

面向对象程序设计基础作业六 设计文档

1. 模型部分

a. 功能简述

本程序实现了一个学生学号与成绩管理系统，并使用双向链表储存数据，便于双向遍历访问数据。此管理系统支持用户添加任意数量（不超过系统内存）的学号、成绩信息（学号限制为 long long 范围的整数，成绩限制为 unsigned int 范围的非负整数），删除或访问第一个特定学号/全部特定分数的档案，或输出全部的档案信息。

本程序中包括的 double_list 类和 node 类稍加修改即可用于其他类型数据的存储。

b. 数据结构

本程序包含 student_system 类，可以实例化为成绩管理系统。此类调用 double_list 类（即双向链表）作为数据存储结构。double_list 类调用了 node 类，作为链表的节点，又设置了 student 类作为最终的学生数据结构（便于传参）。

Double_list 类提供基本的双向链表创建、添加节点、删除节点等功能，而且可扩展性好。和普通双向链表不同的是，本类中的双向链表不使用单一的头指针表示链表，而是使用“哨兵”机制：即，本类有一个成员变量 sentinel，此变量为一个虚拟的节点，其后继为真实的表头，其前驱为真实的表尾。遍历时，如访问到哨兵，就认为是达到了尽头。这样做的好处是，在添加/删除节点时不需要考虑是否处于头/尾位置，大大减少了代码工作量、增加了可读性。

c. 算法

本程序主要使用模拟法。

当用户希望添加节点时，本程序会从表头（即 `sentinel.next`）开始向后搜索，直到搜索到第一个比待加入节点学号大的节点、或学号相同但分数更高的节点、或 `sentinel` 为止，然后将新节点插入于此节点之前。插入的过程是：待插入节点的后继指针指向插入位置；前驱指针指向插入位置的前驱；待插入节点前驱的后继指针指向待插入节点；待插入节点后继的前驱指针指向待插入节点。由于引入了 `sentinel` 机制，无须判断是否处于头指针位置/链表是否为空。

当用户希望删除某特定学号/成绩的节点时，本程序会从表头开始向后搜索，直到搜索到足够数量的符合要求的节点为止。删除节点的过程是：待删除节点前驱的后继指针指向待删除节点的后继；待删除节点后继的前驱指针指向待删除节点的前驱；释放待删除节点。

当程序结束、需要释放所有空间时，本程序使用了 `double_list` 类的析构函数以实现自动回收内存。具体来说，此析构函数会逐一访问每个节点，记录下其后继，并释放它们的空间。

特别地，本程序中遵循安全指针的地方有：本程序使用了 `std::nothrow` 进行申请内存，假如申请失败会将指针置空，不会尝试访问未分配内存。

2. 验证部分

本程序的验证使用了五组不同的数据，代表不同的情形。在下述每个情形处会具体说明。

a. 列表是空表的情形

```
/Users/casorazitor/CLionProjects/HW6/cmake-build-debug/HW6
Input code: 2
Input an id: 2021
No student with ID 2021
Input code: 3
Input a score, data with which score are subject to removal: 100
No student with score 100
Input code: 4
Input an ID: 2021
No student with ID 2021
Input code: 5
Input a score, data with which score are subject to demonstration: 100
Students with score 100 have ID NONE!
Input code: 6
No data!
Input code: -1
Process finished with exit code 0
```

开始执行后，不为此程序读入任何数据，直接进行各项功能的测试。结果表明，每一个功能中都正确输出了“无数据”的结果，没有报错/内存异常。

b. 列表中没有重复分数/学号的情形

```
Input code: 1
Input students' ID and score:
20211251 52
12643334 243
1236 2321
213136 13532
125 41632
215 32235
1216663 2152
0
Input code: 4
ID: 125, score: 41632
ID: 215, score: 32235
ID: 1236, score: 2321
ID: 213136, score: 13532
ID: 1216663, score: 2152
ID: 12643334, score: 243
ID: 20211251, score: 52
```

```
Input code: 2
ID: 125, score: 41632
ID: 215, score: 32235
ID: 1236, score: 2321
ID: 213136, score: 13532
ID: 1216663, score: 2152
ID: 12643334, score: 243
ID: 20211251, score: 52
Input code: 3
Input an id: 213136
Student deleted with ID 213136 and score 13532
Input code: 5
Input a score, data with which score are subject to removal: 2152
Student deleted with ID 1216663 and score 2152
Input code: 4
Input an ID: 12643334
Student with ID 12643334 has score 243
Input code: 5
Input a score, data with which score are subject to demonstration: 243
Students with score 243 have ID 12643334
Input code: 2
ID: 125, score: 41632
ID: 215, score: 32235
ID: 1236, score: 2321
ID: 12643334, score: 243
ID: 20211251, score: 52
Input code: 1
Process finished with exit code 0
```

尽管功能 1 输入时顺序是完全打乱的，但使用功能 6 可以看出，在内存中的各个节点都已经按照学号升序排列。随后，使用 2-5 各个功能均能正确删除/查询所要求的数据。本例中，共删除了 2 次数据，查询了 2 次数据。考虑到列表中没有重复分数、也没有重复学号，输入规模为 7，最终状态仅剩 5 个数据，这是正确的。

c. 列表中有重复分数的情形

```
Input code: 1
Input students' ID and score:
1 1
2 2
3 3
4 4
5 5
6 6
7 6
8 5
9 4
10 3
0
Input code: 4
ID: 1, score: 1
ID: 2, score: 2
ID: 3, score: 3
ID: 4, score: 4
ID: 5, score: 5
ID: 6, score: 6
ID: 7, score: 6
ID: 8, score: 5
ID: 9, score: 4
ID: 10, score: 3
```

```
Input code: 4
Input an ID: 5
Student with ID 5 has score 5
Input code: 5
Input a score, data with which score are subject to demonstration: 6
Students with score 6 have ID 6, 7
Input code: 2
Input an id: 5
Student deleted with ID 5 and score 5
Input code: 3
Input a score, data with which score are subject to removal: 4
Student deleted with ID 4 and score 4
Student deleted with ID 9 and score 4
Input code: 4
ID: 1, score: 1
ID: 2, score: 2
ID: 3, score: 3
ID: 6, score: 6
ID: 7, score: 6
ID: 8, score: 5
ID: 10, score: 3
```

使用功能 1 输入时，有四组数据（3-10，4-9，5-8，6-7）具有相同的分数。可以看出，程序仍然按照学号升序构造了对应的链表。在使用功能 5 查询时，输入分数 6，程序给出了全部的两个具有分数 6 的学号，并使用逗号分隔。使用功能 3 删除时，输入分数 4，程序删除了全部的两个具有分数 4 的数据。加上使用功能 5 删除的一个数据，最终正确地剩余了 7 个数据点。

d. 列表中有重复学号的情形

```
Input students' ID and score:
```

```
1 124  
1 9  
1 5  
1 20  
2 34  
2 53  
5 4  
4 4  
4 7  
5 6  
0
```

```
Input code: 6
```

```
ID: 1, score: 5  
ID: 1, score: 9  
ID: 1, score: 20  
ID: 1, score: 124  
ID: 2, score: 34  
ID: 2, score: 53  
ID: 4, score: 4  
ID: 4, score: 7  
ID: 5, score: 4  
ID: 5, score: 6
```

```
Input code: 2
```

```
Input an id: 1
```

```
Student deleted with ID 1 and score 5
```

```
Input code: 6
```

```
ID: 1, score: 9  
ID: 1, score: 20  
ID: 1, score: 124  
ID: 2, score: 34  
ID: 2, score: 53  
ID: 4, score: 4  
ID: 4, score: 7  
ID: 5, score: 4  
ID: 5, score: 6
```

```
Input code: 4
```

```
Input an ID: 1
```

```
Student with ID 1 has score 9
```

```
Input a score, data with which score are subject to demonstration: 4  
Students with score 4 have ID 4, 5
```

使用功能 1 添加数据时，输入了若干组分数不同、学号相同的数据。使用功能 6 显示，链表确实是按照分数升序排列的。使用功能 2 进行删除，也只会删除此序列里第一个此学号的数据。使用功能 4 进行查询，只会显示此序列里第一个此学号的数据。而使用功能 5（查询所有某成绩的学号）可以正常输出所有有相同成绩的学号。

e. 列表中多次添加新数据的情形

```
Input code: 1
Input students' ID and score:
1235 16
3246 457
4357 45
4356 7
43572 4236
326 54
0
Input code: 6
ID: 326, score: 54
ID: 1235, score: 16
ID: 3246, score: 457
ID: 4356, score: 7
ID: 4357, score: 45
ID: 43572, score: 4236
Input code: 1
Input students' ID and score:
132632 3644
235136 43
0
Input code: 6
ID: 326, score: 54
ID: 1235, score: 16
ID: 3246, score: 457
ID: 4356, score: 7
ID: 4357, score: 45
ID: 43572, score: 4236
ID: 132632, score: 3644
ID: 235136, score: 43
```

```
Input code: 2
Input a score, data with which score are subject to removal: 45
Student deleted with ID 4357 and score 45
Input code: 4
Input an ID: 4356
Student with ID 4356 has score 7
Input code: 1
Input students' ID and score:
1235 457
0
Input code: 6
ID: 326, score: 54
ID: 1235, score: 16
ID: 3246, score: 457
ID: 4356, score: 7
ID: 4356, score: 12
ID: 21351, score: 457
ID: 43572, score: 4236
ID: 132632, score: 3644
ID: 235136, score: 43
Input code: 5
Input a score, data with which score are subject to demonstration: 457
Students with score 457 have ID 3246, 21351
```

此处测试了连续两次调用/相隔若干操作后再次调用功能 1 的场景。可以从功能 6 的结果看到，多次调用添加数据功能后，链表内的元素仍然按照上面提到的顺序排列。同时，其余几项功能也正确运行。