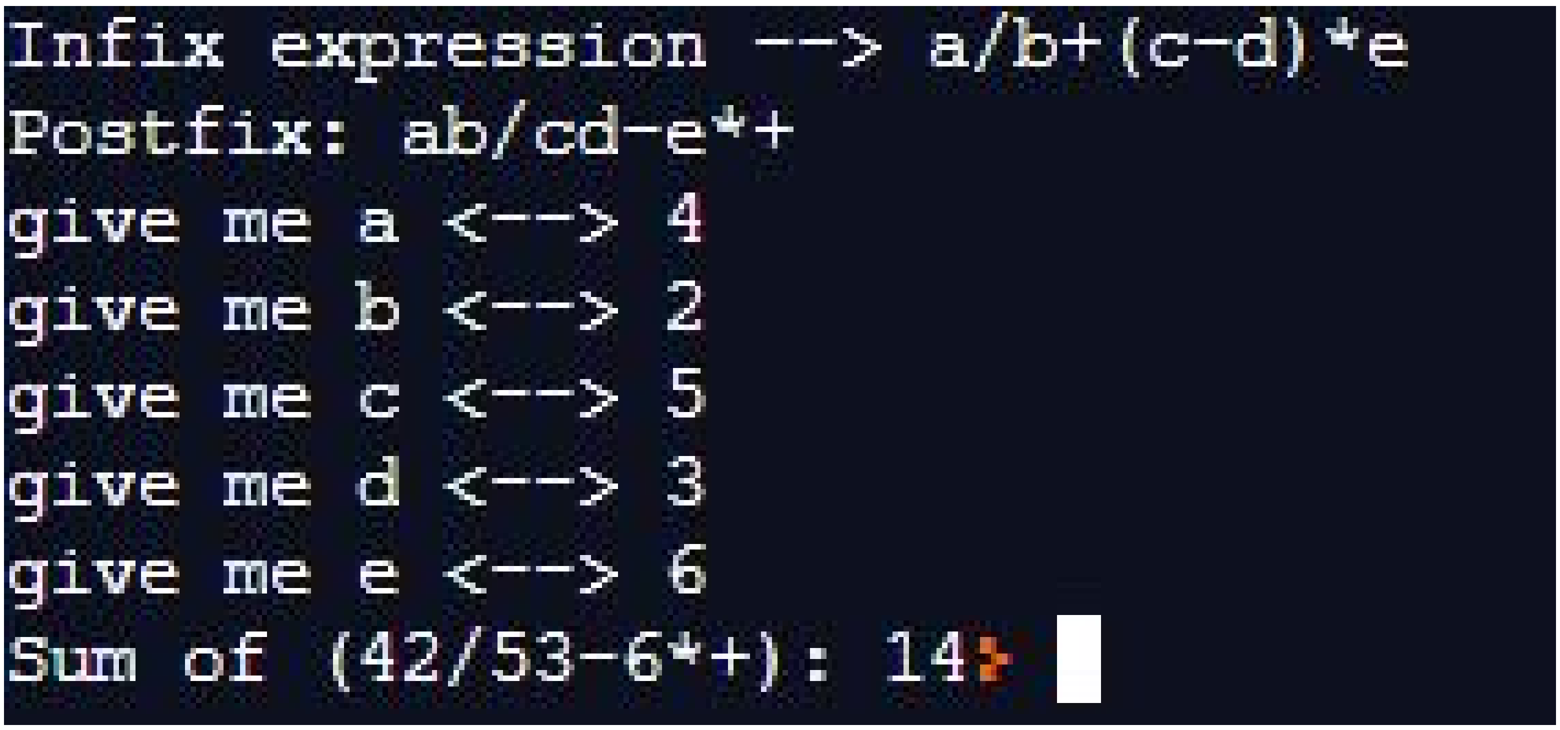
```c
/*This program converts a number from infix
    to postfix and then evaluates the equation*/
#include <stdio.h>
#include <string.h>
#include <ctype.h>
#include <math.h>
int ISP(char op){
    if(op=='^')                         return 3;
    else if(op=='*' || op=='/')         return 2;
    else if(op=='+' || op=='-')         return 1;
    else if(op=='(')                    return 0;
}
int ICP(char op){
    if(op=='^')                         return 4;
    else if(op=='*' || op=='/')         return 2;
    else if(op=='+' || op=='-')         return 1;
    else if(op=='(')                    return 4;
}

/*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';
}
```

```c
/*Integer push*/
int pushInt(int val, int arr[],int* top){
    int i;
    //printf("%d",top);
    arr[*top]=val;
    arr[++*top]='\0';
}
/*Calculate the val of the postfix notation*/
void Calculator(char output[]){
    printf("Sum of (%s): ",output);
    int i=0,res=0,top=0;
    int slashzero=strlen(output);
    int numbers[25];
    while(i<slashzero){
        int digit1,digit2,dcount;
        char ch;

        ch=output[i];
        if (isdigit(ch)!=0){
            pushInt(ch-'0',numbers,&top);

        } else { //if(i==-1){
            digit2=numbers[--top];
            numbers[top]='\0';
            digit1=numbers[--top];
            numbers[top]='\0';

            switch(ch){
                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;
            }
            pushInt(res,numbers,&top);
        }

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

/* Converts the infix expression calling all the functions */
void Postfix(char s1[50]){
    int right=strlen(s1)-1 ,top=0 ,i=0 ;
    char stack[50]={'\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(ICP(left)>ISP(isp)){
                push(isp,stack);
                push(left,stack);

            } else {
                push(isp,output);
                push(left,stack);
            }
        } else if(left==')'){
            popall(stack,output);
        }
        i++;
    }
    printf("Postfix: %s\n",output);
    for(i=0;output[i]!='\0';i++)
    {
        if(isalpha(output[i]))
        {
            printf("give me %c <--> ",output[i]);
            scanf(" %c",&output[i]);
        }
    }
    Calculator(output);
    //printf("stack: %s",stack);
}

/* Adding brackets at '(' begining and end ')' and also adding '#' at end */
char addbracket(char s1[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);
}

int main() {
    printf("Infix expression --> ");
    char s1[50];//="4/2+(5-3)*6";//="(a+b/c)/d+e"; //
    fgets(s1,50,stdin);
    //printf("\nInfix eq: %s\n",s1);
    addbracket(s1);
    Postfix(s1);
    return 0;
}
```



```
Infix expression --> a/b+(c-d)*e
Postfix: ab/cd-e*+
give me a <--> 4
give me b <--> 2
give me c <--> 5
give me d <--> 3
give me e <--> 6
Sum of (42/53-6*+): 14>
```

# Program 5 - Infix to Post

## Algorithm of ISP & ICP

1. START
2. Set

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

3. ~~Set~~ STOP

## Algorithm of pop

1. START
2. Check if the top is empty and exit if true
3. Store the element in top of the stack to another variable and ~~decrease~~ remove the top.
4. Return top ~~variable~~ value stored in variable.
5. STOP.

Algorithm of push

1. START
2. Check if stack is full. and exit is true
3. If stack not full, increment the top to po
   to next empty space.
4. Add data ~~to~~ element to the stack location,
   where top is pointing.
5. ~~Return stack~~ STOP

# Algorithm for infix to Postfix

```
1. Add brackets to the infix element received
   and end it with '#'.

2. Create a stack -> Stack and Output
3. Create and initialise a variable top
   as -1

4. Set item = infix.item() // '(' bracket
5. push ~~to~~(item to stack)
6. While (infix.item() != '#')
    1. item = infix.item()
    2. x = PoP(stack).
    3. Case : if iteam is an operand // a,b,c
        1. Push (x to Stack)
        2. Push (item to Output)

    4. Case : if item == ')'
        1. While x != '(' do
            1. Push (x to Output)
            2. x = PoP (Stack)
        2 End while
    5. Case : if ISP(x) >= ICP(item)
```

```
            1. While (ISP(x) >= ICP(item)) do
                1. Push (x to Output)
                2. x=pop(stack)
            2. End While
            3. ~~Push (x to stack)~~
            4. Push (item to stack.
        6. Case: if ISP(x) < ICP(item)
            1. Push (x to stack)
            2. Push (item to stack)
        7. Otherwise: Print "Invalid Expression"
    7. End While
    8. STOP // now a+b ⇒ [ab+ ⇒ postfix ⇒ output]
```

Algorithm for ~~main~~ Calculating prefix

```
1. ~~Add #~~ START
2. Add '#' to the end of ~~output~~ postfix array
3. Create a stack called numbers.
4. Scan the prefix from left to right
   ∴ item = prefix item.
5. While item != '#'
   5.1. if ~~prefix~~ item is an operand then
```

```
2. Else if item == operator
        1. op = item
        2. y = PoP(numbers)
        3. x = Pop(numbers)
        4. Ask values for operand from user
            // a = 5 , b = 7
        5. Perform Ans = x op y
        6. Push (Ans to numbers).
    3. End If
~~6. Print (PoP(numbers)) // final ans~~
4. item = prefix.item();
6. End while
7. return pop(numbers)
8. STOP.
```

Algorithm for main

```
1. START
2. Write the infix form of expression and store
   it in an array → Infix
3. Call Infix to Prefix() and Calkulating prefix
4. Print out the answer returned in Calculating prefix
5. STOP.
```