《数据结构》上机报告

2019 年 9 月 27 日

姓名：李佳庚 学号：1852409 班级：计算机1班 得分：

<table>
<tr><td>实验题目</td><td colspan="2">顺序表</td></tr>
<tr><td>问题描述</td><td colspan="2">顺序表是指采用顺序存储结构的线性表，它利用内存中的一片连续存储区域存放表中的所有元素。可以根据需要对表中的所有数据进行访问，元素的插入和删除可以在表中的任何位置进行。</td></tr>
<tr><td rowspan="2">基本要求</td><td colspan="2">实验目的:
1. 掌握线性表的定义及顺序表示。
2. 掌握顺序表实现线性表的基本操作，如建立、查找、插入、删除以及去重等。
3. 掌握顺序表的特点。

实验内容:
定义一个包含学生信息（学号，姓名，成绩）的顺序表，使其具有如下功能：
(1) 根据指定学生个数，逐个输入学生信息；
(2) 逐个显示学生表中所有学生的相关信息；
(3) 根据姓名进行查找，返回此学生的学号和成绩；
(4) 根据指定的位置可返回相应的学生信息（学号，姓名，成绩）；
(5) 给定一个学生信息，插入到表中指定的位置；
(6) 删除指定位置的学生记录；
(7) 统计表中学生个数。
(8) 删除某学生的所有记录
(9) 删除所有重复记录</td></tr>
<tr><td>已完成基本内容（序号）:</td><td>（1），（2），（3），（4），（5），（6），（7），（8，）（9）</td></tr>
<tr><td>选做要求</td><td colspan="2">1. 程序可读性较强，并且添加适当的注释，虽然代码是我自己写的，我自己看肯定理解得快一点，但是我觉得没有学过顺序表的人看一遍也能捋下来。
2. 程序完成题目要求的健壮性。
3. 程序界面比较友好，增加输入提示和输出提示。
4. 计算各个算法的复杂度。
5. 完成删除值为 e 的所有记录的算法和去重算法（distinct）的流程图和复杂度分析。</td></tr>
</table>

	已完成选做内容（序号）	（1），（2），（3），（4），（5）

数据结构设计

就当下我个人的知识范围和代码能力，暂且将完成数据结构作业总共划分为四个步骤：

1. 归纳、分析需要解决的问题本身。
2. 了解解决问题过程中的诸多需求。
3. 将需求分散到各个子问题当中。
4. 根据问题和外加限制挑选/设计数据结构。
5. 敲代码，按先前设定步骤，形成可测试独立单元。
6. 对单个独立单元测试。
7. 将正确无误的代码与之前的原型结合。
8. 不断重复 5~7 过程。
9. 最终得到完整的可执行程序。

而数据结构设计本身只涉及了 1~4 步骤。

对于 HW2 的三个题目：

① 顺序表基本操作，

② 删除某学生所有记录，

③ 顺序表去重。

其中设计的操作有：

题目	①	②	③
操作	1. 插入节点 2. 索引删除节点 3. 查找节点 4. 节点数目统计	1. 顺序表创建 2. 数值删除节点 3. 顺序表销毁	1. 去重

总共需要解决的问题就是：

完成一个顺序表，要求这个顺序表可以创建、销毁(1)，可以根据索引位置插入和删除(2)，可以根据索引和值查找返回需要的数据(3)，可以统计表中元素数目(4)，去重(5)。

我写这几道题是比较晚的,听隔壁班的同学说,好像第一道题对时间复杂度卡的比较严格,尤其是排序操作,要求的时间复杂度是 n^2 之下。

对于数据结构，由于题目的要求，只能够选用线性表。于是我根据问题规模，设计了如下的数据结构：

```
class studentSystem {
```

```
public:
    class student {
    public:
        // 学生信息
        char stu_no[8];
        char stu_name[20];
        double stu_score;

    public:
        // 默认构造函数
        student() = default;
        // 普通构造函数：输入学号+姓名+成绩
        student(const char *stu_no_, const char* stu_name_, const double stu_score_)
        // 拷贝赋值运算符重载函数：将另外一个对象进行赋值
        student &operator = (const student &another) { ... }

    };
private:
    static const int maxNum = 10000;
    student systemP[maxNum];
    int length;
private:
    // 构造函数的封装
    // 传入studentlist，对studentlist里面所有的学生输入到sys中
    void _init(const int listsize_, student *studentlist) { ... }
```

```
public:
    // 默认构造函数
    studentSystem() = default;

    // 拷贝构造函数：输入另外一个学生系统
    studentSystem(const studentSystem &another) { ... }

    // 1
    // 在该函数中输入学生的数量，以及学生的信息
    bool create() { ... }


    // 2
    // 展示list中的信息
    void display() { ... }

    // 3
    // 展示对应姓名的学号、成绩
    bool selectByName(const char *name_) { ... }

    // 4
    // 对应索引搜索
    bool selectByLoc(const int index) { ... }

    // 5
    bool insert(const int index, const student inserted) { ... }
```

```
    // 6
    // index从1开始
    bool deleteByLoc(const int index) { ... }

    // 7
    constexpr int getNum() { ... }

    // 8
    // 删除拥有同姓名的所有个体的信息
    bool deleteAllData(const char *name_) { ... }

    // 9
    // length 从0开始
    bool distinct() { ... }


    void clear() { ... }
};
```

studentsystem 中有一嵌套类 student，专门描述学生信息。
maxNum 是静态数据成员，为顺序表最大长度。
Length 为顺序表当前元素个数，即当前长度。
systemP 为学生系统的头指针。

Student 中有学生的学号、姓名、成绩三个数据成员。

功能(函数)说明

1．嵌套类 Student 成员函数：

1．
```
// 默认构造函数
student() = default;
```

2．
```
// 普通构造函数：输入学号+姓名+成绩
student(const char *stu_no_, const char* stu_name_, const double stu_score_)
    : stu_score(stu_score_) {
    strcpy_s(stu_no, stu_no_);
    strcpy_s(stu_name, stu_name_);
}
```

3．
```
// 拷贝赋值运算符重载函数：将另外一个对象进行赋值
student &operator = (const student &another) {
    strcpy_s(stu_no, another.stu_no);
    strcpy_s(stu_name, another.stu_name);
    stu_score = another.stu_score;
    return *this;
}
```

2．Studentsystem 成员函数：

1．
```
// 构造函数的封装
// 传入studentlist，对studentlist里面所有的学生输入到sys中
void _init(const int listsize_, student *studentlist) {
    length = 0;
    for (int i = 0; i < listsize_; i++) {
        systemP[i] = studentlist[i];
        length++;
    }
}
```

2．
```
// 默认构造函数
studentSystem() = default;
```

3.
```
// 拷贝构造函数：输入另外一个学生系统
studentSystem(const studentSystem &another)
    : length(0) {
    for (int i = 0; i < another.length; i++) {
        strcpy_s(systemP[i].stu_no, another.systemP[i].stu_no);
        strcpy_s(systemP[i].stu_name, another.systemP[i].stu_name);
        systemP[i].stu_score = another.systemP[i].stu_score;
    }
}
```

4.
```
// 1
// 在该函数中输入学生的数量，以及学生的信息
bool create() {
    int listsize_;
    std::cout << "请输入学生数量\n";
    std::cin >> listsize_;

    if (listsize_ > maxNum)
        return false;

    // 生成一个空的list，把信息填到list里面
    student studentlist[maxNum];
    std::cout << "请依次输入学生学号、姓名、成绩\n";
    for (int i = 0; i < listsize_; i++) {
        char stu_no_[8];
        char stu_name_[20];
        double stu_score_;
        std::cin >> stu_no_ >> stu_name_ >> stu_score_;
        strcpy_s(studentlist[i].stu_no, stu_no_);
        strcpy_s(studentlist[i].stu_name, stu_name_);
        studentlist[i].stu_score = stu_score_;
    }
    // 传入list，对private中的list进行赋值
    _init(listsize_, studentlist);
    return true;
}
```

5.
```
// 2
// 展示list中的信息
void display() {
    for (int index = 0; index < length; index++)
        std::cout << systemP[index].stu_no
        << " " << systemP[index].stu_name
        << " " << systemP[index].stu_score << '\n';
}
```

6.
```
// 3
// 展示对应姓名的学号、成绩
bool selectByName(const char *name_) {
    bool isSelect = false;
    for (int i = 0; i < length; i++)
        // 比较
        if (0 == strcmp(systemP[i].stu_name, name_)) {
            isSelect = true;
            std::cout << systemP[i].stu_no << " " << systemP[i].stu_score << '\n';
        }

    if (!isSelect)
        std::cout << -1 << '\n';
    return isSelect;
}
```

7.
```cpp
// 4
// 对应索引搜索
bool selectByLoc(const int index) {
    if (index < 0 || index >= length) {
        std::cout << -1 << '\n';
        return false;
    }
    std::cout << systemP[index].stu_no
        << " " << systemP[index].stu_name
        << " " << systemP[index].stu_score << '\n';
    return true;
}
```

8.
```cpp
// 5
bool insert(const int index, const student inserted) {

    // 如果超范围或者length过大，-1
    if (index < 0 || index > length || length >= maxNum)
        std::cout << -1 << '\n';
        return false;
    }

    int i = length;
    for (; i > index; i--)
        systemP[i] = systemP[i - 1];
    systemP[i] = inserted;

    length++;
    return true;
}
```

9.
```cpp
// 6
// index从1开始
bool deleteByLoc(const int index) {

    // 超范围-1
    if (index <= 0 || index > length) {
        std::cout << -1 << '\n';
        return false;
    }
    // 如果是尾，那么直接剪掉
    else if (index == length) {
        length--;
        return true;
    }


    for (int i = index; i < length; i++)
        systemP[i] = systemP[i + 1];

    length--;
}
```

10.
```cpp
// 7
constexpr int getNum() {
    return length;
}
```

11.

```
// 8
// 删除拥有同姓名的所有个体的信息
bool deleteAllData(const char *name_) {

    //
    bool isExist = false;
    student studentlist[maxNum];
    int newLength = length;

    for (int i = 0, j = 0; i < length; i++) {
        if (0 == strcmp(systemP[i].stu_name, name_)) {
            newLength--;
            isExist = true;
        }
        else {
            studentlist[j] = systemP[i];
            j++;
        }
    }

    length = newLength;
    for (int i = 0; i < length; i++)
        systemP[i] = studentlist[i];


    return isExist;
}
```

12.

```
// 9
// length 从0开始
bool distinct() {
    student studentlist[maxNum];
    int newLength = length;
    // i 指向当前即将赋值的sys，j 指向当前即将被赋值的stulist
    // k 为循环变量，指向即将被比较的
    for (int i = 0, j = 0; i < length; i++) {
        bool isEqual = false;
        for (int k = 0; k < j; k++) {
            // 如果完全相同
            if ((0 == strcmp(studentlist[k].stu_name, systemP[i].stu_name))
                && (0 == strcmp(studentlist[k].stu_no, systemP[i].stu_no))
                && studentlist[k].stu_score == systemP[i].stu_score) {
                isEqual = true;
                newLength--;
                break;
            }
        }
        // 如果没有重复的，就复制
        if (!isEqual) {
            studentlist[j] = systemP[i];
            j++;
        }
    }

    // 如果没有相同的
    if (length == newLength)
        return false;

    // 如果有相同的已经被去除了，就clear systemP
    // 之后重新赋值sysP
    clear();
    for (int i = 0; i < newLength; i++, length++)
        systemP[i] = studentlist[i];
    return true;
}
```

	13. ``` void clear() { for (int i = 0; i < maxNum; i++) systemP[i] = student(); length = 0; } ```
开发环境	Win10 Microsoft Visual Studio Community 2017 15.9.3 Debug x86
调试分析	（运行结果截图） 1．测试 create 函数，建立线性表 ``` 进行对 class studentSystem 的测试 请输入学生数量 4 请依次输入学生学号、姓名、成绩 1650400 叶肯 59.18 1650800 张一 65.19 1652501 李红 77.59 1653010 王豪 34.08 ``` 2．测试 insert 函数，插入多个相同节点 ``` 现在线性表中应该有： 1852409 李佶芮 86.23 1852409 李佶芮 86.23 1852409 李佶芮 86.23 1650400 叶肯 59.18 1852409 李佶芮 86.23 1650800 张一 65.19 1652501 李红 77.59 1653010 王豪 34.08 ``` 3．测试 distinct 函数，去重 ``` 进行去重，现在线性表中应该有： 1852409 李佶芮 86.23 1650400 叶肯 59.18 1650800 张一 65.19 1652501 李红 77.59 1653010 王豪 34.08 ``` 4．测试 display 函数，展示现有节点

```
增加了三个李佶芮，现在线性表中应该有
1852409 李佶芮 86.23
1852409 李佶芮 86.23
1852409 李佶芮 86.23
1852409 李佶芮 86.23
1650400 叶肯 59.18
1650800 张一 65.19
1652501 李红 77.59
1653010 王豪 34.08
```

5．测试 delete 函数，删除所有名字与输入匹配的节点

```
删除李佶芮，现在线性表中应该有：
1650400 叶肯 59.18
1650800 张一 65.19
1652501 李红 77.59
1653010 王豪 34.08
```

6．测试 deleteByIndex 函数，删除 index 为 2 的节点

```
删除第二个，现在线性表中应该有：
1650400 叶肯 59.18
1650800 张一 65.19
1653010 王豪 34.08
```

7．测试 getNum 函数，返回线性表总元素个数。

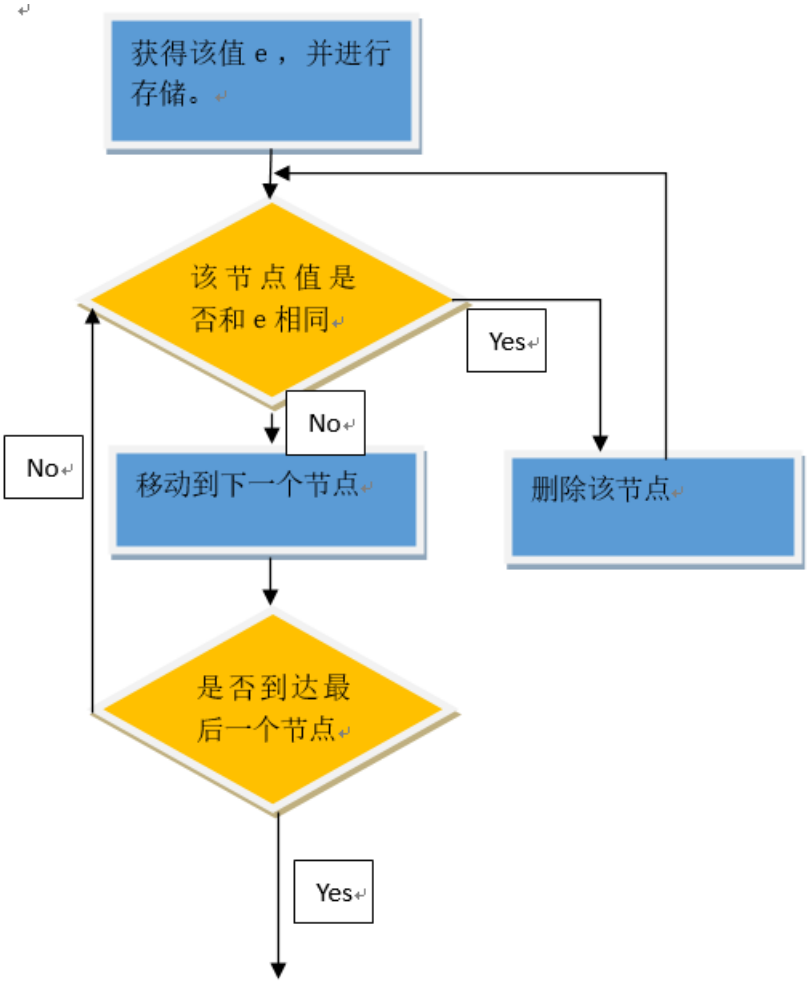
```
现在线性表里面有这么多个元素：
3
```

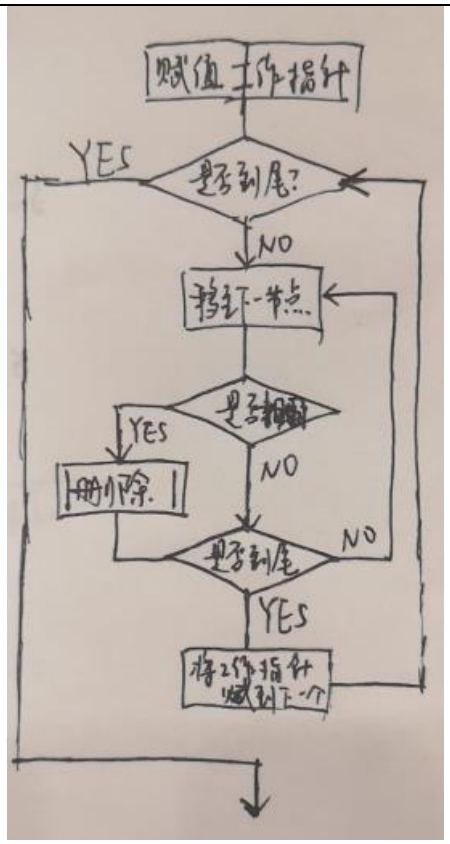
8．测试 SelectByLoc 函数，返回 index = 1 的节点的信息。

```
index = 1的人是：
1650800 张一 65.19
```

9．测试 selectByName 函数，看看李佶芮的节点在不在。

```
李佶芮在不在？
-1
```

心得体会	（对整个实验过程做出总结，对重要的算法做出性能分析。） 1．删除所有值为 e 的算法：  2．删除所有值为 e 的算法的时间复杂度为：O(n) 3．去重操作的时算法：



4 . 去重算法的时间复杂度：

由于设计两个 for 循环，所以复杂度为 $O(n^2)$。