```
//Created By Ritwik Chandra Pandey on 22/03/2021
//183215
//Infix To Postfix using stack

#include<stdio.h>
#include<stdlib.h>      /* for exit() */
#include<ctype.h>     /* for isdigit(char ) */
#include<string.h>

#define SIZE 100


char stack[SIZE];
int top = -1;

/* define push operation */

void push(char item)
{
   if(top >= SIZE-1)
   {
      printf("\nStack Overflow.");
   }
   else
   {
      top = top+1;
      stack[top] = item;
   }
}
```

```
/* define pop operation */
char pop()
{
   char item ;

   if(top <0)
   {
      printf("stack under flow: invalid infix expression");
            exit(1);
   }
   else
   {
      item = stack[top];
      top = top-1;
      return(item);
   }
}

int is_operator(char symbol)
{
   if(symbol == '^' || symbol == '*' || symbol == '/' || symbol == '+' || symbol =='-')
   {
      return 1;
   }
   else
   {
      return 0;
   }
}

int precedence(char symbol)
{
```

```
    if(symbol == '^')/* exponent operator, highest precedence*/
    {
       return(3);
    }
    else if(symbol == '*' || symbol == '/')
    {
       return(2);
    }
    else if(symbol == '+' || symbol == '-')         /* lowest precedence */
    {
       return(1);
    }
    else
    {
       return(0);
    }
}

void InfixToPostfix(char infix_exp[], char postfix_exp[])
{
    int i, j;
    char item;
    char x;

    push('(');                          /* push '(' onto stack */
    strcat(infix_exp,")");                  /* add ')' to infix expression */

    i=0;
    j=0;
    item=infix_exp[i];         /* initialize before loop*/

    while(item != '\0')        /* run loop till end of infix expression */
```

```
{
   if(item == '(')
   {
      push(item);
   }
   else if( isdigit(item) || isalpha(item))
   {
      postfix_exp[j] = item;            /* add operand symbol to postfix expr */
      j++;
   }
   else if(is_operator(item) == 1)       /* means symbol is operator */
   {
      x=pop();
      while(is_operator(x) == 1 && precedence(x)>= precedence(item))
      {
         postfix_exp[j] = x;              /* so pop all higher precendence operator and */
         j++;
         x = pop();                    /* add them to postfix expresion */
      }
      push(x);
            push(item);              /* push current oprerator symbol onto stack */
   }
   else if(item == ')')        /* if current symbol is ')' then */
   {
      x = pop();                 /* pop and keep popping until */
      while(x != '(')               /* '(' encounterd */
      {
         postfix_exp[j] = x;
         j++;
         x = pop();
      }
   }
```

```
        else
        { /* if current symbol is neither operand not '(' nor ')' and nor
             operator */
           printf("\nInvalid infix Expression (Invalid Symbol in infix operation).\n");
          exit(1);
        }
        i++;


        item = infix_exp[i]; /* go to next symbol of infix expression */
    } /* while loop ends here */
    if(top>0)
    {
        printf("\nInvalid infix Expression : Unbalanced Parenthesis.\n");        /* the it is illegeal  symbol */
        getchar();
        exit(1);
    }


    postfix_exp[j] = '\0';}

/* main function begins */
int main()
{
    char infix[SIZE], postfix[SIZE];         /* declare infix string and postfix string */

    /* why we asked the user to enter infix expression
     * in parentheses ( )
     * What changes are required in porgram to
     * get rid of this restriction since it is not
     * in algorithm
     * */
```

```
    printf("ASSUMPTION: The infix expression contains single letter variables and single digit constants only.\n");
    printf("\nEnter Infix expression : ");
    gets(infix);

    InfixToPostfix(infix,postfix);              /* call to convert */
    printf("Postfix Expression: ");
    puts(postfix);                  /* print postfix expression */

    return 0;
}
```