INFIX TO POSTFIX CONVERSION

Exp. No.:
AIM:

ALGORITHM:

PROGRAM:

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

#define arrSize 100

struct node
{
   unsigned int size;
   int top;
   char *arr;
};
typedef struct node Stack;

Stack *createStack(unsigned int size)
{
   Stack *stack = (Stack *)malloc(sizeof(Stack));
   stack->size = size;
   stack->top = -1;
   stack->arr = (char *)malloc(size * sizeof(char));
   return stack;
}

void push(Stack *stack, char elem)
{
   stack->top++;
   stack->arr[stack->top] = elem;
}

char pop(Stack *stack)
{
   char temp;
   if (stack->top == -1)
   {
      return '\0';
```

```
    }
    else
    {
        temp = stack->arr[stack->top];
        stack->top--;
        return temp;
    }
}

int precedence(char ch)
{
    if (ch == '^')
    {
        return 3;
    }
    else if (ch == '*' || ch == '/')
    {
        return 2;
    }
    else if (ch == '+' || ch == '-')
    {
        return 1;
    }
    else
    {
        return 0;
    }
}

int operator(char ch)
{
    if (ch == '(' || ch == ')' || ch == '[' || ch == ']' || ch == '{' || ch == '}')
    {
        return 2;
    }
    else if (ch == '+' || ch == '-' || ch == '*' || ch == '/' || ch == '^')
    {
        return 1;
    }
```

```
    else
    {
        return 0;
    }
}

void infixToPostfix(char infix[], char postfix[], unsigned int size)
{
    Stack *stack;
    char temp, temp2;
    int i, pos = 0;
    stack = createStack(size);
    for (i = 0; i < size; i++)
    {
        temp = infix[i];
        if (operator(temp) == 2)
        {
            if (temp == '(' || temp == '[' || temp == '{')
            {
                push(stack, temp);
            }
            else
            {
                while (stack->top != -1 && (stack->arr[stack->top] != ')' || stack->arr[stack-
>top] != ']' || stack->arr[stack->top] != '}'))
                {
                    temp2 = pop(stack);
                    if (operator(temp2) == 1)
                    {
                        postfix[pos] = temp2;
                        pos++;
                    }
                }
                pop(stack);
            }
        }
        else if (operator(temp) == 1)
        {
            if (precedence(temp) > precedence(stack->arr[stack->top]))
```

```
        {
            push(stack, temp);
        }
        else
        {
            while (stack->top != -1 && precedence(temp) <= precedence(stack->arr[stack->top]))
            {
                temp2 = pop(stack);
                postfix[pos] = temp2;
                pos++;
            }
            push(stack, temp);
        }
    }
    else
    {
        postfix[pos] = temp;
        pos++;
    }
  }
  while (stack->top != -1)
  {
    postfix[pos] = pop(stack);
    pos++;
  }
}

int getSize(char arr[])
{
  int i = 0;
  while (arr[i] != '\0')
  {
    i++;
  }
  return i;
}

int main()
```

```
{
   char infix[arrSize], postfix[arrSize];
   int size, i;
   printf("Enter Infix Expression:");
   gets(infix);
   size = getSize(infix);
   infixToPostfix(infix, postfix, size);
   printf("Postfix Expression:");
   puts(postfix);
   getch();
   clrscr();
   return 0;
}
```

OUTPUT:

RESULT: