5/10/24

2) Infix to Postfix expression, given the expression should contain binary operators +, -, *, /

Pseudo Code

```
Declare index, pos, top, length as int
Declare symbol, temp as char
Declare infix[30], postfix[30], stack[30] as string

Function infixtopostfix()
  set length = length of infix
  call push

while index < length
  symbol = infix[index]

switch symbol
  case '(': call push(symbol)
  case ')': temp = call pop()
  while temp != '('
    postfix[pos] = temp
    increment pos
    temp = call pop()
  end while
  call push(symbol)
default: postfix[pos] = symbol
increment pos
end switch
increment index
end while
```

```
while top > 0
    temp = call pop()
    postfix[pos] = temp
    pos++
end while
postfix[pos] = '\0'
end function

void push(char symbol)
    top++
    stack[top] = symbol

char pop()
 char symb = stack[top];
 top--
 return symb

int precedence(char symbol){
  int P;
  switch(symbol)
  case '^' : P=3
  case '*' : P=2
  case '+', '-' : P=1
  case '(' : P=0
  case '#' : P=-1
end switch
return P
end function
```

Code:

```
#include <stdio.h>
#include <conio.h>
#include <string.h>

int index = 0, pos = 0, top = -1, length;
char symbol, temp, infix[30], postfix[30],
stack[30];

void infixtopostfix();
void push(char symbol);
char pop();
int precedence(char symb);

int main() {
  printf("Enter infix expression:\n");
  scanf("%s", infix);

  infixtopostfix();
  printf("\n Infix expression:\n%s", infix);
  printf("\n Postfix expression:\n%s", postfix);
  return 0;
}

void infixtopostfix() {
  length = strlen(infix);
  push('#');
  while(index < length) {
    symbol = infix[index];
```

```
switch (symbol) {
  case '(':
    push (symbol);
    break;
  case ')':
    temp = pop();
    while ( temp != '(' ) {
      postfix [ pos++] = temp ;
      temp = pop();
    }
    break ;
  case '+';
  case '-' :
  case '*' :
  case '/' :
  case '^' :
    while ( precedence ( stack [ top ]) >= precedence (symbol){
      temp = pop ();
      postfix [ pos++] = temp ; }
    push ( symbol) ;
    break;
  default:
      postfix [ pos++] = symbol ; }
index ++ ; }

while ( top > 0) {
  temp = pop ();
  postfix [pos++] = temp; }
  postfix [ pos] = '\0' ;
}
```

```
void push (char symbol) {
    top = top + 1;
    stack [top] = symbol;

char pop () {
    char symb = stack [top];
    top --;
    return symb;
int precedence (char symbol) {
    int P;
    switch (symbol) {
    case '^': P = 3;
              break;
    case '*':
    case '/': P = 2
              break;
    case '+':
    case '-': P = 1;
              break;
    case '(': P = 0;
              break;
    case '#': P = -1;
              break;
    }
    return P;
}
```

Output

Enter infix expression
a^bc-d+e(f(g+h)

Postfix expression
abc^d-ef(gh+(+