1. **输入三个数求其均值、方差、标准差。也就是 2017的第一题**

```cpp
#include <iostream>
#include <cmath> // 使用 pow 函数以及 sqrt 函数
using namespace std;

int main() {
    float average;  // 用于存储均值
    float fangcha;  // 用于存储方差
    float biaozhuncha;  // 用于存储标准差
    float sum = 0.0;  // 用于求和
    int i, n = 3;  // n 表示输入数字的个数，这里固定为 3
    int* a = new int[3];  // 动态分配一个大小为 3 的整型数组，用于存储输入的三个数

    cout << "输入三个数:" << endl;
    for (i = 0; i < n; i++) {
        cin >> a[i];  // 从用户输入读取一个数并存储到数组 a 中
        sum += a[i];  // 将输入的数累加到 sum 中
    }

    average = sum / n;  // 计算均值，均值等于所有数的总和除以数字的个数
    sum = 0.0;  // 重置 sum 为 0，以便后续计算方差

    // 循环遍历计算方差
    for (i = 0; i < n; i++) {
        sum += pow(average - a[i], 2);  // 计算每个数与均值的差的平方，并累加到 sum 中
    }
    fangcha = sum / n;  // 方差等于每个数与均值的差的平方的总和除以数字的个数

    // 计算标准差
    biaozhuncha = sqrt(fangcha);  // 标准差是方差的平方根

    cout << "均值:" << average << endl
         << "方差:" << fangcha << endl
         << "标准差:" << biaozhuncha << endl;

    delete[] a;  // 释放动态分配的数组内存，避免内存泄漏
    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**

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

// 将字符串转换为大写的函数
void toUpperCase(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
    toUpperCase(str);
    cout << "全部转换成大写之后: " << str << endl;

    int len = 1;
    for (int i = 1; i <= str.length(); i++) {
        if (i == str.length() || str[i] != str[i - 1]) {
            // 输出连续字符及其个数
            cout << str[i - 1] << ":" << len;
            len = 1;
            if (i != str.length()) {
                cout << ",";
            }
        } else {
            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>

// 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(const std::string& str) {
    T temp;
    std::istringstream in(str);  // 绑定 str
    in >> temp;  // 输出到 temp 中，完成转换
    return temp;
}

class Website {
```

```
public:
    // 含参构造函数初始化数据成员
    Website(const std::string& str) {
        // 将字符串分割的细节隐藏起来，调用 stringCut 函数
        stringCut(web = str);
    }

    // 获取用户名、密码、ip、端口号、网址的成员函数
    std::string getWeb() const { return web; }
    std::string getUserName() const { return userName; }
    std::string getPassword() const { return password; }
    std::string getIp() const { return ip; }
    int getNum() const { return num; }

private:
    // 确定数据成员
    std::string web;
    std::string userName;
    std::string password;
    std::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(const std::string& str);
};

// 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(const std::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
    std::string str1 = str.substr(0, temp);
    // 右边 str2：192.168.1.1:2121
    std::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:
    std::cout << "网址为: " << w1.getWeb() << std::endl;
    std::cout << "用户名: " << w1.getUserName() << std::endl;
    std::cout << "密码: " << w1.getPassword() << std::endl;
    std::cout << "ip 地址: " << w1.getIp() << std::endl;
    std::cout << "端口号: " << w1.getNum() << std::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

```
#ifndef _USER_H_
#define _USER_H_

#include <iostream>
#include <string>

class User {
public:
    // 构造函数初始化好友数组，指定大小为 30
    User() {
        fri = new std::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 std::string&, const std::string&, const std::string&, const
std::string&);

    // 获取个人基本信息
    std::string getWeiNum() const { return weiNum; }
    std::string getQQNum() const { return qqNum; }
    std::string getPhoneNum() const { return phoneNum; }
```

```
    std::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 std::string&, const int);
    void findQQNum(const User*, const std::string&, const int);
    int findPhoneNum(const User*, const std::string&, const int, bool);

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

private:
    std::string weiNum;
    std::string qqNum;
    std::string phoneNum;
    std::string email;
    std::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 std::string& infor) {
        fri[friNum++] = infor;
    }
};

#endif

#include "User.h"
#include <iostream>
#include <string>

void User::set(const std::string& Wei, const std::string& QQ, const std::string& Phone,
const std::string& Email) {
    weiNum = Wei;
    qqNum = QQ;
    phoneNum = Phone;
    email = Email;
}

void User::showFriend() const {
    std::cout << "您的好友有:" << std::endl;
    for (int i = 0; i < friNum; i++) {
        std::cout << fri[i] << std::endl;
    }
}

void User::findWeiNum(const User* u, const std::string& weiNum, const int len) {
    bool found = false;
    for (int i = 0; i < len; i++) {
        if (u[i].getWeiNum() == weiNum) {
            this->addFriend(weiNum);
            std::cout << "成功添加好友" << weiNum << std::endl;
            found = true;
            break;
        }
    }
    if (!found) {
        std::cout << "未找到好友" << weiNum << std::endl;
    }
```

```
}

void User::findQQNum(const User* u, const std::string& QQNum, const int len) {
    bool found = false;
    for (int i = 0; i < len; i++) {
        if (u[i].getQQNum() == QQNum) {
            this->addFriend(u[i].getWeiNum());
            std::cout << "成功添加好友" << u[i].getWeiNum() << std::endl;
            found = true;
            break;
        }
    }
    if (!found) {
        std::cout << "未找到好友" << QQNum << std::endl;
    }
}

int User::findPhoneNum(const User* u, const std::string& PhoneNum, const int len, bool isAdd) {
    for (int i = 0; i < len; i++) {
        if (u[i].getPhoneNum() == PhoneNum) {
            if (isAdd) {
                this->addFriend(u[i].getWeiNum());
                std::cout << "成功添加好友" << u[i].getWeiNum() << std::endl;
                return i;
            } else {
                return i;
            }
        }
    }
    if (isAdd) {
        std::cout << "未找到好友" << PhoneNum << std::endl;
    }
    return -1;
}

void User::showTuiJianFriend(const User* u, const std::string* phoneNums, const int len) {
    std::cout << "推荐好友:" << std::endl;
    for (int i = 0; i < 5; i++) {
        int id = this->findPhoneNum(u, phoneNums[i], len, false);
        if (id != -1) {
            std::cout << u[id].getWeiNum() << std::endl;
        }
    }
}

#ifndef _GROUP_H_
#define _GROUP_H_

#include <iostream>
#include <string>

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 std::string& GName, const std::string& CName, const int len = 20) {
        groupName = GName;
        createrName = CName;
        userNum = 0;
        user = new std::string[size = len];
```

```
    }

    ~Group() {
        delete[] user;
    }

    // 成员函数:获取群名称、群主
    std::string getGroupName() const {
        return groupName;
    }

    std::string getCreaterName() const {
        return createrName;
    }

    // 显示群成员
    void showNumbers() const {
        std::cout << groupName << " 群共有成员 " << userNum << " 名, 其中群主是 " << createrName << std::endl;
        for (int i = 0; i < userNum; i++) {
            std::cout << user[i] << std::endl;
        }
    }

    // 添加群成员
    void addNumbers(const std::string& name) {
        user[userNum++] = name;
    }

private:
    std::string groupName;
    std::string createrName;
    std::string* user;
    int size;
    int userNum;
};

#endif

#include <iostream>
#include <string>
#include "User.h"
#include "Group.h"

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.com");
    u[1].set("李三", "12444", "138", "122@qq.com");
    u[2].set("王三", "12344", "187", "122@qq.com");
    u[3].set("韩三", "12344", "147", "122@qq.com");
    u[4].set("赵三", "45454", "117", "122@qq.com");
    u[5].set("冯三", "123432321", "133", "122@qq.com");
    u[6].set("马三", "12323132", "134", "122@qq.com");
    u[7].set("徐三", "124343", "139", "122@qq.com");

    // 按照微信号添加
    u[0].findWeiNum(u, "韩三", 8);
    // 按照 qq 号添加
    u[0].findQQNum(u, "13444", 8);
```

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

```
#ifndef _RECORD_H_
#define _RECORD_H_

#include <iostream>
#include <string>

class Record {
public:
    Record() {}
    ~Record() {}

    // 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 std::string& Name, const std::string& Phone, const std::string& Email) {
        name = Name;
        phone = Phone;
        email = Email;
    }

    // 返回数据成员值
    std::string getName() const { return name; }
    std::string getPhone() const { return phone; }
    std::string getEmail() const { return email; }

    // 显示联系人基本信息
    void show() const {
        std::cout << name << " " << phone << " " << email << std::endl;
    }

```

```
private:
    // 确定数据成员
    std::string name;
    std::string phone;
    std::string email;
};

#endif

#ifndef _CONTACT_H_
#define _CONTACT_H_

#include "Record.h"
#include <fstream>

class Contact {
public:
    // 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];
        num = 0;
    }

    // 释放申请的空间
    ~Contact() {
        delete[] r;
    }

    // 添加联系人到通讯录中
    void add(const std::string& name, const std::string& phone, const std::string& email) {
        r[num++].setRecord(name, phone, email);
    }

    // 显示数组中所有的联系人的信息
    void allShow() const;

    // 获取当前通讯录总人数
    int getNum() const { return num; }

    // 将通讯录导入到文件中去
    void output(const std::string&);

    // 分别按照姓名、电话、email 查找联系人，查找成功返回索引，失败返回 -1
    int findName(const std::string&, bool isshow = true) const;

    // 合并两个通讯录
    void meg(const Contact&);

    // 根据下标位置获取联系人信息
    std::string getNameById(int id) const {
        return r[id].getName();
    }

    std::string getPhoneById(int id) const {
        return r[id].getPhone();
    }

    std::string getEmailById(int id) const {
        return r[id].getEmail();
    }
```

```
91
92      // 模糊查找
93      void search(const std::string&) const;
94
95  private:
96      // 确定私有数据成员
97      Record* r;
98      int num;
99  };
100
101 #endif
102
103 #include "Contact.h"
104
105 void Contact::allShow() const {
106     for (int i = 0; i < num; i++) {
107         r[i].show();
108     }
109 }
```