```
/*This program converts a number from infix
    to prefix and then evaluates the equation*/
#include <stdio.h>
#include <string.h>
#include <ctype.h>
#include <math.h>

int ISP(char op){
    if(op=='^')                         return 4;
    else if(op=='*' || op=='/')         return 2;
    else if(op=='+' || op=='-')         return 1;
    else if(op==')')                    return 0;
}
int ICP(char op){
    if(op=='^')                         return 3;
    else if(op=='*' || op=='/')         return 2;
    else if(op=='+' || op=='-')         return 1;
    else if(op==')')                    return 4;
}

/*reverses the string and and also adding '#' at end*/
char rev(char s1[50],char revs1[50]){
    //printf("Infix: %s\n",s1);
    int len=strlen(s1);
    for(int i=0;i<len;i++){
        revs1[i]=s1[len-i-1];
    }
    revs1[len]='#';
    revs1[len+1]='\0';
    //printf("ReverseInfix: %s\n",revs1);
}

/* Adding brackets at '(' begining and end ')'  */
char addbracket(char s1[50],char revs1[50]){

    int slashzero=strlen(s1);
    s1[slashzero]=')';
    //s1[++slashzero]='#';
    s1[++slashzero]='\0';
    for(int i=slashzero;i>=0;i--){
        s1[i+1]=s1[i];
    }
    s1[0]='(';
    //printf("Infix: %s\n",s1);
    rev(s1,revs1);
}

/*Normal pop operation*/
char pop(char stack[]){
    char out;
```

```
    int top=strlen(stack)-1;
    int num;
    out=stack[top];
    stack[top]='\0';
    return out;
}

/*push into a stack*/
int push(char val, char arr[]){
    int i;
    int top=strlen(arr);
    arr[top]=val;
    arr[top+1]='\0';
}

/*pops the elements till left bracket*/
char popall(char stack[],char output[]){
    int i=0;
    int top=strlen(stack)-1;
    while(stack[top]!=')'){
        int num;
        push(stack[top],output);
        stack[top]='\0';
        top--;
    }
    stack[top]='\0';
}

/*Number push*/
void pushNum(int val, int arr[],int* top){
    int i;
    arr[*top]=val;
    arr[++*top]='\0';
}
/*Number pop*/
int popNum(int arr[], int* top){
    int num=arr[--*top];
    arr[*top+1]='\0';
    return num;
}
/*Calculate the val of the prefix notation*/
void Calculator(char output[]){
    char out[50];
    rev(output,out);
    out[strlen(out)-1]='\0';
    printf("Sum of (%s): ",out);
    int i=0,j=0,res=0,top=0,d=0,digit1=0,digit2=0;
    int slashzero=strlen(output);
    char stak[25]={'\0'},oper;
    int numbers[25]={'\0'};
```

```
    while(i<slashzero){
        char ch;

        ch=output[i];
        if (isdigit(ch)!=0||isalpha(ch)!=0){
            pushNum(ch-'0',numbers,&top);
        }
        else if (isdigit(ch)==0||isalpha(ch)==0){
            push(ch,stak);
        }
            oper=stak[j];
        if (oper){
            digit1=popNum(numbers,&top);
            digit2=popNum(numbers,&top);
            switch(oper){
                case '*': res=digit1*digit2;
                        break;
                case '/': res=digit1/digit2;
                        break;
                case '+': res=digit1+digit2;
                        break;
                case '-': res=digit1-digit2;
                        break;
                case '^': res=pow(digit1,digit2);
                        break;
            }
            j++;
            pushNum(res,numbers,&top);

        }
        ++i;
    }
    printf(" %d",numbers[0]);
}

/* Converts the infix expression calling all the functions */
void Prefixer(char s1[50]){
    int right=strlen(s1)-1 ,top=0 ,i=0 ;
    char stack[25]={'\0'}  ,output[50]={'\0'};
    char left,op,isp;

    while(s1[i]!='#'){
        left=s1[i];
        if(left==')'){
            push(left,stack);

        } else if (isdigit(left)!=0||isalpha(left)!=0){
            push(left,output);

        } else if (left=='*' || left=='/' || left=='-' || left=='+' || left=='^' ){
```

```
            isp=pop(stack);
            if(left&&isp){
                    if(ICP(left)>ISP(isp)){
                        push(isp,stack);
                        push(left,stack);

                    } else {
                        push(isp,output);
                        push(left,stack);
                    }
                }
        } else if(left=='('){
            popall(stack,output);
        }
        i++;
    }
    output[strlen(output)]='\0';
    char prefix[50];
    //addbracket(output,s1);

    rev(output,s1);
    s1[strlen(s1)-1]='\0';
    printf("Prefix : %s\n",s1);
    for(i=0;s1[i]!='\0';i++)
    {
        if(isalpha(s1[i]))
        {
            printf("give me %c <--> ",s1[i]);
            scanf(" %c",&s1[i]);
        }
    }
    rev(s1,output);
    output[strlen(output)-1]='\0';
    Calculator(output);
}

int main() {
    printf("Infix expression --> ");
    char s1[50];//="(a+b/c)/d+e"; //="(1+4/2)/3+4";
    fgets(s1,50,stdin);
    //printf("\nInfix eq: %s\n",s1);
    char revs1[50],num;
    addbracket(s1,revs1);
    Prefixer(revs1);
    return 0;
}
```

```
Infix expression --> (a+b/c)/d+e
Prefix : +/+a/bcde
give me a <--> 1
give me b <--> 4
give me c <--> 2
give me d <--> 3
give me e <--> 4
Sum of (+/+1/4234):  5
```

Prog 6 Infix to Prefix

Algorithm for ISP & ICP

1. START
2. Set

	ISP	ICP
^	4	3
/, *	2	2
+, -	1	1
)	0	4

3. STOP

Algorithm for pop - in the string

1. START
2. Store the last element in a variable num
3. decrement the top of the stack
4. return the num variable

Algorithm for push

1. START
2. store the element from the user to top of the stack after increasing the top
3. STOP.

Algorithm for popall

1. START
2. while stack[top] != ')' do
    1. push the stack[top] to output
    2. ~~decrement the~~ top --
3. STOP

Algorithm for Prefixing

1. START
2. Take the user array and set the last element to '#'.
3. Scan from left to right (int i = 0)
   while (string[i] != '#') do
    1. set the string[i] at left

2. if (left == ')') then push (left, stack);

3. else if (isdigit (left) != 0) then push (left, output);

4. else if (left is an operator) then

    1. isp ← pop (stack),

    2. if (ICP (left) > ISP (isp)) then

        1. push (isp, stack);

        2. push (left, stack);

    3. else

        1. push (isp, output);

        2. push (left, stack);

5. else if (left == '(') then popall (stack, output)

6. i++

4. Reverse the string.

5. STOP

Algorithm for calculating the prefix

1. START

2. Initialize and declare necessary variables. eg:- i = 0

3. while (i < strlen(output)) do

1. set ch ← output[i],
2. if (ch is a digit) then
    1. if (ch is the first digit) then set digit1 ← ch - '0',
    2. else set it as digit2
3. else push (operator to a stack)
4. if (digit1 && digit2) then
    1. set oper ← top operator in stack
    ~~switch (oper)~~
    2. perform [digit1 oper digit2]
    3. decrease the stack
    4. set digit1 ← result of [digit1 'oper' digit2]
5. i++;

4. Print result // or digit1
5. STOP

Algorithm for main.

1. START
2. Ask the user to input an infix expression
3. add brackets at beginning and end
4. ~~Use prefix algorithm and change it~~
4. Reverse the string
5. Use prefix algorithm on string
6. Call the calculation on prefix
7. print the output
8. STOP.