

Laboratory Activity # 4	
Stacks	
Course Code: CPE010	Program: BSCPE
Course Title: Data Structures and Algorithms	Date Performed: 10-04-2024
Section: CPE21S4	Date Submitted: 10-06-2024
Name(s): Masangkay, Frederick D.	Instructor: Ma'am Maria Rizette Sayo

6. Output

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

Table 4 - 1. Output of ILO A

```
void printAll(){
    std::cout << "This is the element inside the stack: " << std::endl;
    while (!isEmpty())
    {
        std::cout << stack[top] << std::endl;
        top--;
    }
}
```

```
}
```

```
/tmp/k5yH1dKVHt.o
Enter number of max elements for new stack: 3
Stack Operations:
1. PUSH, 2. POP, 3. TOP, 4. isEmpty, 5. printAll
1
New Value:
21
Stack Operations:
1. PUSH, 2. POP, 3. TOP, 4. isEmpty, 5. printAll
1
New Value:
42
Stack Operations:
1. PUSH, 2. POP, 3. TOP, 4. isEmpty, 5. printAll
1
New Value:
23
Stack Operations:
1. PUSH, 2. POP, 3. TOP, 4. isEmpty, 5. printAll
5
This is the element inside the stack:
23
42
21
```

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

```
void printAll(Node* head) {
    while (head != nullptr) {
        std::cout<< head->data << " ";
        head = head->next;
    }
    std::cout << std::endl;
```

```

/tmp/HwTgc8ZV8t.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

Printing all the element:5 1

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 ===
}

```

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

7. Supplementary Activity

a. Stack using Arrays

```

#include <iostream>
#include <cstring>

#define MAX 100

class ArrayStack {
private:
    char arr[MAX];
    int top;

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

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

    char pop() {
        if (top < 0) {
            return '\0'; // Stack Underflow
        }
    }
}

```

```

    }
    return arr[top--];
}

bool isEmpty() {
    return top < 0;
}
};

bool isBalanced(const std::string &expr) {
    ArrayStack stack;
    for (char c : expr) {
        if (c == '(' || c == '{' || c == '[') {
            stack.push(c);
        } else if (c == ')' || c == '}' || c == ']') {
            if (stack.isEmpty()) return false;
            char topChar = stack.pop();
            if ((c == ')' && topChar != '(') ||
                (c == '}' && topChar != '{') ||
                (c == ']' && topChar != '[')) {
                return false;
            }
        }
    }
    return stack.isEmpty();
}

int main() {
    std::string expressions[] = {"(A+B)+(C-D)", "((A+B)+(C-D)", "((A+B)+[C-D])", "((A+B)+[C-D])"};
    for (const auto &expr : expressions) {
        std::cout << expr << " is " << (isBalanced(expr) ? "Valid" : "Invalid") << std::endl;
    }
    return 0;
}

```

b. Stack using Linked Lists

```

#include <iostream>
#include <string>

```

```

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

```

```

class LinkedListStack {

```

```

private:
    Node *top;

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

    void push(char c) {
        Node *newNode = new Node{c, top};
        top = newNode;
    }

    char pop() {
        if (top == nullptr) return '\0'; // Stack Underflow
        char poppedValue = top->data;
        Node *temp = top;
        top = top->next;
        delete temp;
        return poppedValue;
    }

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

bool isBalancedLL(const std::string &expr) {
    LinkedListStack stack;
    for (char c : expr) {
        // Step 2a: Ignore non-symbol characters
        if (c == '(' || c == '{' || c == '[') {
            // Step 2b: Push opening symbols onto the stack
            stack.push(c);
        } else if (c == ')' || c == '}' || c == ']') {
            // Step 2c: Handle closing symbols
            if (stack.isEmpty()) {
                std::cout << "Error: Unmatched closing symbol '" << c << "'\n";
                return false;
            }
            char topChar = stack.pop();
            // Step 2d: Check for matching symbols
            if ((c == ')' && topChar != '(') ||
                (c == '}' && topChar != '{') ||
                (c == ']' && topChar != '[')) {
                std::cout << "Error: Mismatched symbols '" << topChar << "' and '" << c << "'\n";
                return false;
            }
        }
    }
}

```

```

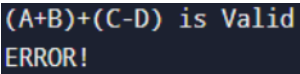
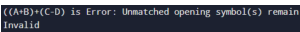
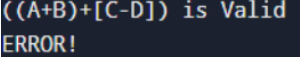
    }
}
// Step 3: Check if the stack is empty at the end
if (!stack.isEmpty()) {
    std::cout << "Error: Unmatched opening symbol(s) remain\n";
    return false;
}
return true;
}

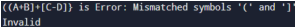
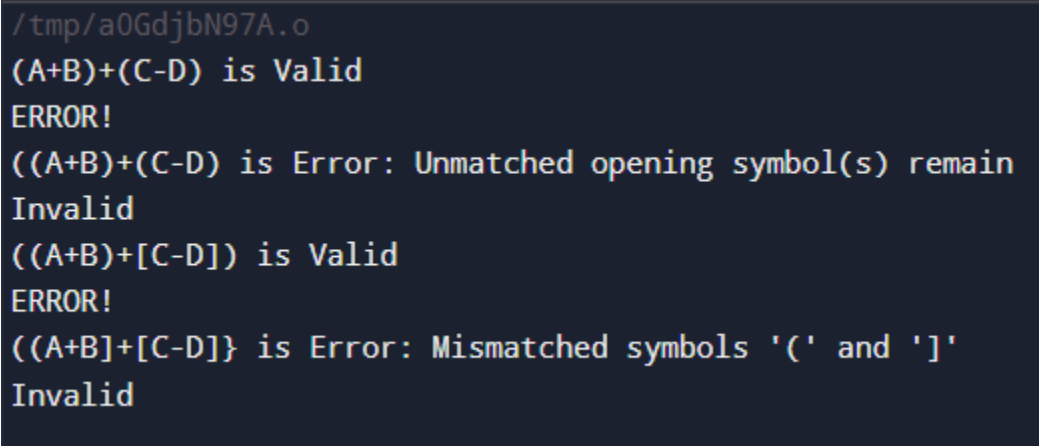
int main() {
    std::string expressions[] = {
        "(A+B)+(C-D)",
        "((A+B)+(C-D)",
        "((A+B)+[C-D])",
        "((A+B)+[C-D])"
    };

    for (const auto &expr : expressions) {
        std::cout << expr << " is " << (isBalancedLL(expr) ? "Valid" : "Invalid") << std::endl;
    }

    return 0;
}

```

Expression	Valid? (Y/N)	Output (Console Screenshot)	Analysis
(A+B)+(C-D)	Y		All opening symbols have matching closing symbols. The stack is empty at the end, indicating balance.
((A+B)+(C-D)	N		There is an unmatched opening symbol (. The stack is not empty at the end, indicating an error.
((A+B)+[C-D])	Y		All symbols are matched correctly. The mixed use of () and [] is valid, and the stack is empty at the end.

$((A+B)+[C-D])$	N	 ((A+B)+[C-D]) is Error: Mismatched symbols '(' and '[' Invalid	There are mismatched symbols: [is closed by], but (is closed by }. This results in an error on mismatch.
 <pre>/tmp/a0GdjbN97A.o (A+B)+(C-D) is Valid ERROR! ((A+B)+(C-D) is Error: Unmatched opening symbol(s) remain Invalid ((A+B)+[C-D]) is Valid ERROR! ((A+B)+[C-D]) is Error: Mismatched symbols '(' and '[' Invalid</pre>			

8. Conclusion

The activity of implementing a stack-based algorithm to check for balanced symbols highlighted the effectiveness of stack data structures in parsing expressions. By developing three types of stacks—array-based, linked list-based, and using C++ STL—we explored their respective advantages and limitations, such as the fixed size of arrays versus the dynamic nature of linked lists. The focus on systematic character processing demonstrated how stacks manage opening and closing symbols to validate expressions, with the crucial final check for an empty stack identifying unmatched opening symbols. This exercise reinforced fundamental concepts of data structures and algorithms, underscoring their importance in software development and illustrating how compilers ensure syntactical correctness in code, making the practice of checking balanced symbols a foundational skill in computer science.

9. Assessment Rubric