

**WAP to convert a given valid parenthesized infix arithmetic expression to postfix expression. The expression consists of single character operands and the binary operators + (plus), - (minus), \* (multiply) and / (divide)**

```
#include <stdio.h>
#include <ctype.h>
#include <string.h>
#define MAX 100
char stack[MAX];
int top = -1;
void push(char c) {
if (top == MAX - 1) {
printf("Stack Overflow\n");
} else {
top = top + 1;
stack[top] = c;
}
}
char pop() {
char val;
if (top == -1) {
printf("Stack Underflow\n");
return -1;
} else {
val = stack[top];
top = top - 1;
return val;
}
}
char peek() {
if (top == -1)
return '\0';
return stack[top];
}
int precedence(char c) {
if (c == '+' || c == '-')
return 1;
if (c == '*' || c == '/')
return 2;
if(c=='^')return 3;
return 0;
}
void infixToPostfix(char infix[], char postfix[]) {
int i, k = 0;
char c;
for (i = 0; infix[i] != '\0'; i++) {
c = infix[i];
if (isalnum(c)) {
postfix[k] = c;
k = k + 1;
}
}
```

```
else if (c == '(') {
push(c);
}
else if (c == ')') {
while (top != -1 && peek() != '(') {
postfix[k] = pop();
k = k + 1;
}
pop();
}

else {
while (top != -1 && precedence(peek()) >= precedence(c)) {
postfix[k] = pop();
k = k + 1;
}
push(c);
}
}

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

int main() {
char infix[MAX], postfix[MAX];
printf("Enter a valid parenthesized infix expression: ");
scanf("%s", infix);
infixToPostfix(infix, postfix);
printf("Postfix Expression: %s\n", postfix);
return 0;
}
```

## Output

The screenshot shows the Programiz C Online Compiler interface. The code editor contains the following C code:

```
1 #include <stdio.h>
2 #include <ctype.h> // for isalnum()
3 #include <string.h> // for strlen()
4 #define MAX 100
5 char stack[MAX];
6 int top = -1;
7 // Function to push into stack
8 void push(char c) {
9 if (top == MAX - 1) {
10 printf("Stack Overflow\n");
11 } else {
12 top = top + 1;
13 stack[top] = c;
14 }
15 }
16 // Function to pop from stack
17 char pop() {
18 char val;
19 if (top == -1) {
20 printf("Stack Underflow\n");
21 return -1;
22 } else {
23 val = stack[top];
24 top = top - 1;
25 }
26 }
```

The output window displays the following results:

Enter a valid parenthesized infix expression: a\*(b+c)/d  
Postfix Expression: abc+\*d/  
== Code Execution Successful ==

The screenshot shows the Programiz C Online Compiler interface. The code editor contains the same C code as the first screenshot.

The output window displays the following results:

Enter a valid parenthesized infix expression: (a+b)\*(c-d)  
Postfix Expression: ab+cd-\*  
== Code Execution Successful ==

The screenshot shows the Programiz C Online Compiler interface. The code editor contains the following C code:

```
1 // Function to calculate value of expression
2 float calculate(char infix[], int i, int j) {
3     float val;
4     if (i > j) {
5         val = stack[j];
6     } else {
7         if (infix[i] == '+') {
8             val = calculate(infix, i + 1, j);
9         } else if (infix[i] == '-') {
10            val = calculate(infix, i + 1, j);
11        } else if (infix[i] == '*') {
12            val = calculate(infix, i + 1, j);
13        } else if (infix[i] == '/') {
14            val = calculate(infix, i + 1, j);
15        } else if (infix[i] == '^') {
16            val = calculate(infix, i + 1, j);
17        }
18    }
19    return val;
20 }
21 // Function to peek stack top
22 char peek() {
23 if (top == -1)
24 return '\0';
25 return stack[top];
26 }
27 // Function to convert infix to postfix
28 void infixToPostfix(char infix[], char postfix[]) {
29 int i, k = 0;
30 char c;
31 for (i = 0; infix[i] != '\0'; i++) {
32 if (infix[i] == ' ')
33 continue;
34 if (infix[i] == '+' || infix[i] == '-')
35 push(infix[i]);
36 if (infix[i] == '*' || infix[i] == '/')
37 push(infix[i]);
38 if (infix[i] == '^')
39 push(infix[i]);
40 }
41 for (i = 0; i < strlen(postfix); i++) {
42 if (postfix[i] == ' ')
43 continue;
44 if (postfix[i] == '+' || postfix[i] == '-')
45 pop(postfix[i]);
46 if (postfix[i] == '*' || postfix[i] == '/')
47 pop(postfix[i]);
48 if (postfix[i] == '^')
49 pop(postfix[i]);
50 }
51 }
```

The output window displays the following results:

Enter a valid parenthesized infix expression: 8-2+(3\*4)/2^2  
Postfix Expression: 82-34\*22^/+  
== Code Execution Successful ==