```
//WAP to implement infix to postfix conversion using stack

#include <stdio.h>
#include <limits.h>
#include <stdlib.h>
#define MAX 20

char stk[20];
int top = -1;

int isEmpty(){
   return top == -1;
}

int isFull(){
   return top == MAX - 1;
}

char peek(){
   return stk[top];
}

char pop(){
   if(isEmpty())
      return -1;
   char ch = stk[top];
   top--;
   return(ch);
}
```

```
void push(char oper){
   if(isFull())
      printf("Stack Full!!!!");
   else{
      top++;
      stk[top] = oper;
 }
}

//Function to check if the given character is operand
int checkIfOperand(char ch)
{
   return (ch >= 'a' && ch <= 'z') || (ch >= 'A' && ch <= 'Z') || (ch >='0' && ch<='9');
}

// Fucntion to compare precedence
int precedence(char ch)
{
   switch (ch)
   {
   case '+':
   case '-':
      return 1;
   case '*':
   case '/':
      return 2;
   case '^':
      return 3;
   }
   return -1;
}
```

```
int covertInfixToPostfix(char* expression)
{
   int i, j;
   for (i = 0, j = -1; expression[i]; ++i)
   {
      //Checking if the character is operand or not and adding to the output
      if (checkIfOperand(expression[i]))
         expression[++j] = expression[i];

      //If character is '(', we need push it to the stack
      else if (expression[i] == '(')
         push(expression[i]);

      //If character is ')', we need to pop and print from the stack
      //Do this until an '(' is encountered in the stack.
      else if (expression[i] == ')')
      {
         while (!isEmpty() && peek() != '(')
            expression[++j] = pop();
         if (!isEmpty() && peek() != '(')
            return -1; // invalid expression
         else
            pop();
      }
      else // if an opertor
      {
         while (!isEmpty() && precedence(expression[i]) <= precedence(peek()))
            expression[++j] = pop();
         push(expression[i]);
      }
```

```
    }

    //Once all inital expression characters are traversed
    //Adding all elements from stack to expression
    while (!isEmpty())
        expression[++j] = pop();
    expression[++j] = '\0';
    printf( "Final postfix expression is: %s", expression);
}

int main()
{
    char expression[100];
    printf("\n");
    printf("Enter the infix expression: ");
    scanf("%s",expression);
    printf("\n");
    covertInfixToPostfix(expression);
    return 0;
}
```