PREFIX EVALUATION

Exp. No.:

AIM:

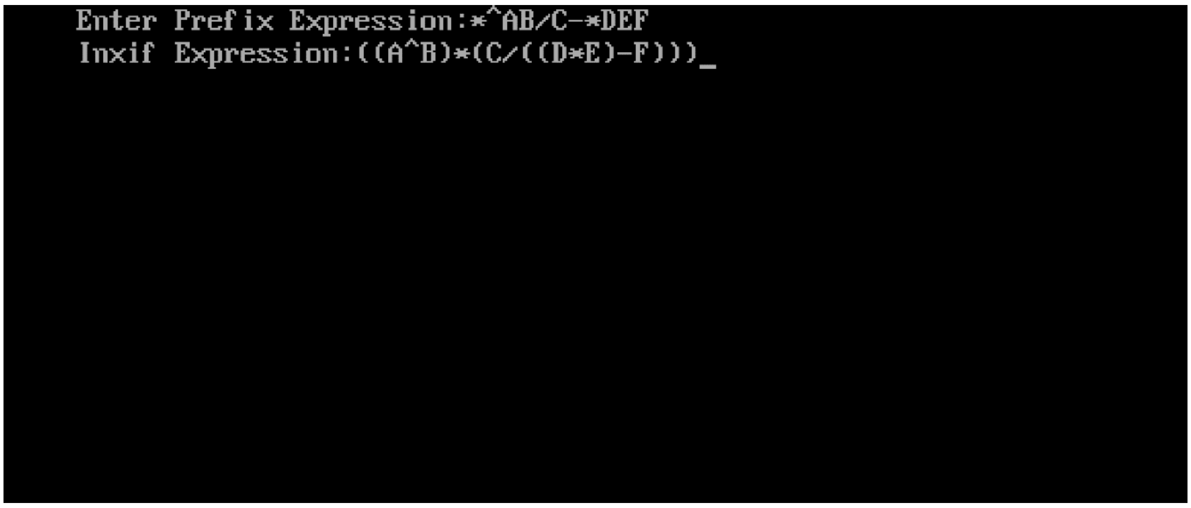
ALGORITHM:

PROGRAM:

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

#define arrSize 100

struct Node
{
   struct Node *left;
   char value;
   struct Node *right;
};
typedef struct Node node;

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

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

stack *createStack(unsigned int size)
{
   stack *st = (stack *)malloc(sizeof(stack));
```

```
    st->size = size;
    st->top = -1;
    st->arr = (node **)malloc(size * sizeof(node));
    return st;
}

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

node *pop(stack *st)
{
    node *temp;
    if (st->top == -1)
    {
        return NULL;
    }
    else
    {
        temp = st->arr[st->top];
        st->top--;
        return temp;
    }
}

void display(node *elem)
{
    if (elem != NULL)
    {
        if (elem->left && elem->right)
        {
            printf("(");
            display(elem->right);
            printf("%c", elem->value);
            display(elem->left);
            printf(")");
        }
        else
        {
            display(elem->right);
            printf("%c", elem->value);
            display(elem->left);
        }
    }
}

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

stack *preEval(char infix[], unsigned int size)
{
    stack *st;
    node *a, *b, *temp;
    int i;
    st = createStack(size);
    for (i = 0; infix[i] != '\0'; i++)
    {
        if (operator(infix[i]) == 0)
        {
            temp = (node *)malloc(sizeof(node));
            temp->value = infix[i];
            temp->left = temp->right = NULL;
            push(st, temp);
        }
        else if (operator(infix[i]) == 1)
        {
            a = pop(st);
            b = pop(st);
            temp = (node *)malloc(sizeof(node));
            temp->value = infix[i];
            temp->left = b;
            temp->right = a;
            push(st, temp);
        }
    }
    return st;
}

int main()
{
    char prefix[arrSize], reverse[arrSize];
    stack *infix;
    int size, i;
    printf("Enter prefix Expression:");
    gets(prefix);
    size = getSize(prefix);
    for (i = 0; i < size; i++)
    {
        reverse[i] = prefix[size - i - 1];
    }
    infix = preEval(reverse, size);
    printf("Inxif Expression:");
    display(pop(infix));
    getch();
    clrscr();
    return 0;
}
```

OUTPUT:



```
Enter Prefix Expression:*^AB/C-*DEF
Inxif Expression:((A^B)*(C/((D*E)-F)))_
```

RESULT: