1. 三个数从小到大排列

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

int main() {
    int a, b, c;
    // 提示用户输入三个整数
    cout << "请输入三个整数: ";
    cin >> a >> b >> c;

    int temp;
    // 确保 a 是最小的
    if (a > b) {
        temp = a;
        a = b;
        b = temp;
    }
    if (a > c) {
        temp = a;
        a = c;
        c = temp;
    }
    // 确保 b 小于 c
    if (b > c) {
        temp = b;
        b = c;
        c = temp;
    }

    // 输出从小到大排列后的三个数
    cout << "从小到大排列为: " << a << " " << b << " " << c << endl;
    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（ ）

```cpp
#include <iostream>
#include <string>

// 查找子字符串的位置
int find(const std::string& str, const std::string& substr) {
    int strLen = str.length();
    int subLen = substr.length();

```

```
    for (int i = 0; i <= strLen - subLen; ++i) {
        int j;
        for (j = 0; j < subLen; ++j) {
            if (str[i + j] != substr[j]) {
                break;
            }
        }
        if (j == subLen) {
            return i;
        }
    }
    return -1;
}

// 提取指定位置的子字符串
void substr(const std::string& str, int start, int length, std::string& result) {
    if (start < 0 || start >= str.length()) {
        result = "";
        return;
    }
    if (length < 0 || start + length > str.length()) {
        length = str.length() - start;
    }
    result = str.substr(start, length);
}

// 拆分字符串成两段
bool split(const std::string& str, const std::string& delimiter, std::string& str1,
std::string& str2) {
    int pos = find(str, delimiter);
    if (pos == -1) {
        return false;
    }
    str1 = str.substr(0, pos);
    str2 = str.substr(pos + delimiter.length());
    return true;
}

// 替换指定位置的子字符串
std::string replace(const std::string& str, int start, int length, const std::string&
replacement) {
    if (start < 0 || start >= str.length()) {
        return str;
    }
    if (length < 0 || start + length > str.length()) {
        length = str.length() - start;
    }
    return str.substr(0, start) + replacement + str.substr(start + length);
}

// 删除指定位置的子字符串
std::string erase(const std::string& str, int start, int length) {
    if (start < 0 || start >= str.length()) {
        return str;
    }
    if (length < 0 || start + length > str.length()) {
        length = str.length() - start;
    }
    return str.substr(0, start) + str.substr(start + length);
}

```

```cpp
// 删除整个字符串
std::string dele(const std::string& str) {
    return "";
}

int main() {
    std::string original = "this 1s string";

    // 测试 substr
    std::string subResult;
    substr(original, 5, 2, subResult);
    std::cout << "substr 结果: " << subResult << std::endl;

    // 测试 split
    std::string splitResult1, splitResult2;
    if (split(original, "1s", splitResult1, splitResult2)) {
        std::cout << "split 结果: " << splitResult1 << " 和 " << splitResult2 << std::endl;
    } else {
        std::cout << "未找到分隔符" << std::endl;
    }

    // 测试 replace
    std::string replaceResult = replace(original, 5, 2, "is");
    std::cout << "replace 结果: " << replaceResult << std::endl;

    // 测试 erase
    std::string eraseResult = erase(original, 2, 6);
    std::cout << "erase 结果: " << eraseResult << std::endl;

    // 测试 dele
    std::string deleResult = dele(original);
    std::cout << "dele 结果: " << deleResult << std::endl;

    return 0;
}
```

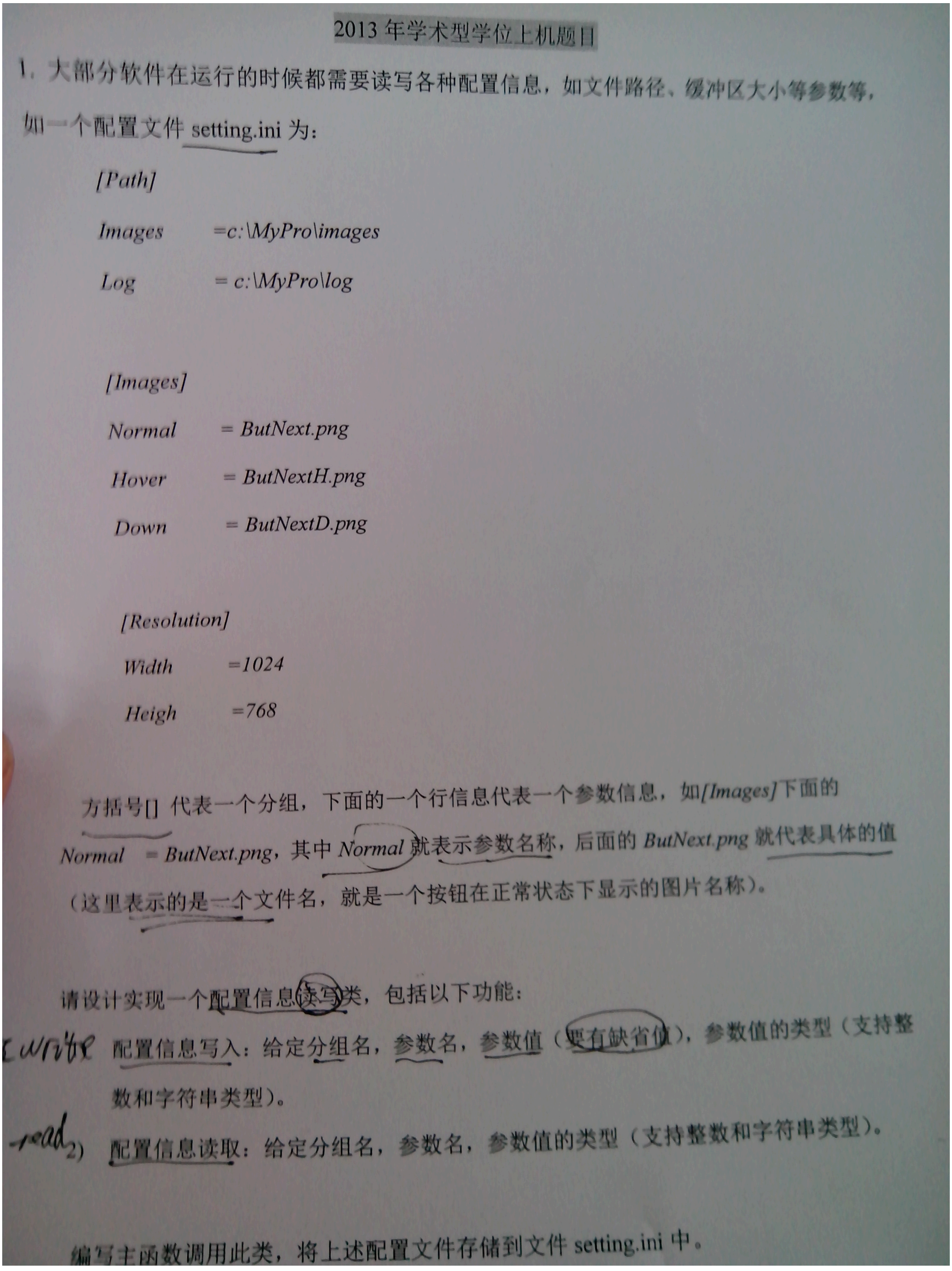
3. **比 2014学硕要简单的手机通讯录的实现：主要是在删除联系人和去重、同步更新问题上注意下**：

```cpp
/*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>
#include <string>

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 std::string&, const std::string&, const std::string&);

```

```
    // 显示全部的联系人信息
    void showAll() const;

    // 通过姓名查找
    int findName(const std::string&, bool) const;

    // 通过姓名删除指定联系人
    void delName(const std::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
    std::ifstream in("list.txt");
    std::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 (std::getline(in, str)) {
        std::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("%");
            std::string Phone = str.substr(0, k);
            str = str.substr(k + 1, str.length() - k - 1);
            std::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)
    std::ofstream out("list.txt");
    for (int i = 0; i < number; i++) {
        out << r[i].getName() << "%" << r[i].getPhone() << "%" << r[i].getEmail() << std::endl;
    }
    out.close();
}

void Contact::delRepate() {
    std::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 delName 函数直接删除
            if (r[j].getName() == tempName) {
                delName(tempName);
            }
```

```
        }
    }
}

void Contact::delName(const std::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 {
        std::cout << "未找到联系人 " << name << " 无法删除" << std::endl;
    }
}

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

void Contact::showAll() const {
    for (int i = 0; i < number; i++) {
        r[i].showFriend();
    }
}

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

#endif
```

学硕



2013 年学术型学位上机题目

1. 大部分软件在运行的时候都需要读写各种配置信息，如文件路径、缓冲区大小等参数等，如一个配置文件 setting.ini 为：

```
[Path]
Images      =c:\MyPro\images
Log         = c:\MyPro\log

[Images]
Normal      = ButNext.png
Hover       = ButNextH.png
Down        = ButNextD.png

[Resolution]
Width       =1024
Heigh       =768
```

方括号[] 代表一个分组，下面的一个行信息代表一个参数信息，如*[Images]*下面的 *Normal = ButNext.png*，其中 *Normal* 就表示参数名称，后面的 *ButNext.png* 就代表具体的值（这里表示的是一个文件名，就是一个按钮在正常状态下显示的图片名称）。

请设计实现一个配置信息读写类，包括以下功能：

1) 配置信息写入：给定分组名，参数名，参数值（要有缺省值），参数值的类型（支持整数和字符串类型）。

2) 配置信息读取：给定分组名，参数名，参数值的类型（支持整数和字符串类型）。

编写主函数调用此类，将上述配置文件存储到文件 setting.ini 中。

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>

// 避免使用 using namespace std;，防止命名冲突
// 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 std::string[10];
        pValue = new std::string[10];
        num = 0;
    }

    // 析构函数
    ~Group() {
        delete[] pName;
        delete[] pValue;
    }

    // 设置分组名: 分组名只需要设置一次，而属性信息需要设置多次
    void setGroupName(const std::string&);

    // 设置每一个分组的信息
    void setGroup(const std::string&, const std::string&);

    // 显示当前分组的信息
    void show() const;

    // 将当前分组输出到文件中去
    void outputFile() const;

private:
    std::string GName;  // 小组的名字
    std::string* pName; // 存储小组各个属性名字的数组
    std::string* pValue; // 属性对应的数值
    int num;  // 记录多少个属性
};

void Group::outputFile() const {
    // app 后缀，写入操作不会删除原来的数据，只是在末尾追加
    std::ofstream out("setting.ini", std::ios_base::app);
    out << "[" << GName << "]" << std::endl;
    for (int i = 0; i < num; i++) {
        out << pName[i] << " =" << pValue[i] << std::endl;
    }
}

void Group::show() const {
    std::cout << "[" << GName << "]" << std::endl;
    for (int i = 0; i < num; i++) {
        std::cout << pName[i] << " =" << pValue[i] << std::endl;
    }
}

void Group::setGroup(const std::string& PN, const std::string& PV) {
    pName[num] = PN;
    pValue[num] = PV;
    num++;
}

void Group::setGroupName(const std::string& GN) {
    GName = GN;
}

#endif
```