DS Lab Assignment 4

Pranav Joshi
CS-B
Roll no: 43

Title: WAP to convert Infix expression into equivalent Postfix and Prefix format using Stack.

Code:

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define size 45
int stack[size];
int top = -1;
char prefix[100];
void push(char ch)
{
    top++;
    stack[top] = ch;
}
char pop()
{
    char x = stack[top];
    top--;
    return x;
}
int isEmpty()
{
    if (top == -1)
        return 1;
    return 0;
}
int prec(char ch)
{
    if (ch == '^')
        return 3;
    else if (ch == '*' || ch == '/' || ch == '%')
        return 2;
    else if (ch == '+' || ch == '-')
        return 1;
    else
```

```c
        return 0;
}
void reverseString(char *str)
{
    int l, i;
    char *begin_ptr, *end_ptr, ch;
    l = strlen(str);
    begin_ptr = str;
    end_ptr = str + l - 1;
    for (i = 0; i < (l - 1) / 2; i++)
    {
        ch = *end_ptr;
        *end_ptr = *begin_ptr;
        *begin_ptr = ch;

        begin_ptr++;
        end_ptr--;
    }
}
int isOperator(char ch)
{
    if (ch == '^' || ch == '*' || ch == '/' || ch == '+' || ch == '-')
        return 1;
    else
        return 0;
}
char *infixtoprefix(char *infix)
{
    reverseString(infix);
    int i = 0, j = 0;
    while(infix[i]!='\0')
    {
        if((infix[i] >= 'a' && infix[i] <= 'z') || (infix[i] >= 'A' &&
infix[i] <= 'Z'))
        {
            prefix[j]=infix[i];
            i++;
            j++;
        }
        else if (infix[i] == ')')
        {
            push(infix[i]);
            i++;
        }
        else if (infix[i] == '(')
        {
            while(!isEmpty() && stack[top] != ')')
            {
                prefix[j] = pop();
                j++;
            }
            pop();
            i++;
        }
        else
        {
            if(prec(infix[i])>=prec(stack[top]))
            {
                push(infix[i]);
                i++;
            }
            else
            {
                while(prec(infix[i])<prec(stack[top]))
                {
                    prefix[j]=pop();
                    j++;
                }
            }
        }
    }
    while (!isEmpty())
    {
        prefix[j] = pop();
        j++;
    }
    prefix[j] = '\0';
    reverseString(prefix);
    return prefix;
}
char *infixtopostfix(char *infix)
{
    char *postfix = (char *)malloc((strlen(infix) + 1) * sizeof(char));
    int i = 0, j = 0;
    while (infix[i] != '\0')
    {
        if ((infix[i] >= 'a' && infix[i] <= 'z') || (infix[i] >= 'A' &&
infix[i] <= 'Z'))
        {
            postfix[j] = infix[i];
            i++;
            j++;
        }
        else if (infix[i] == '(')
        {
            push(infix[i]);
            i++;
        }
        else if (infix[i] == ')')
        {
            while (!isEmpty() && stack[top] != '(')
            {
                postfix[j] = pop();
                j++;
            }
            pop();
            i++;
        }
        else
        {
            while(!isEmpty() && prec(infix[i])<=prec(stack[top]))
            {
                postfix[j]=pop();
                j++;
            }
            push(infix[i]);
            i++;
        }
    }
    while (!isEmpty())
    {
        postfix[j] = pop();
        j++;
    }
    postfix[j] = '\0';
    return postfix;
}

int main()
{
    char infix[100];
    char *prefix_infix, *postfix_infix;

    printf("Enter an infix expression: ");
    scanf("%s", infix);

    prefix_infix = (char*) malloc(strlen(infix)+1);
    postfix_infix = (char*) malloc(strlen(infix)+1);

    strcpy(prefix_infix, infix);
    strcpy(postfix_infix, infix);

    printf("Input Expression: %s\n\n", infix);
```

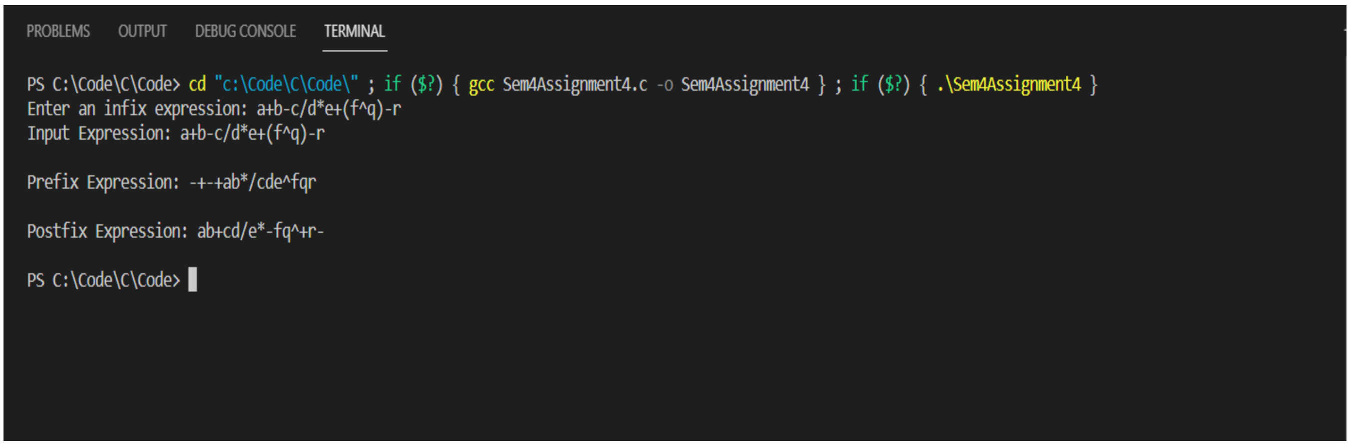
```c
    printf("Prefix Expression: %s\n\n", infixtoprefix(prefix_infix));

    printf("Postfix Expression: %s\n\n", infixtopostfix(postfix_infix));

    free(prefix_infix);
    free(postfix_infix);

    return 0;
}
```

Output:



```
PS C:\Code\C\Code> cd "c:\Code\C\Code\" ; if ($?) { gcc Sem4Assignment4.c -o Sem4Assignment4 } ; if ($?) { .\Sem4Assignment4 }
Enter an infix expression: a+b-c/d*e+(f^q)-r
Input Expression: a+b-c/d*e+(f^q)-r

Prefix Expression: -+-+ab*/cde^fqr

Postfix Expression: ab+cd/e*-fq^+r-

PS C:\Code\C\Code>
```