Name: Deshmukh Mehmood Rehan

MIS No. : 612303050

SY Computer Science – Div 1

Question: Write a C program to convert an Infix expression to Postfix form using ADT stack. Further, evaluate the obtained postfix expression. The program should handle multiple digits and only valid infix expressions.

logic.c : contains the definition of all the functions declared in the header file

along with some helper functions

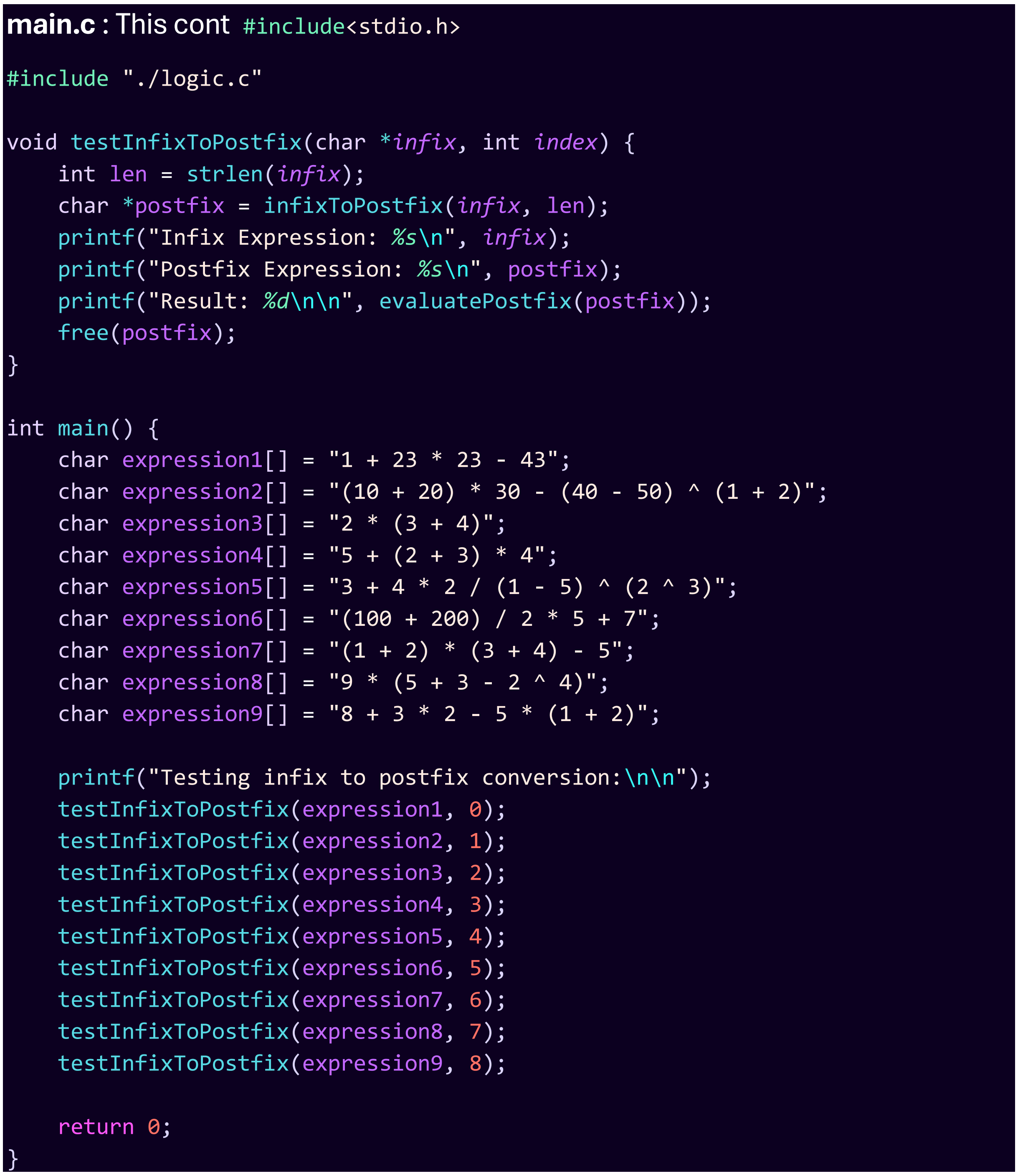
Note: We use a character and an Integer stack to solve this problem

```c
#include "../../../Data Structures/Character Stack Implementation Using Singly
LinkedList/logic.c"
#include "../../../Data Structures/Stack Implementation Using Singly
LinkedList/logic.c"
#include<ctype.h>
#include <string.h>
#include<math.h>

int getOperatorPrecedence(char ch){
    switch(ch){
        case '+':
        case '-':
            return 1;
        case '/':
        case '*':
        case '%':
            return 2;
        case '^':
            return 3;
        case '(':
            return 0;
        default:
            return -1;
    }
}

int isOperator(char ch){
    switch(ch){
        case '+':
        case '-':
```

```c
        case '/':
        case '%':
        case '*':
        case '^':
            return 1;
        default:
            return 0;
    }
}


int isValidInfix(const char *infix) {
    int len = strlen(infix);

    cStack parenthesesStack;
    cinit(&parenthesesStack);

    int lastWasOperator = 1;
    int lastWasOperand = 0;
    int inNumber = 0;

    for (int i = 0; i < len; i++) {
        char ch = infix[i];

        if (isspace(ch)) {
            continue;
        }

        if (isdigit(ch) || ch == '.') {
            if (lastWasOperand && !inNumber) {
                return 0;
            }
            lastWasOperand = 1;
            lastWasOperator = 0;
            inNumber = 1;
        } else {
            inNumber = 0;
            if (isOperator(ch)) {
                if (lastWasOperator) {
                    return 0;
                }
                lastWasOperator = 1;
                lastWasOperand = 0;
            } else if (ch == '(') {
                cpush(&parenthesesStack, ch);
                if (!lastWasOperator && !lastWasOperand) {
                    return 0;
                }
                lastWasOperator = 1;
                lastWasOperand = 0;
            } else if (ch == ')') {
                if (cisEmpty(parenthesesStack) || cpop(&parenthesesStack) !=
'(') {
                    return 0;
                }
                if (lastWasOperator) {
                    return 0;
                }
                lastWasOperator = 0;
                lastWasOperand = 1;
            } else {
                return 0;
            }
        }
    }

    if (!cisEmpty(parenthesesStack) || lastWasOperator) {
        return 0;
    }

    return 1;
}



char *infixToPostfix(char *str, int len){
    if(!isValidInfix(str)) return '\0';
    cStack s;
    cinit(&s);

    char ch;
    char *postfix = (char *)malloc(sizeof(char) * (len + 1));
    if(!postfix) return NULL;

    int i, j;
    i = j = 0;
    while((ch = str[i++]) != '\0'){
        if(isspace(ch)) continue;

        if(ch == '('){
            cpush(&s, ch);
            continue;
        }

        if(ch == ')'){
            while(!cisEmpty(s) && cpeek(s) != '('){
                postfix[j++] = cpop(&s);
                postfix[j++] = ' ';
            }

            cpop(&s);
            continue;
        }

        if(isOperator(ch)){
            while(!cisEmpty(s) && getOperatorPrecedence(cpeek(s)) >=
getOperatorPrecedence(ch)){
                postfix[j++] = cpop(&s);
                postfix[j++] = ' ';
            }

            cpush(&s, ch);
            continue;
        }

        postfix[j++] = ch;
        if (!isdigit(str[i])) postfix[j++] = ' ';
    }

    while (!cisEmpty(s)){
        postfix[j++] = cpop(&s);
        postfix[j++] = ' ';
    }

    postfix[--j] = '\0';


    return postfix;
}

int getNum(char **str){
    char result[1024];
    int i = 0;
    char ch;
    while((ch = **str) != '\0' && isdigit(ch)){
        result[i++] = ch;
        (*str)++;
    }

    result[i] = '\0';
    return atoi(result);
}


int evaluatePostfix(char *postfix){
    Stack s;
    init(&s);
```

```c
    int answer;
    char ch;
    while((ch = *postfix++)!= '\0'){
        if(isspace(ch)) continue;

        if(isdigit(ch)){
            postfix--;
            push(&s, getNum(&postfix));
            continue;
        }

        if(isOperator(ch)){
            int operand2;
            switch(ch){
                case '+':
                    push(&s, pop(&s) + pop(&s));
                    break;
                case '*':
                    push(&s, pop(&s) * pop(&s));
                    break;
                case '-':
                    operand2 = pop(&s);
                    push(&s, pop(&s) - operand2);
                    break;
                case '/':
                    operand2 = pop(&s);
                    push(&s, pop(&s) / operand2);
                    break;
                case '%':
                    operand2 = pop(&s);
                    push(&s, pop(&s) % operand2);
                    break;
                case '^':
                    operand2 = pop(&s);
                    push(&s, (int) pow(pop(&s), operand2));
                    break;
                default:
                    break;
            }
        }
    }

    answer = pop(&s);

    return answer;
}
```

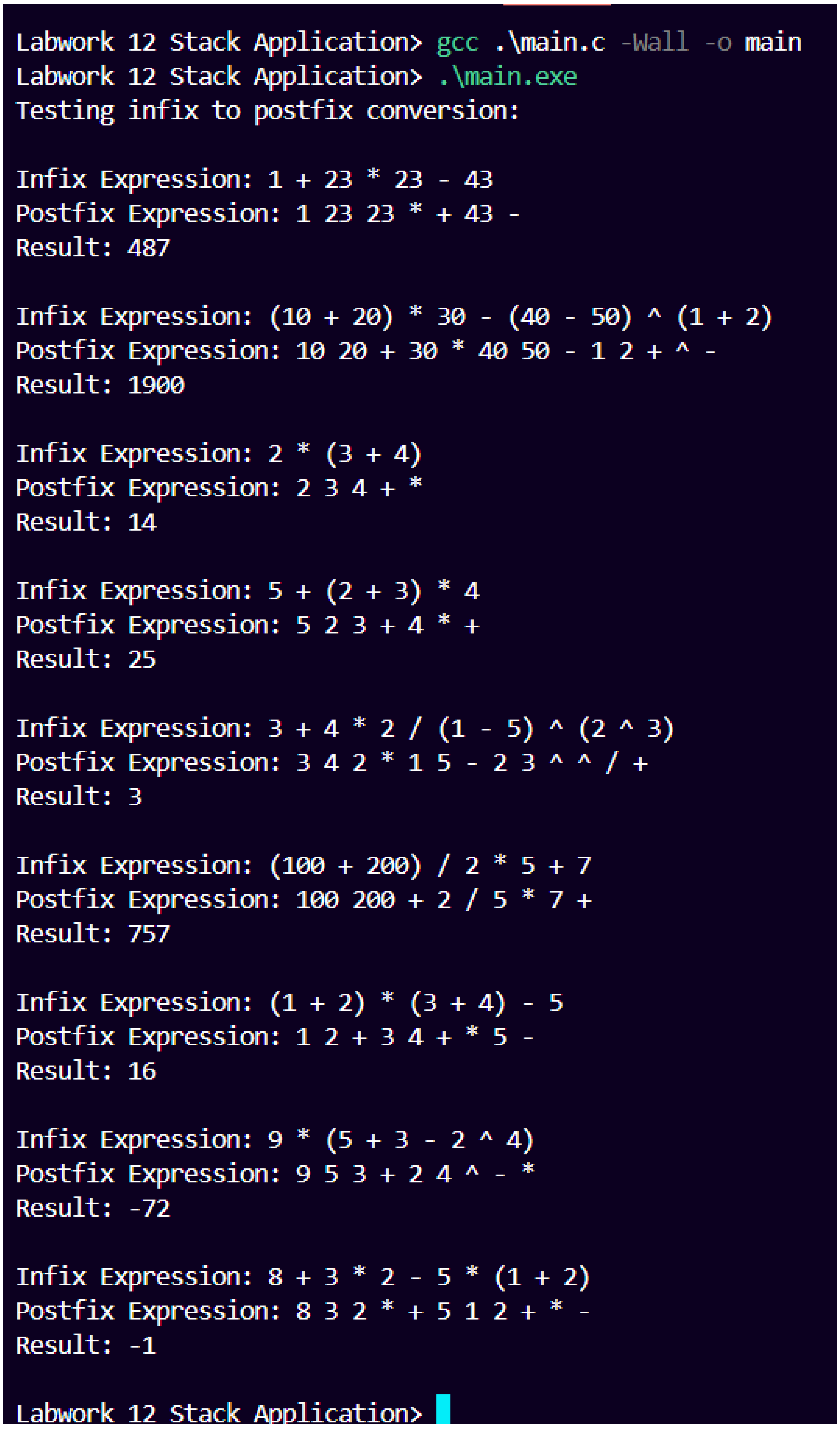
main.c : This cont



```c
#include<stdio.h>

#include "./logic.c"

void testInfixToPostfix(char *infix, int index) {
    int len = strlen(infix);
    char *postfix = infixToPostfix(infix, len);
    printf("Infix Expression: %s\n", infix);
    printf("Postfix Expression: %s\n", postfix);
    printf("Result: %d\n\n", evaluatePostfix(postfix));
    free(postfix);
}

int main() {
    char expression1[] = "1 + 23 * 23 - 43";
    char expression2[] = "(10 + 20) * 30 - (40 - 50) ^ (1 + 2)";
    char expression3[] = "2 * (3 + 4)";
    char expression4[] = "5 + (2 + 3) * 4";
    char expression5[] = "3 + 4 * 2 / (1 - 5) ^ (2 ^ 3)";
    char expression6[] = "(100 + 200) / 2 * 5 + 7";
    char expression7[] = "(1 + 2) * (3 + 4) - 5";
    char expression8[] = "9 * (5 + 3 - 2 ^ 4)";
    char expression9[] = "8 + 3 * 2 - 5 * (1 + 2)";

    printf("Testing infix to postfix conversion:\n\n");
    testInfixToPostfix(expression1, 0);
    testInfixToPostfix(expression2, 1);
    testInfixToPostfix(expression3, 2);
    testInfixToPostfix(expression4, 3);
    testInfixToPostfix(expression5, 4);
    testInfixToPostfix(expression6, 5);
    testInfixToPostfix(expression7, 6);
    testInfixToPostfix(expression8, 7);
    testInfixToPostfix(expression9, 8);

    return 0;
}
```

ains the code to test the implementation

Output:



```
Labwork 12 Stack Application> gcc .\main.c -Wall -o main
Labwork 12 Stack Application> .\main.exe
Testing infix to postfix conversion:

Infix Expression: 1 + 23 * 23 - 43
Postfix Expression: 1 23 23 * + 43 -
Result: 487

Infix Expression: (10 + 20) * 30 - (40 - 50) ^ (1 + 2)
Postfix Expression: 10 20 + 30 * 40 50 - 1 2 + ^ -
Result: 1900

Infix Expression: 2 * (3 + 4)
Postfix Expression: 2 3 4 + *
Result: 14

Infix Expression: 5 + (2 + 3) * 4
Postfix Expression: 5 2 3 + 4 * +
Result: 25

Infix Expression: 3 + 4 * 2 / (1 - 5) ^ (2 ^ 3)
Postfix Expression: 3 4 2 * 1 5 - 2 3 ^ ^ / +
Result: 3

Infix Expression: (100 + 200) / 2 * 5 + 7
Postfix Expression: 100 200 + 2 / 5 * 7 +
Result: 757

Infix Expression: (1 + 2) * (3 + 4) - 5
Postfix Expression: 1 2 + 3 4 + * 5 -
Result: 16

Infix Expression: 9 * (5 + 3 - 2 ^ 4)
Postfix Expression: 9 5 3 + 2 4 ^ - *
Result: -72

Infix Expression: 8 + 3 * 2 - 5 * (1 + 2)
Postfix Expression: 8 3 2 * + 5 1 2 + * -
Result: -1

Labwork 12 Stack Application>
```