表达式求值

一、 表达式求值

表达式求值在信息学竞赛中是一个常见的模拟类问题。算术表达式的表达方式有三种：前缀表达式、中缀表达式、后缀表达式，它们的区别在于运算符的位置不同，在现实中，我们常见的是中缀表达式。如（1+3）*5-6 是一个中缀表达式，它对应的后缀表达式和前缀表达式分别是：１３＋５＊６－，　－６＊５＋３１。前缀表示式也称为波兰表达，后缀表达式又称为逆波兰表达式。前缀、后缀表达式计算方法较中缀表达式而言，过程较为简单，实际中可以先将表达式转换为后、前缀表达式，然后再计算结果。

1、后缀表达式求解过程

扫描后缀表达式，凡遇操作数则将之压进堆栈，遇运算符则从堆栈中弹出两个操作数进行该运算，将运算结果压栈，然后继续扫描，直到后缀表达式被扫描完毕为止，此时栈底元素即为该后缀表达式的值。

2、中缀表达式转后缀表达式

设置两个栈：操作数栈（ovs）和运算符栈（ops），用来分别存放表达式中的操作数和运算符。开始时操作数栈为空，运算符栈中放入一个特殊的标志运算符号@号，并在表达式的末尾相应地加上一个@号，并规定@号的优先级最低，然后从左向右扫描表达式，凡遇操作数便一律进栈；若遇运算符，则判断其优先级是否大于运算符栈栈顶元素的优先级。若小，则栈顶运算符退栈，并从操作数栈中弹出两个操作数（操作数为后缀表达式）进行后缀变换处理，处理结果进操作数栈，重复刚才的比较，直到栈顶运算符的优先级大于等于当前运算符的优先级；此时，若当前运算符的优先级大于栈顶运算符的优先级，则当前运算符进栈，继续扫描；若当前运算符的优先级等于(括号)栈顶运算符的优先级，则弹出栈顶运算符，继续扫描。扫描完该表达式后运算符栈为空，操作数栈中只有一个元素，该元素就是所要求的后缀表达式。

二、 表达式树

表达式树是一种特殊的二叉树，它的叶节点是操作数，其他节点是操作符。假设所有的运算符都是双目运算符，那么刚好形成一颗二叉树。我们可以通过递归计算左子树和右子树的值，从而得到整个表达式树的值。不同的遍历方式可以得到不同类型表达式，如中根序便利表达式树则得到中缀表达式。

1、表达式树求解过程

直接递归求解即可。

2、中缀表达式转表达式

根据表达式树的特点，根节点为整个表达式中优先级最低的运算符，依次类推，所以每次可以找出优先级最低的运算符然后递归构建。

三、 例题解析

1、P2507. 表达式求值

方法一：转后缀计算

```
#include<bits/stdc++.h>
using namespace std;
char f[7]={'+','-','*','/','^','(',')'};
int op[7][7]=
```

```
{
   {1,1,0,0,0,0,1},
   {1,1,0,0,0,0,1},
   {1,1,1,1,0,0,1},
   {1,1,1,1,0,0,1},
    {1,1,1,1,1,0,1},
   {0,0,0,0,0,0,-1},
   {1,1,1,1,1,1,-2}
};
inline int getid(char c){
   for(int i=0;i<7;i++)if(f[i]==c)return i;
}
char s[1000],t[1001];
int st[1001],top,len,lent ;

void zhuan(char s[]){//转后缀

   len=strlen(s);s[len++]=')';
   lent=0;top=0;

   st[++top]=5;//插入一个左括号

   for(int i=0;i<len;){

      while(s[i]=='(')st[++top]=5,i++;   //左括号进栈

      while(s[i]>='0'&&s[i]<='9'){
         t[lent++]=s[i];
         i++;
      }
      t[lent++]=' ';
      int now;
      while(s[i]<'0'||s[i]>'9'){
         if(s[i]=='(')break;
         now=getid(s[i]);

         while (op>0&&op[st[top]][now]==1)// 栈内高出栈

            t[lent++]=f[st[top--]];


         if(op[ st[top] ][now]==0){//栈内低进栈

            st[++top]=now;i++;
         }
         else {

               top--;//相等消括号

               i++;
```

```
            }
            if(i>=len)break;
        }
    }
    t[lent]='\0';
}
int pow(int x,int y){
    int ret=1;
    for(int i=1;i<=y;i++)
        ret*=x;
    return ret;
}
int cal(int x,int y,int o){
    switch(o){
        case 0:return x+y;
        case 1:return x-y;
        case 2:return x*y;
        case 3:return x/y;
         case 4:return pow(x,y);
    }
}
int jisuan(char s[]){
    top=0;
    int i=0,len=strlen(s);
    while(i<len){
        int x=0;
        while(i<len&&s[i]>='0'&&s[i]<='9'){
            x=x*10+s[i]-'0';i++;
        }
        st[++top]=x;i++;
        while(i<len&&s[i]<'0'||s[i]>'9'){
            int now=getid(s[i++]);
            int t=cal(st[top-1],st[top],now);
            st[--top]=t;
        }
    }
    return st[top];
}
int main(){
    scanf("%s",s);
    zhuan(s);
    printf("%d",jisuan(t));
    return 0;
}
```

方法二：表达式树

```
#include<bits/stdc++.h>
using namespace std;
char s[1001];
struct node{
   char op;
   int d,ch[2],flag;
}tree[10001];
int tot=0;
int cal(int x,int y,char o){
   switch(o){
      case '+':return x+y;
      case '-':return x-y;
      case '*':return x*y;
      case '/':return x/y;
       case '^':return pow(x,y);
   }
}

int check(char s[],int L,int R){//判断字符串区间是否为数字

   int ret =0;
   for(int i=L;i<=R;i++){

      if (!(s[i]>='0'&&s[i]<='9'))return -1;//不是一个数

      ret=ret*10+s[i]-'0';
   }
   return ret;
}

int pick(char s[],int p,int R){//找和 p 位置（匹配的）位置

   int t=1;
   p++;
   for(int i=p;i<=R;i++){
       if(s[i]!='('&&s[i]!=')') continue;

       if(s[i]==')')  t--;//右括号减 1

       if(s[i]=='(')  t++;//左括号加 1

       if(t==0)  return i;//匹配 i 位置

    }
  return 0;
}
int find(char s[],int L,int R){
```

```
   int t=0,k=0,f=0;
   for(int i=R;i>=L;i--){
      if(s[i]<='9'&&s[i]>='0')continue;
       if(s[i]=='(') {t--;continue;}
       if(s[i]==')') {t++;continue;}
       if((s[i]=='+'||s[i]=='-')&&t==0)  return i;
       if((s[i]=='*'||s[i]=='/')&&t==0&&f==0)  f=i;//返回最
右边的*、/号
       if(t==0&&k==0)  k=i;
    }
   return f==0?k:f;
}
int  build(char s[],int L,int R){//构建表达式树
   int o=++tot;
   int tmp=check(s,L,R);//检查字符串是否纯数字
   if(tmp!=-1)  {tree[o].flag=1;tree[o].d=tmp;return o;}
   if(s[L]=='('&&pick(s,L,R)==R){//如果最左边是(，检查是否匹配
最右边)
      return build(s,L+1,R-1);
   }
   tmp=find(s,L,R); //找出字符串区间优先级最小运算符
   tree[o].op=s[tmp];tree[o].flag=0;
   tree[o].ch[0]=build(s,L,tmp-1);
   tree[o].ch[1]=build(s,tmp+1,R);
   return o;
}
int jisuan(int p){
   if(tree[p].flag)return tree[p].d;
   return           cal(jisuan(tree[p].ch[0])          ,
jisuan(tree[p].ch[1]) , tree[p].op);
}
int main(){
   scanf("%s",s);
   int rt=build(s,0,strlen(s)-1);
   cout<<jisuan(rt);
   return 0;
}
```