**菜单驱动的学生信息管理系统**

1. **设计目的**

掌握函数、数组、指针、结构体数组、结构体指针的知识及应用。

1. **需求分析**

学生数据中，学号应当是惟一的，作为学生的主要标志。

本程序需要能操作文件，包括读取文件和写入文件。

我按照一般软件的使用模式，入口界面需要提供新建项目、打开已有项目、退出程序的功能。其中新建和打开都应再进入一个选择界面，新建则直接进入项目，打开则输入文件名打开该文件。

选择功能应有输入多个学生数据并排序存储、插入学生数据、查找编辑、输出全部学生信息、保存、退出项目的功能，其中保存应询问路径。退出应该在有更改时询问是否保存。我没有提供单独的手动排序功能，因为可以做到在程序运行的每一步中保持顺序性，可以方便很多操作，也相对安全，在要求排序的操作前不需手动排序。

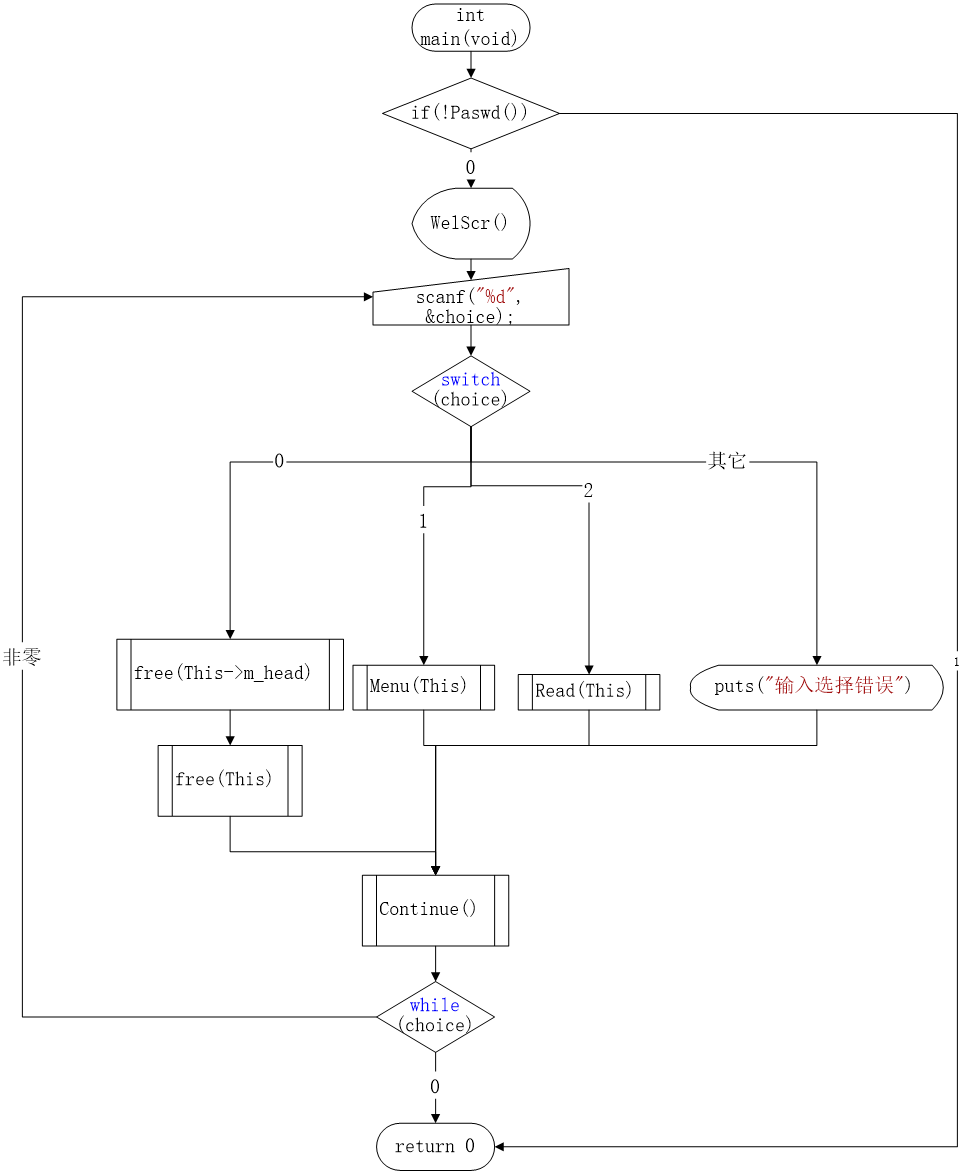
查找编辑应该单独提供分选项，下属：删除此学生信息、更改此学生信息、退出编辑的功能。

1. **概要设计**

说明用到的数据结构、主程序的流程、各函数的接口定义及调用关系。 并给出说明，用文字或框图均可；

数据结构使用了链表。由链表数据域、链表节点和学生信息链表三个结构体构成。

主函数流程图：



四个层次的函数：

第一层：

//主函数

int main(void)

第二层：（都是主函数直接调用的函数）

//口令验证，参数为设定的密码，返回是否通过验证

int Pswd(const char \* pswd)

//欢迎界面

void WelScr()

//选择项目界面

void Project()

//项目内部菜单，参数为链表指针，成功运行则返回0

int Menu(List \* This)

//读取文件并打开项目内部菜单，参数为链表指针，返回是否打开成功

int Read(List \* This)

//保存文件，参数为链表指针，返回是否保存成功

int Save(List \* This)

//选择功能完成后的切换

void Continue()

第三层：（基本都是第二层直接调用的函数，为了将相似的函数放在一起有所微调）

//保留链表结构，删除链表全部链接数据，清空链表数据，参数为链表指针

void DelAll(List \* This)

//表头

void Header()

//输入全部学生的数据，成功则返回1

int In(List \* This, FILE \* fp)

//输出全部学生的数据，返回是否有输出数据

int Out(List \* This, FILE \* fp)

//输入一个学生的数据，参数为文件指针，学生信息指针，返回 fscanf()返回值。

int InLine(FILE \*fp, Info \* p)

//输出一个学生的数据，参数为文件指针，学生信息指针，返回fprintf()返回值

int OutLine(FILE \*fp, Info \* p)

//按顺序插入节点，参数为学生信息链表指针和要插入的学生信息指针

LinkNode \* Insert(List \* This, Info \* pinf)

//搜索编辑的选择菜单

int SearchMenu(List \* This)

第四层：（第三层调用的函数，主要是int SearchMenu(List \* This)调用的）

//按学号进行搜索，参数为开始寻找的节点指针和学号，返回查找到的节点指针

LinkNode \* Search(LinkNode \* head, unsigned long stuID)

//删除单个节点，参数为链表指针和要删除的节点的指针

void Delete(List \* This, LinkNode \* p)

//更新学生信息，参数为学生信息链表指针和指向要更改的学生的指针的二级指针

void Update(List \* This, LinkNode \*\* pp)

二、三、四层间也有相互调用。

1. **详细设计**

数据结构的类型定义，每个函数的算法描述，关键算法的流程图等。

三个结构体：

typedef struct

{

unsigned long m\_stuID;//学号,唯一

char m\_Name[NAME\_SIZE];//姓名

char m\_Class[CLASS\_SIZE];//班号

char m\_Major[MAJOR\_SIZE];//专业

float m\_Gpa;//加权平均成绩

char m\_PhoNum[NUM\_SIZE];//电话号码

char m\_EMA[EMA\_SIZE];//电子邮件地址

}Info;//链表数据域

typedef struct LINKNODE

{

Info m\_data;//数据域

struct LINKNODE \*m\_next;//指针域

}LinkNode;//链表节点

typedef struct

{

LinkNode \*m\_head;//链表头结点

int m\_changed;//是否更改过信息

char m\_fname[fNAME\_SIZE];//关联的文件名

}List;//学生信息链表

构成由学生信息组成的链表及与链表有关的其它信息。

函数算法描述：

//口令验证，参数为设定的密码，返回是否通过验证

int Pswd(const char \* pswd)

{

int i = 3;//密码输入次数，共三次

char inp = 0;//临时存储输入的密码字符

puts("请输入密码，共有三次机会");

while (1)

{

int charWrong = 0;//字符对应出现错误，假设为无错

int sizeWrong = 1;//输入密码长度出现错误，假设为有错

int j = -1;//输入密码对应的数组下标

while (1)

{

inp = \_getch();//无回显式获取字符

j++;

if (inp == '\r')//若是回车符

{

if (pswd[j] == '\0')//若键入Enter时对应密码数组的结束符

sizeWrong = 0;//密码长度无错

break;

}

putchar('\*');//回显'\*'，用于输入反馈

if (inp != pswd[j])//若键入密码字符与密码数组对应元素不同

charWrong = 1;

}

putchar('\n');//密码输入结束，换行（上一个键入的换行符被\_getch()读了）

i--;

if (charWrong || sizeWrong)//若字符错误或长度错误

{

if (i >= 1)

printf("密码错误，还有%d次输入机会\n", i);//显示剩余次数

else

{

puts("输入密码3次错误！");

system("PAUSE");

break;

}

}

else//通过验证

{

puts("密码输入正确！");

Continue();

return 1;

}

}

return 0;//未通过验证

}

//项目内部菜单，参数为链表指针，成功运行则返回0

int Menu(List \* This)

{

int choice = 0;//选择数字

do

{

//省略显示界面. . .

scanf("%d", &choice);

system("cls");

Info inf;//临时存储学生信息

int save\_choice = 0;//是否保存更改选项

switch (choice)

{

case 0://退出项目

if (This->m\_changed)//若做过更改

{

puts("是否保存所做更改？（0：否，非0数字：是）");

scanf("%d", &save\_choice);

if (save\_choice)//要保存更改

Save(This);//保存

}

DelAll(This);//删除链表信息

return 0;

break;

case 1://输入全部学生数据并排序存储

puts("请输入全部学生信息,直到输入的电子邮件地址尾字符为'\\'为止（不读入'\\'）");

Header();//表头

In(This, stdin);//输入全部学生的数据

This->m\_changed = 1;//进行了更改

break;

case 2://插入学生数据

puts("请输入一个学生的信息。");

Header();

InLine(stdin, &inf);//输入

Insert(This, &inf);//插入

This->m\_changed = 1;//进行了更改

break;

case 3://查找编辑

SearchMenu(This);//进入搜索编辑界面

break;

case 4://输出全部学生信息

Header();

Out(This, stdout);

break;

case 5://保存

Save(This);

break;

default:

puts("输入的选择不正确！");

break;

}

Continue();

} while (choice);

return 0;

}

//读取文件并打开项目内部菜单，参数为链表指针，返回是否打开成功

int Read(List \* This)

{

puts("打开已有文件，请输入文件名：");

scanf("%s", This->m\_fname);

FILE \*fp = fopen(This->m\_fname, "r");//打开文件

if (!fp)

{

puts("Read打开文件错误！");

return 0;

}

if (In(This, fp) < 0)//若读取失败

{

puts("In读取文件错误！");

return 0;

}

fclose(fp);//关闭文件

puts("打开成功！");

Continue();

Menu(This);//进入选择功能界面

return 1;//打开成功

}

//保存文件，参数为链表指针，返回是否保存成功

int Save(List \* This)

{

if (!strlen(This->m\_fname))//若无文件名

{

puts("请输入要保存的文件名,若无此文件则新建");

scanf("%s", This->m\_fname);

}

FILE \*fp = fopen(This->m\_fname, "w");//打开文件

if (!fp)

{

puts("Save打开文件错误！");

return 0;

}

if (Out(This, fp) < 0)//使用Out将全部数据输出至文件

{

puts("Out保存文件错误！");

return 0;

}

fclose(fp);//关闭文件

puts("保存成功！");

This->m\_changed = 0;//是否更改的标识符清零

return 1;//打开成功

}

//选择功能完成后的切换

void Continue()

{

system("PAUSE");

system("cls");

}

//保留链表结构，删除链表全部链接数据，清空链表数据，参数为链表指针

void DelAll(List \* This)

{

while (This->m\_head->m\_next)//删除链表全部链接数据

Delete(This, This->m\_head->m\_next);

for (size\_t i = 0; i < fNAME\_SIZE; i++)//文件名置空

This->m\_fname[i] = 0;

This->m\_head->m\_next = NULL;//头结点指针置空

}

//表头

void Header()

{

printf("%-16s%-16s%-16s%-16s%-16s%-16s%-16s\n",

"学号",

"姓名",

"班号",

"专业",

"加权平均",

"电话号码",

"电子邮件地址");

}

//输入全部学生的数据，成功则返回1

int In(List \* This, FILE \* fp)

{

while (!feof(fp))

{

Info inf;//保存输入的节点

char \*p = NULL;

if (InLine(fp, &inf) == EOF)//若遇到文件末尾跳出循环

break;

if (p = strrchr(inf.m\_EMA, '\\'))//若输入的最后电子邮箱地址一个含'\\'

{

\*p = 0;

Insert(This, &inf);

break;

}

else//直到结束符为止一直输入

Insert(This, &inf);

}

return 1;

}

//输出全部学生的数据，返回是否有输出数据

int Out(List \* This, FILE \* fp)

{

if (!This->m\_head->m\_next)//表空

return 0;

LinkNode \*p = This->m\_head->m\_next;

while (p)//遍历输出

{

OutLine(fp, &p->m\_data);

p = p->m\_next;

}

return 1;

}

//输入一个学生的数据，参数为文件指针，学生信息指针，返回 fscanf()返回值。

int InLine(FILE \*fp, Info \* p)

{

return fscanf(fp, "%lu %s %s %s %f %s %s",

&p->m\_stuID,

p->m\_Name,

p->m\_Class,

p->m\_Major,

&p->m\_Gpa,

p->m\_PhoNum,

p->m\_EMA);

}

//输出一个学生的数据，参数为文件指针，学生信息指针，返回fprintf()返回值

int OutLine(FILE \*fp, Info \* p)

{

return fprintf(fp, "%-16lu%-16s%-16s%-16s%-16f%-16s%-16s\n",

p->m\_stuID,

p->m\_Name,

p->m\_Class,

p->m\_Major,

p->m\_Gpa,

p->m\_PhoNum,

p->m\_EMA);

}

//按顺序插入节点，参数为学生信息链表指针和要插入的学生信息指针

LinkNode \* Insert(List \* This, Info \* pinf)

{

if (!This->m\_head)//发生错误

{

puts("This->m\_head为nullptr！");

return NULL;

}

LinkNode \* prepeat = Search(This->m\_head, pinf->m\_stuID);//与插入的学号重复的节点的指针

if (prepeat)//重复

{

puts("发现重复");

puts("是保留原来的学生信息（按0）");

Header();

OutLine(stdout, &prepeat->m\_data);

puts("还是保留更新的学生信息（按非0数字键）");

Header();

OutLine(stdout, pinf);

int choice = 0;

scanf("%d", &choice);

if (choice)//保留更新的学生信息

prepeat->m\_data = \*pinf;

return prepeat;//退出此次插入

}

LinkNode \*p = (LinkNode \*)calloc(1, sizeof(LinkNode));//要插入的节点

if (!p)

{

puts("内存不足！");

return p;

}

p->m\_data = \*pinf;

LinkNode \*pr = This->m\_head;//当前遍历的指针

LinkNode \*temp = This->m\_head;//保存当前遍历的的节点的前面一个节点的指针

while (pr->m\_next&&pr->m\_data.m\_stuID < pinf->m\_stuID)//找到插入位置，保存此位置和前一位置的指针，或到尾节点跳出

{

temp = pr;

pr = pr->m\_next;

}

if (pr->m\_data.m\_stuID >= pinf->m\_stuID)//此条件作用是在尾节点跳出时判断，选择插入尾节点前还是尾节点后

{

pr = temp;

p->m\_next = pr->m\_next;

pr->m\_next = p;

}

else

pr->m\_next = p;

return p;

}

//搜索编辑的选择菜单

int SearchMenu(List \* This)

{

unsigned long stuID = 0;//查找的学号

int choice = 0;//选择数字

puts("请输入要查找学生的学号");

scanf("%lu", &stuID);

LinkNode \*sresult = Search(This->m\_head->m\_next, stuID);//查找

if (!sresult)

{

puts("未找到该学生！");

return 0;

}

do

{

//省略显示界面. . .

printf("%8c", ' ');//begin 搜索结果，与上面界面对齐

Header();

printf("%8c", ' ');

OutLine(stdout, &sresult->m\_data);//end

scanf("%d", &choice);

system("cls");

switch (choice)

{

case 0://退出编辑

return 1;

break;

case 1://删除此学生信息

Delete(This, sresult);

This->m\_changed = 1;//进行了更改

return 0;//删除后直接退出

break;

case 2://更改此学生信息

Update(This, &sresult);

This->m\_changed = 1;//进行了更改

break;

default:

puts("输入选择错误！");

break;

}

Continue();

} while (choice);

return 1;

}

//按学号进行搜索，参数为开始寻找的节点指针和学号，返回查找到的节点指针

LinkNode \* Search(LinkNode \* head, unsigned long stuID)

{

LinkNode \*p = head;//遍历用指针

if (!head)

return NULL;

while (stuID != p->m\_data.m\_stuID)//找到则退出循环

{

if (!p->m\_next)

return NULL;

p = p->m\_next;

}

return p;

}

//删除单个节点，参数为链表指针和要删除的节点的指针

void Delete(List \* This, LinkNode \* p)

{

LinkNode \*pr = This->m\_head;

while (pr->m\_next != p)

pr = pr->m\_next;

pr->m\_next = p->m\_next;

free(p);

}

//更新学生信息，参数为学生信息链表指针和指向要更改的学生的指针的二级指针

void Update(List \* This, LinkNode \*\* pp)

{

Info inf;//更改后的学生信息

LinkNode \*pr;//临时指针，用于删除原信息

puts("请输入一位学生的信息");

Header();

InLine(stdin, &inf);

if (inf.m\_stuID == (\*pp)->m\_data.m\_stuID)//若不更改学号

(\*pp)->m\_data = inf;//直接对现在信息修改

else//若学号更改，则重新插入，并更改搜索编辑的指针

{

pr = \*pp;

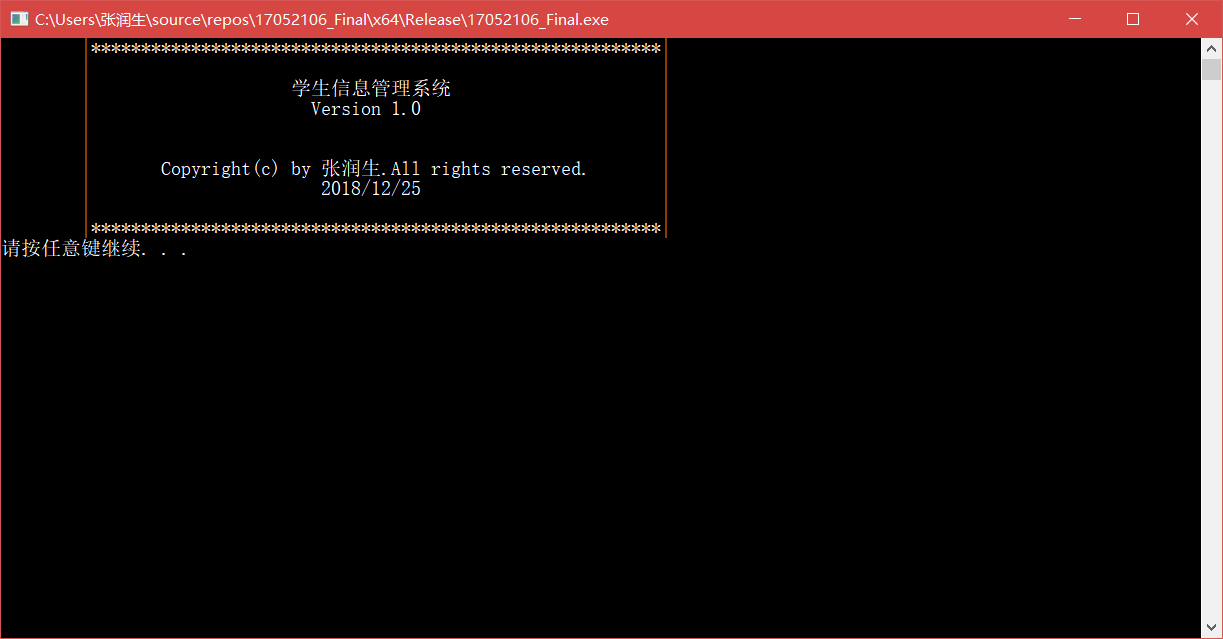
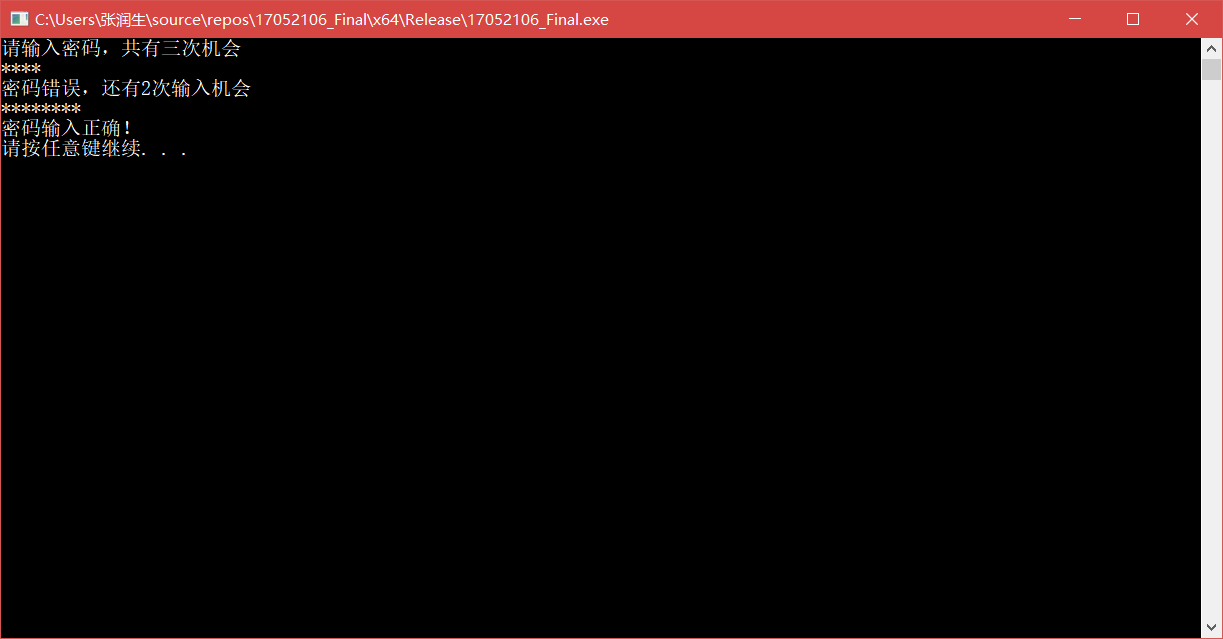
\*pp = Insert(This, &inf);

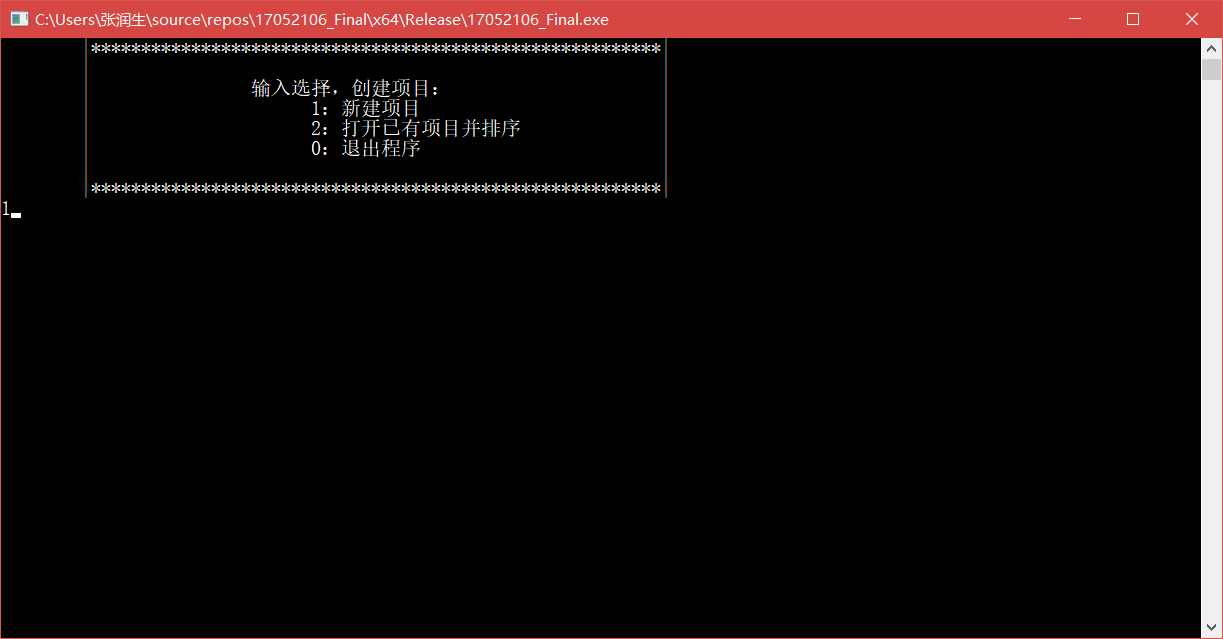
Delete(This, pr);

}

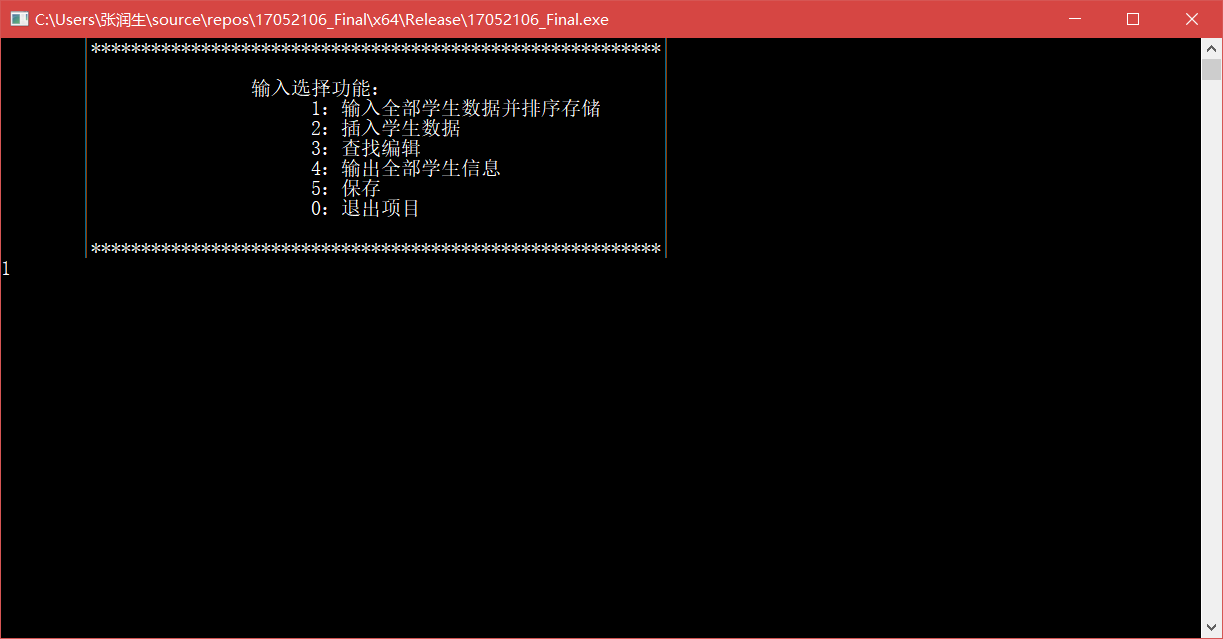
}

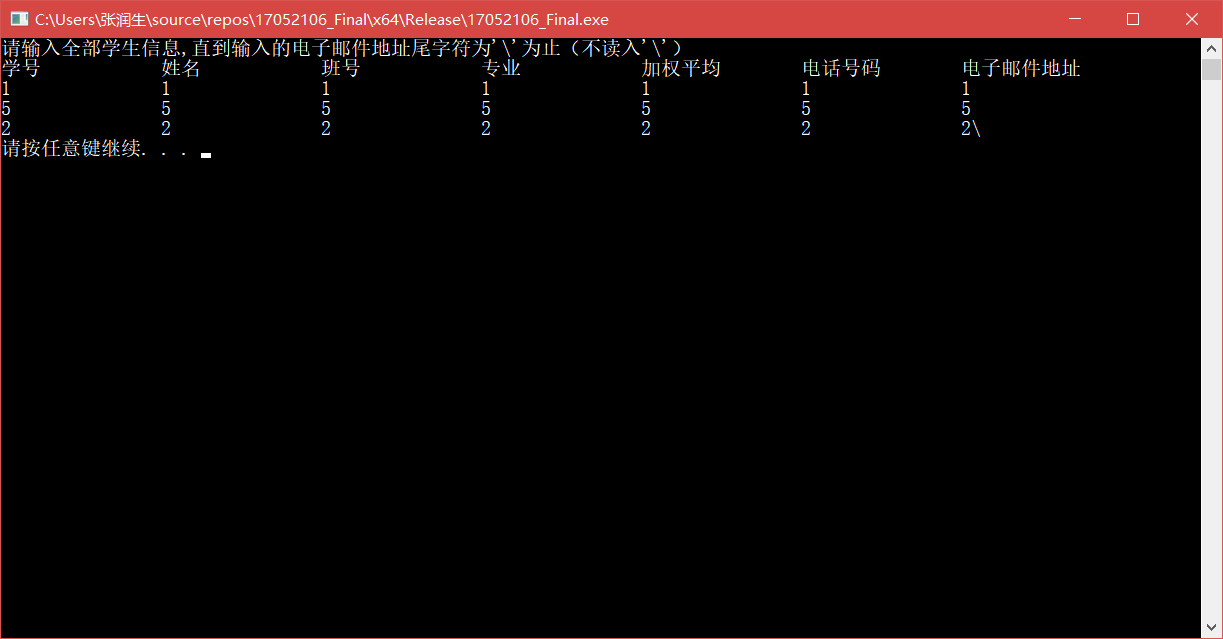
1. **程序的运行测试结果** (运行截图)



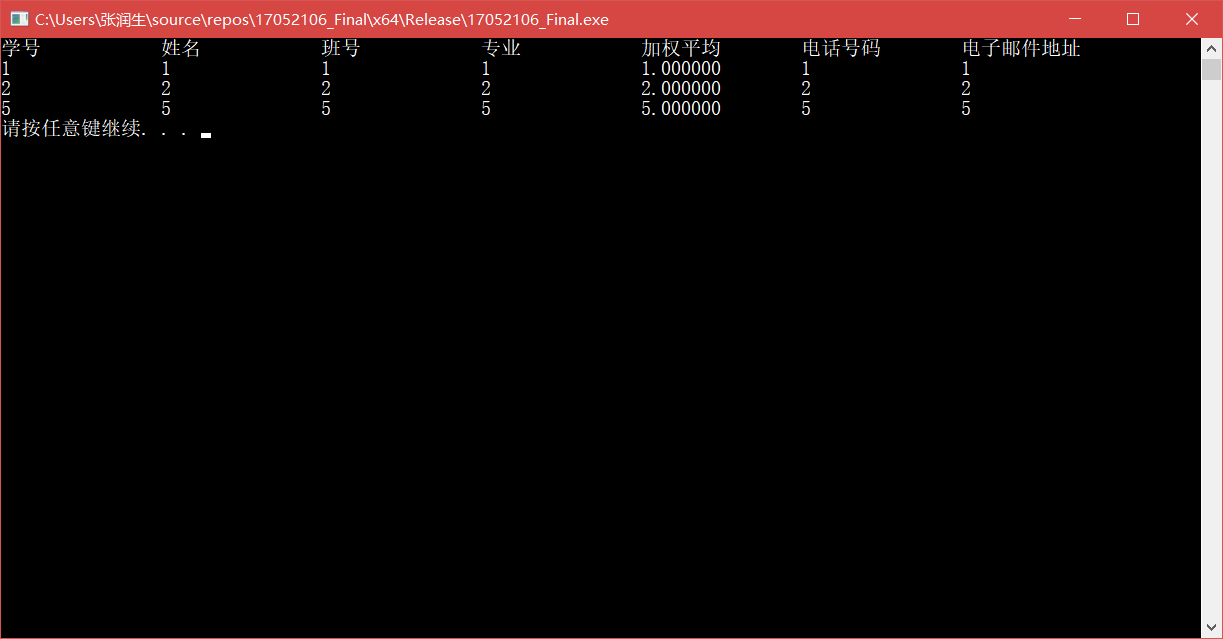


输入全部：

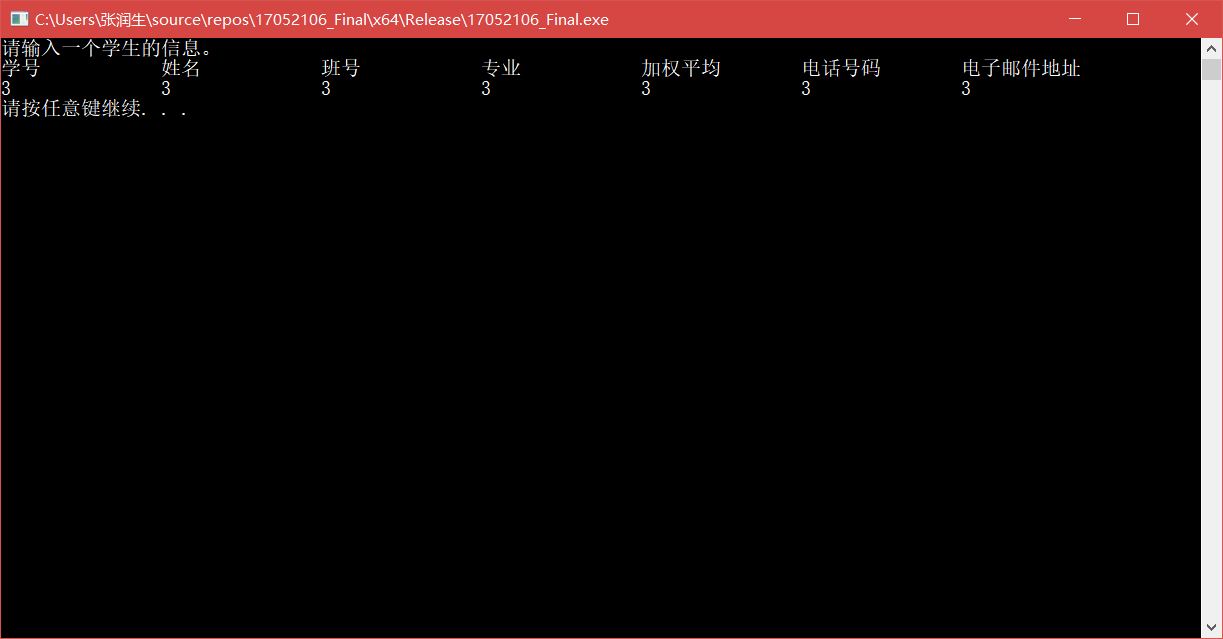




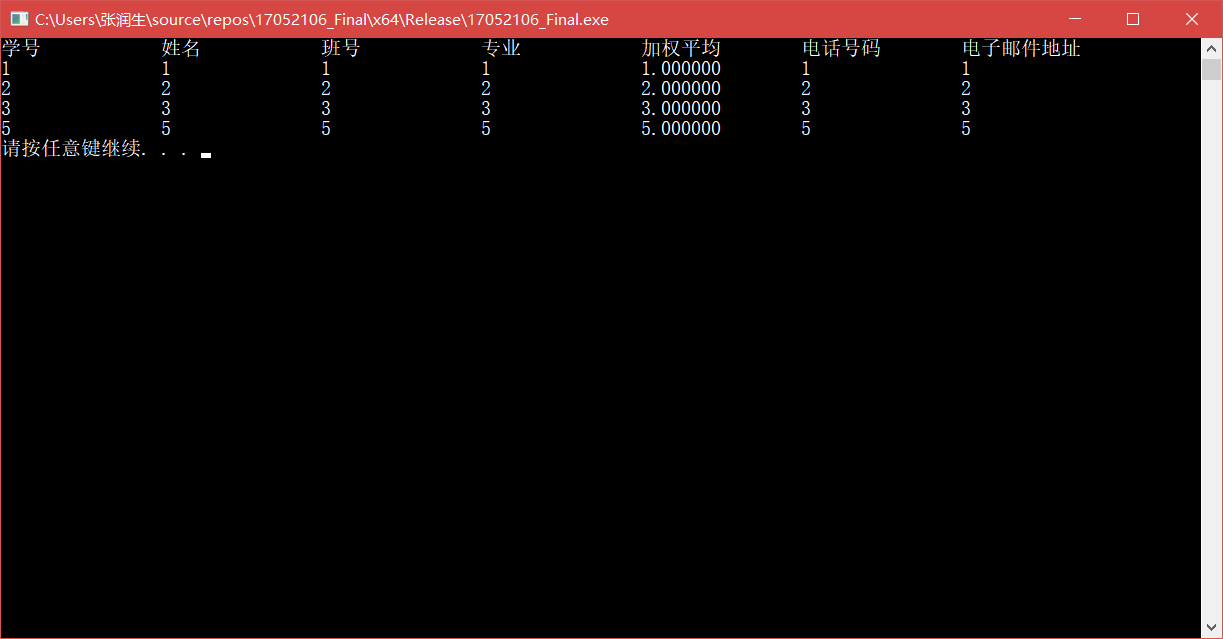
输出：



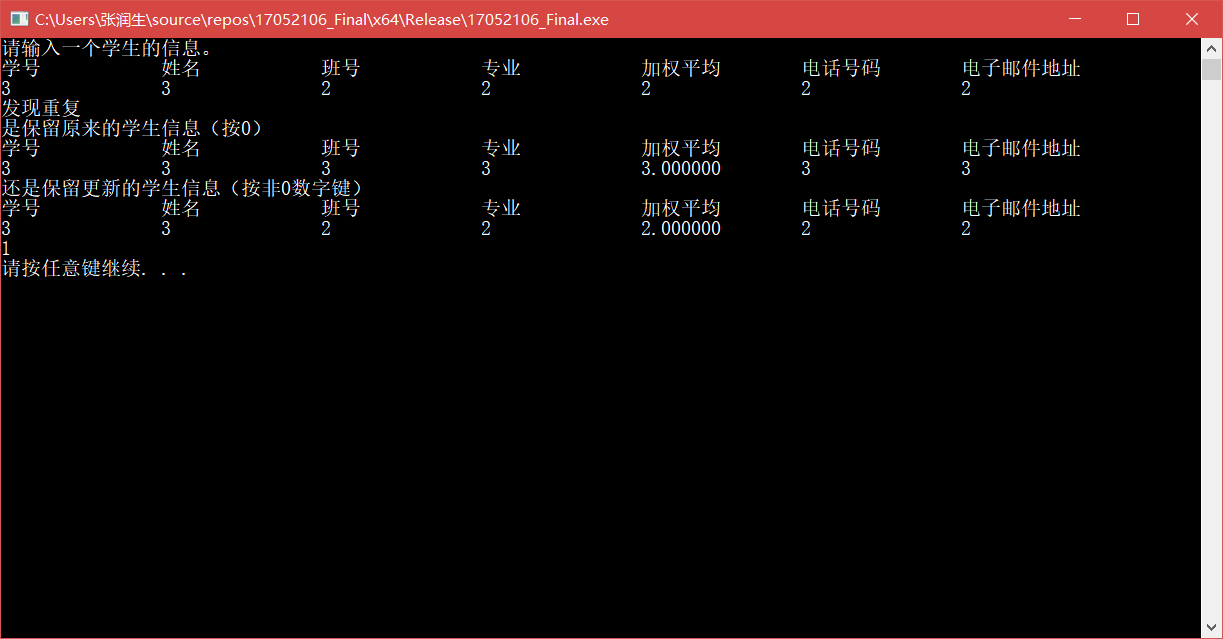
插入：



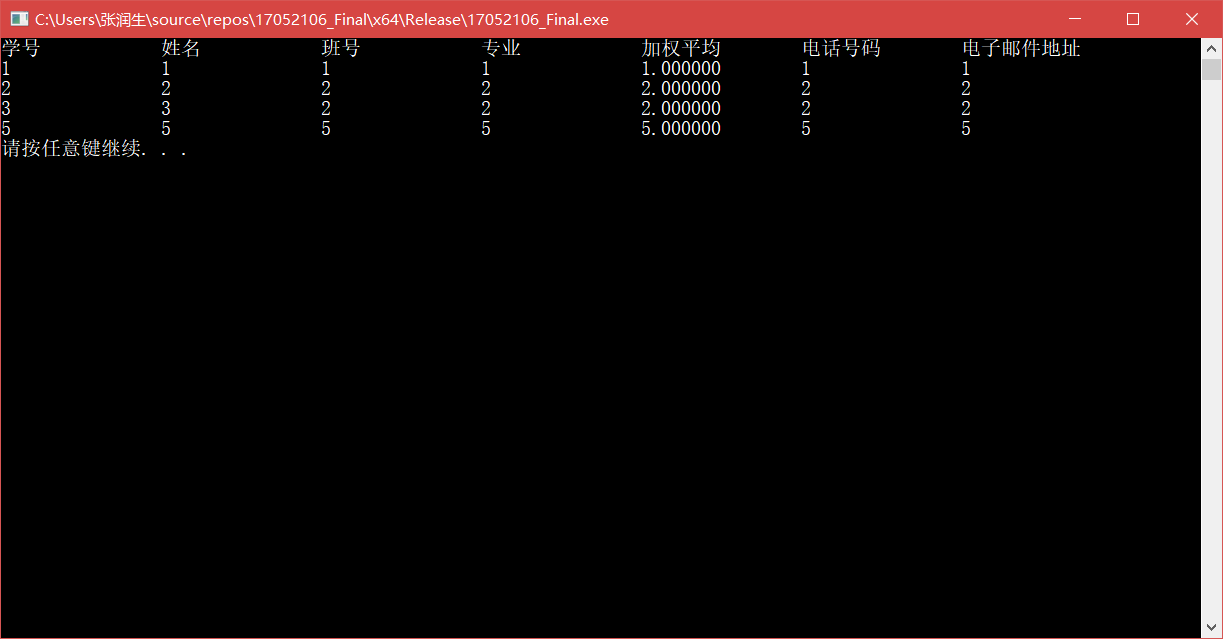
输出：



插入（若重复）

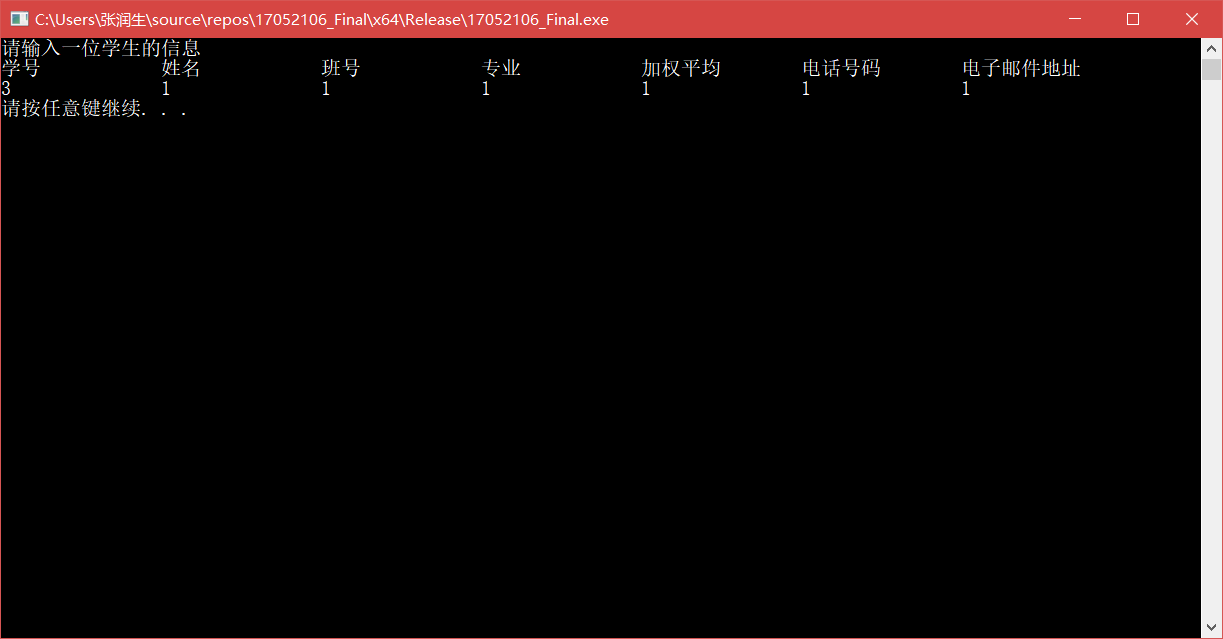


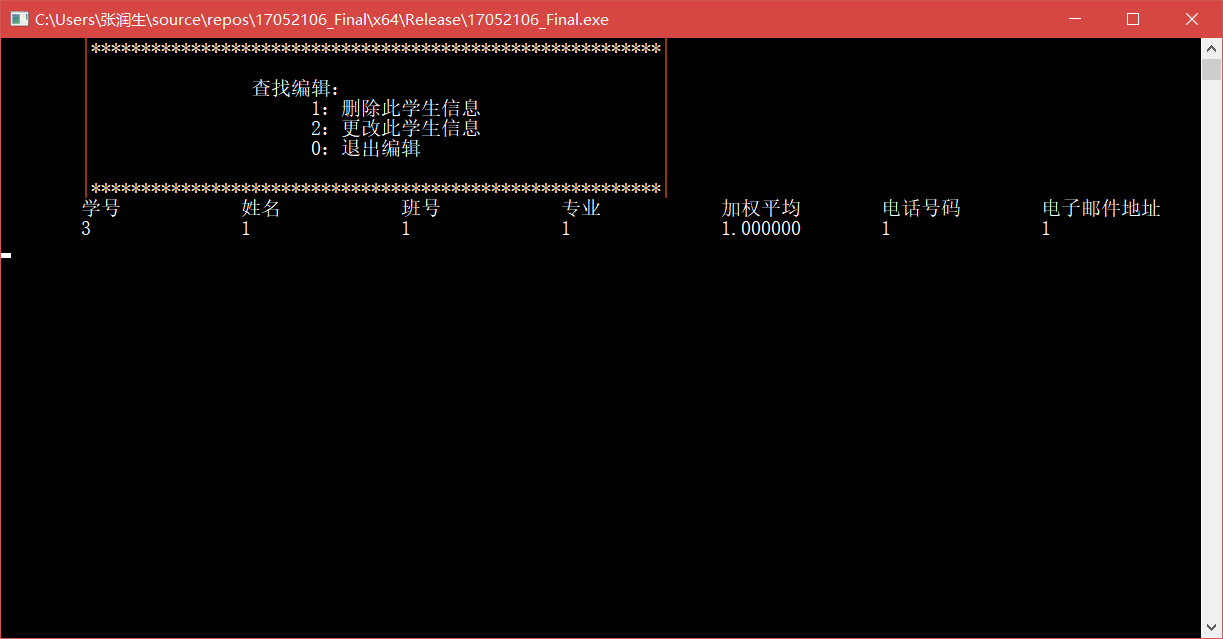
输出：



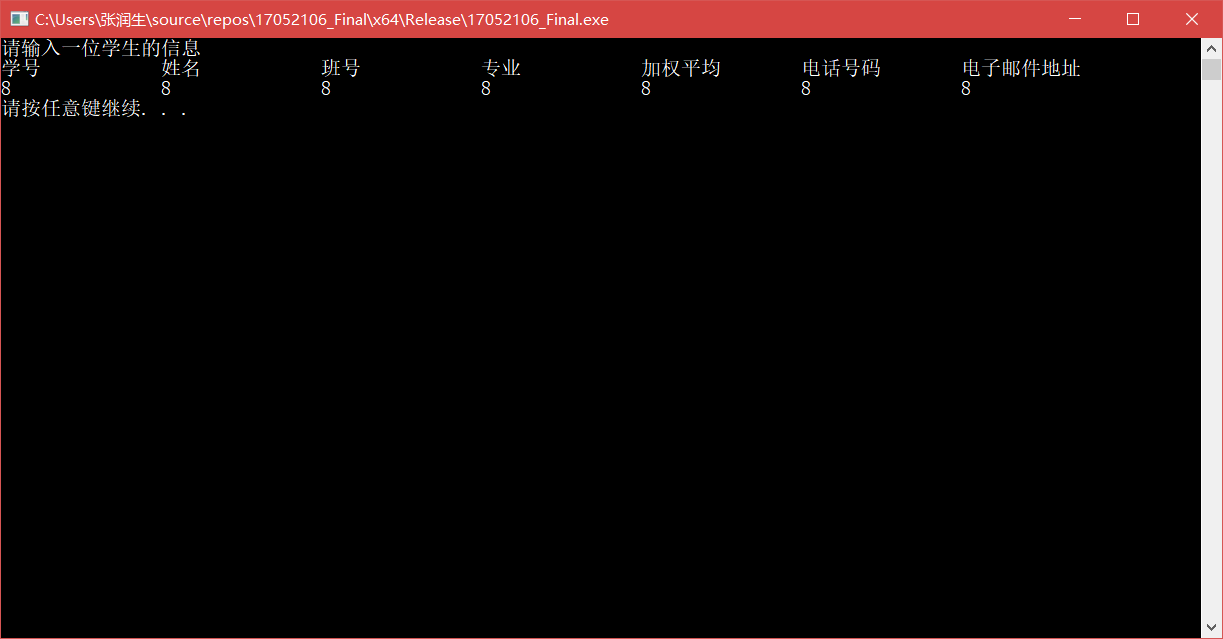
查找：

更改（学号不改）



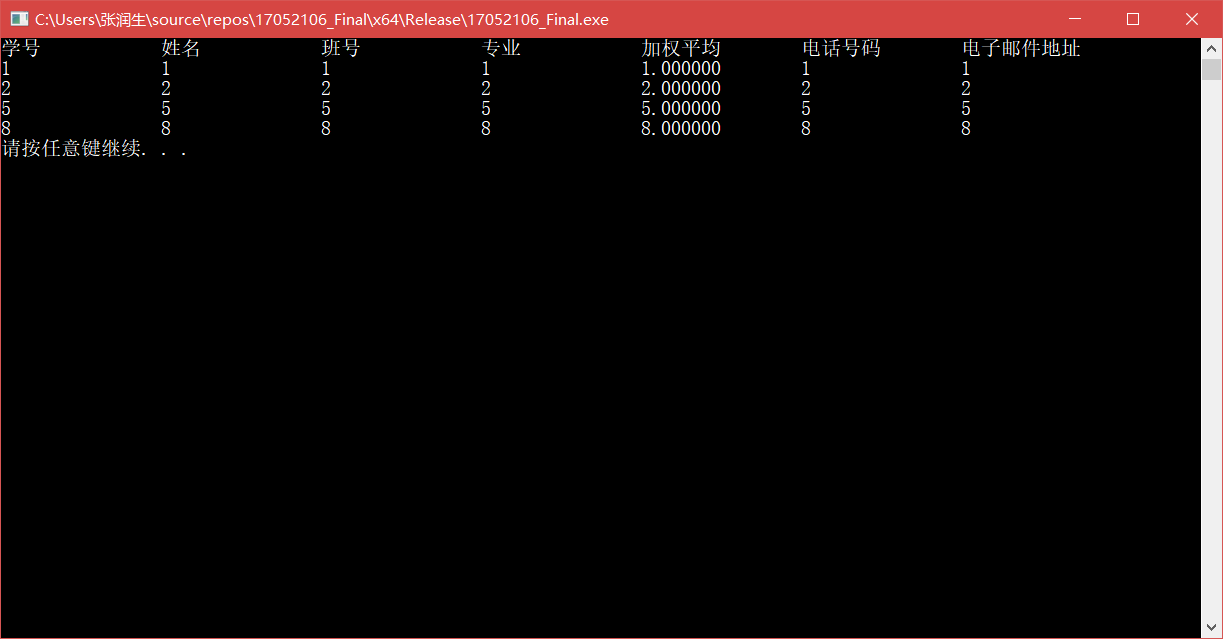


更改（学号改）（重新插入）

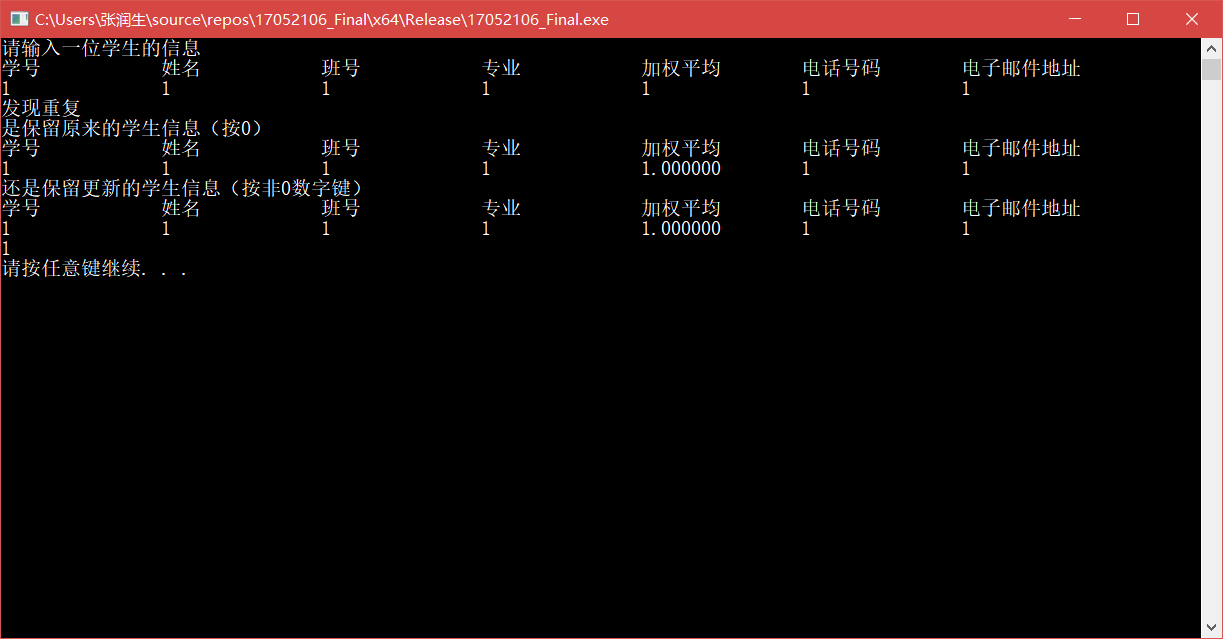




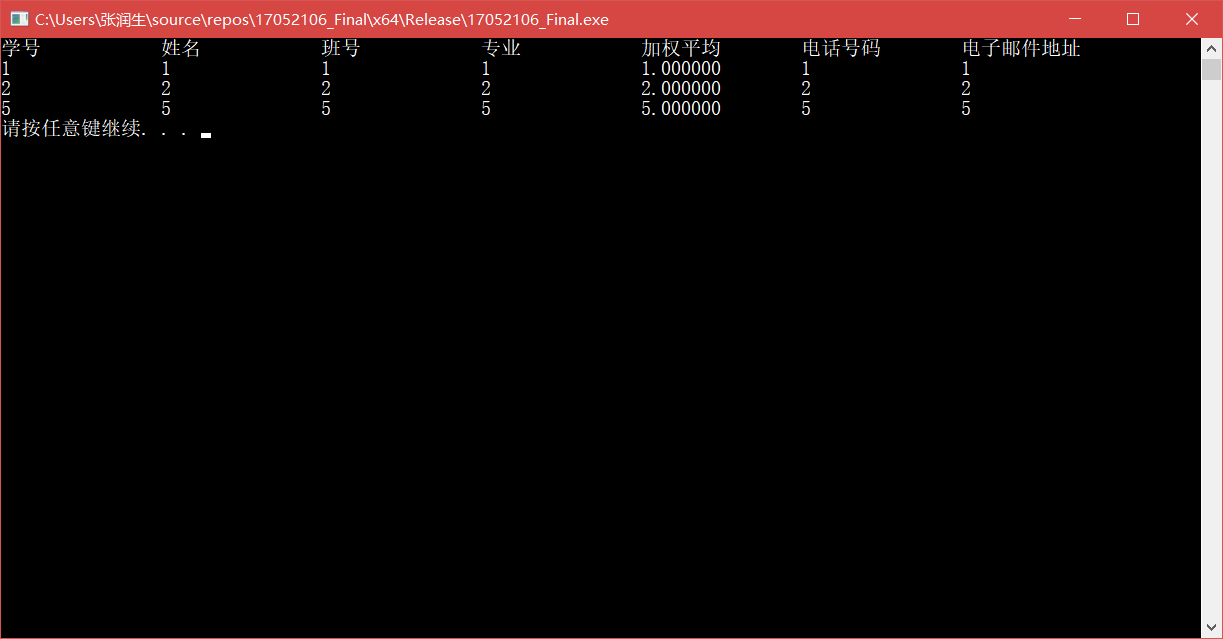
输出：



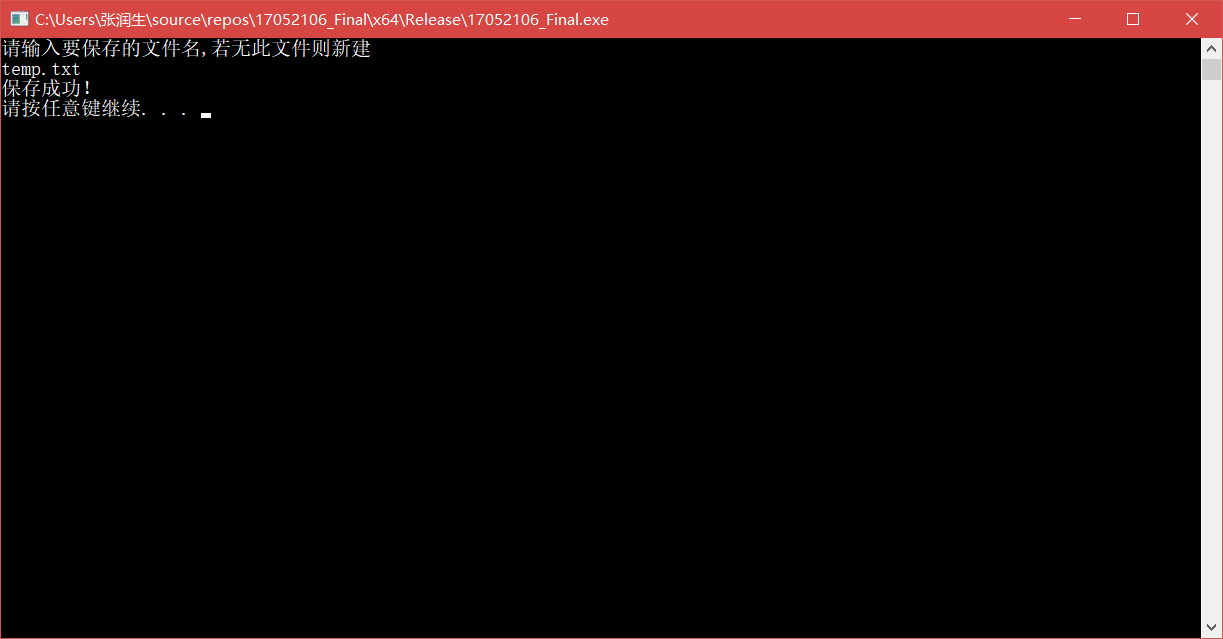
回到8的更改，改成表中已有学号：

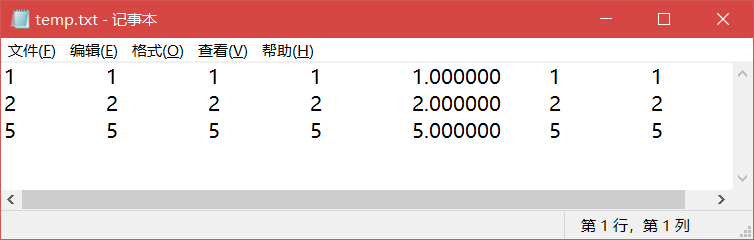


输出：

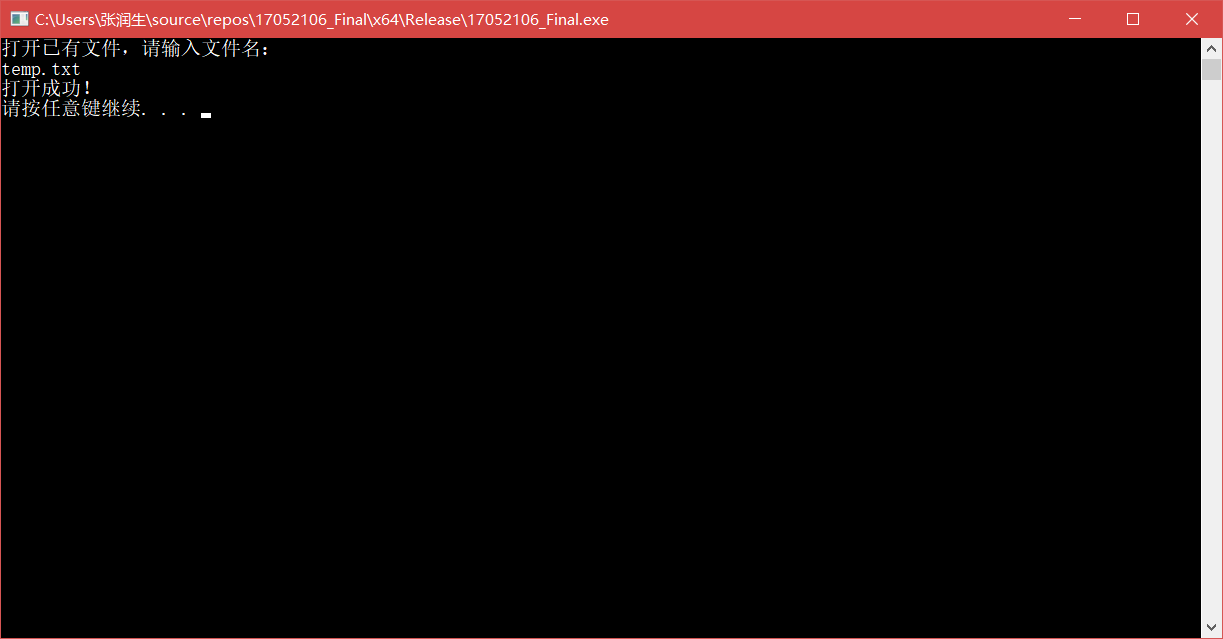


保存：

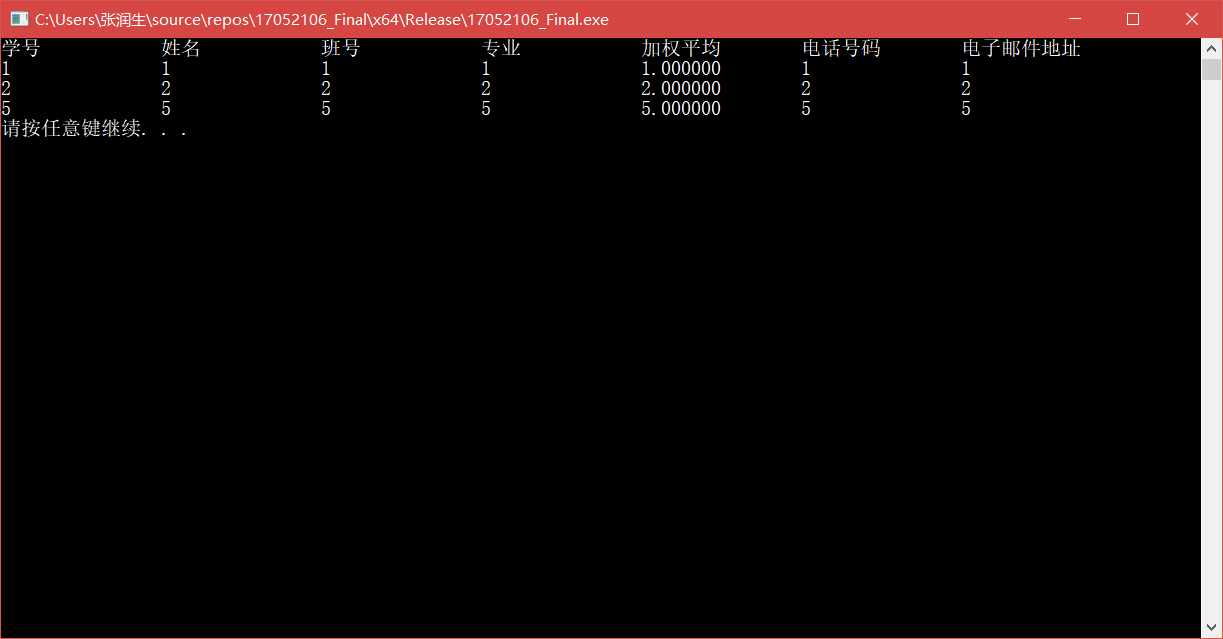




打开：



输出：



1. **遇到的问题、原因及解决方法**

关于排序，我最开始也是设想手动排序，但后来发现链表并不能用类似快速排序之类的排序，只能用时间复杂度较高的排序，因为本程序要求顺序插入，单独提供排序功能好像意义不大。所以我最终选择了直接调用顺序插入来在输入（从文件或从键盘）时自动排序，并在所有操作后都维持排序状态。

这样也避免了未排序就进行插入的可能。

最开始我的链表没有空的头结点。没有头结点的话，增删改查各种操作要考虑很多种情况，非常不方便，并且很容易出错，因为链表的第一个节点的位置是特殊的，而数据是一般的。加入头结点后，至少将所有在链表开头遇到的问题全部转化为和在链表中间一样了，只需单独考虑尾节点，而尾节点的数据是特殊的（含空指针），很容易操作。这样能够实现操作的统一。

更新个人信息后（改变学号）在更改界面不能正常显示，后来找到原因，是因为我传的参数是一级指针，更新后没有成功改变查找时查找到的指针，程序依然按照之前查找的指针显示内容。更改方法是我把参数换成了二级指针，这样就可以在函数里更改指针的指向了。

之前没加查重的功能，现在增改的操作都能够查重。

另外，我最初参考教材上的两段链表操作的代码，似乎全都是有问题的，整个调试的过程中，大部分Bug都出在教材的代码上，现在已经全部重写了。

1. **收获及体会：**

本次实验我了解并实践了链表的构造与操作，链表是一种递归的数据结构，直观上感觉它很有规律，但实际操作的时候有很多特殊情况出现，我觉得添加头结点是一种减少特殊情况的好办法，它让两个特殊位置节点的数据也都是特殊的，让链表变得更规律了。

同样的，我在程序中写了很多以文件指针为参数的函数，因为我想要尽量统一对标准输入输出和对一般文件的操作。这样可以让程序看上去更简洁，也能一定程度上减少代码量，同时也更有规律性。

我之前怎么也想不到，这个程序其实花费我最大量时间的，是在口令验证那里。看似简单的功能实现起来还是要动不少脑子，也要经过反复验证，才能比较好的实现。

**参考文献：**

[1] 苏小红 孙志岗 陈慧鹏 等编著. C语言大学实用教程（第4版）. 电子工业出版社，2004年8月第1版 2017年1月第4版