2009 年

1./*
编写函数 int locStr(char* str1,char* str2)实现字符串匹配定位功能
若在，则返回位置，不在返回-1
相当于在重写 find 函数吧，返回的是第一个字符相同的位置(前提是存在)
***/**

```
#include <iostream>
using namespace std;
int locStr(char* str1,char* str2)
{
    bool flag=false;//标记是否找到
    for(int i=0;i<strlen(str1);i++)
    {
        for(int j=0;j<strlen(str2);j++)
        {
            //先找到字符串相同的起始位置，再在这基础上往后看相同不
            if(str1[i+j]!=str2[j])
            {
                flag=false;
                break;
            }
            flag=true;
        }
        if(flag)
        {
            return i;
        }
    }
    return -1;
}
int main()
{
    //指定字符串
    char* str1="how are you";
    char* str2="aru";
    int k=locStr(str1,str2);
    if(k!=-1)
    {
        cout<<str2<<"在"<<str1<<"的位置是"<<k+1<<endl;
    }else{
        cout<<str2<<"不在"<<str1<<"中"<<endl;
```

```
    }
    return 0;
}
```
**2./*
冒泡排序
*/**
```
#include <iostream>
using namespace std;
int main(){
    int* a;//动态数组
    int n,i,j,temp;
    cout<<"输几个数"<<endl;
    cin>>n;
    a=new int[n];
    for(i=0;i<n;i++)
    {
        cin>>a[i];
    }
    cout<<"输入的数为:"<<endl;
    for(i=0;i<n;i++)
    {
        cout<<a[i]<<"    ";
    }
    cout<<endl;
    //排序
    for(i=0;i<n-1;i++)
    {//n 个数一共排 n-1 次
        /*如：2 1 5 3 0
        第一次：3>0,将 0 与 3 进行对换
        第二次:5>0,0 继续与 5 对换
        .....
        直至 j=0 时，此时 2>0，再次对换。那么此时最小值 0 在第一个数了，下次就不会再比较了(0 最小了)。
        那么 i=1，下次比较到第二个数就截止了
        */
        for(j=n-1;j>i;j--)//从后往前依次将最小的数升至最上边
        {
            if(a[j-1]>a[j])
            {
                temp=a[j];
                a[j]=a[j-1];
                a[j-1]=temp;
            }
        }
```

```
    }
    cout<<"排序后数为:"<<endl;
    for(i=0;i<n;i++)
    {
        cout<<a[i]<<"    ";
    }
    cout<<endl;
    //释放存储空间
    delete[] a;
    return 0;
}
```

2009 保研

1./*
求解一元二次方程组:aX^2+bX+c=0;
首先确定好分类情况(if 情况)
(1)a==0
b==0：无
b!=0：一个
(2)a!=0
dert>0：一个
dert=0：一个
dert<0:虚根
***/**

```
#include <iostream>
#include <cmath>
using namespace std;
int main()
{
    float a,b,c,dert,x1,x2;
    cout<<"输入系数 a,b,c："<<endl;
    cin>>a>>b>>c;
    //dert>0:有两个正解
    if(a!=0)
    {
        dert=b*b-4*a*c;
        if(dert>0)
        {
            dert=sqrt(dert);
            x1=(-b+dert)/2*a;
            x2=(-b-dert)/2*a;
            cout<<"有两个解:"<<endl
```

```
            <<"x1="<<x1<<endl
            <<"x2="<<x2<<endl;
        }else if(dert==0){//dert<0:仅有一个
            x1=(-b)/2*a;
            cout<<"仅有一个解:"<<endl
            <<"x="<<x1<<endl;
        }else{//dert<0:有虚根
            int d=sqrt(-dert)/2*a;
            int e=(-b)/2*a;
            cout<<"虚根为:"<<endl<<
            e<<"+-"<<d<<"i"<<endl;
        }
    }else{
        if(b==0){
            cout<<"无解"<<endl;
        }
        else{
            cout<<"有一个解:"<<-c/b<<endl;
        }
    }
    return 0;
}
```

2.冒泡排序

3./*

使用递归求某个正整数的数字之和

int sumDigits(int n);

如：sumDigits(123456)返回值为 21

*/

```
#include <iostream>
using namespace std;
//显然巧妙的地方在于这个递归调用
int sumDigits(int n)
{
    if(n/10==0)//最后一次的调用，也就是 n 已经只是个各位数了，不用再接着除了
    {
        return n%10;//返回的是最高位的数字
    }else{
        int sum=n%10;
        //递归调用，直至全部调用完，也就是执行 if 函数的时候，开始返回 n%10 的值了，最后 sum 是个总和，返回这个和
        sum+=sumDigits(n/10);
        return sum;
    }
```

```
}
int main()
{
    int n;
    cout<<"输入数字:"<<endl;
    cin>>n;
    cout<<n<<"的各个位的数字之和为:"<<sumDigits(n)<<endl;
    return 0;
}
```

4 胖胖闯关

```
.#include <iostream>
using namespace std;
//开始闯关
void game(int weight,int** a,int n)
{
    int sum=0,i=0,j=0;
    for(i=0;i<n;i++)
    {
        //
        for(;j<n;j++)
        {
            //无论胖胖是撞倒别人，还是被撞倒。都要记录下凡是遇到过的人体重，也就是元素值
            sum+=a[i][j];
            //但是被撞倒了，就不能再在当前行上了，掉到下一行的当前列，所以内层 for 循环的列 j 是不能初始化的
            //所以巧妙的地方就在于 j 的值是只有一次初始化，无论行怎么变化，j 都是继续增加的
            if(weight<a[i][j])
            {
                break;
            }
        }
    }
    //只有走到了列的尽头才说明走出去了，而至于行的尽头是无法说明通关了的
    if(j==n)
    {
        cout<<"胖胖闯关成功,且撞到的人总体重为:"<<sum<<endl;
    }
    else{
        cout<<"胖胖闯关失败,且撞到的人总体重为:"<<sum<<endl;
    }
}
```

```
int main()
{
    //在矩阵运算中也要用到这种方式的二维数组，学会如何申请空间以及释放空间
    int** a;
    int n,i,j,weight;
    cout<<"输入几行几列："<<endl;
    cin>>n;
    //申请动态内存空间 n*n
    a=new int*[n];
    for(i=0;i<n;i++)
    {
        a[i]=new int[n];
    }
    //输入矩阵的值
    for(i=0;i<n;i++)
    {
        cout<<"输入第"<<i+1<<"行元素:"<<endl;
        for(j=0;j<n;j++)
        {
            cin>>a[i][j];
        }
    }
    cout<<"您输入的"<<n<<"*"<<n<<"队列为:"<<endl;
    for(i=0;i<n;i++)
    {
        for(j=0;j<n;j++)
        {
            cout<<a[i][j]<<"    ";
        }
        cout<<endl;
    }
    cout<<"输入胖胖的体重:"<<endl;
    cin>>weight;

    //胖胖开始闯关
    game(weight,a,n);

    //释放内存空间
    for(i=0;i<n;i++)
    {
        delete[] a[i];
    }
    delete[] a;return 0;}
```

2011 年

1. 快速排序

```
/*
快速排序
基本思想:（1）数列中找一个基数
          （2）将比这个基数大的放在基数的右边，比基数小的放在左边
          （3）重复 1 和 2 步骤，直到各区间只有一个数

           0 1 2 3 4 5 6 7 8   9
  举例如：6 1 2 7 9 3 4 5 10 8
第一轮:首先找第一个基准数，为了方便就是第一个数 6 吧：从头往后，从后往前开始遍历(i=0,j=9):
        找到第一个比 6 小的数 5，j=7；第一个比 6 大的数 7，i=3.交换
        6 1 2 5 9 3 4 7 10 8
        再继续 i++,j--;又找到 4，j=6；9，i=4；交换
        6 1 2 5 4 3 9 7 10 8
        再继续 i++,j--；同时找到 3，此时 i=j=4；结束第一次循环，且 3 小于 6，放在左边
        3 1 2 5 4 6 9 7 10 8
第二轮:3 1 2 5 4 和  9 7 10 8 两段分别重复第一轮，基数也默认为最左边的
.....
最后直到区间只有一个元素了，结束
*/

#include <iostream>
using namespace std;

void quickSort(int a[],int left,int right)
{
    int i,j,t,temp;
    //left>right 的时候就是区间只有一个元素的时候
    if(left>right)
    {
        return;
    }

    i=left;
    j=right;
    temp=a[left];//基准数始终在最左边
    while(i!=j)
    {
```

```
        //必须先从右开始找小于基准数的数
        while(a[j]>=temp&&i<j)
        {
            j--;
        }
        while(a[i]<=temp&&i<j)
        {
            i++;
        }
        //交换找到的 i,j 对应的数值
        if(i<j)
        {
            t=a[i];
            a[i]=a[j];
            a[j]=t;
        }
    }
    //基准数归位，放到中间位置(左边是比基准数小的，右边是比基准数大的)
    a[left]=a[i];
    a[i]=temp;

    //对基准数左区间的数排序
    quickSort(a,left,i-1);
    //右区间
    quickSort(a,i+1,right);

}
int main()
{
    //测试数据:6 1 2 7 9 3 4 5 10 8
    int* a;
    int n,i;
    cout<<"准备输入几个数:"<<endl;
    cin>>n;
    a=new int[n+1];
    cout<<"输入数据:"<<endl;
    for(i=1;i<=n;i++)
    {
        cin>>a[i];
    }

    quickSort(a,1,n);

    cout<<"排序之后:"<<endl;
```

```
    for(i=1;i<=n;i++)
    {
        cout<<a[i]<<"   ";
    }
    cout<<endl;
    return 0;
}
/*
排序：1.插入排序：直接插入排序、折半排序、希尔排序
      2.交换排序：冒泡排序、快速排序
      3.选择排序：简单选择排序、堆排序
      4.归并排序
      5.基数排序
*/
#include <iostream>
using namespace std;
/*直接插入排序：
基本思想：将一个记录插入到一个已经排好序的有序表中，从而得到一个新的记录+1 的有序表
如
初始      i= 0 1 2 3 4 5 6 7 8 9
          值=5 2 6 0 3 9 1 7 4 8

 i=1:2<5,哨兵 temp=2，j=0，5>2,那么 5 往前移动一个位置，取代 2 的位置，结束内层 for，2 插入到原来 5 的位置
temp=2       2 5 6 0 3 9 1 7 4 8

 i=2：6<5 不符合

 i=3: 0<6 符合，哨兵 temp=0，j=2，6>0,那么 6 往后移动，取代 0 的位置。j=1,5>0,那么 5 取代原来 6 的位置。j=0,2>0,那么 2 取代原来 5 的位置，结束内层循环，0 取代 2 的位置
temp=0       2 5 6 6 3 9 1 7 4 8
             2 5 5 6 3 9 1 7 4 8
             2 2 5 6 3 9 1 7 4 8
             0 2 5 6 3 9 1 7 4 8
*/

//直接插入排序
void directInsertSort(int a[],int n){
    int i,j,temp;//temp 就是个哨兵，
    for(i=1;i<n;i++)
    {
        if(a[i]<a[i-1])
```

```
        {
            temp=a[i];//哨兵记录下来
            for(j=i-1;a[j]>temp;j--)
            {
                a[j+1]=a[j];
            }
            a[j+1]=temp;
        }
    }
}
/*
冒泡排序：
基本思想：两两相邻记录的关键字，如果反序则交换，直到无反序为止。要一直交换,像其气泡一样往上冒
*/
void bubbleSort(int a[],int n){
    int i,j,temp;
    for(i=0;i<n-1;i++)
    {
        for(j=n-1;j>i;j--)
        {
            if(a[j]<a[j-1])
            {
                temp=a[j];
                a[j]=a[j-1];
                a[j-1]=temp;
            }
        }
    }
    cout<<"冒泡排序之后:";
}
/*
选择法排序：
思想：直到找到最小的值再交换
*/
void selectSort(int a[],int n)
{
    int i,j,k,temp;
    for(i=0;i<n-1;i++)
    {
        k=i;
        for(j=i+1;j<n;j++)
        {
            if(a[j]<a[k])
```

```
                    k=j;
            }
            if(k!=i)
            {
                temp=a[k];a[k]=a[i];a[i]=temp;
            }
        }
}
/*希尔排序*/
void sheelSort(int a[],int n){
    int i,j,temp;//temp 就是个哨兵，
    int gap=n;//gap 就是排序的间隔
    do{
        gap=gap/3+1;
        for(i=gap;i<n;i++)
        {
            if(a[i]<a[i-gap])
            {
                temp=a[i];//哨兵记录下来
                for(j=i-gap;a[j]>temp;j-=gap)
                {
                    a[j+gap]=a[j];
                }
                a[j+gap]=temp;
            }
        }
    }while(gap>1);
}

/*输出数组*/
void printf(int a[],int n)
{
    for(int i=0;i<n;i++)
    {
        cout<<a[i]<<"    ";
    }
    cout<<endl<<endl;
}

int main(){
    int a[]={5,2,6,0,3,9,1,7,4,8};
    int n=10;

    //直接插入排序
```

```
    //directInsertSort(a,n);
    //printf(a,n);

    //bubbleSort(a,n);
    //printf(a,n);

    //selectSort(a,n);
    //printf(a,n);

    //sheelSort(a,n);
    //printf(a,n);
    return 0;
}
```

**2./*
问题：用户输入一共几个猴子，猴子依次报号，用户指定报到数字几就退出，直至最后剩下一个猴子
*/**

```
#include <iostream>
using namespace std;
int main()
{
    int* p;//动态数组：根据一共多少只猴子来分配数组大小
    int n,i;//总数
    int m=0;//退出的猴子数
    int flag;//喊到几退出
    cout<<"输入几个猴子:"<<endl;
    cin>>n;
    cout<<"喊到几退出"<<endl;
    cin>>flag;
    //申请 n 个猴子的空间
    p=new int[n];

    //给每个猴子从 1-n 进行编号
    for(i=0;i<n;i++)
    {
        p[i]=i+1;
    }

    //开始从头喊吧：k 当做报数的变量，直到 k=flag 的时候当前猴子退出
    int k=0;//标记从 1-flag 数
    i=0;//i 就是控制总人数变化循环的

    while(m<n-1)
```

```
    {
        //猴子数值为 0，说明猴子已经被淘汰了，就不能再参与选大王了
        if(p[i]!=0)
        {
            k++;
        }
        //当猴子喊到了 flag，那么就退出了，没有资格当大王了
        if(k==flag)
        {
            p[i]=0;//标记为 0
            cout<<"猴子:"<<i+1<<"退出"<<endl;
            k=0;//重新开始报数
            m++;//退出的猴子数+1
        }
        i++;
        //喊到了最后一个猴子，也就是 i==n 那么从头接着喊，且不能断
        if(i==n)
        {
            i=0;
        }
    }

    //找那个数值不是 0 的猴子，就是大王了
    for(i=0;i<n;i++)
    {
        if(p[i]!=0)
        {
            cout<<"大王是猴子:"<<i+1<<endl;
        }
    }

    //释放存储空间
    delete[] p;
    return 0;
}
```

2012 年专硕

```
2./*
公式：圆周长：2*PI*r
        面积：PI*r*r
       圆球表面积:4*PI*r*r*r
       圆球体积：4/3*PI*r*r
       圆柱体积：H*PI*r*r
       圆锥体积：(1.0/3.0)*H*PI*r*r
*/
#include <iostream>
#include <iomanip>//要使用带参数的操作符函数，如 setw()、setfill()、setprecision()
#include <cmath>//使用 pow()指数函数
using namespace std;
int main(){
    const float R=1.50;
    const float H=3.00;
    const float PI=3.1415926;
    //
    cout<<setiosflags(ios_base::fixed)<<setprecision(2);
    cout<<"圆周长:"<<2*PI*R<<endl;
    cout<<"圆周长:"<<PI*pow(R,2)<<endl;
    cout<<"圆球表面积:"<<4*PI*pow(R,2)<<endl;
    cout<<"圆球的体积:"<<(4.0/3)*PI*pow(R,3)<<endl;
    return 0;
}
3./*
问题描述：如输入 12345，则输出：5
                                    1-2-3-4-5
                                54321
*/
#include <iostream>
using namespace std;
int main(){
    cout<<"输入一个数:"<<endl;
    long a;
    cin>>a;
    int temp;
    int temp2;
    int res=0;
    int len=0;//几位数
    while(true)
    {
        //辗转相除
```

```
        if(a!=0)
        {
            temp=a%10;//每一位的数字，不断变化的
            a=a/10;//a 要循环整除
            res=res*10+temp;//逆序之后的数
            len++;//几位数
            //直到做到 a 整除之后为 0 的时候，那么也就判断出是几位数了
            if(a==0)
            {
                temp2=res;
                cout<<len<<endl;
            }
        }else{
            //注意 temp2 是逆序哦
            //第一次的时候 temp2 是等于 res 的，都是逆序如 54321，直到被整除依次之后才开始用"-"来分割各个数字，
            //显然不可能上来就有"-"吧
            if(temp2!=res)
            {
                cout<<"-";
            }
            //如同 if 语句一样，辗转相除吧，获取每一位的值并输出到屏幕上
            cout<<temp2%10;
            temp2/=10;
            //直至全部获取完毕，直接 break 结束循环就可以了
            if(temp2==0)
            {
                cout<<endl;
                //顺带输出逆序
                cout<<res<<endl;
                break;
            }
        }
    }
    return 0;
}
```

4./*

用户设置一个密码箱：默认初始密码为:0,用户可以指定密码

用户输入密码正确，箱子打开，可以进行修改密码，往箱子里添加字符串，完毕之后关闭

输入密码错误就无法打开

设计:密码箱类

1.确定私有数据成员:密码、存放的字符串、最重要的是箱子的状态!（状态要贯穿成员函数的始终，只有是打开状态才能操作一系列功能

2.重载构造函数:初始化箱子密码(0)和状态(关闭)
3.成员函数:
（1）开箱
（2）关箱
（3）改密码:传新的密码参数进去
（4）改箱子内容:传新内容进去
（5）查看看箱子内容
以上皆需要首先判断箱子的 status，关闭则一切免谈

*/

```
#include <iostream>
#include <string>
using namespace std;
class safeBox{
public:
    //1.重载构造函数
    safeBox(int Pass=0):password(Pass){
        status=false;
    }
    //可以使用默认构造函数，也可以使用重载的构造函数默认 Pass=0；
    /*safeBox(){
        password=0;
        status=false;
    }*/
    //2.成员函数
    //开箱
    void open(int Password){
        if(status){
            cout<<"箱子已经打开，无需再打开!!!"<<endl;
        }else{
            if(Password==password){
                cout<<"箱子打开"<<endl;
                status=true;
            }else{
                cout<<"密码错误"<<endl;
            }
        }
    }
    //关箱
    void close(){
        if(!status){
            cout<<"箱子已经关闭，无需再关闭"<<endl;
        }else{
            cout<<"关闭箱子"<<endl;
```

```
            status=false;
        }
    }
    //更改密码
    void setPassword(int newPassword){
        if(!status){
            cout<<"箱子关闭，请先打开箱子"<<endl;
        }else{
            password=newPassword;
            cout<<"更改密码成功,自动为您关闭箱子"<<endl;
            status=false;
        }
    }
    //查看箱子内容
    void getString()const{
        if(!status){
            cout<<"箱子关闭,请先打开箱子"<<endl;
        }else{
            if(str==""){
                cout<<"箱子为空"<<endl;
            }else{
                cout<<str<<endl;
            }
        }
    }
    //往箱子里面添加数据
    void setString(const string& s){
        if(!status){
            cout<<"箱子关闭，请先打开箱子"<<endl;
        }else{
            str=s;
            cout<<"添加内容成功"<<endl;
        }
    }
private:
    //1.确定数据成员
    int password;//密码
    string str;//密码箱中存储的字符串
    bool status;//箱子的状态
};

//主函数
int main(){
    safeBox box1;
```

```
    box1.open(0);
    box1.close();
    box1.open(0);
    box1.setPassword(123);
    box1.open(123);
    box1.getString();
    box1.setString("assss");
    box1.getString();
    return 0;
}
```

2012 年学硕

```
1./*
输出 2-500 之间的质数
质数:从 2 开始只能被 1 和本身整除的数，显然最小的质数为 2
*/
#include <iostream>
#include <cmath>
using namespace std;
int main()
{
    for(int i=2;i<=500;i++)
    {
        bool flag=true;//标记是否为质数，默认是
        //从 2 开始除，一直除到本身的平方根就可以了
        //注意 2，是最小的质数！
        for(int j=2;j<=sqrt(i);j++)
        {
            //被整除了，那么就没有必要除到平方根了，直接标记不是质数，直接退出内层循环就行了
            if(i!=2&&i%j==0)
            {
                flag=false;
                break;
            }
        }
        //显然只有经受住了实践的检验，最终存活下来的才是真质数
        if(flag){
            cout<<i<<" ";
        }
    }
    cout<<endl;
    return 0;
}
2.#include <iostream>
using namespace std;
#define PI 3.1415
/*父类"房间类"：因为是 Room 虚基类，所以其派生类(方形房间类和圆形房间类)必须继承某些函数(纯虚函数 S)，
                否则派生类就不能实例化，所以相当于是个公共接口，被所有其派生类共享，且必须重写
                抽象基类也不能实例化抽象对象
*/
```

```
class Room{
public:
    //只需要在虚函数声明时，后面加上"=0"，就可以声明纯虚函数了
    virtual float S()=0;
};
//长形房间类
class FRoom:public Room{
public:
    //构造函数初始化长宽
    FRoom(const float X=0,const float Y=0):x(X),y(Y){}
    //继承父类的纯虚成员函数 S
    float S()
    {
        return x*y;
    }
private:
    float x,y;
};
//圆形类
class YRoom:public Room{
public:
    YRoom(const float R=0):r(R){}
    float S()
    {
        return PI*r*r;
    }
private:
    float r;
};
//计算租金类
class Money{
public:

    //注意：在创建 Money 对象时，需要传入 Room 类型的引用，也就是可以是其派生类的 YRoom 或 FRoom 类型都可以
    Money(Room *p,float Z,float Y,float T):zujin(Z),yongjin(Y),time(T)
    {
        //因为相当于实例化了一个方形类 p 或圆形类 p，所以就可以调用里面公有的虚函数 S 了。
        sum=p->S()*(zujin*time+yongjin);
    }
    //获取租金值
    float allMoney()
    {
```

```
            return sum;
        }
private:
        float zujin;
        float yongjin;
        float time;
        float sum;
};
int main(){
        YRoom y1(2);
        FRoom f1(1,2);
        //分别建 m1 圆形房间的租金和方形房间的租金对象，那么分别传入自己不同的类型
        Money m1(&y1,1,2,3);
        Money m2(&f1,1,2,3);
        cout<<"长形房间租金:"<<m2.allMoney()<<endl;
        cout<<"圆形房间租金:"<<m1.allMoney()<<endl;
        return 0;
}
```

2013 年专硕

1. 三个数从小到大排列

```
#include <iostream>
#include <string>
using namespace std;
//使用模板函数找最小值
template<class T>
T find(T* a,int n)
{
    //打擂台算法
    T min=a[0];
    for(int i=1;i<n;i++){
        if(min>a[i]){
            min=a[i];
        }
    }
    return min;
}
//使用模板函数来排序，因为 string 类型的是可以直接用<、>、=如同整型一样来比较的，所以我们就可以用模板了
template<class T>
void sort(T* a,int n)
{
    T temp;
    //显然用到简单选择法排序
    for(int i=0;i<n-1;i++)
    {
        int k=i;
        for(int j=i+1;j<n;j++)
        {
            if(a[j]<a[k])
            {
                k=j;
            }
        }
        if(k!=i)
        {
            temp=a[k];
            a[k]=a[i];
            a[i]=temp;
        }
    }
```

```
}
int main()
{
	cout<<"想要输入几个内容:"<<endl;
	int n,i;
	cin>>n;
	string* a;//动态数组空间
	a=new string[n];
	cout<<"请输入:"<<endl;
	for(i=0;i<n;i++){
		cin>>a[i];
	}
	//打擂台算法找出最小值以及下标，用模板函数实现
	cout<<"最小值是:"<<find(a,n)<<endl;
	//排序算法同样用函数模板实现
	cout<<"排序之后:"<<endl;
	sort(a,n);
	for(i=0;i<n;i++){
		cout<<a[i]<<"    ";
	}
	//释放空间
	delete[] a;

	cout<<endl<<"想要输入几个内容:"<<endl;
	cin>>n;
	int* a2;
	a2=new int[n];
	cout<<"请输入:"<<endl;
	for(i=0;i<n;i++){
		cin>>a2[i];
	}
	cout<<"最小值是:"<<find(a2,n)<<endl;
	cout<<"排序之后:"<<endl;
	sort(a2,n);
	//输出看看
	for(i=0;i<n;i++){
		cout<<a2[i]<<"    ";
	}
	delete[] a2;
	return 0;
}
```

2. 实现字符串类型的一些函数:find、substr、replace、split

自己实现字符串类型的一些函数：

如：this 1s string
提取指定位置的字符串函数：substr(开始位置，提取长度，被赋值的字符串 str)
substr(5,2,str);把 1s 子串赋值给 str:
拆分字符串成两段函数：split("1s",str1,str2);以 is 为中间，将字符串拆成 this 和 string 分别给 str1 和 str2
因此需要知道"1s"是否在字符串中，若在则其位置，然后分成两段分别给 str1，str2

替换字符串函数：replace(5,2,"i");此时字符串为 this is string

删除指定位置的字符串函数:erase(2,6);此时字符串为 th string

删除整个字符串函数：dele（）

/*

substr、split、replace、erase、dele 函数的实现类

*/

```
#ifndef _STRING_H_
#define _STRING_H_
class String{
public:
    //2.重载构造函数、析构函数
    String(const char* c){
        //若使用 string 类型，直接一句：str=c 就可以了
        len=strlen(c);
        p=new char[len+1];//字符数组最后一位为'\0'，所以+1
        strcpy(p,c);//使用 c 语言的 strcpy 函数将字符数组 c 拷贝给 p，千万不能用 string 类型的等号来拷贝
    }
    ~String(){
        delete[] p;
    }
    //3.显示当前数组元素
    void show()const;
    //4.substr 函数，传入开始位置和长度，无返回值，直接传字符串地址进来
    void substr(const int,const int,char*);
    //5.实现 split 方法之前需要判断要分割处的的字符串是否在原字符串中，若不在那就没有替换的必要了
    //若在，那么就返回其开始位置
    //所以 find 函数的重写是比较关键的，其他函数好写点
    int isFind(const char*);
    //6.实现 split 方法
```

```
    void split(const char*,char*,char*);
    //7.实现替换功能
    void replace(const char*,int);
    //8.删除指定位置的字符串
    void erase(int,int);
    //9.删除整个字符串
    void del();
private:
    //1.确定数据成员
    //是可以使用 string 类型的，会更加方便的
    //string str;
    char* p;
    int len;//字符数组的长度
};
#endif
/*
```

String 类方法的实现

```
*/
#include <iostream>
#include <string>
using namespace std;
#include "string.h"
void String::show()const{
    cout<<"当前字符串为："<<endl;
    for(int i=0;i<strlen(p);i++)
        cout<<p[i];
    cout<<endl;
}
void String::substr(const int start,const int len,char* s){
    for(int i=0;i<len;i++){
        s[i]=p[start+i];//直接定位到子串在父串开始的位置，然后逐个字符提取到
s 中
    }
    s[i]='\0';
}
int String::isFind(const char* c){
    bool flag=false;//标志子串是否在字符串中
    for(int i=0;i<strlen(p);i++)
    {
        for(int j=0;j<strlen(c);j++)
        {
            if(p[i+j]!=c[j])
            {
```

```
                flag=false;//一定要注意 flag:若测试为第一个字符相同，但是第二个字符不同，那么在匹配第二个的时候不同，
                //必须将 flag 归为 flag，否则就会沿用上一次的 true 了，只是第一个字符相同而已
                    break;
                }
                flag=true;
            }
            if(flag){
                return i;
            }
        }
        return -1;
}
void String::split(const char* c1,char* c2,char* c3){//从 c1 位置开始切割成左右两段到 c2，c3 中
        int k=isFind(c1);//k 就是 c1 在字符串中的位置下标
        if(k!=-1)
        {
            for(int i=0;i<k;i++){
                c2[i]=p[i];
            }
            c2[i]='\0';
            //右段不好弄
            for(int j=0;j<strlen(p)-k-strlen(c1);j++){
                c3[j]=p[strlen(c1)+k+j];
            }
            c3[j]='\0';
        }else{
            cout<<c1<<"不在字符串中，无法完成替换!!! "<<endl;
        }
}

void String::replace(const char* c,int location){
        for(int i=0;i<strlen(c);i++){
            p[location++]=c[i];
        }
}

void String::erase(int location,int len){
        for(int i=location;i<strlen(p);i++)
        {
            p[i]=p[i+len];//后面的往前移动
        }
```

```
}
void String::del(){
    for(int i=0;i<strlen(p);i++){
        p[i]='\0';//全部设为空
    }
}
/*
测试编写的方法是否可行
*/
#include <iostream>
#include "string.h"
using namespace std;
int main(){
    String s("this 1s string");
    s.show();

    char c1[30];
    char c2[30];
    char c3[30];

    s.substr(5,2,c1);
    cout<<"测试 substr 函数,提取子串:"<<endl;
    cout<<c1<<endl;

    cout<<"测试 find 函数是否成功:"<<endl;
    cout<<s.isFind(c1)<<endl;

    cout<<"测试 split 函数是否能正确找到切割处的子串:"<<endl;
    s.split("asd",c2,c3);

    cout<<"测试 split 函数:"<<endl;//从"1s"位置开始切割成左右两段到 c2，c3 中
    s.split("1s",c2,c3);
    cout<<c2<<endl;
    cout<<c3<<endl;

    cout<<"测试 replace 函数:"<<endl;
    s.replace("i",5);
    s.show();

    cout<<"测试 erase 函数"<<endl;
    s.erase(3,7);
```

```
    s.show();

    cout<<"测试 del 全部删除函数"<<endl;
    s.del();
    s.show();
    return 0;
}
```

4.比 2014 学硕要简单的手机通讯录的实现：主要是在删除联系人和去重、同步更新问题上注意下：

```
/*
通讯录:添加、查询、删除、去重、同步更新
*/
#ifndef _CONTACT_H_
#define _CONTACT_H_
#include "record.h"
#include <fstream>
class Contact{
public:
    //构造与析构函数
    Contact()
    {
        r=new Record[255];
        number=0;
    }
    ~Contact(){
        delete[] r;
    }
    //向联系人数组中添加联系人基本信息
    void addFriend(const string&,const string&,const string&);
    //显示全部的联系人信息
    void showAll()const;
    //通过姓名查找
    int findName(const string&,bool)const;
    //通过姓名删除指定联系人
    void delName(const string&);
    //去重:寻找联系人列表中姓名相同的人，不考虑重名的因素
    void delRepate();
    //同步更新(从文件中读取新联系人到数组中以及将当前联系人重新放回文件中去)
    void upDate();
private:
    Record* r;
    int number;
};
```

```
void Contact::upDate()
{
    //先从文件中读取联系人数组中没有的联系人
    ifstream in("list.txt");
    string str;
    //getline 从文件中读取到的数据是一行行的，也就是一个完整的联系人信息
(如：张三%13899%1064@qq.com)
    //我们约定是文件中以%分割联系人不同信息
    while(getline(in,str))
    {
        string Name;
        int k=str.find("%");
        Name=str.substr(0,k);
        //从文件中取出的姓名看是否在联系人数组中
        if(findName(Name,false)==-1)
        {
            str=str.substr(k+1,str.length()-k-1);
            k=str.find("%");
            string Phone=str.substr(0,k);
            str=str.substr(k+1,str.length()-k-1);
            string Email=str;
            addFriend(Name,Phone,Email);
        }

    }
    in.close();
    //再将当前联系人放回原文件(有可能原文件中也不存在当前数组中的联系
人)
    ofstream out("list.txt");
    for(int i=0;i<number;i++)
    {
        out<<r[i].getName()<<"%"<<r[i].getPhone()<<"%"<<r[i].getEmail()<<endl;
    }
    out.close();
}
void Contact::delRepate()
{
    string tempName;
    for(int i=0;i<number;i++)
    {
        //从联系人数组的前面开始依次向后找是否有与其重名的
        tempName=r[i].getName();
        for(int j=i+1;j<number;j++)
        {
```

```
                //找到之后调用 deleName 函数直接删除
                if(r[j].getName()==tempName)
                {
                    delName(tempName);
                }
            }
        }
}
void Contact::delName(const string& name)
{
    int k=findName(name,false);
    if(k!=-1)
    {
        for(int i=k+1;i<number;i++)
        {
            r[i-1]=r[i];
        }
        number--;
    }else{
        cout<<"未找到联系人"<<name<<"无法删除"<<endl;
    }
}
int Contact::findName(const string& name,bool flag)const
{
    for(int i=0;i<number;i++)
    {
        if(name==r[i].getName())
        {
            if(flag)
            {
                cout<<"信息如下:"<<endl;
                cout<<r[i].getName()<<"                    "<<r[i].getPhone()<<"
"<<r[i].getEmail()<<endl;
            }
            return i;
        }
    }
    return -1;
}
void Contact::showAll()const
{
    for(int i=0;i<number;i++)
    {
        r[i].showFriend();
```

```
    }
}
void Contact::addFriend(const string& name,const string& phone,const string& email)
{
    r[number++].setFriend(name,phone,email);
}
#endif
```

2013 年学硕

1. 配置文件信息：也就是 2016 年的第四题

这个程序实际就是手机通讯录的一个子分支而已：
主类：能够进行配置文件信息的手工输入设定（设置分组的名字、分组的属性及属性值），并写入到指定文件中去，
能从文件中读取配置信息到分组数组中，并输出到屏幕上
内部类：记录文件分组信息，因为有好多个分组且一个分组中包含很多配置信息的名字、值，都是不同类型的数据，
所以一个分组就是一个对象，那么我们就需要很多个对象了，于是用内部类实现对象数组
分组的信息就是内部类的私有数据成员，信息的配置就是一些成员函数的编写
这就相当于手机通讯录，分组就相当于一个联系人，分组的信息就相当于一个联系人的信息如姓名、电话、email
但是有一点，小组属性的信息如名字和值是不确定的，可能有上百个信息，这就与联系人的信息有不同了，
联系人的信息基本确定了就 3 个左右。所以我们对分组属性的存储需要用到字符串数组了

设计:
内部类:1.私有数据成员：某一个分组的名字、分组的属性(注意是字符串型数组)、分组属性的值、下标
2.重载默认构造函数：指定一个分组的属性最多不能超过 10 个属性，相应的值也是不会超过 10 个的
3.析构函数：释放属性、属性值空间

4.成员函数：(1)设定一个分组的姓名函数：设置分组名:为什么不跟属性一起设置呢？显然分组名只需要设置一次，
但是属性信息需要设置很多次
(2)设定一个分组的属性及对应的属性值函数：

(3)显示当前分组的信息的函数：主类的显示全部分组信息的函数是可以调用这个函数的，也就是每个分组都去调用

(4)将当前分组信息输入到文件中去存储函数:这只是存储的一个分组，所以主类也是同样多次调用这个函数

，依次将每个分组输出到文件中去。

注意：打开文件的时候要在末尾加 ios_base::app,否则每次写入一个分组就会覆盖之前的分组，

因为我们不是一次性全部输入到文件中去，而是要多次去打开文件

主类：1.私有数据成员：分组对象数组，也就是可以拥有多个分组、数组下标

2.重载默认构造函数：默认分组对象最多 10 个。

3.设定一个分组名字：也就是调用内部类的设定分组名字

4.设定分组的一些属性：调用内部类的函数

//特别注意，我们先创建的一个分组，让 n++了，那么在设定属性的时候此时 n 必须-1 才是你添加的分组哦

5.显示全部分组：循环调用内部类的显示

6.输出到文件中去：也是循环调用

7.从文件中读取配置信息到分组数组中：

依次提取每一行字符串（行结束为标记），那么当读取的第一个字符是"["时，显然它是分组名

否则就是属性+属性值：那么我们以"="为标记，利用 find 函数找到"="号，

左边的是属性名，右边的就是属性值。利用 substr 函数提取对应的字符串到对应的变量中就可以了

```
/*
内部类：
*/
#ifndef _GROUP_H_
#define _GROUP_H_
#include <iostream>
#include <string>
#include <fstream>
using namespace std;
class Group{
public:
    //重载构造函数：设置一个分组的属性最多为 10 个
    Group(){
        pName=new string[10];
        pValue=new string[10];
        num=0;
    }
    //析构函数
    ~Group(){
```

```
        delete[] pName;
        delete[] pValue;
    }
    //设置分组名:为什么不跟属性一起设置呢？显然分组名只需要设置一次，但是属性信息需要设置很多次
    void setGroupName(const string&);
    //设置每一个分组的信息
    void setGroup(const string&,const string&);
    void show()const;//显示当前分组的信息
    void outputFile()const;//将当前分组输出到文件中去

private:
    string GName;//小组的名字
    string *pName;//存储小组各个属性名字的数组
    string *pValue;//属性对应的数值
    int num;//记录多少个属性
};
void Group::outputFile()const
{
    ofstream out("setting.ini",ios_base::app);//app 后缀，那么写入操作不会删除原来的数据，只是在末尾追加而已
    out<<"["<<GName<<"]"<<endl;
    for(int i=0;i<num;i++)
    {
        out<<pName[i]<<"    ="<<pValue[i]<<endl;
    }
}
void Group::show()const
{
    cout<<"["<<GName<<"]"<<endl;
    for(int i=0;i<num;i++)
    {
        cout<<pName[i]<<"    ="<<pValue[i]<<endl;
    }
}
void Group::setGroup(const string& PN,const string& PV)
{
    pName[num]=PN;
    pValue[num]=PV;
    num++;
}
void Group::setGroupName(const string& GN)
{
    GName=GN;
```

```
}
#endif
/*
主类：能够进行配置文件信息的手工输入并写入到指定文件中去，
      能从文件中读取配置信息到屏幕上
*/
#ifndef _WANDR_H_
#define _WANDR_H_
#include "group.h"
class WriteRead{
public:
    //重载构造函数
    WriteRead(){
        g=new Group[10];//默认一个文件配置信息有 10 个分组
        n=0;//初始分组肯定为空
    }
    ~WriteRead(){
        delete[] g;
    }
    //添加分组名
    void addGroupName(const string&);
    //添加一个分组属性:属性名+值
    void addGroup(const string&,const string&);
    //显示全部分组：使用内部类的方法
    void show()const;
    //全部分组输出到文件中去，使用内部类的输出方法
    void outputFile()const;

    //获取文件中数据到分组中
    void getGroup();
private:
    Group* g;//创建对象数组
    int n;//控制分组个数
};
void WriteRead::getGroup()
{
    cout<<"正在读取配置文件信息..."<<endl;
    ifstream in("setting.ini");
    string str;
    while(getline(in,str))//将文件中数据依次读入到分组中
    {
        if(str.find("[")==0)//若读到的字符串的第一个字符是"["，显示读的是分组
名而不是属性
        {
```

```
            addGroupName(str.substr(1,str.length()-2));
        }else{
            //读的是属性
            string s1,s2;
            int k=str.find("=");//找到中间的=位置分割
            s1=str.substr(0,k-2);
            s2=str.substr(k+1,str.length()-k-1);
            addGroup(s1,s2);
        }
    }
    cout<<"读取完毕,文件配置信息如下:"<<endl;

}
void WriteRead::outputFile()const
{
    for(int i=0;i<n;i++)
    {
        g[i].outputFile();
    }
}
void WriteRead::show()const
{
    for(int i=0;i<n;i++)
    {
        g[i].show();
    }
}
void WriteRead::addGroup(const string& PN,const string& PV)
{
    //特别注意，我们先创建的一个分组，让 n++了，那么此时 n 必须-1 才是你添加的分组哦
    g[n-1].setGroup(PN,PV);
}
void WriteRead::addGroupName(const string& GN)
{
    g[n++].setGroupName(GN);
}
#endif
/*
```

主函数 main

***/**

```
#include "WAndR.h"
```

```
int main(){
	WriteRead g1;
	//添加分组
	g1.addGroupName("path");
	g1.addGroup("Images","aaaa");
	g1.addGroup("log","aaasss");

	g1.addGroupName("Images");
	g1.addGroup("Normal","Butnext");
	g1.addGroup("Hover","sasa");
	g1.addGroup("Down","ddsds");

	g1.show();
	g1.outputFile();

	cout<<"测试读取配置信息..."<<endl;
	WriteRead g2;
	g2.getGroup();
	g2.show();
	return 0;
}
```

2014 专硕

1./*
输入三个数求其均值、方差、标准差。也就是 2017 的第一题
***/**

```
#include <iostream>
#include <cmath>//使用 pow 函数以及 sqrt 函数
using namespace std;
int main(){
	float average;
	float fangcha;
	float biaozhuncha;
	float sum=0.0;
	int i,n=3;
	int* a=new int[3];//若手工输入数字使用动态数组就比较方便了
	cout<<"输入三个数:"<<endl;
	for(i=0;i<n;i++){
		cin>>a[i];
		sum+=a[i];
	}
	average=sum/n;//计算均值
	sum=0.0;
	//循环遍历计算方差
	for(i=0;i<n;i++){
		sum+=pow(average-a[i],2);
	}
	fangcha=sum/n;
	//计算标准差
	biaozhuncha=sqrt(fangcha);
	cout<<"均值:"<<average<<endl
		<<"方差:"<<fangcha<<endl
		<<"标准差:"<<biaozhuncha<<endl;
	return 0;
}
```

2./*
统计字符串中连续出现的字符个数：也就是 2017 的第三题
***/**

```
#include <iostream>
#include <string>
using namespace std;
void toString(string& str){
	for(int i=0;i<str.length();i++)
	{
```

```
            if(str[i]>='a'&&str[i]<='z')
            {
                str[i]-=32;
            }
        }
}
int main()
{
    cout<<"输入字符串"<<endl;
    string str;
    cin>>str;
    //将其全部转换成大写
    toString(str);
    cout<<"全部转换成大写之后:"<<str<<endl;

    int len=1;
    for(int i=1;i<str.length()+1;i++)
    {
        //这样就可以不用临时字符来储存上一个字符了，简单些
        if(str[i]!=str[i-1])
        {//前后进行对比，注意最后一个元素要与'\0'进行对比（i 的位置就是），否则就不能输出最后的字符了
            cout<<str[i-1]<<":"<<len;
            len=0;
            if(i!=str.length())
            {
                cout<<",";
            }
        }
        len++;
    }
    cout<<endl;
    return 0;
}
```

3.用户输入-http://test:12345@192.168.1.1:2121，
提取用户名：test
密码：12345
ip 地址：192.168.1.1
端口号：2121

关键：构造函数初始化数据成员网址后，调用私有的成员函数（把切割的细节隐藏起来，遵循了信息隐藏原则）
实现将网址切割，切割的每一小段分别放到对应的 userName、password、ip、num。
注意：端口的时候是整型，所以我们需要

模板函数将 string 类型的字符串转换成整型。最后用户调用间接获取每个私有成员值就可以了。

步骤:
1.设计类：（1）确定私有数据成员：网址、用户名、密码、ip 地址、端口号(int 型)
（2）重载构造函数：初始化网址，将网址的切割细节隐藏起来，也就是在构造函数中调用私有成员函数 split()，来完成切割
（3）写公有成员函数:能够返回网址、用户名....，也就是设计 5 个数据成员的 get 方法
（4）重点写私有成员函数 split:多次利用 find、substr 方法不断切割字符串，第一次以"@"为中心切两半
左侧再以"/"为中心切两半：--test:12345---那么再以"："为中心，左侧就是用户名，右侧就是密码
右侧再以"："为中心切两半，左侧就是 ip，右侧就是端口。
特别注意在提取右侧值的时候:substr(k+1,str.length()-k-1);k+1 就是起始位置，然后就是提取的长度

```
#include <iostream>
#include <string>
#include <sstream>
using namespace std;

//模板函数用来将 string 类型的字符串转换成 T 类型
template <class T>
T fromString(string& str)
{
    T temp;
    istringstream in(str);//绑定 str
    in>>temp;//输出到 temp 中,完成转换
    return temp;
}

class Website{
public:
    //2.含参构造函数初始化数据成员
    Website(const string& str){
        //4.将字符串分割的细节隐藏起来，调用 stringCut 函数
        stringCut(web=str);
    }
    //3.获取用户名、密码、ip、端口号、网址的成员函数
    string getWeb()const{return web;}
    string getUserName()const{return userName;}
    string getPassword()const{return password;}
```

```
    string getIp()const{return ip;}
    int getNum()const{return num;}
private:
    //1.确定数据成员
    string web;
    string userName;
    string password;
    string ip;
    int num;//只有端口号为 int 类型，因此我们就需要考虑如何将字符串类型的子串转换成整型
    void stringCut(string&);
};
//stringCut 函数实现切割字符串获取用户名、密码、ip、端口信息
void Website::stringCut(string& str){
    int temp=str.find("@");//找到@的位置
    //以@为分界点将字符串分成两部分:
    //左边 str1：http://test:12345
    string str1=str.substr(0,temp);
    //右边 str2：192.168.1.1:2121
    string str2=str.substr(temp+1,str.length()-temp-1);

    //右部分 str1 再以/为分界点
    temp=str1.find("/");
    //右部分又变为：test:12345
    str1=str1.substr(temp+2,str1.length()-temp-2);
    //找到":"位置，那么左边就是用户名，右边就是密码了
    temp=str1.find(":");
    userName=str1.substr(0,temp);
    password=str1.substr(temp+1,str1.length()-temp-1);

    //左部分 str2 再以:为分界点，左边就是 ip，右边就是端口号
    temp=str2.find(":");
    ip=str2.substr(0,temp);
    num=fromString<int>(str2.substr(temp+1,str2.length()-temp-1));//注意：num 是整型，所以用模板函数使 string 类型转换成 int 型的端口


}
int main(){
    Website w1("http://test:12345@192.168.1.1:2121");
    //测试：
    cout<<"网址为:"<<w1.getWeb()<<endl;
    cout<<"用户名:"<<w1.getUserName()<<endl;
    cout<<"密码:"<<w1.getPassword()<<endl;
```

```
    cout<<"ip 地址:"<<w1.getIp()<<endl;
    cout<<"端口号:"<<w1.getNum()<<endl;
    return 0;
}
```

4.微信类的实现：

1.个人信息类：User(先完成这个类，群组类不急设计，群组类是与用户类没有任何联系的，只是在创建群的时候用了用户的姓名而已)

（1）确定私有数据成员：个人基本信息：微信号、qq 号、电话号、email

重点是：存储好友的数组!!!指向下一个好友的下标

添加好友函数：向数组中添加元素，实现信息隐藏原则，把添加的细节隐藏起来，

简单的把微信号添加进来就可以了，不要想的太复杂!!!

（2）成员函数：构造函数：初始化好友数组，指定最多只能 30 个好友　析构函数：释放数组空间

用户个人信息初始化函数：set，一次性将个人基本信息初始化完毕

获取用户个人信息函数：get 的 4 个方法，分别获取微信号、qq 号、电话号、email

查找好友函数(找到之后就调用添加好友函数直接添加进去就可以了)：

注意！传进来的参数：微信成员的数组，里面有很多成员，然后我们去找我们要添加的是否存在，存在就添加到好友数组中

同时注意根据电话查找的时候，因为在根据电话推荐好友的时候也要用到查找电话，所以要判断是否需要添加

根据电话数组推荐好友函数：推荐 5 个好友，传入一个电话数组进去，与用户列表的电话进行对比，也就去调用查找电话函数，看是否存在，存在就推荐给用户

显示全部好友函数

在创建 user 类对象的时候，一次性创建 8 个用户，那么添加好友就是把这 8 个用户传到添加好友函数中，根据微信号等查找是否存在，存在的话就直接添加到数组中去就可以了

2.群类：Group

(1)私有：群名称、群主微信号、

群成员数组、指向下一个群成员的下标

(2)成员函数：构造函数：创建群对象的时候必须指定群名称、群主微信号、群的容量（知道容量了就可以初始化群数组了）、默认容量是 20

析构函数：释放群数组的空间

添加群成员函数：简单的把微信号添加进来就行了

显示群成员

```
/*
首先设计用户类，不忙着设计群组类
*/
#ifndef _USER_H_
#define _USER_H_
#include <iostream>
using namespace std;
class user{
public:
    //构造函数初始化好友数组，指定大小为 30
    user(){
        fri=new string[30];friNum=0;
    }
    //析构函数释放好友数组空间
    ~user(){
        delete[] fri;
    }
    //初始化用户个人基本信息
    void set(const string&,const string&,const string&,const string&);
    //获取个人基本信息
    string getWeiNum()const{return weiNum;}
    string getQQNum()const{return qqNum;}
    string getPhoneNum()const{return phoneNum;}
    string getEmail()const{return email;}
    //显示全部好友列表
    void showFriend()const;
    //查找（按微信号、qq 号、手机号、email）并添加好友
    void findWeiNum(const user*,const string&,const int);
    void findQQNum(const user*,const string&,const int);
    int findPhoneNum(const user*,const string&,const int,bool);

    //推荐好友函数
    void showTuiJianFriend(const user*,const string*,const int);

private:
    string weiNum;
    string qqNum;
    string phoneNum;
    string email;
    string* fri;//用户好友数组
    int friNum;

    //添加好友，遵循信息隐藏原则
```

```
    void addFriend(const string& infor){
        fri[friNum++]=infor;
    }
};
#endif
/*
实现方法，可以放在同一个.h 文件下写
*/
#include "user.h"
#include <iostream>
#include <string>
using namespace std;
void user::set(const string& Wei,const string& QQ,const string& Phone,const string&
Email){
    weiNum=Wei;
    qqNum=QQ;
    phoneNum=Phone;
    email=Email;
}
void user::showFriend()const{
    cout<<"您的好友有:"<<endl;
    for(int i=0;i<friNum;i++){
        cout<<fri[i]<<endl;
    }
}
void user::findWeiNum(const user* u,const string& weiNum,const int len){
    for(int i=0;i<len;i++){
        if(u[i].getWeiNum()==weiNum){
            this->addFriend(weiNum);
            cout<<"成功添加好友"<<weiNum<<endl;
            break;
        }
    }
    if(i==len){
        cout<<"未找到好友"<<weiNum<<endl;
    }
}
void user::findQQNum(const user* u,const string& QQNum,const int len){
    for(int i=0;i<len;i++){
        if(u[i].getQQNum()==QQNum){
            this->addFriend(u[i].getWeiNum());
            cout<<"成功添加好友"<<u[i].getWeiNum()<<endl;
            break;
        }
```

```
    }
    if(i==len){
        cout<<"未找到好友"<<u[i].getWeiNum()<<endl;
    }
}
int user::findPhoneNum(const user* u,const string& PhoneNum,const int len,bool isAdd){
    for(int i=0;i<len;i++){
        if(u[i].getPhoneNum()==PhoneNum){
            if(isAdd)//确定是否添加，还是只是找到这个位置而已
            {
                this->addFriend(u[i].getWeiNum());
                cout<<"成功添加好友"<<u[i].getWeiNum()<<endl;
                break;
                return i;
            }
            else{
                return i;//返回电话在的位置
            }
        }
    }
    if(i==len&&isAdd){
        cout<<"未找到好友"<<u[i].getWeiNum()<<endl;
    }
    return -1;
}
void user::showTuiJianFriend(const user* u,const string* phoneNums,const int len){
    cout<<"推荐好友:"<<endl;
    for(int i=0;i<5;i++){
        int id=this->findPhoneNum(u,phoneNums[i],len,false);
        if(id!=-1)
            cout<<u[id].getWeiNum()<<endl;
    }
}

/*
最后再来设计群组吧，和用户类没有任何关系!!!
*/
#ifndef _GROUP_H_
#define _GROUP_H_
#include <iostream>
#include <string>
using namespace std;
class group{
```

```
public:
    //构造函数和析构函数初始化群名称、群主、群容量
    group(const string& GName,const string& CName,const int len=20){
        groupName=GName;
        createrName=CName;
        userNum=0;
        user=new string[size=len];
    }
    ~group(){
        delete[] user;
    }
    //成员函数:获取群名称、群主
    string getGroupName()const{
        return groupName;
    }
    string getCreaterName()const{
        return createrName;
    }
    //显示群成员
    void showNumbers()const{
        cout<<groupName<<" 群 共 有 成 员 "<<userNum<<" 名 , 其 中 群 主 是
"<<createrName<<endl;
        for(int i=0;i<userNum;i++){
            cout<<user[i]<<endl;
        }
    }
    //添加群成员
    void addNumbers(const string& name){
        user[userNum++]=name;
    }
private:
    string groupName;
    string createrName;
    string* user;
    int size;
    int userNum;
};
#endif


/*
Main 测试
*/
#include <iostream>
```

```
#include <string>
#include "user.h"
#include "group.h"
//using namespace std;
int main(){
    //创建 8 个用户对象，并初始化 8 个用户的基本信息
    user* u=new user[8];
    u[0].set("张三","13444","157","122@qq.con");
    u[1].set("李三","12444","138","122@qq.con");
    u[2].set("王三","12344","187","122@qq.con");
    u[3].set("韩三","12344","147","122@qq.con");
    u[4].set("赵三","45454","117","122@qq.con");
    u[5].set("冯三","123432321","133","122@qq.con");
    u[6].set("马三","12323132","134","122@qq.con");
    u[7].set("徐三","124343","139","122@qq.con");

    u[0].findWeiNum(u,"韩三",8);
    //按照 qq 号添加
    u[0].findQQNum(u,"13444",8);
    //按照电话添加
    u[0].findPhoneNum(u,"139",8,true);
    u[0].showFriend();
    //按照电话号码推荐好友
    string pNums[]={"138","147","133","222","333"};
    u[0].showTuiJianFriend(u,pNums,8);

    group g1("一家人",u[3].getWeiNum());
    g1.addNumbers(u[0].getWeiNum());
    g1.addNumbers(u[1].getWeiNum());
    g1.showNumbers();

    delete[] u;//释放 user 类型数组空间
    return 0;
}
```

2014 学硕手机通讯录

说明：私有数据成员的这个数组存的就是联系人的一些信息，如：姓名、电话、邮箱。

所以我们自己要创造个信类型 Record 来存放这些不同类型的信息

//而之前有个微信类的实现：数组只是存的微信好友的姓名，而没有存这么多信息，所以相对简单。

//只是用 string* friendName;来创建的数组

1.设计内部类 Record 来添加每个对象的信息（姓名、电话、email），也就是 setRecord 方法，

并且外部类能够通过成员函数间接访问私有数据成员，也就是内部类创建 get 成员函数。

成员函数 show()能够显示当前对象的信息

2.设计外部类：Contact:能够创建并储存多个 Record 对象，也就是有个私有数据成员的类型就是 Record 类型指针，

那么数组中每个元素就是一个 Record 对象，就都有自己的基本信息了，从而创建了通讯录。

1.含参构造函数：初始化通讯录容量，若用户不指定，默认为 255，申请这么大的内存给指针

2.析构函数：释放创建的空间内存

3.添加数据：也就是向 Record 类型数组添加数据，间接的去添加每个元素的姓名、电话、email

4.将通讯录存储到文件中去：也就是循环遍历数组每个元素（也就是每个联系人），依次输出到文件中去。

5.按照姓名查找：要考虑后续是否使用这个查找函数，显示在模糊查找还有合并通讯录的时候要用到，那么就要全面考虑。

首先：在我们模糊查找的时候不需要将查找到的内容显示到屏幕上去，我们只需要知道它找到还是没有找到

6.合并两个通讯录：先遍历一个通讯录看是否在另一个通讯录中有重名的联系人，没有那么就合并，有就去掉

7.模糊查找：用到 r[i].getName.find(str);.find 函数，找 r[i].getName 中是否存在 str 字符串，没有的话返回一个 npos 值

/*

内部类：联系人基本信息类 Record，外部类 Contact 创建私有的对象，实现信息隐藏原则

***/**

//防止该头文件被重复引用：含义如下：

#ifndef _RECORD_H_//如果不存在 record.h 文件

#define _RECORD_H_H//那么就引入文件

```
#include <iostream>
#include <string>
using namespace std;

class Record{
public:
    Record(){}
    ~Record(){}
    //2.设置数据成员值(添加新记录功能)：即创建 Record 对象时，调用 setRecord 函数用户来初始化联系人信息
    void setRecord(const string& Name,const string& Phone,const string& Email){
        name=Name;
        phone=Phone;
        email=Email;
    }
    //3.返回数据成员值
    string getName()const{return name;}
    string getPhone()const{return phone;}
    string getEmail()const{return email;}
    //4.显示联系人基本信息
    void show()const{
        cout<<name<<"    "<<phone<<"    "<<email<<endl;
    }
private:
    //1.确定数据成员
    string name;
    string phone;
    string email;
};
#endif//否则不引入文件 Record.h

/*
通讯录类：实现查看全部联系人功能
          按姓名查找
          按电话查找
          按 email 查找
          模糊查找：如客户输入'王'，列出所有王姓的联系人
          将全部联系人存储到文件上去，客户输入文件名称
          合并两个通讯录到一个文件中去
*/
#ifndef _CONTACT_H_
#define _CONTACT_H_

#include "Record.h"
```

#include <fstream>//要使用文件来存储数据，而其他头文件如:iostream,using namespace std,已经在 Record.h 中声明过了

```
class Contact{
public:
    //2.含参构造函数：默认通讯录容量为 255，当客户指定大小那么就用指定的，否则默认 255
    Contact(int s=255){
        r=new Record[s];//申请 size 大小的 Record 类型的数组
        num=0;//初始化没有任何联系人信息
    }
    //3.释放申请的空间
    ~Contact(){delete[] r;}

    //4.添加联系人到通讯录中
    void add(const string& name,const string& phone,const string& email){
        r[num++].setRecord(name,phone,email);//访问的是内部类的成员函数来设置联系人信息
    }
    //5.显示数组中所有的联系人的信息
    void allShow()const;
    //6.获取当前通讯录总人数
    int getNum()const{return num;}
    //7.将通讯录导入到文件中去
    void output(const string&);

    //8.分别按照姓名、电话、email 查找联系人,查找成功返回-1
    int findName(const string&,bool isshow=true)const;//isshow 表示按姓名查找后是否显示到屏幕上，
    //因为我们合并两个通讯录的时候也需要用到查找，但是我们却不需要显示了
    //如同我们在设计微信类的时候，我们需要根据电话号码来推荐联系人，也要用到查找电话，但是不需要存储到好友中，只是查看有没有而已

    //9.合并两个通讯录
    void meg(const Contact&);
    //根据下标位置获取联系人信息
    string getNameById(int id)const{
        return r[id].getName();//通过成员函数来间接访问内部类的私有数据成员
    }
    string getPhoneById(int id)const{
        return r[id].getPhone();
    }
```

```
    string getEmailById(int id)const{
        return r[id].getEmail();
    }
    //10.模糊查找
    void search(const string&)const;
private:
    //1.确定私有数据成员
    Record* r;//创建指向 Record 类类型的指针，那么在构造函数中就可以申请内存了，也就是有了一个数组空间了
    //这个数组存的就是联系人的一些信息，如：姓名、电话、邮箱。
    //而之前有个微信类的实现：数组只是存的微信好友的姓名，而没有存这么多信息，所以相对简单。
    int num;//当前存储的联系人人数
};
#endif

/*
辅助函数：实现 Contact 类中的方法的定义
*/
#include "Contact.h"

void Contact::allShow()const{
    for(int i=0;i<num;i++){
        r[i].show();
    }
}

void Contact::output(const string& fname){
    ofstream out(fname.c_str());
    for(int i=0;i<num;i++){
        out<<r[i].getName()<<"    "<<r[i].getPhone()<<"    "<<r[i].getEmail()<<endl;
    }
    out.close();
}

int Contact::findName(const string& name,bool isshow)const{
    for(int i=0,flag=1;i<num;i++){
        if(r[i].getName()==name){
            if(isshow){
                cout<<"信息如下："<<endl;
                r[i].show();
                flag=0;
            }
            return -1;
```

```
            }
        }
        if(flag&&isshow){
            cout<<"未找到"<<name<<endl;
        }
        return 0;
}
void Contact::meg(const Contact& c1){
        int f;
        for(int i=0;i<c1.getNum();i++){
            f=findName(c1.getNameById(i),false);//目的就是筛出掉同一个人(不考虑重名因素)
            if(!f){
                add(c1.getNameById(i),c1.getPhoneById(i),c1.getEmailById(i));//放当某一个通讯录中
            }
        }
        cout<<"同步成功"<<endl;
}

void Contact::search(const string& name)const{
        int flag=0;//标记是否找到
        for(int i=0;i<num;i++){
            int res=r[i].getName().find(name);//使用 find 函数，若在 r[i].getName 中未发现 name 字符串，就返回一个无穷大的数字
            if(res<r[i].getName().length()){//因此判断条件若找到则必然会小于其总长度
                flag=1;
                r[i].show();
            }
        }
        if(!flag){
            cout<<"未找到"<<endl;
        }
}
/*
主函数用来测试
*/
#include "Contact.h"
int main(){
        /*测试 Record 类创建成功
        Record r1;
        r1.setRecord("张三","1234","23444");
        r1.show();*/
```

```
    //向通讯录添加五条数据
    cout<<"通讯录 1:"<<endl;
    Contact c1;
    c1.add("韩明旭","1234","3334");
    c1.add("马航","2333","5555");
    c1.add("李四","2333","5555");
    c1.add("张三","2333","5555");
    c1.add("网二","2333","5555");
    c1.allShow();
    //将添加的数据存储到文件中去
    c1.output("list1.txt");

    cout<<"通讯录 2:"<<endl;
    Contact c2;
    c2.add("韩寒","1234","3334");
    c2.add("韩三","2333","5555");
    c2.add("李四","2333","5555");
    c2.add("张八","2333","5555");
    c2.add("李二","2333","5555");
    c2.allShow();
    c2.output("list2.txt");

    cout<<"******按姓名查找******"<<endl;
    //分别按照姓名、电话、email 查找联系人
    c2.findName("韩寒");

    //合并通讯录 1 和 2，将合并的数据放到文件中去
    c1.meg(c2);
    cout<<"******通讯录合并后******"<<endl;
    c1.allShow();
    c1.output("listAll.txt");

    //使用模糊查找所有姓韩的
    cout<<"*******使用模糊查找******"<<endl;
    c1.search("韩");
    return 0;

}
```

2015 专硕

1. 均值方差

2.ip 地址划分，判断是否合法，构造函数中参数为 ip 地址字符串

如 192.168.15.1 是属于 C 类 ip 地址

关键在于:
1.怎么把字符串分割成四小段:使用字符串中.find 函数找到"."的位置，
使用.substr()提取"."之前的字符串并将其转换成整型，存储到数组中
2.怎么将这四小段字符串转换成整型：使用模板函数，函数中使用 istringstream 流，绑定字符串，然后将其输出到整型的变量中去

3.然后分别把这四小段整型元素判断条件就可以了：也就是数组中元素是否都在 0-255 之间。
若合法那么数组中元素依次判别，确定 ip 的类型：A,B,C,D,E

注意：在构造函数中传入的参数要设置成 const string& ip,而 不是 string& ip，不知道为什么不设置不对

```
#include <iostream>
#include <string>
#include <sstream>
using namespace std;

//4.模板函数用来将字符串类型转换成整型，返回值就是转换的类型值
template <class T>
T fromString(const string& str){
    //使用 istringstream 来绑定当前的字符串 str，导入头文件 sstream
    istringstream in(str);
    T temp;//需要转换成的临时变量
    in>>temp;//将这个字符串输入到 T 类型的变量中，完成转换
    return temp;
}

//创建 Ip 对象
class Ip{
public:
    //2.构造函数初始化数据成员 a[4]，也就是我们要进行分割字符串了
    Ip(const string& ip){
```

```
        getIntIp(ip);//将分割的实现细节隐藏起来，遵循信息隐藏原则
    }
    //3.测试函数:是否完成分割，也就是整型数组中是否有数字了
    void show()const{
        for(int i=0;i<4;i++)
        {
            cout<<a[i]<<endl;
        }
    }

    //4.判断 ip 地址是否合法：也就是数组中每一位数字是否在 0-255 之间,返回值为 bool，若不合法也就不用去判断使哪种类型 ip 了
    inline bool isLegal()const;
    //5.查看是哪种类型的 ip 地址
    inline void type()const;
private:
    //1.确定数据成员
    int a[4];//存储分割后的数字(字符串要转换成整型)

    //3.遵循信息隐藏原则，隐藏转换的实现细节
    void getIntIp(const string& ip);
};
void Ip::type()const{
        if(isLegal)
        {
            if(a[0]>=1 && a[0]<=127){//只要修改访问或添加新的限制条件就能改范围
                cout<<"A 类地址！"<<endl;
            }
            else if(a[0]>=128 && a[0]<=191){
                cout<<"B 类地址！"<<endl;
            }
            else if(a[0]>=192 && a[0]<=223){
                cout<<"C 类地址！"<<endl;
            }
            else if(a[0]>=224&&a[0]<=239){
                cout<<"D 类地址！"<<endl;
            }else if(a[0]>=240&&a[0<=255]){
                cout<<"E 类地址！"<<endl;
            }
        }
    }
bool Ip::isLegal()const{
        bool is=true;//默认情况下是合法的
```

```
        //循环遍历，依次看每个元素是否位于 0-255 之间，若不是那么该 ip 就不合法
        for(int i=0;i<4;i++)
        {
            if(a[i]<0||a[i]>255)
            {
                is=false;
                break;
            }
        }
        if(!is){
            cout<<"该 ip 地址不合法"<<endl;
            return is;
        }
        cout<<"该 ip 地址合法"<<endl;
        return is;
    }
void Ip::getIntIp(const string& ip){
    string str=ip;//存储分割下来的字符串，当前为原字符串
    for(int i=0;i<4;i++)
    {
        //若是最后一个字符串，就不用分割了，已经就是 str 了
        if(i==3)
        {
            a[i]=fromString<int>(str);//将字符串类型转换成整型，使用模板函数 fromString
        }
        //若不是最后一个字符串那么就需要去提取字符串了，也就是提取"."之前的字符串
        //首先找到当前 str 的第一个"."位置，然后把"."之前的字符串提取出来
        int location=str.find(".");
        //提取"."之前的字符串，然后转换成整型
        a[i]=fromString<int>(str.substr(0,location));
        //剔除调用已经转换的字符串
        str=str.substr(location+1,str.length()-location-1);
    }
}

int main(){
    Ip ip("192.168.12.1");//创建 Ip 对象，并初始化数据成员
    //测试字符串是否分割成功
    ip.show();

    ip.isLegal();
```

```
    ip.type();
    return 0;
}
```

3.赫夫曼编码:符号出现频率高的用较短的编码，反之用长的。目的就是提高通信传输的效率，压缩文件大小，使编码之后的字符串长度降低，期望值降低，达到无损压缩数据

BABACACADADABBCBABEBEDDABEEEBB

1 首先计算出每个字符出现的次数（概率）：B:10 A:8 C:3 D:4 E:5
2 把出现次数（概率）最小的两个相加，并作为左右子树，重复此过程，直到概率值为 1
第一次：将概率最低值 3 和 4 相加，组合成 7：

第二次：将最低值 5 和 7 相加，组合成 12：

第三次：将 8 和 10 相加，组合成 18：

第四次：将最低值 12 和 18 相加，结束组合：

3 将每个二叉树的左边指定为 0，右边指定为 1

4 沿二叉树顶部到每个字符路径，获得每个符号的编码

我们可以看到出现次数（概率）越多的会越在上层，编码也越短，出现频率越少的就越在下层，编码也越长。
当我们编码的时候，我们是按“bit”来编码的，解码也是通过 bit 来完成，
如果我们有这样的 bitset “10111101100″ 那么其解码后就是 “ABBDE”。
所以，我们需要通过这个二叉树建立我们 Huffman 编码和解码的字典表。

设计：
（1）内部类 HuffNode，储存的是结点的一些属性：例如权值、左右孩子是谁(前提要有)、结点代表的字符、双亲是谁
（2）类 HuffTree：私有数据成员：创建大小为 2*128 的内部类 HuffNode 的对象数组，那么每个元素就是一个对象，也就是一个结点了，拥有内部类的属性
整型数组 count 来记录客户输入字符串中每个字符出现的频率，而这个频率就是权值。ASCII 码对应的码值就是元素下标的值
客户输入的字符串 str、记录一共多少个叶子结点的 m

（3）默认构造函数：初始化结点数组以及整型数组 count，默认结点 m=128 个

（4）输入函数 input()：客户输入要编码的字符串，顺便统计每个字符出现的次数，也就是权值，存放在 count 数组中

（5）创建赫夫曼树函数 createTree：首先创建 i 个叶子结点，也就是 m 的值，数组 count 的值就是权值，下标就是每个结点代表的字符

所有叶子结点创建完毕，就开始创建结点，也就是逐步生成二叉树，遍历结点数组，找权值最小的两个结点组成一个新的结点。

（6）找权值最小的两个结点在数组中的位置 select()：思想就是打擂台算法，

先放两个最大的变量，依次遍历结点数组的权值，进行比对。前提是此结点没有双亲，也就是 parent=0.否则是已经生成过的了

（7）遍历二叉树了，输出对应的二进制编码：用二维指针 char 来存储每个字符的编码。刚开始申请 m 个指针，然后每个指针指向的列根据编码的长度来申请内存

从二叉树的头开始遍历，也就是结点数组的最后一个元素就是头!!，p=2*m-1

if 判断是遍历左边还是右边，若遍历了左边，用权值 weight 来=1 表示已经遍历过了，下次不用再遍历了。

（注：weight 权值只是在创建最优二叉树的时候要根据权值来创建使用，此时已经不再使用了）

若遍历的左边，那么对于的编码就是 0，将它存储在临时的 char 型数组中，

等遍历到叶子结点，也就是末尾了，也就是结点已经没有左右孩子了，再把这个临时的编码数组赋值给二维的那个指针数组

最后，也就是 p=0 的时候，循环也就结束了，将二维数组的编码值依次输出出来，也就是每个结点对应的编码

```
#ifndef _HUFFNODE_H_
#define _HUFFNODE_H_
//1.创建结点类:属性有权值、左右孩子、代表的字符(注意只有叶子结点才会有代表的字符)
class HuffNode{
public:
    int weight;//权值
    int parent;//双亲
    int lchild;//左孩子
    int rchild;//右孩子
    char c;//结点代表的字符
};
#endif

/*
```

创建赫夫曼树
*/
```
#ifndef _HUFFTREE_H_
#define _HUFFTREE_H_
#include <iostream>
#include <string>
#include "huffNode.h"
#define MAX 128
using namespace std;

class HuffTree{
public:
    HuffTree()
    {
        int m=128;//默认叶子结点为 128 个
        //初始化结点数组全部属性
        for(int i=1;i<=(2*MAX-1);i++)
        {
            huff[i].weight=0;
            huff[i].lchild=0;
            huff[i].rchild=0;
            huff[i].parent=0;
        }
        for(int j=0;j<MAX;j++)
        {
            count[j]=0;
        }
    }
    ~HuffTree(){}

    void input();
    void creatTree();
    void LeafCode();

private:
    //创建结点数组：存储每一种字符对应的结点，从下标 1 的位置开始，固不是 2*MAX-1
    HuffNode huff[2*MAX];
    int count[MAX];
    int m;//实际叶子结点数
    string str;//输入的字符串

    //找最小权值的过程可以隐藏起来
    void select(int,int&,int&);
```

```
};


//采用非递归方法遍历最优二叉树，求 n 个叶子的编码
void HuffTree::LeafCode()
{
    //HC 就好比如是个二维的指针数组，每一维对应一个字符的编码
    char * * Hc;
    //m 是实际的叶子结点数，但是实际的结点数是 2*m-1,因为还有很多包含没有代表字符的结点，也就是桥梁的结点存在
    //虽然我们申请了 2*MAX 的结点数组，但是显然到目前为止我们只用到了 2*m-1 个的结点
     int i, p = 2*m - 1, cdlen = 0;//p 就是实际的结点数组的大小
     char code[30];//每个结点对应的二进制码

     //Hc =(char**)malloc((m+ 1) * sizeof(char *));//确定申请多少维的内存，也就是知道了有多少个叶子结点
    Hc=new char*[m+1];
    //但是每一维对应的多少列是不确定的，因为编码的长度是不同的，所以要随机申请

    for(i = 1; i <= p; i++)
     {
        //权值已经不再使用了，权值只是在创建最优二叉树的时候使用
         huff[i].weight = 0;//遍历二叉树时用作结点的状态标志
     }

     while(p)//挨个结点遍历最优二叉树，求 n 个叶子的编码
     {
         if(0 == huff[p].weight)//weight=0 表明左边没有被遍历
         {
             huff[p].weight = 1;//标记已经开始遍历左边

             if(huff[p].lchild != 0)//该结点有左孩子
             {
                 p =huff[p].lchild;   //存在左孩子，
               //关键！：此时 p 的位置就是左孩子的位置，那么下一次循环 p 从当前孩子继续向下遍历，直到叶子结点末尾
                 code[cdlen++] = '0';
             }
            //这一步是直到遍历到最后叶子结点才会做的！就是把编码的二进制输出出来
             else if(0 ==huff[p].rchild)//叶子结点也没有右孩子，已走到叶子结点,
```

记下该叶子结点的字符的编码

```
                {
                  //申请当前维的内存，也就是用了多少列
                    Hc[p] = new char[cdlen+1];
                    code[cdlen] = '\0';
                    strcpy(Hc[p], code);
                }
            }
            else if(1 ==huff[p].weight)//向右走:此时说明左孩子已经遍历完毕，开始右孩子
            {
                huff[p].weight = 2;
                if(huff[p].rchild != 0)//该结点有右孩子
                {
                    p = huff[p].rchild;//若该右孩子还有左右孩子，那么就是 p 不会是 0，循环继续走下去
                    code[cdlen++] = '1';
                }
            }
            else//2 == Ht[p].weight, 回退
            {
                huff[p].weight = 0;
                p =huff[p].parent;
                cdlen--;
            }
        }//end of while


        //输出每个叶子结点对应的编码：
        for(i = 1; i <=m; i++)
        {
            cout<<"字符"<<huff[i].c<<"的编码为"<<Hc[i]<<endl;
        }

        //释放内存空间
        /*for(i=0;i<m+1;i++)
        {
            delete[] Hc[i];
        }*/
        delete[] Hc;
}

void HuffTree::select(int n,int& left,int& right)
{//n 就是当前结点数组的结点数，left 为最小的结点的下标位置，right 为第二小
```

```
结点的下标位置
    //打擂台算法
    int min1;
    int min2;
    min1=min2=32767;
    left=right=1;
    for(int i=1;i<=n;i++)
    {
        if(huff[i].parent==0&&huff[i].weight<min1)
        {
            min2=min1;
            right=left;
            min1=huff[i].weight;
            left=i;
        }else if(huff[i].parent==0&&huff[i].weight<min2)
        {
            min2=huff[i].weight;
            right=i;
        }
    }
//方法二
 /*int x1=32767,x2=32767;//打擂台的方法，将结点数组的所有权值依次对比，留下两个最小的权值
    left=1;right=1;//将下标位置初始化为 0
    for(int i=1;i<=n;i++)
    {
        if(huff[i].parent==0&&huff[i].weight<x1)//
        {
            x2=x1;//我们约定的是 x1 存储最小的权值，而此时又来了一个更小的，那么 x1 自然是第二小的权值了
            right=left;//那么第二小孩子的位置自然是在右边了，而不是左边

            x1=huff[i].weight;
            left=i;//最小的下标位置
        }else if(huff[i].parent==0&&huff[i].weight<x2){
            x2=huff[i].weight;
            right=i;
        }
    }*/
}
void HuffTree::creatTree()
{
    int i,k,left=0,right=0;
    //先分配输入的字符所对应的结点，从结点数组 1 开始到一共输入多少字符
```

结束，以及他的权值
//我们从结点数组 1 的位置开始存储对应的字符，而不是 0 的位置。
//所以创建的结点数组会浪费一块很大的内存，因为只有存储了全部的 128 个字符的时候才会用到这么大数组
```
for(i=1,k=0;k<MAX;k++)
{
    if(count[k]!=0)
    {
        huff[i].c=(char)k;//结点所代表的字符，而 k 为下标，就是 ASCII 码值，那么转换成对应的字符。如 65-->A
        huff[i].weight=count[k];//分配对应的权值,也就是字符串中一共出现了几次
        i++;//此时结点数组中存储一个实际的有属性的结点，
        //比如我们输入了 AAADDDSSS 字符串，那么其实结点数组目前只用了 3 个元素，也就是目前只有三个结点
    }
}
//m 就是用来储存一共有多少个叶子结点，也就是带字符含义的结点的个数
m=i-1;//最后 i++了，所以实际的结点数只有 i-1 个

//此时我们就开始形成新的结点了（两个权值最小的结点，形成一个新的结点），
//注意！不是叶子结点！，那么就需要从结点数组的没有存储结点的位置开始存储没有代表字符的结点了
//一共 m 个叶子结点，就会对应 2*m-1 个结点，注意是包含那 m 个叶子结点的，所以我们要从 m+1 的位置开始创建新的结点
for(k=m+1;k<=(2*m-1);k++)
{
    //从结点数组中 1 至 k-1 位置，寻找两个权值最小的结点，来当做当前结点的左右孩子
    //left，right 是对应的下标位置
    select(k-1,left,right);
    //左右两个孩子的双亲皆是刚生成的父结点 k
    huff[left].parent=k;
    huff[right].parent=k;

    //父结点的左右两个孩子分别就是获取到的权值最小的两个叶子结点
    huff[k].lchild=left;
    huff[k].rchild=right;

    huff[k].weight=huff[left].weight+huff[right].weight;
}

}
```

```
void HuffTree::input()
{
    cout<<"输入编码的字符串："<<endl;
    cin>>str;

    //统计输入字符串中每个字符出现的次数
    for(int i=0;i<str.length();i++)
    {
        count[str[i]]++;//str[i]是对应的字符，但是会自动转换成对应的十进制数值，
        //如 A---对应 65，那么就是 count[65]++
    }
}
#endif
/*
主函数测试
*/
#include "hufftree.h"
int main()
{
    //BABACACADADABBCBABEBEDDABEEEBB
    HuffTree huff;
    huff.input();//输入要编码的字符串

    huff.creatTree();//根据字符串来创建赫夫曼树

    huff.LeafCode();//对字符进行编码，并输出对应的二进制码

    return 0;
}
```

4.微信类

2016 专硕

1. 均值、最大值
2. FTP 提取用户名、密码、ip、端口号。同 2014 专硕第三题
3. 矩阵加减乘转置：同 2017 专硕第五题
4. 配置文件信息。同 2013 学硕

2017 专硕

1. 均值方差
2./*

2.求数组中后一个与前一个相差为 1 的数对，打印出来

例：输入：2674831
　　输出：（6,7）　（7,8）
*/

```
#include <iostream>
using namespace std;
int main(){
	int n,i;
	int* a;
	cout<<"准备输入几个数:"<<endl;
	cin>>n;
	a=new int[n];
	cout<<"输入这几个数:"<<endl;
	for(i=0;i<n;i++)
	{
		cin>>a[i];
	}
	cout<<"你输入的数字是:"<<endl;
	for(i=0;i<n;i++)
	{
		cout<<a[i]<<"    ";
	}
	cout<<endl;

	int first,second;
	cout<<endl<<"其中前后相差为 1 的数对有:"<<endl;
	for(i=0;i<n-1;i++)
	{
		if(a[i]+1==a[i+1]||a[i]==a[i+1]+1)
```

```
        {
            cout<<"("<<a[i]<<","<<a[i+1]<<")"<<"    ";
        }
    }
    return 0;
}
```

3.统计连续字符个数：同 2014 专硕第二题

4.ip 地址划分并判断类型。同同 2015 专硕第二题

5.1.设计矩阵类:私有数据成员：row，col，int** a(双重指针)

2.重载构造函数中初始化双重指针（申请内存空间，也就是形成了二维数组，并全部赋值为 0）

```
a=new int*[row];//先申请每一行
for(i=0;i<row;i++)
    a[i]=new int[col];再为每一行申请 col 个元素
```

3.拷贝构造函数：与构造类似，也是申请二维数组空间，但是传入的参数是(const juzhen& j2),拷贝的格式就是这样的

4.析构函数:释放内存空间，一定要注意步骤，先释放行:每一行的元素，再释放每一行，也就是与申请的时候相反

5.=，赋值运算符的重载：为什么要用到这个呢？（不用的话就是：juzhen j3(j1+j2);调用的就是拷贝构造函数）

就是在如 j3=j1+j2;的时候我们先做的+运算符的重载，结果返回的矩阵要赋值给 j3，所以我们要首先将 j3 的内存释放掉，

重新申请内存，就是为了防止 j3 的行列与 j1，j2 不同。而函数中的 j3 就是*this 这个引用

然后就是简单的将 j1，j2 对应的每行没列元素加给 j3 就行了，最后返回 j3 就行了，注意返回的是引用，也就是*this

6.+，-运算符的重载：首先建立一个新的矩阵对象用来储存运算出来的矩阵，然后判断相加矩阵是否符合行列相同的原则，

否则是无法进行加法运算的。若符合，那么就是两个 for 循环，完成相加减

7.*运算符的重载：乘法的规则：行与列相同才可以相乘，如 2*3 矩阵可以与 3*5 矩阵=2*5 矩阵，否则是无法相乘的。

符合条件，三个 for 循环：第一层 for 是第一个矩阵的行，第二层 for 是第二个矩阵的列，第三层 for 是第一个矩阵的列，

注意：结果是第一个矩阵的某一行*第二个矩阵的某一列的和，所以在第二层矩阵里面加一个 sum，最后将 sum 依次赋值给第三个矩阵对应的位置

8.转置矩阵的函数：也就是例如原来的 2*3 变为 3*2，那么首先是将当前的矩阵的元素赋值给一个临时的 3*2 矩阵，然后将原矩阵申请

的空间释放，重新申请 3*2 的二维数组而不是 2*3，最后直接 a=temp 就可以了

```
#include <iostream>
using namespace std;
class juzhen{
public:
    //构造函数初始化二维数组
    juzhen(int R,int C){
        row=R;
        col=C;
        a=new int*[row];//使用二维数组
        int i,j;
        for(i=0;i<row;i++){
            a[i]=new int[col];
        }
        for(i=0;i<row;i++)
        {
            for(j=0;j<col;j++)
            {
                a[i][j]=0;//初始化二维数组
            }
        }
    }
    //拷贝构造函数:一个类包含指向动态存储空间的数据成员
    juzhen(const juzhen& j2){
        int i,j;
        row=j2.row;
        col=j2.col;
        a=new int*[row];
        for(i=0;i<row;i++)
        {
            a[i]=new int[col];
        }
        for(i=0;i<row;i++){
            for(j=0;j<col;j++){
                a[i][j]=j2.a[i][j];
            }
        }
    }
    //析构函数,释放内存空间
    ~juzhen(){
        for(int i=0;i<row;i++){
            delete[] a[i];//释放每一列元素
        }
        delete[] a;//释放每一行
    }
```

```
void input(){
    int i,j;
    cout<<"输入矩阵每一位值"<<endl;
    for(i=0;i<row;i++)
    {
        cout<<"第"<<i+1<<"行"<<endl;
        for(j=0;j<col;j++){
            cin>>a[i][j];
        }
    }
}
//输出矩阵值函数
void output()const{
    cout<<"当前矩阵为:"<<endl;
    for(int i=0;i<row;i++)
    {
        for(int j=0;j<col;j++)
        {
            cout<<a[i][j]<<"    ";
        }
        cout<<endl;
    }
}
//=、+、-、*操作符的重载
//=用于将相加减乘的矩阵赋值给新的矩阵，传入的参数就是如 j1+j2 做+重载的时候返回的矩阵
juzhen& operator=(const juzhen& j2){
    int i,j;
    //防止进行自身赋值
    if(this==&j2){
        return *this;
    }
    //我感觉没有必要释放原来的存储空间吧
    //答：要释放!!!，因为在如 j3=j1+j2;的时候，若 j3 与 j1，j2 不同行列，是无法完成赋值的
    //释放原矩阵空间
    for(i=0;i<row;i++){
        delete[] a[i];
    }
    delete[] a;
    row=j2.row;
    col=j2.col;
    //申请新的空间
    a=new int*[row];//先申请行
```

```
    for(i=0;i<row;i++){
        a[i]=new int[col];//每行再申请列
    }
    //赋值
    for(i=0;i<row;i++)
    {
        for(j=0;j<col;j++)
        {
            a[i][j]=j2.a[i][j];
        }
    }
    return *this;
}
//+运算符的重载:注意返回的值直接是 juzhen 类型的对象而没有引用
juzhen operator+(const juzhen& j2)const{
    juzhen j3(row,col);
    if(row!=j2.row||col!=j2.col){
        cout<<"不同行或不同列，无法相加"<<endl;
    }else{
        for(int i=0;i<row;i++)
        {
            for(int j=0;j<col;j++)
            {
                j3.a[i][j]=a[i][j]+j2.a[i][j];
            }
        }
    }
    return j3;
}
//-运算符的重载
juzhen operator-(const juzhen& j2)const{
    juzhen j3(row,col);
    if(row!=j2.row||col!=j2.col){
        cout<<"不同行列，无法相减"<<endl;
    }else{
        for(int i=0;i<row;i++)
        {
            for(int j=0;j<col;j++)
            {
                j3.a[i][j]=a[i][j]-j2.a[i][j];
            }
        }
    }
    return j3;
```

```
}
juzhen operator*(const juzhen& j2)const{
    juzhen j3(row,j2.col);
    if(col!=j2.row){
        cout<<"第一个矩阵的列与第二个矩阵的行不等，无法相乘"<<endl;
    }else{
        for(int i=0;i<row;i++)
        {
            for(int j=0;j<j2.col;j++)
            {
                int sum=0;
                for(int k=0;k<col;k++)
                {
                    sum+=a[i][k]*j2.a[k][j];
                }
                j3.a[i][j]=sum;

            }
        }
    }
    return j3;
}
//矩阵的转置
void cover()
{
    int i,j;
    juzhen temp(col,row);
    for(i=0;i<row;i++)
    {
        for(j=0;j<col;j++)
        {
            temp.a[j][i]=this->a[i][j];
        }
    }
*this=temp;//充分利用=赋值运算符，
//因为在=赋值中就已经重新申请内存空间了
}
//方法二：不是很好
/*void cover(){
    int tempRow=col;
    int tempCol=row;
    int i,j;
    //当前矩阵的元素赋值给临时转置矩阵
    juzhen temp(col,row);
```

```
        for(i=0;i<row;i++)
        {
            for(j=0;j<col;j++)
            {
                temp.a[j][i]=a[i][j];
            }
        }
        //删除当前矩阵的空间
        for(i=0;i<row;i++)
        {
            delete[] a[i];
        }
        delete[] a;
        //申请新的转置之后的空间
        a=new int*[tempRow];
        for(i=0;i<tempRow;i++)
        {
            a[i]=new int[tempCol];
        }
        //赋值过去就行了，因为已经写了=运算符的重载了
        *this=temp;
    } */
private:
    int row;
    int col;
    int** a;//使用指针的指针，也可以使用一维的指针
};
```

/*实现矩阵的加、减、乘:

***/**

```
#include <iostream>
#include "juzhen.h"
using namespace std;
int main(){
    juzhen j1(2,3);
    juzhen j2(2,3);
    j1.input();
    j2.input();
    j1.output();
    j2.output();

    cout<<"测试矩阵的加法运算(利用拷贝构造函数):"<<endl;//j3 还没有被实例化
```

```
    juzhen j3(j1+j2);
    j3.output();
    cout<<"测试矩阵的加法运算(利用=运算符的重载函数):"<<endl;
    juzhen j4(3,3);//即使 j4 为 3*3，但是我们在=运算符重载的时候，我们已经将
j4 的原空间释放了，申请了 2*3 的了
    j4=j1+j2;
    j4.output();

    cout<<"测试矩阵的减法运算(利用=运算符的重载函数):"<<endl;
    juzhen j5(3,4);
    j5=j1-j2;
    j5.output();

    cout<<"测试矩阵的乘法运算(利用=运算符的重载函数):"<<endl;
    juzhen j6(3,3);
    j6=j1*j2;//2*3-2*3 是无法乘的
    juzhen j7(3,2);//所以我们创建一个 3*2 的矩阵
    j7.input();
    j7.output();
    j6=j1*j7;
    j6.output();

    cout<<"测试矩阵的转置函数:"<<endl;
    j1.cover();
    j1.output();
    return 0;
}
```

2018 专硕上机

1. 任意输入 N 个正整数，找到其中出现频率最高的数，以及他的次数，如果有两个相同频率的数，则去最小的那个数 10 分

```
#include <iostream>
#define N 10
using namespace std;
int main()
{
    //b 数组记录出现的数字以及数字出现的次数
    int a[N];
    int b[10];
    int i;
    for(i=0;i<10;i++)
    {
      b[i]=0;
    }
    //用户输入 N 个数，记录数字次数到 b 数组中
    for(i=0;i<N;i++)
    {
      cin>>a[i];
      b[a[i]]++;
    }
    //找出出现次数最多的数字以及次数
    int max1=0,max2=0,flag1=0,flag2=0;
    for(i=0;i<10;i++)
    {
      if(b[i]!=0)
      {
            if(max1<b[i])
            {
                  max2=max1;
                  flag2=flag1;
                  max1=b[i];
                  flag1=i;
            }else if(max2<b[i])
            {
                  max2=b[i];
                  flag2=i;
            }
      }
    }
    //输出数字及出现的次数
    //若两个数字出现次数相同那么输出最小的那个
```

```
    if(max1==max2)
    {
      if(flag1>flag2)
      {
            cout<<flag2<<"::"<<max2<<endl;
      }else{
            cout<<flag1<<"::"<<max1<<endl;
      }
    }else if(max1>max2)
    {
      cout<<flag1<<"::"<<max1<<endl;
    }else{
      cout<<flag2<<"::"<<max2<<endl;
    }
    return 0;
}
```

2. 任意输入 N 个不为 0 的数，求其相反数的对数。

```
#include <iostream>
#define N 6
using namespace std;
int main()
{
    int a[N];
    int count=0,i,j;
    //用户输入数字
    for(i=0;i<N;i++)
    {
      cin>>a[i];
    }
    //开始从第一个向后找是否是相反数的数字
    for(i=0;i<N-1;i++)
    {
      //每次从其下一个开始找
      for(j=i+1;j<N;j++)
      {
            if(a[i]==-a[j])
            {
                  count++;
                  break;
            }
      }
    }
    //输出相反数的个数
    cout<<count<<endl;
```

```
    return 0;
}
```

3. ftp 原题改为 ftp://ouc:test@116.119.0.1:2121

4. 矩阵题 自定义输入输出，重载*运算符 以及实现 4*3 矩阵和 3*4 矩阵的乘法（要求矩阵各数随机输入）

```
#include <iostream>
using namespace std;
class Juzhen{
public:
    //构造函数初始化矩阵行列
    Juzhen(const int R,const int C):row(R),col(C)
    {
      int i,j;
      a=new int*[row];
      for(i=0;i<row;i++)
      {
            a[i]=new int[col];
      }
      for(i=0;i<row;i++)
      {
            for(j=0;j<col;j++)
            {
                  a[i][j]=0;
            }
      }
    }
    //拷贝构造函数
    Juzhen(const Juzhen& j1)
    {
      row=j1.row;
      col=j1.col;
      int i,j;
      a=new int*[row];
      for(i=0;i<row;i++)
      {
            a[i]=new int[col];
      }
      for(i=0;i<row;i++)
      {
            for(j=0;j<col;j++)
            {
                  a[i][j]=j1.a[i][j];
```

```
            }
        }
    }
    //析构函数释放内存空间
    ~Juzhen()
    {
        for(int i=0;i<row;i++)
        {
            delete[] a[i];
        }
        delete[] a;
    }
    //随机生成矩阵值
    void input();
    //显示当前矩阵值
    void show()const;
    //重载=操作符
    Juzhen& operator=(const Juzhen&);
    //重载*操作符
    Juzhen operator*(const Juzhen&);
private:
    int** a;
    int row,col;
};
Juzhen Juzhen::operator *(const Juzhen& j1)
{
    Juzhen j2(row,j1.col);
    if(col==j1.row)
    {
        for(int i=0;i<row;i++)
        {
            for(int j=0;j<j1.col;j++)
            {
                int sum=0;
                for(int k=0;k<col;k++)
                {
                    sum+=a[i][k]*j1.a[k][j];
                }
                j2.a[i][j]=sum;
            }
        }
    }

    return j2;
```

```
}
Juzhen& Juzhen::operator =(const Juzhen& j1)
{
    if(this==&j1)
    {
      return *this;
    }
    int i,j;
    for(i=0;i<row;i++)
    {
      delete[] a[i];
    }
    delete[] a;
    row=j1.row;
    col=j1.col;
    a=new int*[row];
    for(i=0;i<row;i++)
    {
      a[i]=new int[col];
    }

    for(i=0;i<row;i++)
    {
      for(j=0;j<col;j++)
      {
            a[i][j]=j1.a[i][j];
      }
    }
    return *this;
}
void Juzhen::show()const
{
    cout<<"当前矩阵为：  "<<endl;
    for(int i=0;i<row;i++)
    {
      for(int j=0;j<col;j++)
      {
            cout<<a[i][j]<<" ";
      }
      cout<<endl;
    }
}
void Juzhen::input()
{
```

```
    cout<<"输入矩阵值：  "<<endl;
    for(int i=0;i<row;i++)
    {
      for(int j=0;j<col;j++)
      {
            cin>>a[i][j];
      }
    }
}

int main()
{
    //分别创建矩阵 A,B
    Juzhen A(3,4);
    A.input();
    Juzhen B(4,3);
    B.input();
    Juzhen C(3,3);
    //矩阵相乘
    C=A*B;
    C.show();
    return 0;
}
```

5.15 年微信原题