Assignment No. 3

Name: Bhavin Ratansing Patil

Roll No.: 26 SEDA

Q.1 Expression Conversion and evaluation using stack:

- **Infix to Postfix Conversion:**

Infix expressions are easy to understand and evaluate to us human whereas it is difficult for machines to determine the operator, operand and precedence of the operators to evaluate the infix expressions. To concur this, we need to convert infix expressions to postfix for machines to understand.

Algorithm:

```
Step 1:  create Stack (stack)
Step 2:  loop (for each character in Expression)
    i. if (character is open parenthesis)
        i. pushStack (stack, character)
    ii. elseif (character is close parenthesis)
        a. popStack (stack, character)
        b. loop (character not open parenthesis)
            i. concatenate character to postFixExpr
            ii. popStack (stack, character)
        c. end loop
    iii. elseif (character is operator) //Test priority of token to token at top of stack
        a. stackTop (stack, topToken)
        b. loop (not emptyStack (stack) AND priority(character) <=
        priority(topToken))
            i. popStack (stack, tokenOut)
            ii. concatenate tokenOut to postFixExpr
            iii. stackTop (stack, topToken)
        c. end loop
        d. pushStack (stack, token)
        iv. else // Character is operand
                a. Concatenate token to postFixExpr
        v. end if
Step 3:  end loop //Input Expression empty.
//Pop stack to postFix
Step 4:  loop (not emptyStack (stack))
        i. popStack (stack, character)
        ii. concatenate token to postFixExpr
Step 5:  end loop
Step 6: return postFix
//end inToPostFix
```

- **Infix to prefix conversion:**

Infix expressions are easy to understand and evaluate to us human whereas it is difficult for machines to determine the operator, operand and precedence of the operators to evaluate the infix expressions. To concur this, we need to convert infix expressions to prefix for machines to understand.

Algorithm:

```
Step 1:  create Stack (stack)
Step 2: Input infix expression and reverse it
Step 3: loop (for each character in Expression)
        i. if (character is close parenthesis)
                i. pushStack (stack, character)
        ii. elseif (character is open parenthesis)
                a.popStack (stack, character)
                b. loop (character not close parenthesis)
                        i. concatenate character to prefixExpr
                        ii. popStack (stack, character)
                c. end loop
        iii. elseif (character is operator) //Test priority of token to token at top of stack
                a. stackTop (stack, topToken)
        b. loop (not emptyStack (stack) AND priority(character) <= priority(topToken))
            i. popStack (stack, tokenOut)
            ii. concatenate tokenOut to prefixExpr
            iii. stackTop (stack, topToken)
        c. end loop
        d. pushStack (stack, token)
    iv. else // Character is operand
        a. Concatenate token to prefixExpr
    v. end if
Step 4:  end loop //Input Expression empty. //Pop stack to prefix
Step 5: loop (not emptyStack (stack))
    i. popStack (stack, character)
    ii. concatenate token to prefixExpr
Step 6: end loop
Step 7: reverse prefix
Step 8: return prefix //end inToPrefix
```

- **Postfix Evaluation Algorithm:**

Step 1: Create an operand stack.

Step 2: If the character is an operand, push it to the operand stack.

Step 3: If the character is an operator, pop two operands from the stack, operate and push the result back to the stack.

Step 4: After the entire expression has been traversed, pop the final result from the stack.

- **Prefix Evaluation Algorithm:**

Step 1: Reverse the prefix expression.

Step 2: Create an operand stack.

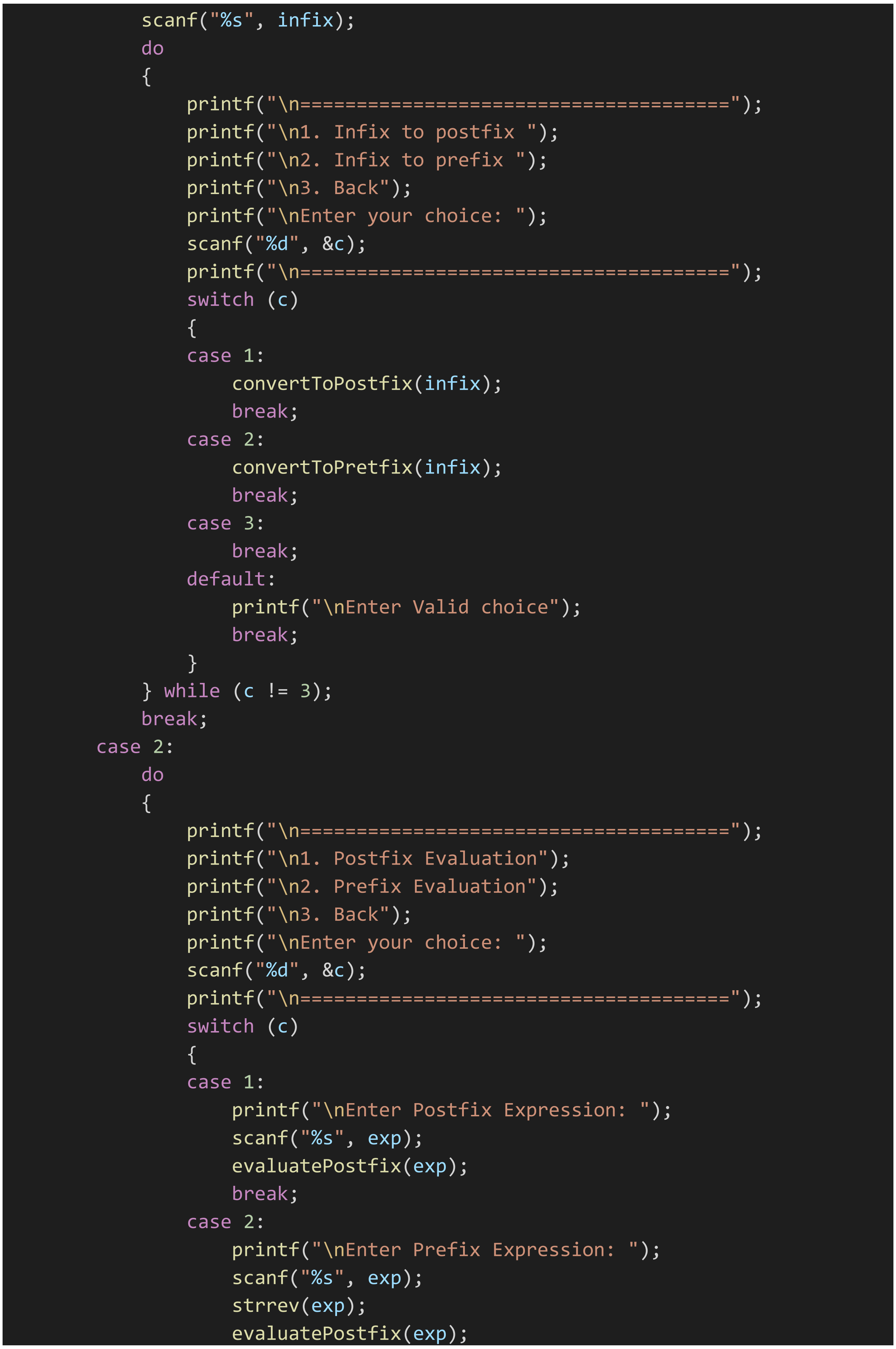
Step 3: If the character is an operand, push it to the operand stack.

Step 4: If the character is an operator, pop two operands from the stack, operate and push the result back to the stack.

Step 5: After the entire expression has been traversed, pop the final result from the stack.

Program:

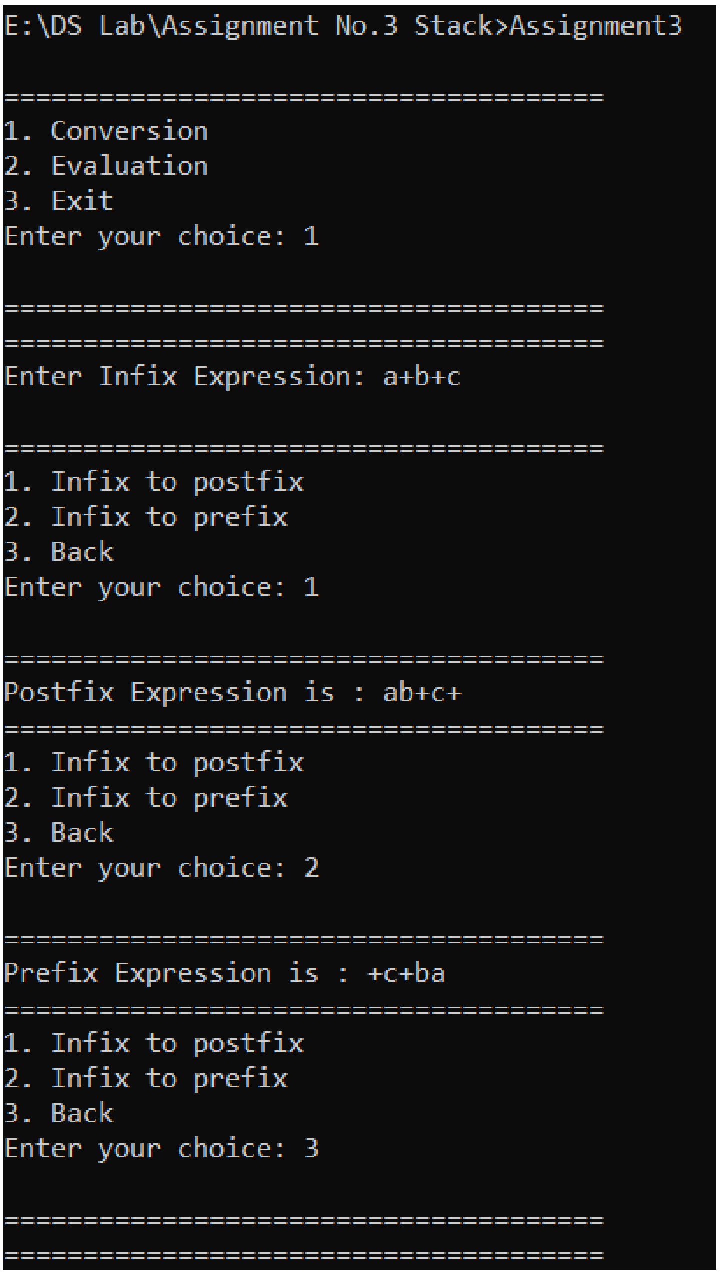
```c
#include <stdio.h>
#include <ctype.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
char st[100];
int top = -1;
void push(char x)
{
    st[++top] = x;
}
char pop()
{
    return st[top--];
}
int priority(char x)
{
    if (x == '^')
        return 3;
    if (x == '*' || x == '/')
        return 2;
    if (x == '+' || x == '-')
        return 1;
    return 0;
}
void convertToPostfix(char infix[])
{
    char postfix[20], *s, x;
    int i = -1;
    s = infix;
    while (*s != '\0')
    {
        if (isalnum(*s))
            postfix[++i] = *s;
        else if (*s == '(')
            push(*s);
        else if (*s == ')')
            while ((x = pop()) != '(')
                postfix[++i] = x;
        else
        {
            while (priority(st[top]) >= priority(*s))
                postfix[++i] = pop();
            push(*s);
        }
        s++;
    }
    while (top != -1)
        postfix[++i] = pop();
    postfix[++i] = '\0';
    printf("\nPostfix Expression is : %s", postfix);
}
void convertToPretfix(char infix[])
{
    char prefix[20], *s, x;
    int i = -1;
    s = infix;
    while (*s != '\0')
    {

        if (isalnum(*s))
            prefix[++i] = *s;
        else if (*s == ')')
            push(*s);
        else if (*s == '(')
            while ((x = pop()) != ')')
                prefix[++i] = x;

        else
        {
            while (priority(st[top]) >= priority(*s))
                prefix[++i] = pop();
            push(*s);
        }
        s++;
    }
    while (top != -1)
        prefix[++i] = pop();
    prefix[++i] = '\0';
    strrev(prefix);
    printf("\nPrefix Expression is : %s", prefix);
}
int operation(char ch, int a, int b)
{
    switch (ch)
    {
    case '+':
        return b + a;
    case '-':
        return b - a;
    case '*':
        return b * a;
    case '/':
        return b / a;
    case '^':
        return pow(b, a);
    }
    return 0;
}
void evaluatePostfix(char exp[])
{
    char *s;
    int x, a, b, c;
    s = exp;
    while (*s != '\0')
    {
        if (isalnum(*s))
        {
            x = *s - '0';
            push(x);
        }
        else
        {
            a = pop();
            b = pop();
            c = operation(*s, a, b);
            push(c);
        }
        s++;
    }
    printf("\n%d", pop());
}
void main()
{
    char infix[20], exp[20];
    int ch, c;
    do
    {
        printf("\n======================================");
        ;
        printf("\n1. Conversion\n2. Evaluation \n3. Exit ");
        printf("\nEnter your choice: ");
        scanf("%d", &ch);
        printf("\n======================================");
        switch (ch)
        {
        case 1:
            printf("\n======================================");
            printf("\nEnter Infix Expression: ");
```



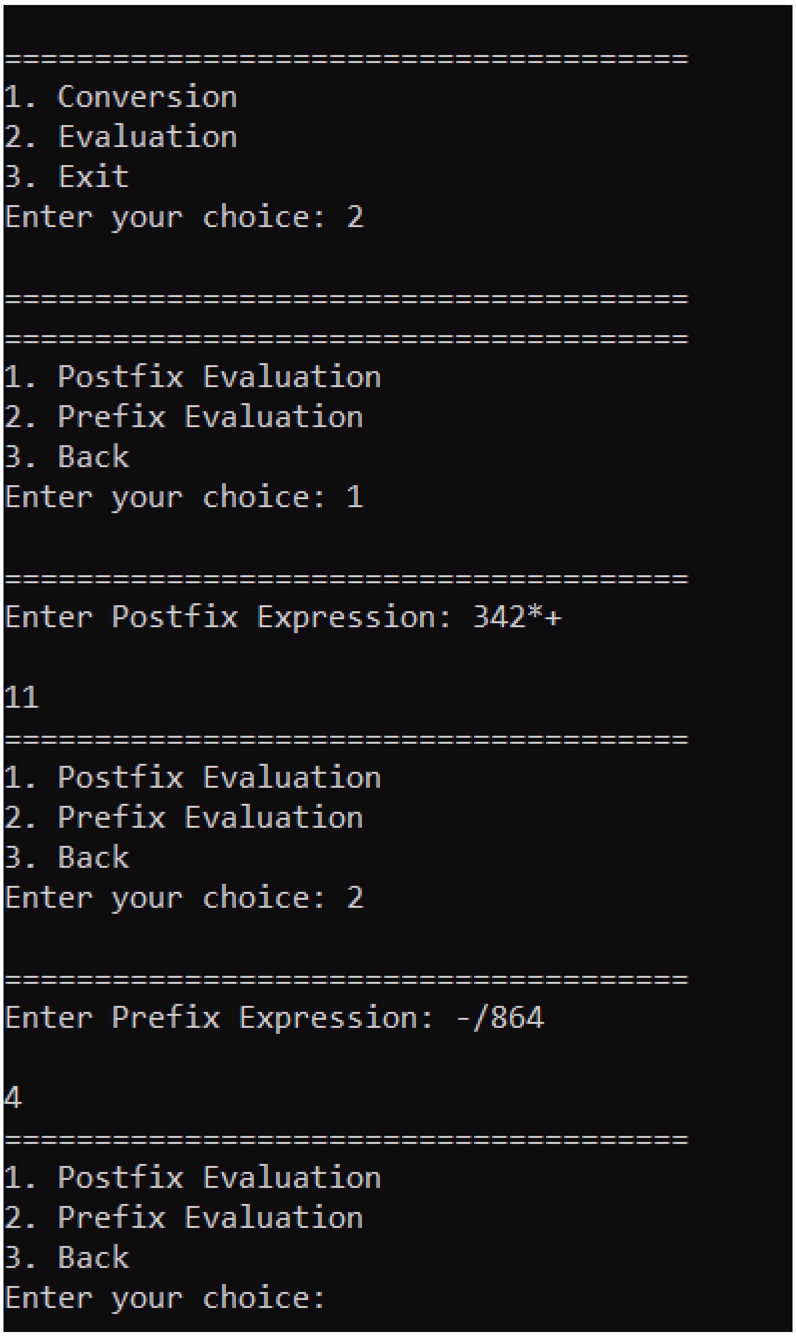
```
            scanf("%s", infix);
            do
            {
                printf("\n======================================");
                printf("\n1. Infix to postfix ");
                printf("\n2. Infix to prefix ");
                printf("\n3. Back");
                printf("\nEnter your choice: ");
                scanf("%d", &c);
                printf("\n======================================");
                switch (c)
                {
                case 1:
                    convertToPostfix(infix);
                    break;
                case 2:
                    convertToPretfix(infix);
                    break;
                case 3:
                    break;
                default:
                    printf("\nEnter Valid choice");
                    break;
                }
            } while (c != 3);
            break;
        case 2:
            do
            {
                printf("\n======================================");
                printf("\n1. Postfix Evaluation");
                printf("\n2. Prefix Evaluation");
                printf("\n3. Back");
                printf("\nEnter your choice: ");
                scanf("%d", &c);
                printf("\n======================================");
                switch (c)
                {
                case 1:
                    printf("\nEnter Postfix Expression: ");
                    scanf("%s", exp);
                    evaluatePostfix(exp);
                    break;
                case 2:
                    printf("\nEnter Prefix Expression: ");
                    scanf("%s", exp);
                    strrev(exp);
                    evaluatePostfix(exp);
```

```c
                    break;
                case 3:
                    break;
                default:
                    printf("\nEnter Valid choice");
                }
            } while (c != 3);
        case 3:
            break;
        default:
            printf("\nEnter valid Choice");
            break;
        }
    } while (ch != 3);
}
```

Output:



```
E:\DS Lab\Assignment No.3 Stack>Assignment3

=======================================
1. Conversion
2. Evaluation
3. Exit
Enter your choice: 1

=======================================
=======================================
Enter Infix Expression: a+b+c

=======================================
1. Infix to postfix
2. Infix to prefix
3. Back
Enter your choice: 1

=======================================
Postfix Expression is : ab+c+
=======================================
1. Infix to postfix
2. Infix to prefix
3. Back
Enter your choice: 2

=======================================
Prefix Expression is : +c+ba
=======================================
1. Infix to postfix
2. Infix to prefix
3. Back
Enter your choice: 3

=======================================
=======================================
```



```
========================================
1. Conversion
2. Evaluation
3. Exit
Enter your choice: 2

========================================
========================================
1. Postfix Evaluation
2. Prefix Evaluation
3. Back
Enter your choice: 1

========================================
Enter Postfix Expression: 342*+

11
========================================
1. Postfix Evaluation
2. Prefix Evaluation
3. Back
Enter your choice: 2

========================================
Enter Prefix Expression: -/864

4
========================================
1. Postfix Evaluation
2. Prefix Evaluation
3. Back
Enter your choice:
```