

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 (isdigit(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

Programiz

C Online Compiler

Programiz PRO

Never struggle with DSA again.
Learn with interactive visuals

Try Now

Programiz PRO >

main.c

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;

Run

Share

Output

Clear

Enter a valid parenthesized infix expression: a*(b+c)/d

Postfix Expression: abc*d/

=== Code Execution Successful ===

Programiz

C Online Compiler

Programiz PRO

Never struggle with DSA again.
Learn with interactive visuals

Try Now

Programiz PRO >

main.c

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;

Run

Share

Output

Clear

Enter a valid parenthesized infix expression: (a+b)*(c-d)

Postfix Expression: ab+cd-*

=== Code Execution Successful ===

Programiz

C Online Compiler

Programiz PRO

Never struggle with DSA again.
Learn with interactive visuals

Try Now

Programiz PRO >

main.c

23 val = stack[top];

24 top = top - 1;

25 return val;

26 }

27 }

28 // Function to peek stack top

29- char peek() {

30 if (top == -1)

31 return '\0';

32 return stack[top];

33 }

34 // Function to check precedence of operators

35- int precedence(char c) {

36 if (c == '+' || c == '-') return 1;

37 if (c == '*' || c == '/') return 2;

38 if(c=='^')return 3;

39 return 0;

40 }

41 // Function to convert infix to postfix

42- void infixToPostfix(char infix[], char postfix[]) {

43 int i, k = 0;

44 char c;

45- for (i = 0; infix[i] != '\0'; i++) {

46 c = infix[i];

Run

Share

Output

Clear

Enter a valid parenthesized infix expression: 8-2+(3*4)/2^2

Postfix Expression: 82-34*22^/+

=== Code Execution Successful ===