

《数据结构》上机报告

2020 年 10 月 1 日

姓名：马家昱 学号：1950509 班级：王晓国 得分：300

实验题目	顺序表	
问题描述	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。	
基本要求	1. (p1)实现顺序表的基本操作，包括顺序表的初始化、第 i 个元素前插入一个新的元素、删除第 i 个元素、查找某元素、顺序表的销毁。 2. (p2)实现对有序表（非递减）插入元素、删除元素、查找元素的功能 3. (p3)实现两个有序（非递减）表合并生成新的有序表的功能 4. (p4)顺序查找顺序表，删除元素 e（删除所有值为 e 的元素）。 5. (p5)删除顺序表中所有多余的元素（即没有相同的元素）。	
	已完成基本内容（序号）：	1, 2, 3, 4, 5
选做要求	1. 程序要添加适当的注释，程序的书写要采用缩进格式。 2. 程序要具有一定的健壮性，即当输入数据非法时，程序也能适当地做出反应，如 插入删除时指定的位置不对 等等。	
	已完成选做内容（序号）	1, 2
数据结构设计	本实验三道题目均采用顺序表的数据结构。其中，第一道题目 学生信息管理 从底层搭建了静态数组的基本结构，并附加了增加长度的功能，其数据元素为包含两个数据成员的学生类。第二与第三道题目 去重与删除 均仅使用的静态数组。在顺序表中，数据元素的逻辑结构与物理结构均相邻。	

1. 学生信息管理

学生类 Student: 构造函数使学生初始信息为空

init: 对学生信息进行输入

```
class Student {
public:
    char ID[20]; //学号
    char name[20]; //姓名
public:
    Student(); //初始化
    void init(); //对学生信息进行输入
};

Student::Student() {
    ID[0] = '\0';
    name[0] = '\0';
}

void Student::init() {
    cin >> this->ID >> this->name;
}
```

顺序表的初始化 Sqliist 给顺序表分配空间, 将长度置 0 并将容量设定为初始容量。

```
//初始化构造函数
Sqliist::Sqliist() {
    this->elem = (Student*)malloc(LIST_INIT_SIZE * sizeof(Student));
    this->length = 0;
    this->listSize = LIST_INIT_SIZE;
}
```

Insert 函数: 将一个学生类插入到顺序表中指定的位置, 如果插入位置不合理, 则输出-1。

```
bool Sqliist::insert(int i, Student stu) {
    if (i < 1 || i > this->length + 1)
        return false;
    if (this->length >= this->listSize) {
        Student* newbase = (Student*)realloc(this->elem, (this->listSize + LIST_INCREMENT) * sizeof(Student));
        if (!newbase)
            return false;
        this->elem = newbase;
        this->listSize += LIST_INCREMENT;
    }
    Student* q = &(this->elem[i - 1]);
    if (this->length > 0) { //这里要判断长度是否为0, 否则会出现访问越界
        for (Student* p = &(this->elem[this->length - 1]); p >= q; --p)
            *(p + 1) = *p;
        *q = stu;
    }
    else *q = stu; //如果长度为0, 则直接赋值
    this->length++;
    return true;
}
```

Remove 函数: 删除顺序表指定位置的元素, 若位置不合理, 则输出-1

CheckName 函数: 按照字符串参数 搜索表中是否存在姓名与字符串一致的学生, 若存在, 则返回位置, 否则输出-1

CheckNo 函数: 按照字符串参数 搜索表中是否存在学号与字符串一致的学生, 若存在, 则返回位置, 否则输出-1

```

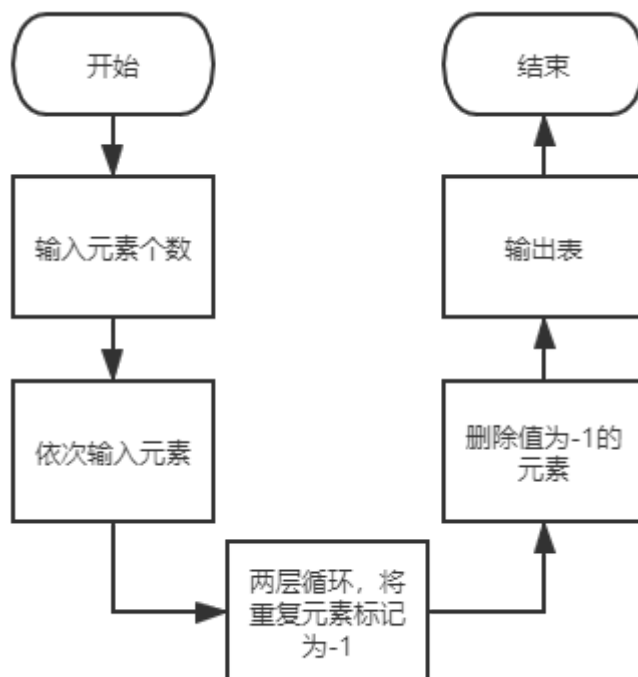
bool Sqlist::remove(int i) {
    if (i < 1 || i > this->length)
        return false;
    Student* p = &(this->elem[i - 1]);
    p++;
    if (this->length > 0) //判断长度是否为0, 否则会出现访问越界
        for (Student* q = this->elem + this->length - 1; p <= q; ++p)
            *(p - 1) = *p;
    else return false;
    this->length--;
    return true;
}

int Sqlist::checkName(char name[]) {
    for (int i = 0; i < this->length; i++)
        if (!strcmp(name, this->elem[i].name))
            return i + 1;
    return -1;
}

int Sqlist::checkNo(char id[]) {
    for (int i = 0; i < this->length; i++)
        if (!strcmp(id, this->elem[i].ID))
            return i + 1;
    return -1;
}

```

2. 顺序表去重 流程图:



其中, 标记与删除代码:

```

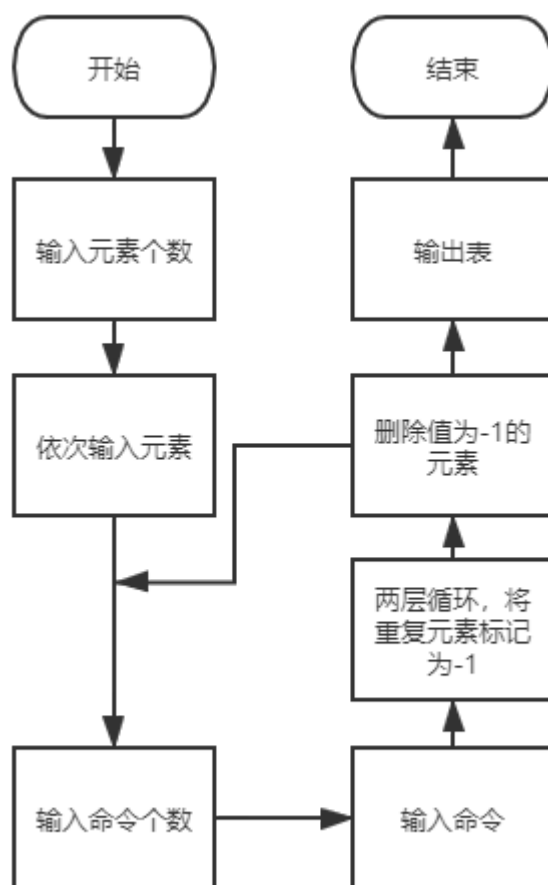
for (int i = 0; i < cnt; i++){ //去重
    if (a[i] == -1)
        continue;
    int t = a[i];
    for (int j = i + 1; j < cnt; j++){
        if (a[j] == t) {
            a[j] = -1; //将需要删除的数据替换为-1
        }
    }
}

int remove = 0; //每个-1后面的数据需要向前移动的位数
for (int i = 0; i < cnt; i++){
    if (a[i] == -1) {
        remove++;
        continue;
    }
    if (remove == 0)
        continue;
    a[i - remove] = a[i]; //遇到需要移动的数据时 向前移动remove位
}
cnt -= remove; //改变顺序表长度

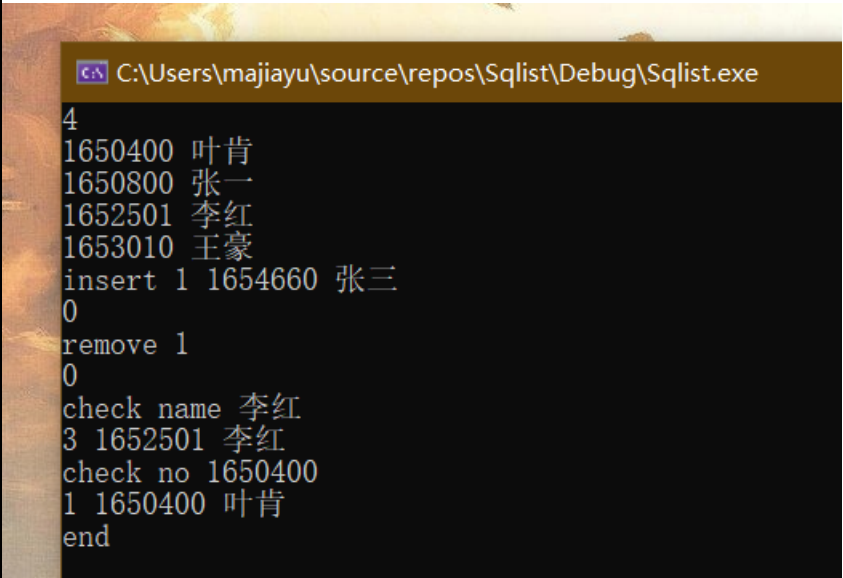
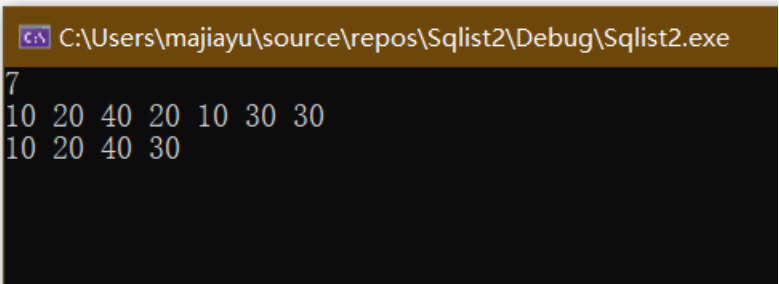
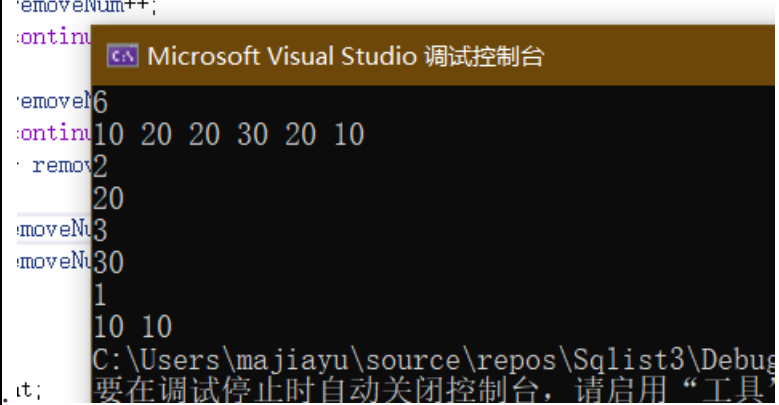
```

3. 顺序表删除

流程图：



标记与删除部分算法与去重代码相同。

开发环境	Windows10 编译与调试: visual studio 2019
调试分析	<p>(运行结果截图)</p> <p>1.</p>  <pre>C:\Users\majiayu\source\repos\Sqllist\Debug\Sqllist.exe 4 1650400 叶肯 1650800 张一 1652501 李红 1653010 王豪 insert 1 1654660 张三 0 remove 1 0 check name 李红 3 1652501 李红 check no 1650400 1 1650400 叶肯 end</pre> <p>2.</p> <p>数据</p>  <pre>C:\Users\majiayu\source\repos\Sqllist2\Debug\Sqllist2.exe 7 10 20 40 20 10 30 30 10 20 40 30</pre> <p>3</p>  <pre>removeNum++; :ontinu Microsoft Visual Studio 调试控制台 removeNum :ontinu removeNum 20 moveNum 3 moveNum 30 1 10 10 C:\Users\majiayu\source\repos\Sqllist3\Debug it;</pre> <p>三道题目均已完成题目要求并通过 oj 测试。</p>

心得体会	<p>(对整个实验过程做出总结，对重要的算法做出性能分析。)</p> <p>第一题中，对顺序表的插入与删除操作，平均要移动表中一半的元素，时间复杂度为 $O(n)$。查找操作为遍历一遍整个顺序表，时间复杂度为 $O(n)$。</p> <p>第二题中，去重时，标记需要删除的元素需要嵌套两层循环，时间复杂度为 $O(n^2)$，删除元素实际上是将后面的元素与需要删除的元素调换位置并且改变表的长度，时间复杂度为 $O(n)$。</p> <p>第三题中，标记需要删除的元素只需遍历一遍整个表，时间复杂度为 $O(n)$，删除操作与去重相同，时间复杂度也为 $O(n)$。</p> <p>通过本次实验，我最大的收获就是不能照搬课本上的内容。例如课本上关于顺序表的插入与删除操作中，忽略了线性表长度为 0 的情况，使得可能访问到 <code>L.elem[-1]</code>，即数组访问越界，这会使 oj 产生 <code>run time error</code> 的错误。而且通过上机使我更加注重边界数据，在写代码时要时刻注意这些不容易注意到的细节。</p> <p>其次是让我对顺序表有了更深一层的理解，理解了顺序表的优缺点。顺序表在查找与获取数据元素时非常方便，速度很快，并且能通过某个元素查找到其余所有元素。但是在进行插入与删除操作时，需要移动其他数据，造成了效率低下。</p>
------	--