Activity No. 4	
STACKS	
Course Code: CPE010	**Program:** Computer Engineering
Course Title: Data Structures and Algorithms	**Date Performed:** 10/04/24
Section: CPE21S4	**Date Submitted:** 10/04/24
Name(s): Magistrado, Aira Pauleen M.	**Instructor:** Maria Rizette Sayo

6. Output

```
C/C++
//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); //Adds 8 to the stack
newStack.push(15); //Adds 15 to the stack

// 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
/tmp/B2mQoTHqph.o
Stack Empty? 0
Stack Size: 3
Top Element of the Stack: 15
Top Element of the Stack: 8
Stack Size: 2
```

Table 4-1. Output of ILO A

C/C++

```
#include <iostream>
const size_t maxCap = 100;
int stack[maxCap]; // stack with max of 100 elements
int top = -1, i, newData;

void push(); //it will add a new element to the top of the stack
void pop(); // it will remove an the element from the top of the stack
void Top(); // it will return the value on the top without removing it
bool isEmpty(); //it will check if the stack is empty
void stackDisplay(); //displays the elements in the stack

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 ELEMENTS" <<
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: stackDisplay();
                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;
}

void stackDisplay() {
    if (isEmpty()) {
        std::cout << "Stack is Empty." << std::endl;
        return;
    }

    std::cout << "Stack elements: ";
    for (int j = top; j >= 0; j--) {
        std::cout << stack[j] << " ";
    }
    std::cout << std::endl;
}
```

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

```
C/C++
#include<iostream>
class Node{ //node = linked list node
public:
    int data;
    Node *next;
```

```
};

Node *head = NULL, *tail = NULL;

void push(int newData){ //it will add a new element to the top of the stack
    Node *newNode = new Node;
    newNode->data = newData;
    newNode->next = head;
    if(head == NULL){
        head = tail = newNode;
    } else {
        newNode->next = head;
        head = newNode;
    }
}

int pop(){ // it will remove an the element from the top of the stack
    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(){ //it will return the value on the top without removing it
    if(head == NULL){
        std::cout << "Stack is Empty." << std::endl;
        return;
    } else {
        std::cout << "Top of Stack: " << head->data << std::endl;
    }
}

// print stack elements from top to bottom
void PrintStack(Node* node) {
    if (node == NULL) {
        return;
    }

    // it will output the current top element
    std::cout << node->data << " ";

    PrintStack(node->next);
}

int main(){
    push(1);
```

```
    std::cout << "After first PUSH, top of stack is: ";
    Top();

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

    push(10);
    std::cout << "After third PUSH, top of stack is: ";
    Top();

    std::cout << "Current stack is: ";
    PrintStack(head); // Printing all elements in the stack
    std::cout << std::endl;

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

    std::cout << "Current stack is: ";
    PrintStack(head); // Printing all elements in the stack
    std::cout << std::endl;

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

    std::cout << "Current stack is: ";
    PrintStack(head); // Printing all elements in the stack
    std::cout << std::endl;

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

    return 0;
}
```

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

7. Supplementary Activity

C/C++

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

const size_t maxCap = 100;
char stack[maxCap]; // Stack with max of 100 elements
int top = -1;

bool isEmpty() {
```

```
    return top == -1;
}

void push(char c) {
    if (top == maxCap - 1) {
        std::cout << "Stack Overflow." << std::endl;
        return;
    }
    stack[++top] = c;
}

char pop() {
    if (isEmpty()) {
        std::cout << "Stack Underflow." << std::endl;
        return '\0';
    }
    return stack[top--];
}

char Top() {
    return isEmpty() ? '\0' : stack[top];
}

// it will check balanced symbols
bool checkBalance(const std::string &expr) {
    for (char c : expr) {
        if (c == '(' || c == '{' || c == '[') {
            push(c);
        } else if (c == ')' || c == '}' || c == ']') {
            if (isEmpty()) {
                return false;
            }
            char topSymbol = pop();
            if ((c == ')' && topSymbol != '(') ||
                (c == '}' && topSymbol != '{') ||
                (c == ']' && topSymbol != '[')) {
                return false;             }
        }
    }
    return isEmpty();
}

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

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

    return 0;
```

```
}
```

C/C++

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

class Node {
public:
    char data;
    Node* next;
};

Node* head = nullptr;

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

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

char pop() {
    if (isEmpty()) {
        std::cout << "Stack Underflow." << std::endl;
        return '\0';
    }
    char data = head->data;
    Node* temp = head;
    head = head->next;
    delete temp;
    return data;
}

// it will check balanced symbols
bool checkBalance(const std::string& expr) {
    for (char c : expr) {
        if (c == '(' || c == '{' || c == '[') {
            push(c);
        } else if (c == ')' || c == '}' || c == ']') {
            if (isEmpty()) {
                return false;
            }
            char topSymbol = pop();
            if ((c == ')' && topSymbol != '(') ||
                (c == '}' && topSymbol != '{') ||
                (c == ']' && topSymbol != '[')) {
                return false;
            }
```

```
            }
        }
        return isEmpty();
    }

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

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

        return 0;
    }
```

Tools Analysis:

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

If you use an array you set aside a fixed amount of memory from the start no matter how many items are added to the stack which means that it can run out of space making it limited but arrays use simple implementations when compared with linked lists memory is used only when needed and elements can be pushed or popped making them flexible but more complex due to the nodes and pointers.

Self-Checking:

Expression	Valid? (Y/N)	Output (Console Screenshot)	Analysis
(A+B)+(C-D)	Y	Expression: (A+B)+(C-D) is Valid	The Expression is valid because the parentheses are balanced correctly
((A+B)+(C-D)	N	Expression: ((A+B)+(C-D) is Invalid	The Expression is invalid because there is an extra opening parenthesis
((A+B)+[C-D])	Y	Expression: ((A+B)+[C-D]) is Valid	The Expression is valid because the parentheses are balanced correctly
((A+B]+[C-D]}	N	Expression: ((A+B]+[C-D]} is Invalid	The Expression is invalid because there is a mismatch

			in the brackets.

8. Conclusion

In conclusion, I have learned how stack arrays and linked lists are implemented in this lab activity. I've also learned about push which will add a new element to the top of the stack, pop which will remove an element from the top of the stack, Top which will return the value on the top without removing it and isEmpty which will check if the stack is empty. Implementing STL in the code is also efficient and quick. The supplementary activity checks the balance of the symbols. Both arrays and linked lists were used in the supplementary activity along with STL. This activity has enhanced my knowledge of arrays and linked lists I think I did pretty well but I still have a lot of areas for improvement.

9. Assessment Rubric