《数据结构》上机报告

2022 年 9 月 20 日

姓名：郑博远　学号：2154312　班级：计科 1 班　得分：

<table>
<tr><td>实验题目</td><td colspan="2">运用线性表实现学生选课系统</td></tr>
<tr><td>实验目的</td><td colspan="2">1. 掌握线性表的逻辑结构特性，和基本操作，给出线性表的抽象数据类型定义；
2. 掌握线性表的顺序存储结构（顺序表）的定义和实现。
3. 掌握线性表的链式存储结构（链表）的定义和实现。
4. 从时间复杂度和空间复杂度的角度综合比较线性表两种存储结构的不同特点及其适用场合。</td></tr>
<tr><td>问题描述</td><td colspan="2">为一所拥有接近 40000 名学生和 3000 门课程的大学，生成一个选课系统；实现顺序表和链表两种通用数据结构，为该题目设计出具体的高效的数据结构。</td></tr>
<tr><td>基本要求</td><td colspan="2">1. 能够输入所有学生信息（学号，姓名，性别，…）；
2. 能够输入所有课程信息（课号，课名，学分，…）；
3. 能够查找、插入、删除学生记录；
4. 能够查找、插入、删除课程记录；
5. 能够输入学生选课信息，例如给定（学号 a，课号 b），就表示学生 a 注册了课程 b；
6. 能够输出某门课程的所有选课学生的名单，包含学生所有信息（学号、姓名、性别…）；
7. 能够输出某位学生的所有选课课程清单，包含课程的所有信息（课号、课名、学分…）；</td></tr>
<tr><td rowspan="2">选做要求</td><td colspan="2"></td></tr>
<tr><td>已完成选做内容（序号）</td><td></td></tr>
</table>

数据结构设计	学校的选课系统信息量较大，学生数量、课程数量都较多，且学生信息与课程信息的存储涉及到不断的插入、删除操作，因此我希望能尽可能的降低时间复杂度。我选择了哈希表的方式存储学生信息与课程信息；存放在哈希表中下标相同的信息用链表记录。对于打印某学生的所有选课或某课程的所有选课学生这一功能，我选择将每个学生的选课课号或每门课程的学生学号存入对应元素的数据中；考虑到这些数据量不算太大且没有对频繁指定位置插入、删除的需求，选择用顺序表记录信息。最后，使用模板类来封装顺序表、链表与哈希表，使得学生信息与课程信息能够方便地共用代码，使程序更为简洁。以下是数据结构设计介绍： `/* 学生信息 */` `struct StudentInfo{` `  long long ID;  //学号` `  char name[32];  //姓名` `  char gender;  //性别 F/M` `  SqList<long long> courses;  //选课的课号` `};` `/* 课程信息 */` `struct CourseInfo {` `  long long ID;  //课号` `  char name[32];  //课名` `  float credit;  //学分` `  SqList<long long> students;  //所选学生学号` `};` `/* 顺序表 用于存储每个学生的选课信息 课程的学生信息*/` `template <class Sqlist_ElemType>` `class SqList {` `  SqList_ElemType* base;  //头指针` `  int length;  //表长` `  int listsize;  //申请的表空间` `}` `/* 链表节点 */` `template <class LinkList_ElemType>` `class LNode {` `public:` `  LinkList_ElemType data;  //存放的数据`

	`` LNode* next; //直接后继的指针 }; /* 链表 用于存储学生信息 与 课程信息 */ template <class LinkList_ElemType> class LinkList { LNode <LinkList_ElemType>* base; } /* 哈希表 用于存储学生信息 与 课程信息 */ template <class Hash_ElemType> class Hashtable { LinkList<Hash_ElemType>* base; } /* 选课系统 */ class EduSystem { HashTable<StudentInfo> StudentList; //学生列表 HashTable<CourseInfo> CourseList; //课程列表 int StudentNum, CourseNum; //学生、课程数量 } ```
功能(函数)说明	1. 顺序表类的成员函数 ``` /** * @brief 顺序表的建立 */ template <class SqList_ElemType> SqList<SqList_ElemType>::SqList() /** * @brief 顺序表的销毁 */ template <class SqList_ElemType> SqList<SqList_ElemType>::~SqList() /** * @brief 重载= 目的是在复制线性表时不浅拷贝 防止反复delete * @param that 赋值的另一个顺序表 */ template <class SqList_ElemType> const SqList<SqList_ElemType>& SqList<SqList_ElemType>::SqList::operator=(const SqList<SqList_ElemType>& that) /** * @brief 求顺序表表长 * @tparam SqList_ElemType 顺序表元素 * @return 表长 */ ```

```
template <class SqList_ElemType>
int SqList<SqList_ElemType>::ListLength()

/**
 * @brief 在顺序表位置i处插入元素
 * @param i 插入位置i
 * @param e 欲插入的元素
*/
template <class SqList_ElemType>
Status SqList<SqList_ElemType>::ListInsert(int i, SqList_
ElemType e)

/**
 * @brief 取出顺序表位置i的元素
 * @param i 元素位置i
 * @param e 取出的元素
*/
template <class SqList_ElemType>
Status SqList<SqList_ElemType>::GetElem(int i, SqList_Ele
mType& e)

/**
 * @brief 查找元素是否在顺序表中
 * @tparam SqList_ElemType 顺序表元素
 * @param e 查找的元素
 * @return 是否在表中
*/
template <class SqList_ElemType>
  bool SqList<SqList_ElemType>::IsInList(const SqList_Elem
Type& e)
```

2. 链表类的成员函数

```
/**
 * @brief 建立带头节点的空链表
*/
template <class LinkList_ElemType>
LinkList<LinkList_ElemType>::LinkList()

/**
 * @brief 链表的销毁
*/
template <class LinkList_ElemType>

/**
 * @brief 向链表中位置i处插入元素
 * @param i 位置i
 * @param e 插入的元素
```

```
*/
template <class LinkList_ElemType>
Status LinkList<LinkList_ElemType>::ListInsert(int i, const
LinkList_ElemType& e)

/**
 * @brief 在链表中位置i处删除元素
 * @param i 删除位置i
 * @param e 取出删除的元素
*/
template <class LinkList_ElemType>
Status LinkList<LinkList_ElemType>::ListDelete(int i, Lin
kList_ElemType& e)

/**
 * @brief 取出链表中位置i处的元素
 * @param i 元素的位置i
 * @param e 取出的元素
*/
template <class LinkList_ElemType>
Status LinkList<LinkList_ElemType>::GetElem(int i, LinkLi
st_ElemType& e)

/**
 * @brief 取出链表中相应元素的地址
 * @param e 欲取出地址的元素
 * @param ep 取出的元素地址
*/
template <class LinkList_ElemType>
Status  LinkList<LinkList_ElemType>::LocatePosition(const
LinkList_ElemType& e, LinkList_ElemType*& ep)

/**
 * @brief 返回第一个链表中该元素的index 若不存在返回NULL
 * @param e 查找的元素
*/
template <class LinkList_ElemType>
int         LinkList<LinkList_ElemType>::LocateElem(const
LinkList_ElemType& e)

/**
 * @brief 找元素的详细信息(通过重载=通过某个结构体成员变量找到对应
元素 直接取出所有信息)
 * @param e 查找的元素
*/
template <class LinkList_ElemType>
Status LinkList<LinkList_ElemType>::SearchElem(LinkList
```

```
_ElemType& e)
```

3．哈希表类的成员函数

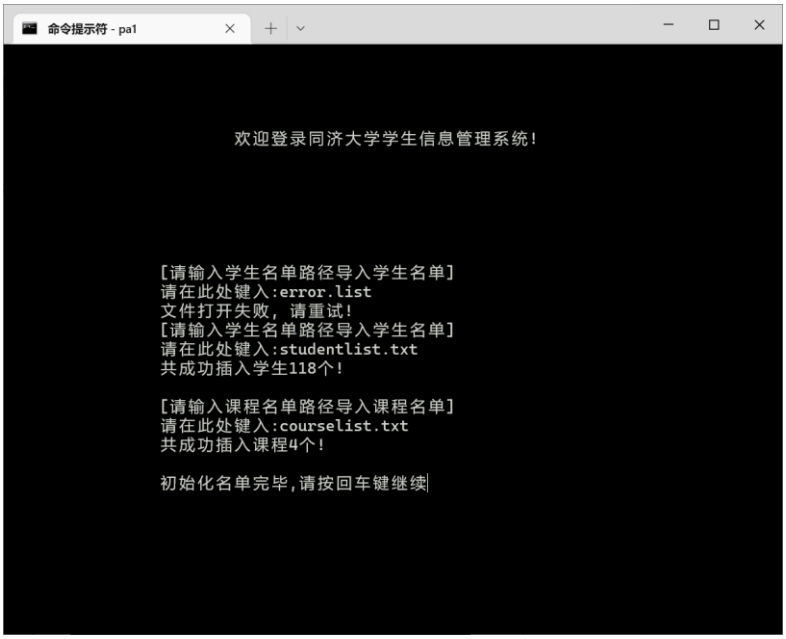
```
/**
 * @brief 空哈希表的建立
*/
template <class Hash_ElemType>
Hashtable<Hash_ElemType>::Hashtable()

/**
 * @brief 哈希表的销毁
*/
template <class Hash_ElemType>
Hashtable<Hash_ElemType>::~Hashtable()

/**
 * @brief 在哈希表中插入新元素
 * @param e 当前插入的元素
 * @param GetIndex 取得key值下标的函数
*/
template <class Hash_ElemType>
Status Hashtable<Hash_ElemType>::TableInsert(const
Hash_ElemType& e, int(*GetIndex)(const Hash_ElemType& e))

/**
 * @brief 在哈希表中查找已有元素，返回地址以修改
 * @param e 当前查找的元素
 * @param ep 找到的元素地址
 * @param GetIndex 取得key值下标的函数
*/
template <class Hash_ElemType>
Status Hashtable<Hash_ElemType>::LocateElem(const
Hash_ElemType& e, Hash_ElemType*& ep,

/**
 * @brief 在哈希表中查找已有元素
 * @param e 当前查找的元素
 * @param GetIndex 取得key值下标的函数
*/
template <class Hash_ElemType>
Status
Hashtable<Hash_ElemType>::TableSearch(Hash_ElemType& e,
int(*GetIndex)(const Hash_ElemType& e))
```

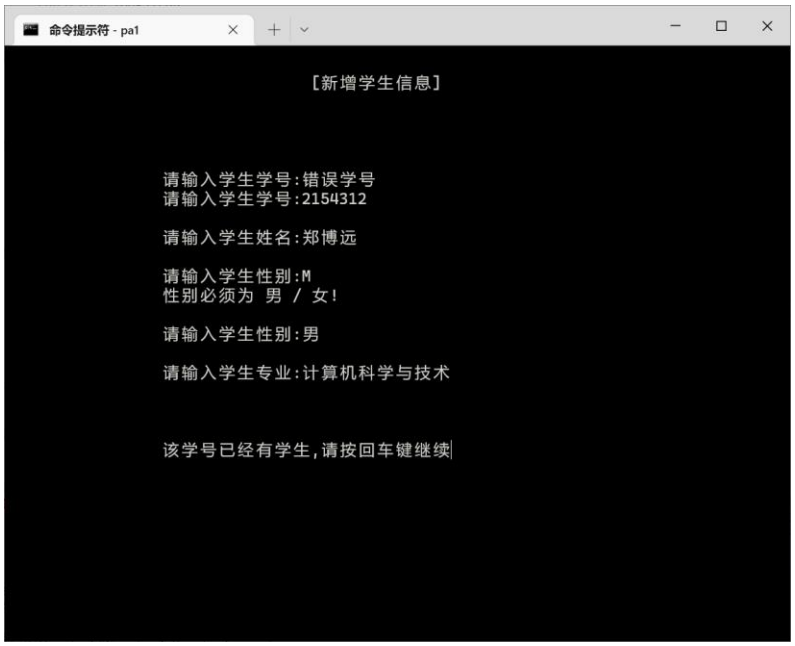
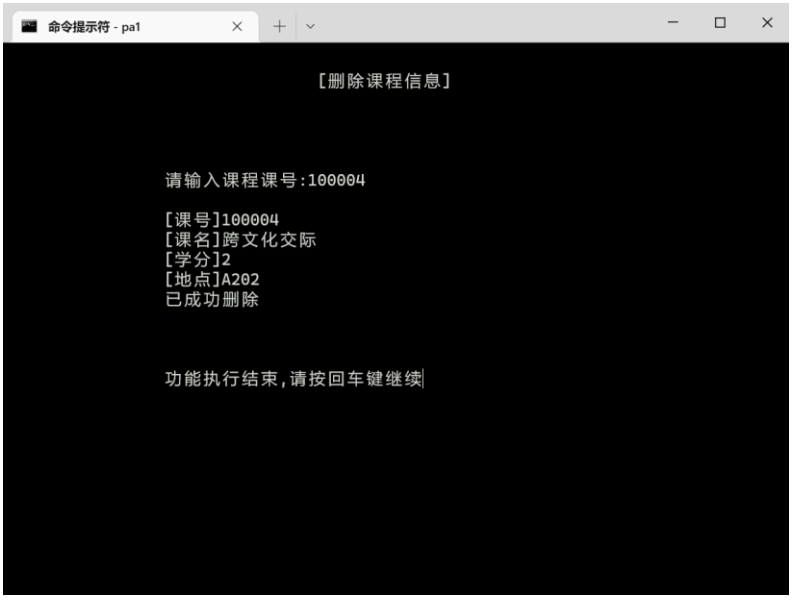
```
 /**
  * @brief 在哈希表中删除已有元素
  * @param e 准备删除的元素
  * @param GetIndex 取得key值下标的函数
 */
 template <class Hash_ElemType>
 Status
Hashtable<Hash_ElemType>::TableDelete(Hash_ElemType& e,
int(*GetIndex)(const Hash_ElemType& e))

 /**
  * @brief 在哈希表中查找元素是否存在
  * @param e 当前查找的元素
  * @param GetIndex 取得key值下标的函数
 */
 template <class Hash_ElemType>
 bool Hashtable<Hash_ElemType>::IsInTable(const
Hash_ElemType& e, int(*GetIndex)(const Hash_ElemType& e))
```

4. 选课系统类的成员函数

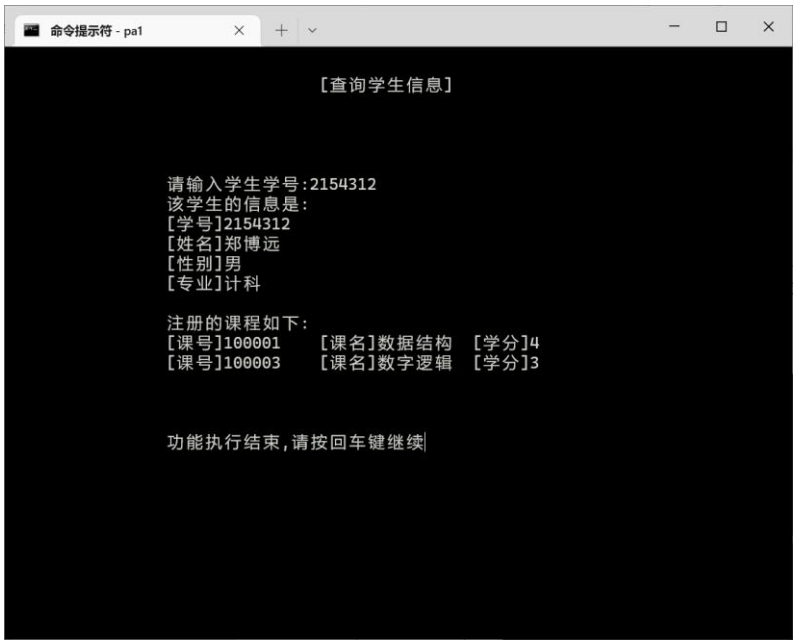
```
 /**
  * @brief 取系统学生数量
 */
 int EduSystem::GetStudentNum()

 /**
  * @brief 取系统课程数量
 */
 int EduSystem::GetCourseNum()

 /**
  * @brief 初始化学生名单
  * @param in 输入流
 */
 Status EduSystem::InitStudentList(istream& in)

 /**
  * @brief 初始化课程名单
  * @param in 输入流
 */
 Status EduSystem::InitCourseList(istream& in)
 /**
  * @brief 插入新学生信息
```

```
 * @param e 学生信息
*/
Status EduSystem::InsertStudent(const StudentInfo& e)

/**
 * @brief 插入新课程信息
 * @param e 课程信息
*/
Status EduSystem::InsertCourse(const CourseInfo& e)

/**
 * @brief 删除学生信息
 * @param e 删除的学生信息
*/
Status EduSystem::DeleteStudent(StudentInfo& e)

/**
 * @brief 删除课程信息
 * @param e 删除的课程信息
*/
Status EduSystem::DeleteCourse(CourseInfo& e)

/**
 * @brief 查找学生信息
 * @param e 学生信息(传入时通过学号查找, 传出时用于打印信息)
 * @param out 输出流
*/
Status EduSystem::SearchStudent(StudentInfo& e,
ostringstream& out)

/**
 * @brief 查找课程信息
 * @param e 课程信息(传入时通过课号查找, 传出时用于打印信息)
 * @param out 输出流
*/
Status EduSystem::SearchCourse(CourseInfo& e,
ostringstream& out)

/**
 * @brief 学生注册新课程
 * @param ec 学生信息
 * @param es 课程信息
*/
Status EduSystem::Register(CourseInfo& ec, StudentInfo& es)
```

界面设计和使用说明	程序开始，考虑到学生系统初始化有大量信息，从文件初始化学生信息与课程信息。文本文件的格式为“学号 姓名 性别 专业”（学生）或“课号 课名 学分 地点”（课程）。若文件路径错误无法打开，程序能够给出错误提示直至输入正确。界面如下：  初始化完毕后，程序进入功能选择界面如下：  输入对应数字选择具体功能，若数字非法则重复读入。 选择数字 1，进入新增学生功能；选择数字 2，进入新增课程功能（与前者界面类似）。程序能够对错误输入给出提示（如：学

	号是非数字会重复读入直至正确；性别输入非“男”或“女”，给出相应错误提示并重新读入），若插入的学号在表中已经存在则给出信息如下：  选择数字 4，进入删除课程功能；选择数字 3，进入删除学生功能（与前者界面类似）。若课号存在，则给出成功删除提示如下；若课号不存在，则给出课程不存在的错误提示。  选择数字 5，进入查询学生信息功能；选择数字 6，进入查询课程信息功能（与前者界面类似）。若学号存在，则给出学号、性

别、专业等个人信息，并打印其注册的所有课程及对应信息；若学号不存在，则给出学生不存在的错误提示。



选择数字 7，进入课程注册功能。分别输入注册的学生学号与注册的课程课号，进行课程注册。若学生学号或课程课号不存在，程序给出错误提示；若两者均存在，则给出注册成功提示。

若该学生已经注册过该课程，能够给出错误提示信息“该学生已注册过该课程”如下图所示：

	 选择数字 0，退出选课系统。
调试分析	1. 在链表 GetElem、ListDel 时，程序经常在运行中卡死。原因是：链表中的信息元素中包含存储选课信息、学生信息的顺序表的头指针 base。在没有对顺序表的类做“=”重载时，由于 GetElem、DelElem 函数中均包含“=”的赋值，并通过引用的传递将获取到的元素传给函数体外的临时变量，此时 base 指针也被直接复制给函数体外的临时变量；当临时变量在函数体外被析构时，首先会将 base 指针 delete 一次，这会导致此后已经被 delete 过的 base 指针被反复 delete，使程序卡死报错； 2. 链表的 ListInsert 函数会在运行时卡死，其原因与此前类似。这是由于 ListInsert 函数在传入参数时，会调用系统的复制构造函数，此时将 base 指针复制给函数中的临时变量；函数执行完毕后，函数体中的临时变量和体外的变量分别会进行两次析构，导致 base 指针被两次 delete，使程序出错。解决方法是，在函数的形参部分使用常引用，就可以避免调用构造函数导致出错； 3. 在学生注册课程操作的实现时，原先采用的是 GetElem 取出对应学生或课程元素，在其中的顺序表中进行 ListInsert；这样会导致元素的插入在输出信息操作时全部不显示，即注册操作并不

	成功。这是因为取出的元素是链表中元素的拷贝（在前两点的调试中重载了“=”运算符让 base 指针不同，而复制其指向的具体内容），因此给取出后的元素的线性表 ListInsert 并不会改变链表中元素的线性表。为解决此操作，添加了函数 LocatePosition，通过形参返回对应元素数据指针的引用，通过指针进行修改。
心得体会	本次实验中，我尝试用 C++方式实现顺序表与链表，将操作其的函数都封装在类中，使得在不同数据的应用场景中都能方便使用。由于学生、课程信息的链表有大部分重合，我将学生信息和课程信息分别构建结构体，用模板类的方式共用一个链表类与哈希表类，使代码更简洁明了。 在调试的过程中，我对链表与顺序表的构建以及其各个成员函数的写法、使用都比较熟练，说明对线性表结构的相关知识掌握还算不错。在涉及到构造、析构、动态内存申请、模板类等部分知识有所遗忘，这次实验起到了很好的温习作用。 起初，我单纯使用链表这一数据结构来存储学生信息与课程信息；在此情况下，学生、课程信息的插入、删除、查询的时间复杂度均为 O(n)，学生注册新课程的时间复杂度为 O(nm)。但考虑到学生选课系统较大的信息量，我对存储方式进行了改进。我使用哈希表这一特殊的线性表结构来存储信息，将学号（课号）模一个较大的质数作为数组下标；当多个元素下标相同时，再采用链表的方式存储。通过这样的方式，每个数组下标对应的链表长度都极短，插入、删除、查询的时间复杂度都可以近似为 O(1)，可以通过调整较大质数的值获得比较理想的效果。

附.完整代码

1.顺序表类的源文件(sqlist.hpp)

```cpp
#pragma once
#include <iostream>
#include <cstring>
#define LIST_INIT_SIZE  50
#define LISTINCREMENT   10
typedef int Status;
#define LOVERFLOW-1
#define ERROR    -2
#define OK       0
using namespace std;

template <class SqList_ElemType>
/* 顺序表 */
class SqList {
    SqList_ElemType* base;      //头指针
    int length;                 //表长
    int listsize;               //申请的表空间
public:
    //顺序表的建立
    SqList();
    //顺序表的销毁
    ~SqList();
    //重载等于号 目的是在复制顺序表时不仅做浅拷贝 防止反复delete
    const SqList& operator=(const SqList& that);
    //求表长
    int ListLength();
    //插入元素
    Status ListInsert(int i, SqList_ElemType e);
    //取表中元素
    Status GetElem(int i, SqList_ElemType& e);
    //查找元素是否在顺序表中
    bool IsInList(const SqList_ElemType& e);
};

/**
 * @brief 顺序表的建立
 * @tparam SqList_ElemType 顺序表元素
*/
template <class SqList_ElemType>
```

```
SqList<SqList_ElemType>::SqList()
{
    this->base = new(nothrow) SqList_ElemType[LIST_INIT_SIZE];
    if (!this->base)
        exit(LOVERFLOW);
    this->length = 0;
    this->listsize = LIST_INIT_SIZE;
}

/**
 * @brief 顺序表的销毁
 * @tparam SqList_ElemType 顺序表元素
*/
template <class SqList_ElemType>
SqList<SqList_ElemType>::~SqList()
{
    if (this->base)
        delete[] this->base;
    this->base = NULL;
    this->length = 0;
    this->listsize = 0;

}

/**
 * @brief 重载等于号 目的是在复制线性表时不仅做浅拷贝 防止反复delete
 * @tparam SqList_ElemType 顺序表元素
 * @param that 赋值的另一个顺序表
 * @return 赋值后结果
*/
template <class SqList_ElemType>
const SqList<SqList_ElemType>&
SqList<SqList_ElemType>::SqList::operator=(const
SqList<SqList_ElemType>& that)
{
    length = that.length;
    listsize = that.listsize;
    base = new(nothrow) SqList_ElemType[listsize];
    if (!base)
        exit(LOVERFLOW);
    memcpy(base, that.base, listsize * sizeof(SqList_ElemType));
    return *this;
}
```

```
/**
 * @brief 求顺序表表长
 * @tparam SqList_ElemType 顺序表元素
 * @return 表长
*/
template <class SqList_ElemType>
int SqList<SqList_ElemType>::ListLength()
{
    return this->length;
}

/**
 * @brief 在顺序表位置i处插入元素
 * @tparam SqList_ElemType 顺序表元素
 * @param i 插入位置i
 * @param e 欲插入的元素
 * @return 插入状态
*/
template <class SqList_ElemType>
Status SqList<SqList_ElemType>::ListInsert(int i, SqList_ElemType e)
{
    if (i < 1 || i > this->length + 1)
        return ERROR;
    if (this->length >= this->listsize) {
        SqList_ElemType* newbase = new SqList_ElemType[this->length +
LISTINCREMENT];
        if (!newbase)
            exit(LOVERFLOW);
        memcpy(newbase, this->base, this->length *
sizeof(SqList_ElemType));
        delete[] this->base;
        this->base = newbase;
        this->listsize += LISTINCREMENT;
    }
    SqList_ElemType* q = &(this->base[i - 1]);
    for (SqList_ElemType* p = &(this->base[this->length - 1]); p >= q;
--p)
        *(p + 1) = *p;
    *q = e;
    ++this->length;
    return OK;
}

/**
```

```
 * @brief 取出顺序表位置i的元素
 * @tparam SqList_ElemType 顺序表元素
 * @param i 元素位置i
 * @param e 取出的元素
 * @return 取出状态
*/
template <class SqList_ElemType>
Status SqList<SqList_ElemType>::GetElem(int i, SqList_ElemType& e)
{
    if (i < 1 || i > this->length)
        return ERROR;
    e = this->base[i - 1];
    return OK;
}

/**
 * @brief 查找元素是否在顺序表中
 * @tparam SqList_ElemType 顺序表元素
 * @param e 查找的元素
 * @return 是否在表中
*/
template <class SqList_ElemType>
bool SqList<SqList_ElemType>::IsInList(const SqList_ElemType& e)
{
    for (int i = 0; i < length; i++) {
        if (base[i] == e)
            return true;
    }
    return false;
}
```

2.链表类的源文件(linklist.hpp)

```
#pragma once
typedef int Status;
#define LOVERFLOW-1
#define ERROR    -2
#define OK       0
using namespace std;

/* 链表节点 */
template <class LinkList_ElemType>
class LNode {
public:
```

```
    LinkList_ElemType data;     //存放的数据
    LNode* next;              //直接后继的指针
};

/* 链表 */
template <class LinkList_ElemType>
class LinkList {
    LNode <LinkList_ElemType>* base;
    typedef LNode<LinkList_ElemType> LNode_;
    typedef LNode<LinkList_ElemType>* LNodep;
public:
    //建立带头节点的空链表
    LinkList();
    //表的销毁
    ~LinkList();
    //插入元素
    Status ListInsert(int i, const LinkList_ElemType& e);
    //删除元素
    Status ListDelete(int i, LinkList_ElemType& e);
    //取元素
    Status GetElem(int i, LinkList_ElemType& e);
    //取元素地址
    Status LocatePosition(const LinkList_ElemType& e,
LinkList_ElemType*& ep);
    //返回第一个链表中该元素的index 若不存在返回NULL
    int LocateElem(const LinkList_ElemType& e);
    //找到第一个链表中的该元素，并替换(重载元素=后可以用于查找详细信息)
    Status SearchElem(LinkList_ElemType& e);
};

/**
 * @brief 建立带头节点的空链表
 * @tparam LinkList_ElemType 链表元素
*/
template <class LinkList_ElemType>
LinkList<LinkList_ElemType>::LinkList()
{
    this->base = new(nothrow) LNode_;
    if (!base)
        exit(LOVERFLOW);
    base->next = NULL;   //带头节点的空链表
}

/**
```

```
 * @brief 链表的销毁
 * @tparam LinkList_ElemType 链表元素
*/
template <class LinkList_ElemType>
LinkList<LinkList_ElemType>::~LinkList()
{
    LNodep p = base, q;
    while (p) {
        q = p->next;
        delete p;
        p = q;
    }

}

/**
 * @brief 向链表中位置i处插入元素
 * @tparam LinkList_ElemType 链表元素
 * @param i 位置i
 * @param e 插入的元素
 * @return 插入状态
*/
template <class LinkList_ElemType>
Status LinkList<LinkList_ElemType>::ListInsert(int i, const
LinkList_ElemType& e)
{
    LNodep p = base;
    int j = 0;
    while (p && j < i - 1) {
        p = p->next;
        ++j;
    }
    if (!p || j > i - 1)
        return ERROR;
    LNodep s = new(nothrow)LNode_;
    if (!s)
        return LOVERFLOW;
    s->data = e;
    s->next = p->next;
    p->next = s;
    return OK;
}
```

```
/**
 * @brief 在链表中位置i处删除元素
 * @tparam LinkList_ElemType 链表元素
 * @param i 删除位置i
 * @param e 取出删除的元素
 * @return 删除状态
*/
template <class LinkList_ElemType>
Status LinkList<LinkList_ElemType>::ListDelete(int i,
LinkList_ElemType& e)
{
    LNodep p = base, q;
    int j = 0;
    while (p->next && j < i - 1) {
        p = p->next;
        ++j;
    }
    if (!(p->next) || j > i - 1)
        return ERROR;
    q = p->next;
    p->next = q->next;
    e = q->data;
    delete q;
    return OK;
}

/**
 * @brief 取出链表中位置i处的元素
 * @tparam LinkList_ElemType 链表元素
 * @param i 元素的位置i
 * @param e 取出的元素
 * @return 取出状态
*/
template <class LinkList_ElemType>
Status LinkList<LinkList_ElemType>::GetElem(int i, LinkList_ElemType&
e)
{
    LNodep p = base->next;
    int j = 1;
    while (p && j < i) {
        p = p->next;
        ++j;
    }
    if (!p || j > i)
```

```
        return ERROR;
    e = p->data;
    return OK;
}

/**
 * @brief 取出链表中相应元素的地址
 * @tparam LinkList_ElemType 链表元素
 * @param e 欲取出地址的元素
 * @param ep 取出的元素地址
 * @return 取出状态
*/
template <class LinkList_ElemType>
Status LinkList<LinkList_ElemType>::LocatePosition(const
LinkList_ElemType& e, LinkList_ElemType*& ep)
{
    LNodep p = base->next;
    while (p) {
        if (p->data == e) {
            ep = &p->data;
            return OK;
        }
        p = p->next;
    }
    return ERROR;  //没有该元素
}

/**
 * @brief 返回第一个链表中该元素的index 若不存在返回NULL
 * @tparam LinkList_ElemType 链表元素
 * @param e 查找的元素
 * @return 查找状态
*/
template <class LinkList_ElemType>
int LinkList<LinkList_ElemType>::LocateElem(const LinkList_ElemType&
e)
{
    LNodep p = base;
    int j = 0;
    while (p->next) {
        p = p->next;
        ++j;
        if (p->data == e) {
            return j;
```

```
        }
    }
    return NULL;
}

/**
 * @brief 找元素的详细信息(通过重载=通过某个结构体成员变量找到对应元素 直接取出所有信息)
 * @tparam LinkList_ElemType 链表元素
 * @param e 查找的元素
 * @return 查找状态
*/
template <class LinkList_ElemType>
Status LinkList<LinkList_ElemType>::SearchElem(LinkList_ElemType& e)
{
    LNodep p = base->next;
    while (p) {
        if (p->data == e) {
            e = p->data;
            return OK;
        }
        p = p->next;
    }
    return ERROR;  //没有该元素


}
```

4. 哈希表类的源文件(hashtable.hpp)

```
#include <iostream>
#include "./sqlist.hpp"
#include "./linklist.hpp"
#define LOVERFLOW-1
#define ERROR    -2
#define OK       0
#define HASHTABLE_LEN 19997

using namespace std;

template <class Hash_ElemType>
class Hashtable {
private:
    LinkList<Hash_ElemType>* base;
public:
```

```
    //构造函数 哈希表的建立
    Hashtable();
    //析构函数 哈希表的销毁
    ~Hashtable();
    //在哈希表中插入新元素
    Status TableInsert(const Hash_ElemType& e, int(*GetIndex)(const
Hash_ElemType& e));
    //在哈希表中查找已有元素
    Status TableSearch(Hash_ElemType& e, int(*GetIndex)(const
Hash_ElemType& e));
    //在哈希表中删除已有元素
    Status TableDelete(Hash_ElemType& e, int(*GetIndex)(const
Hash_ElemType& e));
    //在哈希表中取元素的位置(以修改其中内容)
    Status LocateElem(const Hash_ElemType& e, Hash_ElemType*& ep,
int(*GetIndex)(const Hash_ElemType& e));
    //查找哈希表中是否有该元素
    bool IsInTable(const Hash_ElemType&e, int(*GetIndex)(const
Hash_ElemType& e));
};

/**
 * @brief 空哈希表的建立
 * @tparam Hash_ElemType 哈希表的元素类型
 * @param len 哈希表的长度
*/
template <class Hash_ElemType>
Hashtable<Hash_ElemType>::Hashtable()
{
    base = new(nothrow) LinkList<Hash_ElemType>[HASHTABLE_LEN];
    if (!base)
        exit(LOVERFLOW);
}

/**
 * @brief 哈希表的销毁
 * @tparam Hash_ElemType 哈希表的元素类型
*/
template <class Hash_ElemType>
Hashtable<Hash_ElemType>::~Hashtable()
{
    delete[] base;
}
```

```cpp
/**
 * @brief 在哈希表中插入新元素
 * @tparam Hash_ElemType 哈希表的元素类型
 * @param e 当前插入的元素
 * @param GetIndex 取得key值下标的函数
 * @return 是否成功插入
*/
template <class Hash_ElemType>
Status Hashtable<Hash_ElemType>::TableInsert(const Hash_ElemType& e,
int(*GetIndex)(const Hash_ElemType& e))
{
    int ret = GetIndex(e);
    if (ret >= HASHTABLE_LEN || ret < 0)
        return ERROR;
    return base[ret].ListInsert(1, e);
}

/**
 * @brief 在哈希表中查找已有元素，返回地址以修改
 * @tparam Hash_ElemType 哈希表的元素类型
 * @param e 当前查找的元素
 * @param ep 找到的元素地址
 * @param GetIndex 取得key值下标的函数
 * @return 是否成功查找
*/
template <class Hash_ElemType>
Status Hashtable<Hash_ElemType>::LocateElem(const Hash_ElemType& e,
Hash_ElemType*& ep, int(*GetIndex)(const Hash_ElemType& e))
{
    int ret = GetIndex(e);
    if (ret >= HASHTABLE_LEN || ret < 0)
        return ERROR;

    return base[ret].LocatePosition(e, ep);
}

/**
 * @brief 在哈希表中查找已有元素
 * @tparam Hash_ElemType 哈希表的元素类型
 * @param e 当前查找的元素
 * @param GetIndex 取得key值下标的函数
 * @return 是否成功查找
*/
```

```cpp
template <class Hash_ElemType>
Status Hashtable<Hash_ElemType>::TableSearch(Hash_ElemType& e,
int(*GetIndex)(const Hash_ElemType& e))
{
    int ret = GetIndex(e);
    if (ret >= HASHTABLE_LEN || ret < 0)
        return ERROR;

    return base[ret].SearchElem(e);
}

/**
 * @brief 在哈希表中删除已有元素
 * @tparam Hash_ElemType 哈希表的元素类型
 * @param e 准备删除的元素
 * @param GetIndex 取得key值下标的函数
 * @return 元素本身是否存在
*/
template <class Hash_ElemType>
Status Hashtable<Hash_ElemType>::TableDelete(Hash_ElemType& e,
int(*GetIndex)(const Hash_ElemType& e))
{
    int ret = GetIndex(e);
    if (ret >= HASHTABLE_LEN || ret < 0)
        return ERROR;

    int index = base[ret].LocateElem(e);
    //表中不存在该元素
    if (!index)
        return ERROR;

    return base[ret].ListDelete(index, e);
}

/**
 * @brief 在哈希表中查找元素是否存在
 * @tparam Hash_ElemType 哈希表的元素类型
 * @param e 当前查找的元素
 * @param GetIndex 取得key值下标的函数
 * @return 是否存在
*/
template <class Hash_ElemType>
bool Hashtable<Hash_ElemType>::IsInTable(const Hash_ElemType& e,
int(*GetIndex)(const Hash_ElemType& e))
```

```
{
    int ret = GetIndex(e);
    if (ret >= HASHTABLE_LEN || ret < 0)
        return false;

    int index = base[ret].LocateElem(e);
    return index;     //是否存在对应的链表中
}
```

4. 选课系统类的头文件(system.h)

```
#pragma once
#include <iostream>
#include <sstream>
#include <cstring>
#include "./hashtable.hpp"
#define HashLength 19997
#define NO_STUDENT 10
#define NO_COURSE 11
#define ALREADY_REGISTER 12
using namespace std;

/* 学生信息 */
struct StudentInfo {
    long long ID;                   //学号
    char name[32];                  //姓名
    char major[64];                 //专业
    char gender;                    //性别 F/M
    SqList<long long> courses;      //选课(存课号)
    /* 用于查找学生信息 学号匹配即可 */
    bool operator==(const StudentInfo& that) const
    {
        return ID == that.ID;
    }
};

/* 课程信息 */
struct CourseInfo {
    long long ID;                   //课号
    char name[32];                  //课名
    float credit;                   //学分
    char venue[64];                 //上课地点
    SqList<long long> students;     //所选学生(存学号)
    /* 用于查找课程信息 课号匹配即可 */
```

```
    bool operator==(const CourseInfo& that) const
    {
        return ID == that.ID;
    }
};

/* 选课系统 */
class EduSystem {
    Hashtable<StudentInfo> StudentList;       //学生列表
    Hashtable<CourseInfo> CourseList;     //课程列表
    int StudentNum, CourseNum;            //学生、课程数量
public:
    //获取学生数量
    int GetStudentNum();
    //获取课程数量
    int GetCourseNum();
    //初始化学生列表
    Status InitStudentList(istream& in);
    //初始化课程列表
    Status InitCourseList(istream& in);
    //插入学生信息
    Status InsertStudent(const StudentInfo& e);
    //插入课程信息
    Status InsertCourse(const CourseInfo& e);
    //删除学生信息
    Status DeleteStudent(StudentInfo& e);
    //删除课程信息
    Status DeleteCourse(CourseInfo& e);
    //查询学生信息
    Status SearchStudent(StudentInfo& e, ostringstream& out);
    //查询课程信息
    Status SearchCourse(CourseInfo& e, ostringstream& out);
    //注册课程
    Status Register(CourseInfo& ec, StudentInfo& es);
};
```

5. 选课系统类的源文件(system.cpp)

```
#define _CRT_SECURE_NO_WARNINGS
#include "./system.h"
/**
 * @brief 计算在哈希表中对应下标
 * @param e 计算的元素
 * @return 下标值
```

```
*/
static int GetIndex(const CourseInfo& e)
{
    return e.ID % HashLength;
}

/**
 * @brief 计算在哈希表中对应下标
 * @param e 计算的元素
 * @return 下标值
*/
static int GetIndex(const StudentInfo& e)
{
    return e.ID % HashLength;
}

/**
 * @brief 取系统学生数量
 * @return 学生数量
*/
int EduSystem::GetStudentNum()
{
    return StudentNum;
}

/**
 * @brief 取系统课程数量
 * @return 课程数量
*/
int EduSystem::GetCourseNum()
{
    return CourseNum;
}

/**
 * @brief 初始化学生名单
 * @param in 输入流
 * @return 初始化状态
*/
Status EduSystem::InitStudentList(istream& in)
{
    StudentNum = 0;
    while (in.peek() != EOF) {
        StudentInfo tmp;
```

```
        char temp[20];
        in >> tmp.ID >> tmp.name >> temp >> tmp.major;
        if (strcmp(temp, "男") == 0)
            tmp.gender = 'M';
        else if (strcmp(temp, "女") == 0)
            tmp.gender = 'F';
        StudentList.TableInsert(tmp, &GetIndex);
        StudentNum++;
        in.get();
    }
    return OK;
}

/**
 * @brief 初始化课程名单
 * @param in 输入流
 * @return 初始化状态
*/
Status EduSystem::InitCourseList(istream& in)
{
    CourseNum = 0;
    while (in.peek() != EOF) {
        CourseInfo tmp;
        in >> tmp.ID >> tmp.name >> tmp.credit >> tmp.venue;
        CourseList.TableInsert(tmp, &GetIndex);
        CourseNum++;
        in.get();
    }
    return OK;
}


/**
 * @brief 插入新学生信息
 * @param e 学生信息
 * @return 插入状态
*/
Status EduSystem::InsertStudent(const StudentInfo& e)
{
    if (StudentList.IsInTable(e, &GetIndex))
        return ERROR;        //学号重复
    return StudentList.TableInsert(e, &GetIndex);
}
```

```
/**
 * @brief 插入新课程信息
 * @param e 课程信息
 * @return 插入状态
*/
Status EduSystem::InsertCourse(const CourseInfo& e)
{
    if (CourseList.IsInTable(e, &GetIndex))
        return ERROR;        //课号重复
    return CourseList.TableInsert(e, &GetIndex);
}


/**
 * @brief 删除学生信息
 * @param e 删除的学生信息
 * @return 删除状态
*/
Status EduSystem::DeleteStudent(StudentInfo& e)
{
    return StudentList.TableDelete(e, &GetIndex);
}


/**
 * @brief 删除课程信息
 * @param e 删除的课程信息
 * @return 删除状态
*/
Status EduSystem::DeleteCourse(CourseInfo& e)
{
    return CourseList.TableDelete(e, &GetIndex);
}


/**
 * @brief 查找学生信息
 * @param e 学生信息(传入时通过学号查找, 传出时用于打印信息)
 * @param out 输出流
 * @return 查找状态
*/
Status EduSystem::SearchStudent(StudentInfo& e, ostringstream& out)
{
    if (!StudentList.IsInTable(e, &GetIndex))
```

```
        return ERROR;
    else {
        StudentList.TableSearch(e, &GetIndex);
        for (int i = 1; i <= e.courses.ListLength(); i++) {
            CourseInfo c;
            //找到对应的课号
            e.courses.GetElem(i, c.ID);
            //找具体信息
            int pst = CourseList.TableSearch(c, &GetIndex);
            //可能已选的课程被删除了
            if (pst == OK) {
                out << "\t\t[课号]" << c.ID << "\t[课名]" << c.name << "\t[学
分]" << c.credit << endl;
            }
        }
    }
    return OK;
}


/**
 * @brief 查找课程信息
 * @param e 课程信息(传入时通过课号查找, 传出时用于打印信息)
 * @param out 输出流
 * @return 查找状态
*/
Status EduSystem::SearchCourse(CourseInfo& e, ostringstream& out)
{
    if (!CourseList.IsInTable(e, &GetIndex))
        return ERROR;
    else {
        CourseList.TableSearch(e, &GetIndex);
        for (int i = 1; i <= e.students.ListLength(); i++) {
            StudentInfo s;
            //找到对应的课号
            e.students.GetElem(i, s.ID);
            //找具体信息
            int pst = StudentList.TableSearch(s, &GetIndex);
            //可能已选的课程被删除了
            if (pst == OK) {
                out << "\t\t[学号]" << s.ID << "\t[姓名]" << s.name << "\t[性
别]" << (s.gender == 'M' ? "男" : "女") << "\t[专业]" << s.major << endl;
            }
        }
```

```
    }

    return OK;
}

/**
 * @brief 学生注册新课程
 * @param ec 学生信息
 * @param es 课程信息
 * @return 注册状态
*/
Status EduSystem::Register(CourseInfo& ec, StudentInfo& es)
{
    StudentInfo * studentp;
    CourseInfo * coursep;
    int ret1, ret2;
    ret1 = StudentList.LocateElem(es, studentp, &GetIndex);
    ret2 = CourseList.LocateElem(ec, coursep, &GetIndex);
    if (ret1 == ERROR)
        return NO_STUDENT;
    if (ret2 == ERROR)
        return NO_COURSE;
    if (studentp->courses.IsInList(coursep->ID))
        return ALREADY_REGISTER;
    studentp->courses.ListInsert(1, coursep->ID);
    coursep->students.ListInsert(1, studentp->ID);
    return OK;
}
```

5. 选课系统展示的主程序(main.cpp)

```
#define _CRT_SECURE_NO_WARNINGS
#include <iostream>
#include <sstream>
#include <fstream>
#include <cstring>
#include <conio.h>
#include <iomanip>
#include "./system.h"
using namespace std;

/**
 * @brief 等待回车输入
 * @param prompt 提示语
```

```
*/
void wait_for_enter(const char* prompt)
{
    cout << endl << prompt << ",请按回车键继续";
    while (getchar() != '\n')
        ;
    cout << endl << endl;
}

/**
 * @brief 初始化系统
 * @param tj 同济大学选课系统
*/
void InitSystem(EduSystem& tj)
{
    char FileDir[128];
    system("mode con: cols=83 lines=30");
    ifstream File;
    cout << endl << endl << endl << endl;
    cout << "\t\t\t欢迎登录同济大学学生信息管理系统!" << endl;
    cout << endl << endl << endl << endl << endl << endl;

    while (true) {
        cout << "\t\t[请输入学生名单路径导入学生名单]" << endl;
        cout << "\t\t请在此处键入:";
        cin >> FileDir;
        File.open(FileDir, ios::in);
        if (File.is_open())
            break;
        else
            cout << "\t\t文件打开失败, 请重试!" << endl;
    }
    tj.InitStudentList(File);
    cout << "\t\t共成功插入学生" << tj.GetStudentNum() << "个!" << endl <<
endl;
    File.close();

    while (true) {
        cout << "\t\t[请输入课程名单路径导入课程名单]" << endl;
        cout << "\t\t请在此处键入:";
        cin >> FileDir;
        File.open(FileDir, ios::in);
        if (File.is_open())
            break;
```

```
        else
            cout << "\t\t文件打开失败, 请重试!" << endl;
    }
    tj.InitCourseList(File);
    cout << "\t\t共成功插入课程" << tj.GetCourseNum() << "个!" << endl;

    getchar();
    wait_for_enter("\t\t初始化名单完毕");
}

/**
 * @brief 可视化菜单界面
 * @return 选择的菜单项
*/
int Menu()
{
    system("mode con: cols=83 lines=30");
    cout << endl << endl << endl << endl;
    cout << "\t\t\t\t同济大学选课系统" << endl;

    cout << endl << endl;
    cout << "\t\t\t\t    菜单选择" << endl;
    cout << '\t' << setw(65) << setfill('-') << "" << endl;
    cout << "\t|\t1\t|\t2\t|\t3\t|\t4\t|" << endl;
    cout << '\t' << setw(65) << setfill('-') << "" << endl;
    cout << "\t|\t\t|\t\t|\t\t|\t\t|" << endl;
    cout << "\t|   新增学生\t|    新增课程\t|    删除学生\t|    删除课程\t|" <<
endl;
    cout << "\t|\t\t|\t\t|\t\t|\t\t|" << endl;
    cout << '\t' << setw(65) << setfill('-') << "" << endl;
    cout << "\t|\t5\t|\t6\t|\t7\t|\t0\t|" << endl;
    cout << '\t' << setw(65) << setfill('-') << "" << endl;
    cout << "\t|\t\t|\t\t|\t\t|\t\t|" << endl;
    cout << "\t|   查询学生\t|    查询课程\t|    课程注册\t|    退出系统\t|" <<
endl;
    cout << "\t|\t\t|\t\t|\t\t|\t\t|" << endl;
    cout << '\t' << setw(65) << setfill('-') << "" << endl << endl << endl;
    cout << "\t\t\t    [请按对应数字选择功能]" << endl;
    cout << "\t\t\t\t\t";
    while (char ch = _getch())
        if (ch >= '0' && ch <= '7')
            return ch - '0';
    return 0;
}
```

```
int main()
{
    EduSystem TongjiEdu;
    InitSystem(TongjiEdu);
    int ret;
    while (ret = Menu())
    {
        StudentInfo tmps;
        CourseInfo tmpc;
        ostringstream out;
        switch (ret) {
            case 1:
                system("cls");
                char tmpt[20];
                cout << "\n\t\t\t\t[新增学生信息]" << endl;
                cout << endl << endl << endl << endl;
                while (true) {
                    cout << "\t\t请输入学生学号:";
                    cin >> tmps.ID;
                    if (cin.good())
                        break;
                    cin.clear();
                    cin.ignore(65536, '\n');
                }
                cout << "\n\t\t请输入学生姓名:";
                cin >> tmps.name;
                while (true) {
                    cout << "\n\t\t请输入学生性别:";
                    cin >> tmpt;
                    if (strcmp(tmpt, "男") == 0) {
                        tmps.gender = 'M';
                        break;
                    }
                    else if (strcmp(tmpt, "女") == 0) {
                        tmps.gender = 'F';
                        break;
                    }
                    cout << "\t\t性别必须为 男 / 女!" << endl;
                }
                cout << "\n\t\t请输入学生专业:";
                cin >> tmps.major;
                if (TongjiEdu.InsertStudent(tmps) == ERROR) {
                    getchar();
```

```
                wait_for_enter("\n\n\t\t该学号已经有学生");
            }
            else {
                getchar();
                wait_for_enter("\n\n\t\t学生信息录入完毕");
            }
            break;
        case 2:
            system("cls");
            cout << "\n\t\t\t\t[新增课程信息]" << endl;
            cout << endl << endl << endl << endl;
            while (true) {
                cout << "\t\t请输入课程课号:";
                cin >> tmpc.ID;
                if (cin.good())
                    break;
                cin.clear();
                cin.ignore(65536, '\n');
            }
            cout << "\n\t\t请输入课程课名:";
            cin >> tmpc.name;
            cout << "\n\t\t请输入课程学分:";
            cin >> tmpc.credit;
            cout << "\n\t\t请输入上课地点:";
            cin >> tmpc.venue;
            if (TongjiEdu.InsertCourse(tmpc) == ERROR) {
                getchar();
                wait_for_enter("\n\n\t\t该课号已经有课程");
            }
            else {
                getchar();
                wait_for_enter("\n\n\t\t课程信息录入完毕");
            }
            break;
        case 3:
            system("cls");
            cout << "\n\t\t\t\t[删除学生信息]" << endl;
            cout << endl << endl << endl << endl;
            while (true) {
                cout << "\t\t请输入学生学号:";
                cin >> tmps.ID;
                if (cin.good())
                    break;
                cin.clear();
```

```
                cin.ignore(65536, '\n');
            }
            if (TongjiEdu.DeleteStudent(tmps) == ERROR)
                cout << "\t\t不存在该学生!!!" << endl;
            else
                cout << "\t\t\n\t\t[学号]" << tmps.ID << "\n\t\t[姓名]"
<< tmps.name << "\n\t\t[性别]" << (tmps.gender == 'M' ? "男" : "女") <<
"\n\t\t[专业]" << tmps.major << "\n\t\t已成功删除" << endl;
            getchar();
            wait_for_enter("\n\n\t\t功能执行结束");
            break;
        case 4:
            system("cls");
            cout << "\n\t\t\t\t[删除课程信息]" << endl;
            cout << endl << endl << endl << endl;
            while (true) {
                cout << "\t\t请输入课程课号:";
                cin >> tmpc.ID;
                if (cin.good())
                    break;
                cin.clear();
                cin.ignore(65536, '\n');
            }
            if (TongjiEdu.DeleteCourse(tmpc) == ERROR)
                cout << "\t\t不存在该课程!!!" << endl;
            else
                cout << "\t\t\n\t\t[课号]" << tmpc.ID << "\n\t\t[课名]"
<< tmpc.name << "\n\t\t[学分]" << tmpc.credit << "\n\t\t[地点]" << tmpc.venue
<< "\n\t\t已成功删除" << endl;
            getchar();
            wait_for_enter("\n\n\t\t功能执行结束");
            break;
        case 5:
            system("cls");
            cout << "\n\t\t\t\t[查询学生信息]" << endl;
            cout << endl << endl << endl << endl;
            while (true) {
                cout << "\t\t请输入学生学号:";
                cin >> tmps.ID;
                if (cin.good())
                    break;
                cin.clear();
                cin.ignore(65536, '\n');
            }
```

```
                if (TongjiEdu.SearchStudent(tmps, out) == ERROR)
                    cout << "\t\t不存在该学生!!!" << endl;
                else
                    cout << "\t\t该学生的信息是:\n\t\t[学号]" << tmps.ID <<
"\n\t\t[姓名]" << tmps.name << "\n\t\t[性别]" << (tmps.gender == 'M' ? "
男" : "女") << "\n\t\t[专业]" << tmps.major << endl;
                cout << "\n\t\t注册的课程如下:\n" << out.str();
                getchar();
                wait_for_enter("\n\n\t\t功能执行结束");
                break;
            case 6:
                system("cls");
                cout << "\n\t\t\t\t[查询课程信息]" << endl;
                cout << endl << endl << endl << endl;
                while (true) {
                    cout << "\t\t请输入课程课号:";
                    cin >> tmpc.ID;
                    if (cin.good())
                        break;
                    cin.clear();
                    cin.ignore(65536, '\n');
                }
                if (TongjiEdu.SearchCourse(tmpc, out) == ERROR)
                    cout << "\t\t不存在该课程!!!" << endl;
                else
                    cout << "\t\t该课程的信息是:\n\t\t[课号]" << tmpc.ID <<
"\n\t\t[课名]" << tmpc.name << "\n\t\t[学分]" << tmpc.credit << "\n\t\t[地
点]" << tmpc.venue << endl;
                cout << "\n\t\t注册的学生如下:\n" << out.str();
                getchar();
                wait_for_enter("\n\n\t\t功能执行结束");
                break;
            case 7:
                system("cls");
                cout << "\n\t\t\t\t[学生注册课程]" << endl;
                cout << endl << endl << endl << endl;
                while (true) {
                    cout << "\t\t请输入学生学号:";
                    cin >> tmps.ID;
                    if (cin.good())
                        break;
                    cin.clear();
                    cin.ignore(65536, '\n');
                }
```

```
                cout << endl << endl << endl << endl;
                while (true) {
                    cout << "\t\t请输入课程课号:";
                    cin >> tmpc.ID;
                    if (cin.good())
                        break;
                    cin.clear();
                    cin.ignore(65536, '\n');
                }
                int ret = TongjiEdu.Register(tmpc, tmps);
                if (ret == NO_STUDENT)
                    cout << "\n\n\t\t学生不存在!" << endl;
                else if (ret == NO_COURSE)
                    cout << "\n\n\t\t课程不存在!" << endl;
                else if (ret == ALREADY_REGISTER)
                    cout << "\n\n\t\t该学生已注册过该课程" << endl;
                else
                    cout << "\n\n\t\t课程注册成功" << endl;
                getchar();
                wait_for_enter("\n\n\t\t功能执行结束");
                break;    }
    }

    return 0;
}
```