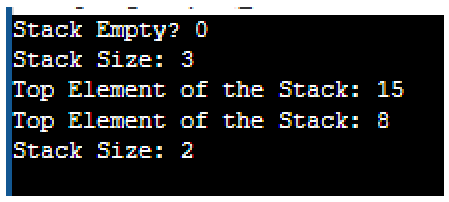
Activity No. 4	
STACKS	
Course Code: CPE010	**Program:** Computer Engineering
Course Title: Data Structures and Algorithms	**Date Performed:10/04/24**
Section:CPE21S3	**Date Submitted:10/04/24**
Name(s):Sanchez, Christan Ray R.	**Instructor: Prof. Ma Rizette Sayo**
6. Output	

Table 4.1

SOURCE CODE

```
//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

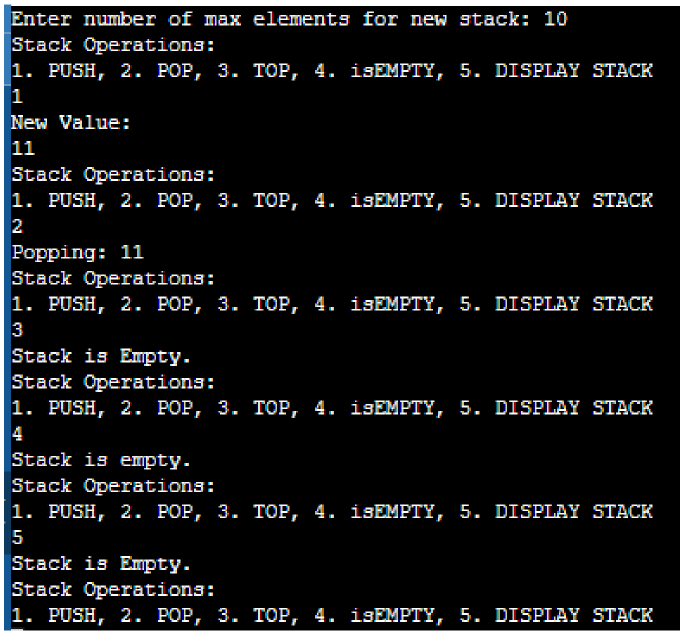


```
Stack Empty? 0
Stack Size: 3
Top Element of the Stack: 15
Top Element of the Stack: 8
Stack Size: 2
```

Table 4.2

SOURCE CODE

```
#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 displayStack();  // New function to display all
elements

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 STACK" << 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() {
    // check if full -> if yes, return error
    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() {
    // check if empty -> if yes, return error
    if(isEmpty()) {
        std::cout << "Stack Underflow." << std::endl;
        return;
    }
    // display the top value
    std::cout << "Popping: " << stack[top] << std::endl;
    // decrement top value from stack
    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;
}

// New function to display all elements in the stack
void displayStack() {
    if(isEmpty()) {
        std::cout << "Stack is Empty." << std::endl;
        return;
    }
    std::cout << "Stack elements: ";
    for(int j = 0; j <= top; ++j) {
        std::cout << stack[j] << " ";
    }
    std::cout << std::endl;
}
```

OUTPUT	

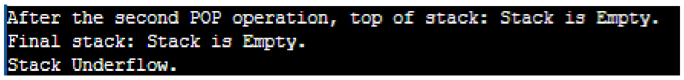
```
Enter number of max elements for new stack: 10
Stack Operations:
1. PUSH, 2. POP, 3. TOP, 4. isEMPTY, 5. DISPLAY STACK
1
New Value:
11
Stack Operations:
1. PUSH, 2. POP, 3. TOP, 4. isEMPTY, 5. DISPLAY STACK
2
Popping: 11
Stack Operations:
1. PUSH, 2. POP, 3. TOP, 4. isEMPTY, 5. DISPLAY STACK
3
Stack is Empty.
Stack Operations:
1. PUSH, 2. POP, 3. TOP, 4. isEMPTY, 5. DISPLAY STACK
4
Stack is empty.
Stack Operations:
1. PUSH, 2. POP, 3. TOP, 4. isEMPTY, 5. DISPLAY STACK
5
Stack is Empty.
Stack Operations:
1. PUSH, 2. POP, 3. TOP, 4. isEMPTY, 5. DISPLAY STACK
```

Table 4.3

SOURCE CODE

```
#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;
    if (head == NULL) {
        head = tail = newNode;
        newNode->next = NULL;
    } else {
        newNode->next = head;
        head = 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;
    }
}

// New function to display all elements in the stack
void displayStack() {
    if (head == NULL) {
        std::cout << "Stack is Empty." << std::endl;
        return;
    }
    Node *temp = head;
    std::cout << "Stack elements: ";
    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();

    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();

    pop();
```

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

	    std::cout << "Final stack: ";
	    displayStack();

	    pop(); // Will trigger stack underflow message
	    return 0;
	}
	```

OUTPUT	
	```
	After the second POP operation, top of stack: Stack is Empty.
	Final stack: Stack is Empty.
	Stack Underflow.
	```

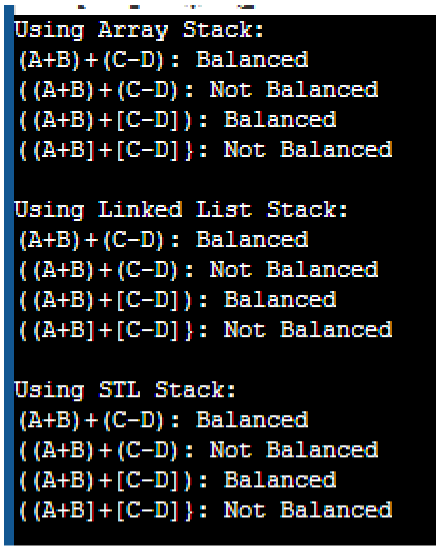
7. Supplementary Activity

A. Stack using Arrays

SOURCE CODE

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

#define MAX 100

// Array-based Stack Implementation
class ArrayStack {
    char stack[MAX];
    int top;

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

    void pushArray(char c) {
        if (top >= MAX - 1) {
            std::cout << "Array Stack Overflow\n";
            return;
        }
        stack[++top] = c;
    }

    char popArray() {
        if (isEmptyArray()) {
            std::cout << "Array Stack Underflow\n";
            return '\0';
        }
        return stack[top--];
    }
    bool isEmptyArray() {
        return top == -1;
    }
};

bool isMatchingPairArray(char open, char close) {
    return (open == '(' && close == ')') ||
           (open == '{' && close == '}') ||
           (open == '[' && close == ']');
}

bool checkBalancedArray(const std::string &expr) {
    ArrayStack stack;
    for (char c : expr) {
        if (c == '(' || c == '{' || c == '[') {
            stack.pushArray(c);
        } else if (c == ')' || c == '}' || c == ']') {
            if (stack.isEmptyArray() || !isMatchingPairArray(stack.popArray(), c)) {
                return false;
            }
        }
    }
    return stack.isEmptyArray();
}

// Linked List-based Stack Implementation
class Node {
public:
    char data;
    Node *next;
};

class LinkedListStack {
    Node *top;

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

    void pushLinkedList(char c) {
        Node *newNode = new Node();
        newNode->data = c;
        newNode->next = top;
        top = newNode;
    }

    char popLinkedList() {
        if (isEmptyLinkedList()) {
            std::cout << "Linked List Stack Underflow\n";
            return '\0';
        }
        Node *temp = top;
        char poppedValue = top->data;
        top = top->next;
        delete temp;
        return poppedValue;
    }

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

bool isMatchingPairLinkedList(char open, char close) {
    return (open == '(' && close == ')') ||
           (open == '{' && close == '}') ||
           (open == '[' && close == ']');
}

bool checkBalancedLinkedList(const std::string &expr) {
    LinkedListStack stack;
    for (char c : expr) {
        if (c == '(' || c == '{' || c == '[') {
            stack.pushLinkedList(c);
        } else if (c == ')' || c == '}' || c == ']') {
            if (stack.isEmptyLinkedList() || !isMatchingPairLinkedList(stack.popLinkedList(), c)) {
                return false;
            }
        }
    }
    return stack.isEmptyLinkedList();
}

// STL-based Stack Implementation
bool isMatchingPairSTL(char open, char close) {
    return (open == '(' && close == ')') ||
           (open == '{' && close == '}') ||
           (open == '[' && close == ']');
}

bool checkBalancedSTL(const std::string &expr) {
    std::stack<char> stack;
    for (char c : expr) {
        if (c == '(' || c == '{' || c == '[') {
            stack.push(c);
        } else if (c == ')' || c == '}' || c == ']') {
            if (stack.empty() || !isMatchingPairSTL(stack.top(), c)) {
                return false;
            }
            stack.pop();
        }
    }
    return stack.empty();
```

```
}

// Main Function to Test All Implementations
int main() {
    std::string expressions[] = {
        "(A+B)+(C-D)",
        "((A+B)+(C-D)",
        "((A+B)+[C-D])",
        "((A+B]+[C-D]}"
    };

    // Array Stack Test
    std::cout << "Using Array Stack:\n";
    for (const std::string &exp : expressions) {
        std::cout << exp << ": " << (checkBalancedArray(exp) ? "Balanced" : "Not Balanced") <<
std::endl;
    }

    // Linked List Stack Test
    std::cout << "\nUsing Linked List Stack:\n";
    for (const std::string &exp : expressions) {
        std::cout << exp << ": " << (checkBalancedLinkedList(exp) ? "Balanced" : "Not Balanced") <<
std::endl;
    }

    // STL Stack Test
    std::cout << "\nUsing STL Stack:\n";
    for (const std::string &exp : expressions) {
        std::cout << exp << ": " << (checkBalancedSTL(exp) ? "Balanced" : "Not Balanced") <<
std::endl;
    }

    return 0;
}
```

OUTPUT	

```
Using Array Stack:
(A+B)+(C-D): Balanced
((A+B)+(C-D): Not Balanced
((A+B)+[C-D]): Balanced
((A+B]+[C-D]}: Not Balanced

Using Linked List Stack:
(A+B)+(C-D): Balanced
((A+B)+(C-D): Not Balanced
((A+B)+[C-D]): Balanced
((A+B]+[C-D]}: Not Balanced

Using STL Stack:
(A+B)+(C-D): Balanced
((A+B)+(C-D): Not Balanced
((A+B)+[C-D]): Balanced
((A+B]+[C-D]}: Not Balanced
```

Expression	Valid? (Y/N)	Output (Console Screenshot)	Analysis
(A+B)+(C-D)	**Yes**	Balanced	Correctly matched opening and closing symbols.
((A+B)+(C-D)	**No**	Not Balanced	A closing parenthesis is missing.
((A+B)+[C-D])	**Yes**	Balanced	Both parentheses and brackets match properly.
((A+B]+[C-D]}	**No**	Not Balanced	There's a mismatch between] and }

8. Conclusion

This activity highlighted the use of different stack implementations—array-based, linked list, and C++ STL—to check for balanced symbols in expressions. The array stack offered simplicity but lacked flexibility due to its fixed size, while the linked list stack dynamically managed memory. The STL stack demonstrated the convenience of using built-in libraries.

Overall, we learned how the choice of stack implementation can impact performance and memory usage, emphasizing the importance of selecting the right data structure for specific programming challenges.

9. Assessment Rubric