```c
/*
Convert ((A – (B + C)) * D) @ (E + F) infix expression to postfix form:
Answer is    A B C + - D * E F + @

Convert a + b * c + (d * e + f) * g the infix expression into postfix form.
Answer is   a b c * + d e * f + g * +

Convert the following infix expression A + B * C – D / E * H into its equivalent postfix
expression.
Answer is   A B C * + D E / H * -

Convert the following infix expression A + (B * C – (D / E @ F) * G) * H into its
equivalent postfix expression.
Answer is   A B C * D E F @ / G * - H * +
*/
#include<stdio.h>
#define SIZE 50            /* Size of Stack */
#include <ctype.h>
char stack[SIZE];
int top=-1;       /* Global declarations */

void push(char token)
{                        /* Function for PUSH operation */
    stack[++top]=token;
}

char pop()
{                       /* Function for POP operation */
    return(stack[top--]);
}

 /* Function for precedence */
int precedence(char token) // Higher the return value, higher is the precedence
{
    switch(token)
    {
        case '#': return 0;
        case '(': return 1;
        case '+':
        case '-': return 2;
        case '*':
        case '/':
        case '%': return 3;
        case '@': return 4; // '@' is being used for exponential operator
    }
}

int main()
{                         /* Main Program */
    char infix[50],postfix[50],ch,token;
    int i=0,k=0;
    printf("\n Input the Infix Expression with spaces in between: ");
    //scanf("%s",infix); //Input the infix expression without spaces
    gets(infix);//Input the infix expression with spaces in between

    push('#'); // # is being inserted in the stack to mark the end of the stack
    while( (ch=infix[i++]) != '\0')
    {
        if (ch==' ') continue;
        else if( ch == '(' ) push(ch); // add ( to stack
        else if(isalnum(ch)) postfix[k++]=ch; // add operand to postfix expression
        /* if ) bracket is encountered, repeatedly pop the stack and add it to postfix
        until ( bracket is found. Finally, remove the ( bracket from the stack by
        popping it.
        */
        else if( ch == ')' )
                {
                    while( stack[top] != '(')
                        {
                            postfix[k++]=pop(); // repeatedly pop and add to postfix
                        }
                    token=pop(); /* Remove ( from the stack*/
                }
        else
            {
            /* if the incoming symbol is an Operator:-
            if the precedence of incoming operator is greater than the precedence of
            the operator at the top of the stack, simply add it to the stack.
            But, if the precedence of incoming operator is less than or equal to the
            precedence of the operator at the top of the stack, repeatedly pop the
            operators from the stack and add to postfix expression until the precedence
            of the incoming operator becomes greater than the operator at the top of
            the stack or left bracket ( is found at the top of the stack.
            After that, simply add the incoming operator to the stack.
             */
                 while( precedence(ch)<=precedence(stack[top]) )
                  {
                         postfix[k++]=pop();
                  }
                push(ch);// Incoming operator is anyway pushed to the stack
            }
    }

    while( stack[top] != '#') /* Pop from stack till stack is empty */
    {
                postfix[k++]=pop();
    }

    postfix[k]='\0'; /* Make postfix as valid string by inserting '\0' at its end */
    printf("\n\nGiven Infix Expn: %s \n\nPostfix Expn: %s\n",infix,postfix);
    return 0;
}
```