

# 作业 PA1 实验报告

---

## 0. 索引

- 涉及数据结构和相关背景
- 实验内容
  - 问题描述
  - 基本要求
  - 数据结构设计
  - 主要算法设计
  - 复杂度分析
  - 主要功能说明
  - 健壮性分析
  - 调试分析
- 预期实现与改进
- 实验总结
- 源代码

## 1. 涉及数据结构和相关背景

### 1.1 线性表

#### 1.1.1 顺序表

- 顺序表查询的速度很快（通过下标）
- 顺序表删除操作繁琐，需要不断移动数据位置保证没有空位
- 利用顺序表的索引可由其他数据映射，实现快速对应查找
- 依照顺序，规律来存，查询时会很方便

#### 1.1.2 链表

- 主要优点是删除时很方便，通过指针域变换可以实现
- 遍历或者寻找删除位置会消耗时间，所以需要尽量保证链表的长度比较小，删除才会有效率
- 也可以设置一个尾节点，来实现头插或者尾插
- 从头遍历到尾部进行尾插的一个优点是可以进行查重操作，保证健壮性
- 注意删除节点的顺序，保证后边的链表不会丢
- 注意头节点的存储

### 1.1.3 小规模数组模拟链表（此项目仅在哈希表中使用）

- 我们注意到，链表便于删除而线性表便于查找，在数据规模不是很大的情况下，看可以进行二者的结合
- 一个链表的节点具有数据域和指针域，则可以把数据域与指针域分别用两个线性表：数据表和指针表进行存储，相同的下标表示是一个节点
- 插入的元素直接存储在数据表的后边，需要改动的是指针表中的数值，这样可以完成 $O(1)$ 时间复杂度的插入操作， $O(1)$ 时间复杂度的删除操作

例：实现链表 head  $\rightarrow$  2  $\rightarrow$  5  $\rightarrow$  4  $\rightarrow$  1  $\rightarrow$  ^ (假设链接顺序为4 5 2 1)

	0	1	2	3	4
data	-1	4	5	2	1
pointer	3	4	1	2	10086

pointer存储的就是下一个元素的下标位置，这里用较大的数替换NULL

## 1.2 顺序存储与链式存储结合

### 1.2.1 字符串哈希

- 考虑到课程本身会有不同的长度以及可能会有非数字类型的字符穿插，利用字符串进行存储可以具有很好的适应性
- 注意到要求中会进行多次的查询操作，而且字符串的查询用朴素枚举的方法时间复杂度过高，可以利用字符串到数组下标的映射来实现字符串的快速匹配（字符串哈希）。

### 1.2.2 哈希冲突解决

- 利用拉链法来解决哈希冲突问题
- 这种方法中同时存在顺序表和链表。顺序表是用来与哈希值对应的，每一个哈希值对应一个内存单元。链表是用来存储相同哈希值的字符串的
- 如果哈希冲突发生，可以在记录头节点的顺序表后边链接链表，如果哈希函数足够均匀，每次映射之后进行查询不需要遍历链表很长时间

## 2.实验内容

### 2.1 问题描述

- 为接近40000名学生和3000门课程大学生成一个选课系统

## 2.2 基本要求

### 功能描述

- 输入所有学生信息（学号，姓名、性别...）
- 输入所有课程信息（课号，课名，学分...）
- 查找、插入、删除学生记录
- 查找、插入、删除课程记录
- 输入学生选课信息，给定学号a，课号b，表示学生a注册了课程b
- 输出某门课程的所有学生的选课名单，包含学生所有信息（学号、姓名、性别...）
- 输出某位学生的所有选课清单，包含课程的所有信息（课号、课名、学分...）
- 学生单独删除课程（退出课程）
- 课程将学生退课

## 2.3 数据结构设计

- 总体设计
  - 利用哈希表进行学生（或课程）实例的存储
  - 在每一个学生（或课程）的实例中存储学生（或课程）的基本信息
  - 在每一个学生（或课程）中存储其课程表链表对应的头节点，相当于每一个实例对应都会有一个课程表（选课名单）的链表
  - 链表中存储着一门课（一个学生）的id，了解具体信息可以通过查询此id解决
- 想法
  - 将名单查询表设置成哈希表：这样设计主要是考虑到在进行删除或者插入学生（课程）或者学生选课的过程中，需要频繁进行查找操作，所以将查找操作用哈希表进行优化之后，可以快速进行其他附加操作。
  - 将课程表和选课名单设置成链表：选课名单以及课程表会进行频繁的插入删除操作，应用链表进行操作相对时间可以节省一些
  - 在链表中存储id信息：原来将链表中存储的是一个学生（课程）的实例化对象，但是考虑到这样会产生很多重复的课程（学生）实例化对象，产生空间浪费，故采用存储id的方法，配合哈希表来对名单查询表查找，牺牲一点线性查找的时间，节约了很多空间
- 学生与课程抽象类
  - 因为在学生与课程中具有共同的属性（名称，序列号），以及相同的方法（删除，添加，查找元素）所以可以进行抽象，利用多态实现学生类与课程类

```

class abstract
{
public:
    virtual int deleteItem(string s) = 0; // 删除元素

    virtual int addItem(string s) = 0; // 插入元素

    virtual int find(string s) = 0; // 查询元素

    string name;
    string id = " ";

};

```

为方便查询，需要对id进行初始化

考虑到每一位学生要有一个课程表链表，所以在进行学生类的定义时同时需要对学生节点（也就是选课名单节点）和课程节点（课程表节点）进行定义

- 学生类
  - 在学生类中主要重写父类虚函数，以及学生特殊信息：课程表头节点和性别
  - 头文件

```

class Student : public abstract
{
public:
    Student(); // 无参构造
    Student(string id, string name, string gender); // 有参构造

    // 重写父类虚函数

    int deleteItem(string s);

    int addItem(string s);

    int find(string s);

    CurriculumNode* head; // 课程表头节点
    string gender; // 性别

};

```

- 有参构造与无参构造

```

Student::Student() {}

/*head指针申请内存*/
Student::Student(string id = " ", string name = " ", string gender = "
")
{
    this->id = id;
    this->name = name;
    this->gender = gender;
    this->head = (CurriculumNode*)malloc(sizeof(CurriculumNode));
    this->head->next = NULL;
}

```

注意给head头指针找空间赋初值

#### ◦ deleteItem()函数实现

```

int Student:: deleteItem(string s)
{
    /*
     * @param s : 学生学号
     */
    for (CurriculumNode* ptr = this->head; ptr->next != NULL; ptr = ptr-
>next)
    {
        //遍历目标节点的前一个节点
        if (ptr->next->id == s)
        {
            CurriculumNode* q = ptr->next;
            ptr->next = q->next;
            free(q);
            cout << setw(34) << "已删除" << endl;
            return 1;
        }
    }
    return 0;
}

```

注意内存的即时释放

#### ◦ addItem函数实现

- 采用尾插的方法，遍历过程中检查有无重复，有重复就不再添加，否则利用尾指针进行尾插

```

int Student:: addItem(string id)
{
    /*获取尾指针*/

```

```

CurriculumNode* rear = new(nothrow)CurriculumNode;
rear = this->head;
if (rear == NULL) exit(1);

/*遍历，顺便查重*/
for (CurriculumNode* ptr = this->head; ptr->next != NULL; ptr = ptr->next)
{
    if (ptr->next->id == id)
    {
        cout << setw(34) << "已经存在此课" << endl;
        return 0;
    }
    rear = ptr;
}

/*进行插入*/
CurriculumNode* p = new(nothrow)CurriculumNode;
if (p == NULL) exit(1);
p->id = id;
p->next = rear->next;
rear->next = p;
cout << setw(34) << "插入成功" << endl;
return 1;
}

```

特别注意rear尾指针一定要在初始时定位在头指针的位置，表示链表为空

插入注意指针先连后边，后连前边，保证链表不丢失

- addItem函数实现

- 这里直接比较id是否相同即可

```

int Student::find(string s)
{
    for (CurriculumNode* ptr = this->head; ptr != NULL; ptr = ptr->next)
    {
        if (ptr->id == s)
        {
            return 1;
        }
    }
    return 0;
}

```

- 学生节点类

- 学生节点类中主要存储学生的id和next指针，也就是课程表节点

```
class StudentNode
{
public:
    string id;
    StudentNode* next;
};
```

同理对于课程类会有相同的数据结构，分为课程类和节点类

- 课程类
  - 重写父类的虚函数，以及其特殊信息（学分，课程头节点）

```
class Curriculum : public abstract
{
public:
    Curriculum();

    Curriculum(string id, string name, string credit);

    int deleteItem(string s);

    int addItem(string s);

    int find(string s);

    StudentNode* head;
    string credit;
};
```

- 有参构造与无参构造

```
Curriculum::Curriculum() {}
Curriculum::Curriculum(string id = " ", string name = " ", string credit = " ")
{
    this->id = id;
    this->name = name;
    this->credit = credit;
    StudentNode* head = (StudentNode*)malloc(sizeof(StudentNode));
    this->head = head;
    this->head->next = NULL;
}
```

同样注意head赋初值的问题

- deleteItem()实现

```
int Curriculum::deleteItem(string s)
{
    for (StudentNode* ptr = this->head; ptr->next != NULL; ptr = ptr->next)
    {
        if (ptr->next->id == s)
```

```

        {
            StudentNode* q = ptr->next;
            ptr->next = q->next;
            free(q);
            cout << setw(34) << "已删除" << endl;
            return 1;
        }
    }
    return 0;
}

```

#### ◦ addItem()实现

```

int Curriculum::addItem(string id)
{
    StudentNode* rear = new(nothrow)StudentNode;
    rear = this->head;
    if (rear == NULL) exit(1);

    /*查重*/
    for (StudentNode* ptr = this->head; ptr->next != NULL; ptr = ptr->next)
    {
        if (ptr->next->id == id)
        {
            cout << setw(34) << "学生已经存在" << endl;
            return 0;
        }
        rear = ptr;
    }

    StudentNode* p = new(nothrow)StudentNode;
    if (p == NULL) exit(1);
    p->id = id;
    p->next = rear->next;
    rear->next = p;
    cout << setw(34) << "插入成功" << endl;
    return 1;
}

```

如果有重复，就不进行插入，防止信息重复

#### ◦ find函数实现



```
int Curriculum::find(string s)
{
    for (StudentNode* ptr = this->head; ptr != NULL; ptr = ptr->next)
    {
        if (ptr->id == s)
        {
            return 1;
        }
    }
    return 0;
}
```

- 课程节点类

- 满足学生课程表链表的需要

```
class CurriculumNode
{
public:
    string id;
    CurriculumNode* next;
};
```

- 哈希表

- 注意到两个查询表之间有高度相似性，而且没有特殊差异，同一个查询表内存储的是同一类元素，则可以用类模板进行统一，建表时直接实例化就可以了
- 注意模板类的头文件具体实现需要在.hpp文件中，放在头文件文件夹中，而不是头文件和cpp文件分开写，保证编译器在编译时不会找不到函数定义
- 模板类功能

```
template<class T>
class HashTable
{
public:
    HashTable(); //无参构造函数

    HashTable(int N){} //有参构造函数

    int BKDRHash(string s){} //哈希函数

    int insert(T x){} //插入数据

    T find(string id){} //查找数据

    T deleteItem(string id){} //删除数据

    int N; //线性表模拟哈希表的表长
    int idx = 0; //元素顺序表的元素个数
    T* e = new T[1e5]; //元素表
    int* h = new int[1e5]; //哈希值对应头节点的位置
```

```
int* ne = new int[1e5]; //指针顺序表

};
```

用顺序表模拟哈希表时，每一个哈希值对应的头节点元素在元素顺序表中的位置需要额外一个顺序表h进行记录，这样顺序表中相当于既有头节点又有链表节点，而h这个数组记录的就是头节点的位置

#### ○ 构造函数实现

```
HashTable();

HashTable(int N) //哈希表索引长度
{
    this->N = N;
    /*建表*/
    for (int i = 0; i < N; i++)
    {
        h[i] = -1;
        ne[i] = 0;
        /*e[i]的初始化*/
        T ori;
        e[i] = ori;
    }
}
```

注意初始化h[i]的每一个值为-1，元素顺序表中的元素初始化

#### ○ 哈希函数

```
int BKDRHash(string s)
{
    unsigned int seed = 31;
    unsigned int key = 0;

    for (int i = 0; i < s.size(); i++)
    {
        key = key * seed + s[i];
    }

    return (key & 0x7fffffff) % N;
}
```

- 此处选用BKDRHash，保证哈希冲突最少，性能相对更优
- 有人对不同哈希函数进行测试

Hash函数	数据1	数据2	数据3	数据4	数据1得分	数据2得分	数据3得分	数据4得分	平均分
BKDRHash	2	0	4774	481	96.55	100	90.95	82.05	92.64
APHash	2	3	4754	493	96.55	88.46	100	51.28	86.28
DJBHash	2	2	4975	474	96.55	92.31	0	100	83.43
JSHash	1	4	4761	506	100	84.62	96.83	17.95	81.94
RSHash	1	0	4861	505	100	100	51.58	20.51	75.96
SDBMHash	3	2	4849	504	93.1	92.31	57.01	23.08	72.41
PJWHash	30	26	4878	513	0	0	43.89	0	21.95
ELFHash	30	26	4878	513	0	0	43.89	0	21.95

可以看到BKDRHash表现最优（依据哈希冲突减分），平均分为平方平均数

- 其本质是类似于十进制的思想，每一位根据其位数的顺序会有相应的位权，于是可得,对于string s,  
最后取余保证其映射到数组下标范围内

$$HashValue(s) = \sum_{i=0}^{n-1} s[i] \cdot seed^i$$

- insert函数实现

```
int insert(T x)
{
    int k = BKDRHash(x.id); //计算哈希值
    e[idx] = x;
    ne[idx] = h[k];
    h[k] = idx++;

    cout << setw(34) << "完成插入" << endl;
    return 1;
}
```

通过修改指针顺序表中的指向位置就可以进行元素的添加

- find函数实现

```
T find(string id) //返回实例
{
    int k = BKDRHash(id); //计算哈希值
    for (int i = h[k]; i != -1; i = ne[i])
    {
        if (e[i].id == id)
        {
            cout << e[i].name << " " << e[i].id << endl;
            return e[i];
        }
    }
    cout << setw(34) << "查无此人" << endl;
    T temp = T();
    return temp; //返回空对象
}
```

这里设置未找到时返回的是一个空对象，便于检验是否找到

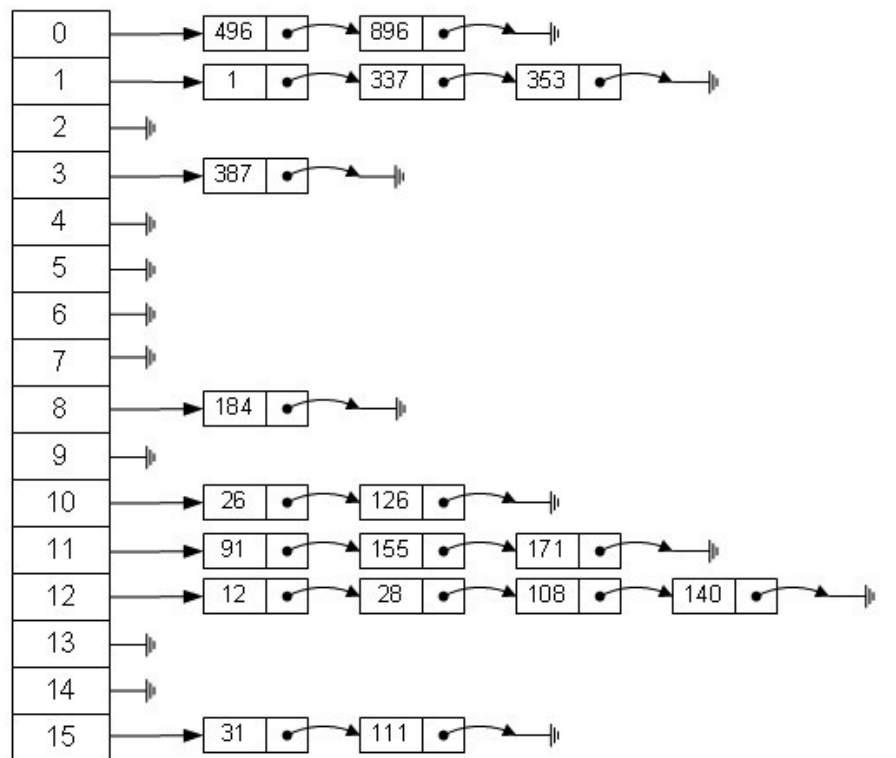
但是这样会浪费一部分空间，产生很多空对象

- 拉链法模拟
  - $e[i]$ 记录插入的元素，依次往后排即可
  - $h[i]$ 记录每一个哈希值对应的指针（这里指的就是相应链表的第一个元素在 $e[i]$ 中的下标）
  - $ne[i]$ 记录每一个节点的指针域（ $e[i]$ 的数组下标），与元素顺序表相对应
  - 每次加入或删除或查找一个元素，需要先计算哈希值，然后直接到对应头节点差相应的链表即可

## 2.4 主要算法设计

- 标签查找
  - 就是根据学生或者课程的id号来进行查找，而不是在课程表或者选课名单中不断重建实例化对象
  - 这样设计可以节省很多重复的实例化对象，减少空间浪费
- 哈希表查找
  - 利用字符串哈希进行学生和课程的查找，极大降低了字符串匹配的时间复杂度，提升了删除、查找、插入等操作的效率
  - 拉链法解决哈希冲突，将链表和顺序表进行结合，最大化查找效率
    - 顺序表是用来与哈希值对应的，每一个哈希值对应一个内存单元。链表是用来存储相同哈希值的字符串的
    - 如果哈希冲突发生，可以在记录头节点的顺序表后边链接链表，如果哈希函数足够均匀，每次映射之后进行查询不需要遍历链表很长时间

拉链法的图示：



## 2.5 框架结构设计

- 多态
  - 学生类和课程类有相同的方法以及两个相同的属性，所以可以进行抽象

```
/*-----Abstract.h-----*/
class abstract
{
public:
    virtual int deleteItem(string s) = 0;

    virtual int addItem(string s) = 0;

    virtual int find(string s) = 0;
    string name;

    string id = " ";

};
/*-----Student.h-----*/
#include "abstract.h"
#include <iomanip>
using namespace std;

class StudentNode;
class CurriculumNode;
class Student : public abstract
{...}

/*-----Curriculum.h-----*/
#include <iostream>
#include <iomanip>
#include "abstract.h"
//#include "student.h"
using namespace std;
class CurriculumNode;
class StudentNode;
class Curriculum : public abstract
{...}
```

抽象类方法要使用纯虚函数

- 哈希表模板类
  - 学生名单表和课程名单表具有高度相似性，1可以使用类模板提高复用性

```
template<class T>
class HashTable
{
public:
    HashTable(); //无参构造函数

    HashTable(int N){} //有参构造函数

    int BKDRHash(string s){} //哈希函数
```

```

int insert(T x){} //插入数据

T find(string id){} //查找数据

T deleteItem(string id){} //删除数据


int N; //线性表模拟哈希表的表长
int idx = 0; //元素顺序表的元素个数
T* e = new T[1e5]; //元素表
int* h = new int[1e5]; //哈希值对应头节点的位置
int* ne = new int[1e5]; //指针顺序表

};

```

注意类模板实现在.hpp文件中

- 用一个类或者在主系统cpp文件中进行实例化

```

class NameTables
{
public:

    HashTable<Student> studentHashTable = HashTable<Student>(53334);
    HashTable<Curriculum> curriculumHashTable = HashTable<Curriculum>
(4000);
};

```

课程数据大约3000条，学生数据大约40000条，为使得哈希表冲突减小，查询时间不能过长且空间占用适中，可以将装载因子设置为0.75，所以在初始化时将哈希表表长分别设置为53334和4000

装载因子：

加载因子是哈希表在其容量自动扩容之前可以达到多满的一种度量。当哈希表中的条目数超出了加载因子与当前容量的乘积时，则要对该哈希表进行扩容、rehash操作（即重建内部数据结构），扩容后的哈希表将具有两倍的原容量。

$$loadFactor = \frac{number\ of\ records}{length\ of\ HashMap}$$

- 操作模块
  - 在操作模块中调用相应方法实现相应功能，并构建相应的界面
- 文件存储
  - 有效与本地的小数据库（txt文档存储）进行交互。保证数据的完整性
  - 存储时将人员名单与每一位学生的课程分别进行存储，便于后期查看
  - 对存储时需要进行单独删除某行的操作进行实现

```

void deletespecificLine(string filePath, string idToDel)
{
    fstream in(filePath, ios::in); //原文件

```

```

    fstream out("dataBase\\test.txt", ios::out | ios::trunc); //中间文件
    string id, name, info;
    while (in >> id && in >> name && in >> info)
    {
        if (id == idToDel) //比较test.txt每一行的内容和要删除的是否一致，一致就
        跳过(不懂为啥跳过看文章开头的方法)
            continue;
        out << id << " " << name << " " << info << "\n"; //不一致的内容写到
        temp.txt中，注意换行
    }
    in.close(); //关闭流
    out.close();

    fstream outfile("dataBase\\test.txt", ios::in);
    fstream infile(filePath, ios::out);
    while (outfile >> id && outfile >> name && outfile >> info) //将
    temp.txt的内容写到test.txt
    {
        infile << id << " " << name << " " << info << "\n";
    }
    string pat = "dataBase\\test.txt";
    const char* path = pat.c_str();
    remove(path); //删除temp.txt
    outfile.close(); //关闭流
    infile.close();
}

```

就是找临时文件写入再复写回来

## 2.6 复杂度分析

- 查找学生与课程
  - 哈希表查找的时间复杂度为 $O(1)$
- 删除学生与课程
  - 删除包括查找和删除两个步骤，查找步骤在 $O(1)$ 的时间复杂度，删除因为涉及到链表的查找，时间复杂度在 $O(n)$ ， $n$ 是一个班级的人数，所以链表查找不会过长，超过200个节点的可能性不大，所以总体时间消耗并不大
- 输出课程表与选课名单
  - 输出同样涉及到遍历和输出操作，需要遍历到尾节点，所以其时间复杂度在 $O(n)$ 左右
- 查找这一关键性能优化后，以此为基础的所有操作时间消耗大大降低，注意此处的 $O(n)$ 和学生数量不同，与正常字符串匹配进行的查找不是在一个数量级上的

## 2.7 主要功能说明

- 功能概览

```
void opening();
```

```
//引导界面
```

```

void init(NameTables& nameTable);           //初始化学生和课程
名单
void BatchImportStudentInfo(NameTables& nametable); //批量导入学生信息
void BatchImportCurriculumInfo(NameTables& nametable); //批量导入课程信息
void FindStudent(NameTables& nametable); //查找学生信息
void DeleteStudent(NameTables& nametable); //删除学生
void InsertStudent(NameTables& nametable); //插入学生
void FindCurriculum(NameTables& nametable); //查找课程信息
void DeleteCurriculum(NameTables& nametable); //删除课程
void InsertCurriculum(NameTables& nametable); //插入课程
void StudentRegisterForCurriculum(NameTables& nametable); //学生注册课程
void StudentQuitCurriculum(NameTables& nametable); //学生退出课程
void CurriculumEliminateStudent(NameTables& nametable); //课程将学生退课
void PrintRegisters(NameTables& nametable); //打印注册课程学生
名单
void PrintSchedules(NameTables& nametable); //打印学生课程表

```

- 界面引导

- opening函数实现

```

void opening()
{
    cout << setiosflags(ios::left);
    cout << "*****" << "*****" << "***** " << endl;
    cout << "*-----" << setw(18) << "1.输入所有学生信息" << "-----*" <<
endl;
    cout << "*-----" << setw(18) << "2.输入所有课程信息" << "-----*" <<
endl;
    cout << "*-----" << setw(18) << "3.查找学生记录" << "-----*" <<
endl;
    cout << "*-----" << setw(18) << "4.删除学生记录" << "-----*" <<
endl;
    cout << "*-----" << setw(18) << "5.插入学生记录" << "-----*" <<
endl;
    cout << "*-----" << setw(18) << "6.查找课程记录" << "-----*" <<
endl;
    cout << "*-----" << setw(18) << "7.删除课程记录" << "-----*" <<
endl;
    cout << "*-----" << setw(18) << "8.插入课程记录" << "-----*" <<
endl;
    cout << "*-----" << setw(18) << "9.学生注册课程" << "-----*" <<
endl;
    cout << "*-----" << setw(18) << "10.学生退出课程" << "-----*" <<
endl;
    cout << "*-----" << setw(18) << "11.课程退课学生" << "-----*" <<
endl;
    cout << "*-----" << setw(18) << "12.输出选课名单" << "-----*" <<
endl;
    cout << "*-----" << setw(18) << "13.输出课程列表" << "-----*" <<
endl;
    cout << "*-----" << setw(18) << "14.退出选课程序" << "-----*" <<
endl;
}

```



```

        cout << "*****" << setw(18) << "*****" << "*****"
" << endl;
        cout << setiosflags(ios::right);

}

```

<iomanip>实现格式控制

- 主函数循环中输入提示

```

opening();
cout << "请输入功能序号: " << endl;
cout << "输入栏=" << "==" << "==== << "==" << "输出栏=" << endl;

```

- 用户输入与输入提示信息 和 系统反馈信息分开

```

cout << setw(34)<< "Info" << endl;

```

- 实现效果

```

*****
*-----1. 输入所有学生信息-----*
*-----2. 输入所有课程信息-----*
*-----3. 查找学生记录-----*
*-----4. 删除学生记录-----*
*-----5. 插入学生记录-----*
*-----6. 查找课程记录-----*
*-----7. 删除课程记录-----*
*-----8. 插入课程记录-----*
*-----9. 学生注册课程-----*
*-----10. 学生退出课程-----*
*-----11. 课程退课学生-----*
*-----12. 输出选课名单-----*
*-----13. 输出课程列表-----*
*-----14. 退出选课程序-----*
*****
请输入功能序号:
=输入栏==  ==  ==  ==输出栏=
14
系统已关闭

```

- 每次操作结束后按任意键会自动清屏，保证界面清洁

- 批量输入学生与课程信息

- BatchImportStudentInfo()函数实现

```

void BatchImportStudentInfo(NameTables& nametable)
{
    cout << "请输入学生人数(小于" << N_stu << ")" << endl;
    int n;
    cin >> n;

    n = checkStuNum(n);

    for (int i = 1; i <= n; i++)
    {
        cout << "输入 学号 姓名 性别 (male / female)" << endl;
        string id, name, gender;
    }
}

```

```

cin >> id >> name >> gender;

/*检验输入合法性*/
while (!checkStu(id, name, gender))
{
    cout << "请重新输入" << endl;
    cout << "输入 学号 姓名 性别 (male / female)" << endl;
    cin >> id >> name >> gender;
}

Student stu = nametable.studentHashTable.find(id);
if (stu.id == " ")
{
    stu = Student(id, name, gender);
    nametable.studentHashTable.insert(stu);

    /*创建课程表文件*/
    ofstream ofs;
    string path = "dataBase\\StudentFiles\\" + id +
"Schedule.txt";
    ofs.open(path, ios::out);
    ofs.close();

    /*在学生总表中添加名字*/
    ofstream namefile;
    namefile.open(StudentNameTable, ios::app);
    namefile << id << " " << name << " " << gender << endl;
    namefile.close();
}

else
{
    cout << setw(34) << "已经有此人" << endl;
    cout << setw(34) << "此次不计,重新输入" << endl;
    i -= 1;
}
}
}

```

注意到输入错误不计在输入次数之内

批量导入课程信息类似

- 效果

```

*****
*-----1. 输入所有学生信息-----*
*-----2. 输入所有课程信息-----*
*-----3. 查找学生记录-----*
*-----4. 删除学生记录-----*
*-----5. 插入学生记录-----*
*-----6. 查找课程记录-----*
*-----7. 删除课程记录-----*
*-----8. 插入课程记录-----*
*-----9. 学生注册课程-----*
*-----10. 学生退出课程-----*
*-----11. 课程退课学生-----*
*-----12. 输出选课名单-----*
*-----13. 输出课程列表-----*
*-----14. 退出选课程序-----*
*****
请输入功能序号：
=输入栏=== ==== ===输出栏=
1
请输入学生人数(小于3000)
3
输入 学号 姓名 性别
2151564 qwe 男
查无此人
完成插入

输入 学号 姓名 性别
2151536 wer 女
查无此人
完成插入

输入 学号 姓名 性别
2151587 ert 男
查无此人
完成插入

请按任意键继续. . .

```

- 查找学生（课程）记录
  - FindStudent()函数实现

```

void FindStudent(NameTables& nametable)
{
    cout << "输入查找的学生的学号: " << endl;
    string id;
    cin >> id;

    while (!checkStuId(id))
    {
        cout << setw(34) << "请重新输入" << endl;
        cin >> id;
    }

    nametable.studentHashTable.find(id);
}

```

利用哈希表进行查找

- 效果

- 删除学生（课程）记录
  - DeleteStudent()函数实现

```
void DeleteStudent(NameTables& nametable)
{
    cout << "输入学生学号" << endl;
    string s;
    cin >> s;

    while (!checkStuId(s))
    {
        cout << "请重新输入" << endl;
        cin >> s;
    }

    Student stu = nametable.studentHashTable.deleteItem(s);

    if (stu.id != " ")
    {
        /*先删除课程表文件*/
        string temp_p = "dataBase\\StudentFiles\\" + s + "Schedule.txt";
        const char* path = temp_p.c_str();
        remove(path);

        /*再删除总表学生*/
        deletespecificLine(temp_p, stu.id);

        for (CurriculumNode* ptr = stu.head->next; ptr != NULL; ptr =
ptr->next)
        {
            string temp_id = ptr->id;
            Curriculum cri =
nametable.curriculumHashTable.find(temp_id);
            cri.deleteItem(temp_id);
        }
    }
}
```

```

        /*课程注册名单里删除注册的学生*/
        ofstream ofs;
        string path = "dataBase\\CurriculumFiles\\" + cri.id +
"Register.txt";
        ofs.open(path, ios::out);
        deleteSpecificLine(path, stu.id);
        ofs.close();
    }

}
else
{
    cout << setw(34) << "查无此人" << endl;
}
}

```

直接删除一个课程或者老师之前，需要这位学生所选的所有课程都将这名学生删除掉，因此涉及到

- 查找这名学生
- 遍历其每一门课
- 将每一门课中这名学生删除掉
- 将这名学生从姓名表中删除掉

- DeleteCurriculum()函数实现类似
- 效果

```

*****
*-----1. 输入所有学生信息-----*
*-----2. 输入所有课程信息-----*
*-----3. 查找学生记录-----*
*-----4. 删除学生记录-----*
*-----5. 插入学生记录-----*
*-----6. 查找课程记录-----*
*-----7. 删除课程记录-----*
*-----8. 插入课程记录-----*
*-----9. 学生注册课程-----*
*-----10. 学生退出课程-----*
*-----11. 课程退课学生-----*
*-----12. 输出选课名单-----*
*-----13. 输出课程列表-----*
*-----14. 退出选课程序-----*
*****
请输入功能序号:
=输入栏==  ====  ====  ===输出栏=
4
输入学生学号
2151536
                                     删除成功
请按任意键继续. . .

```

- 插入学生（课程）记录
  - InsertStudent()函数实现

```

void InsertStudent(NameTables& nametable)
{
    cout << "输入学生 学号 姓名 性别" << endl;
    string id, name, gender;
    cin >> id >> name >> gender;
}

```

```

while (!checkStu(id, name, gender))
{
    cout << "请重新输入" << endl;
    cout << "输入 学号 姓名 性别" << endl;
    cin >> id >> name >> gender;
}

Student stu = nametable.studentHashTable.find(id);
if (stu.id == " ")
{
    stu = Student(id, name, gender);
    nametable.studentHashTable.insert(stu);

    /*创建课程表文件*/
    ofstream ofs;
    string path = "dataBase\\StudentFiles\\" + id + "Schedule.txt";
    ofs.open(path, ios::out);
    ofs.close();

    /*在学生总表中添加名字*/
    ofstream namefile;
    namefile.open(StudentNameTable, ios::app);
    namefile << id << " " << name << " " << gender << endl;
    namefile.close();
}

else
{
    cout << setw(34) << "已经有此人" << endl;
}
}

```

插入课程类似，效果等同于批量导入

- 学生注册课程
  - StudentRegisterForCurriculum()函数实现

```

void StudentRegisterForCurriculum(NameTables& nametable)
{
    cout << "输入学生 学号 姓名 性别 (male / female)" << endl;
    string id, name, gender;
    cin >> id >> name >> gender;

    while (!checkStu(id, name, gender))
    {
        cout << "请重新输入" << endl;
        cout << "输入 学号 姓名 性别 (male / female)" << endl;
        cin >> id >> name >> gender;
    }
}

```

```

Student stu = nametable.studentHashTable.find(id);
if (stu.id != " ")
{
    cout << "输入课程课号" << endl;
    string id;
    cin >> id;

    while (!checkCriId(id))
    {
        cout << "请重新输入" << endl;
        cin >> id;
    }

    Curriculum cri = nametable.curriculumHashTable.find(id);
    if (cri.id == " ")
    {
        cout << setw(34) << "没有此课" << endl;
        return;
    }
    stu.addItem(cri.id);
    cri.addItem(stu.id);

    ofstream schedule;
    string path1 = "dataBase\\StudentFiles\\" + stu.id +
    "Schedule.txt";
    schedule.open(path1, ios::app);
    schedule << cri.id << " " << cri.name << " " << cri.credit <<
endl;
    schedule.close();

    ofstream registers;
    string path2 = "dataBase\\CurriculumFiles\\" + cri.id +
    "Register.txt";
    registers.open(path2, ios::app);
    registers << stu.id << " " << stu.name << " " << stu.gender <<
endl;
    registers.close();
}
else
{
    cout << setw(34) << "查无此人" << endl;
}
}

```

注意在学生添加课程的同时，课程本身也要添加这位同学

- 效果

- 学生退出课程
  - StudentQuitCurriculum()函数实现

```
void StudentQuitCurriculum(NameTables& nametable)
{
    cout << "输入学生学号" << endl;
    string s;
    cin >> s;

    while (!checkStuId(s))
    {
        cout << "请重新输入" << endl;
        cin >> s;
    }

    Student stu = nametable.studentHashTable.find(s);
    if (stu.id != " ")
    {
        cout << "输入课程课号" << endl;
        string criId;
        cin >> criId;

        while (!checkCriId(criId))
        {
            cout << "请重新输入" << endl;
            cin >> criId;
        }

        Curriculum cri = nametable.curriculumHashTable.find(criId);
        if (cri.id != " ")
        {
            stu.deleteItem(cri.id);
            cri.deleteItem(stu.id);

            /*文件*/
            ofstream schedule;
            string path1 = "dataBase\\StudentFiles\\" + stu.id +
"Schedule.txt";
            deleteSpecificLine(path1, cri.id);

            ofstream registers;
            string path2 = "dataBase\\CurriculumFiles\\" + cri.id +
"Register.txt";
```



```

        deleteSpecificLine(path2, stu.id);
    }

    else
    {
        cout << setw(34) << "查无此课" << endl;
    }

}
else
{
    cout << setw(34) << "查无此人" << endl;
}
}

```

在学生退出课程后，同时课程也要将学生删除掉

- 课程将学生退课类似

- 输出选课名单

- PrintRegesters()函数实现

```

void PrintRegesters(NameTables& nametable)
{
    cout << "输入课程课号: : " << endl;
    string id;
    cin >> id;

    while (!checkCriId(id))
    {
        cout << "请重新输入" << endl;
        cin >> id;
    }

    Curriculum cri = nametable.curriculumHashTable.find(id);
    for (StudentNode* ptr = cri.head->next; ptr != NULL; ptr = ptr-
>next)
    {
        string temp_id = ptr->id;
        nametable.studentHashTable.find(temp_id);
    }
}

```

首先找到这名学生

通过其含有的头指针对其课程表遍历

- 打印选课名单类似
- 效果(下边是课程名称和课号)

- 退出选课程序
  - 显示系统已关闭，退出系统
  - 效果

```

*****
*-----1. 输入所有学生信息-----*
*-----2. 输入所有课程信息-----*
*-----3. 查找学生记录-----*
*-----4. 删除学生记录-----*
*-----5. 插入学生记录-----*
*-----6. 查找课程记录-----*
*-----7. 删除课程记录-----*
*-----8. 插入课程记录-----*
*-----9. 学生注册课程-----*
*-----10. 学生退出课程-----*
*-----11. 课程退课学生-----*
*-----12. 输出选课名单-----*
*-----13. 输出课程列表-----*
*-----14. 退出选课程序-----*
*****
请输入功能序号:
-输入栏===  ====  ===-输出栏=
14
                                系统已关闭

```

## 2.8 健壮性分析

- 学生信息输入错误
  - 主要检查了学生的三个基本信息：姓名 学号 性别等
  - checkStu()函数实现

```

bool checkStu(string id, string name, string gender)
{
    if (id.size() != 7)
    {
        cout << setw(34) << "学号位数不对,应该是7位" << endl;
        return false;
    }

    else if (id[0] == 0)
    {
        cout << setw(34) << "同学,你是第几届的呀,咋学号第一位是0呢" << endl;
        return false;
    }

    else if (name.size() > 120)
    {
        cout << setw(34) << "同学,你这名字够长,比毕加索的都长" << endl;
        return true;
    }
}

```

```

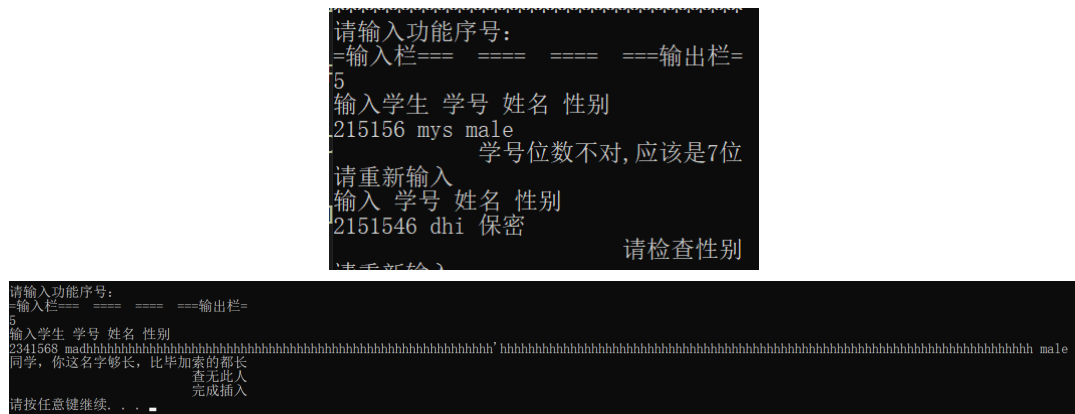
        else if (!(gender == "男" || gender == "女" || gender == "male" ||
gender == "female" || gender == "中性" || gender == "neutral gender"))
        {
            cout << setw(34) << "请检查性别" << endl;
            return false;
        }

        else
        {
            return true;
        }
    }
}

```

其中包括了学号的数字位数，学号标志位数字的正确性以及姓名长度的正确性和性别

- 效果



```

请输入功能序号:
=输入栏==  ====  ====  ===输出栏=
5
输入学生 学号 姓名 性别
215156 mys male
学号位数不对, 应该是7位
请重新输入
输入 学号 姓名 性别
2151546 dhi 保密
请检查性别

请输入功能序号:
=输入栏==  ====  ====  ===输出栏=
5
输入学生 学号 姓名 性别
2341568 mads
同学, 你这名字够长, 比毕加索的都长
          查无此人
          完成插入
请按任意键继续. . .

```

- 课程信息输入错误

- 类似检查学生信息
- checkCri()函数实现

```

bool checkCri(string id, string name, string credit)
{
    for (int i = 0; i < credit.size(); i++)
    {
        if (!isdigit(credit[i]))
        {
            cout << setw(34) << "学分必须是纯数字" << endl;
            return false;
        }
    }

    double sc = atof(credit.c_str());

    if (id.size() > 10)
    {
        cout << setw(34) << "课程号太长" << endl;
        return false;
    }
    else if (name.size() > 120)
    {

```

```

        cout << setw(34) << "课程名太长" << endl;
        return false;
    }

    else if (upLimitOfcredit - sc < 1e-6 || sc - bottomLimitOfcredit <
1e-6)
    {
        cout << setw(34) << "学分范围不对, 请检查范围" << endl;
        return false;
    }

    else
    {
        return true;
    }
}

```

包括学分的输入格式, 课程号的长度 (参考1系统最长课号长度), 课程名长度以及学分范围 (已知中德工程学院德语课程11学分最高)

全局变量中设置范围

```

const double upLimitOfcredit = 11;
const double bottomLimitOfcredit = 0.5;

```

- 学生(课程)输入重复
  - 涉及到添加的时候会有添加重复的问题
  - 在遍历链表进行尾插的时候可以对是否重复进行检查

```

...
/*查重*/
CurriculumNode* rear = new(nothrow)CurriculumNode;
rear = this->head;
if (rear == NULL) exit(1);
for (CurriculumNode* ptr = this->head; ptr->next != NULL; ptr = ptr-
>next)
{
    if (ptr->next->id == id)
    {
        cout << setw(34) << "已经存在此课" << endl;
        return 0;
    }
    rear = ptr;
}
...

```

理论上可以对添加重复学生和课程进行检查, 只要在遍历哈希表中链表看是否有重复元素即可

- 效果

- 删除学生（课程）不存在
  - 在查询表查找时出现不存在这门课（这个学生的情况）

```
T find(string id)
{
    int k = BKDRHash(id);
    for (int i = h[k]; i != -1; i = ne[i])
    {
        if (e[i].id == id)
        {
            cout << e[i].name << " " << e[i].id << endl;
            return e[i];
        }
    }
    cout << setw(34) << "查无此人" << endl;
    T temp = T();
    return temp;
}
```

此操作见于增删改查之中，以及学生添加课程，退出课程，课程退课学生，所以在输出提示中会出现是否查找成功的提示

- 效果

```
*****
*-----1. 输入所有学生信息-----*
*-----2. 输入所有课程信息-----*
*-----3. 查找学生记录-----*
*-----4. 删除学生记录-----*
*-----5. 插入学生记录-----*
*-----6. 查找课程记录-----*
*-----7. 删除课程记录-----*
*-----8. 插入课程记录-----*
*-----9. 学生注册课程-----*
*-----10. 学生退出课程-----*
*-----11. 课程退课学生-----*
*-----12. 输出选课名单-----*
*-----13. 输出课程列表-----*
*-----14. 退出选课程序-----*
*****
请输入功能序号：
=输入栏==  ====  ====  ===输出栏=
4
输入学生学号
2151537
                                     查无此人
请按任意键继续. . .
```

## 2.9 调试分析

- 链表指针的申请与变换
  - 初始化链表头指针，插入指针变换，删除指针变换
- 链表遍历

- 注意for循环找到相应节点的前驱节点循环终止是ptr->next, 循环到相应节点是ptr,以及尾指针的判断
- 顺序表模拟的优缺点
  - 真正链表可以进行内存的申请和释放
  - 模拟链表会有一个最大长度, 超过长度模拟链表失效了
  - 而且在删除之后内存无法得到及时释放, 造成线性表空间的浪费, 但是在有限较少的学生及课程数量中这样的数据结构能够有效结合二者优点, 提升查找效率
- 头文件的相互引用冲突
  - 在构建学生类和课程类中遇到了头文件互相包含的问题
  - 一种解决方案是在一个类定义时使用类, 另一个类中使用类指针
  - 另一种方案是在cpp文件中包含相应的头文件
  - 本实验采用后者
- 文件读写与删除特定行
  - 利用临时文件作为中介来重写一遍原文件 (效率降低)
- 哈希表解决哈希冲突
  - 拉链法解决哈希冲突因为有链表的存在, 会使得性能有所下降, 好像可以用红黑树对链表进行优化, 查找更迅速
- 模板类
  - 在写模板类时一定要在头文件中将其实现, 并改为hpp文件, 否则编译报错 (无实际类, 编译器找不到函数的实现: 无法解析的外部命令)

### 3.预期实现与改进

- 管理员与学生权限
  - 功能的使用区分管理员权限和学生权限, 使用不同功能
  - 登录界面, 密码输入并与本地的信息记录进行比对
- 课程选课时间冲突
  - 在选课信息中添加时间信息
  - 同一个学生选课时间冲突之后不能进行选课

### 4.实验总结

本实验综合运用了顺序表与链表, 实现了选课系统的相应功能, 利用顺序表和链表实现的哈希表极大提升了查找效率, 学习了哈希函数相关知识。同时运用了面向对象的相关知识, 实现了类与类之间的抽象和关联。

## 5. 源代码

### 头文件

- abstract.h

```
#pragma once
#include <iostream>
#include <string>
using namespace std;

class abstract
{
public:
    virtual int deleteItem(string s) = 0;

    virtual int addItem(string s) = 0;

    virtual int find(string s) = 0;
    string name;

    string id = " ";

};
```

- curriculum.h

```
#pragma once
#include <iostream>
#include <iomanip>
#include "abstract.h"
// #include "student.h"
using namespace std;
class CurriculumNode;
class StudentNode;
class Curriculum : public abstract
{
public:
    Curriculum();

    Curriculum(string id, string name, string credit);

    int deleteItem(string s);

    int addItem(string s);

    int find(string s);

    StudentNode* head;
    string credit;
};

class CurriculumNode
```

```
{
public:
    string id;
    CurriculumNode* next;
};
```

- GlobalFile.h

```
#pragma once

#define StudentNameTable "dataBase\\StudentFiles\\StudentNameTable.txt"

#define CurriculumNameTable
"dataBase\\CurriculumFiles\\CurriculumNameTable.txt"
```

- hashTable.hpp

```
#pragma once
#include <iostream>
#include <cstring>
#include <fstream>
#include "GlobalFile.h"
#include <cstdlib>
#include <iomanip>
// #include <fstream>
using namespace std;

template<class T>
class HashTable
{
public:
    HashTable();

    HashTable(int N) //哈希表索引长度
    {
        this->N = N;
        //this->idx = 0;
        /*建表*/
        for (int i = 0; i < N; i++)
        {
            h[i] = -1;
            ne[i] = 0;
            /*e[i]的初始化*/
            T ori;
            e[i] = ori;
        }
    }

    int BKDRHash(string s)
```



```

{
    unsigned int seed = 31;
    unsigned int key = 0;

    for (int i = 0; i < s.size(); i++)
    {
        key = key * seed + s[i];
    }

    return (key & 0x7fffffff) % N;
}

int insert(T x)
{
    int k = BKDRHash(x.id);
    e[idx] = x;
    ne[idx] = h[k];
    h[k] = idx++;

    cout << setw(34) << "完成插入" << endl;
    return 1;
}

T find(string id)
{
    int k = BKDRHash(id);
    for (int i = h[k]; i != -1; i = ne[i])
    {
        if (e[i].id == id)
        {
            cout << e[i].name << " " << e[i].id << endl;
            return e[i];
        }
    }
    cout << setw(34) << "查无此人" << endl;
    T temp = T();
    return temp;
}

```

/\*这样用线性表模拟的话，删除时会在数据域线性表里留下很多垃圾\*/

```

T deleteItem(string id)
{
    int k = BKDRHash(id);
    int pos = h[k];
    cout << h[k] << endl;
    for (int i = h[k]; i != -1; i = ne[i])
    {
        if (e[i].id == id)
        {
            cout << pos << " " << i << endl;
            ne[pos] = ne[i];
            if (i == h[k]) h[k] = -1;
            return e[i];
        }
    }
}

```

```

        }
        pos = i;
    }
    T temp = T();
    return temp;
}

//int h[1e5], ne[1e5], idx;

int N;
/*int idx; */
//int* h = (int*)malloc(N * sizeof(int));
int idx = 0;
T* e = new T[1e5];
int* h = new int[1e5];
int* ne = new int[1e5];

//T* e = (T*)malloc(1e5 * sizeof(T));
/*int* ne = (int*)malloc(N * sizeof(int)); */

};

```

- nameTable.h

```

#pragma once
#include "hashTable.hpp"
#include "student.h"
#include "curriculum.h"

class NameTables
{
public:

    HashTable<Student> studentHashTable = HashTable<Student>(3000);
    HashTable<Curriculum> curriculumHashTable = HashTable<Curriculum>
(40000);
};

```

- student.h

```

#pragma once
#include <iostream>
#include "abstract.h"
//#include "curriculum.h"
#include <iomanip>
using namespace std;

class StudentNode;
class CurriculumNode;

```

```

class Student : public abstract
{
public:
    Student();
    Student(string id, string name, string gender);

    int deleteItem(string s);

    int addItem(string s);

    int find(string s);

    CurriculumNode* head;
    string gender;
};

class StudentNode
{
public:
    string id;
    StudentNode* next;
};

```

## 源文件

- curriculum.cpp

```

#include "curriculum.h"
#include "student.h"
#include "nameTables.h"

Curriculum::Curriculum() {}
Curriculum::Curriculum(string id = " ", string name = " ", string credit = " ")
{
    this->id = id;
    this->name = name;
    this->credit = credit;
    StudentNode* head = (StudentNode*)malloc(sizeof(StudentNode));
    this->head = head;
    this->head->next = NULL;
}

int Curriculum::deleteItem(string s)
{
    for (StudentNode* ptr = this->head; ptr->next != NULL; ptr = ptr->next)
    {
        if (ptr->next->id == s)
        {
            StudentNode* q = ptr->next;
            ptr->next = q->next;

```

```

        free(q);
        cout << setw(34) << "已删除" << endl;
        return 1;

    }
}
return 0;
}

int Curriculum::addItem(string id)
{
    /*查重*/
    StudentNode* rear = new(nothrow)StudentNode;
    rear = this->head;
    if (rear == NULL) exit(1);

    for (StudentNode* ptr = this->head; ptr->next != NULL; ptr = ptr->next)
    {
        if (ptr->next->id == id)
        {
            cout << setw(34) << "学生已经存在" << endl;
            return 0;
        }
        rear = ptr;
    }

    StudentNode* p = new(nothrow)StudentNode;
    if (p == NULL) exit(1);
    p->id = id;
    p->next = rear->next;
    rear->next = p;
    cout << setw(34) << "插入成功" << endl;
    return 1;
}

int Curriculum::find(string s)
{
    for (StudentNode* ptr = this->head; ptr != NULL; ptr = ptr->next)
    {
        if (ptr->id == s)
        {
            //cout << ptr->student.name << " " << ptr->student.id << " " <<
            ptr->student.gender << endl;
            return 1;
        }
    }
    return 0;
}
}

```

- student.cpp

```
#include "student.h"
```

```

#include "curriculum.h"

Student::Student() {}

/*head指针申请内存*/
Student::Student(string id = " ", string name = " ", string gender = " ")
{
    this->id = id;
    this->name = name;
    this->gender = gender;
    this->head = (CurriculumNode*)malloc(sizeof(CurriculumNode));
    //this->head = head;
    this->head->next = NULL;
}

/*fix me : 根据不同的元素查找课程*//*暂定为根据名称查找*/
/*成功删除几门课*/

int Student::deleteItem(string s)
{
    /*
    * @param s : 学生学号
    */
    for (CurriculumNode* ptr = this->head; ptr->next != NULL; ptr = ptr->next)
    {
        //遍历目标节点的前一个节点
        if (ptr->next->id == s)
        {
            CurriculumNode* q = ptr->next;
            ptr->next = q->next;
            free(q);
            cout << setw(34) << "已删除" << endl;
            return 1;
        }
    }
    return 0;
}

int Student:: addItem(string id)
{
    /*查重*/
    CurriculumNode* rear = new(nothrow)CurriculumNode;
    rear = this->head;
    if (rear == NULL) exit(1);
    for (CurriculumNode* ptr = this->head; ptr->next != NULL; ptr = ptr->next)
    {
        if (ptr->next->id == id)
        {
            cout << setw(34) << "已经存在此课" << endl;
            return 0;
        }
    }
}

```

```

        }
        rear = ptr;
    }

    CurriculumNode* p = new(nothrow)CurriculumNode;
    if (p == NULL) exit(1);
    p->id = id;
    p->next = rear->next;
    rear->next = p;
    cout << setw(34) << "插入成功" << endl;
    return 1;
}

int Student::find(string s)
{
    for (CurriculumNode* ptr = this->head; ptr != NULL; ptr = ptr->next)
    {
        if (ptr->id == s)
        {
            return 1;
        }
    }
    return 0;
}

```

- systemCR.cpp

```

#include <iostream>
#include <iomanip>
#include "nameTables.h"
#include "student.h"
#include "curriculum.h"
#include <fstream>
#include "GlobalFile.h"
#include <cstdlib>
#include <windows.h>
#include <cctype>
#include <cstring>
using namespace std;
const int N_stu = 3000;
const int N_cur = 40000;
const double upLimitOfcredit = 11;
const double bottomLimitOfcredit = 0.5;

void opening()
{
    cout << setiosflags(ios::left);
    cout << "*****" << "*****" << "*****" << endl;
    cout << "*-----" << setw(18) << "1.输入所有学生信息" << "-----*" <<
    endl;
}

```

```

        cout << "*-----" << setw(18) << "2.输入所有课程信息" << "-----*" <<
endl;
        cout << "*-----" << setw(18) << "3.查找学生记录" << "-----*" << endl;
        cout << "*-----" << setw(18) << "4.删除学生记录" << "-----*" << endl;
        cout << "*-----" << setw(18) << "5.插入学生记录" << "-----*" << endl;
        cout << "*-----" << setw(18) << "6.查找课程记录" << "-----*" << endl;
        cout << "*-----" << setw(18) << "7.删除课程记录" << "-----*" << endl;
        cout << "*-----" << setw(18) << "8.插入课程记录" << "-----*" << endl;
        cout << "*-----" << setw(18) << "9.学生注册课程" << "-----*" << endl;
        cout << "*-----" << setw(18) << "10.学生退出课程" << "-----*" << endl;
        cout << "*-----" << setw(18) << "11.课程退课学生" << "-----*" << endl;
        cout << "*-----" << setw(18) << "12.输出选课名单" << "-----*" << endl;
        cout << "*-----" << setw(18) << "13.输出课程列表" << "-----*" << endl;
        cout << "*-----" << setw(18) << "14.退出选课程序" << "-----*" << endl;
        cout << "*****" << setw(18) << "*****" << "***** " <<
endl;
        cout << setiosflags(ios::right);

    }

bool checkStu(string id, string name, string gender)
{
    if (id.size() != 7)
    {
        cout << setw(34) << "学号位数不对,应该是7位" << endl;
        return false;
    }

    else if (id[0] == 0)
    {
        cout << setw(34) << "同学,你是第几届的呀,咋学号第一位是0呢" << endl;
        return false;
    }

    else if (name.size() > 120)
    {
        cout << setw(34) << "同学,你这名字够长,比毕加索的都长" << endl;
        return true;
    }

    else if (!(gender == "男" || gender == "女" || gender == "male" || gender
== "female" || gender == "中性" || gender == "neutral gender"))
    {
        cout << setw(34) << "请检查性别" << endl;
        return false;
    }

    else
    {
        return true;
    }
}

```

```
bool checkStuId(string id)
{
    if (id.size() != 7)
    {
        cout << setw(34) << "学号位数不对,应该是7位" << endl;
        return false;
    }

    else if (id[0] == 0)
    {
        cout << setw(34) << "同学,你是第几届的呀,咋学号第一位是0呢" << endl;
        return false;
    }

    else
    {
        return true;
    }
}

bool checkCri(string id, string name, string credit)
{
    for (int i = 0; i < credit.size(); i++)
    {
        if (!isdigit(credit[i]))
        {
            cout << setw(34) << "学分必须是纯数字" << endl;
            return false;
        }
    }

    double sc = atof(credit.c_str());

    if (id.size() > 10)
    {
        cout << setw(34) << "课程号太长" << endl;
        return false;
    }
    else if (name.size() > 120)
    {
        cout << setw(34) << "课程名太长" << endl;
        return false;
    }

    else if (upLimitOfcredit - sc < 1e-6 || sc - bottomLimitOfcredit < 1e-6)
    {
        cout << setw(34) << "学分范围不对,请检查范围" << endl;
        return false;
    }

    else
    {
        return true;
    }
}
```



```

bool checkCriId(string id)
{
    if (id.size() > 10)
    {
        cout << setw(34) << "课程号太长" << endl;
        return false;
    }
    else
    {
        return true;
    }
}

int checkStuNum(int n)
{
    int s = n;
    while (s > N_stu)
    {
        cout << setw(34) << "人数太多啦!!!" << endl;
        cout << setw(34) << "人数小于" << N_stu << "请重新输入:" << endl;
        cin >> s;
    }

    return s;
}

void init(NameTables& nameTable)
{
    /*初始化学生*/
    ifstream stufile;
    stufile.open(StudentNameTable, ios::in);
    if (!stufile.is_open())
    {
        cout << "文件不存在" << endl;
        stufile.close();
        return;
    }

    string id, name, gender;

    while (stufile >> id && stufile >> name && stufile >> gender)
    {
        /*添加学生*/
        Student stu = Student(id, name, gender);
        nameTable.studentHashTable.insert(stu);

        /*添加学生课程表*/
        ifstream schedule;
        string path = "dataBase\\StudentFiles\\" + id + "Schedule.txt";
        schedule.open(path, ios::in);

        if (!schedule.is_open())
        {
            cout << "文件不存在" << endl;

```

```

        schedule.close();
        return;
    }

    string criId;
    while (schedule >> criId)
    {
        stu.addItem(criId);
    }

}

/*初始化课程*/
ifstream crifile;
crifile.open(CurriculumNameTable, ios::in);
if (!crifile.is_open())
{
    cout << "文件不存在" << endl;
    crifile.close();
    return;
}

//string id, name, credit;
string credit;

while (crifile >> id && crifile >> name && crifile >> credit)
{
    /*添加课程*/
    Curriculum cri = Curriculum(id, name, credit);
    nameTable.curriculumHashTable.insert(cri);

    /*添加人员名单*/
    ifstream registers;
    string path = "dataBase\\CurriculumFiles\\" + id + "Register.txt";
    registers.open(path, ios::in);

    if (!registers.is_open())
    {
        cout << "文件不存在" << endl;
        registers.close();
        return;
    }

    string stuId;
    while (registers >> stuId)
    {
        cri.addItem(stuId);
    }

}

}

void deleteSpecificLine(string filePath, string idToDel)
{

```

```

    fstream in(filePath, ios::in); // 原文件
    fstream out("dataBase\\test.txt", ios::out | ios::trunc); // 中间文件
    string id, name, info;
    while (in >> id && in >> name && in >> info)
    {
        if (id == idToDelete) // 比较test.txt每一行的内容和要删除的是否一致，一致就跳过
            // (不懂为啥跳过看文章开头的方法)
            continue;
        out << id << " " << name << " " << info << "\n"; // 不一致的内容写到
        // temp.txt中，注意换行
    }
    in.close(); // 关闭流
    out.close();

    fstream outfile("dataBase\\test.txt", ios::in);
    fstream infile(filePath, ios::out);
    while (outfile >> id && outfile >> name && outfile >> info) // 将temp.txt
        // 的内容写到test.txt
    {
        infile << id << " " << name << " " << info << "\n";
    }
    string pat = "dataBase\\test.txt";
    const char* path = pat.c_str();
    remove(path); // 删除temp.txt
    outfile.close(); // 关闭流
    infile.close();
}

void BatchImportStudentInfo(NameTables& nametable)
{
    cout << "请输入学生人数(小于" << N_stu << ")" << endl;
    int n;
    cin >> n;

    n = checkStuNum(n);

    for (int i = 1; i <= n; i++)
    {
        cout << "输入 学号 姓名 性别" << endl;
        string id, name, gender;
        cin >> id >> name >> gender;

        /* 检验输入合法性 */
        while (!checkStu(id, name, gender))
        {
            cout << "请重新输入" << endl;
            cout << "输入 学号 姓名 性别" << endl;
            cin >> id >> name >> gender;
        }

        Student stu = nametable.studentHashTable.find(id);
        if (stu.id == " ")
        {
            stu = Student(id, name, gender);
        }
    }
}

```

```

        nametable.studentHashTable.insert(stu);

        /*创建课程表文件*/
        ofstream ofs;
        string path = "dataBase\\StudentFiles\\" + id + "Schedule.txt";
        ofs.open(path, ios::out);
        ofs.close();

        /*在学生总表中添加名字*/
        ofstream namefile;
        namefile.open(StudentNameTable, ios::app);
        namefile << id << " " << name << " " << gender << endl;
        namefile.close();
    }

    else
    {
        cout << setw(34) << "已经有此人" << endl;
        cout << setw(34) << "此次不计,重新输入" << endl;
        i -= 1;
    }
}

}

void BatchImportCurriculumInfo(NameTables& nametable)
{
    cout << "请输入课程数量" << endl;
    int n;
    cin >> n;

    n = checkStuNum(n);

    for (int i = 1; i <= n; i++)
    {
        cout << "输入 课号 名称 学分" << endl;
        string id, name, credit;
        cin >> id >> name >> credit;

        while (!checkCri(id, name, credit))
        {
            cout << "请重新输入" << endl;
            cout << "输入 课号 名称 学分" << endl;
            cin >> id >> name >> credit;
        }

        Curriculum cri = nametable.curriculumHashTable.find(id);
        if (cri.id == " ")
        {
            cri = Curriculum(id, name, credit);
            nametable.curriculumHashTable.insert(cri);

            /*创建课程表文件*/
            ofstream ofs;

```

```

        string path = "dataBase\\CurriculumFiles\\" + id +
"Register.txt";
        ofs.open(path, ios::out);
        ofs.close();

        /*在课程总表中添加名字*/
        ofstream namefile;
        namefile.open(CurriculumNameTable, ios::app);
        namefile << id << " " << name << " " << credit << endl;
        namefile.close();
    }
    else
    {
        cout << setw(34) << "已经有此课" << endl;
    }
}
}

void FindStudent(NameTables& nametable)
{
    cout << "输入查找的学生的学号: " << endl;
    string id;
    cin >> id;

    while (!checkStuId(id))
    {
        cout << setw(34) << "请重新输入" << endl;
        cin >> id;
    }

    nametable.studentHashTable.find(id);
}

void DeleteStudent(NameTables& nametable)
{
    cout << "输入学生学号" << endl;
    string s;
    cin >> s;

    while (!checkStuId(s))
    {
        cout << "请重新输入" << endl;
        cin >> s;
    }

    Student stu = nametable.studentHashTable.deleteItem(s);

    if (stu.id != " ")
    {
        /*先删除课程表文件*/
        string temp_p = "dataBase\\StudentFiles\\" + s + "Schedule.txt";
        const char* path = temp_p.c_str();
        remove(path);

        /*再删除总表学生*/
    }
}

```

```

deleteSpecificLine(temp_p, stu.id);

    for (CurriculumNode* ptr = stu.head->next; ptr != NULL; ptr = ptr->next)
    {
        string temp_id = ptr->id;
        Curriculum cri = nametable.curriculumHashTable.find(temp_id);
        cri.deleteItem(temp_id);

        /*课程注册名单里删除注册的学生*/
        ofstream ofs;
        string path = "dataBase\\CurriculumFiles\\" + cri.id +
"Register.txt";
        ofs.open(path, ios::out);
        deleteSpecificLine(path, stu.id);
        ofs.close();
    }

}
else
{
    cout << setw(34) << "查无此人" << endl;
}
}

void InsertStudent(NameTables& nametable)
{
    cout << "输入学生 学号 姓名 性别" << endl;
    string id, name, gender;
    cin >> id >> name >> gender;

    while (!checkStu(id, name, gender))
    {
        cout << "请重新输入" << endl;
        cout << "输入 学号 姓名 性别" << endl;
        cin >> id >> name >> gender;
    }

    Student stu = nametable.studentHashTable.find(id);
    if (stu.id == " ")
    {
        stu = Student(id, name, gender);
        nametable.studentHashTable.insert(stu);

        /*创建课程表文件*/
        ofstream ofs;
        string path = "dataBase\\StudentFiles\\" + id + "Schedule.txt";
        ofs.open(path, ios::out);
        ofs.close();

        /*在学生总表中添加名字*/
        ofstream namefile;
        namefile.open(StudentNameTable, ios::app);
    }
}

```

```

        namefile << id << " " << name << " " << gender << endl;
        namefile.close();
    }

    else
    {
        cout << setw(34) << "已经有此人" << endl;
    }
}

void FindCurriculum(NameTables& nametable)
{
    cout << "输入查找的课程的课号: " << endl;
    string id;
    cin >> id;

    while (!checkCriId(id))
    {
        cout << "请重新输入" << endl;
        cin >> id;
    }

    nametable.curriculumHashTable.find(id);
}

void DeleteCurriculum(NameTables& nametable)
{
    cout << "输入课程课号" << endl;
    string s;
    cin >> s;

    while (!checkCriId(s))
    {
        cout << "请重新输入" << endl;
        cin >> s;
    }

    Curriculum cri = nametable.curriculumHashTable.deleteItem(s);

    /*先删除课程注册的学生名单文件*/
    string temp_p = "dataBase\\CurriculumFiles\\" + cri.id + "Register.txt";
    const char* path = temp_p.c_str();
    remove(path);

    /*再删除总表课程*/
    deleteSpecificLine(temp_p, cri.id);

    if (cri.id != " ")
    {
        for (StudentNode* ptr = cri.head->next; ptr != NULL; ptr = ptr->next)
        {
            string temp_id = ptr->id;
            Student stu = nametable.studentHashTable.find(temp_id);

```

```

        stu.deleteItem(temp_id);

        /*课程注册名单里删除注册的学生*/
        ofstream ofs;
        string path = "dataBase\\StudentFiles\\" + stu.id +
"Schedule.txt";
        ofs.open(path, ios::out);
        deletespecificLine(path, cri.id);
        ofs.close();
    }

}
else
{
    cout << setw(34) << "查无此课" << endl;
}
}

void InsertCurriculum(NameTables& nametable)
{
    cout << "输入课程 课号 名称 学分" << endl;
    string id, name, credit;
    cin >> id >> name >> credit;

    while (!checkCri(id, name, credit))
    {
        cout << "请重新输入" << endl;
        cout << "输入 课号 名称 学分" << endl;
        cin >> id >> name >> credit;
    }

    Curriculum cri = nametable.curriculumHashTable.find(id);
    if (cri.id == " ")
    {
        cri = Curriculum(id, name, credit);
        nametable.curriculumHashTable.insert(cri);

        /*创建课程表文件*/
        ofstream ofs;
        string path = "dataBase\\CurriculumFiles\\" + id + "Register.txt";
        ofs.open(path, ios::out);
        ofs.close();

        /*在课程总表中添加名字*/
        ofstream namefile;
        namefile.open(CurriculumNameTable, ios::app);
        namefile << id << " " << name << " " << credit << endl;
        namefile.close();
    }
    else
    {
        cout << setw(34) << "已经有此课" << endl;
    }
}
}

```



```

void StudentRegisterForCurriculum(NameTables& nametable)
{
    cout << "输入学生 学号 姓名 性别 (male / female) " << endl;
    string id, name, gender;
    cin >> id >> name >> gender;

    while (!checkStu(id, name, gender))
    {
        cout << "请重新输入" << endl;
        cout << "输入 学号 姓名 性别 (male / female) " << endl;
        cin >> id >> name >> gender;
    }

    Student stu = nametable.studentHashTable.find(id);
    if (stu.id != " ")
    {
        cout << "输入课程课号" << endl;
        string id;
        cin >> id;

        while (!checkCriId(id))
        {
            cout << "请重新输入" << endl;
            cin >> id;
        }

        Curriculum cri = nametable.curriculumHashTable.find(id);
        if (cri.id == " ")
        {
            cout << setw(34) << "没有此课" << endl;
            return;
        }
        stu.addItem(cri.id);
        cri.addItem(stu.id);

        ofstream schedule;
        string path1 = "dataBase\\StudentFiles\\" + stu.id + "Schedule.txt";
        schedule.open(path1, ios::app);
        schedule << cri.id << " " << cri.name << " " << cri.credit << endl;
        schedule.close();

        ofstream registers;
        string path2 = "dataBase\\CurriculumFiles\\" + cri.id +
"Register.txt";
        registers.open(path2, ios::app);
        registers << stu.id << " " << stu.name << " " << stu.gender << endl;
        registers.close();
    }
    else
    {
        cout << setw(34) << "查无此人" << endl;
    }
}

```

```

void StudentQuitCurriculum(NameTables& nametable)
{
    cout << "输入学生学号" << endl;
    string s;
    cin >> s;

    while (!checkStuId(s))
    {
        cout << "请重新输入" << endl;
        cin >> s;
    }

    Student stu = nametable.studentHashTable.find(s);
    if (stu.id != " ")
    {
        cout << "输入课程课号" << endl;
        string criId;
        cin >> criId;

        while (!checkCriId(criId))
        {
            cout << "请重新输入" << endl;
            cin >> criId;
        }

        Curriculum cri = nametable.curriculumHashTable.find(criId);
        if (cri.id != " ")
        {
            stu.deleteItem(cri.id);
            cri.deleteItem(stu.id);

            /*文件*/
            ofstream schedule;
            string path1 = "dataBase\\StudentFiles\\" + stu.id +
"Schedule.txt";
            deleteSpecificLine(path1, cri.id);

            ofstream registers;
            string path2 = "dataBase\\CurriculumFiles\\" + cri.id +
"Register.txt";
            deleteSpecificLine(path2, stu.id);
        }

        else
        {
            cout << setw(34) << "查无此课" << endl;
        }
    }
    else
    {
        cout << setw(34) << "查无此人" << endl;
    }
}

```

```

void CurriculumEliminateStudent(NameTables& nametable)
{
    cout << "输入课程课号" << endl;
    string s;
    cin >> s;

    while (!checkCriId(s))
    {
        cout << "请重新输入" << endl;
        cin >> s;
    }

    Curriculum cri = nametable.curriculumHashTable.find(s);
    if (cri.id != " ")
    {
        cout << "输入学生学号" << endl;
        string stuId;
        cin >> stuId;

        while (!checkStuId(stuId))
        {
            cout << "请重新输入" << endl;
            cin >> stuId;
        }

        Student stu = nametable.studentHashTable.find(stuId);
        if (stu.id != " ")
        {
            stu.deleteItem(cri.id);
            cri.deleteItem(stu.id);

            ofstream schedule;
            string path1 = "dataBase\\StudentFiles\\" + stu.id +
"Schedule.txt";
            deleteSpecificLine(path1, cri.id);

            ofstream registers;
            string path2 = "dataBase\\CurriculumFiles\\" + cri.id +
"Register.txt";
            deleteSpecificLine(path2, stu.id);
        }

        else
        {
            cout << setw(34) << "查无此人" << endl;
        }
    }
    else
    {
        cout << setw(34) << "查无此课" << endl;
    }
}

```

```

}

void PrintRegisters(NameTables& nametable)
{
    cout << "输入课程课号: : " << endl;
    string id;
    cin >> id;

    while (!checkCriId(id))
    {
        cout << "请重新输入" << endl;
        cin >> id;
    }

    Curriculum cri = nametable.curriculumHashTable.find(id);
    for (StudentNode* ptr = cri.head->next; ptr != NULL; ptr = ptr->next)
    {
        string temp_id = ptr->id;
        nametable.studentHashTable.find(temp_id);
    }
}

void PrintSchedules(NameTables& nametable)
{
    cout << "输入学生学号: " << endl;
    string id;
    cin >> id;

    while (!checkStuId(id))
    {
        cout << "请重新输入" << endl;
        cin >> id;
    }

    Student stu = nametable.studentHashTable.find(id);
    for (CurriculumNode* ptr = stu.head->next; ptr != NULL; ptr = ptr->next)
    {
        string temp_id = ptr->id;
        nametable.curriculumHashTable.find(temp_id);
    }
}

int main()
{
    int choice = 0;
    NameTables nametable;

    init(nametable);
    system("cls");

    while (1)
    {
        openning();
        cout << "请输入功能序号: " << endl;
    }
}

```

```
cout << "输入栏=" << "==" << "====" << "==" << "输出栏=" << endl;
cin >> choice;

switch (choice)
{
case 1: /*输入所有学生信息*/
{
    BatchImportStudentInfo(nametable);
    system("pause");
    system("cls");
    break;
}

case 2: /*输入所有课程信息*/
{
    BatchImportCurriculumInfo(nametable);
    system("pause");
    system("cls");
    break;
}

case 3: /*查找学生记录*/
{
    FindStudent(nametable);
    system("pause");
    system("cls");
    break;
}

case 4: /*删除学生记录*/
{
    DeleteStudent(nametable);
    system("pause");
    system("cls");
    break;
}

case 5: /*插入学生记录*/
{
    InsertStudent(nametable);
    system("pause");
    system("cls");

    break;
}

case 6: /*查找课程记录*/
{
    FindCurriculum(nametable);
    system("pause");
    system("cls");
    break;
}

case 7: /*删除课程记录*/
```

```

{
    DeleteCurriculum(nametable);
    system("pause");
    system("cls");
    break;
}

case 8: /*插入课程记录*/
{
    InsertCurriculum(nametable);
    system("pause");
    system("cls");
    break;
}

case 9: /*学生注册课程*/
{
    StudentRegisterForCurriculum(nametable);
    system("pause");
    system("cls");
    break;
}

case 10: /*学生退出课程*/
{
    StudentQuitCurriculum(nametable);
    system("pause");
    system("cls");
    break;
}

case 11: /*课程退课学生*/
{
    CurriculumEliminateStudent(nametable);
    system("pause");

    system("cls");
    break;
}

case 12: /*输出选课名单*/
{
    PrintRegesters(nametable);
    system("pause");
    system("cls");
    break;
}

case 13: /*输出课程列表*/
{
    PrintSchedules(nametable);
    system("pause");
    system("cls");
}

```

```
        break;

    }

    default:
    {
        cout << setw(34) << "系统已关闭" << endl;
        return 0;
        break;
    }

    }
    system("cls");
}
return 0;
}
```