2) WAP to convert a given valid parenthesized infix arithmatic expression to postfix exp. The exp consists of single character operands and the binary operations + (plus), - (minus) * (multiply) and / (divide)

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
int F (char symbol) {
    switch (symbol) {
      case '+' :
      case '-' : return 2;
      case '*' :
      case '/' : return 4;
      case '^' :
      case '$' : return 5;
      case '(' : return 0;
      case '#' : return -1;
      default : return 8;
}}

int G(char symbol) {
    switch(symbol) {
  case ):
  case (:
```

```
case '+' :
case '-' : return 1;
case '*' :
case '/' : return 3;
case '^' :
case '$' : return 6;
case '(' : return 9;
case ')' : return 0;
default : return 7;
}
}

void infix_postfix(char infix[]) {
int top, j, i;
char s[30], postfix[30];
char symbol;
top = -1;
s[++top] = '#';
j = 0
for (i=0; i < strlen(infix); i++) {
    symbol = infix[i];
    while (f(s[top]) > g(symbol)) {
        postfix[j] = s[top--];
        j++;
    }
else
if (f(s[top]) != g(symbol)) {
    s[++top] = symbol;
```

```
else
    top--;
}
while (s[top] != '#') {
    postfix[j++] = s[top--];
}
Postfix[j] = '\0';
 puts(postfix);
}

int check(char infix[])
int i, c1=0, c2=0;
 for(i=0; i<strlen(infix); ++i)
  {
     char a;
     a = infix[];
      if(a=='(')
         c1++;
      if(a==')')
       return
       c2++;
  }
   if(c1!=c2)
     return -1;
   else
     return 0;
}
```

```
int main()
{ char exp[30];
  int c;
  printf("enter an expression");
  gets(exp);
   c = check(exp);
  if(c==-1)
     puitf("Invalid input");
  else
   infix_postfix(exp);
  return 0;
}
```