Date : 23.02.2025
Lakshana Baskaran
24011103026

LAB - 6
Stack ADT - Array & Singly Linked List

QUESTION 1:

A. Write a separate C++ menu-driven program to implement stack ADT using a character array of size 5. Maintain proper boundary conditions and follow good coding practices. Stack ADT has the following operations:

1. Push
2. Pop
3. Peek
4. Exit

SOURCE CODE:

```
//Implementation of stack ADT using array
#include <stdio.h>
#include <iostream>
using namespace std;
#define SIZE 5

class list {
    private:
        int arr[SIZE];
        int curr = -1;

    public:
        void push(int*, int);
        int pop(int*);
        int peek(int*);

        bool empty();
        void print(int*);

    int getCurr() const {
        return curr;
    }
};
```

Date : 23.02.2025
Lakshana Baskaran
24011103026

```
//appends to the stack
void list::push(int arr[SIZE], int x) {
    if (curr < SIZE-1) {
        arr[curr+1] = x;
        curr++;
        return;
    }
    cout << "OverFlowError: Stack is full\n";
    return;
}

//deletes the top
int list::pop(int arr[SIZE]) {
    if (empty()){
        printf("UnderflowError: Stack is empty.\n");
        return 0;
    }
    int key = arr[curr];

    curr = curr - 1;

    return key;
}

//returns the top
int list::peek(int arr[SIZE]) {
    return arr[curr];
}


//Prints the stack elements without modifying them in the terminal
void list::print(int A[SIZE]) {
    for (int i = 0; i <= curr; i++) {
        printf("%d ", A[i]);
    }
    cout << endl;
}

#include <stdbool.h>
bool list::empty() {
    return curr == -1;
}
```

Date : 23.02.2025
Lakshana Baskaran
24011103026

```
int main() {
    int arr[SIZE] = {0};
    list stk;

    int x, pos, choice = 0;
    printf("MENU\n1 - PUSH\n2 - POP\n3 - PEEK\n4 - Exit\n");

    while (choice != 4) {
        printf("\nEnter your choice: ");
        scanf("%d", &choice);
        switch (choice) {
            case 1:
                printf("Enter Element to be pushed: ");
                scanf("%d", &x);
                stk.push(arr, x);
                break;

            case 2:
                cout << stk.pop(arr) << endl;
                break;

            case 3:
                cout << "TOP = " << stk.peek(arr) << endl;
                break;

            case 4:
                printf("Exiting...\n");
                break;

            default:
                printf("Invalid choice. Enter again.\n");
                break;
        }
        printf("\n\tThe stack : ");
        stk.print(arr);
    }
}
```

Date : 23.02.2025
Lakshana Baskaran
24011103026

OUTPUT:

```
lemon@jupiter:~/workspace/college/DSA/Lab-6$ g++ -o out stack_array.cpp
lemon@jupiter:~/workspace/college/DSA/Lab-6$ ./out
MENU
1 - PUSH
2 - POP
3 - PEEK
4 - Exit

Enter your choice: 1
Enter Element to be pushed: 2

        The stack : 2

Enter your choice: 3
TOP = 2

        The stack : 2

Enter your choice: 2
2

        The stack :

Enter your choice: 2
UnderflowError: Stack is empty.
0

        The stack :

Enter your choice: 4
Exiting...
```

Date : 23.02.2025
Lakshana Baskaran
24011103026

QUESTION 2:

Write a separate C++ menu-driven program to implement stack ADT using a character singly linked list. Maintain proper boundary conditions and follow good coding practices. Stack ADT has the following operations:

1. Push
2. Pop
3. Peek
4. Exit

SOURCE CODE:

```
//Implementation of stack ADT using singly linked list
#include <stdio.h>
#include <iostream>
#include <stdlib.h>
#include <stdbool.h>
using namespace std;

class node {
    private:
        int data;
        struct node *next;
    public:
        void push(int);
        int pop();
        int peek();

        void print();

} *head = NULL;

//appends element to the stack
void node::push(int x) {
    struct node* newnode = (struct node*) malloc (sizeof(struct
node));
    newnode -> data = x;
    newnode -> next = NULL;

    if (head == NULL) {
        head = newnode;
        return;
    }
```

```
    struct node* temp = head;
    for (; temp -> next != NULL; temp = temp -> next) {
    }
    temp -> next = newnode;

}

//deletes the top and returns it
int node::pop() {
    if (head == NULL) {
        printf("UnderFlow Error: Stack is Empty.\n");
        return 0;
    }

    int elem;
    if (head -> next == NULL) {
        elem = head -> data;
        head = NULL;
        return elem;
    }

    struct node* temp = head;
    for (;temp -> next -> next != NULL; temp = temp -> next) {

    }

    elem = temp -> next -> data;
    temp -> next = NULL;

    return elem;
}

//returns the top
int node::peek() {
    struct node* temp = head;
    for (; temp -> next != NULL; temp = temp -> next) {
    }

    return temp -> data;
}

//prints without modifying the stack in the terminal
```

Date : 23.02.2025
Lakshana Baskaran
24011103026

```
void node::print() {
    struct node *temp = head;
    if (temp == NULL) {
        printf("head = NULL\n");
        return;
    }

    //printf("head -> ");
    for (; temp -> next != NULL; temp = temp -> next) {
        printf("%d -> ", temp -> data);
    }
    printf("%d -> NULL\n", temp -> data);
}

int main() {
    node stk;
    int x, pos, choice = 0;
    printf("MENU\n1 - PUSH\n2 - POP\n3 - PEEK\n4 - Exit\n");

    while (choice != 4) {
        printf("\nEnter your choice: ");
        scanf("%d", &choice);
        switch (choice) {
            case 1:
                printf("Enter Element to be pushed: ");
                scanf("%d", &x);
                stk.push(x);
                break;

            case 2:
                cout << stk.pop() << endl;
                break;

            case 3:
                cout << "TOP = " << stk.peek() << endl;
                break;

            case 4:
                printf("Exiting...\n");
                break;

            default:
                printf("Invalid choice. Enter again.\n");
```

Date : 23.02.2025
Lakshana Baskaran
24011103026

```
                break;
            }
            printf("\n\tThe stack : ");
            stk.print();
        }
}
```

SOURCE CODE:

```
lemon@jupiter:~/workspace/college/DSA/Lab-6$ g++ -o out stack_sll.cpp
lemon@jupiter:~/workspace/college/DSA/Lab-6$ ./out
MENU
1 - PUSH
2 - POP
3 - PEEK
4 - Exit

Enter your choice: 1
Enter Element to be pushed: 2

        The stack : 2 -> NULL

Enter your choice: 3
TOP = 2

        The stack : 2 -> NULL

Enter your choice: 2
2

        The stack : head = NULL

Enter your choice: 2
UnderFlow Error: Stack is Empty.
0

        The stack : head = NULL

Enter your choice: 4
Exiting...

        The stack : head = NULL
lemon@jupiter:~/workspace/college/DSA/Lab-6$
```

Date : 23.02.2025
Lakshana Baskaran
24011103026

QUESTION 3:

Write a C++ menu-driven program to implement infix to postfix conversion and postfix evaluation. Use a singly linked list (SLL) to implement the stack ADT as a header file. Maintain proper boundary conditions and follow good coding practices. The program has the following operations:

1. Get Infix
2. Convert Infix
3. Evaluate Postfix
4. Exit

Get Infix: Gets a valid infix expression and stores it efficiently.
Convert Infix: Converts the stored infix expression into a postfix expression. It prints the postfix expression after conversion.
Evaluate Postfix: Evaluates and prints the result of the converted infix expression.

SOURCE CODE:

```
//program accepts an infix expression converts it into postfix and
evaluates the postfix expression
#include "stack.h"
#include "float_stack.h"
#include <string>
#include <cctype>
using namespace std;

//function definitions
List get_infix();
List in_to_post(List);
float postfix_eval(List);

//helper functions
int len(string);
bool is_operator(string);
int precedence(string);
bool is_digit(string);

//1/2+3-4*9
int main() {
    List input, output, post;
    int choice = 0;
    int ans;
    string c;
```

Date : 23.02.2025
Lakshana Baskaran
24011103026

```
    printf("MENU\n1 - Get Infix\n2 - Convert Infix to Postfix\n3 -
Evaluate Postfix\n4 - Exit \n");
    while (choice != 4) {
        cout << "\n\nEnter your choice: ";
        cin >> choice;
        switch (choice) {
            case 1:
                cout << "Enter your expression element by element.
When the expression is done enter 'q' to continue:\n";
                input = get_infix();
                break;
            case 2:
                cout << "The Equivalent POSTFIX Expression: ";
                output = in_to_post(input);
                output.print();
                cout << endl;
                break;
            case 3:
                while (!output.empty()) {
                    c = output.pop();
                    post.push(c);
                }
                cout << "Evaluation of the postfix expression gives "
<< postfix_eval(post);
                break;
            case 4:
                cout << "exiting...\n";
                break;
            default:
                cout << "Invalid choice. Enter again\n";
        }
    }
    return 0;
}

//Takes input from the user for infix expression and stores it
efficiently in a stack and returns the stack
List get_infix() {
    List input, infix;
    string c, prev;

    cin >> c;
```

```
    while (c != "q") {
        input.push(c);
        cin >> c;
        prev = c;
    }
    cout << "\nThe infix expression obtained: ";
        input.print();
    while (!input.empty()) {
        c = input.pop();
        infix.push(c);
    }
    return infix;
}

//Accepts an infix expression in the form of a stack and converts it
into a postfix expression and returns the postfix stack
List in_to_post(List input) {
    List operators;
    List output;

    int p = -1;
    string a, top;
    while(input.empty() != true) {
        top = input.pop();
        if (is_operator(top)) {
            if (precedence(top) < p) {
                p = precedence(top);
                while(true) {
                    a = operators.pop();
                    output.push(a);
                    if (precedence(a) <= p) {
                        break;
                    }
                    if (operators.empty()) {
                        break;
                    }
                }
                operators.push(top);
            }
            else if (p == precedence(top)) {
                a = operators.pop();
                output.push(a);
                operators.push(top);
```

```
            }
            else if (precedence(top) > p) {
                operators.push(top);
                p = precedence(top);
            }
            else {
                continue;
            }
        }
        else {
            output.push(top);
        }
    }
    while (!operators.empty()) {
        a = operators.pop();
        output.push(a);
    }
    return output;
}

//Accepts a postfix expression in the form of a stackand evaluates
and returns the answer as float
float postfix_eval(List input) {
    string top;
    node evaluate;
    float ans;
    float val1, val2;
    while (!input.empty()) {
        top = input.pop();
        if (is_digit(top)) {
            evaluate.push(stoi(top));
        }
        else if (is_operator(top)) {
            val2 = evaluate.pop();
            val1 = evaluate.pop();
            switch (top[0]) {
                case '+': evaluate.push(val1 + val2); break;
                case '-': evaluate.push(val1 - val2); break;
                case '*': evaluate.push(val1 * val2); break;
                case '/': evaluate.push(val1 / val2); break;
                case '%': evaluate.push((int)val1 % (int)val2);
break;
            }
```

```
        }
        else {
            continue;
        }
    }

    return evaluate.pop();
}

//Checks if the string contains only digits
bool is_digit(string str) {
    for (char c: str) {
        if (!isdigit(c)) {
            return false;
        }
    }
    return true;
}

//returns the length of the string
int len(string str) {
    int i = 0;
    while (str[i] != '\0') {
        i++;
    }
    return i;
}

//returns the precedence of the operator
int precedence(string opr) {
    if (opr == "*" || opr == "/" || opr == "%") {
        return 2;
    }
    if (opr == "+" || opr == "-") {
        return 1;
    }
    if (opr == "(") {
        return 0;
    }
    return -1;
}

//checks if the operator is valid
```

Date : 23.02.2025
Lakshana Baskaran
24011103026

```
bool is_operator(string c) {
    std::string operators[] = {"+", "-", "*", "/", "%"};
    for (const std::string& op : operators) {
        if (c == op) {
            return true;
        }
    }
    return false;
}
```

OUTPUT:

```
lemon@jupiter:~/workspace/college/DSA/Lab-6$ g++ -o out in_to_post.cpp
lemon@jupiter:~/workspace/college/DSA/Lab-6$ ./out
MENU
1 - Get Infix
2 - Convert Infix to Postfix
3 - Evaluate Postfix
4 - Exit


Enter your choice: 1
Enter your expression element by element. When the expression is done enter 'q' to continue:
1
/
2
+
34
-
13
*
9
q

The infix expression obtained: 1 / 2 + 34 - 13 * 9


Enter your choice: 2
The Equivalent POSTFIX Expression: 1 2 / 34 + 13 9 * -



Enter your choice: 3
Evaluation of the postfix expression gives -82.5

Enter your choice: 4
exiting...
lemon@jupiter:~/workspace/college/DSA/Lab-6$
```

Date : 23.02.2025
Lakshana Baskaran
24011103026

QUESTION 4:

Write a C++ menu-driven program to get a string of '(' and ')' parentheses from the user and check whether they are balanced. Identify the optimal ADT and data structure to solve the problem. You can consider all previous header files for the solution's implementation. Maintain proper boundary conditions and follow good coding practices.
The program has the following operations:

1. Check Balance
2. Exit

Check Balance: Gets a string of open and closed parentheses and displays whether the parentheses are balanced or not.

SOURCE CODE:

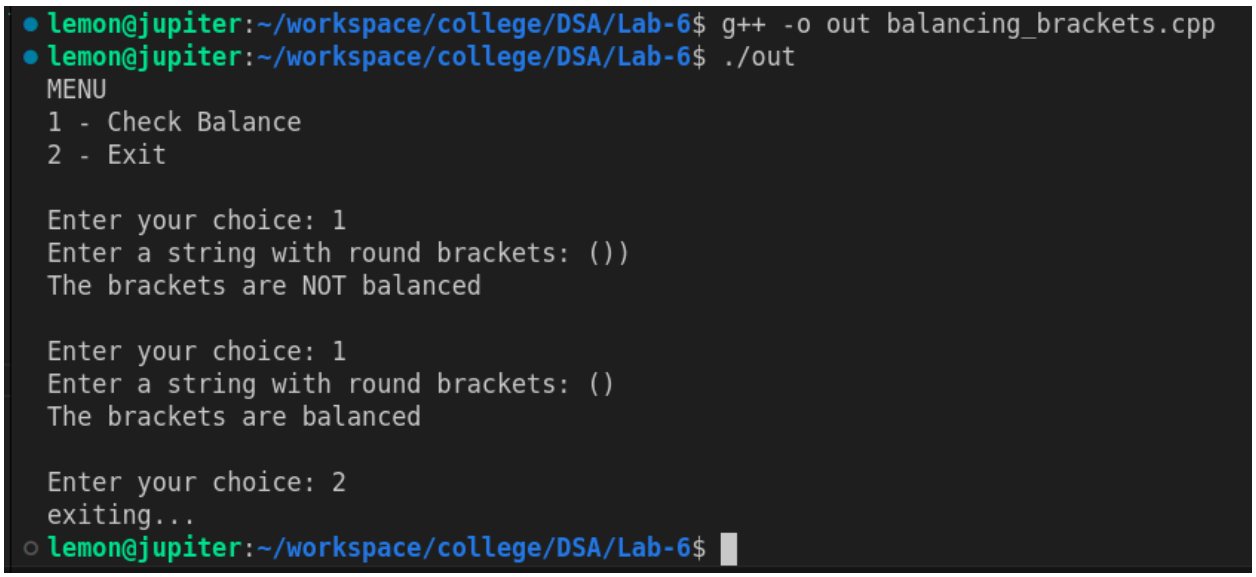
```
//balancing of brackets using stack
#include "stack.h"

//function definitions
int bracket_check(string);
int len(string);

int main(int argc, char* argv[]) {
    string str;
    printf("MENU\n1 - Check Balance\n2 - Exit\n");
    int choice = 20;
    while (choice != 2) {
        cout << "\nEnter your choice: ";
        cin >> choice;
        switch (choice) {
            case 1:
            cout << "Enter a string with round brackets: ";
            cin >> str;

            if (bracket_check(str)) {
                cout << "The brackets are balanced\n";
            }
            else {
                cout << "The brackets are NOT balanced\n";
            }
            break;
            case 2:
```

```
                cout << "exiting...\n";
                break;
            default:
                cout << "invalid option. Enter again.\n";
                break;
        }
    }
    return 0;
}

//checks for balanced brackets
int bracket_check(string str) {
    List stk;
    string ran;
    for (int i = 0; i < len(str); i++) {
        ran += str[i];
        if (str[i] == '(') {
            stk.push(ran);
        }
        else if (str[i] == ')') {
            while (true) {
                if (stk.empty()) {
                    return 0;
                }
                if (stk.pop() == "(") {
                    break;
                }

            }
        }
        else {
            continue;
        }
    }

    if (stk.empty()) {
        return 1;
    }
    else {
        return 0;
    }
}
```

Date : 23.02.2025
Lakshana Baskaran
24011103026

```
//returns the length of the string
int len(string str) {
    int i = 0;
    while (true) {
        if (str[i] == '\0') {
            break;
        }
        i++;
    }
    return i;
}
```

SOURCE CODE:



—