① Write a program to convert a given valid parenthesized infix arithmetic expression to postfix expression. The valid expression consists of single character operands & the binary operand operator +[plus], -(minus), * Multiply, /(divide) & ^ (Power).

```
#include <stdio.h>
#include <stdlib.h>
#define max_size 100
char stack[max_size];
int top = -1;
void push(char item){
  if(top == max_size - 1){
    printf("stack overflow\n");
    exit(1); }
  stack[++top] = item; }

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

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

```
void infixTopostfix(char infix[]){
  char postfix [max_size];
  int i,j = 0;
for (i = 0; infix[i] != '\0'; i++){
  if (isalnum(infix[i])){
      postfix[j++] = infix[i];}
else if (infix[i] == '('){
  push(infix[i]);}
else if (infix[i] == ')'){
  while(top != -1 && stack[top] != '('){
      postfix[j++] = pop();}
  if (top != -1 && stack[top] == '('){
      pop();
  }else {
      printf("Invalid Expression\n");
      exit(1);}}
else if (isoperator(infix[i])){
  while (top != -1 && precedence(stack[top]) >=
precendence(infix[i])){
  postfix[j++] = pop();}
push(infix[i]);}

else {
printf("Invalid character\n");
    exit(1);

  }
}
```

```
while (top != -1) {
  if (stack[top] == '(') {
    Printf ("Invalid expression\n");
    exit(1);
  }
  postfix [j++] = pop();}
  postfix [j] = '\0';
  Printf("Postfix Expression : %s\n", postfix);
}

int main() {
  char infix[max_size];
  Printf("Enter a infix expression : ");
  scanf ("%s", infix);
  infixtopostfix(infix);
  return 0;
}
```

output:

Enter a valid parenthesized infix arithmetic expression : (A+B) * (C+D)

Postfix Expression : AB+CD++