Lab-03 [Infix to postfix conversion]

⊛ Pseudocode:

```
1)      Function F(symbol as input parameter)
   => Switch case(symbol)
   a) if '+' or '-' return 2
   b) if '*' or '/' return 4
   c) if '^' or '$' return 5
   d) if '(' return 0
   e) if '#' return -1
   f) other values return 8
2)      Function G(symbol as input parameter)
   => Switch case(symbol)
   a) if '+' or '-' return 1
   b) if '*' or '/' return 3
   c) if '^' or '$' return 6
   d) if '('        return 9
   e) if ')'        return 0
   f) other values return 7

3) Function infix_postfix (infix and postfix string as
   parameter)
      a) set top = -1
      b) set stack [0] as # by incrementing top to 0
```

```
c) For each symbol in infix expression:
d) ~~si~~ set symbol = infix[i]
e) Then in while loop if F(S[top]) > G(symbol)
       remove and place in postfix
f) If F(S[top]) != G(symbol)
      stack precedence    input precedence
      push symbol onto stack
                          S[++top] = symbol
g) else decrement top
          top --
h) pop remaining symbols in stack and place
   in postfix while S[top] != '#'.
      postfix[j++] = S[top--]

4) Main function
a) Print enter input expression
b) Scan expression
c) call function infix_postfix (pass infix and postfix string)
d) print postfix expression.
```

Lab-3 Infix to Postfix conversion.

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

→ Main code :-

```
        # 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 '-': return 1;
  case '*':
  case '/': return 3;
  case '^':
  case '$': return 6;
  case '(': return 9;
  case ')': return 0;
  default : return 7;
  }
}
void infix-postfix(char infix[]){
    int top, j, i;
    char s[30], postfix[30];
    char symlol;
    top = -1;
    s[++top] = '$';
    j=0;
    for(i=0; i< strlen(infix); i++){
            symbol=infix[i];
    while (F(S[top]) >G (symbol)){
          postfix [j]=S[top--];
          j++;
    }
    if (F(S[top])!=G(symbol)){
            S[++top]=symbol;
    }
    else
          top--;
  }
  while (S[top]!='$'){
     postfix [j++]=s[top--];
  }
  postfix [j]='\0';
  printf ("Postfix expression is : \n");
  puts( postfix);
}
int main()
{
  char exp [30];
  printf ("Enter an expression : \n"];
  gets (exp];
  infix_postfix (exp);
  return0;
}
```