学生籍贯信息记录簿系统

一、问题描述

编制一个学生籍贯信息记录簿，每个学生信息包括：学号、姓名、籍贯。具体功能：

1）创建信息链表并以磁盘文件保存；

2）读取磁盘文件并显示输出所有学生的籍贯信息；

3）按学号或姓名查询其籍贯；

4）按籍贯查询并输出该籍贯的所有学生；

5）能添加、删除和修改学生的籍贯信息； 

6）显示输出四川籍和非四川籍学生的信息并可分别存盘。

二、设计任务分析

1 数据结构分析

根据上面的问题描述可知，该系统使用的主要数据结构是链表，问题说提到的六点要求都是于链表的基本操作相关，所以设计该系统的首要任务是定义并实现一个链表。进一步分析可知，仅使用单链表即可完成问题所要求的所有功能。针对该系统的需求，我们设计的单链表至少应该具备如下功能：

1. 创建链表
2. 销毁链表
3. 清空链表
4. 链表判空
5. 添加结点
6. 删除结点
7. 查找结点
8. 修改结点
9. 获取链表长度
10. 链表排序
11. 遍历链表

链表是一种抽象数据类型（Abstract Data Type, ADT），所谓的抽象数据类型其实是计算机科学中具有类似行为的特定类别的数据结构的数学模型；或者具有类似语义的一种或多种程序设计语言的数据类型。通常来说，定义一种抽象数据类型时应该同时具备以下两部分定义：1) 定义一种的新的数据；2) 定义与这种数据相关的操作。显而易见，抽象数据类型的定义和C++语言里面“类”的定义是相似的，故我们可以使用C++的“类”来定义我们的单链表。

一般地，定义链表时应该先定义链表结点，单链表的结点由两部分组成：数据域和指针域。对于该系统来说，链表结点的数据域应该用来保存学生的学号、姓名和籍贯信息，考虑数据的独立性，我们单独设立了一个Student类来管理单个学生的信息。最终我们设计了如下两个类：a）Student b）LinkList。

2 功能模块分析

为了满足题目的要求，我们为系统设计了7个功能模块：1）学生信息搜索模块 2）学生信息编辑模块 3）添加学生信息模块 4）删除学生信息模块 5）提取学生信息模块 6）显示学生信息模块 7）退出程序模块

三、模块设计

根据上面的分析，我们为设计设计了7个功能模块。对于每个模块我们分别进行了设计：

1 学生信息搜索模块

该模块对应的功能执行函数是 searchFunc()。该模块共支持三种搜索模式：按学号查询、按姓名查询和籍贯查询。对于查询到的学生应该进行整齐打印。

2 学生信息编辑模块

该模块对应的功能执行函数是 editFunc()。在进行修改学生信息的时候应该同时支持查询和显示学生的信息，以方便管理员确定修改的学生是否是期待修改的学生。

3 添加学生信息模块

该模块对应的功能执行函数是 addFunc()。在添加学生的时候应该对新的学生信息进行验证，不能出现学号相同的学生（注：姓名和籍贯可以相同），故应该对用户的错误输入进行处理，并在发生错误时能够提示并让用户重新输入。同时在最终添加新数据到数据库中前应该让用户进一步确定是否添加。

4 删除学生信息模块

该模块对应的功能执行函数是 deleteFunc()。该模块提供了三种删除方式：按学号删除、按名字删除和按位置删除。

5 学生数据抽取模块

该模块对应的功能执行函数是 extractFunc()。该模块按先查询后保存的方式进行数据地抽取和存档。其中查询支持反选，以方便数据提取。

6 学生信息显示模块

该模块对应的功能执行函数是 showFunc()。该模块主要是所有学生的信息进行格式化输出。共提供了四种格式：直接输出、按学号排序、按姓名排序和按籍贯排序，排序支持逆序排序。

7 程序退出模块

该模块对用的功能执行函数是 quitFunc()。此模块主要是打印一些退出时的问候语。

三 主要数据结构

这里主要用到了两个类：

class Student

{

private:

string id; // 学生学号

string name; // 学生姓名

string address; // 学生籍贯

public:

Student();

Student(const Student &stud);

Student(const vector<string> &stud);

Student(string \_id, string \_name, string \_address);

~Student();

// 设置学生学号

void setID(string \_id);

// 设置学生姓名

void setName(string \_name);

// 设置学生籍贯

void setAddress(string \_address);

// 从键盘输入学生信息

void input();

// 获取学生学号

inline string getID() const { return id; }

// 获取学生姓名

inline string getName() const { return name; }

// 获取学生籍贯

inline string getAddress() const { return address; }

Student & operator=(const Student &stud);

// 显示学生信息

void show() const;

};

class LinkList

{

private:

typedef Student Item;

struct Node

{

Item data; // 数据域

Node \* next; // 指针域

};

Node \*head; // 头指针

Node \*tail; // 尾指针

int size; // 当前链表长度

// 简介: 排序结点比较函数

// 参数:

// x : 结点x \< LinkList::Node >

// y : 结点y \< LinkList::Node >

// key : 用于比较的的属性名 \< std::string >

// reverse : 是否降序排序 \< bool >

// 返回值:

// 返回一个布尔值

bool cmpfunc(const Node \*x, const Node \*y,

const string key, const bool reverse);

public:

typedef struct Node\* PNode;

LinkList();

LinkList(const LinkList &L);

~LinkList();

bool isEmpty();

bool addNode(const Item &item, string type = "head"); // 头插法或尾插法

bool addNode(const Item &item, Node \* ppos); // 在ppos所指向结点的前面插入新结点

bool addNode(const Item &item, int pos); // 在第pos个位置前插入新结点

bool removeNode(Node \* ppos); // 删除ppos所指向的结点

bool removeNode(int pos); // 删除第pos个位置处的结点

bool removeNode(const string str, string type = "id"); // 按指定方式删除结点

void sort(const string key = "id", bool reverse = false); // 链表排序

void clean(); // 清空链表

void traverse() const; // 遍历链表

int getSize() const; // 获取当前链表长度

PNode getNode(int pos) const; // 获取第 pos 个结点

PNode getNode(const string id) const; // 获取学号为id的结点

vector<Student> search(string str, string type = "name", bool inverse = false) const;

LinkList & operator=(const LinkList &L);

friend LinkList operator+(const LinkList &L1, const LinkList &L2);

};

五 流程图

1 搜索模块流程图



2 编辑模块流程图



3 显示模块流程图



六 测试情况

七 设计总结与体会