

## Lab 02

TASK: WAP to convert a given valid parenthesized infix arithmetic expression to postfix expression. The expression consists of single character operands and the binary operators + , - , \* , / , ^

DATE: 06-10-25

```
Infix Expression: a+b*c-d/e+f*g/h^i
Postfix Expression: abc*+de/-fg*hi^/+
```

```
Infix Expression: (a+b*c/t(h*t/e-463))
Postfix Expression: abc*tht*e/463-/+
```

lab-03 06-10-25

WAP to Convert a given valid parenthesised infix arithmetic expression to post-fix expression. The expression consists of a single character operands and the binary operands +, -, \*, /, ^

// INFIX TO POSTFIX

#include <stdio.h>

#include <ctype.h>

#include <string.h>

#define MAX 100

char stack[MAX];

int top = -1;

// Push function

void push(char c) {

if (top == MAX - 1) {

printf("Stack Overflow");

return;

}

top++;

stack[top] = c;

}

```

//Pop function
char pop() {
    if (top == -1) {
        printf("Stack Underflow\n");
        return -1;
    }
    top = top - 1;
    return stack[top];
    char element-pop = stack[top];
    top--;
    return element-pop;
}

```

```

//Function to return precedence
int precedence(char op) {

```

```

    switch (op) {
        case '+':
        case '-':
            return 1;

```

```

        case '*':
        case '/':
            return 2;

```

```

        case '^':
            return 3;
        case '(':
            return 0;
    }

```

```

    }
    return -2;
}

// Function to return Associativity
// 0 = left to right, 1 = Right to Left
int associativity(char op) {
    if (op == "^") return 1;
    return 0; // +, -, *, /
}

```

```

// Function to convert infix to postfix
void infix-to-postfix(char infix[], char postfix[]) {
    int i, k = 0;
    char c;
    for (int i = 0; infix[i] != '\0'; i++) {
        c = infix[i];
        if (isalnum(c)) {
            // If operand direct to postfix
            postfix[k++] = c;
        }
    }
}

```

// Pop remaining operators.

while (top != -1) {

postfix[k++] = pop();

}

postfix[k] = "\0";

int main() {

char infix[MAX], postfix[MAX];

printf("Infix Expression: ");

scanf("%s", infix);

infix-to-postfix(infix, postfix);

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

return 0;

}

Output :-

~~\*\*\*A/B/C\*\*\*~~

Infix Expression: A+B/C\*D

Postfix Expression: ABC/D\*+

Infix Expression: (A+(B\*C-(D/E^F)\*G)\*H)

Postfix Expression: ABC\*DEF/G\*+H\*+

```
1  #include<stdio.h>
2  #include<ctype.h>
3  #include<string.h>
4
5
6  #define MAX 100
7
8  void push(char c);
9  char pop();
10 char peek();
11 int associativity(char operator);
12 int precedence(char operator);
13 void infix_to_postfix(char infix[], char postfix[]);
14
15
16 char infix[MAX];
17 char postfix[MAX];
18 char stack[MAX];
19 int top = -1;
20 int k = 0;
21
22 int main(void){
23
24     printf("Infix Expression: ");
25     scanf("%s", infix);
```

```
27     infix_to_postfix(infix, postfix);
28
29     printf("Postfix Expression: ");
30     printf("%s", postfix);
31 }
32
33 void push(char element){
34     if(top == MAX-1) return;
35     top++;
36     stack[top] = element;
37 }
38
39 char pop(){
40     char element = stack[top];
41     top--;
42     return element;
43 }
44
45
46 char peek(){
47     if (top == -1) return '\0';
48     return stack[top];
49 }
```

```
51 int precedence(char op) {
52     switch(op) {
53         case '+':
54         case '-':
55             return 1;
56         case '*':
57         case '/':
58             return 2;
59         case '^':
60             return 3;
61         default:
62             return -1; // for non-operators
63     }
64 }
65
66
67 int associativity(char op){
68     switch(op){
69         case '+':
70         case '-':
71         case '*':
72         case '/':
73             return 1;
74         case '^':
75             return 2;
76         default:
77             return -1;
78     }
79 }
80
81
```



```

82 void infix_to_postfix(char infix[], char postfix[]){
83     for(int i = 0; i < strlen(infix); i++){
84         char incoming = infix[i];
85
86         if (incoming == '(') push(incoming);
87         else if (isalnum(incoming)) postfix[k++] = incoming;
88
89         else if (incoming == ')'){
90             while((top != -1) && (peek() != '(')) postfix[k++] = pop();
91             pop();//to remove the (
92         }
93
94         else if (precedence(incoming) > precedence(peek())) push(incoming);
95
96         else{
97             while (top != -1 && ((precedence(incoming) < precedence(peek())) ||
98                 ((precedence(incoming) == precedence(peek()) && associativity(incoming) == 1))))
99             {
100                 postfix[k++] = pop();
101             }
102
103             push(incoming);
104         }
105     }
106
107 }
108
109 while(top != -1) postfix[k++] = pop();
110
111 postfix[k++] = '\0';
112
113 }

```

TASK: Check for balanced parentheses

DATE: 06-09-25

```
Expression: {[[(
```

```
Unbalanced
```

```
Expression: Expression: ([({{[]}}))
```

```
Balanced
```

```
Expression: [(this_is_an_example)]
```

```
Balanced
```

## Lab 02 Balancing Parentheses. 29/09/25

```
#include <stdio.h>
#include <string.h>
#define MAX 100

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

void push(char i);
void pop();
char peek();
int match(char top, char incoming);

int main() {
    char expression[MAX];
    printf("Expression: ");
    scanf("%s", &expression);

    for (int i = 0; i < strlen(expression); i++) {
        if (expression[i] == "(" ||
            expression[i] == "{" ||
            expression[i] == "[") {
            push(expression[i]);
        }
    }
```

```

if (expression[i] == '(' ||
    expression[i] == '[' ||
    expression[i] == '{') {
    char stack-top = peek();
    int i = check-match(stack-top, expression[i]);
    if (i == 1) {
        pop();
    }
    else {
        printf("Unbalanced");
    }
}
}

if (top == -1) {
    printf("Balanced");
}

return 0;

```

```
void push(char i) {  
    if (top == MAX-1) {  
        printf("Stack Overflow");  
    }  
    else {  
        top++;  
        stack[top] = i;  
        //  
    }  
}  
  
void pop() {  
    if (top == -1) {  
        printf("Underflow");  
    }  
    else {  
        return stack[top];  
    }  
}
```

```

int match(char top, char incoming) {
    if (top == "(" && incoming == ")") {
        return 1;
    }
    if (top == "[" && incoming == "]") {
        return 1;
    }
    return 0;
}

```

Output:-

Expression: [{}]

Balanced.

Expression: [{}]

UnBalanced.

```

1  #include <stdio.h>
2  #include <string.h>
3
4  #define MAX 100
5
6  char stack[MAX];
7  int top = -1;
8
9  void push(char i);
10 void pop();
11 char peek();
12 int match(char top, char incoming);
13
14 int main(){
15     char expression[MAX];
16     printf("Expression: ");
17     scanf("%s", &expression);
18
19     for(int i = 0; i < strlen(expression); i++){
20         if (expression[i] == '(' || expression[i] == '{' || expression[i] == '['){
21             push(expression[i]);
22         }
23
24         if (expression[i] == ')' || expression[i] == '}' || expression[i] == ']){
25             char stack_top = peek();
26
27             int j = match(stack_top, expression[i]);
28
29             if (j == 1){
30                 pop();
31             }
32
33             else{
34                 printf("Unbalanced");
35             }
36         }
37     }

```

```
38
39     if (top == -1){
40         printf("Balanced");
41     }
42
43     return 0;
44 }
45
46 //this is shrihari viswanahan program'
47
48 void push(char i){
49     if (top == MAX - 1){
50         printf("Stack Overflow");
51     }
52     else{
53         top++;
54         stack[top] = i;
55     }
56 }
57
58 void pop(){
59     if (top == -1){
60         printf("Stack underflow");
61     }
62     else{
63         int item = stack[top];
64         top--;
65     }
66 }
67
```



```
68 char peek(){
69     if (top == -1){
70         printf("Underflow");
71     }
72     else{
73         return stack[top];
74     }
75 }
76
77 int match(char top, char incoming){
78     if (top == '(' && incoming == ')'){
79         return 1;
80     }
81
82     if (top == '{' && incoming == '}'){
83         return 1;
84     }
85
86     if (top == '[' && incoming == '']{
87         return 1;
88     }
89
90     return 0;
91 }
92
```