**暨南大學本科實驗報告**

課程名稱 C＋＋程序設計 成績評定

實驗項目名稱 课程设计 指導教師 周密

實驗項目編號 實驗項目類型 驗證 實驗地點

學生姓名 陈明晶 學號 2014053689

學院 電氣信息學院 專業 软件工程

實驗時間 2016年 5 月11日上午

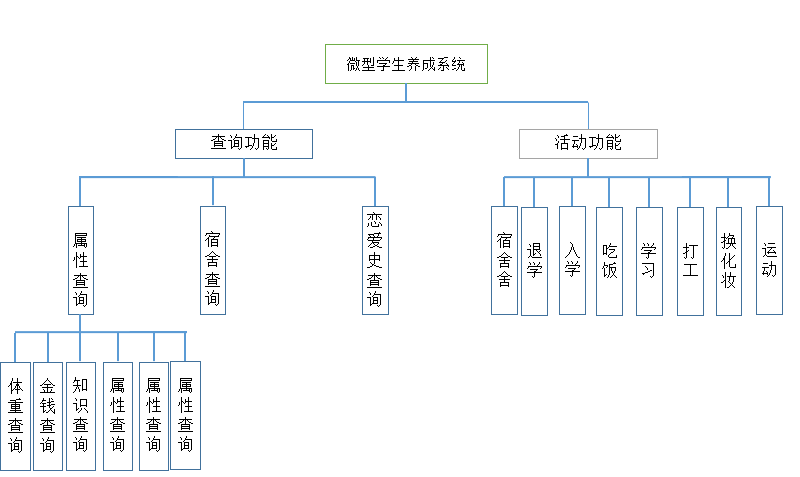
**一、實驗目的**

通過設計、實現一個微型學生養成系統，熟悉對象導向的設計思想；進一步熟悉、掌握STL中有關容器、算法的使用，以及泛型編程的風格。

**二、實驗環境**

Microsoft Visual C++ 2015編程環境。

**三、實驗內容及分析**

**1.系統功能模塊圖**

**2.程序實現思路**

(1). 设计学生及宿舍的数据结构，并实现相关的功能、初始化

(2). 读入文件内容，初始化学生和学生宿舍的数据，并对指令文件(instruction.txt)的内容进行解析，完成学生活动的各种操作。

(3). 完成个各种活动操作后，得到学生变化后的数据，对学生的具体数据的查询

**3.设计说明**

**Config文件的读取**

Config文件里的数据需要在创建和各操作的过程中多次用到，若反复使用map太时间开销大且不够清楚表达常数的使用，所以这里用了全局变量，但千万不要再使用过程中改变全局变量

Config.cpp

//关于宿舍的常量

extern int FLOOR;

extern int ROOM;

extern int CAPACITY;

**学生类**

学生类分为学生基类,有換宿舍等行为和相应的变量,，男学生(运动和健康)和女学生类(化妆和容貌)

学生基类

在创建一个学生后，需要对其尽心初始化，但初始化分为两种情况:①从配置文件读取学生类的初始化，因为配置文件里只有宿舍、性别、姓名基本信息，而属性等初始化数据在config文件，所以之后还要给其赋值，完成这个操作的函数为

void read(const std::vector<std::string>& );

②在执行入学指令时，需要创建学生，其初始化函数为

void orderRead(std::string n, std::string gen,int f,int r) { name = n; gender = gen; floorID = f; roomID = r; }

求爱与被求爱的时间一致，每调用一次wood,恋爱时间加1，但wooded不加1.

恋爱史的记住，这里在love\_history.h建立了恋爱史表，含有恋人姓名和恋爱时间，设置恋爱状态love\_status用于避免吃着碗里望着锅里的行为。

**宿舍樓、樓層、房間类**

宿舍楼包含各个楼层的数据，各个楼层包含房间的数据，房间类的数据结构包含

容量capacity,已入住学生(students)人数lived\_ss,宿舍入住的性别(因为男女不能同宿舍，且当宿舍没人时，性别为”none“,男女均可入住)

class Room

{

public:

Room():lived\_ss(0), gender("none") { capacity = CAPACITY; }

int capacity ;

int lived\_ss;

std::string gender; //性别初始化为none，当女生住是个别为"女"，男生则相反

std::vector<std::string> students; //学生姓名

};

**文件的指令解析和执行**

完成解析功能的文件有analyze\_orders.cpp (读入并解析指令操作码)，

void executeOrders

int ordersImplement

orders\_meaning.cpp(各个操作码的操作函数)如ADD();EAT()，等

，split.cpp(将指令分成几个字符串)，因为第一个大写字符串的为操作码，如ADD 5 FLOOR(s)、华为 WORK 6，

ADD\WORK操作码都是第一个大写字母字符串，我们遇到操作码时进行相关的操作，每次调用ordersImplement实现一次操作码的操作，若一条指令有多个操作码，则需要多次调用ordersImplement，改变i值可以实现每条指令的多个操作码的功能，详细见源代码

for (int i = 0; i < words.size(); i++)//words为指令分成的danci

{

if (words[i][0] >= 'A' && words[i][0] <= 'Z')

{

ordersImplement(i, words, building, m\_ss, f\_ss,Qm\_ss,Qf\_ss);

}

}

**查询功能设计**

**宿舍(dorm)查询**

**(search\_dorm\_info.cpp)**

根据名字查询，map的映射功能可以解决

查询宿舍里的成员，直接因为房间数据结构里存有学生姓名，

vector可以直接找到对应宿舍的宿舍成员

**恋爱史(love history)查询**

**(student\_history\_love.cpp)**

为了方便排序和查询，这里有建立恋人信息表，与前面的恋爱史表不同的是多了恋人的魅力值

typedef struct lover\_info

{

string name ;

int wood\_time ;

double charm ;

} Lover\_info;

恋爱史包括在校生(ss)和已退学生（Q\_ss）,

根据名字查询一个学生的恋爱史，只要用map的功能就可实现，恋爱史的显示可以有按建立戀愛關係的時間排序，戀人姓名排序，魅力值排序、戀人魅力值最高的恋人

对恋爱史最多的学生查询，用的办法比较笨，即从第一个遍历(因为时间关系，只在属性查询时改用了先排序后查询的办法)

**属性(attribute)查询**

**(search\_attribute\_list.cpp ; search\_attribute.cpp)**

體重，金錢，知識，容貌（女生適用），健康（男生適用），魅力進行最大、最小值查詢，範圍查詢（亦即查詢屬性值屬於某個範圍的所有學生），每个功能块大同小异。因为类的数据被封装，用户无法调用，所以用了在函数中用return 取得个属性的值，之前想过用指针，但属性变量已被保护。所以又在attribute\_name.h文件里设一个含有属性值(attribute)和学生姓名的数据结构

typedef struct attribute\_name

{

double attribute;

std::string name;

} attributeName;

因为要对六种属性的查询，所以建立了下面六个vector向量

vector<attributeName> weight\_names;

vector<attributeName> money\_names;

vector<attributeName> knowledge\_names;

vector<attributeName> charm\_names;

vector<attributeName> health\_names;

vector<attributeName> looks\_names;

然后学生数据给它们分别初始化，并降序排序，該屬性相等的學生再按照姓名升序或降序排列。

bool compare\_attribute\_name(const attributeName& x, const attributeName& y)

{

if (x.attribute > y.attribute)

return true;

else if (x.attribute == y.attribute && x.name < y.name)

return true;

return false;

}

void MySort(vector<attributeName>& attribute\_names)

{

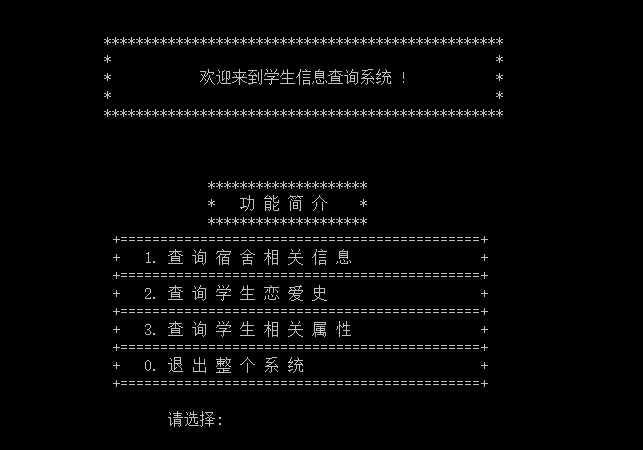
sort(attribute\_names.begin(), attribute\_names.end(), compare\_attribute\_name);

}

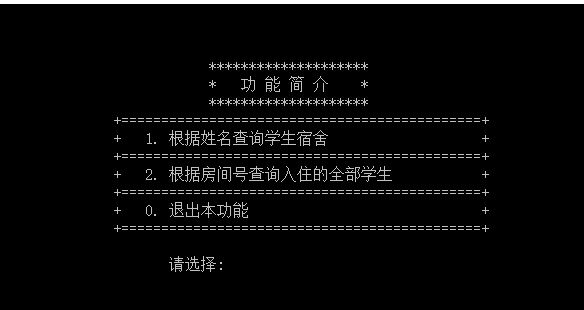
每个属性查询函数调用对应的函数实现即可

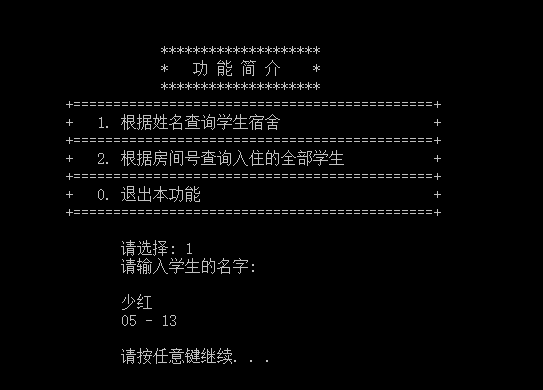
**四、调试结果**

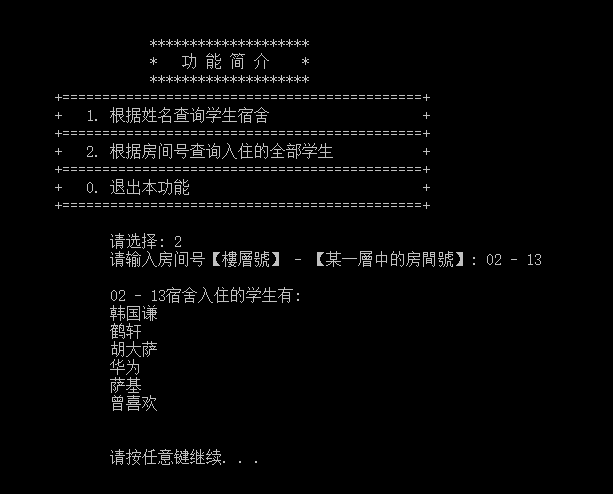
**开始界面**

****

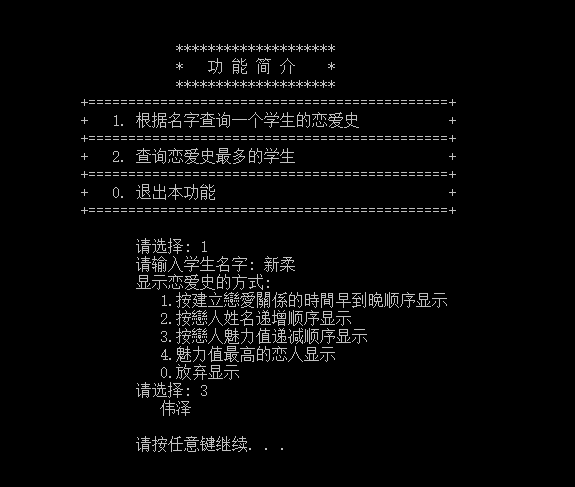
**宿舍查询**

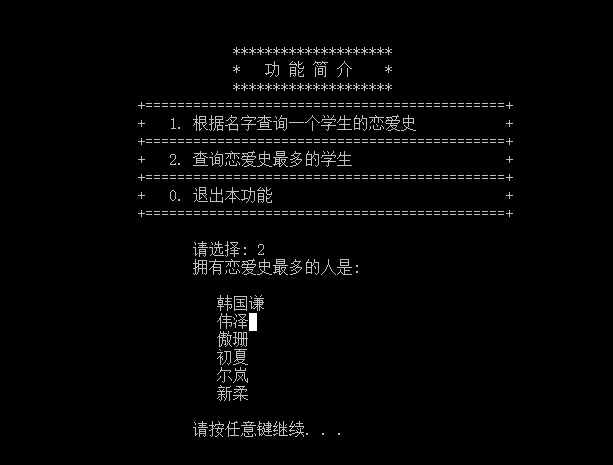
****



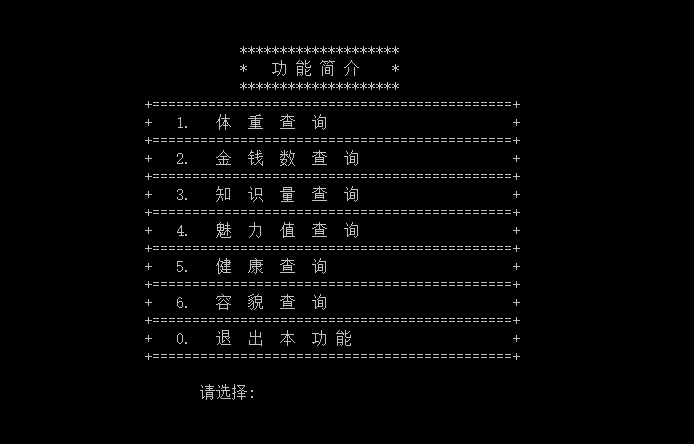
****

**恋爱史查询**

****

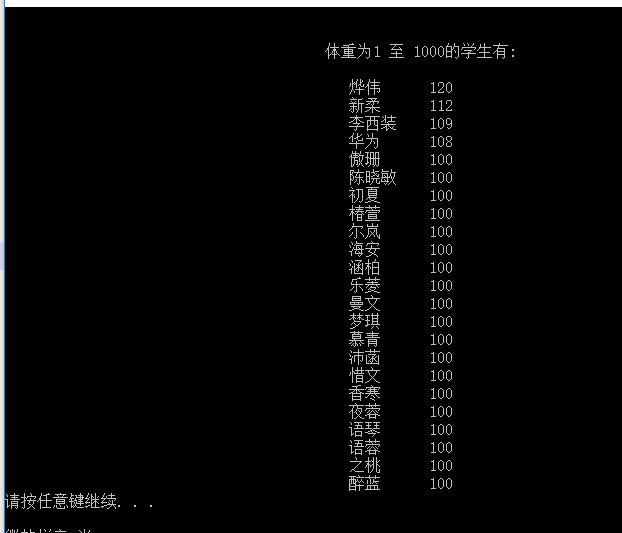
****

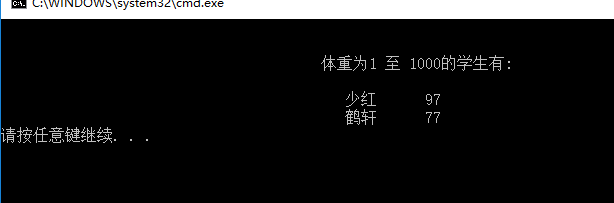
**学生属性查询(以查询体重为例)**

****

****

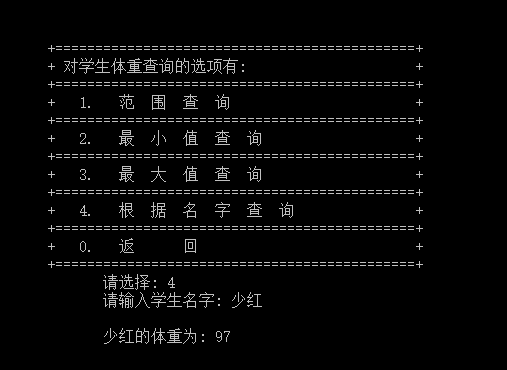
**继续显示**

****

****

****

****

****

**五、設計心得**

课程项目设计中，考虑的细节甚多。比如，在选择数据类型上，因为有大量以名字为关键字的查询，若用链表或vector存储，则查找上的开销很大，而map既有多元的映射功能，其数据结构为红黑树，查找起来开销也小。除了这以外，在查找方面也许考虑性能方面的问题，存储方面要考虑内存开销和使用性能方面的问题，考虑代码的冗余、可扩充、可阅读性等问题，考虑算法性能问题…… 甚是想十全十美，但感觉程序越大越难做到，各方面要有适当的取舍，如在本项目中牺牲了存储而成全了性能，到底作何选择，应按需而定。