

DATA STRUCTURES

Division	CS(AIML) -A
Batch	2
GR-no	12311493
Roll no	54
Name	Atharva Kangralkar

Assignment 9:

**WAP to convert given Infix expression into its
Equivalent Prefix and Postfix form using
Stack.**

Code:-

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

```
#include <stdlib.h>
```

```
#include <string.h>
```

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

```
#define MAX 100
```

```
char stack[MAX];
```

```
int top = -1;
```

```
void push(char c) {
```

```
    stack[++top] = c;
```

```
}
```

```
char pop() {
```

```
    if (top == -1) return -1;
```

```
    return stack[top--];
```

```
}
```

```
char peek() {
```

```
    if (top == -1) return -1;
```

```
    return stack[top];
```

```
}
```

```
int precedence(char op) {
```

```
    if(op == '^') return 3;
```

```
    if(op == '*' || op == '/') return 2;
```

```
    if(op == '+' || op == '-') return 1;
    return 0;
}
```

```
int isOperator(char c) {
    return (c == '+' || c == '-' || c == '*' || c == '/' || c == '^');
}
```

```
void infixToPostfix(char* infix, char* postfix) {
    int i = 0, k = 0;
    char ch;
    while((ch = infix[i++]) != '\0') {
        if(isalnum(ch)) {
            postfix[k++] = ch;
        } else if(ch == '(') {
            push(ch);
        } else if(ch == ')') {
            while(peek() != '(') {
                postfix[k++] = pop();
            }
            pop()
        } else if(isOperator(ch)) {
            while(top != -1 && precedence(peek()) >= precedence(ch)) {
                postfix[k++] = pop();
            }
            push(ch);
        }
    }
}
```

```

while(top != -1) {
    postfix[k++] = pop();
}
postfix[k] = '\0';
}

```

```

void reverse(char* str) {
    int i, j;
    char temp;
    for(i = 0, j = strlen(str) - 1; i < j; i++, j--) {
        temp = str[i];
        str[i] = str[j];
        str[j] = temp;
    }
}

```

```

void infixToPrefix(char* infix, char* prefix) {
    char revInfix[MAX], revPostfix[MAX];
    int i;

    strcpy(revInfix, infix);
    reverse(revInfix);

    for(i = 0; revInfix[i] != '\0'; i++) {
        if(revInfix[i] == '(')
            revInfix[i] = ')';
        else if(revInfix[i] == ')')
            revInfix[i] = '(';
    }
}

```

```

    }

    top = -1;

    infixToPostfix(revInfix, revPostfix);

    reverse(revPostfix);

    strcpy(prefix, revPostfix);
}

int main() {
    char infix[MAX], postfix[MAX], prefix[MAX];

    printf("Enter infix expression: ");

    scanf("%s", infix);

    infixToPostfix(infix, postfix);

    infixToPrefix(infix, prefix);

    printf("Postfix: %s\n", postfix);

    printf("Prefix: %s\n", prefix);

    return 0;
}

```

Code Screenshot:-

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <string.h>
4  #include <ctype.h>
5
6  #define MAX 100
7
8  char stack[MAX];
9  int top = -1;
10
11 void push(char c) {
12     stack[++top] = c;
13 }
14
15 char pop() {
16     if (top == -1) return -1;
17     return stack[top--];
18 }
19
20 char peek() {
21     if (top == -1) return -1;
22     return stack[top];
23 }
24
25 int precedence(char op) {
26     if (op == '^') return 3;
```

```

26     if(op == '^') return 3;
27     if(op == '*' || op == '/') return 2;
28     if(op == '+' || op == '-') return 1;
29     return 0;
30 }
31
32 ~ int isOperator(char c) {
33     return (c == '+' || c == '-' || c == '*' || c == '/' || c == '^');
34 }
35
36 ~ void infixToPostfix(char* infix, char* postfix) {
37     int i = 0, k = 0;
38     char ch;
39 ~ while((ch = infix[i++]) != '\0') {
40 ~     if(isalnum(ch)) {
41         postfix[k++] = ch;
42 ~     } else if(ch == '(') {
43         push(ch);
44 ~     } else if(ch == ')') {
45 ~         while(peek() != '(') {
46             postfix[k++] = pop();
47         }
48         pop()
49 ~     } else if(isOperator(ch)) {
50 ~         while(top != -1 && precedence(peek()) >= precedence(ch)) {
51             postfix[k++] = pop();
52         }
53         push(ch);
54     }
55 }
56 ~ while(top != -1) {
57     postfix[k++] = pop();
58 }

```

```

59     postfix[k] = '\0';
60 }
61
62 void reverse(char* str) {
63     int i, j;
64     char temp;
65     for(i = 0, j = strlen(str) - 1; i < j; i++, j--) {
66         temp = str[i];
67         str[i] = str[j];
68         str[j] = temp;
69     }
70 }
71
72 void infixToPrefix(char* infix, char* prefix) {
73     char revInfix[MAX], revPostfix[MAX];
74     int i;
75
76     strcpy(revInfix, infix);
77     reverse(revInfix);
78
79     for(i = 0; revInfix[i] != '\0'; i++) {
80         if(revInfix[i] == '(')
81             revInfix[i] = ')';
82         else if(revInfix[i] == ')')
83             revInfix[i] = '(';
84     }
85
86     top = -1;
87     infixToPostfix(revInfix, revPostfix);
88     reverse(revPostfix);
89     strcpy(prefix, revPostfix);
90 }
91
92 int main() {

```



```

92 int main() {
93     char infix[MAX], postfix[MAX], prefix[MAX];
94
95     printf("Enter infix expression: ");
96     scanf("%s", infix);
97
98     infixToPostfix(infix, postfix);
99     infixToPrefix(infix, prefix);
100
101     printf("Postfix: %s\n", postfix);
102     printf("Prefix: %s\n", prefix);
103
104     return 0;
105 }
106

```

Output:-

Output
Preorder: 1 2 4 5 3 6 7
Inorder: 4 2 5 1 6 3 7
Postorder: 4 5 2 6 7 3 1
=== Code Execution Successful ===