**问题描述**

学生成绩管理系统可以对学生表（学号、姓名、性别）、课程表（课程号、

课程名、学分）、选课表（学号、课程号、成绩）进行管理，学生可以选择某门课程，考试得到一个成绩。要求学生表使用学号作为关键字构建二叉排序树（学号不能重复），课程表（课程号不能重复）和选课表采用顺序结构。

1. **系统功能说明**

**2.1系统功能**

学生信息、课程信息预先设置好，程序启动时，从文件中读入学生、课程和

选课信息并自动建立数据结构，使用菜单实现以下功能（可以重复操作），每次操作后应该输出当前结果：

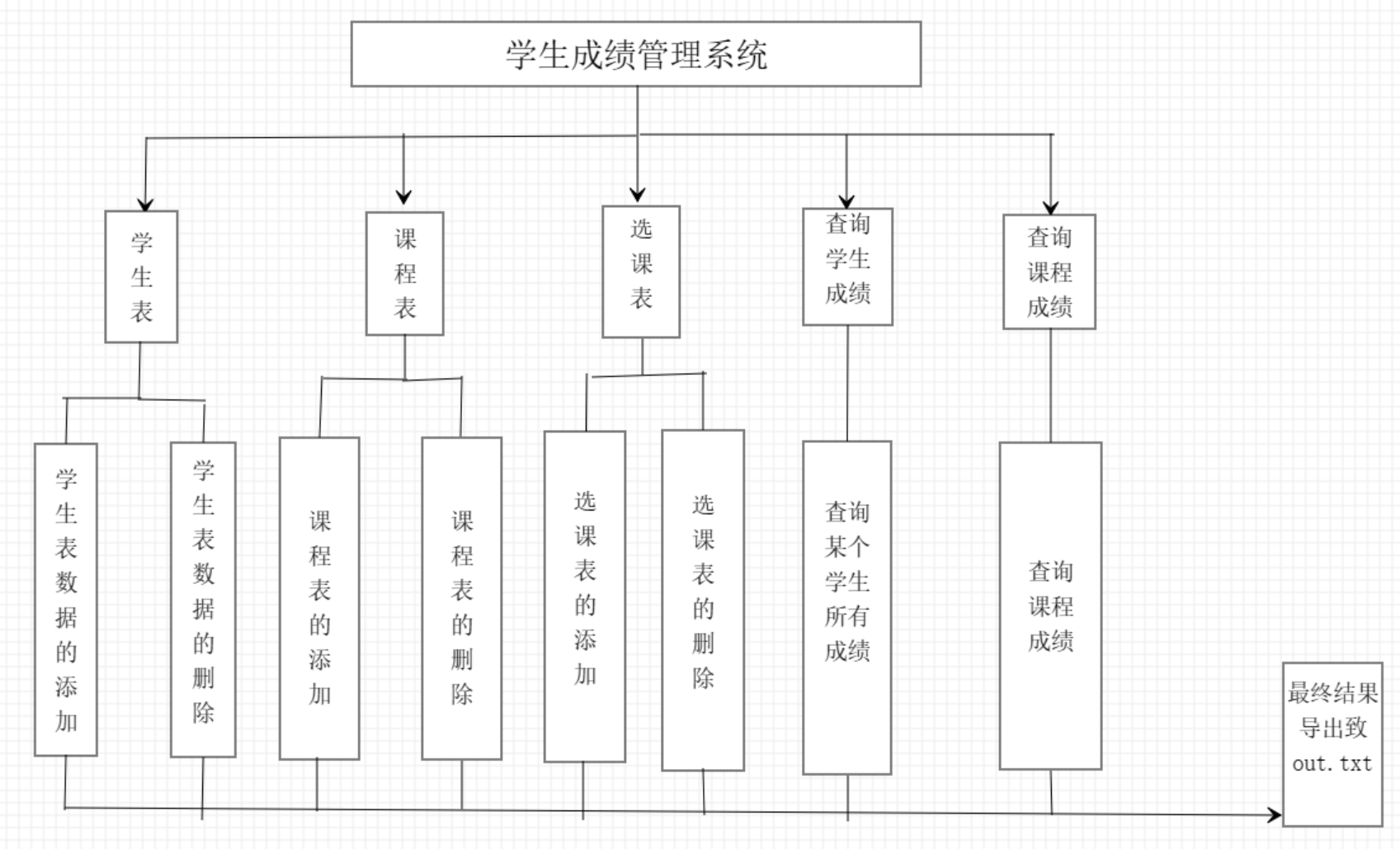
（1）学生表的插入和删除

（2）课程表的插入和删除

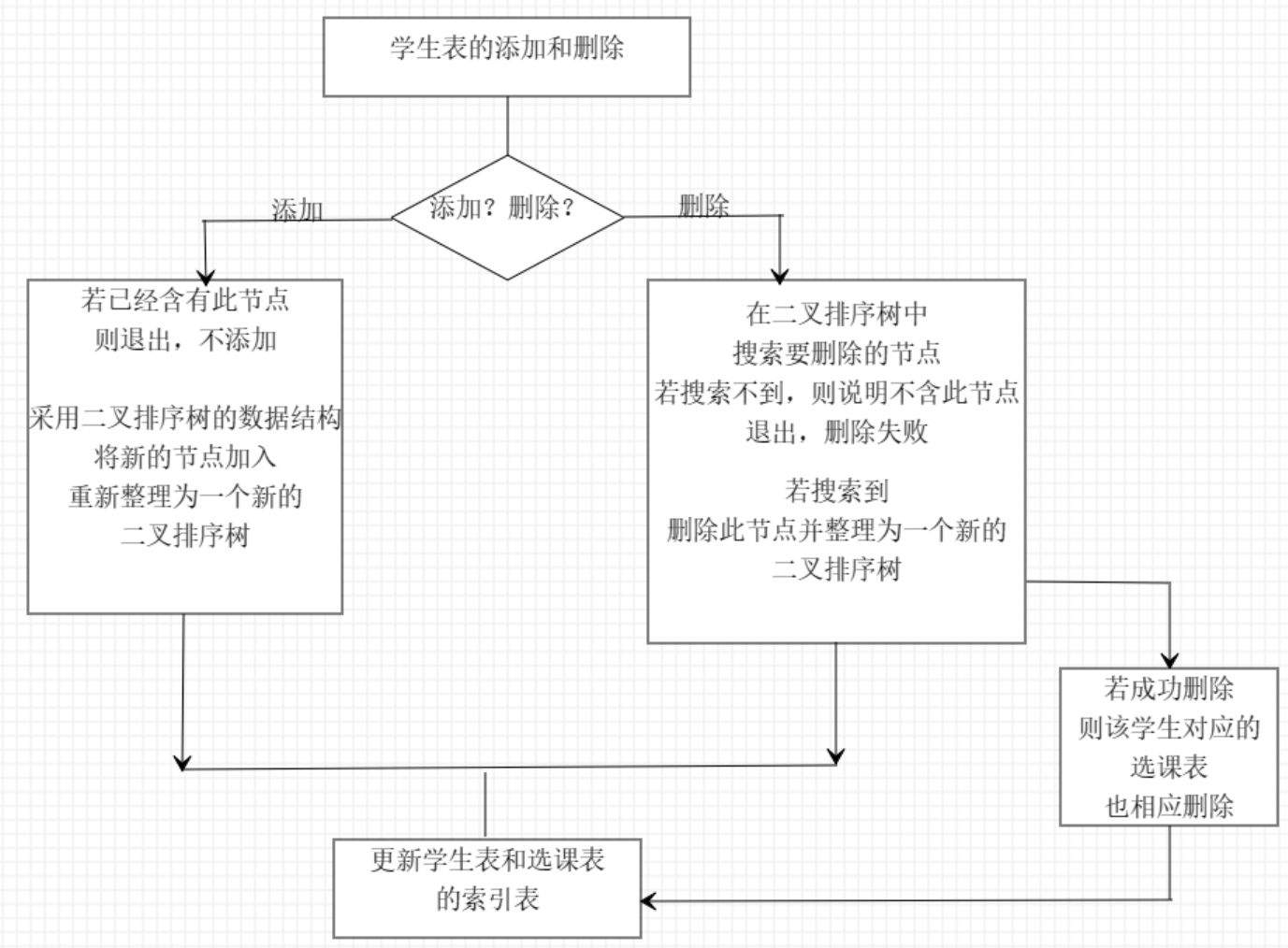
（3）选课表的插入和删除

（4）根据姓名，查询某个学生的所有成绩（课程名，得分）和平均成绩

（5）根据课程名，查询某门课的所有成绩（姓名，得分）和平均成绩

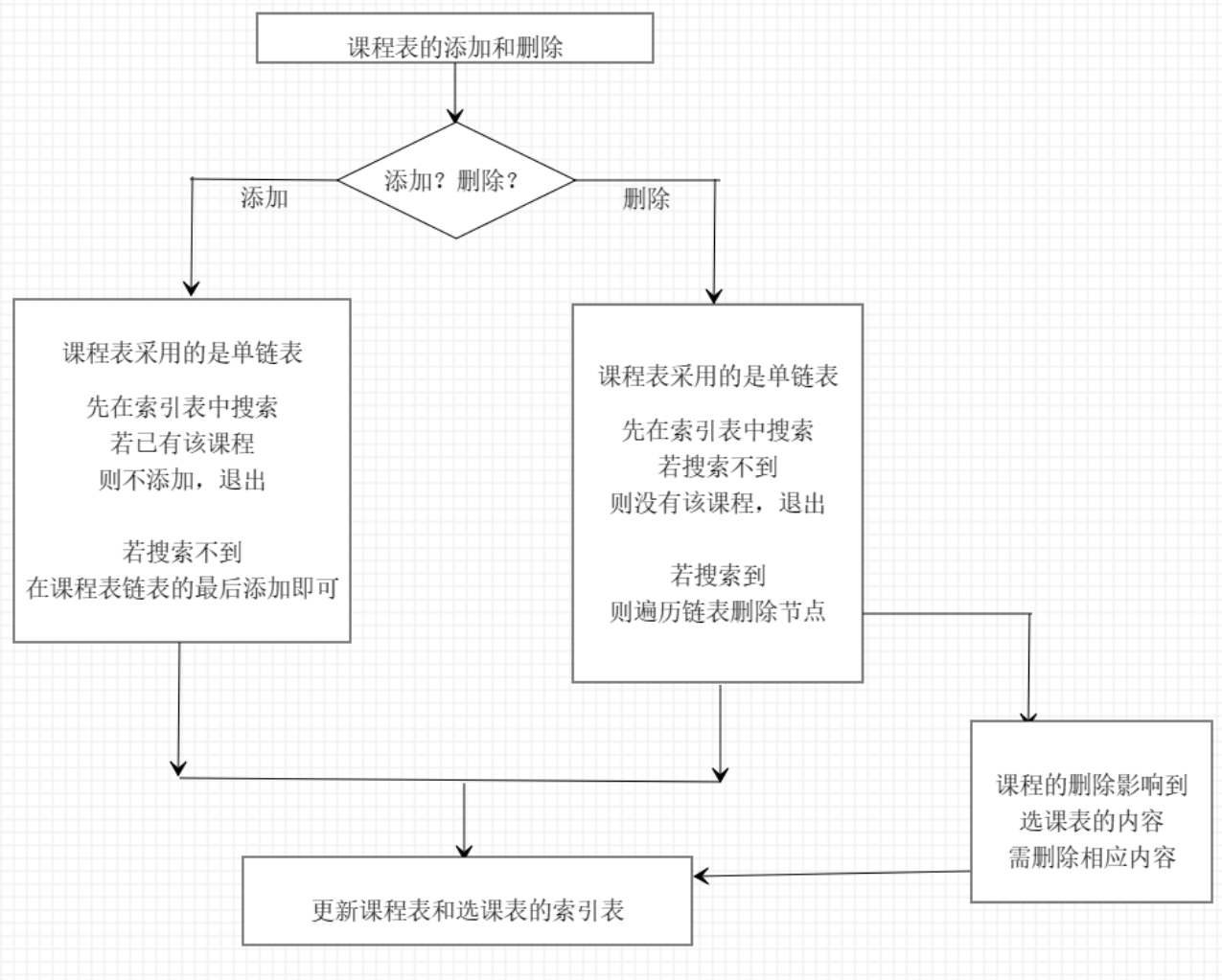
****（6）退出时，将学生、课程及选课信息保存到文件中

**2.2 功能详细说明**

1. **学生表的添加和删除**

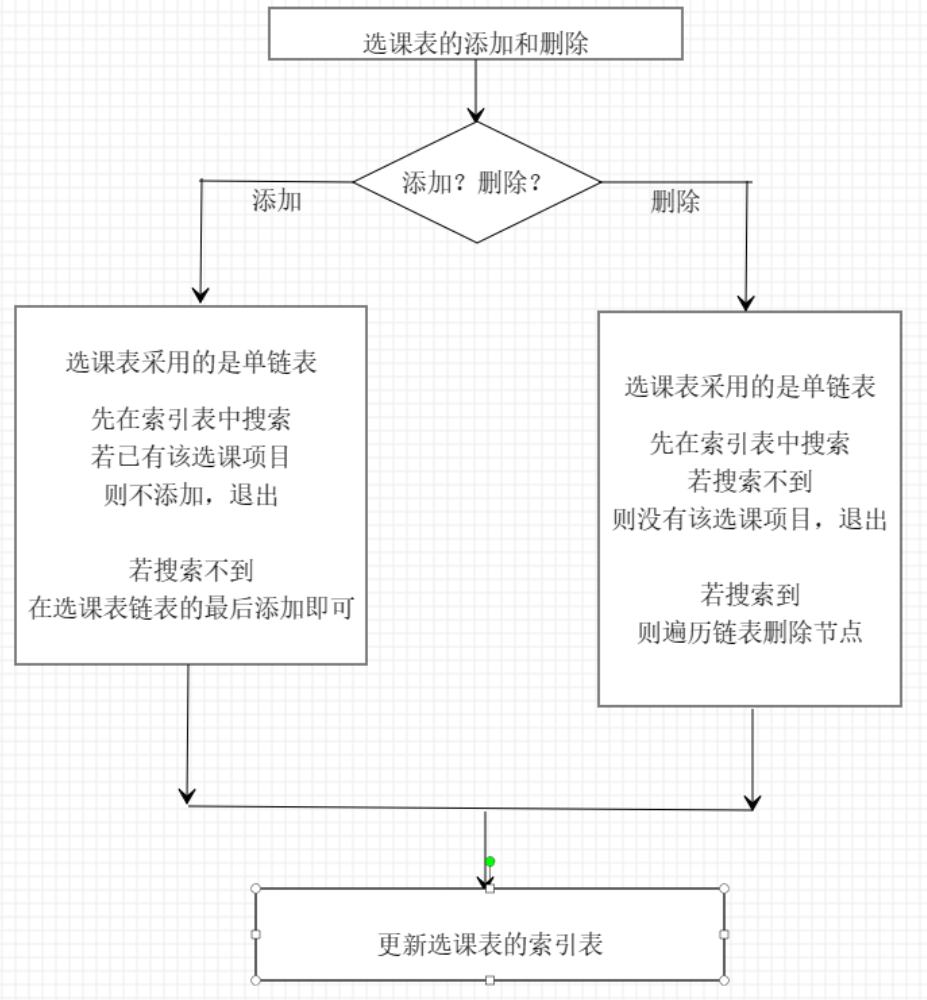
根据用户的输入选择相应的功能，再用户输入完成之后，检查用户的输入是否满足要求，防止（重复添加/无效的删除），若输入合法，执行相应操作。

由于学生表和选课表关联，删除学生时需要删除相应的选课项目

1. **课程表的添加和删除**

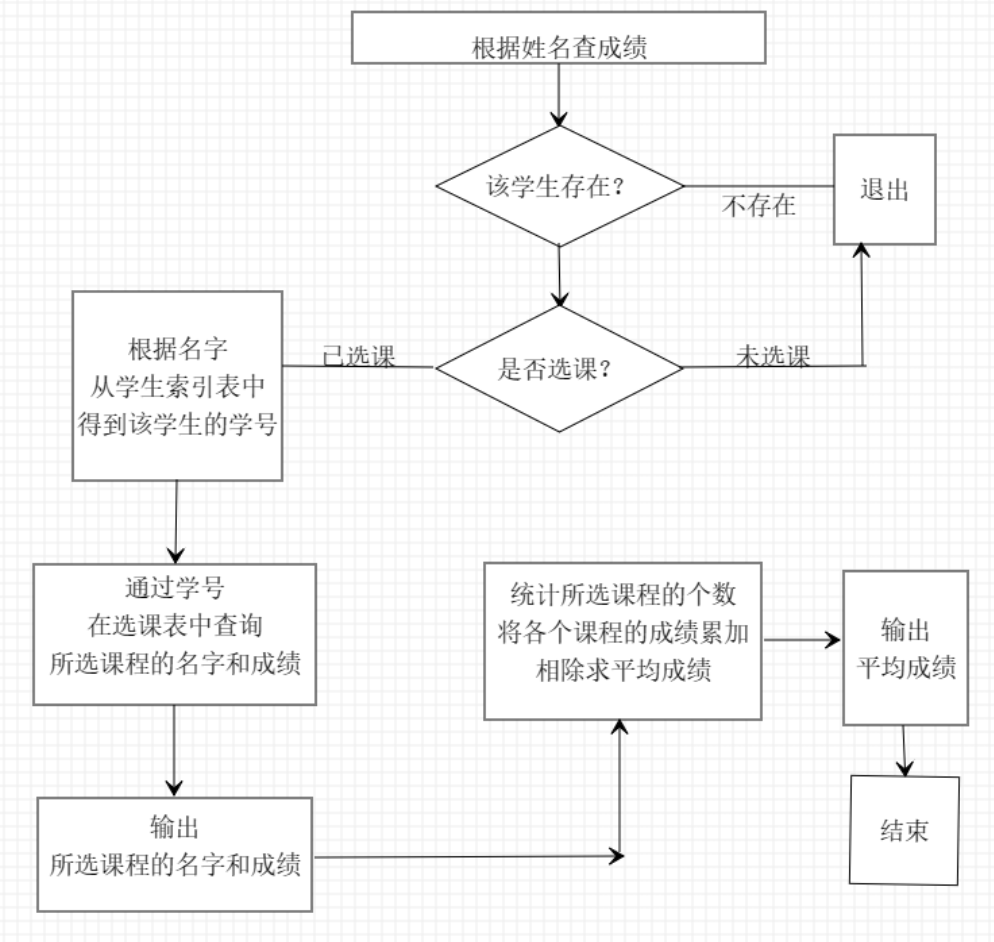
根据用户的输入选择相应的功能，再用户输入完成之后，检查用户的输入是否满足要求，防止（重复添加/无效的删除），若输入合法，执行相应操作。

由于课程表和选课表关联，删除课程时需要删除相应的选课项目

1. **选课表的添加和删除**

根据用户的输入选择相应的功能，再用户输入完成之后，检查用户的输入是否满足要求，防止（重复添加/无效的删除），若输入合法，执行相应操作。

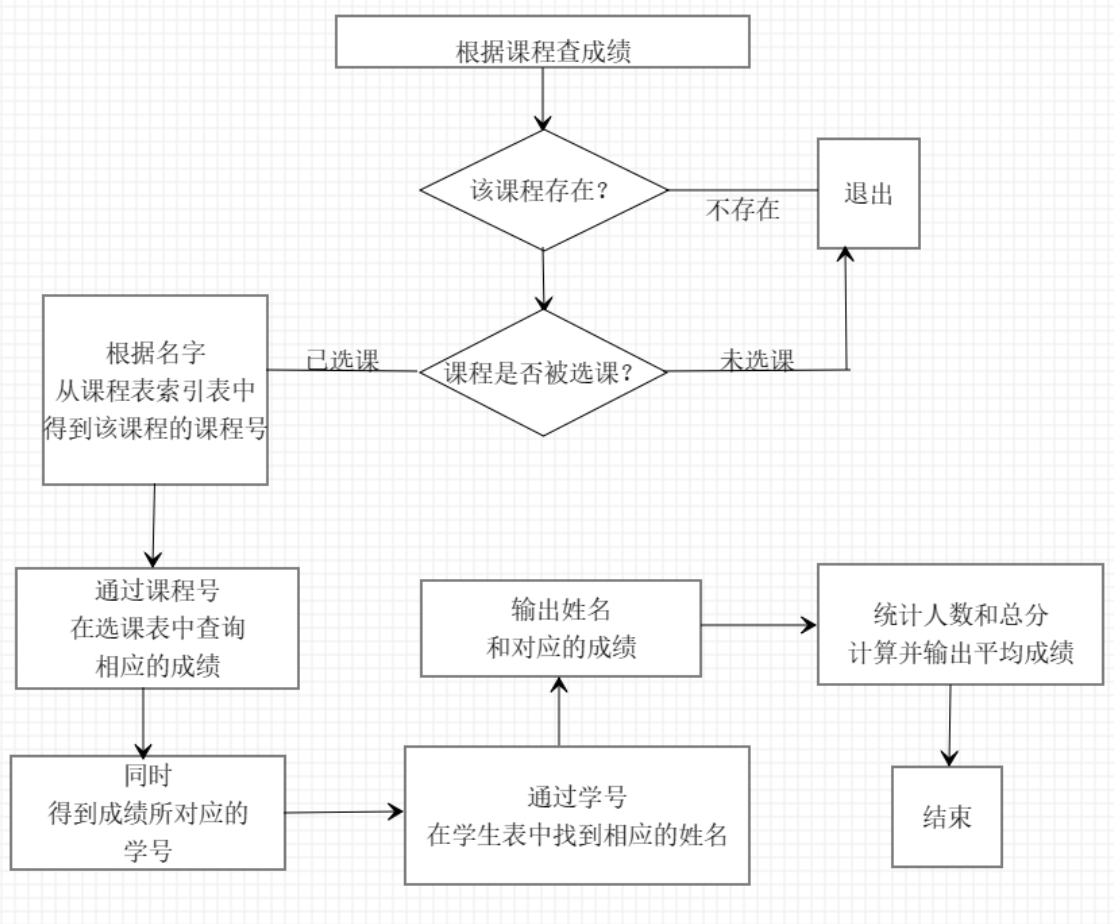
由于选课表不和其他表关联，在添加和删除时只需检验相应的学号和课程号是否存在，并进行相应操作即可，操作完成之后，更新索引表。

1. **根据姓名查询学生成绩**

根据用户的输入选择相应的功能，再用户输入完成之后，检查用户的输入是否满足要求，这里检查两个点，1.学生姓名是否存在，2.学生是否选课。

先从学生索引表中得到学号，因为学号与选课表关联，由此得到课程号和成绩，又因为课程号和课程表关联，由此得到课程的名称。

最后，将课程数和总分统计，相除得到最后的平均成绩（取整数）。

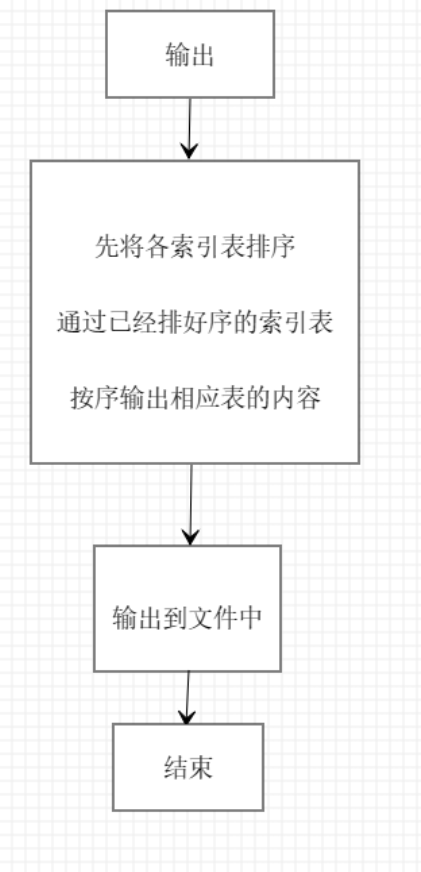
1. **根据课程名查成绩**

根据用户的输入选择相应的功能，再用户输入完成之后，检查用户的输入是否满足要求，这里检查两个点，1.课程名是否存在，2.课程是否被选课。

先从课程索引表中得到课程号，因为课程号与选课表关联，由此得到学号和成绩，又因为学号号和学生表关联，由此得到学生的姓名。

最后，将人数和总分统计，相除得到最后的平均成绩（取整数）。

1. **输出各表**

用户输入相应内容时，执行操作

这里需要注意的是

程序中的每一个表

都有一个相应的索引表

我们通过给索引表按关键字排序

从而达到不给表本体排序

而按序输出的效果

最后

将内容输出到文件中即可

1. **数据与结构说明**

**3.1 数据结构**

1. **学生表**

学生表要求采用二叉排序树的数据结构。定义如下

struct student//学生表

{

int key;//学号

string name;//姓名

string size;//性别

student\* lchild, \* rchild;

};

二叉排序树可以高效的存储学生数据，同时完成排序，添加和删除节点较为方便，不必专门进行排序操作，在添加/生成/删除节点时，会根据关键字自动完成排序操作。

1. **学生表的索引表**

为了更方便的获取学生学号和姓名的关系，创建了学生表索引表，保存学号和姓名两个信息，定义如下。

struct student\_index\_table//学生表索引表（以S\_key,学号为关键字）

{

int S\_key;//学号

string S\_name;//姓名

};

此数据类型主要为了和其他两个表（课程表，选课表联动），可以更快更有效的得到学号和姓名的对应关系。

1. **课程表和选课表**

课程表和选课表要求使用顺序存储结构，我的程序中，两个表均采用单链表的链式存储结构，定义如下。

struct course//课程表

{

int id;//课程号

string course\_name;//课程名

int credits;//学分

course\* next = NULL;

};

struct clAss//选课表

{

int key;//学号

int id;//课程号

int scores;//成绩

clAss\* next = NULL;

};

链式存储相应的数据，采用此种方法的原因是，方便添加和删除，在频繁的添加和删除之后能够完成的输出剩余的数据。

1. **课程表和选课表的索引表**

课程表和选课表是单链表，如果想要取得某一单元的信息，就需要遍历，因此，设计了索引表保存每一个单元的地址信息，方便获取，方便和其他两表（学生表，选课/课程表联动），定义如下

struct clAss\_index\_table//选课索引表结构体定义

{

int key;//学号

clAss\* body;//地址

};

struct course\_index\_table//课程索引表结构体定义

{

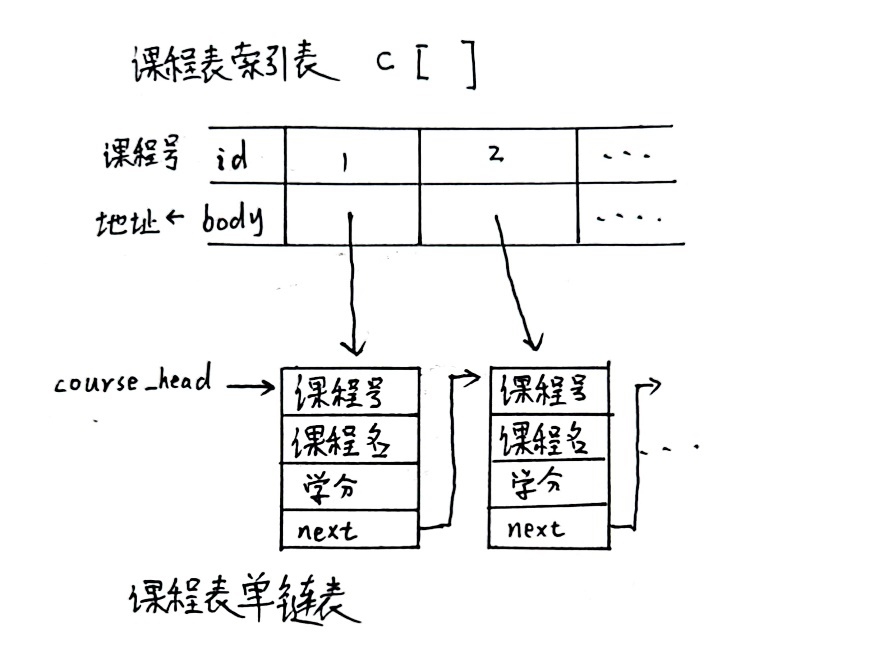
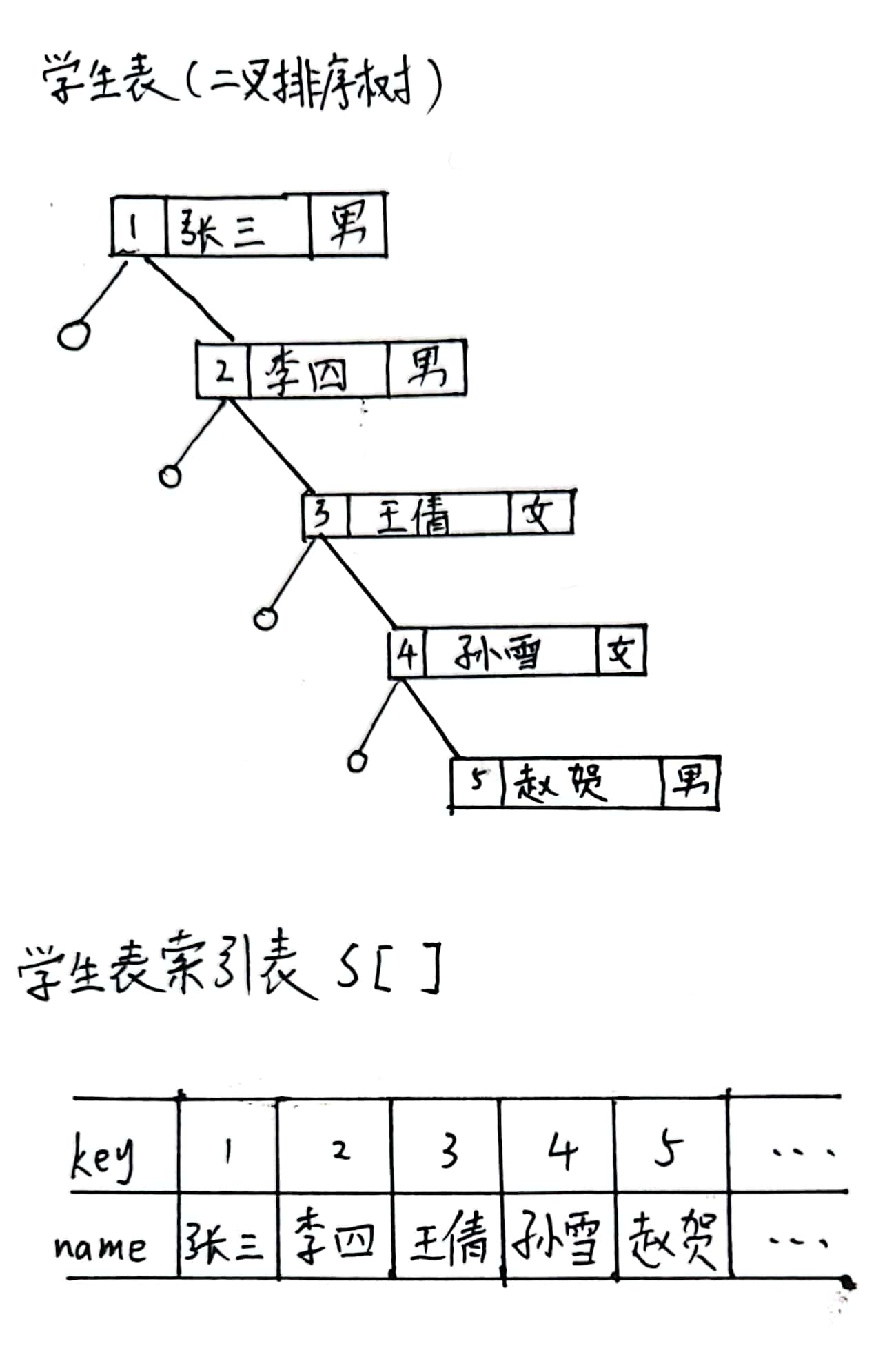
int key;//课程号

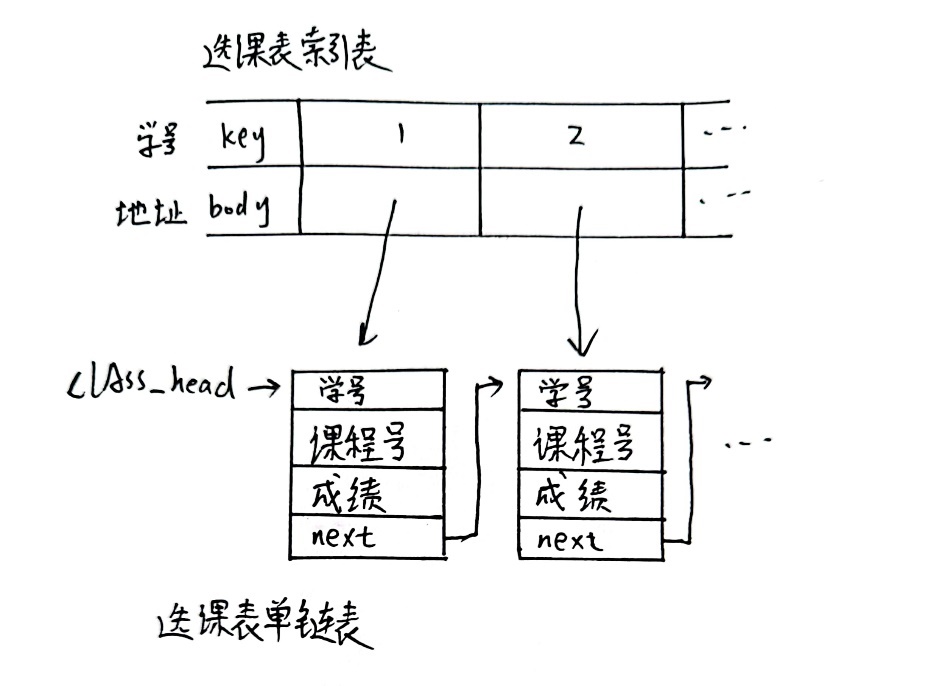
course\* body;//地址

};

两个索引表都采用顺序存储结构，数组。方便查找和添加。

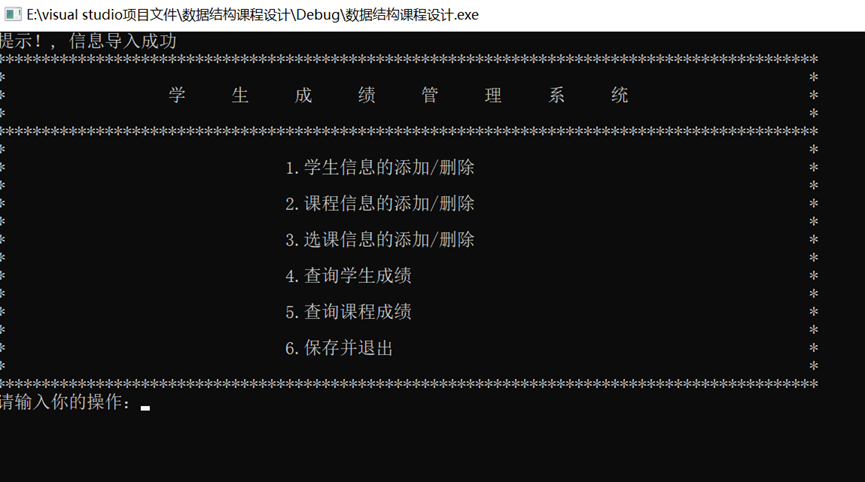
**3.2 初始化逻辑结构**

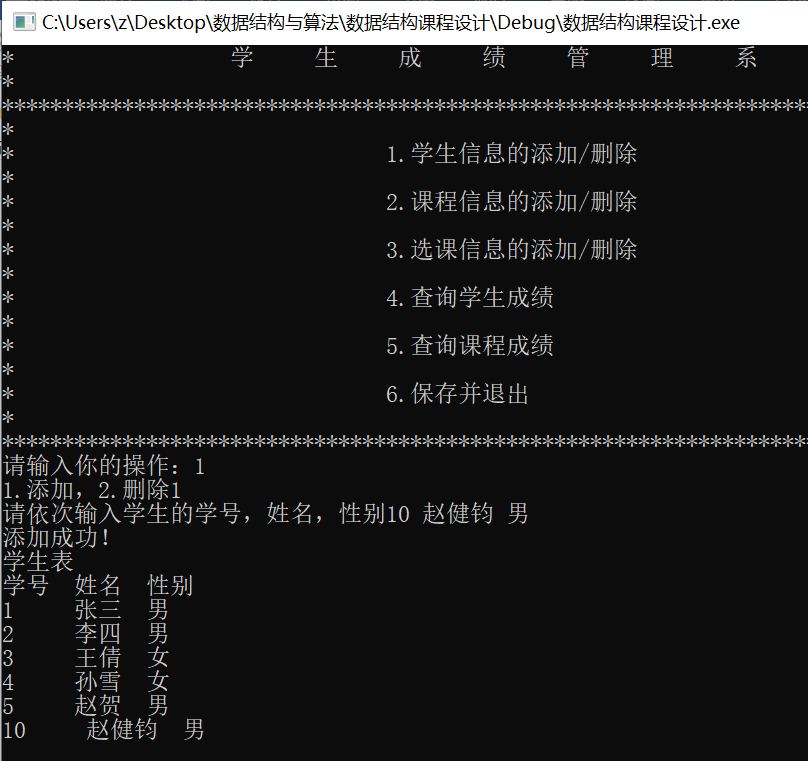
****导入数据之后，初始化的逻辑结构如图所示

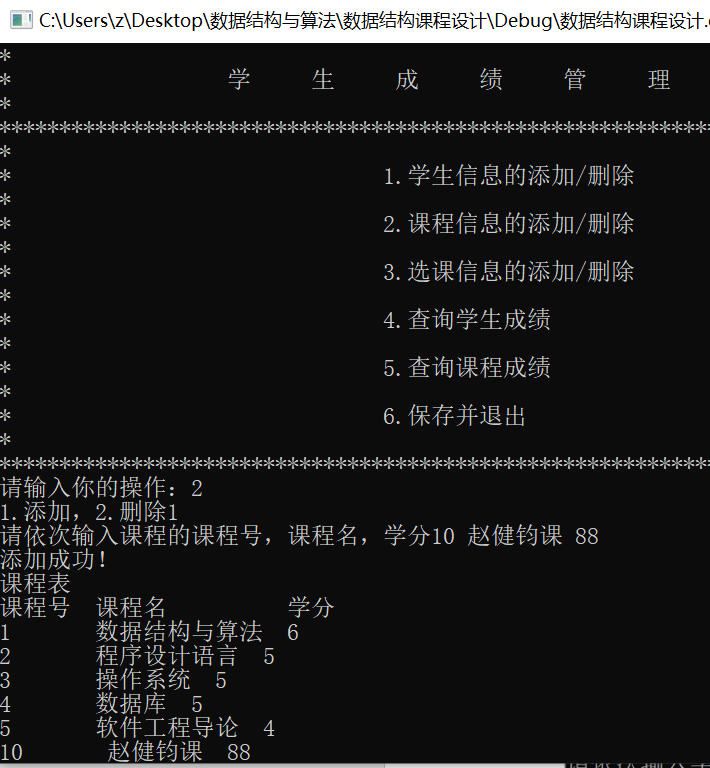


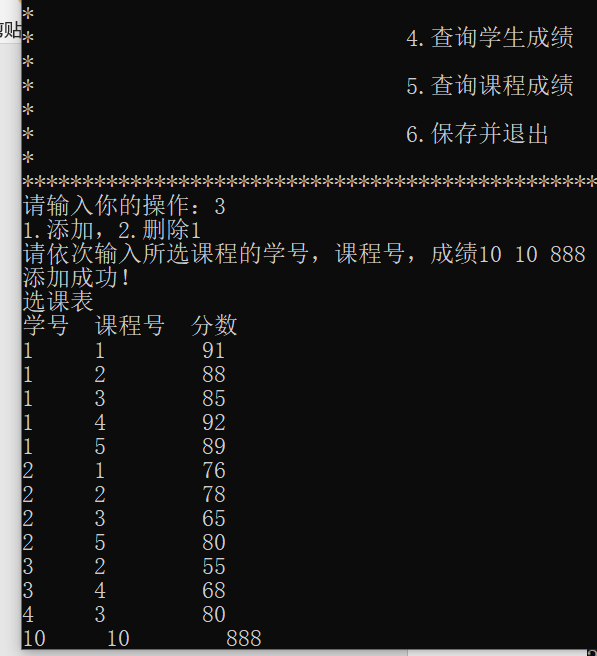
**3.2 模块函数**

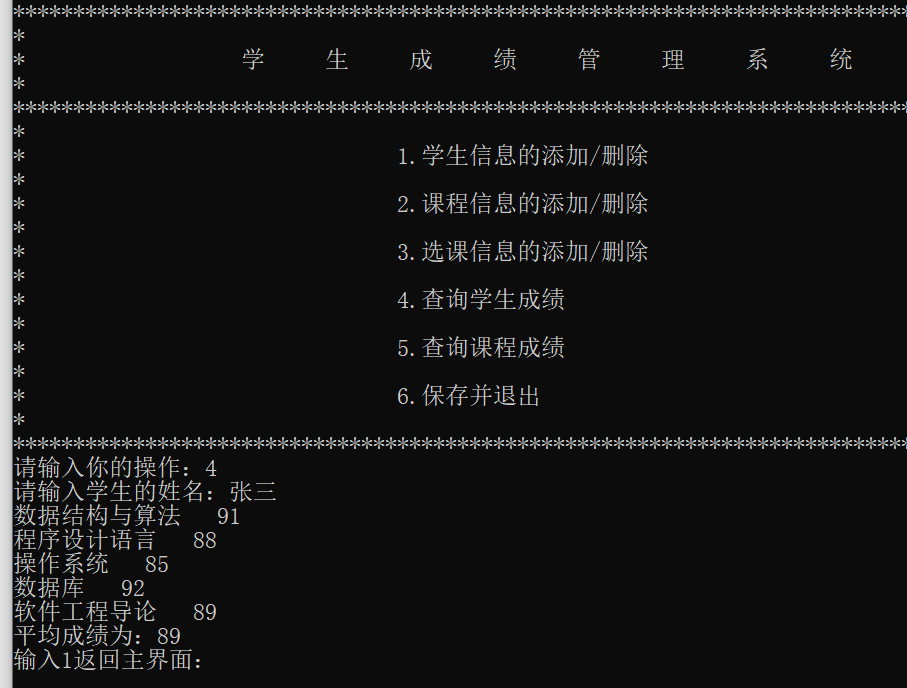
1. student\* InserBST(student\* bt, int k, string name, string size) //插入节点数据and创建树
2. student\* find(const int& key, student\* bt)//在二叉排序树中寻找节点
3. void Delete(const int& k, student\*& p) //在p为根的二叉排序树上删除关键字为k的结点
4. void Shellsort\_student\_index\_table(student\_index\_table R[], int n)//对学生表索引进行希尔排序
5. void create\_course\_table(course\*& course\_head, int ID, string c\_name, int cr)//创建和添加课程表
6. void Shellsort\_course\_index\_table(course\_index\_table R[], int n)//对课程表索引进行希尔排序
7. void create\_clAss\_table(clAss\*& clAss\_head, int k, int ID, int sc)//创建和添加选课表
8. void Shellsort\_clAss\_index\_table(clAss\_index\_table R[], int n)//对选课表索引进行希尔排序
9. void Delete\_clAss\_table(clAss\*& clAss\_head, int key, int ID)//删除选课表的数据
10. void Delete\_clAss\_table\_for\_student(clAss\*& clAss\_head, int key)//按照学号删除选课表的数据，此函数用于，删除学生数据时，处理对选课表造成的影响
11. void Delete\_clAss\_table\_for\_course(clAss\*& clAss\_head, int ID)//按照课程号删除选课表的数据，此函数用于，删除课程数据时，处理对选课表造成的影响
12. void Delete\_course\_table(course\*& course\_head, int key)//删除课程数据
13. void InOrder(student\* b) //中序遍历递归，输出学生表的信息
14. **运行截图**

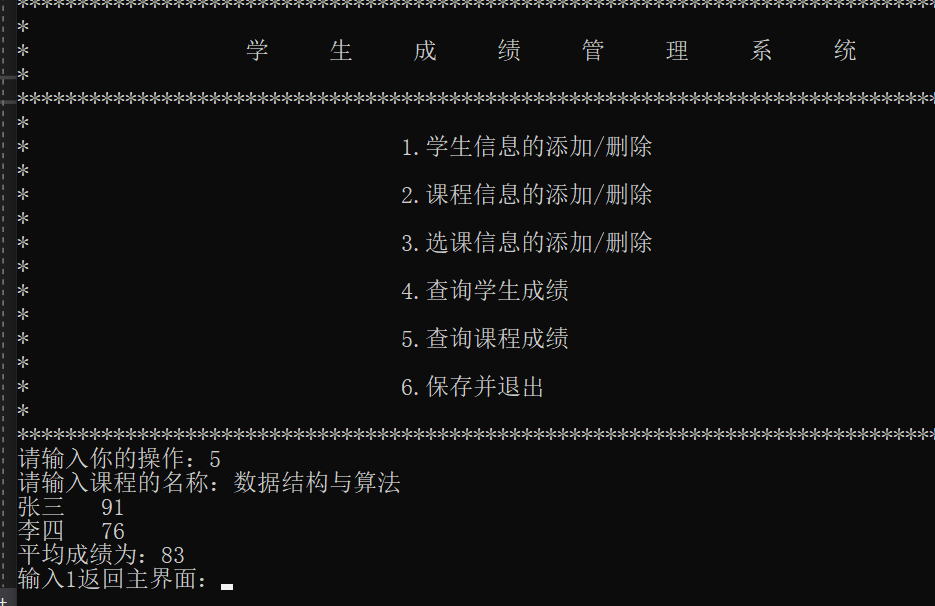
开始界面

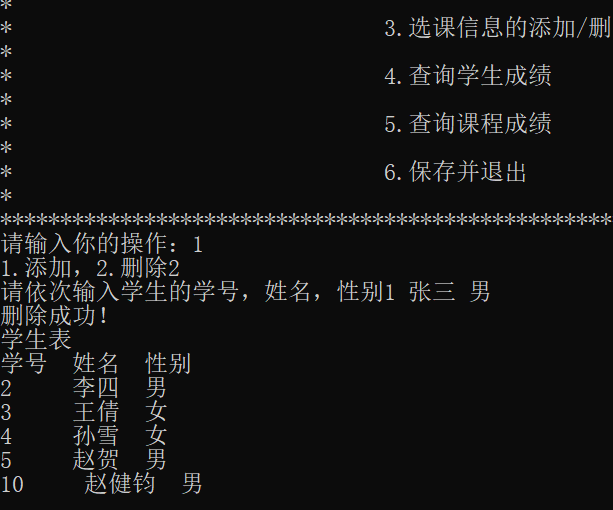
下面的操作是添加一个学生

下面添加一个课程信息

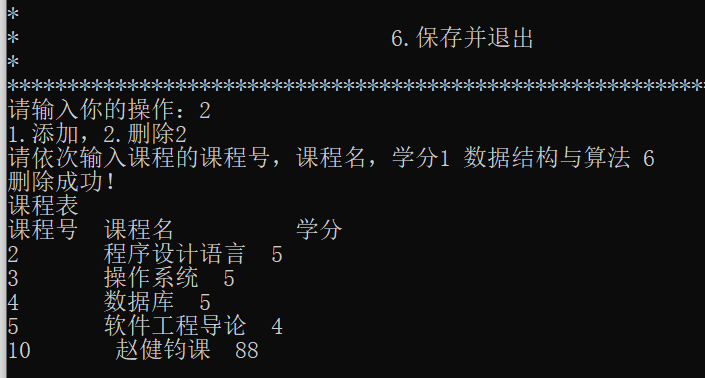
下面添加一个选课信息

下面查询张三的成绩

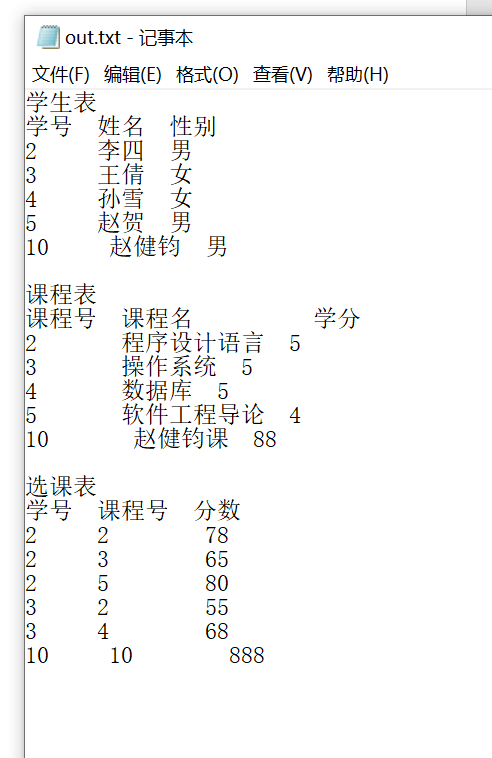
下面查询数据结构课程的成绩

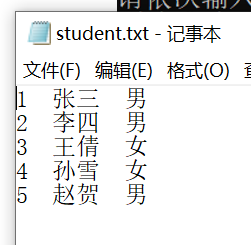
下面删除学生张三，同时删除了选课表中张三的选课

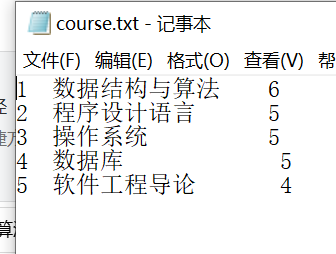
下面删除数据结构与算法的课程，同时删除了选课表中的项目

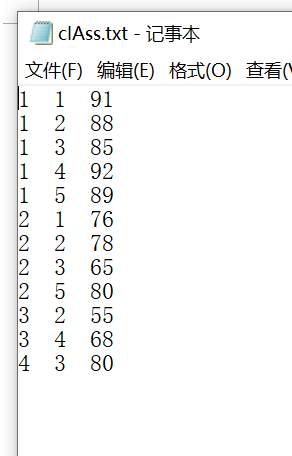


保存并导出，查看导出的文件



程序开始时导入的三个文件如图





1. **遇到问题及心得体会**

**5.1 遇到问题**

之前没有编写过需要多个表联动查找和输出的问题，在编写时，很多时候由于数据结构太抽象而导致对算法的设计不清楚，导致算法错误

针对多个表进行联动查询和输出的时候，经常出现访问受限的错误，某个指针会变成NULL，导致程序中断。

解决

对于算法的问题，我针对三个表设计了索引表，使的算法的设计在逻辑上简单了很多，思路也更加清晰。

针对指针的问题，多出现在添加和删除数据之后，我尝试各种操作，每一步进行调试，逐步修改算法中指针的赋值，解决了问题。

**5.2 心得体会**

学会了多个表和存储结构的联动，熟悉了索引的概念和用法，设计了相对简单的算法，对程序的整体性有了更多的了解，也对我写程序，设计算法有莫大的帮助，运用了各种数据结构，程序的逻辑更清楚。收获颇丰。

1. **带注释的代码**

struct clAss//选课表

{ int key;//学号

int id;//课程号

int scores;//成绩

clAss\* next = NULL;

};

struct clAss\_index\_table//索引表结构体定义

{ int key;

clAss\* body;

};

void create\_clAss\_table(clAss\*& clAss\_head, int k, int ID, int sc)//创建和添加选课表

{clAss\* p = new clAss;

if (clAss\_head == NULL)

{ clAss\_head = p;

clAss\_head->key = k; clAss\_head->id = ID; clAss\_head->scores = sc;

clAss\_head->next=NULL;

}

else

{clAss\* j = clAss\_head;

while (j->next != NULL)

j = j->next;

j->next = p;

p->key = k; p->id = ID; p->scores = sc;

}

}

void Shellsort\_clAss\_index\_table(clAss\_index\_table R[], int n)//对选课表索引进行希尔排序

{ int i, j, d;

clAss\_index\_table tmp;

d = n / 2; //增量置初值

while (d > 0)

{for (i = d; i < n; i++)

{ //对相隔d位置的元素组直接插入排序

tmp = R[i];

j = i - d;

while (j >= 0 && tmp.key < R[j].key)

{R[j + d] = R[j];

j = j - d;

}

R[j + d] = tmp;

}

d = d / 2; //减小增量

}

}

struct course//课程表

{ int id;//课程号

string course\_name;//课程名

int credits;//学分

course\* next = NULL;

};

struct course\_index\_table//索引表结构体定义

{ int key;

course\* body;

};

void create\_course\_table(course\*& course\_head, int ID, string c\_name, int cr)//创建和添加课程表

{course\* p = new course;

if (course\_head == NULL)

{ course\_head = p;

course\_head->id = ID; course\_head->course\_name = c\_name; course\_head->credits = cr;

course\_head->next = NULL;

}

else

{course\* j = course\_head;

while (j->next != NULL)

j = j->next;

j->next = p;

p->id = ID; p->course\_name = c\_name; p->credits = cr;

}

}

void Shellsort\_course\_index\_table(course\_index\_table R[], int n)//对课程表索引进行希尔排序

{int i, j, d;

course\_index\_table tmp;

d = n / 2; //增量置初值

while (d > 0)

{for (i = d; i < n; i++)

{ //对相隔d位置的元素组直接插入排序

tmp = R[i];

j = i - d;

while (j >= 0 && tmp.key < R[j].key)

{ R[j + d] = R[j];

j = j - d;

}

R[j + d] = tmp;

}

d = d / 2; //减小增量

}

}

void Shellsort\_student\_index\_table(student\_index\_table R[], int n)//对学生表索引进行希尔排序

{int i, j, d;

student\_index\_table tmp;

d = n / 2; //增量置初值

while (d > 0)

{for (i = d; i < n; i++)

{ //对相隔d位置的元素组直接插入排序

tmp = R[i];

j = i - d;

while (j >= 0 && tmp.S\_key < R[j].S\_key)

{ R[j + d] = R[j];

j = j - d;

}

R[j + d] = tmp;

}

d = d / 2; //减小增量

}

}struct student//student，学生表

{

int key;//学号

string name;//姓名

string size;//性别

student\* lchild, \* rchild;

};

struct student\_index\_table//学生表索引

{int S\_key;

string S\_name;

};

student\* InserBST(student\* bt, int k, string name, string size) //插入节点数据and创建树

{if (bt == NULL)

{bt = new student;

bt->key = k; bt->name = name; bt->size = size;

bt->lchild = bt->rchild = NULL;

}

else if (k < bt->key)

bt->lchild = InserBST(bt->lchild, k, name,size);

else if (k > bt->key)

bt->rchild = InserBST(bt->rchild, k, name, size);

return bt;

}

student\* find(const int& key, student\* bt)//在二叉排序树中寻找节点

{student\* p = bt;

student\* f = NULL;//初始化

while (p != NULL && p->key != key) {//往下搜索

if (key < p->key) {

f = p;

p = p->lchild;

}

else {

f = p;

p = p->rchild;

}

}

return p;//若数值不存在，则指向NULL结点被返回

}

void Delete(const int& k, student\*& p) {

//在p为根的二叉排序树上删除关键字为k的结点

student\* s, \* temp;

if (p != NULL)

if (k < p->key)//还没找到p

Delete(k, p->lchild);//递归地在p的左子树上删除关键字为k的结点

else if (k > p->key)

Delete(k, p->rchild);//递归地在p的左子树上删除关键字为k的结点

else if (p->lchild != NULL && p->rchild != NULL) {//找到p,但是p的左右子树都不空

//s = Min(p->rchild);

temp = p->rchild;

while (temp->lchild != NULL) {

temp = temp->lchild;

}//找到p的右子树上最小的数s，替换掉p，然后删掉s

s = temp;

p->key = s->key;//将p替换成s

Delete(s->key, p->rchild);//递归地删掉s

}

else {//相等找到,但是左或右为空

temp = p;

if (p->lchild == NULL) p = p->rchild;//左子树空，则将p替换为右子树上第一个结点

else if (p->rchild == NULL) p = p->lchild;//右子树空，则将p替换为左子树上第一个结点

delete temp;

}

}

int student\_num=0, clAss\_num=0, course\_num=0;//全局变量

void Delete\_clAss\_table(clAss\*& clAss\_head, int key, int ID)//删除选课表的数据

{

clAss\* p = NULL, \* q = NULL;

if (clAss\_head == NULL)

cout << "无数据表！";

p = clAss\_head;

if (ID == p->id && key == p->key)

{

clAss\_head = clAss\_head->next;

delete p;

clAss\_num--;

}

while ((ID != p->id || key != p->key) && p->next != NULL)

{

q = p;

p = p->next;

}

if (ID == p->id && key == p->key)

{

q->next = p->next;

delete p;

clAss\_num--;

}

}

void Delete\_clAss\_table\_for\_student(clAss\*& clAss\_head, int key)//按照学号删除选课表的数据

//此函数用于，删除学生数据时，处理对选课表造成的影响

{

clAss\* p = NULL, \* q = NULL;

if (clAss\_head == NULL)

cout << "无数据表！";

p = clAss\_head;

goon: if (key == p->key)

{

clAss\_head = clAss\_head->next;

delete p;

p = clAss\_head;

clAss\_num--;

goto goon;

}

while (p->next != NULL)

{

if (key != p->key)

{

q = p;

p = p->next;

}

if (key == p->key)

{

q->next = p->next;

delete p;

p = q;

clAss\_num--;

}

}

}

void Delete\_clAss\_table\_for\_course(clAss\*& clAss\_head, int ID)//按照课程号删除选课表的数据

//此函数用于，删除课程数据时，处理对选课表造成的影响

{

clAss\* p = NULL, \* q = NULL;

if (clAss\_head == NULL)

cout << "无数据表！";

p = clAss\_head;

goon: if (ID == p->id )

{

clAss\_head = clAss\_head->next;

delete p;

clAss\_num--;

p = clAss\_head;

goto goon;

}

while ( p->next != NULL)

{

if (ID != p->id)

{

q = p;

p = p->next;

}

if (ID == p->id)

{

q->next = p->next;

delete p;

p = q;

clAss\_num--;

}

}

}

void Delete\_course\_table(course\*& course\_head, int key)//删除课程数据

{

course\* p = NULL, \* q = NULL;

if (course\_head == NULL)

cout << "无数据表！";

p = course\_head;

if (key == p->id)

{

course\_head = course\_head->next;

delete p;

course\_num--;

p = course\_head;

}

while (p->next != NULL)

{

if (key != p->id)

{

q = p;

p = p->next;

}

if (key == p->id)

{

q->next = p->next;

delete p;

p = q;

course\_num--;

}

}

}

void InOrder(student\* b) //中序遍历递归，输出学生表的信息

{

ofstream out("out.txt");

if (b != NULL)

{

InOrder(b->lchild);

out << b->key << " " << b->name << " " << b->size << endl;

InOrder(b->rchild);

}

}

void main()

{

ifstream student\_in("student.txt");

ifstream course\_in("course.txt");

ifstream clAss\_in("clAss.txt");//导入三张表的信息

int flag = 0;//标志位，0为文件打开/导入失败，1为成功

if (student\_in && course\_in && clAss\_in)//检查三个文件是否全部成功打开

{

cout << "提示！, 信息导入成功" << endl;

flag = 1;//表示文件导入成功

}

else

{

cout<<"警告！, 文件读入/打开错误，请检查文件！"<<endl;

}

student\* bt = NULL;//学生表的根节点

clAss\* clAss\_head = NULL;//选课表表头

course\* course\_head = NULL;//课程表表头

int k, id, scores, credits;//用来存储输入数据的变量

string st\_name, st\_size, course\_name;//用来存储输入数据的变量

student\_index\_table student\_index[20];//学生表索引表

clAss\_index\_table clAss\_index[50];//选课表索引表

course\_index\_table course\_index[20];//课程表索引表

while (!student\_in.eof() && flag == 1)//创建学生表（二叉排序树）

{

student\_in >> k >> st\_name >> st\_size;

bt = InserBST(bt, k, st\_name, st\_size);

student\_index[student\_num].S\_key = k;//创建学生表索引

student\_index[student\_num].S\_name = st\_name;

student\_num++;

}

while (!clAss\_in.eof() && flag == 1)//创建选课表（顺序表）

{

clAss\_in >> k >> id >> scores;

create\_clAss\_table(clAss\_head, k, id, scores);

clAss\_num++;

}

while (!course\_in.eof() && flag == 1)//创建课程表（顺序表）

{

course\_in >> id >> course\_name >> credits;

create\_course\_table(course\_head, id, course\_name, credits);

course\_num++;

}

//下面，初始化clAss和course的索引表

clAss\* p = new clAss;

p = clAss\_head;

for (int i = 0; i < clAss\_num; i++)//初始化选课表

{

clAss\_index[i].key = p->key;//这里以选课表中的学号作为关键字

clAss\_index[i].body = p;

p = p->next;

}

course\* q = new course;

q = course\_head;

for (int i = 0; i < course\_num; i++)//初始化课程表

{

course\_index[i].key = q->id;//这里以课程表中的课程代号作为关键字

course\_index[i].body = q;

q = q->next;

}

//下面对课程表和选课表的两个索引表按关键字排序(采用希尔排序)

Shellsort\_course\_index\_table(course\_index, course\_num);

Shellsort\_clAss\_index\_table(clAss\_index, clAss\_num);

student\_in.close();

course\_in.close();

clAss\_in.close();//关闭文件

goon:while (1)

{//菜单栏

cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl;

cout << "\* \*" << endl;

cout << "\* 学 生 成 绩 管 理 系 统 \*" << endl;

cout << "\* \*" << endl;

cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl;

cout << "\* \*" << endl;

cout << "\* 1.学生信息的添加/删除 \*" << endl;

cout << "\* \*" << endl;

cout << "\* 2.课程信息的添加/删除 \*" << endl;

cout << "\* \*" << endl;

cout << "\* 3.选课信息的添加/删除 \*" << endl;

cout << "\* \*" << endl;

cout << "\* 4.查询学生成绩 \*" << endl;

cout << "\* \*" << endl;

cout << "\* 5.查询课程成绩 \*" << endl;

cout << "\* \*" << endl;

cout << "\* 6.保存并退出 \*" << endl;

cout << "\* \*" << endl;

cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl;

int Flag = 0;//用户操作指示标志，通过判断它的值来判别用户执行的操作

cout << "请输入你的操作：";

cin >> Flag;//选择操作

if (Flag == 1) //学生信息的添加/删除

{

int f=0;//添加删除标志，1为添加，2为删除

cout << "1.添加，2.删除"; cin >> f;

int st\_key; string st\_name, st\_size;//存储输入的临时变量

cout << "请依次输入学生的学号，姓名，性别";

cin >> st\_key >> st\_name >> st\_size;

if (f == 1) //添加

{

int i;

for ( i = 0; i < student\_num; i++)//在学生表索引表中查找用户输入的学号和姓名

{

if (st\_key == student\_index[i].S\_key || st\_name == student\_index[i].S\_name)//如果学号或姓名已经存在

{

cout << "学号或姓名已存在，添加失败"<<endl;

break;//退出返回主菜单

}

}

if (i == student\_num)//在学生表索引表中没有找到，证明可以添加

{

bt = InserBST(bt, st\_key, st\_name, st\_size);//二叉树添加节点

student\_index[student\_num].S\_key = st\_key;

student\_index[student\_num].S\_name = st\_name;//学生表索引表添加节点

student\_num++;//学生总数+1

cout << "添加成功！" << endl;

}

}

else//删除

{

int i;

for (i = 0; i < student\_num; i++)//在学生表索引表中查找

{

if (st\_key == student\_index[i].S\_key && st\_name == student\_index[i].S\_name)//如果找到该学生

{

Delete(st\_key, bt);//二叉树删除节点

Delete\_clAss\_table\_for\_student(clAss\_head, st\_key);//相应的选课表删除内容

cout << "删除成功！" << endl;

for (int d = 0; d < student\_num; d++)

{

if (student\_index[d].S\_key == st\_key && student\_index[d].S\_name == st\_name)//学生表索引表删除内容

{

student\_index[d].S\_key = student\_index[student\_num-1].S\_key;

student\_index[d].S\_name = student\_index[student\_num-1].S\_name;

break;

}

}

student\_num--;//学生数-1

//更新选课表索引表

clAss\* p = new clAss;

p = clAss\_head;

for (int i = 0; i < clAss\_num; i++)

{

clAss\_index[i].key = p->key;//这里以选课表中的学号作为关键字

clAss\_index[i].body = p;

p = p->next;

}

break;//返回主菜单

}

}

if (i == student\_num)//在学生表索引中找不到该学生

{

cout << "该学生不存在，无法删除！" << endl;

}

}

//输出当前学生表结果

cout << "学生表" << endl;

cout << "学号 姓名 性别" << endl;

student\* st;

Shellsort\_student\_index\_table(student\_index, student\_num);//对学生索引表排序，保证输出是顺序的

for (int i = 0; i < student\_num; i++)

{

cout << student\_index[i].S\_key << " " << student\_index[i].S\_name << " ";//从学生表中输出学号和名字

st = find(student\_index[i].S\_key, bt);//在二叉树中查找性别

if (st != NULL)cout << st->size << endl;

}

cout << endl;

goto goon;//该操作完成，返回主菜单判断用户的其他操作

}if (Flag == 2) //课程表的添加和删除

{

int f = 0;//添加删除标志，1为添加，2为删除

cout << "1.添加，2.删除"; cin >> f;

int c\_id,c\_credits; string c\_name;//临时变量

cout << "请依次输入课程的课程号，课程名，学分";

cin >>c\_id >> c\_name >> c\_credits;

if (f == 1)

{

int i;

for (i = 0; i < course\_num; i++)

{

if (c\_id == course\_index[i].key )//在课程表索引表中查找用户输入的课程号

{

cout << "课程已存在，添加失败"<<endl;

break;

}

}

if (i == course\_num)//在课程表索引表中没有找到，证明可以添加

{

create\_course\_table(course\_head, c\_id, c\_name, c\_credits); //添加课程表数据

course\_index[course\_num].key = c\_id;//更新课程表索引

course\* w = NULL; w = course\_head;

for (int e = 0; e < course\_num+1; e++)//使新的选课表索引指向新的数据

{

if (w->next == NULL)

{

course\_index[course\_num].body = w;

break;

}

else

w = w->next;

}

course\_num++;//课程数+1

cout << "添加成功！"<<endl;

}

}

else//删除

{

int i;

for (i = 0; i < course\_num; i++)//在课程表索引表中查找

{

if (c\_id == course\_index[i].key )//如果找到该课程

{

cout << "删除成功！" << endl;;

for (int d = 0; d < course\_num; d++)

{

Delete\_clAss\_table\_for\_course(clAss\_head, c\_id);//删除链表节点

if (course\_index[d].key == c\_id)//找到索引表中的相应元素删除

{

course\_index[d].key = course\_index[course\_num].key;

course\_index[d].body = course\_index[course\_num].body;

break;//因为课程只有一个，删除完一次之后就可以退出循环了

}

}

Delete\_course\_table(course\_head, c\_id);//相应的选课表删除内容

course\_num--;//课程数-1

//更新选课索引表

clAss\* p = new clAss;

p = clAss\_head;

for (int i = 0; i < clAss\_num; i++)

{

clAss\_index[i].key = p->key;//这里以选课表中的学号作为关键字

clAss\_index[i].body = p;

p = p->next;

}

break;

}

}

if (i == course\_num)//在课程表索引中找不到该课程

{

cout << "该课程不存在，无法删除！"<<endl;

}

//输出此时课程表的结果

cout << "课程表" << endl;

cout << "课程号 课程名 学分" << endl;

Shellsort\_course\_index\_table(course\_index, course\_num);//先对课程表索引希尔排序

course\* p = NULL,\*r=course\_head;

while (r != NULL)

{

p = r;

cout << p->id << " " << p->course\_name << " " << p->credits << endl;

r = r->next;

}

cout << endl;

goto goon;//返回主菜单

}

if (Flag == 3)//选课信息的添加/删除

{

int f = 0; // 添加删除标志，1为添加，2为删除

cout << "1.添加，2.删除"; cin >> f;

int st\_key, c\_id,c\_scores;//临时变量

cout << "请依次输入所选课程的学号，课程号，成绩";

cin >> st\_key>>c\_id >> c\_scores ;

if (f == 1)//添加

{

int i;

for (i = 0; i < clAss\_num; i++)//在选课表索引表中查找用户输入的学号和课程号

{

if (st\_key == clAss\_index[i].key&&c\_id==clAss\_index[i].body->id)

{

cout << "选课项目已存在，添加失败"<<endl;

break;

}

}

if (i == clAss\_num)//在选课表索引表中没有找到，证明可以添加

{

create\_clAss\_table(clAss\_head, st\_key, c\_id, c\_scores);//添加选课表数据

clAss\_index[clAss\_num].key = st\_key;//更新选课表索引

clAss\* w = NULL; w = clAss\_head;

for (int e = 0; e < clAss\_num+1; e++)//使新的选课表索引指向新的数据

{

if (w->next == NULL)

{

clAss\_index[clAss\_num].body = w;

break;

}

else

w = w->next;

}

clAss\_num++;

cout << "添加成功！"<<endl;

}

}

else//删除

{

int i;

for (i = 0; i < clAss\_num; i++)//在选课表索引表中查找

{

if (st\_key == clAss\_index[i].key&&c\_id==clAss\_index[i].body->id)//如果找到该项目

{

cout << "删除成功！"<<endl;

for (int d = 0; d < clAss\_num; d++) {

if (clAss\_index[d].key == st\_key&&clAss\_index[d].body->id==c\_id)//索引表删除内容

{

clAss\_index[d].key = clAss\_index[clAss\_num].key;

clAss\_index[d].body = clAss\_index[clAss\_num].body;

break;

}

}

Delete\_clAss\_table(clAss\_head, st\_key, c\_id);//选课表删除内容

clAss\_num--;//选课数-1

break;

}

}

if (i == course\_num)//在选课表索引中找不到该项目

{

cout << "该选课项目不存在，无法删除！"<<endl;

}

}

//输出当前选课表的结果

cout << "选课表" << endl;

cout << "学号 课程号 分数" << endl;

Shellsort\_clAss\_index\_table(clAss\_index, clAss\_num);//对索引表希尔排序

clAss\* q = NULL,\*u=clAss\_head;

while (u != NULL)

{

q = u;

cout << q->key << " " << q->id << " " << q->scores << endl;

u = u->next;

}

cout << endl;

goto goon;//返回

}if (Flag == 4)//学生姓名查成绩

{

string st\_name; int kkk,total\_s=0,total\_n=0;///kkk学号（关键字）

//total\_s=总分数累加。tolal\_n=总课程数

cout << "请输入学生的姓名：";

cin >> st\_name;

int d = 0;

for (d; d < student\_num; d++)//从学生表索引中搜索该学生并获取学号

{

if (student\_index[d].S\_name == st\_name)

kkk = student\_index[d].S\_key;//得到学号

}

if (d == student\_num+1)//找不到该学生

{

cout << "该学生不存在"<<endl;

goto goon;

}

else //找到学生，开始获取成绩

{

for (int x = 0; x < clAss\_num; x++)//按学号在选课表里找选课的项目

{

if (clAss\_index[x].key == kkk)

{

for (int j = 0; j < course\_num; j++)//按照选课表中的项目的课程号，在课程表中找到课程名

{

if (course\_index[j].key == clAss\_index[x].body->id)//都找到之后输出课程名和成绩

{

cout << course\_index[j].body->course\_name << " " << clAss\_index[x].body->scores << endl;

total\_s+= clAss\_index[x].body->scores;

total\_n++;//累加求平均值

}

}

}

}

cout << "平均成绩为：" << total\_s / total\_n << endl;//平均值

}

int lll = 0;

cout << "输入1返回主界面：";

cin >> lll;

if (lll == 1)goto goon;

else break;

}

if (Flag == 5)//按照课程查成绩

{

string cl\_name; int kkk, total\_s = 0,total\_n=0;

cout << "请输入课程的名称：";

cin >> cl\_name;

int d = 0;

for (d; d < course\_num; d++)//在课程表索引中找

{

if (course\_index[d].body->course\_name == cl\_name)//找到该课程并获取课程号

{

kkk = course\_index[d].key;

break;

}

}

if (d == course\_num)//找不到该课程

{

cout << "该课程不存在"<<endl;

goto goon;

}

else//下面输出课程成绩

{

for (int x = 0; x < clAss\_num; x++)//选课表里找该课程号对应的项目

{

if (clAss\_index[x].body->id == kkk)

{

for (int j = 0; j < student\_num; j++)//找到该项目对应的学生的姓名

{

if (student\_index[j].S\_key == clAss\_index[x].key)//输出相应内容

{

cout << student\_index[j].S\_name << " " << clAss\_index[x].body->scores << endl;

total\_s += clAss\_index[x].body->scores;

total\_n++;//累加

}

}

}

}

cout << "平均成绩为：" << total\_s / total\_n << endl;//平均数

}

int lll = 0;

cout << "输入1返回主界面：";

cin >> lll;

if (lll == 1)goto goon;

else break;

}

if (Flag == 6)//导出到文件

{

ofstream out("out.txt");//打开文件

out << "学生表" << endl;

out << "学号 姓名 性别" << endl;

student\* st;

Shellsort\_student\_index\_table(student\_index, student\_num);//学生表索引排序

for (int i = 0; i < student\_num; i++)//输出学生表

{

out << student\_index[i].S\_key << " " << student\_index[i].S\_name << " ";

st = find(student\_index[i].S\_key, bt);

if (st != NULL)out << st->size << endl;

}

out << endl;

out << "课程表" << endl;

out << "课程号 课程名 学分" << endl;

Shellsort\_course\_index\_table(course\_index, course\_num);//课程表索引排序

course\* p = NULL;

while (course\_head != NULL)//输出课程表

{

p = course\_head;

out << p->id << " " << p->course\_name << " " << p->credits << endl;

course\_head = course\_head->next;

}

out << endl;

out << "选课表" << endl;

out << "学号 课程号 分数" << endl;

Shellsort\_clAss\_index\_table(clAss\_index, clAss\_num);//选课表索引排序

clAss\* q = NULL;

while (clAss\_head != NULL)//输出选课表

{

q = clAss\_head;

out << q->key << " " << q->id << " " << q->scores << endl;

clAss\_head = clAss\_head->next;

}

cout << "已退出";

out.close();

break;

}

}

}