Activity No. 4.1	
Hands-on Activity 4.1 Stacks	
Course Code: CPE010	Program: Computer Engineering
Course Title: Data Structures and Algorithms	Date Performed: 10/4/24
Section: CPE21S4	Date Submitted: 10/6/24
Name(s): SANTOS, ANDREI R.	Instructor: Professor Maria Rizette Sayo
6. Output	

Table 4.1 Output of ILO A

```cpp
#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;
}
```

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


...Program finished with exit code 0
Press ENTER to exit console.
```

Table 4.1 Output of ILO B.1

```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();

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;
    }

```

```cpp
    }
}
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;
}

```

```
Enter number of max elements for new stack: 2
Stack Operations:
1. PUSH, 2. POP, 3. TOP, 4. isEMPTY
1
New Value:
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
3
The element on the top of the stack is 10
Stack Operations:
1. PUSH, 2. POP, 3. TOP, 4. isEMPTY
4
0
Stack Operations:
1. PUSH, 2. POP, 3. TOP, 4. isEMPTY
2
Popping: 10Stack Operations:
1. PUSH, 2. POP, 3. TOP, 4. isEMPTY
3
The element on the top of the stack is 5
Stack Operations:
1. PUSH, 2. POP, 3. TOP, 4. isEMPTY
4
0
Stack Operations:
1. PUSH, 2. POP, 3. TOP, 4. isEMPTY
```

Table 4.1 Output of ILO B.1
WITH ADDED CODE FROM THE TASKS

```
void Display(); // decalre another void
```

```
case 5: Display(); // adding this fro having to have another a case which is the display
    break;
```

```
83  void Display()
84  {
85      if(isEmpty()) // if statement, checking if the stack is empty
86      {
87          std::cout << "Stack is Empty" << std::endl; // if it is empty, this will be printed
88          return; // and will return after executing the code above
89      }
90      std::cout << "Stack Elements: ";
91      for(int j = 0; j <= top; j++) // using for loop, it will run thorugh all the elements in the stack
92          {
93              std::cout << stack[j] << " "; // now, it will print eadch stack, one by one with a space
94          std::cout << std::endl; // it will now move to the next time after printing all the elements
95          }
96  }
```

OUTPUT :

```
Enter number of max elements for new stack: 2
 Stack Operations:
1. PUSH, 2. POP, 3. TOP, 4. isEMPTY, 5. DISPLAY
1
New Value:
12
 Stack Operations:
1. PUSH, 2. POP, 3. TOP, 4. isEMPTY, 5. DISPLAY
1
New Value:
23
 Stack Operations:
1. PUSH, 2. POP, 3. TOP, 4. isEMPTY, 5. DISPLAY
3
The element on the top of the stack is 23
 Stack Operations:
1. PUSH, 2. POP, 3. TOP, 4. isEMPTY, 5. DISPLAY
4
0
 Stack Operations:
1. PUSH, 2. POP, 3. TOP, 4. isEMPTY, 5. DISPLAY
5
Stack Elements: 12
23
 Stack Operations:
1. PUSH, 2. POP, 3. TOP, 4. isEMPTY, 5. DISPLAY
2
Popping: 23 Stack Operations:
1. PUSH, 2. POP, 3. TOP, 4. isEMPTY, 5. DISPLAY
3
The element on the top of the stack is 12
 Stack Operations:
1. PUSH, 2. POP, 3. TOP, 4. isEMPTY, 5. DISPLAY
```

Table 4.1 Output of ILO B.2

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

```cpp
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;
}
```

```
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.
```

Table 4.1 Output of ILO B.2
WITH ADDED CODE FROM THE TASKS

```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;
        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;
    }
}

void display()
{
    if(head == NULL) // checking if the stack is empty or not
    {
        std::cout << "Now, the Stack is empty. " << std::endl;
        return; // after checking that the stack is empty, return.
    }
    Node* temp = head;
    std :: cout << "This is the stack elements: "; // if not empty, this will be printed and will show the output
    while(temp != NULL) // having a loop for each node that will pass thorugh
    {
        std::cout << temp -> data << " "; // each output is printed with space " "
        temp = temp -> next;
    }
    std::cout << std::endl; // end line after printing the elemnts in the stack
}

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();

    display();

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

    display();
    pop();
    std::cout<<"After the second POP operation, top of stack: ";
    Top();
    display();
    pop();
    return 0;
}
```

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


...Program finished with exit code 0
Press ENTER to exit console.
```

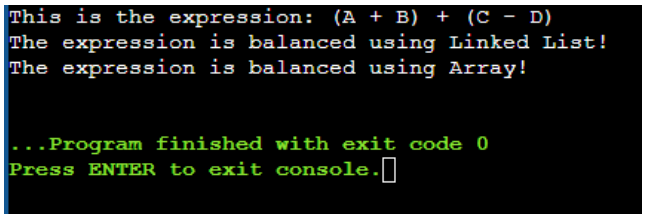
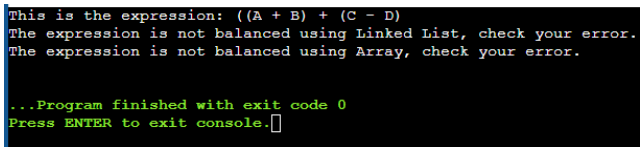
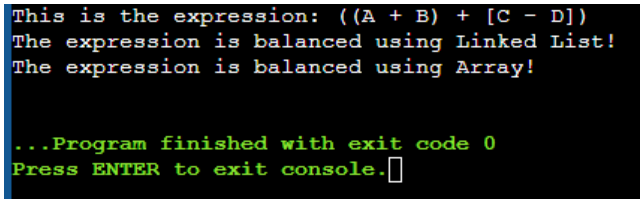
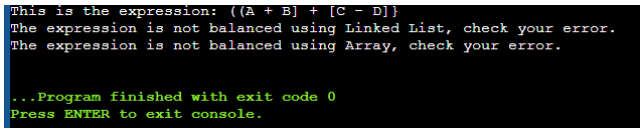
7. Supplementary Activity

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

// creating a class for node
class Node {
public:
    char data;
    Node* next;
};

// Linked List Class
class Lin_Lis_Stack {
    Node* top;
public:
    Lin_Lis_Stack() : top(nullptr) {}

    // Push character onto stack
    void push(char item) {
        Node* newNode = new Node{item, top}; // Create new node
        top = newNode; // Update the top as the new node
```

```
    }

    // Pop character from stack
    char pop() {
        if (!top) return '\0'; // Return null if stack is empty that is why '\0'
        Node* temp = top;
        char popped = temp->data;
        top = top->next;
        delete temp; // delete the temp
        return popped; // return the pop to avoid empty
    }

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

// Array Stack Class
class Arr_Stck {
    int top;
    int capacity;
    char* stackArray;
public:
    Arr_Stck(int size) : capacity(size), top(-1) {
        stackArray = new char[capacity]; // assign the memory
    }

    ~Arr_Stck() {
        delete[] stackArray; // now, delete the memory
    }

    void push(char item) {
        if (top < capacity - 1)
            stackArray[++top] = item;
    }

    char pop() {
        if (top >= 0) return stackArray[top--]; // Return item
        return '\0'; // Return null if empty
    }

    bool isEmpty() {
        return top == -1; // Return if empty
    }
};

// Function to check balanced symbols using Linked List Stack
bool Check_Expr_Lin_Lis(string expr) {
    Lin_Lis_Stack stack;
    for (char ch : expr) {
```

```
        if (ch == '(' || ch == '{' || ch == '[') stack.push(ch); // Push opening symbols
        else if (ch == ')' || ch == '}' || ch == ']') {
            if (stack.isEmpty() ||
                (ch == ')' && stack.pop() != '(') ||
                (ch == '}' && stack.pop() != '{') ||
                (ch == ']' && stack.pop() != '['))
                return false;
        }
    }
    return stack.isEmpty(); // Return if it is balanced
}

// Function bool to check balanced symbols using Array Stack
bool Check_Expr_Array(string expr) {
    Arr_Stck stack(expr.length()); // Create stack based on expression length
    for (char ch : expr) {
        if (ch == '(' || ch == '{' || ch == '[') stack.push(ch); // Push symbols
        else if (ch == ')' || ch == '}' || ch == ']') {
            if (stack.isEmpty() ||
                (ch == ')' && stack.pop() != '(') ||
                (ch == '}' && stack.pop() != '{') ||
                (ch == ']' && stack.pop() != '['))
                return false;
        }
    }
    return stack.isEmpty(); // Return if balanced
}

int main() {
    string expr = "(A+B)+(C-D)"; // This will be the example expression that will be checked
    cout << "This is the expression: " << expr << endl;

    // Check expression for balanced symbols using Linked List
    if (Check_Expr_Lin_Lis(expr)) {
        cout << "The expression is balanced using Linked List!" << endl; // This will be printed if the expr is balanced with
the use of linked list
    } else {
        cout << "The expression is not balanced using Linked List, check your error." << endl; // This will be printed if the
expr is not balanced with the use of linked list
    }

    // Check expression for balanced symbols using Array
    if (Check_Expr_Array(expr)) {
        cout << "The expression is balanced using Array!" << endl; // This will be printed if the expr is balanced with the use
of array
    } else {
        cout << "The expression is not balanced using Array, check your error." << endl; // This will be printed if the expr is
not balanced with the use of array
    }

    return 0;
```

}

Expression	Valid? (Y/ N)	Output (Console Screenshot)	Analysis
(A + B) + (C - D)	Y		The expression is balanced since the symbols such as parenthesis are equal, since each symbol (has the corresponding symbol of) . Thus it is Y valid.
((A + B) + (C - D)	N		The expression is not balanced since there are already two opening symbol ((but only one closing symbol) . Thus it is N valid.
((A + B) + [C - D])	Y		The expression is balanced since the symbols are matched to its corresponding given such as the () and the []. Thus it is a Y valid.
((A + B] + [C - D]}	N		The expression is not balanced since the symbols are mismatched like (to] and (to } which are not matched. Thus it is N valid.

Tools Analysis:

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

  The internal representation of stacks affects their implementation and usage. Arrays have a fixed size, limiting their storage and requiring resizing for additional data, but offer fast access due to direct indexing. Linked lists are dynamic and allow flexible growth, though accessing elements is slower due to pointer management and has a memory for addressing head and nodes. The STL stack in C++ provides dynamic sizing and good performance for both size and time, with easy implementation and high-level functionality, which shows flexibility and versatility when it comes to coding.

8. Conclusion

In conclusion, I now understand how to implement displays in two different ILOs. I learned that for expressions to be balanced, each symbol must have a matching one. If not, the expression is unbalanced. This activity helped me learn

more about stacks and linked lists and how complex they can be in coding. At first, the procedures were tough to grasp, but over time I got the hang of the logic. Now, I will focus on the logic and structure to better understand future lessons.
9. Assessment Rubric