Activity No. 4	
Stacks	
Course Code: CPE010	**Program:** Computer Engineering
Course Title: Data Structures and Algorithms	**Date Performed: 10/4/24**
Section: CPE21S4	**Date Submitted: 10/4/24**
Name(s): CALVIN EARL PLANTA	**Instructor: Prof. Sayo**
6. Output	

Output:

```
Output

/tmp/zejDDWjJIr.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 ===
```

Modified:

```cpp
#include <iostream>
#include <stack>
using namespace std;

void displayStack(stack<int> s) {
    stack<int> tempStack;

    while (!s.empty()) {
        int topElement = s.top();
        cout << topElement << " ";
        tempStack.push(topElement);
        s.pop();
    }

    while (!tempStack.empty()) {
        s.push(tempStack.top());
        tempStack.pop();
    }
    cout << endl;
}

int main() {
    stack<int> newStack;
    newStack.push(3);
    newStack.push(8);
    newStack.push(15);

    cout << "Stack Empty? " << newStack.empty() << endl;

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

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

    cout << "Elements in the Stack: ";
    displayStack(newStack);

    newStack.pop();

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

    return 0;
}
```

```
Output

/tmp/vXOltjClMq.o
Stack Empty? 0
Stack Size: 3
Top Element of the Stack: 15
Elements in the Stack: 15 8 3
Top Element of the Stack after pop: 8
Stack Size after pop: 2


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

Operations used:

- push (int value) - Adds a new element to the top of the stack.
- pop() - Removes the top element from the stack. This operation does not return the removed element.
- top() - Returns the top element of the stack without removing it. This allows you to see which element is at the top.
- empty() - Checks whether the stack is empty. It returns true if there are no elements in the stack, otherwise returns false.
- size() - Returns the number of elements currently in the stack.
- displayStack(stack<int> s) - A custom function that displays all the elements in the stack. It accepts a stack by

Table 4-1. Output of ILO A

Output:

```
Output

/tmp/aUEdq0uBO0.o
Enter number of max elements for new stack: 5
Stack Operations:
1. PUSH, 2. POP, 3. TOP, 4. isEMPTY
1
New Value:
10
Stack Operations:
1. PUSH, 2. POP, 3. TOP, 4. isEMPTY
2
Popping: 10Stack Operations:
1. PUSH, 2. POP, 3. TOP, 4. isEMPTY
3
Stack is Empty.
Stack Operations:
1. PUSH, 2. POP, 3. TOP, 4. isEMPTY
4
1
Stack Operations:
1. PUSH, 2. POP, 3. TOP, 4. isEMPTY
5
Invalid Choice.
Stack Operations:
1. PUSH, 2. POP, 3. TOP, 4. isEMPTY
```

Modified:

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

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

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

    while (true) {
        std::cout << "Stack Operations: " << std::endl;
        std::cout << "1. PUSH, 2. POP, 3. TOP, 4. isEMPTY, 5. DISPLAY" << std::endl;
        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: displayStack();
                    break;
            default: std::cout << "Invalid Choice." << std::endl;
                    break;
        }
```

```
    }
    return 0;
}

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

void push() {
    if (top == i - 1) {
        std::cout << "Stack Overflow." << std::endl;
        return;
    }
    std::cout << "New Value: " << std::endl;
    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 << "Elements in the stack: ";
    for (int j = top; j >= 0; --j) {
        std::cout << stack[j] << " ";
    }
    std::cout << std::endl;
}
```

```
Output
/tmp/QKXfvOeiZm.o
Enter number of max elements for new stack: 10
Stack Operations:
1. PUSH, 2. POP, 3. TOP, 4. isEMPTY, 5. DISPLAY
1
New Value:
9
Stack Operations:
1. PUSH, 2. POP, 3. TOP, 4. isEMPTY, 5. DISPLAY
1
New Value:
8
Stack Operations:
1. PUSH, 2. POP, 3. TOP, 4. isEMPTY, 5. DISPLAY
1
New Value:
7
Stack Operations:
1. PUSH, 2. POP, 3. TOP, 4. isEMPTY, 5. DISPLAY
2
Popping: 7
Stack Operations:
1. PUSH, 2. POP, 3. TOP, 4. isEMPTY, 5. DISPLAY
2
Popping: 8
Stack Operations:
1. PUSH, 2. POP, 3. TOP, 4. isEMPTY, 5. DISPLAY
3
The element on the top of the stack is 9
Stack Operations:
1. PUSH, 2. POP, 3. TOP, 4. isEMPTY, 5. DISPLAY
4
Stack is not empty.
1. PUSH, 2. POP, 3. TOP, 4. isEMPTY, 5. DISPLAY
5
Elements in the stack: 9
Stack Operations:
1. PUSH, 2. POP, 3. TOP, 4. isEMPTY, 5. DISPLAY
```

Operations used:

- push () - Adds a new element to the top of the stack.
- pop() - Removes the top element from the stack. This operation does not return the removed element.
- top() - Returns the top element of the stack without removing it. This allows you to see which element is at the top.
- isEmpty() - This function checks if the stack is empty.
- displayStack() - This function prints all elements currently in the stack from top to bottom.

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

Output:

```
Output

/tmp/URfPnLRU3W.o
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 :Stack is Empty.
Stack Underflow.


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

Modified:

```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;
    head = newNode;
    if (tail == NULL) {
        tail = newNode;
    }
}

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

void displayStack() {
    if (head == NULL) {
        std::cout << "Stack is empty." << std::endl;
    } else {
        Node *current = head;
        std::cout << "Elements in Stack: ";
        while (current != NULL) {
            std::cout << current->data << " ";
            current = current->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();

    std::cout << "Current Stack: ";
    displayStack();

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

    std::cout << "Current Stack: ";
    displayStack();

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

    std::cout << "Current Stack: ";
    displayStack();

    pop();

    return 0;
}
```

```
Output

/tmp/1og9WyvSoi.o
After the first PUSH, top of stack is: Top of Stack: 1
After the second PUSH, top of stack is: Top of Stack: 5
Current Stack: Elements in Stack: 5 1
After the first POP operation, top of stack is: Top of Stack: 1
Current Stack: Elements in Stack: 1
After the second POP operation, top of stack is: Stack is Empty.
Current Stack: Stack is empty.
Stack Underflow.
```

Operations used:

- push (int newData) - Adds a new element to the top of the stack.
- pop() - Removes the top element from the stack. This operation does not return the removed element.
- top() - Returns the top element of the stack without removing it. This allows you to see which element is at the top.
- displayStack() - This function prints all elements currently in the stack from top to bottom.
- main() - The main function where the program execution begins. It demonstrates the usage of the stack by performing various operations.

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

7. Supplementary Activity

Stack using arrays:

```cpp
#include <iostream>
#include <cstring>
using namespace std;

#define MAX_SIZE 100

class Stack {
private:
    char items[MAX_SIZE];
    int top;

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

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

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

    void push(char item) {
        if (!isFull()) {
            items[++top] = item;
        }
    }

    char pop() {
        if (!isEmpty()) {
            return items[top--];
        }
        return '\0';
    }

```

```
    char peek() {
        if (!isEmpty()) {
            return items[top];
        }
        return '\0';
    }
};

bool isBalanced(const char* input) {
    Stack symbolStack;
    const char* openingSymbols = "({[";
    const char* closingSymbols = ")}]";

    for (int i = 0; input[i] != '\0'; ++i) {
        char ch = input[i];

        if (strchr(openingSymbols, ch) != nullptr) {
            symbolStack.push(ch);
        }
        else if (strchr(closingSymbols, ch) != nullptr) {
            if (symbolStack.isEmpty()) {
                std::cout << "Error: Unmatched closing symbol for '" << ch << endl;
                return false;
            } else {
                char topSymbol = symbolStack.pop();
                if ((topSymbol == '(' && ch != ')') ||
                    (topSymbol == '{' && ch != '}') ||
                    (topSymbol == '[' && ch != ']')) {
                    std::cout << "Error: Mismatched symbols '" << topSymbol << "' and '" <<
                        ch << endl;
                    return false;
                }
            }
        }
    }

    if (!symbolStack.isEmpty()) {
        std::cout << "Error: Unmatched opening symbol(s): ";
        while (!symbolStack.isEmpty()) {
            std::cout << symbolStack.pop() << " ";
        }
        std::cout << endl;
        return false;
    }

    return true;
}

int main() {
    char input[256];

    std::cout << "Enter a string to check for balanced symbols: ";
    std::cin.getline(input, 256);

    if (isBalanced(input)) {
        std::cout << "Symbols are balanced." << endl;
    } else {
        std::cout << "Symbols are not balanced." << endl;
    }

    return 0;
}
```

Self-Checking:

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

(A+B)+(C-D)	Y	Output /tmp/Myq28K36Zo.o Enter a string to check for balanced symbols: (A+B)+(C-D) Symbols are balanced. === Code Execution Successful ===	All parentheses (are paired with closed parentheses), therefore the symbols are balanced.
((A+B)+(C-D)	N	Output /tmp/sdtJbTjuNg.o Enter a string to check for balanced symbols: ((A+B)+(C-D)ERROR! Error: Unmatched opening symbol(s): (Symbols are not balanced. === Code Execution Successful ===	A (parentheses is unmatched, it is missing a closed parentheses, therefore the symbols are not balanced
((A+B)+[C-D])	Y	Output /tmp/Y6bpGXmw3y.o Enter a string to check for balanced symbols: ((A+B)+[C-D]) Symbols are balanced. === Code Execution Successful ===	All parentheses (are paired with closed parentheses), therefore the symbols are balanced.
((A+B]+[C-D]}	N	Output /tmp/7VKmm6d4dE.o Enter a string to check for balanced symbols: ((A+B]+[C-D]} ERROR! Error: Mismatched symbols '(' and '] Symbols are not balanced. === Code Execution Successful ===	A parentheses is mismatched with a closing brace rather than a closing parentheses, therefore symbols are not balanced.

Stacks using Linked Lists:

```cpp
#include <iostream>
#include <cstring>

struct Node {
    char data;
    Node* next;

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

class Stack {
private:
    Node* top;

public:
    Stack() : top(nullptr) {} // Initialize the stack

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

    void push(char item) {
        Node* newNode = new Node(item); // Create a new node
        newNode->next = top; // Link new node to the current top
        top = newNode; // Update top to the new node
    }

    char pop() {
        if (isEmpty()) {
            return '\0'; // Return null character if stack is empty
        }
        char poppedValue = top->data; // Get the value of the top node
        Node* temp = top; // Store the current top node
        top = top->next; // Update top to the next node
        delete temp; // Delete the old top node
        return poppedValue; // Return the popped value
```

```
    }

    char peek() {
        return isEmpty() ? '\0' : top->data; // Return the top item without removing
    }

    ~Stack() {
        while (!isEmpty()) {
            pop(); // Clear the stack
        }
    }
};

bool isBalanced(const char* input) {
    Stack symbolStack;
    const char* openingSymbols = "({[";
    const char* closingSymbols = ")}]";

    for (int i = 0; input[i] != '\0'; ++i) {
        char ch = input[i];

        // Check if the character is an opening symbol
        if (strchr(openingSymbols, ch) != nullptr) {
            symbolStack.push(ch); // Push opening symbol onto the stack
        }
        // Check if the character is a closing symbol
        else if (strchr(closingSymbols, ch) != nullptr) {
            if (symbolStack.isEmpty()) {
                std::cout << "Error: Unmatched closing symbol for '" << ch << "'\n";
                return false; // Unmatched closing symbol
            } else {
                char topSymbol = symbolStack.pop(); // Pop the top symbol
                if ((topSymbol == '(' && ch != ')') ||
                    (topSymbol == '{' && ch != '}') ||
                    (topSymbol == '[' && ch != ']')) {
                    std::cout << "Error: Mismatched symbols '" << topSymbol << "' and '"
                        << ch << "'\n";
                    return false; // Mismatched symbols
                }
            }
        }
        // Ignore non-symbol characters
    }

    // If the stack is not empty, there are unmatched opening symbols
    if (!symbolStack.isEmpty()) {
        std::cout << "Error: Unmatched opening symbol(s): ";
        while (!symbolStack.isEmpty()) {
            std::cout << symbolStack.pop() << " "; // Print unmatched opening symbols
        }
        std::cout << "\n";
        return false;
    }

    return true; // Symbols are balanced
}

int main() {
    char input[256];

    std::cout << "Enter a string to check for balanced symbols: ";
    std::cin.getline(input, 256); // Read input string

    if (isBalanced(input)) {
        std::cout << "Symbols are balanced.\n";
    } else {
```

```
103         std::cout << "Symbols are not balanced.\n";
104     }
105
106     return 0;
107 }
```

8. Conclusion

This exercise taught me how to create stacks and check for balance using C++ STL, linked lists, and arrays. expressions with symbols. Simple but constrained by fixed memory is the array-based stack; in contrast, the linked list offers dynamic allocation, albeit with additional management complexity. The STL stack makes implementation simpler by taking care of automatically in memory. Experimenting with various formulations strengthened my comprehension of managing nested symbols and edge cases. I applied these concepts successfully, but I can do better by addressing more complex edge cases, streamlining my code, and improving error messages.