Infix to postfix

0 WAP to convert a given valid parenthesized infix arithmetic expression to Postfix expression.

Pseudo code:

```
Infix to postfix(exp)
{
  Create a stacks
  {
    if exp[i] is operand then
        res ← res + exp[i]
    else if exp[i] is operator (+,-,*,/,^) then
    {
      while (!S.empty() && S has higher pre(S.top(), exp[i]))
      {
        res ← res + !S.top()
        S.pop()
      }
      S.push(exp[i])
    }
  else if IS opening parenthesis (exp[i]) then
        S.push(exp[i])
  else if IS closing parenthesis (exp[i]) then
    {
      while (!S.empty() && ! Is opening parenthesis (S.top())
      {
        res ← res + S.top()
        S.pop()
      }
      S.pop()
    }
  }
  while (!S.empty())
    {
      res ← res + S.top()
      S.pop
    }
```

Code :→

```
#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");
        return;
    }
    stack[++top] = c;
}

char pop(){
    if(top == -1){
        printf("Stack is empty\n");
        return -1;
    }
    return stack[top--];
}

char peek(){
    if(top == -1)
        return -1;
    return stack[top];
}
```

```
int precedence(char op){
    switch(op){
        case '+':
        case '-':
            return 1;
        case '*':
        case '/':
            return 2;
        default:
            return 0;
    }
}
int associativity(char op){
    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);
        }
        else if(c==')'){
            while(top!=-1 && 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:");
scanf("%s", infix);

infixTopostfix(infix, postfix);
printf("postfix expression is %s", postfix);
    return 0;
    }
```

output:- Enter a valid parenthesized infix expression:

6*7/5+4-7

postfix expression is 67*5/4+7-

14/10