**Design and implement an algorithm to evaluate an arithmetic expressions which may be any form (postfix, prefix, infix), and demonstrate its working with suitable examples.**

**POSTFIX EVALUATION**

**THEORY**

The Postfix notation is used to represent algebraic expressions. The expressions written in postfix form are evaluated faster compared to infix notation as parenthesis are not required in postfix. We have discussed [infix to postfix conversion](http://quiz.geeksforgeeks.org/stack-set-2-infix-to-postfix/). In this post, evaluation of postfix expressions is discussed.

**CODE WITH COMMENTS**

#include<stdio.h>  
void sum();//functions declaration  
void diff();  
void mul();  
void divi();  
void power();  
  
int stack[50], top=-1;//univaersal declaration of stack  
  
int main()  
{  
    char st[50];  
    int i;  
    printf("enter the postfix expression\n");  
    scanf("%[^\n]s",st);//expression is scanned untill enter is clicked  
    for(i=0;st[i]!='\0';i++)//loop runs untill null value is found  
    {  
        if(st[i]!=' ')//neglecting the white space  
        {  
            switch(st[i])//respective functins are performed  
            {  
                case '+': sum();  
                break;  
                case '-': diff();  
                break;  
                case '\*': mul();  
                break;  
                case '/': divi();  
                break;  
                case '^': power();  
                break;  
                default: top++;  
                stack[top]=st[i]-48;  
                break;  
            }  
        }  
    }  
    printf("the result is %d\n",stack[top]);//result is declared  
}  
  
void sum()  
{  
    int res, op1, op2;  
    op1=stack[top];//stack top is operand1  
    top--;//top is popped  
    op2=stack[top];//new stack top is operand2  
    top--;//top is popped again  
    res=op1+op2;//sum  
    top++;//the result i saved as new top of the stack  
    stack[top]=res;  
}  
  
void diff()  
{  
    int res, op1, op2;  
    op1=stack[top]; ];//stack top is operand1  
    top--;  
    op2=stack[top]; //new stack top is operand2  
    top--;  
    res=op1-op2;  
    top++;//the result i saved as new top of the stack  
    stack[top]=res;  
}  
  
void mul()  
{  
    int res, op1, op2;  
    op1=stack[top]; ];//stack top is operand1  
    top--;  
    op2=stack[top]; //new stack top is operand2  
    top--;  
    res=op1\*op2;  
    top++;//the result i saved as new top of the stack  
    stack[top]=res;  
}  
  
void divi()  
{  
    int res, op1, op2;  
    op1=stack[top]; ];//stack top is operand1  
    top--;  
    op2=stack[top]; //new stack top is operand2  
    top--;  
    res=op1/op2;  
    top++;//the result i saved as new top of the stack  
    stack[top]=res;  
}  
  
void power()  
{  
    int res, op1, op2, i;  
    op1=stack[top]; ];//stack top is operand1  
    top--;  
    op2=stack[top]; //new stack top is operand2  
    top--;  
    for(i=0;i<op1;i++)//loop runs operand1 times  
    res=res\*op2;  
    top++;//the result i saved as new top of the stack  
    stack[top]=res;  
}

**OUTPUT**

****

**ALGORITHM**

Create an empty stack and start scanning the postfix expression from left to right.

* If the element is an operand, push it into the stack.
* If the element is an operator **O**, pop twice and get A and B respectively. Calculate B**O**A and push it back to the stack.
* When the expression is ended, the value in the stack is the final answer.

**POSTFIX EVALUATION**

**THEORY**

The Postfix notation is used to represent algebraic expressions. The expressions written in postfix form are evaluated faster compared to infix notation as parenthesis are not required in postfix. We have discussed [infix to postfix conversion](http://quiz.geeksforgeeks.org/stack-set-2-infix-to-postfix/). In this post, evaluation of postfix expressions is discussed.

**CODE WITH COMMENTS**

#include<stdio.h>

#include<string.h>

#include<math.h>

#include<stdlib.h>//header file declaration

#define max 20

int top=-1, stack[top], opp1, opp2;//universal variable declaration

void push(int x)//function defination

{

top++;

stack[top]=x;//item pushed

}

int pop()//function defination

{

char c;

c=stack[top];//variable declaration

top--;//item popped

printf("%c",c);

}

int prefixeval()//function defination

{

int len, res, i;//variabl declaration

char prefix[20];

scanf("%s",prefix);

len=strlen(prefix);//length of infix exp

for(i=len-1;prefix[i]>=;i--)//checks condition

{

if(isdigit(prefix[i]))//checks condition

push(prefix[i]-48))//item pushed

else

{

opp1=pop();

opp2=pop();

switch(prefix[i])//checks cases

{

case '+': push(opp1+opp2);

break;

case '-': push(opp1-opp2);

break;

case '\*': push(opp1\*opp2);

break;

case '/': push(opp1/opp2);

break;

case '^': res=pow(opp1,opp2);

push(res);

break;

}

}

printf("result is %d",pop());

}

int main()//main function

{

char prefix[20];//variabe declaration

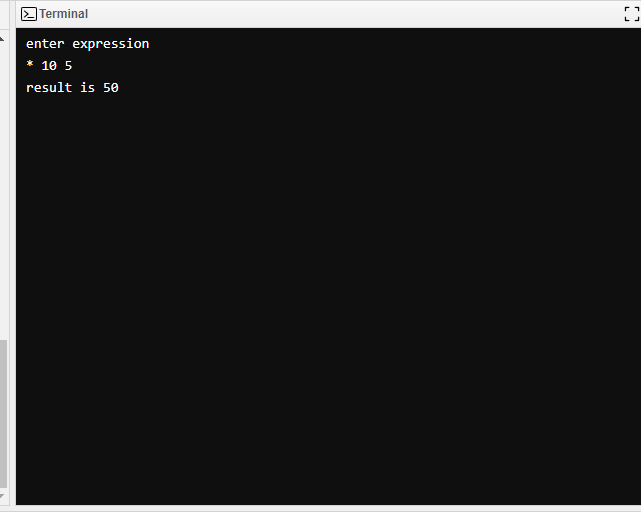
printf("enter expression\n");

scanf("%s",prefix);//prefix exp scanned

prefixeval(prefix);//function call

}

**OUTPUT**

****

**ALGORITHM**

Create an empty stack and start scanning the postfix expression from left to right.

* If the element is an operand, push it into the stack.
* If the element is an operator **O**, pop twice and get A and B respectively. Calculate B**O**A and push it back to the stack.
* When the expression is ended, the value in the stack is the final answer.

**INFIX EVALUATION**

**THEORY**

The Postfix notation is used to represent algebraic expressions. The expressions written in postfix form are evaluated faster compared to infix notation as parenthesis are not required in postfix. We have discussed [infix to postfix conversion](http://quiz.geeksforgeeks.org/stack-set-2-infix-to-postfix/). In this post, evaluation of postfix expressions is discussed.

**CODE WITH COMMENTS**

#include<stdio.h>//header file

int top = -1, stack [100];//variable declaratin

main ( ){//main function

char a[50], ch;

int i,op1,op2,res,x;//variable declaratin

void push (int);

int pop( );//function call

int eval (char, int, int);

printf("enter a postfix expression:");

gets (a);//exp scanned

for(i=0; a[i]!='\0'; i++){//condition checked

ch = a[i];

if (ch>='0' && ch<='9')//condition checked

push('0');

else{

op2 = pop ( );

op1 = pop ( );

res = eval (ch, op1, op2);

push (res);//result pushed

}

}

x = pop ( );

printf("evaluated value = %d", x);//result printed

getch ( );

}

void push (int n){//function defination

top++;

stack [top] = n;//item pushed

}

int pop ( ){//function defination

int res ;//

res = stack [top];

top--;//item popped

return res;

}

int eval (char ch, int op1, int op2){//function defination

switch (ch){//condition check

case '+' : return (op1+op2);

case '-' : return (op1-op2);

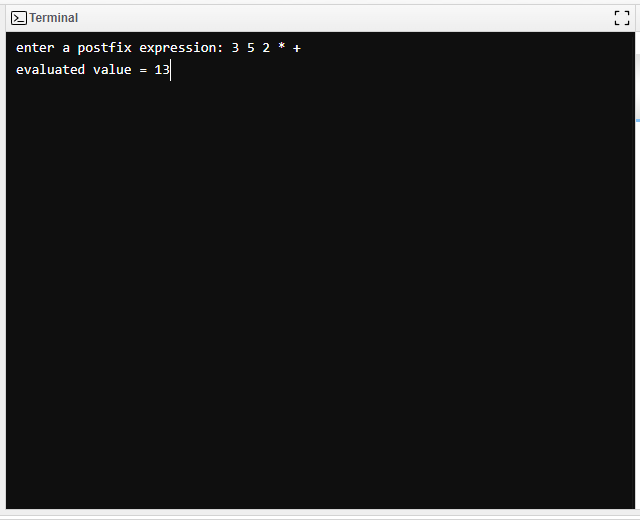
case '\*' : return (op1\*op2);

case '/' : return (op1/op2);//operation performed

}

}

**OUTPUT**



**ALGORITHM**

Scan the input string from left to right.

For each input symbol,

* If it is a digit then, push it on to the stack.
* If it is an operator then, pop out the top most two contents from the stack and apply the operator on them. Later on, push the result on to stack.
* If the input symbol is ‘\0’, empty the stack.