

```
infutopostfix.c - CodeBlocks 20.03
File Edit View Search Project Build Debug Fortran wxSmith Tools Tools Plugins DccyBlocks Settings Help
Start here X infutopostfix.c X
1 #include <stdio.h>
2 #include <ctype.h>
3 #include <string.h>
4 #define MAX 100
5 char stack[MAX];
6 int top=-1;
7 void push(char c){
8     if(top==MAX){
9         printf("Stack overflow");
10        return;
11    }
12    stack[++top]=c;
13}
14
15 char pop(){
16     if(top==0){
17         printf("Stack is underflow");
18         return -1;
19     }
20     return stack[top--];
21}
22
23 char peek(){
24     if(top==0) return -1;
25     return stack[top];
26}
27
28 int precedence(char op){
29     switch (op)
30     {
31         case '+':
32             return 1;
33         case '-':
34             return 1;
35         case '*':
36             return 2;
37         case '/':
38             return 2;
39         case '^':
40             return 3;
41     }
42     return -1;
43}
44
45 int ass(char op){
46     if (op=="^")
47         return 1;
48     return 0;
49}
50
51 void infixtopostfix(char infix[], char postfix[]){
52     int i,k=0;
53     char c;
54     for(i=0;infix[i]!='\0';i++){
55         c=infix[i];
56         if (isalnum(c)){
57             postfix[k++]=c;
58         }
59         else if (c=="("){
60             push(c);
61         }
62         else if (c=="") {
63             while (peek()!="("){
64                 postfix[k++]=pop();
65             }
66             pop();
67         }
68         else {
69             while (top!=-1 && (precedence(peek())>precedence(c))){
70                 postfix[k++]=pop();
71             }
72             push(c);
73         }
74     }
75     while (top!=-1){
76         postfix[k++]=pop();
77     }
78     postfix[k]='\0';
79 }
80
81 int main(){
82     char infix[MAX], postfix[MAX];
83     printf("Enter a valid parentheses infix exp:");
84     scanf("%s",infix);
85     infixtopostfix(infix,postfix);
86     printf("postfix exp is %s",postfix);
87     return 0;
88 }
```

```
infutopostfix.c - CodeBlocks 20.03
File Edit View Search Project Build Debug Fortran wxSmith Tools Tools Plugins DccyBlocks Settings Help
Start here X infutopostfix.c X
36         return 0;
37     case '+':
38         return 0;
39 }
40 return -1;
41 }
42
43 int ass(char op){
44     if (op=="^")
45         return 1;
46     return 0;
47}
48
49 void infixtopostfix(char infix[], char postfix[]){
50     int i,k=0;
51     char c;
52     for(i=0;infix[i]!='\0';i++){
53         c=infix[i];
54         if (isalnum(c)){
55             postfix[k++]=c;
56         }
57         else if (c=="("){
58             push(c);
59         }
60         else if (c=="") {
61             while (peek()!="("){
62                 postfix[k++]=pop();
63             }
64             pop();
65         }
66         else {
67             while (top!=-1 && (precedence(peek())>precedence(c))){
68                 postfix[k++]=pop();
69             }
70             push(c);
71         }
72     }
73     while (top!=-1){
74         postfix[k++]=pop();
75     }
76     postfix[k]='\0';
77 }
78
79 int main(){
80     char infix[MAX], postfix[MAX];
81     printf("Enter a valid parentheses infix exp:");
82     scanf("%s",infix);
83     infixtopostfix(infix,postfix);
84     printf("postfix exp is %s",postfix);
85     return 0;
86 }
87
88 }
```

```
infiswapfix.c - CodeBlocks 20.03
File Edit View Search Project Build Debug Fortran wxSmith Tools Tools+ Plugins DsxyBlocks Settings Help

Start here X infiswapfix.c X
1 #include<stdio.h>
2 #include<ctype.h>
3 #include<string.h>
4 #define MAX 100
5 char stack[MAX];
6 int top=-1;
7 void push(char c){
8     if(top==MAX){
9         printf("stack overflow");
10        return;
11    }
12    stack[++top]=c;
13 }
14
15 char pop(){
16     if(top==0){
17         printf("stack is underflow");
18         return -1;
19     }
20     return stack[top--];
21 }
22 char peek(){
23     if (top==0) return -1;
24     return stack[top];
25 }
26 int precedence(char op){
27     switch (op)
28     {
29         case '+':
30             return 1;
31         case '-':
32             return 1;
33         case '/':
34             return 2;
35         case '*':
36             return 3;
37         case '(':
38             return 0;
39     }
40     return -1;
41 }
42 int ans(char op){
43     if (op=='+'){
44         return 1;
45     }
46     return 0;
47 }
48 void infiswapfix(char infix[], char postfix[]){
49     int i,k=0;
50     char c;
51     for(i=0;infix[i]!='\0';i++){
52         c=infix[i];
53         if(isalnum(c)){
54             postfix[k++]=c;
55         }
56     }
57 }
58
59 int main(){
60     char infix[100], postfix[100];
61     printf("Enter a valid parenthesized infix exp:A*B+C*D-E\n");
62     gets(infix);
63     infiswapfix(infix, postfix);
64     printf("Postfix exp:AB*CD+E-+\n");
65     return 0;
66 }
```

D:\chethan\CPP\infiswapfix.exe

Enter a valid parenthesized infix exp:A*B+C*D-E
postfix exp:AB*CD+E-+
Process returned 0 (0x0) execution time : 267.259 s
Press any key to continue.

D:\chethan\CPP\infiswapfix.c C/C++ Windows (C/C++) WINDOWS-1252 Line 54, Col 1, Pos 974 Insert Read/Write default ENG IN 09:52 06-10-2025

Infix to postfix conversion

```
#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");
```

```
        return;
```

```
    }
```

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

```
}
```

```
char pop() {
```

```
    if (top == -1) {
```

```
        printf("stack underflow\n");
```

```
        return -1;
```

```
    }
```

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

```
}
```

```
char peek() {
```

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

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

```
}
```

WAP to convert a given valid parenthesized infix arithmetic expression to postfix expression. The expression consist of single character operands and the binary operators.

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

Algorithm:-

- 1) Start
- 2) Initialize an empty stack for operators
- 3) Initialize an empty string postfix
- 4) Scan the infix expression from left to right, one symbol at a time
- 5) If operand is '(' push to stack
- 6) If operand is ')' pop from stack and append the postfix until '(' open bracket
- 7) If operator (+, -, *, /)
 - * If operator have high precedence push to stack
 - * If operator have lower precedence append it to postfix, pop and print the top
 - * If incoming operator is equal precedence, then go to associativity laws.
- 8) At the end of the expression, pop and print all operators.
- 9) Stop

✓
6/6

```

else if (c == ')') {
    while (peek() != '(') {
        postfix[k++] = pop();
    }
    pop();
}
else {
    while (top != -1 &&
        ((precedence(peek()) > precedence(c)) ||
         (precedence(peek()) == precedence(c) &&
          associativity(c) == 0))) {
        postfix[k++] = pop();
    }
    push(c);
}
}
while (top != -1) {
    postfix[k++] = pop();
}
postfix[k] = '\0';
}

```

```

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

```


int precedence (char op) {
switch (op) {

case '+':

case '-':

return 1;

case '*':

case '/':

return 2;

case '^':

return 3;

case '(':

return 0;

}

return -1;

}

int associativity (char op) {

if (op == '^')

return 1;

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;

}

else if (c == '(') {

push(c);

}

Output:

Enter a valid Parenthesized infix exp: $A*B-C+D+E$
possible exp: $AB*C-DE+1$

✓
4/12

Queue:

- 1) Define an array `queue[Max]` to hold elements.
- 2) set two integer variables
 $front = -1$
 $rear = -1$ to indicate the queue is empty
- 3) check if $rear == Max-1$
print queue overflow
- 4) If $front == -1$, set $front = 0$
- 5) Increment $rear$ by 1 $\rightarrow rear = rear + 1$
- 6) Insert the new element $\rightarrow queue[rear] = item$
- 7) stop

Dequeue

- 1) start
- 2) check if $front == -1$ or $front > rear$
print queue underflow
- 3) Remove the element $\rightarrow item = queue[front]$
- 4) $front = front + 1$ Increment
- 5) stop