Lab 8

Program:

Write a program to take a string of parenthesis as input and to check whether the parenthesis are balanced or not with the use of stacks.

Source Code:

```
#include<stdio.h>
#include<string.h>
#define MAX 100
void parenthesis_check(char s[]) {
    char stack[MAX];
    int stack_index = -1;
    int len = strlen(s);
    char input[MAX];
    int input_index = -1;
    int flag = 0;
    for(int i = 0; i < len; i++) {
        char c = s[i];
        if(c == '(' || c == '[' || c == '{') {
            stack_index++;
            stack[stack_index]  = c;
        }
        else if(c == ')' || c == ']' || c == '}') {
            if(stack_index > -1) {
                if((c == ')' && stack[stack_index] == '(') || (c == ']' &&
stack[stack_index] == '[') || (c == '}' && stack[stack_index] == '{')) {
                    stack_index--;
                    // printf("%d \n", stack_index);
                }
                else {
                    // printf("Unbalanced!\n");
                    flag = 1;
    break;
                }
            }
            else if(stack_index == -1) {
                // printf("Unbalanced!\n");
                flag = 1;
    break;
            }
        }
    }
    if(stack_index == -1) {
        if(flag == 0) {
            printf("\nParenthesis string is balanced...\n\n");
        }
```

```
        else {
            printf("\nStirng is unbalanced...\n\n");
        }
    }
    else if(stack_index >= 0) {
        printf("\nString is unbalanced...\n\n");
    }
}
int main() {
    char string[MAX];
    printf("Parenthesis Checker:\n\n");
    printf("Enter parenthesis string: ");
    fgets(string, MAX, stdin);
    parenthesis_check(string);
    return 0;
}
```

Output:

```
Parenthesis Checker:

Enter parenthesis string: {()[([[])[]])]}

Parenthesis string is balanced...

PS C:\Users\Faizan\Desktop\UNI-STUFF\Semester 3\Data Structures in C> cd "c:\Users\Faizan\Deskt
sis_Checker.c -o Parenthesis_Checker } ; if ($?) { .\Parenthesis_Checker }
Parenthesis Checker:

Enter parenthesis string: {{()[([[])[]])]}

String is unbalanced...

PS C:\Users\Faizan\Desktop\UNI-STUFF\Semester 3\Data Structures in C>
```

Lab 9

Program:

Write a program to perform infix to prefix conversion on an expression and to perform evaluation of the prefix expression.

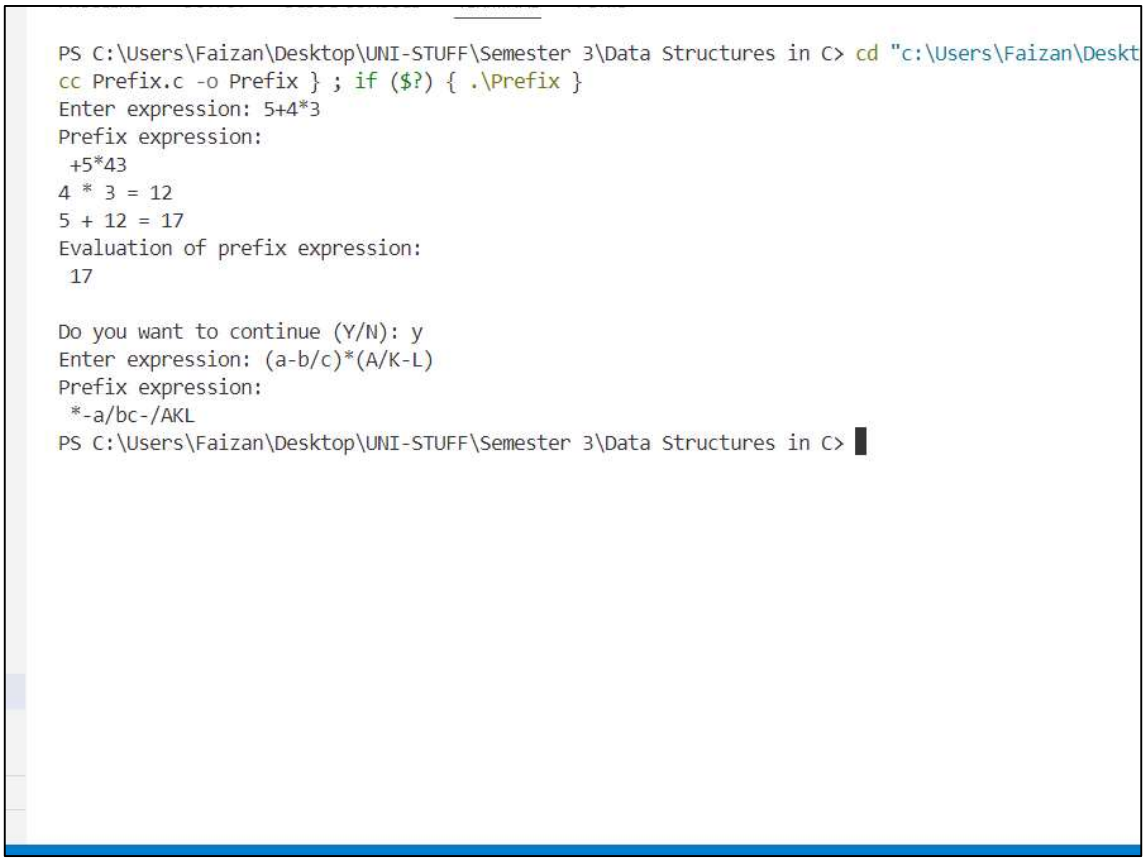
Source Code:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#define MAX 100
int precedence(char c) {
    if (c == '^')
        return 3;
    else if (c == '/' || c == '*')
        return 2;
    else if (c == '+' || c == '-')
        return 1;
    else
        return -1;
}
void ReverseString(char* str) {
    int len = strlen(str);
    for (int i = 0; i < len / 2; i++) {
        char temp = str[i];
        str[i] = str[len - 1 - i];
        str[len - 1 - i] = temp;
    }
}
void Prefix(char s[]) {
    char exp[MAX];
    int len = strlen(s);
    char stack[MAX];
    int stack_index = -1;
    int output_index = 0;
    ReverseString(s);
    for (int i = 0; i < len; i++) {
        char c = s[i];
        if (isdigit(c) || isalpha(c)) {
            exp[output_index++] = c;
        } else if (c == ')') {
            stack[++stack_index] = c;
        } else if (c == '(') {
            while (stack_index >= 0 && stack[stack_index] != ')') {
                exp[output_index++] = stack[stack_index--];
            }
```

```
            stack_index--;
        } else {
            while (stack_index >= 0 && precedence(c) <
precedence(stack[stack_index])) {
                exp[output_index++] = stack[stack_index--];
            }
            stack[++stack_index] = c;
        }
    }
    while (stack_index >= 0) {
        exp[output_index++] = stack[stack_index--];
    }
    exp[output_index] = '\0';
    ReverseString(exp);
    printf("Prefix expression:\n %s\n", exp);
    int evaluation[MAX];
    int eval_index = -1;
    for (int i = strlen(exp) - 1; i >= 0; i--) {
        char e = exp[i];
        if (isdigit(e)) {
            evaluation[++eval_index] = e - '0';
        } else if (e == '+' || e == '-' || e == '*' || e == '/') {
            int a = evaluation[eval_index--];
            int b = evaluation[eval_index--];
            int result = 0;
            if (e == '+') {
                result = a + b;
                printf("%d + %d = %d\n", a, b, result);
            } else if (e == '-') {
                result = a - b;
                printf("%d - %d = %d\n", a, b, result);
            } else if (e == '*') {
                result = a * b;
                printf("%d * %d = %d\n", a, b, result);
            } else if (e == '/') {
                result = a / b;
                printf("%d / %d = %d\n", a, b, result);
            }
            evaluation[++eval_index] = result;
        }
    }
    printf("Evaluation of prefix expression:\n %d\n", evaluation[eval_index]);
}
int main() {
    char exp[MAX];
    char contn;
    do {
        printf("Enter expression: ");
```

```
        fgets(exp, MAX, stdin);
        exp[strcspn(exp, "\n")] = '\0';
        Prefix(exp);
        int ch;
        while ((ch = getchar()) != '\n' && ch != EOF);
        printf("Do you want to continue (Y/N): ");
        scanf(" %c", &contn);
        while ((ch = getchar()) != '\n' && ch != EOF);
    }
    while(contn == 'y' || contn == 'Y');
    printf("\nProgram terminated successfully...\n\n");
    return 0;
}
```

Output:



Lab 10

Program:

Write a program to perform infix to postfix conversion on an expression and to perform evaluation of the postfix expression.

Source Code:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#define MAX 100
int precedence(char c) {
    if (c == '^')
        return 3;
    else if (c == '/' || c == '*')
        return 2;
    else if (c == '+' || c == '-')
        return 1;
    else
        return -1;
}
void Postfix(char s[]) {
    char output_exp[MAX];
    int output_index = 0;
    int len = strlen(s);
    char stack[MAX];
    int stack_index = -1;
    for (int i = 0; i < len; i++) {
        char c = s[i];
        if ((c >= 'a' && c <= 'z') || (c >= 'A' && c <= 'Z') || (c >= '0' && c
<= '9')) {
            output_exp[output_index++] = c;
        }
        else if (c == '(') {
            stack[++stack_index] = c;
        }
        else if (c == ')') {
            while (stack_index >= 0 && stack[stack_index] != '(') {
                output_exp[output_index++] = stack[stack_index--];
            }
            stack_index--;
        }
        else {
            while (stack_index >= 0 && (precedence(s[i]) <
precedence(stack[stack_index]) ||
```

```
                                        precedence(s[i]) ==
precedence(stack[stack_index]))) {
                output_exp[output_index++] = stack[stack_index--];
            }
            output_exp[output_index++] = ' ';
            stack[++stack_index] = c;
        }
    }
    while (stack_index >= 0) {
        output_exp[output_index++] = stack[stack_index--];
        output_exp[output_index++] = ' ';
    }
    output_exp[output_index] = '\0';
    printf("Postfix expression:\n %s\n", output_exp);
    int evaluation[MAX];
    int eval_index = -1;
    int i = 0;
    while (output_exp[i] != '\0') {
        if (isdigit(output_exp[i])) {
            int num = 0;
            while (isdigit(output_exp[i])) {
                num = num * 10 + (output_exp[i] - '0');
                i++;
            }
            evaluation[++eval_index] = num;
        } else if (output_exp[i] == ' ') {
            i++;
        } else if (output_exp[i] == '+' || output_exp[i] == '-' ||
output_exp[i] == '*' || output_exp[i] == '/') {
            int b = evaluation[eval_index--];
            int a = evaluation[eval_index--];
            int result = 0;
            if (output_exp[i] == '+') {
                result = a + b;
                printf("%d + %d = %d\n", a, b, result);
            } else if (output_exp[i] == '-') {
                result = a - b;
                printf("%d - %d = %d\n", a, b, result);
            } else if (output_exp[i] == '*') {
                result = a * b;
                printf("%d * %d = %d\n", a, b, result);
            } else if (output_exp[i] == '/') {
                result = a / b;
                printf("%d / %d = %d\n", a, b, result);
            }
            evaluation[++eval_index] = result;
            i++;
        } else {
```

```
            i++;
        }
    }
    printf("Evaluation of expression:\n %d\n", evaluation[eval_index]);
}
int main() {
    char exp[MAX];
    char contn;
    int ch;
    do {
        printf("Enter expression: ");
        fgets(exp, MAX, stdin);
        exp[strcspn(exp, "\n")] = '\0';
        Postfix(exp);
        while ((ch = getchar()) != '\n' && ch != EOF);
        printf("Do you want to continue (Y/N): ");
        scanf(" %c", &contn);
        while ((ch = getchar()) != '\n' && ch != EOF);
    }
    while (contn == 'y' || contn == 'Y');
    printf("\nProgram terminated successfully...\n\n");
    return 0;
}
```

Output: