**CODE:-**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <ctype.h>

#include <math.h>

#define MAX 100

char infix[MAX], stack[MAX], postfix[MAX], prefix[MAX];

int intstack[MAX];

int top = -1;

void push(char[], char);

char pop(char[]);

void intpush(int[], int);

int intpop(int[]);

int isFull();

int isEmpty();

int instack\_priority(int, char);

int symbol\_priority(int, char);

void infix\_to\_postfix()

{

int i, p = 0;

char next;

char symbol;

for (i = 0; infix[i] != '\0'; i++)

{

symbol = infix[i];

switch (symbol)

{

case '(':

push(stack, symbol);

break;

case ')':

while ((next = pop(stack)) != '(')

postfix[p++] = next;

break;

case '+':

case '-':

case '\*':

case '/':

case '%':

case '^':

while ((top > -1) && instack\_priority(1, stack[top]) >= symbol\_priority(1, symbol))

postfix[p++] = pop(stack);

push(stack, symbol);

break;

default:

postfix[p++] = symbol;

}

}

while (!isEmpty())

postfix[p++] = pop(stack);

postfix[p] = '\0';

}

void infix\_to\_prefix()

{

int i, p = 0, j;

char next;

char symbol;

for (i = strlen(infix) - 1; i >= 0; i--)

{

symbol = infix[i];

switch (symbol)

{

case ')':

push(stack, symbol);

break;

case '(':

while ((next = pop(stack)) != ')')

prefix[p++] = next;

break;

case '+':

case '-':

case '\*':

case '/':

case '%':

case '^':

while ((top != -1) && instack\_priority(2, stack[top]) > symbol\_priority(2, symbol))

prefix[p++] = pop(stack);

push(stack, symbol);

break;

default:

prefix[p++] = symbol;

}

}

while (!isEmpty())

prefix[p++] = pop(stack);

prefix[p++] = '\0';

for (i = 0, j = p - 2; i < j; i++, j--)

{

char temp;

temp = prefix[i];

prefix[i] = prefix[j];

prefix[j] = temp;

}

}

void eval\_postfix()

{

char symbol;

int next1, next2, next3;

for (int i = 0; postfix[i] != '\0'; i++)

{

symbol = postfix[i];

if (!(symbol == ' '))

{

if (isdigit(symbol))

{

intpush(intstack, symbol - '0');

}

else

{

next1 = intpop(intstack);

next2 = intpop(intstack);

switch (symbol)

{

case '+':

next3 = next2 + next1;

break;

case '-':

next3 = next2 - next1;

break;

case '\*':

next3 = next2 \* next1;

break;

case '/':

next3 = next2 / next1;

break;

case '^':

next3 = pow(next2, next1);

break;

case '%':

next3 = next2 % next1;

}

intpush(intstack, next3);

}

}

}

}

void eval\_prefix()

{

char symbol;

int next1, next2, next3;

for (int i = strlen(prefix) - 1; i >= 0; i--)

{

symbol = prefix[i];

if (!(symbol == ' '))

{

if (isdigit(symbol))

{

intpush(intstack, symbol - '0');

}

else

{

next1 = intpop(intstack);

next2 = intpop(intstack);

switch (symbol)

{

case '+':

next3 = next1 + next2;

break;

case '-':

next3 = next1 - next2;

break;

case '\*':

next3 = next1 \* next2;

break;

case '/':

next3 = next1 / next2;

break;

case '^':

next3 = pow(next1, next2);

break;

case '%':

next3 = next1 % next2;

}

intpush(intstack, next3);

}

}

}

}

int main()

{

int choice;

while (1)

{

printf("Enter 1 for infix to postfix.\n");

printf("Enter 2 for infix to prefix.\n");

printf("Enter 3 for evaluating postfix.\n");

printf("Enter 4 for evaluating prefix.\n");

printf("Enter 5 to exit.\n");

scanf("%d", &choice);

switch (choice)

{

case 1:

printf("Enter the infix array.\n");

getchar();

gets(infix);

infix\_to\_postfix();

printf("The postfix array is : ");

puts(postfix);

break;

case 2:

printf("Enter the infix array.\n");

getchar();

gets(infix);

infix\_to\_prefix();

printf("The prefix array is : ");

puts(prefix);

break;

case 3:

printf("Enter the postfix array.\n");

getchar();

gets(postfix);

eval\_postfix();

printf("The evaluated value is : %d\n", intpop(intstack));

break;

case 4:

printf("Enter the prefix array\n");

getchar();

gets(prefix);

eval\_prefix();

printf("The evaluated value is : %d\n", intpop(intstack));

break;

case 5:

exit(1);

default:

printf("Erroneous input.\n");

}

}

return 0;

}

int isFull()

{

if (top == MAX - 1)

return 1;

else

return 0;

}

int isEmpty()

{

if (top == -1)

return 1;

else

return 0;

}

int instack\_priority(int n, char c)

{

switch (c)

{

case '+':

case '-':

return 1;

case '\*':

case '/':

case '%':

return 2;

case '^':

if (n == 1)

return 3;

else

return 4;

}

}

int symbol\_priority(int n, char c)

{

switch (c)

{

case '+':

case '-':

return 1;

case '\*':

case '/':

case '%':

return 2;

case '^':

if (n == 1)

return 4;

else

return 3;

}

}

void push(char a[], char item)

{

if (isFull())

{

printf("Stack Overflow\n");

exit(1);

}

a[++top] = item;

}

char pop(char a[])

{

char item;

if (isEmpty())

{

printf("Stack Underflow.\n");

exit(1);

}

else if (top > -1)

{

return a[top--];

}

}

void intpush(int a[], int item)

{

if (isFull())

{

printf("Stack Overflow\n");

exit(1);

}

a[++top] = item;

}

int intpop(int a[])

{

int item;

if (isEmpty())

{

printf("Stack Underflow.\n");

exit(1);

}

else if (top > -1)

{

return a[top--];

}

}

**OUTPUT:-**

Enter 1 for infix to postfix.

Enter 2 for infix to prefix.

Enter 3 for evaluating postfix.

Enter 4 for evaluating prefix.

Enter 5 to exit.

1

Enter the infix array.

A + B / C \* ( D - E) ^ F

The postfix array is : A B C / D E- F^\*+

Enter 1 for infix to postfix.

Enter 2 for infix to prefix.

Enter 3 for evaluating postfix.

Enter 4 for evaluating prefix.

Enter 5 to exit.

2

Enter the infix array.

A + B / C \* ( D - E) ^ F

The prefix array is : +A \*/ B C ^ - D E F

Enter 1 for infix to postfix.

Enter 2 for infix to prefix.

Enter 3 for evaluating postfix.

Enter 4 for evaluating prefix.

Enter 5 to exit.

3

Enter the postfix array.

5 3 2 / 3 2 - 2 ^ \* +

The evaluated value is : 6

Enter 1 for infix to postfix.

Enter 2 for infix to prefix.

Enter 3 for evaluating postfix.

Enter 4 for evaluating prefix.

Enter 5 to exit.

4

Enter the prefix array.

+ 5 \* / 3 2 ^ - 3 2 2

The evaluated value is : 6

Enter 1 for infix to postfix.

Enter 2 for infix to prefix.

Enter 3 for evaluating postfix.

Enter 4 for evaluating prefix.

Enter 5 to exit.

5