Activity No. 4.1	
Course Code: CPE010	**Program:** Computer Engineering
Course Title: Data Structures and Algorithms	**Date Performed: 8/26/2025**
Section: CPE21S4	**Date Submitted: 8/26/2025**
Name(s): Anastacio, Lester Arvid P.	**Instructor: Engr. Jimlord M. Quejado**

6. Output

A. Create a stack using the C++ STL

main.cpp — Run — Output

```
1  //Tests the push, empty, size, pop, and top methods
       of the stack library.
2  #include <iostream>
3  #include <stack> // Calling Stack from the STL
4  using namespace std;
5  int main() {
6  stack<int> newStack;
7  newStack.push(3); //Adds 3 to the stack
8  newStack.push(8);
9  newStack.push(15);
10 // returns a boolean response depending on if the
       stack is empty or not
11 cout << "Stack Empty? " << newStack.empty() << endl
       ;
12 // returns the size of the stack itself
13 cout << "Stack Size: " << newStack.size() << endl;
14 // returns the topmost element of the stack
15 cout << "Top Element of the Stack: " << newStack
       .top() << endl;
16 // removes the topmost element of the stack
17 newStack.pop();
18 cout << "Top Element of the Stack: " << newStack
       .top() << endl;
19 cout << "Stack Size: " << newStack.size() << endl;
20 return 0;
21 }
22
```

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


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

B.1. Stacks using Arrays

Original:

```
#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();

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" << std::endl;
        std::cin >> choice;

        switch(choice){
            case 1: push();
                    break;
            case 2: pop();
                    break;
            case 3: Top();
                    break;
            case 4: std::cout << isEmpty() << std::endl;
                    break;
            default: std::cout << "Invalid Choice." << std::endl;
                     break;
        }
    }


    return 0;

}

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

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];
    //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;
}
```

Modified:

main.cpp

```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 displayStack();  // New function prototype

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

    while(true){
        std::cout << "\nStack 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;   // call to new function
            default: std::cout << "Invalid Choice." << std::endl; break;
```

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: 200

Stack Operations:
1. PUSH, 2. POP, 3. TOP, 4. isEMPTY, 5. DISPLAY STACK
1
New Value: 300

Stack Operations:
1. PUSH, 2. POP, 3. TOP, 4. isEMPTY, 5. DISPLAY STACK
1
New Value: 300

Stack Operations:
1. PUSH, 2. POP, 3. TOP, 4. isEMPTY, 5. DISPLAY STACK
5
Stack elements are: 200 300 300

Stack Operations:
1. PUSH, 2. POP, 3. TOP, 4. isEMPTY, 5. DISPLAY STACK
```

This is what's added:

```cpp
}

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

B.2. Stacks using Linked Lists

ORIGINAL:

```
#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;

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

int pop(){
    int tempVal;
    Node *temp;
    if(head == NULL){
        head = tail = 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;
        return;
    } else {
        std::cout << "Top of Stack: " << head->data << 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 :";
    Top();
    pop();

    return 0;
}
```

MODIFIED:

main.cpp

Share Run

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

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

```
        tempVal = temp->data;
        head = head->next;
        delete temp;
        return tempVal;
    }
}

void Top() {
    if (head == NULL) {
        std::cout << "Stack is Empty." << std::endl;
        return;
    } 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;
    }
    std::cout << "Stack elements (top to bottom): ";
    Node* current = head;
    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();
    displayStack();

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

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

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

    pop(); // extra pop to test underflow
    return 0;
}
```

Output

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

7. Supplementary Activity

SCREENSHOTS:

a. **Stack using Arrays**
   CODE:

main.cpp

Share

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

#define MAX 100

class StackArray {
    char arr[MAX];
    int top;

public:
    StackArray() { top = -1; }

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

    void push(char ch) {
        if (isFull()) {
            cout << "Stack overflow\n";
            return;
        }
        arr[++top] = ch;
    }

    char pop() {
        if (isEmpty()) {
            cout << "Stack underflow\n";
            return '\0';
        }
        return arr[top--];
    }

```

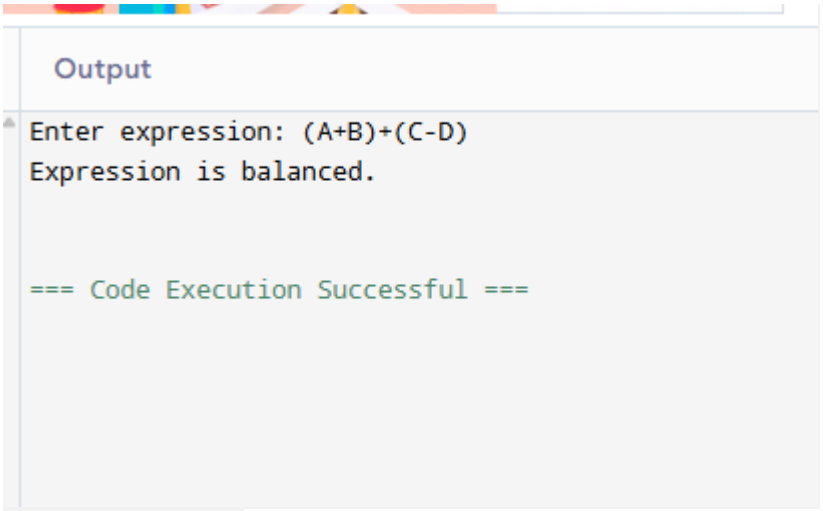
main.cpp

Share Run

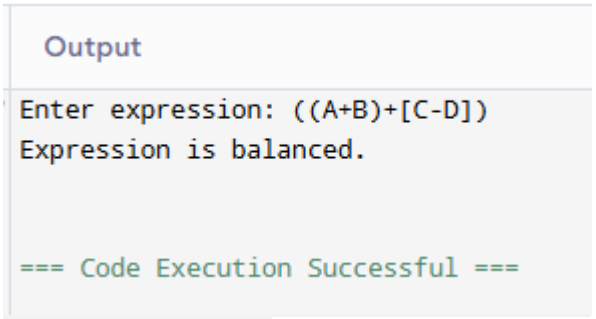
```
    char peek() {
        if (isEmpty()) return '\0';
        return arr[top];
    }
};

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

bool checkBalanced(string expr) {
    StackArray stack;
    for (char ch : expr) {
        if (ch == '(' || ch == '{' || ch == '[') {
            stack.push(ch);
        } else if (ch == ')' || ch == '}' || ch == ']') {
            if (stack.isEmpty()) {
                cout << "Error: Closing symbol '" << ch << "' found but
                    stack is empty\n";
                return false;
            }
            char open = stack.pop();
            if (!isMatchingPair(open, ch)) {
                cout << "Error: Mismatched pair '" << open << "' and '"
                    << ch << "'\n";
                return false;
            }
        }
    }
    if (!stack.isEmpty()) {
        cout << "Error: Stack not empty at end of input\n";
        return false;
    }
    return true;
}

int main() {
    string expr;
    cout << "Enter expression: ";
    getline(cin, expr);

    if (checkBalanced(expr)) {
        cout << "Expression is balanced.\n";
    } else {
        cout << "Expression is not balanced.\n";
    }
    return 0;
}
```

OUTPUT:

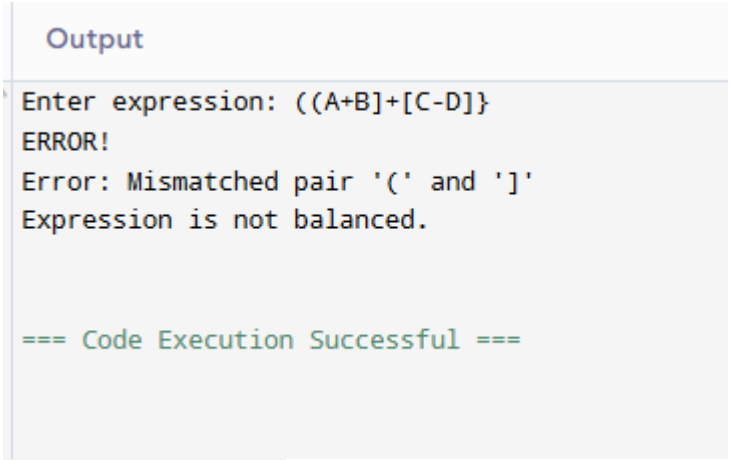
Expression 1: VALID



Expression 2: VALID



Expression 3: INVALID



b. Stack using Linked Lists

CODE:

main.cpp

Share

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

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

class StackLinkedList {
    Node* top;

public:
    StackLinkedList() { top = nullptr; }

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

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

    char pop() {
        if (isEmpty()) {
            cout << "Stack underflow\n";
            return '\0';
        }
        Node* temp = top;
        char val = top->data;
        top = top->next;
```

main.cpp

Share Run

```
31          top = top->next;
32          delete temp;
33          return val;
34      }
35
36      char peek() {
37          if (isEmpty()) return '\0';
38          return top->data;
39      }
40
41      ~StackLinkedList() {
42          while (!isEmpty()) {
43              pop();
44          }
45      }
46  };
47
48  bool isMatchingPair(char open, char close) {
49      return (open == '(' && close == ')') ||
50             (open == '{' && close == '}') ||
51             (open == '[' && close == ']');
52  }
53
54  bool checkBalanced(string expr) {
55      StackLinkedList stack;
56      for (char ch : expr) {
57          if (ch == '(' || ch == '{' || ch == '[') {
58              stack.push(ch);
59          } else if (ch == ')' || ch == '}' || ch == ']') {
60              if (stack.isEmpty()) {
61                  cout << "Error: Closing symbol '" << ch << "' found but
                        stack is empty\n";
```

main.cpp

Share Run

```
                    stack is empty\n";
62                  return false;
63              }
64              char open = stack.pop();
65              if (!isMatchingPair(open, ch)) {
66                  cout << "Error: Mismatched pair '" << open << "' and '"
                        << ch << "'\n";
67                  return false;
68              }
69          }
70      }
71
72      if (!stack.isEmpty()) {
73          cout << "Error: Stack not empty at end of input\n";
74          return false;
75      }
76      return true;
77  }
78
79  int main() {
80      string expr;
81      cout << "Enter expression: ";
82      getline(cin, expr);
83
84      if (checkBalanced(expr)) {
85          cout << "Expression is balanced.\n";
86      } else {
87          cout << "Expression is not balanced.\n";
88      }
89      return 0;
90  }
```

OUTPUT:

Expression 1: VALID

Output

```
Enter expression: (A+B)+(C-D)
Expression is balanced.


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

Expression 2: VALID

```
Output

Enter expression: ((A+B)+[C-D])
Expression is balanced.


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

Expression 3: INVALID

```
Output

Enter expression: ((A+B]+[C-D]}
ERROR!
Error: Mismatched pair '(' and ']'
Expression is not balanced.


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

TOOL ANALYSIS:

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

- When using arrays, it makes stacks simple and fast but fixed in size while linked lists allow dynamic sizing with more memory overhead and STL stacks provide easy, flexible, and safe usage with internal optimizations.

8. Conclusion

In this activity, I learned how different stack implementations work and their trade-offs, successfully applied the symbol balancing algorithm through careful stack operations, found the hands-on coding valuable for understandings memory management, and feel confident but recognize I can improve in optimizing code and handling complex cases.

9. Assessment Rubric