DATA STRUCTURES AND ALGORITHMS LAB ASSIGNMENT – 4

Student Details

- Name: Pratyush Deo Singh
- Registration Number: 25BCE5101
- Course / Branch: B Tech CSE
- Semester: 2nd
- Subject: Data Structures and Algorithms
- Faculty Name: Dr. Malini A

Program 1: Infix to Postfix Conversion using Stack

Aim

To write a C program to convert an infix expression into postfix expression using stack.

Program Code

```c
#include <stdio.h>
#include <ctype.h>

char stack[50];
int top = -1;

void push(char x) {
    stack[++top] = x;
}

char pop() {
    return stack[top--];
}

int priority(char x) {
    if(x == '+' || x == '-') return 1;
    if(x == '*' || x == '/') return 2;
    return 0;
}

int main() {
    char infix[50], postfix[50];
    int i = 0, k = 0;
    printf("Enter infix expression: ");
    scanf("%s", infix);

    while(infix[i]) {
        if(isalnum(infix[i]))
            postfix[k++] = infix[i];
        else if(infix[i] == '(')
            push(infix[i]);
        else if(infix[i] == ')') {
            while(stack[top] != '(')
                postfix[k++] = pop();
            pop();
        } else {
            while(top != -1 && priority(stack[top]) >= priority(infix[i]))
                postfix[k++] = pop();
            push(infix[i]);
        }
        i++;
    }

    while(top != -1)
        postfix[k++] = pop();

    postfix[k] = '\0';
    printf("Postfix expression: %s", postfix);
    return 0;
}
```

Output

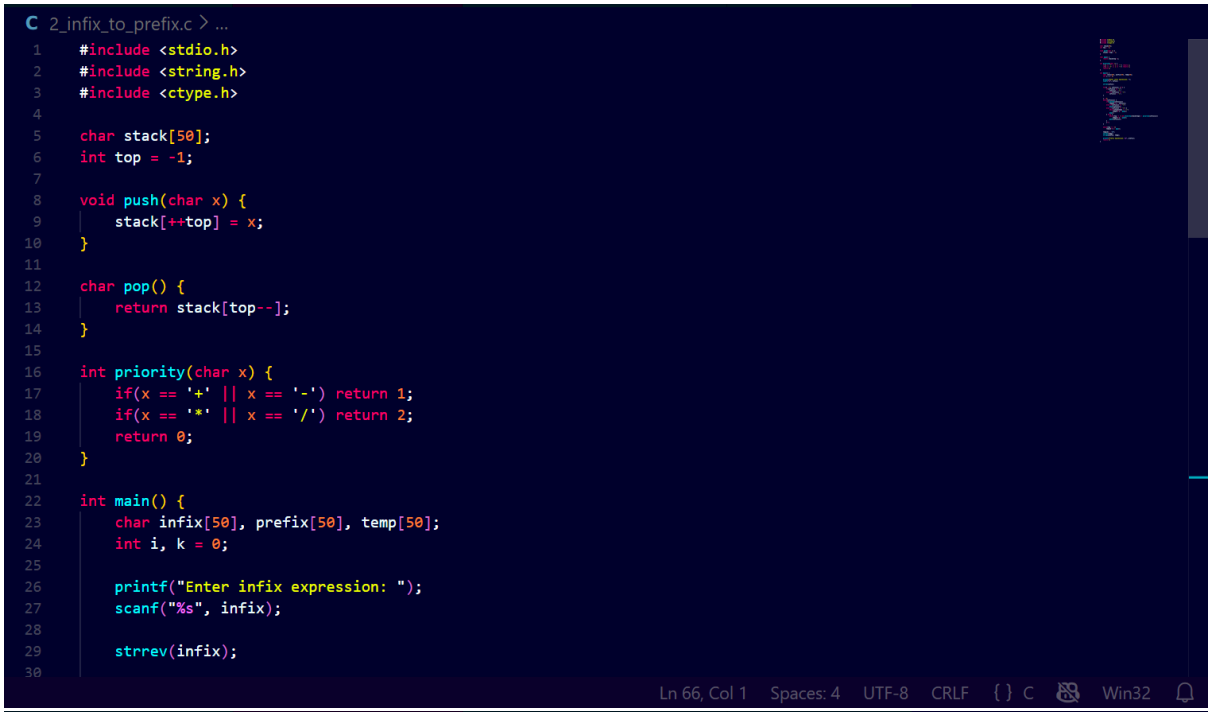
```
Enter infix expression: (A+B)*C
Postfix expression: AB+C*
```

Program 2: Infix to Prefix Conversion using Stack

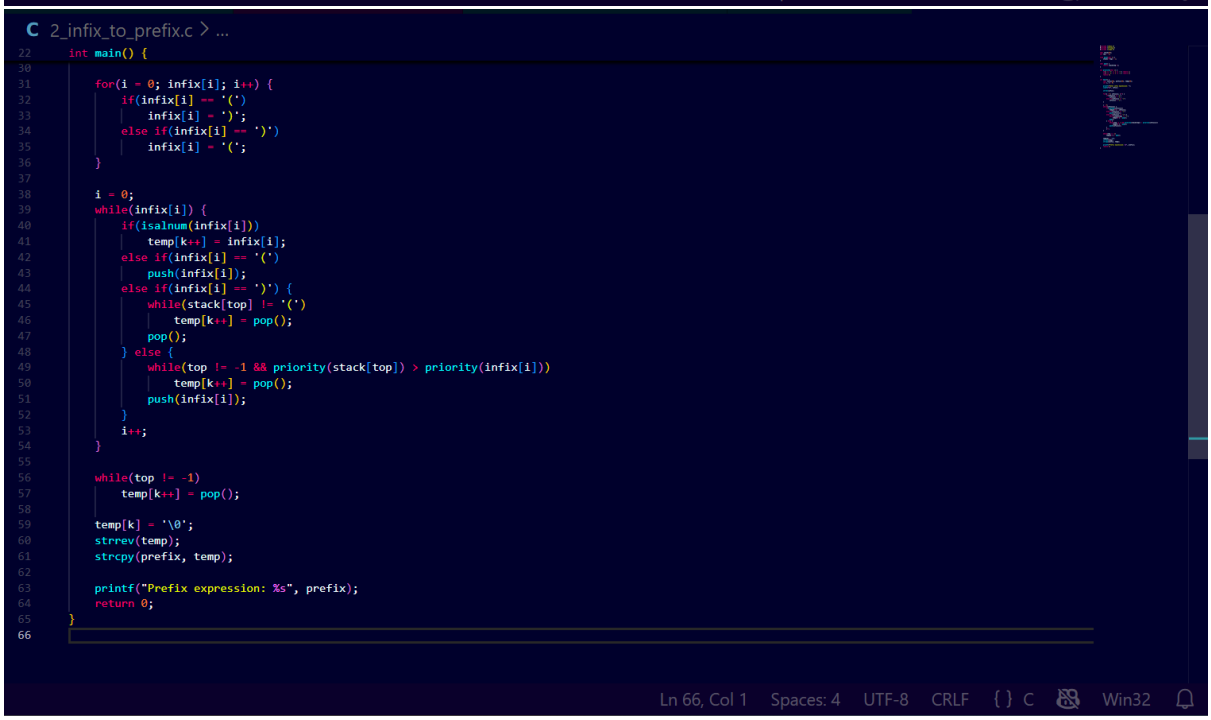
Aim

To write a C program to convert an infix expression into prefix expression using stack.

Program Code



```c
#include <stdio.h>
#include <string.h>
#include <ctype.h>

char stack[50];
int top = -1;

void push(char x) {
    stack[++top] = x;
}

char pop() {
    return stack[top--];
}

int priority(char x) {
    if(x == '+' || x == '-') return 1;
    if(x == '*' || x == '/') return 2;
    return 0;
}

int main() {
    char infix[50], prefix[50], temp[50];
    int i, k = 0;

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

    strrev(infix);
```



```c
int main() {

    for(i = 0; infix[i]; i++) {
        if(infix[i] == '(')
            infix[i] = ')';
        else if(infix[i] == ')')
            infix[i] = '(';
    }

    i = 0;
    while(infix[i]) {
        if(isalnum(infix[i]))
            temp[k++] = infix[i];
        else if(infix[i] == '(')
            push(infix[i]);
        else if(infix[i] == ')') {
            while(stack[top] != '(')
                temp[k++] = pop();
            pop();
        } else {
            while(top != -1 && priority(stack[top]) > priority(infix[i]))
                temp[k++] = pop();
            push(infix[i]);
        }
        i++;
    }

    while(top != -1)
        temp[k++] = pop();

    temp[k] = '\0';
    strrev(temp);
    strcpy(prefix, temp);

    printf("Prefix expression: %s", prefix);
    return 0;
}
```

Output

```
Enter infix expression: (A+B)*C
Prefix expression: *+ABC
```

Program 3: Postfix Expression Evaluation

Aim

To write a C program to evaluate a postfix expression using stack.

Program Code

3_postfix_evaluation.c > ...

```c
#include <stdio.h>
#include <ctype.h>

int stack[50];
int top = -1;

void push(int x) {
    stack[++top] = x;
}

int pop() {
    return stack[top--];
}

int main() {
    char postfix[50];
    int i, a, b;

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

    for(i = 0; postfix[i]; i++) {
        if(isdigit(postfix[i]))
            push(postfix[i] - '0');
        else {
            b = pop();
            a = pop();
            if(postfix[i] == '+') push(a + b);
            if(postfix[i] == '-') push(a - b);
            if(postfix[i] == '*') push(a * b);
            if(postfix[i] == '/') push(a / b);
        }
    }

    printf("Result = %d", pop());
    return 0;
}
```

Ln 38, Col 1 Spaces: 4 UTF-8 CRLF { } C Win32

Output

```
Enter postfix expression: 23*54*+9-
Result = 17
```

Program 4: Prefix Expression Evaluation

Aim

To write a C program to evaluate a prefix expression using stack.

Program Code

Output

Result

All stack-based expression conversion and evaluation programs were successfully implemented and executed.