NAME: AGNIV PRAMANICK **SECTION:** A **USN:** 1NT21IS017 **DATE:** 29/12/22

Q. Write a program to commit an infix to prefix conversion.

THEORY

Infix Expression:

When the operator is written in between the operands, then it is known as **infix notation**. Operand does not have to be always a constant or a variable; it can also be an expression itself.

Example:
(A - B) / (C - D)

Infix notation's syntax is:
<operand> <operator> <operand>

Prefix Expression:

Another way to describe anything is with a prefix notation, which does not require knowledge about precedence or associativity but does when used with an infix notation. It is also known as **polish notation**. In prefix notation, an operator comes before the operands.

The syntax of prefix notation is given below:
<operator> <operand> <operand>

Example:

a b - c d + /

ALGORITHM

Step-1: First, reverse the infix expression given in the problem.
Step-2: Scan the expression from left to right.
Step-3: Whenever the operands arrive, print them.
Step-4: If the operator arrives and the stack is found to be empty, then simply push the operator into the stack.
Step-5: If the incoming operator has **higher precedence than the TOP of the stack, push the incoming operator into the stack.**
Step-6: If the incoming operator has **the same precedence with a TOP of the stack, push the incoming operator into the stack.**
Step-7: If the incoming operator has **lower precedence than the TOP of the stack, pop, and print the top of the stack.** Test the incoming operator against the top of the stack again and **pop the operator from the stack till it finds the operator of a lower precedence or same precedence.**
Step-8: If the incoming operator has the **same precedence with the top of the stack and the incoming operator is ^, then pop the top of the stack till the condition is true. If the condition is not true, push the ^ operator.**
Step-9: When we reach the end of the expression, pop, and print all the operators from the top of the stack.
Step-10: If the operator is ')', then push it into the stack.
Step-11: If the operator is '(', then pop all the operators from the stack till it finds) opening bracket in the stack.
Step-12: If the top of the stack is ')', push the operator on the stack.

Step-13: At the end, reverse the output.

CODE

```
# include <stdio.h>
# include <string.h>    //library functions inserted
#include<ctype.h>
# define MAX 100   //maximum size is 100

//declaring globally all the user defined functions.

void infixtoprefix(char infix[20],char prefix[20]);     //to convert
void reverse(char array[30]);                          // to reverse
char pop();                                    // to pop
void push(char symbol);                                //to push
int isOperator(char symbol);
int prcd(symbol);
int top=-1;
char stack[MAX];  // array named stack is defined

// declaring the main function (code starts from here)
int main()
{
 char infix[20],prefix[20],temp;
 printf("Enter infix operation: ");
 gets(infix);
 infixtoprefix(infix,prefix);
 reverse(prefix);
 printf("the prefix expression is %s",prefix);       // prefix expression is printed
}
//-----------------------------------------------------------------

void infixtoprefix(char infix[20],char prefix[20])
{
 int i,j=0;
 char symbol;
 stack[++top]='#';
 reverse(infix);                                //function to reverse the string is called
 for (i=0;i<strlen(infix);i++)
 {
  symbol=infix[i];
  if (isOperator(symbol)==0)
  {
   prefix[j]=symbol;
   j++;
  }
  else
  {
   if (symbol==')')
   {
    push(symbol);  //push operator used to call
```

```
   }
   else if(symbol == '(')
   {
    while (stack[top]!=')')
    {
     prefix[j]=pop();       //pop function called
     j++;     //j value incremented
    }
   pop();
   }
   else
   {
   if (prcd(stack[top])<=prcd(symbol))      //if stack top precedence is less or equal
    {
     push(symbol);
    }
   else
    {
     while(prcd(stack[top])>=prcd(symbol))   //if stack top precedence is more
     {
      prefix[j]=pop();
      j++;                                  //j increment
     }
     push(symbol);       //push function called
    }
//end for else
   }
  }
//end for else
 }
//end for for
while (stack[top]!='#')
 {
  prefix[j]=pop();
  j++;
 }
 prefix[j]='\0';
}
////-----------------------------------------------------
void reverse(char array[30])                    //here reverse function is defined
{
 int i,j;
 char temp[100];
 for (i=strlen(array)-1,j=0;i+1!=0;--i,++j)
 {
  temp[j]=array[i];
 }
 temp[j]='\0';
 strcpy(array,temp);               //string is copied
 return array;
}
//--------------------------------
```

```
char pop()
{
 char a;
 a=stack[top];
 top--;                    //top is decremented
 return a;
}
//-----------------------------------
void push(char symbol)          //push is defined as function
{
 top++;
 stack[top]=symbol;
}
//-------------------------------------------
int prcd(symbol)          //this function is defined to check the precedence of the symbol
{
 switch(symbol)
 {
  case '+':
  case '-':
          return 2;
          break;
  case '*':
  case '/':
          return 4;
          break;
  case '$':
  case '^':
          return 6;
          break;
  case '#':
  case '(':
  case ')':
          return 1;
          break;
 }
}
//--------------------------------------------------
int isOperator(char symbol)                //isOperator used to check whether it is an operator or not.
{
 switch(symbol)           //cases allotted based on symbol type
 {
  case '+':
  case '-':
  case '*':
  case '/':
  case '^':
  case '$':
  case '&':
  case '(':
  case ')':
          return 1;
```

```
        break;
  default:
        return 0;
// returns 0 if the symbol is other than given above
 }
}
// END OF PROGRAM
```

output