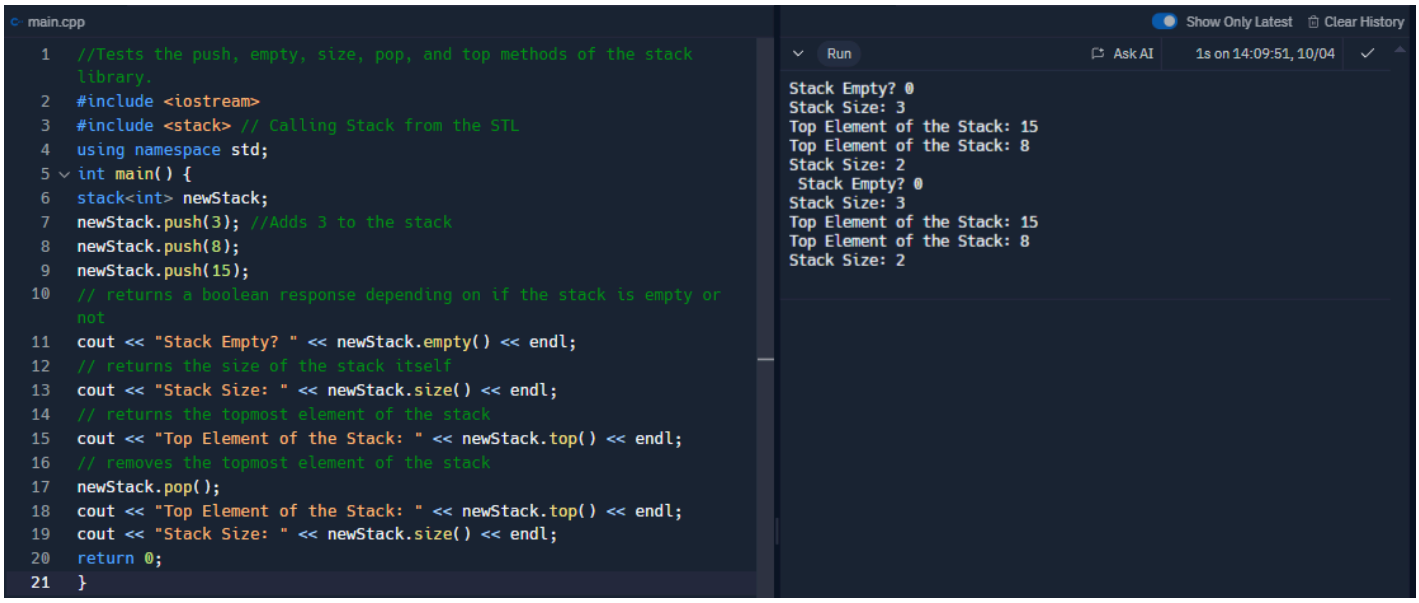
Activity No. 4	
STACKS	
Course Code: CPE010	**Program:** Computer Engineering
Course Title: Data Structures and Algorithms	**Date Performed:** 04/10/24
Section: CPE21S4	**Date Submitted:** 04/10/24
Name(s): Prince Wally G. Esteban	**Instructor:** Mrs. Marizette Sayo
6. Output	

TABLE 4-1



This C++ program tests basic stack operations using the Standard Template Library (STL) `stack` container. Here's a brief observation of each operation:

Push Operation: Three integers (3, 8, 15) are pushed onto the stack.

Empty Check: The program checks if the stack is empty using `empty()` (returns `false` since elements were pushed).

Size of the Stack: The stack size is printed using `size()`, showing there are 3 elements.

Top Element: The topmost element (15) is accessed using `top()`.

Pop Operation: The topmost element (15) is removed using `pop()`, and the next top element (8) is displayed.

Final Size: The stack size after popping an element is printed (now 2 elements).

This demonstrates basic stack functionality: LIFO (Last In, First Out) behavi**or.**

TABLE 4-2

```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 display();
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() << std::endl;
break;
case 5: display();
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;
}
void display(){
if(isEmpty()) {
std::cout << "Stack is Empty." << std::endl;
return;
}
std::cout << "Stack Elements: " << std::endl;
for(int j=0; j<=top; j++){
std::cout << stack[j] << " ";
}
std::cout << std::endl;
}
```

```
Enter number of max elements for new stack: 3
Stack Operations:
1. PUSH, 2. POP, 3. TOP, 4. isEMPTY, 5. DISPLAY
4
1
Stack Operations:
1. PUSH, 2. POP, 3. TOP, 4. isEMPTY, 5. DISPLAY
1
New Value:
20
Stack Operations:
1. PUSH, 2. POP, 3. TOP, 4. isEMPTY, 5. DISPLAY
1
New Value:
30
Stack Operations:
1. PUSH, 2. POP, 3. TOP, 4. isEMPTY, 5. DISPLAY
1
New Value:
40
Stack Operations:
1. PUSH, 2. POP, 3. TOP, 4. isEMPTY, 5. DISPLAY
5
Stack Elements:
20 30 40
Stack Operations:
1. PUSH, 2. POP, 3. TOP, 4. isEMPTY, 5. DISPLAY
```

`push()`: Inserts a new element at the top of the stack. If the stack is full (stack overflow), it prints an error message.
`pop()`: Removes the top element from the stack. If the stack is empty (stack underflow), it prints an error message.
`Top()`: Returns the element at the top of the stack. If the stack is empty, it prints an error message.
`isEmpty()`: Checks if the stack is empty. Returns `true` if empty, `false` otherwise.
`display()`: Prints all the elements in the stack from the bottom to the top. If the stack is empty, it prints an error message.

TABLE 4-3

```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;
}
}
```

```
Run  Ask AI  755ms on 14:17:27, 10/0

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

main.cpp

```cpp
if(head==NULL){
std::cout << "Stack is Empty." << std::endl;
return;
}
Node *temp = head;
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 << "Stack elements after two pushes: ";
display();
pop();
std::cout<<"After the first POP operation, top of stack is:";
Top();
std::cout << "Stack elements after the first pop: ";
display();
pop();
std::cout<<"After the second POP operation, top of stack :";
Top();
std::cout << "Stack elements after the second pop: ";
display();
pop();
return 0;
}
```

Show Only Latest Clear History

```
Run  Ask AI  755ms on 14:17:27, 10/0

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

In the original code, I had the stack operations (`push`, `pop`, and `Top`) defined outside of any class. To improve organization, I realized I should encapsulate these methods within a class, like a `Stack` class, which would manage the stack's behavior.

By moving these methods inside the `Stack` class, I was able to make the structure of my code cleaner and more modular. The `push` method now inserts elements at the top of the stack, the `pop` method removes the top element while handling cases of underflow, and the `Top` method displays the top element of the stack. This change helped me keep the stack's data and functionality well-organized and easier to manage.

7. Supplementary Activity

main.cpp

```cpp
#include <iostream>
#include <string>

using namespace std;

const int MAX_SIZE = 100;

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

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

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

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

    void push(char data) {
        if (isFull()) {
            cout << "Stack Overflow!" << endl;
            return;
        }
        arr[++top] = data;
    }

    char pop() {
        if (isEmpty()) {
            cout << "Stack Underflow!" << endl;
            return -1;
        }
        return arr[top--];
    }

    char peek() {
        if (isEmpty()) {
            cout << "Stack is Empty!" << endl;
            return -1;
        }
        return arr[top];
    }
```

Show Only Latest | Clear History

Run | Ask AI | 28s on 14:24:15, 10/04

```
Enter an expression: (A+B)+(C-D)
Expression has balanced symbols.
```

main.cpp

```cpp
};

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

bool checkBalancedSymbols(string expression) {
    Stack stack;
    for (int i = 0; i < expression.length(); i++) {
        char ch = expression[i];
        if (ch == '(' || ch == '{' || ch == '[') {
            stack.push(ch);
        } else if (ch == ')' || ch == '}' || ch == ']') {
            if (stack.isEmpty()) {
                cout << "Error: Unmatched closing symbol: " << ch << endl;
                return false;
            }
            char topChar = stack.pop();
            if (!isMatchingPair(topChar, ch)) {
                cout << "Error: Mismatched symbols: " << topChar << " and " << ch << endl;
                return false;
            }
        }
    }
    if (!stack.isEmpty()) {
        cout << "Error: Unmatched opening symbols remain." << endl;
        return false;
    }
    return true;
}

int main() {
    string expression;
    cout << "Enter an expression: ";
    getline(cin, expression); // Use getline for input with spaces

    if (checkBalancedSymbols(expression)) {
        cout << "Expression has balanced symbols." << endl;
    } else {
        cout << "Expression has unbalanced symbols." << endl;
    }
    return 0;
}
```

Show Only Latest | Clear History

Run | Ask AI | 8s on 14:26:43, 10/04

```
Enter an expression: (A+B
Error: Unmatched opening symbols remain.
Expression has unbalanced symbols.
```

EXPRESSION

1. (A+B)+(C-D) VALID
2. ((A+B)+(C-D) INVALID
3. ((A+B)+[C-D]) VALID
4. ((A+B]+[C-D]} INVALID

8. Conclusion
In conclusion, by encapsulating the stack operations within a dedicated `Stack` class, I significantly improved the organization and clarity of my code. This structure not only enhances readability but also ensures that the stack's behavior and data are managed cohesively. The separation of concerns makes it easier to maintain and extend the functionality of the stack in the future. Overall, this approach fosters better programming practices and helps create more robust and efficient code.
9. Assessment Rubric