }
        return top->data;
    }
};

bool checkBalancedLinkedListStack(const string& expr) {
    LinkedListStack stack;
    for (char ch : expr) {
        if (isOpeningDelimiter(ch)) {
            stack.push(ch);
        } else if (isClosingDelimiter(ch)) {
            if (stack.isEmpty() || !isMatchingPair(stack.pop(), ch)) {
                return false; // Mismatch or empty stack
            }
        }
    }
    return stack.isEmpty(); // True if all delimiters match and stack is empty
}



bool checkBalancedSTLStack(const string& expr) {
    stack<char> st; // Using C++ STL stack
    for (char ch : expr) {
        if (isOpeningDelimiter(ch)) {
            st.push(ch);
        } else if (isClosingDelimiter(ch)) {
            if (st.empty() || !isMatchingPair(st.top(), ch)) {
                return false; // Mismatch or empty stack
            }
            st.pop();
        }
    }
    return st.empty(); // True if all delimiters match and stack is empty
}

int main() {
    string expression;
    cout << "Enter an expression: ";
    cin >> expression;

        try {
        if (checkBalancedArrayStack(expression)) {
            cout << "The expression is balanced (Array Stack)." << endl;
        } else {
            cout << "The expression is not balanced (Array Stack)." << endl;
        }
    } catch (const runtime_error& e) {
        cerr << "Error: " << e.what() << endl;
    }

        try {
        if (checkBalancedLinkedListStack(expression)) {
            cout << "The expression is balanced (Linked List Stack)." << endl;
        } else {
            cout << "The expression is not balanced (Linked List Stack)." << endl;
        }
    } catch (const runtime_error& e) {
        cerr << "Error: " << e.what() << endl;
    }

      if (checkBalancedSTLStack(expression)) {
        cout << "The expression is balanced (STL Stack)." << endl;
    } else {
        cout << "The expression is not balanced (STL Stack)." << endl;
    }

    return 0;
}
```

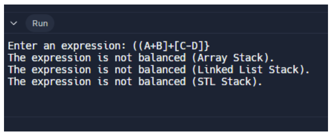
Self-Checking:
For the following cases, complete the table using the code you created.

Expression	Valid? (Y/N) Output	Output (Console Screenshot)	Analysis

(A+B)+(C-D)	Valid	 `Enter an expression: (A+B)+(C-D)`
		`The expression is balanced (Array Stack).`
		`The expression is balanced (Linked List Stack).`
		`The expression is balanced (STL Stack).`
			Since the operations used are correct, the output of this code is valid.

((A+B)+(C-D)	Not Valid / Invalid	 `Enter an expression: ((A+B)+(C-D)`
		`The expression is not balanced (Array Stack).`
		`The expression is not balanced (Linked List Stack).`
		`The expression is not balanced (STL Stack).`
			The use of parenthesis in the expression is incorrect that's why the output of the code is invalid.

((A+B)+[C-D])	Valid	 `Enter an expression: ((A+B)+[C-D])`
		`The expression is balanced (Array Stack).`
		`The expression is balanced (Linked List Stack).`
		`The expression is balanced (STL Stack).`
			Same with the first one, since the operations used are correct and also the use of the parenthesis and brackets are executed properly, the output of this code is valid.

((A+B]+[C-D]}	Not Valid / Invalid	 `Enter an expression: ((A+B]+[C-D]}`
		`The expression is not balanced (Array Stack).`
		`The expression is not balanced (Linked List Stack).`
		`The expression is not balanced (STL Stack).`
			Same with the second expression, since the operations used are incorrect and also the use of the parenthesis and brackets are not properly executed, the output of this code is invalid.

8. Conclusion

In conclusion, using stacks in C++ is usually easy, but it depends on what the program needs. Doing this activity is quite challenging for me but I know that I just need more practice to improve my skills in programming. Basic operations like adding and removing items are simple. However, more complex uses might need extra work to get everything working correctly. So, the difficulty varies based on the program's requirements.

9. Assessment Rubric