Write a program to convert an expression from infix to postfix expression.(reverse polish)

Code:

```
#include <stdio.h>
#include <conio.h>
#include <ctype.h>
#include <string.h>
#include <stdlib.h>
#define MAX 100

char st[MAX];
int top = -1;
void push(char st[], char);
char pop(char st[]);
void InfixtoPostfix(char source[], char target[]);
int getPriority(char);
int main()
{
  char infix[100], postfix[100];
  printf("\n Enter any infix expression : ");
  gets(infix);
  strcpy(postfix, "");
  InfixtoPostfix(infix, postfix);
  printf("\n Postfix Expression is : ");
  puts(postfix);
  return 0;
}
void push(char st[], char val)
{
  if (top == MAX - 1)
  {
    printf("\n Stack Overflow");
  }
  else
  {
    top++;
    st[top] = val;
  }
}
int getPriority(char op)
{
  if (op == '/' || op == '*' || op == '%')
    return 1;
  else if (op == '+' || op == '-')
    return 0;
}
```

```
char pop(char st[])
{
  char val = ' ';
  if (top == -1)
  {
    printf("\n STACK UNDERFLOW");
  }
  else
  {
    val = st[top];
    top--;
  }
  return val;
}
void InfixtoPostfix(char source[], char target[])
{
  int i = 0, j = 0;
  char temp;
  strcpy(target, "");
  while (source[i] != '\0')
  {
    if (source[i] == '(')
    {
      push(st, source[i]);
      i++;
    }
    else if (source[i] == ')')

    {
      while ((top != -1) && (st[top] != '('))
      {
        target[j] = pop(st);
        j++;
      }
      if (top == -1)
      {
        printf("\n INCORRECT EXPRESSION");
        exit(1);
      }
      temp = pop(st); // remove left parenthesis
      i++;
    }
    else if (isdigit(source[i]) || isalpha(source[i]))
    {
      target[j] = source[i];
      j++;
```

```
      i++;
    }
    else if (source[i] == '+' || source[i] == '-' || source[i] == '*'
||
             source[i] == '/' || source[i] == '%')
    {
      while ((top != -1) && (st[top] != '(') && (getPriority(st[top])
>= getPriority(source[i])))
      {
        target[j] = pop(st);
        j++;
      }
      push(st, source[i]);
      i++;
    }
    else
    {
      printf("\n INCORRECT ELEMENT IN EXPRESSION");
      exit(1);
    }
  }
  while ((top != -1) && (st[top] != '('))
  {
    target[j] = pop(st);

    j++;
  }
  target[j] = '\0';
}
```

Output:

```
PS C:\Users\Ajay kumar\Desktop\SEIT-B\DSA\Lab\8 - Infix Postfix>

 Enter any infix expression : (2+4)-4*(23*4)

 Postfix Expression is : 24+4234**-

PS C:\Users\Ajay kumar\Desktop\SEIT-B\DSA\Lab\8 - Infix Postfix>
```

1.Write pseudocode to Convert the given expression from infix to postfix

Answer:

1. Create an empty stack for operators and an empty string for the result.
2. Initialize an empty list to store the postfix expression.
3. Scan the infix expression from left to right.
4. For each character in the infix expression:
    a. If it is an operand (letter or number), append it to the result string.
    b. If it is an open parenthesis '(', push it onto the operator stack.
    c. If it is a close parenthesis ')', pop operators from the stack and append them to the result until an open parenthesis is encountered. Pop and discard the open parenthesis.
    d. If it is an operator (+, -, *, /, %), pop operators from the stack and append them to the result until an operator with lower precedence or an open parenthesis is encountered. Then push the current operator onto the stack.
5. After scanning the infix expression, pop any remaining operators from the stack and append them to the result.
6. The result string is the postfix expression.

2.Convert the given expression from infix to postfix (Show all step of conversion)
A – (B / C + (D % E * F) / G)* H

Answer:

Step 1: Expression: A Operators Stack: (empty) Postfix Expression: A

Step 2: Expression: - Operators Stack: - Postfix Expression: A

Step 3: Expression: (Operators Stack: - (Postfix Expression: A

Step 4: Expression: B Operators Stack: - (Postfix Expression: A B

Step 5: Expression: / Operators Stack: - / Postfix Expression: A B

Step 6: Expression: C Operators Stack: - / Postfix Expression: A B C

Step 7: Expression: + Operators Stack: - + Postfix Expression: A B C /

Step 8: Expression: (Operators Stack: - + (Postfix Expression: A B C / +

Step 9: Expression: D Operators Stack: - + (Postfix Expression: A B C / + D

Step 10: Expression: % Operators Stack: - + % (Postfix Expression:
A B C / + D E F *

Step 11: Expression:) Operators Stack: - + Postfix Expression: A B C / + D E F * %

Step 12: Expression: / Operators Stack: - / Postfix Expression: A B C / + D E F * % +

Step 13: Expression: G Operators Stack: - Postfix Expression: A B C / + D E F * % +
G

Step 14: Expression:) Operators Stack: (empty) Postfix Expression:
A B C / + D E F * % + G /

Step 15: Expression: * Operators Stack: * Postfix Expression:
A B C / + D E F * % + G / -

Step 16: Expression: H Operators Stack: * - Postfix Expression:
A B C / + D E F * % + G / - H

Step 17: Operators Stack: (empty) Postfix Expression: A B C / + D E F * % + G / - H

* The final postfix expression is: A B C / + D E F * % + G / - H *