```
//
//  standerd3.cpp
//
//
//  Created by 朱自通 on 2021/4/20.
//

#include "standerd3.hpp"

//科学计数法
int main(){
    char str[10010];
    gets(str);
    int len=strlen(str);
    if(str[0]=='-')printf("-");//如果是负数，输出负号

    int pos=0;
    while(str[pos]!='E'){//pop存放字符串E的位置
        pop++;
    }

    int exp=0;//特判指数为零的情况
    for(int i=pop+2;i<len;i++){
        exp=exp*10+(str[i]-'0');
    }
    if(exp==0){
        for(int i=0;i<pop;i++){
            printf("%c",str[i]);
        }
    }

    if(str[pos+1]=='-'){//如果指数为负
        printf("0.");
        for(int i=0;i<exp-1;i++){//输出exp-1个零
            printf("0");
        }
        printf("%c",str[i]);//输出除了小数点以外的数字
        for(int i=3;i<pos;i++){
            printf("%c",str[i]);
        }
    }
    else{//如果指数为正
        for(int i=0;i<pos;i++){//输出小数点移动后的数
            if(str[i]='.')continue;//略过小数点
        }
        printf("%c",str[i]);//输出当前数位
        if(i==exp+2&&pos-3!=exp){//小数点加在exp+2的位置上
            printf(".");
        }
    }
    for(int i=0;i<exp-(pos-3);i++){//如果指数较大，输出多余的0
        printf("0");
```

```
    }
    return 0;
}

//反转字符串
void reverse(char s[]){
    int len=strlen(s);
    for(int i=0;i<len/2;i++){
        int temp=s[i];
        s[i]=s[len-1-i];
        s[len-1-i]=temp;
    }
}

//多进制数映射到十进制
void init(){
    for(int i=0;i<len;i++){
        if(str[i]>='0'&&str[i]<='9'){
            map[i]=str[i]-'0';
        }
        else{
            map[i]=c-'a'+10;
        }
    }
}

//查询两个不同进制数的比较
ll convertNum(char a[],ll radix,ll t){//t为上界
    ll ans=0;
    int len=strlen(a);
    for(int 0;i<len;i++){
        ans=ans*10+a[i];
        if(ans<0||ans>t)return -1;//转换成十进制之后发生了溢出
    }
    return ans;
}

int cmp(char N2[],ll radix;ll t){//num大输出1，否则为-1，相等为0
    int len=strlen(N2);
    ll num=convertNum(N2,radix,t);
    if(num<0)return 1;
    if(num<t)return -1;
    else if(num==t) return 0;//t其实就有点像N1【】
    else return 1;
}

ll binerysearch(char N2[],ll left,ll right,ll t){
    ll mid;
    while(left<right){
        mid=(left+right)/2;
        int flag=cmp(N2,mid,t);
        if(flag==0) return mid;
        else if(falg==-1) left=mid+1;
        else right=mid;
```

```
    }
    return -1;//解不存在
}

//(实现计算器的核心操作)将中缀表达式转化为后缀表达式
struct node{
    double num;//操作数
    char op;//操作符
    bool flag;//true表示操作数，false表示操作符
}
string str;
stack<node> s;
queue<node> q;
map<char,int> op;

op['+']=op['-']=1;//设定操作符的优先级
op['*']=op['/']=2;

void change(){//将中缀表达式转化成后缀表达式
    double num;
    node temp;
    for(int i=0;i<str.length();){
        if(str[i]>='0'&&str[i]<'9'){
            temp.flag=true;
            temp.num=str[i]-'0';
            while(i<str.length()&&str[i]>='0'&&str[i]<'9'){
                temp.num=temp.num*10+(str[i]-'0');
                i++;
            }
            q.push(temp);
        }
        else {
            temp.flag=false;
            temp.op=str[i];

            while(!s.empty()&&op[str[i]]<=op[s.top().op]){
                q.push(s.top());
                s.pop();
            }
            s.push(temp);
            i++;
        }
    }
    while(!s.empty()){
        q.push(s.top());
        s.pop();
    }
}

double cal(){//计算后缀表达式
    double temp1,temp2;
    node cur,temp;
    while(!q.empty()){
```

```
        cur=q.front();
        q.pop();
        if(cur.flag==true)s.push(cur);
        else{
            temp2=s.top();
            s.pop();
            temp1=s.top();
            s.pop();
            temp.flag=true;
            if(cur.op=='+')temp.num=temp1+temp2;
            else if(cur.op='-') temp.num=temp1-temp2;
            else if(cur.op='*') temp.num=temp1*temp2;
            else temp.num=temp1/temp2;
        }
        s.push(temp);
    }
    return s.top().num;
}

//去掉字符串中的某个字符
string str;
for(string::iterator it=str.begin();it!=str.end();it++){
    if(*it=='//这里可以是任何字符') str.erase(it);
}

//进制转换
1) 十进制转换radix进制
int d[MAX],ind=0;
while(n!=0){
    d[ind++]=n%radix;
    n/=radix;
}
2) radix进制转化十进制
int ans=0;
int len=strlen(str);//必须提前知道str的长度
for(int i=0;i<len;i++){
    ans=ans*radix+str[i];
}

//判断一个高精度数是否回文数
bool judge(bign a){
    for(int i=0;i<a.len/2;i++){
        if(a.d[i]!=a.d[a.len-1-i]){
            return false;
        }
    }
    return true;
}

//将字符串转化为数字，作为唯一标识
int getid(char name[]){//看作26进制的数
    int ans=0;
    int len=strlen(name);
```

```
    for(int i=0;i<len;i++){
        ans=ans*26+name[i];
    }
    /*如果还有数字的话
     ans=ans*10+(name[len]-'0');
     */
    return ans;
}

//将字符串使用字典序排列
bool cmp(int a,int b){
    return strcmp(name[a],name[b])<0;
    //不能直接写成name[a]<name[b];
    //而且不能写==-1，只能写<0;
}

//map,string的初始化
string numTostr[170];
map<string,int> strTonum;
for(int i=0;i<13;i++){
    numTostr[i]=unitdigit[i];//unitdigit[i]是个字符串
    strTonum[unitdigit[i]]=i;
}

//清空栈
1) s.clear();
2)while(!s.empty(){
    s.pop();
}

//队列实现败者树问题
while(q.size()!=1){
    if(temp%ng==0) group=temp/ng;//计算分组数
    else group=temp/ng+1;
    for(int i=0;i<group;i++){
        int k=q.front();//存储最大老鼠编号
        for(int j=0;j<ng;j++){//对于组内
            int front=q.front();
            if(mouse[front].weigh>mouse[k].weigh){
                k=front;
            }
            mouse[front].rank=group+1;
            q.pop();
        }
        q.push(k);
    }
    temp=group;
}
mouse[q.front].rank=1;

//对于静态链表，获取有效结点
struct Node{
    int data,next;
```

```
    bool flag;
}node[MAX];

bool cmp(Node a,Node b){
    if(a.flag==false||b.flag==false) return a.flag>b.flag;
    //就是实现了有效的结点都往前排
    else{
        return a.data<b.data;
    }
}

int main()
{
    //初始化
    for(int i=0;i<MAX;i++){//注意这里是对所有的MAX结点，不是n；
        node[i].flag==false;
    }

    //读入数据

    //开始寻找有效结点
    int cnt=0;//有效结点的个数
    while(p!=-1){
        node[p].flag=true;
        cnt++;
        p=node[p].next;
    }

    sort(node,node+MAX,cmp);
    //这样前N个就是有效结点
    for(int i=0;i<count;i++){
        ''''''
    }
}

//对于树的DFS
vector<int> child[MAX];//使用二维数组
void DFS (int root,int height) {
    if(child[root].size()==0){
        if(height>maxheight){
            maxheight=height;
            num=1;
        }
        else if(height==maxheight){
            num++;
        }
        return;
    }
    for(int i=0;i<child[root].size();i++){
        DFS(child[root][i],height+1);
    }
}
```

```
//STL操作
//vector，p.end()是尾元素的下一个地址，左闭右开；
1）创建
        vector<type> name;
        eg:vector<int> p;vector<int> p[MAX]
2）增加
        p.push_back();
        p[i].push_back();
3）删除
        1)p.pop_back();
        2)p.erase(p.begin()+i)//删除第i个位置的元素
        3）p.erase(p.begin()+1,p.begin()+4)//删除1，2，3元素，没有4.
5）查询
        //下标的方式
        p[i];
        p[i][j];//和一般数组没区别

        //迭代器
        1）vector<int>::iterator it=p.begin();
        for(int i=0;i<len;i++){
            printf("%d",*(it+i));//类似于指针
        }
        2）for(vector<int>::iterator it=p.begin();it!=p.end();it++){
            printf("%d",*it);
        }

        //p[i]和*(p.begin()+i)无区别，但是p.begin()+i的写法只有vector和string
6）插入
        p.insert(p.begin()+2,-1)//将-1插入到第二个位置。
7）判空
        p.empty();
8）清空
        p.clear();
9）大小
        p.size();

//set
1）创建
set<int> s;

2）插入
s.insert(x);//将x插入set中

3）删除
s.erase(it);
s.erase(value);
s.(first,end);

5）查询
//只能通过迭代器查询,地址查询
for(set<int>::iterator it=s.begin();it!=s.end();it++){
```

```
    printf("%d",*it);
}

//find()返回目标值的迭代器，值查询
printf("%d",*(find(2)));
6）清空
s.clear();

7）大小
s.size();

//string
1）创建
string str;
2）增加
string str1,str2,str3;//拼接
str3=str1+str2;
3）删除
str.erase(it);删除迭代器为it的元素
str.erase(first,last);
4）修改
str.substr(pos,len)//返回从pos开始的，长度为len的子串。
5）查询
//通过下标访问，和数组无区别
printf("%c",str[i]);

//通过迭代器访问
string::iterator it;
string str="abcdefg";
for(it=str.begin();it!=str.end();it++){
    printf("%c",*it);
}

//使用c_str()将string转化成字符数组，否则只能用cin,cout输入输出元素。
string str="abcdefg";
printf("%s",str.c_str());

//find
str.find(str2);//当str2是str子串时，返回其第一次出现的位置。
//如果没成功返回string::npos
str.find(str2,pos)//从pos位置开始适配str2;
if(str.find(str2)!=string::npos)

6）插入
string str1,str2;
str1.insert(str.begin()+3,str2);//在str.begin()+3位置插入str2
str.insert(it,it1,it2)//在str,it位置，插入第二个字符串的it1到it2位置
7）比较
//可以直接比较，不用想字符数组使用strcmp()
string str1;str2;
if(str1==str2),,,,
```

8）大小
```
str.length();//两种的效果一样。
str.size();
```
9）清空
```
str.clear();
```

```
//map
```
1）创建
```
map<char,int> mp;
map<char,set<int>> mp;
```
2）删除
```
mp.erase(it);
mp.erase(key);//还可以支持删除键
mp.erase(first,last);
```
3）查询
```
//以字符为索引，类似支持按照值来查询
mp['c']=20;
//按照迭代器查询,同一个迭代器访问键和值
for(map<char,int>::iterator it=mp.begin();it!=mp.end();it++){
    printf("%c %d",it->first,it->second);
}
//find
map<char,int>::iterator it=find('c');
printf("%c %d",it->first,it->second);
```

4）清空
```
mp.clear();
```
5）大小
```
mp.size();
```

```
//map,set，会自动按照键从小到大的顺序排序。
```