项目说明文档

数据结构课程设计

——考试报名系统

作 者 姓 名： 陈翔飞

学 号： 1851756

指 导 教 师： 张颖

学院、 专业： 软件学院 软件工程

同济大学

Tongji University

目 录

[0 项目须知 3](#_Toc27854902)

[0.1 编码及字体事项 3](#_Toc27854903)

[0.2 测试样例替换事项 3](#_Toc27854904)

[1 分析 3](#_Toc27854905)

[1.1 背景分析 3](#_Toc27854906)

[1.2 功能分析 3](#_Toc27854907)

[2 设计 4](#_Toc27854908)

[2.1 算法设计 4](#_Toc27854909)

[2.2 数据结构设计 4](#_Toc27854910)

[2.2 类结构设计 4](#_Toc27854911)

[2.3 成员与操作设计 4](#_Toc27854912)

[3 实现 12](#_Toc27854913)

[3.1 单次排课功能的实现 12](#_Toc27854914)

[3.1.1 单次排课功能流程图 12](#_Toc27854915)

[3.1.2 单次排课核心代码 13](#_Toc27854916)

[3.2 全体排课功能的实现 18](#_Toc27854917)

[3.2.1 全体排课功能流程图 18](#_Toc27854918)

[3.2.2 全体排课功能核心代码 18](#_Toc27854919)

[3.6 总体系统的实现 21](#_Toc27854920)

[3.6.1 总体系统流程图 21](#_Toc27854921)

[3.6.2 总体系统核心代码 22](#_Toc27854922)

[4 测试 24](#_Toc27854923)

[4.1 正常测试 24](#_Toc27854924)

[4.2 错误测试 32](#_Toc27854925)

[4.2.1 每学期开课数之和不等于课程总数 32](#_Toc27854926)

[4.2.2 课程存在循环依赖 33](#_Toc27854927)

# 项目须知

## 0.1 编码及字体事项

本项目文件编码均为GBK编码，如果评测时出现乱码，请把输入输出文件编码以及终端显示编码改为GBK编码。为保证输出文件中的格式化课表查看时美观，请将记事本的字体改为等宽字体（推荐使用console）。

## 测试样例替换事项

本项目测试样例均放在与可执行文件同目录的Test文件夹内，并带有中文解释。若要替换测试样例，请将对应样例内容复制入in.txt文件内（请将原内容覆盖），输出结果将写入out.txt文件中，如果是健壮性测试样例，out.txt文件将为空，错误提示会显示于终端上。

# 1 分析

## 背景分析

排课是每个学期都必须面临的一件繁重的日常教务工作，是为下学期教学工作能顺利组织实施的一个前提条件。排课管理是高校每个学期教学教务管理工作的一个主要内容，是整个综合教学教务管理系统中必不可少的部分。

排课也是一项复杂而精细的工作，需要考虑的因素很多，涉及到方方面面，归纳起来主要有五方面相互作用、相互关联的因素，即课程、教师、时间、班级以及教室。如何把各要素科学、规范、高效地进行优化组合，以保证教学秩序的稳定，促进教学质量的提高，这是每所高校都面临的课题。

## 1.2 功能分析

假设周一至周五上课，每天上10节课，第1大节为第1-2节课，第二大节为第3-5节课，第3大节为第6-7节课，第4大节为8-10节课，在排课时，如一门课程有3节课，则优先安排3节课连续上；如3节课连续无法按排，再优先安排两节课连续上，最后再安排单节课上的情况；如果一门课程需要安排上两天，为教学效果较好，最好不安排在相邻的两天，比如优先安排相隔2天上课。

1 输入数据包括：各学期所开的课程数（必须使每学期所开的课程数之和与课程总数相等），课程编号，课程名称，周学时数，指定开课学期，先决条件。如指定开课学期为0，表示有电脑自行指定开课学期。

2 如输入数据不合理，比如每学期所开的课程数值和与课程总数不相等，应显示适当的提示信息。

3 用文本文件存储输入数据，并且读入计算机。

4 用文本文件存储产生的各学期的课表。

# 2 设计

## 2.1 算法设计

课程之间存在先修关系，容易看出先修关系应该要满足偏序关系，否则会形成循环依赖，所以可以把课程编号作为顶点，根据先修关系连边，将所有课程储存在一个有向图中，然后对有向图做拓扑排序，每次取出入度为0的课程排入课表，然后把它的后继的入度减去一，以此类推。当排课完成后如果还有课未排入课表，说明课程存在循环关系，课程设计有问题，则在屏幕上打印出对应提示，不对输出文件做任何操作。

## 2.2 数据结构设计

如上功能分析所述，本排课系统需要将课程用图储存，所以需要图的数据结构。为了通过课程编号查找到对应课程，本项目选用了AVL平衡树，拓扑排序本项目选择使用宽度优先搜索的策略，所以还会需要队列的数据结构。

## 2.2 类结构设计

为了保证设计的数据结构的泛用性，本项目选择将Graph类和AVLTree类都设计为模板类。为了让模板类能实际应用于本项目，还设计了Course类、CourseTable类以及SchedulingSystem类，分别用于存储课程，存储课程表，以及将所有信息集合在一起提供排课的接口。同时为了动态读入信息，还设计了Vector类来辅助系统运行。

## 2.3 成员与操作设计

**树结点类（TreeNode）**

**类定义：**

1. **template** <**typename** Key, **typename** Value> **class** TreeNode
2. {
3. **public**:
4. **void** MidVisit(ostream& os);
5. TreeNode<Key, Value>() = **default**;
6. TreeNode<Key, Value>(Key K, Value V);
7. ~TreeNode<Key, Value>();
8. Key ElementKey;
9. Value ElementValue;
10. TreeNode<Key, Value>\* Left = nullptr, \*Right = nullptr;
11. **int** Height = 0;
12. };

**公有成员：**

Key ElementKey；//树节点的键，作为平衡树的关键词

Value ElementValue；//树节点的值，储存需要保存的信息

TreeNode<Key,Value>\* Left；//树节点的左儿子

TreeNode<Key,Value>\* Right；//树节点的右儿子

Int Height；//树节点的高度，用于维护平衡

**公有操作：**

TreeNode<Key, Value>() = default; //默认构造函数

TreeNode<Key, Value>(Key K, Value V); //含键值对参数的构造函数

void MidVisit(ostream& os);//中序遍历该节点并输出值

**AVL树类（AVLTree）**

**类定义：**

1. **template** <**typename** Key = **int**, **typename** Value = **int**> **class** AVLTree
2. {
4. **public**:
5. AVLTree<Key, Value>();
6. AVLTree<Key, Value>(Key K, Value V);
7. ~AVLTree<Key, Value>();
8. TreeNode<Key, Value>\* Change(Key K, Value V);
9. TreeNode<Key, Value>\* Insert(Key K, Value V = Value());
10. TreeNode<Key, Value>\* Find(Key K);
11. TreeNode<Key, Value>\* Delete(Key K);
12. **void** MakeEmpty();
13. **void** DisPlay(ostream& os);
14. **int** GetHeight();
15. TreeNode<Key, Value>\* UncheckChange(TreeNode<Key, Value>\* P, Value V);
16. TreeNode<Key, Value>\* UncheckInsert(Key K, Value V);
17. **int** GetSize();
18. **private**:
19. **int**  Max(**int** x, **int** y);
20. **int** GetHeight(TreeNode<Key, Value>\* P);
21. TreeNode<Key, Value>\* SingleRotateWithLeft(TreeNode<Key, Value>\* K2);
22. TreeNode<Key, Value>\* SingleRotateWithRight(TreeNode<Key, Value>\* K2);
23. TreeNode<Key, Value>\* DoubleRotateWithLeft(TreeNode<Key, Value>\* K3);
24. TreeNode<Key, Value>\* DoubleRotateWithRight(TreeNode<Key, Value>\* K3);
25. TreeNode<Key, Value>\* Find(Key K, TreeNode<Key, Value>\* P);
26. TreeNode<Key, Value>\* FindMin(TreeNode<Key, Value>\* P);
27. TreeNode<Key, Value>\* FindMax(TreeNode<Key, Value>\* P);
28. TreeNode<Key, Value>\* Insert(Key K, Value V, TreeNode<Key, Value>\* P);
29. TreeNode<Key, Value>\* Delete(Key K, TreeNode<Key, Value>\* P);
30. TreeNode<Key, Value>\* MakeEmpty(TreeNode<Key, Value>\* P);
31. **int** Size = 0;
32. TreeNode<Key, Value>\* Root = nullptr;
33. };

**私有成员：**

int Size = 0;//AVL树的大小，即储存键值对数

TreeNode<Key, Value>\* Root ;//AVL树的树根

**私有操作：**

int Max(int x, int y); //返回x，y中的最大值

int GetHeight(TreeNode<Key, Value>\* P); //获取节点P的高度

TreeNode<Key, Value>\* SingleRotateWithLeft(TreeNode<Key, Value>\* K2);

//左单旋，用于维护平衡

TreeNode<Key, Value>\* SingleRotateWithRight(TreeNode<Key, Value>\* K2);

//右单旋，用于维护平衡

TreeNode<Key, Value>\* DoubleRotateWithLeft(TreeNode<Key, Value>\* K3);

//左双旋，用于维护平衡

TreeNode<Key, Value>\* DoubleRotateWithRight(TreeNode<Key, Value>\* K3);

//右双旋，用于维护平衡

TreeNode<Key, Value>\* Find(Key K, TreeNode<Key, Value>\* P);

//以节点P为根，查找有关键词K的节点

TreeNode<Key, Value>\* FindMin(TreeNode<Key, Value>\* P);

//以节点P为根，查找最小节点

TreeNode<Key, Value>\* FindMax(TreeNode<Key, Value>\* P);

//以节点P为根，查找最大节点

TreeNode<Key, Value>\* Insert(Key K, Value V, TreeNode<Key, Value>\* P);

//以节点P为根，插入键值对<K,V>

TreeNode<Key, Value>\* Delete(Key K, TreeNode<Key, Value>\* P);

//以节点P为根，删除有关键词K的节点

TreeNode<Key, Value>\* MakeEmpty(TreeNode<Key, Value>\* P);

//清空节点P以及P的所有子节点并释放内存

**公有操作：**

AVLTree<Key, Value>();

//AVLTree的默认构造函数

AVLTree<Key, Value>(Key K, Value V);

//AVLTree的带键值对参数的构造函数，以参数中的键值对开辟一个树节点并使Root指向该节点

~AVLTree<Key, Value>();

//AVLTree的析构函数，通过调用MakeEmpty()实现对内存的回收

TreeNode<Key, Value>\* Change(Key K, Value V);

//将树中以K为键的节点的值改为V，如果不存在就插入键值对<K,V>

TreeNode<Key, Value>\* Insert(Key K, Value V = Value());

//将键值对<K,V>插入树中，如果键K已经存在则将对应节点中的值改为V

TreeNode<Key, Value>\* Find(Key K);

//返回键为K的树节点指针

TreeNode<Key, Value>\* Delete(Key K);

//删除键为K的树节点

void MakeEmpty();

//清空整个树并释放内存

void DisPlay(ostream& os);

//中序展示整棵树的所有树节点

int GetHeight();

//返回树的高度

TreeNode<Key, Value>\* UncheckChange(TreeNode<Key, Value>\* P, Value V);

//直接将节点P的值改为V

TreeNode<Key, Value>\* UncheckInsert(Key K, Value V);

//不检查K是否存在于树中，直接插入键值对<K,V>，如果插入失败则返回空

int GetSize();

//返回已经储存的键值对数量

**队列结点类（Node）**

**类定义：**

1. **template**<**typename** ElementType> **class** Node
2. {
3. **private**:
4. **friend** **class** Queue<ElementType>;
5. Node<ElementType>() : Pre(nullptr), Next(nullptr) {};
6. Node<ElementType>(**const** ElementType& e) :Element(e) {};
7. Node<ElementType>\* Next=nullptr, \*Pre=nullptr;
8. ElementType Element;
9. };

**私有成员：**

Node<ElementType>\* Next=nullptr; //队列中的前一个节点

Node<ElementType>\* Pre=nullptr;//队列中的后一个节点

ElementType Element;//队列元素

**私有操作：**

Node<ElementType>() : Pre(nullptr), Next(nullptr) {};

//默认构造函数

Node<ElementType>(const ElementType& e) :Element(e) {};

//拷贝构造函数

**队列类（Queue）**

**类定义：**

1. **template**<**typename** ElementType> **class** Queue
2. {
3. **public**:
4. Queue<ElementType>();
5. ~Queue<ElementType>();
6. **int** GetSize();
7. **bool** Empty();
8. **void** EnQueue(ElementType E);
9. **void** DeQueue();
10. ElementType Front();
11. **void** Clear();
12. **private**:
13. **int** Size=0;
14. Node<ElementType>\* Head, \*Tail;
15. };

**私有成员：**

int Size = 0;//队列的大小;

Node<ElementType>\* Head;//队首节点

Node<ElementType>\* Tail;//队尾节点

**公有操作：**

Queue<ElementType>();

//默认构造函数

~Queue<ElementType>();

//析构函数，通过调用Clear（）来回收内存

int GetSize();

//返回队列大小

bool Empty();

//判断队列是否为空

void EnQueue(ElementType E);

//入队操作

void DeQueue();

//出队操作

ElementType Front();

//返回队首

void Clear();

//清空队列，回收内存

**向量类（Vector）**

**类定义：**

1. **template**<**typename** ElementType> **class** Vector
2. {
3. **public**:
4. ~Vector<ElementType>();
5. Vector<ElementType>() = **default**;
6. Vector<ElementType>(**const** Vector<ElementType> & v);
7. Vector<ElementType>& operator = (**const** Vector<ElementType>&v);
8. **void** PushBack(**const** ElementType& t);
9. **void** PopBack();
10. **void** Clear();
11. **int** GetSize() **const**;
12. **void** ReSize(**int** NewSize);
13. ElementType& operator[](**int** Index) **const**;
14. **bool** Empty()**const**;
15. **const** ElementType& Back() **const**;
16. **private**:
17. **void** Extend();
18. **int** Size = 0;
19. **int** Capacity = 0;
20. ElementType\* Array = nullptr;
21. };

**私有成员：**

int Size;//Vector中实际储存的元素数量

int Capacity;//Vector已经申请的空间

ElementType\* Array;//储存的数据的起始地址

**私有操作：**

void Extend();

//扩容函数，当容量不足时调用

**公有操作：**

~Vector<ElementType>();

//析构函数，通过调用Clear()来释放内存

Vector<ElementType>() = default;

//默认构造函数

Vector<ElementType>(const Vector<ElementType> & v);

//拷贝构造函数

Vector<ElementType>& operator = (const Vector<ElementType>&v);

//重载=运算符，使该类支持赋值运算

void PushBack(const ElementType& t);

//向Vector末尾添加一个元素

void PopBack();

//删除末尾的元素

void Clear();

//清空Vector，释放内存

int GetSize() const;

//返回储存元素的数量

void ReSize(int NewSize);

//重设Vector的大小

ElementType& operator[](int Index) const;

//重载[]运算符，使Vector可以像数组一样使用

bool Empty()const;

//判断Vector是否为空

const ElementType& Back() const;

//返回末尾的元素

**课程类（Vector）**

**类定义：**

1. **class** Course
2. {
3. **public**:
4. Course() = **default**;
5. string Num, Name;
6. **int** ClassHours;
7. **int** OpenTerm;
8. **int** EarliestOpenTerm = 0;
9. Vector<string> PreCourse;
10. };

**公有成员：**

string Num;//课程编号

string Name;//课程名称

int ClassHours;//课程学时数

int OpenTerm;//指定开课学期

int EarliestOpenTerm ;//最早开课学期

Vector<string> PreCourse;//要求的先修课程编号

**课程表类（Vector）**

**类定义：**

1. **class** CourseTable
2. {
3. **public**:
4. **const** **int**  Gap = 8;
5. **const** **int** ExtraGap = 4;
6. **int** MaxCourseNum;
7. **int** CourseCnt;
8. string Table[5][10];
9. **void** DisPlay(ostream& os);
11. };

**公有成员：**

const int Gap = 8;//格式化输出时的课程单行最大长度

const int ExtraGap = 4;//格式化输出时每列额外间隔

int MaxCourseNum;//该学期最大开课数

int CourseCnt;//该学期已安排课程数

string Table[5][10];//课程表

**公有操作：**

void DisPlay(ostream& os);

//向流os格式化输出排好的课表

**排课系统类（SchedulingSystem）**

**类定义：**

1. **class** SchedulingSystem
2. {
3. **public**:
4. CourseTable TermTable[8];
5. Graph<string> CourseNet;
6. AVLTree<**int**, Course> CourseMap;
7. **void** PutCourse(Course& CurCourse, **int**& CurTerm);
8. **void** Arrange();
9. }; ;

**公有成员：**

CourseTable TermTable[8]; //八学期的课程表

Graph<string> CourseNet;//按课程关系储存的课表网络

AVLTree<int, Course> CourseMap;//储存编号—课程键值对的平衡树

**公有操作：**

void PutCourse(Course& CurCourse, int& CurTerm);

//将单个课程排入当前学期

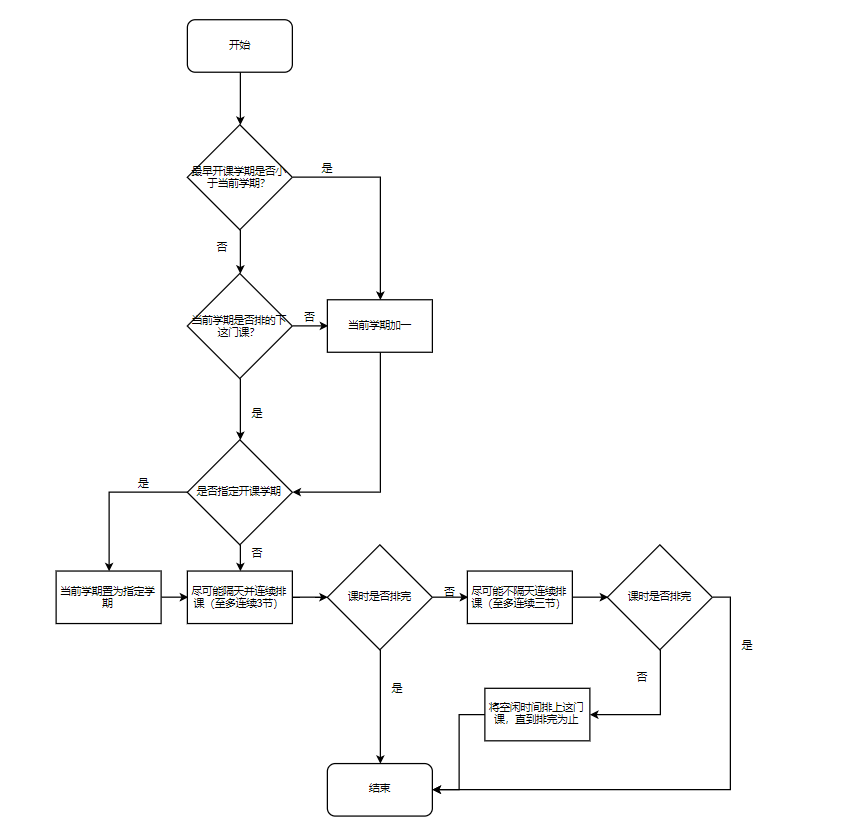
void Arrange();

//全体排课接口

# 3 实现

## 3.1 单次排课功能的实现

### 3.1.1 单次排课功能流程图

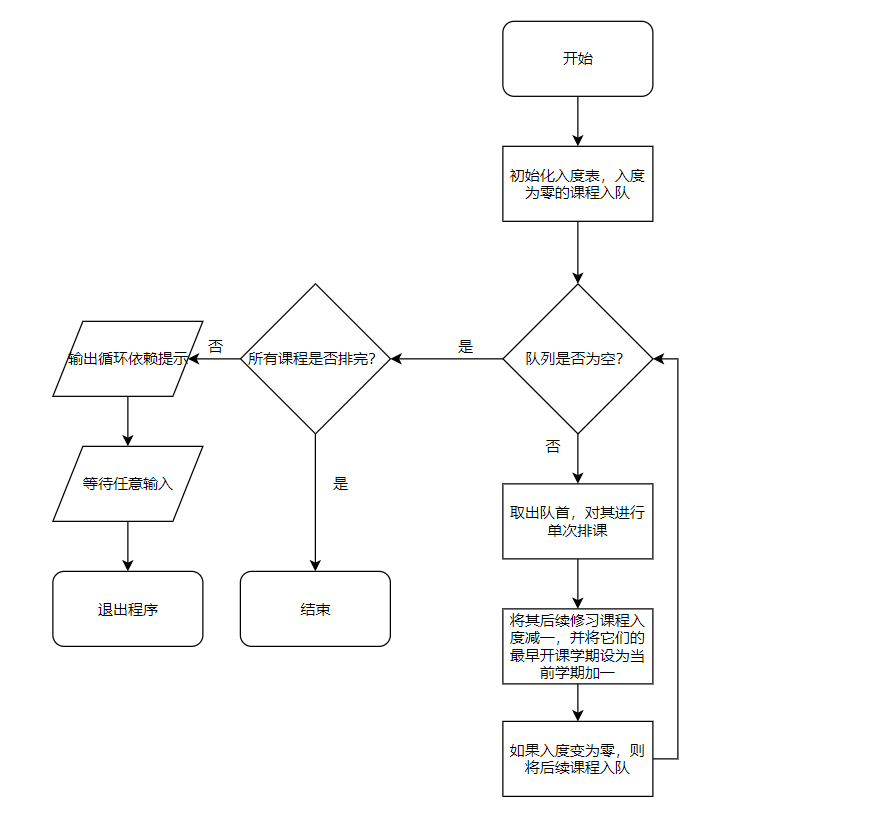


### 3.1.2 单次排课核心代码

1. **void** PutCourse(Course& CurCourse, **int**& CurTerm)
2. {
3. **int** Temp = CurTerm;
4. **if** (CurCourse.EarliestOpenTerm > CurTerm) CurTerm++;//如果先修课在这学期不被满足，则排下一个学期
5. **if** (CurTerm >= 8)
6. {
7. cout << "课程设计存在问题！排课失败！" << endl;
8. cout << "按任意键退出！" << endl;
9. getchar();
10. exit(-1);//八个学期已经排完;
11. }
12. **if** (TermTable[CurTerm].CourseCnt == TermTable[CurTerm].MaxCourseNum) CurTerm++;//这学期课已经排满，则排下一个学期
13. **bool** Recover = **false**;
15. **int** i, j;
16. **int** ResCourse = 0;
17. **bool** IsPut[5] = { **false**,**false**,**false**,**false**,**false** };
18. **for** (i = 0; i < 5; ++i)
19. {
20. **for** (j = 0; j < 10; ++j)
21. {
22. **if** (TermTable[CurTerm].Table[i][j].empty()) ResCourse++;
23. }
24. }
26. **if** (CurCourse.OpenTerm != 0)
27. {
28. CurTerm = CurCourse.OpenTerm-1;
29. Recover = **true**;
30. }
31. **if** (CurTerm >= 8)
32. {
33. cout << "课程设计存在问题！排课失败！" << endl;
34. cout << "按任意键退出！" << endl;
35. getchar();
36. exit(-1);//八个学期已经排完
37. }
38. **if** (ResCourse < CurCourse.ClassHours) CurTerm++;//如果排不下，则排下一学期
39. **int** ResHour = CurCourse.ClassHours;
40. TermTable[CurTerm].CourseCnt++;//该学期排课数量加一
41. **if** (ResHour >= 3)//尝试隔天连续三节排课
42. {
43. **for** (i = 0; i < 5; ++i)
44. {
45. **if** (ResHour < 3)**break**;
46. **if** (IsPut[i] == **true**) **continue**;
47. **if** (i > 0 && IsPut[i - 1] == **true**)**continue**;
48. **if** (i < 4 && IsPut[i + 1] == **true**) **continue**;
49. **for** (j = 0; j < 8; ++j)
50. {
51. **if** (ResHour < 3)**break**;
52. **if** (j == 3)
53. {
54. ++j;
55. **continue**;
56. }
57. **if** (TermTable[CurTerm].Table[i][j].empty() && TermTable[CurTerm].Table[i][j + 1].empty() && TermTable[CurTerm].Table[i][j + 2].empty())//如果接下来三节都空闲,则排上
58. {
59. TermTable[CurTerm].Table[i][j] = CurCourse.Name;
60. TermTable[CurTerm].Table[i][j + 1] = CurCourse.Name;
61. TermTable[CurTerm].Table[i][j + 2] = CurCourse.Name;
62. ResHour -= 3;
63. IsPut[i] = **true**;
64. **break**;
65. }
66. }
67. }
68. }
69. **if** (ResHour >= 2)//尝试隔天连续两节排课
70. {
71. **for** (i = 0; i < 5; ++i)
72. {
73. **if** (ResHour < 2)**break**;
74. **if** (IsPut[i] == **true**) **continue**;
75. **if** (i > 0 && IsPut[i - 1] == **true**)**continue**;
76. **if** (i < 4 && IsPut[i + 1] == **true**) **continue**;
77. **for** (j = 0; j < 9; ++j)
78. {
79. **if** (ResHour < 2) **break**;
80. **if** (j == 4) **continue**;
81. **if** (TermTable[CurTerm].Table[i][j].empty() && TermTable[CurTerm].Table[i][j + 1].empty())//如果接下来两节都空闲,则排上
82. {
83. TermTable[CurTerm].Table[i][j] = CurCourse.Name;
84. TermTable[CurTerm].Table[i][j + 1] = CurCourse.Name;
85. ResHour -= 2;
86. IsPut[i] = **true**;
87. **break**;
88. }
89. }
90. }
91. }
92. **if** (ResHour >= 1)//尝试隔天排课
93. {
94. **for** (i = 0; i < 5; ++i)
95. {
96. **if** (ResHour < 1)**break**;
97. **if** (IsPut[i] == **true**)**continue**;
98. **if** (i > 0 && IsPut[i - 1] == **true**)**continue**;
99. **if** (i < 4 && IsPut[i + 1] == **true**) **continue**;
100. **for** (j = 0; j < 10; ++j)
101. {
102. **if** (TermTable[CurTerm].Table[i][j].empty() && TermTable[CurTerm].Table[i][j + 1].empty())//如果这节空闲,则排上
103. {
104. TermTable[CurTerm].Table[i][j] = CurCourse.Name;
105. ResHour -= 1;
106. IsPut[i] = **true**;
107. **break**;
108. }
109. }
110. }
111. }
112. **if** (ResHour >= 3)//尝试不隔天连续三节排课
113. {
114. **for** (i = 0; i < 5; ++i)
115. {
116. **if** (ResHour < 3)**break**;
117. **for** (j = 0; j < 8; ++j)
118. {
119. **if** (ResHour < 3)**break**;
120. **if** (j == 3)
121. {
122. ++j;
123. **continue**;
124. }
125. **if** (TermTable[CurTerm].Table[i][j].empty() && TermTable[CurTerm].Table[i][j + 1].empty() && TermTable[CurTerm].Table[i][j + 2].empty())//如果接下来三节都空闲,则排上
126. {
127. TermTable[CurTerm].Table[i][j] = CurCourse.Name;
128. TermTable[CurTerm].Table[i][j + 1] = CurCourse.Name;
129. TermTable[CurTerm].Table[i][j + 2] = CurCourse.Name;
130. ResHour -= 3;
131. IsPut[i] = **true**;
132. }
133. }
134. }
135. }
136. **if** (ResHour >= 2)//尝试不隔天连续两节排课
137. {
138. **for** (i = 0; i < 5; ++i)
139. {
140. **if** (ResHour < 2)**break**;
141. **for** (j = 0; j < 9; ++j)
142. {
143. **if** (ResHour < 2) **break**;
144. **if** (j == 4) **continue**;
145. **if** (TermTable[CurTerm].Table[i][j].empty() && TermTable[CurTerm].Table[i][j + 1].empty())//如果接下来两节都空闲,则排上
146. {
147. TermTable[CurTerm].Table[i][j] = CurCourse.Name;
148. TermTable[CurTerm].Table[i][j + 1] = CurCourse.Name;
149. ResHour -= 2;
150. IsPut[i] = **true**;
151. }
152. }
153. }
154. }
155. **if** (ResHour >= 1)//仍然未排完
156. {
157. **for** (i = 0; i < 5; ++i)
158. {
159. **if** (ResHour < 1)**break**;
160. **for** (j = 0; j < 10; ++j)
161. {
162. **if** (TermTable[CurTerm].Table[i][j].empty() && TermTable[CurTerm].Table[i][j + 1].empty())//如果这节空闲,则排上
163. {
164. TermTable[CurTerm].Table[i][j] = CurCourse.Name;
165. ResHour -= 1;
166. }
167. }
168. }
169. }
170. **if** (Recover)
171. CurTerm = Temp;
172. }

## 3.2 全体排课功能的实现

### 3.2.1 全体排课功能流程图



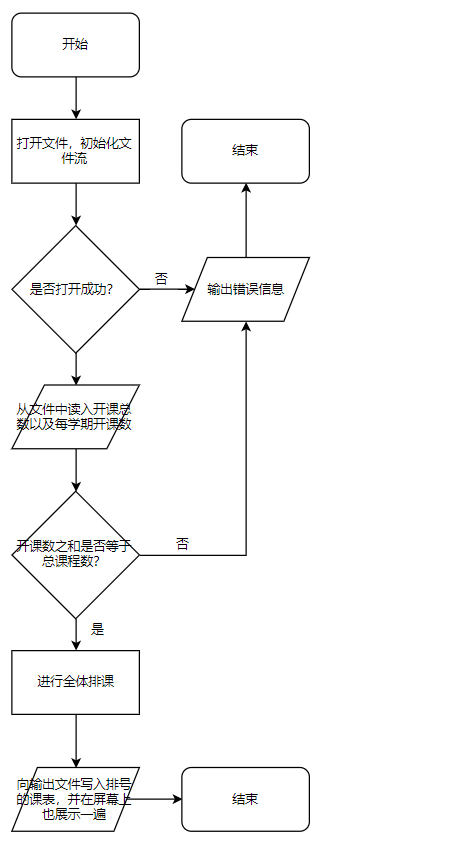
### 3.2.2 全体排课功能核心代码

**用户界面接口：**

1. **void** Arrange()
2. {
3. **int** CurTerm = 0;
4. **int** CourseCnt = 0;
5. Vector<**int**> Indegrees;
6. Queue<**int**> MaxVertexes;
7. **int** i;
8. **for** (i = 0; i < CourseNet.Vertexes.GetSize(); ++i)
9. {
10. Indegrees.PushBack(CourseNet.Vertexes[i]->InDegree);
11. }
12. **for** (i = 0; i < Indegrees.GetSize(); ++i)
13. {
14. **if** (CourseMap.Find(i)->ElementValue.OpenTerm != 0) MaxVertexes.EnQueue(i);
15. }
16. **for** (i = 0; i < Indegrees.GetSize(); ++i)
17. {
18. **if** (Indegrees[i]==0&&CourseMap.Find(i)->ElementValue.OpenTerm == 0) MaxVertexes.EnQueue(i);
19. }
20. **while** (!MaxVertexes.Empty())
21. {
23. **int** Cur = MaxVertexes.Front();
24. MaxVertexes.DeQueue();
25. auto CurPtr = CourseMap.Find(Cur);
26. auto CurCourse = CurPtr->ElementValue;
27. **int** k = 0;
28. k++;
29. PutCourse(CurCourse, CurTerm);
30. CourseCnt++;
31. **for** (i = 0; i < CourseNet.Vertexes[Cur]->Next.GetSize(); ++i)
32. {
33. **int** NextVertexes = CourseNet.Vertexes[Cur]->Next[i]->Index;
34. Indegrees[NextVertexes]--;
35. **if** (Indegrees[NextVertexes] == 0)
36. {
37. MaxVertexes.EnQueue(NextVertexes);
38. CourseMap.Find(NextVertexes)->ElementValue.EarliestOpenTerm = (CurTerm + 1);
39. }
40. }
42. }
43. **if** (CourseCnt < CourseMap.GetSize())
44. {
45. cout << "课程出现循环依赖关系！" << endl;
46. cout << "按任意键退出！" << endl;
47. getchar();
48. exit(-1);
49. }
50. }

## 3.6 总体系统的实现

### 3.6.1 总体系统流程图



### 3.6.2 总体系统核心代码

1. **int** main(**void**)
2. {
4. SchedulingSystem MySchedulingSystem;
5. Vector<Course> AllCourse;
7. in.open("in.txt", ios\_base::in | ios\_base::binary);
8. out.open("out.txt", ios\_base::out | ios\_base::binary);
9. **if** (in.is\_open() == **false** || out.is\_open() == **false**)
10. {
11. cout << "文件打开失败，请检查文件是否正确命名！" << endl;
12. cout << "按任意键退出 ！" << endl;
13. getchar();
14. exit(-1);
15. }
16. **int** n, i, j;
17. in >> n;
19. **int** SumCourseNum = 0;
20. **for** (i = 0; i < 8; ++i)
21. {
22. in >> MySchedulingSystem.TermTable[i].MaxCourseNum;
23. SumCourseNum+=MySchedulingSystem.TermTable[i].MaxCourseNum;
24. }
25. **while** (in.get() != '\n') **continue**;
26. **if** (SumCourseNum != n)
27. {
28. cout << "开课数量和每学期开课数总和不同！请检查课程设计是否出错！" << endl;
29. cout << "按任意键退出 ！" << endl;
30. getchar();
31. exit(-1);
32. }
33. AllCourse.ReSize(n);
34. **for** (i = 0; i < n; ++i)
35. {
36. string s;
37. getline(in, s);
38. stringstream ss(s);
39. ss >> AllCourse[i].Num;
40. ss >> AllCourse[i].Name;
41. Check(AllCourse[i].Name);
42. ss >> AllCourse[i].ClassHours;
43. ss >> AllCourse[i].OpenTerm;
44. **while** (ss >> s)
45. {
46. AllCourse[i].PreCourse.PushBack(s);
47. }
48. MySchedulingSystem.CourseMap.Insert(i, AllCourse[i]);
49. }
50. in.close();
51. **for** (i = 0; i < n; ++i)
52. {
53. MySchedulingSystem.CourseNet.AddVertex(AllCourse[i].Num);
54. }
55. **for** (i = 0; i < n; ++i)
56. {
57. **for** (j = 0; j < AllCourse[i].PreCourse.GetSize(); ++j)
58. {
59. MySchedulingSystem.CourseNet.Link(AllCourse[i].PreCourse[j], AllCourse[i].Num);
60. }
61. }
62. MySchedulingSystem.Arrange();
63. **for** (i = 0; i < 8; ++i)
64. {
65. //将结果打印至屏幕
66. cout << "第" << i +1<< "学期课表为:" << endl;
67. MySchedulingSystem.TermTable[i].DisPlay(cout);
68. cout << endl << endl;
69. //将结果写入文件
70. out << "第" << i +1<< "学期课表为:" << endl;
71. MySchedulingSystem.TermTable[i].DisPlay(out);
72. out << endl << endl;
73. }
74. cout << "课表安排完成！" << endl;
75. cout << "请按任意键退出" << endl;
76. getchar();
77. out.close();
78. **return** 0;
79. }

# 4 测试

## 4.1 正常测试

**测试用例**：

38

5 5 5 2 5 5 6 5

c01 程序设计基础 5 0

c02 离散数学 6 0 c01

c03 数据结构算法 4 0 c01 c02

c04 汇编语言 5 0 c01

c05 算法设计 4 0 c03 c04

c06 计算机组成原理 6 0

c07 微机原理 4 0 c03

c08 单片机应用 3 0 c03

c09 编译原理 5 0 c03

c10 操作系统原理 4 0 c03

c11 数据库原理 5 0 c03

c12 高等数学 6 0

c13 线性代数 6 0

c14 数值分析 6 0 c12

c15 普通物理 4 0 c12

c16 计算机文化 3 0

c17 计算机系统结构 6 0 c06

c18 计算机网络 5 0 c03

c19 数据通信 6 0

c20 面向对象程序设计 3 0 c01 c03

c21 Java 3 0 c01 c03

c22 C#.net 5 0 c01 c03

c23 PowerBuilder 5 0 c01 c03

c24 VC++ 3 0 c01 c03

c25 ASP程序设计 5 0 c01 c03

c26 JSP程序设计 5 0 c01 c03

c27 VB.net 5 0 c01 c03

c28 Delphi 5 0 c01 c03

c29 C++Builder 5 0 c01 c03

c30 英语 5 1

c31 英语 5 2

c32 英语 5 3

c33 英语 5 4

c34 英语 5 5

c35 英语 5 6

c36 英语 5 7

c37 英语 5 8

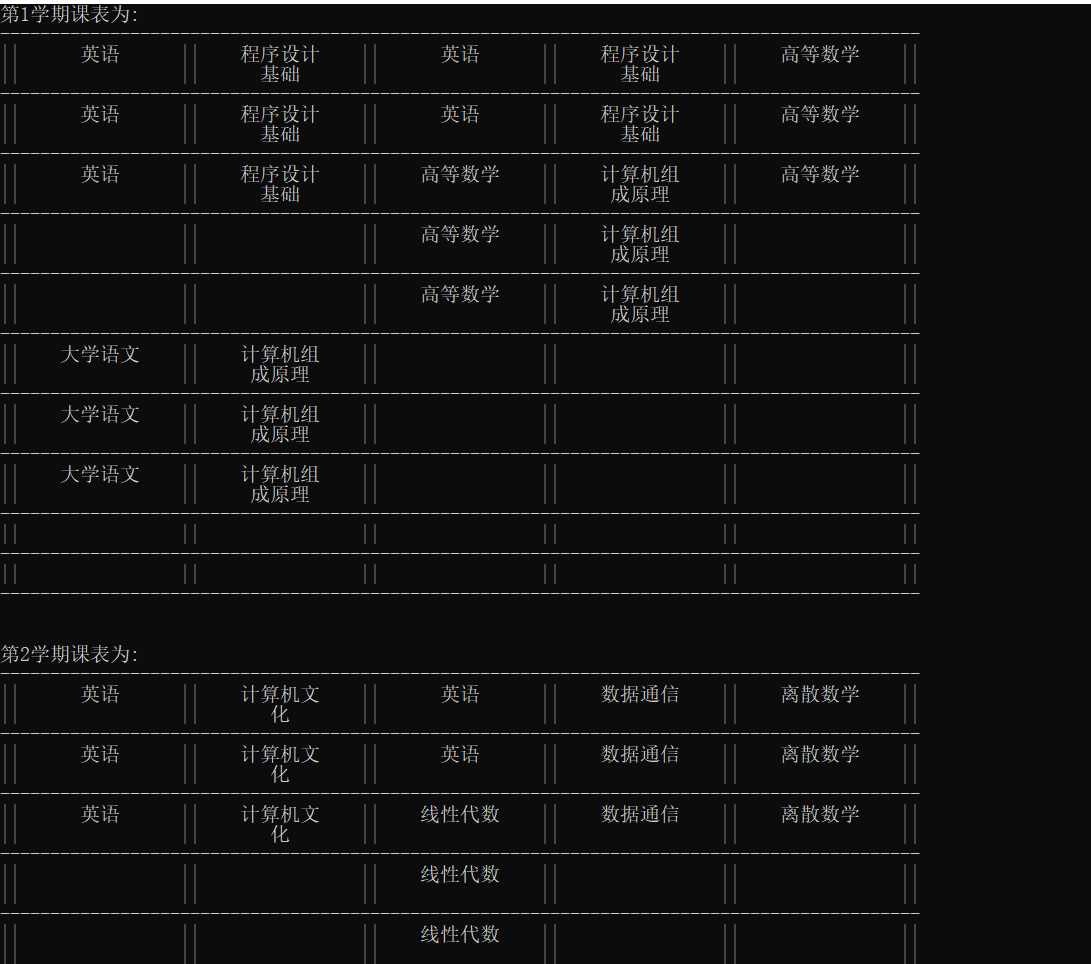
c38 大学语文 3 1

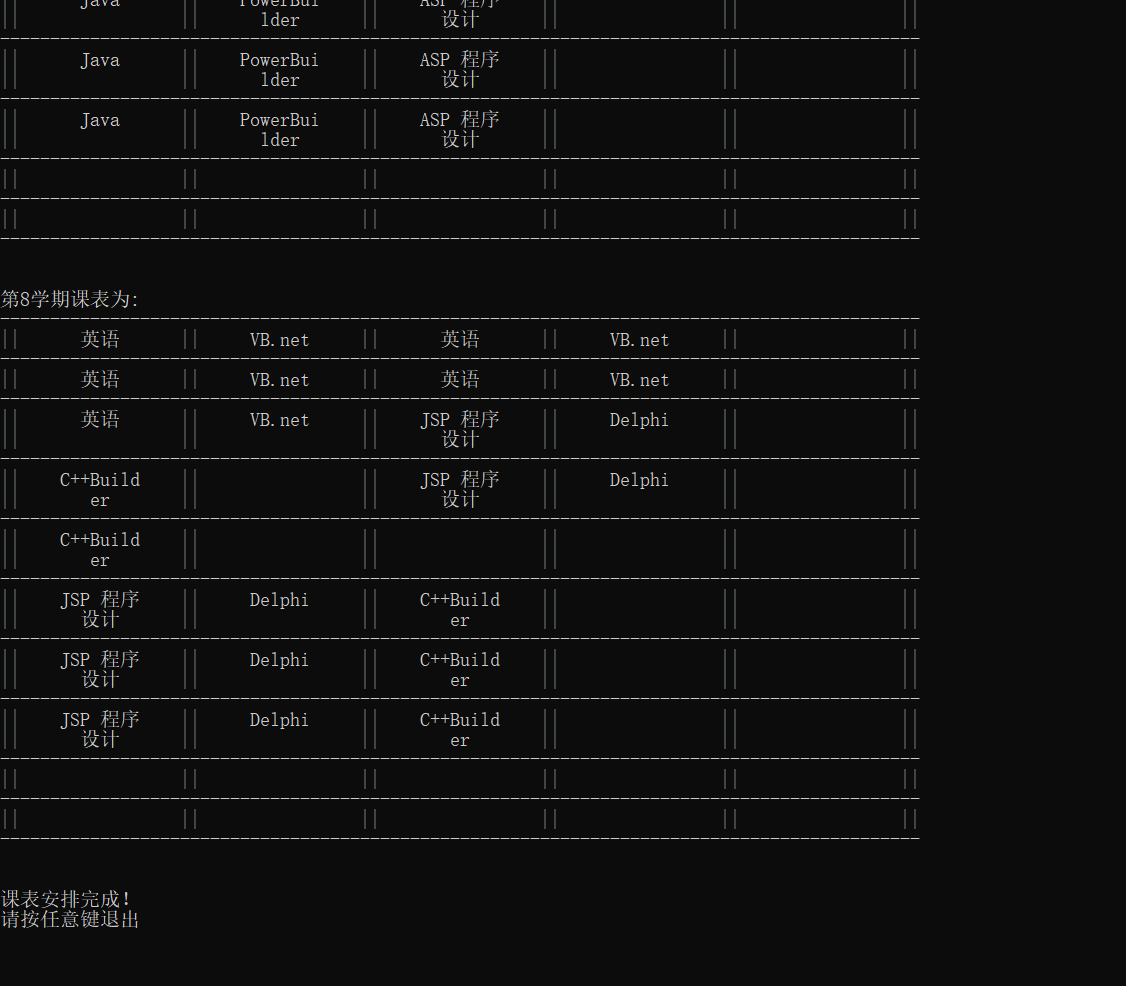
**预期结果**：

排课成功，文件中写入八个学期的课表

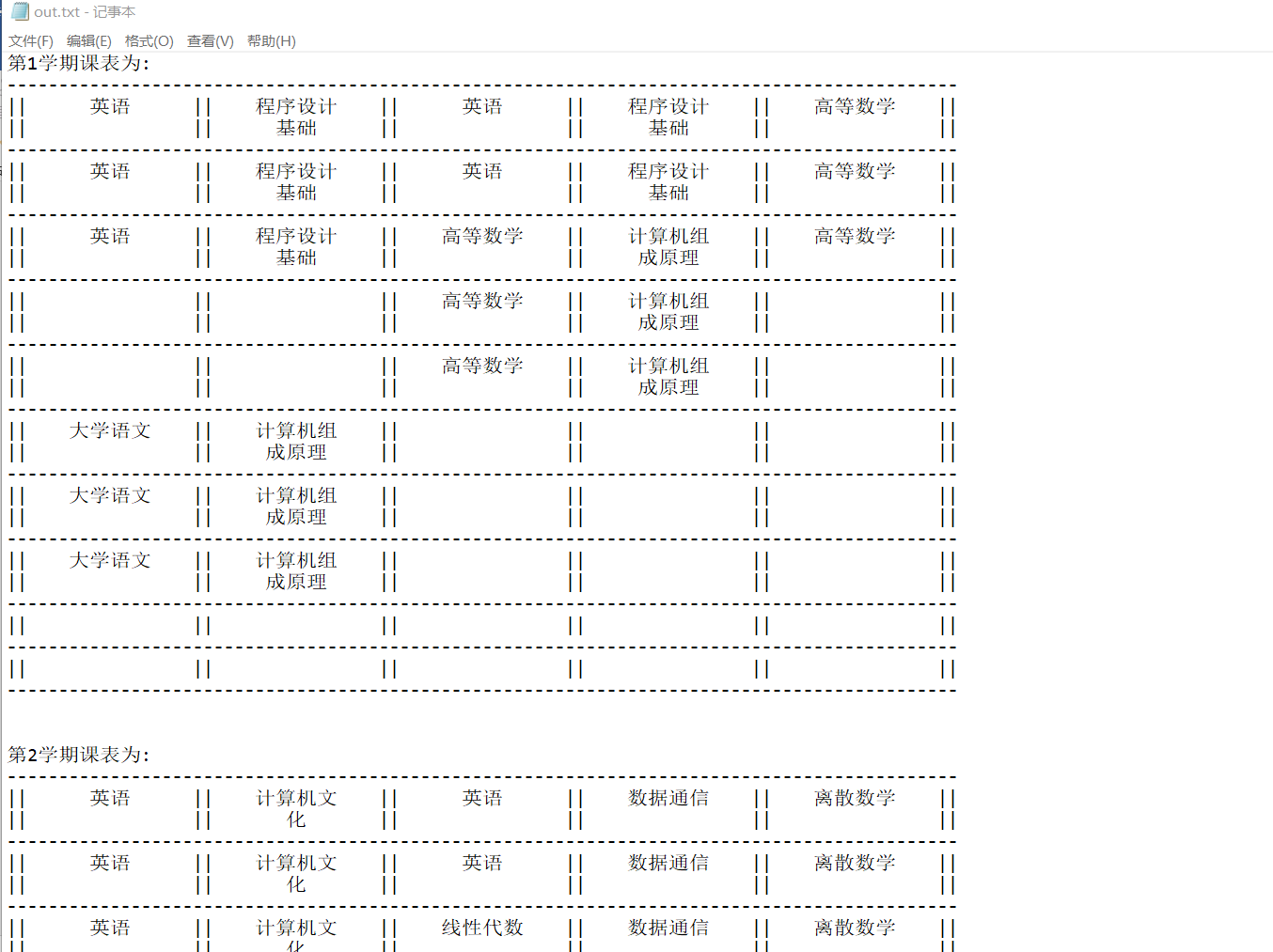
**实验结果：**

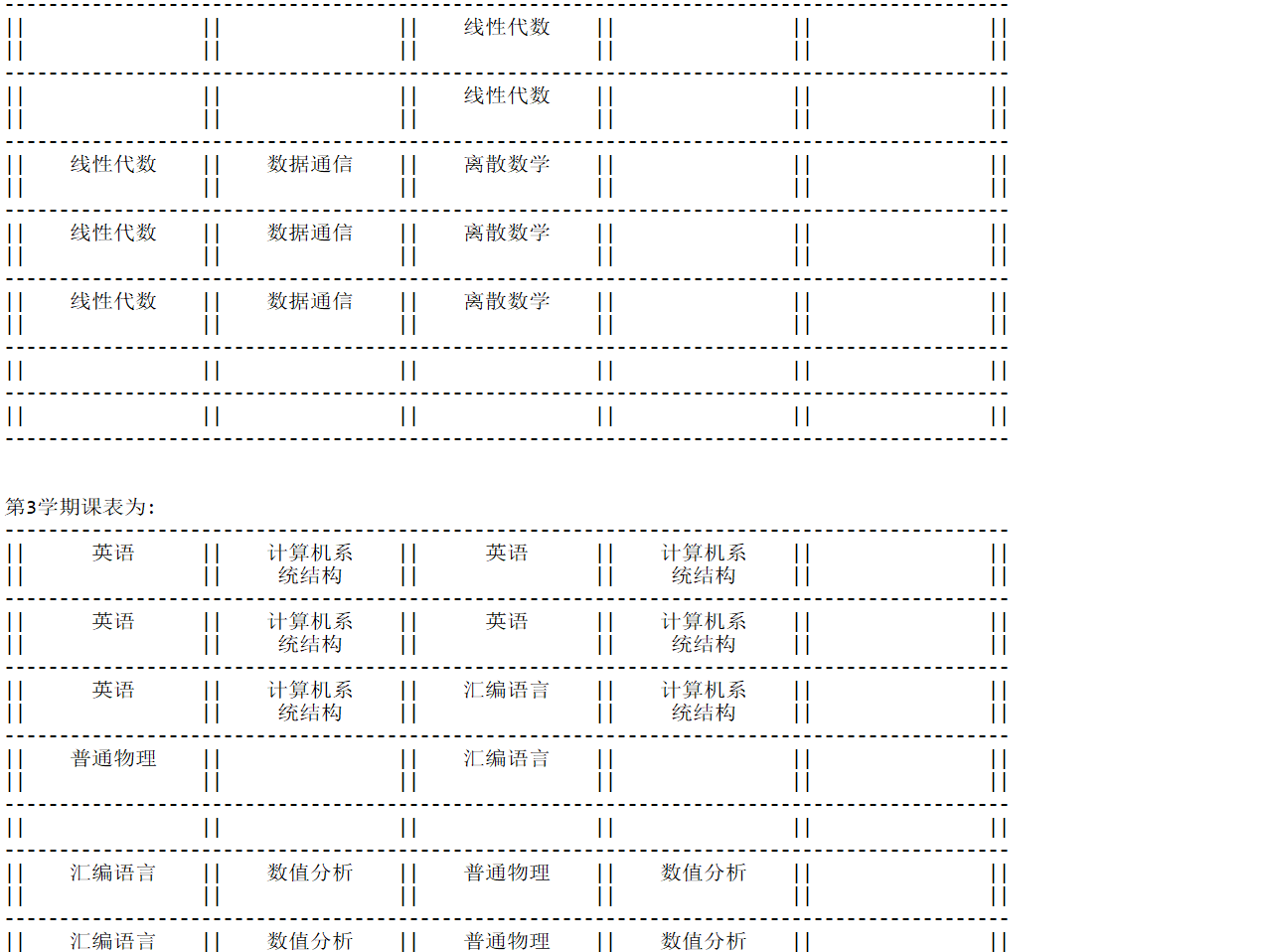
**终端结果：**

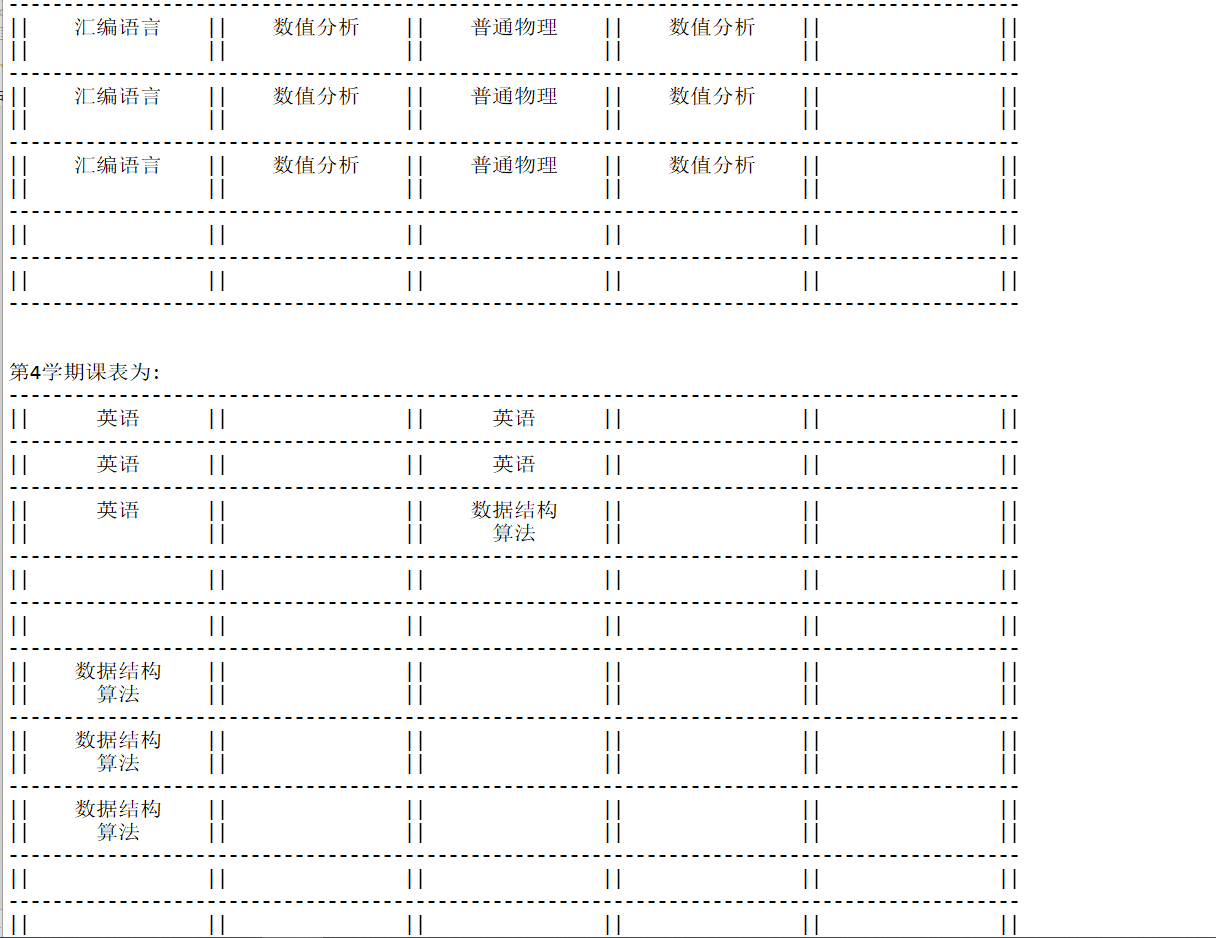


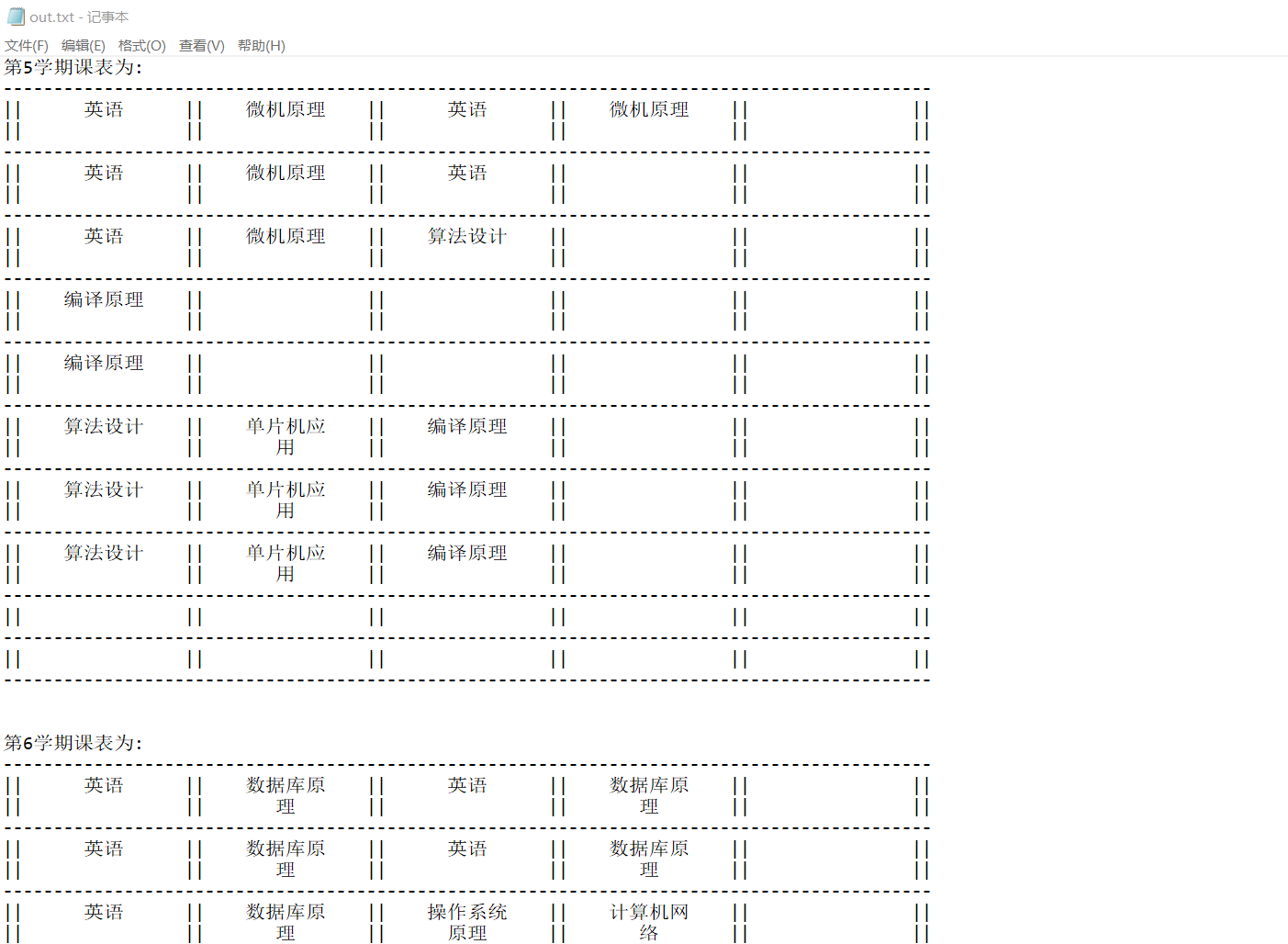


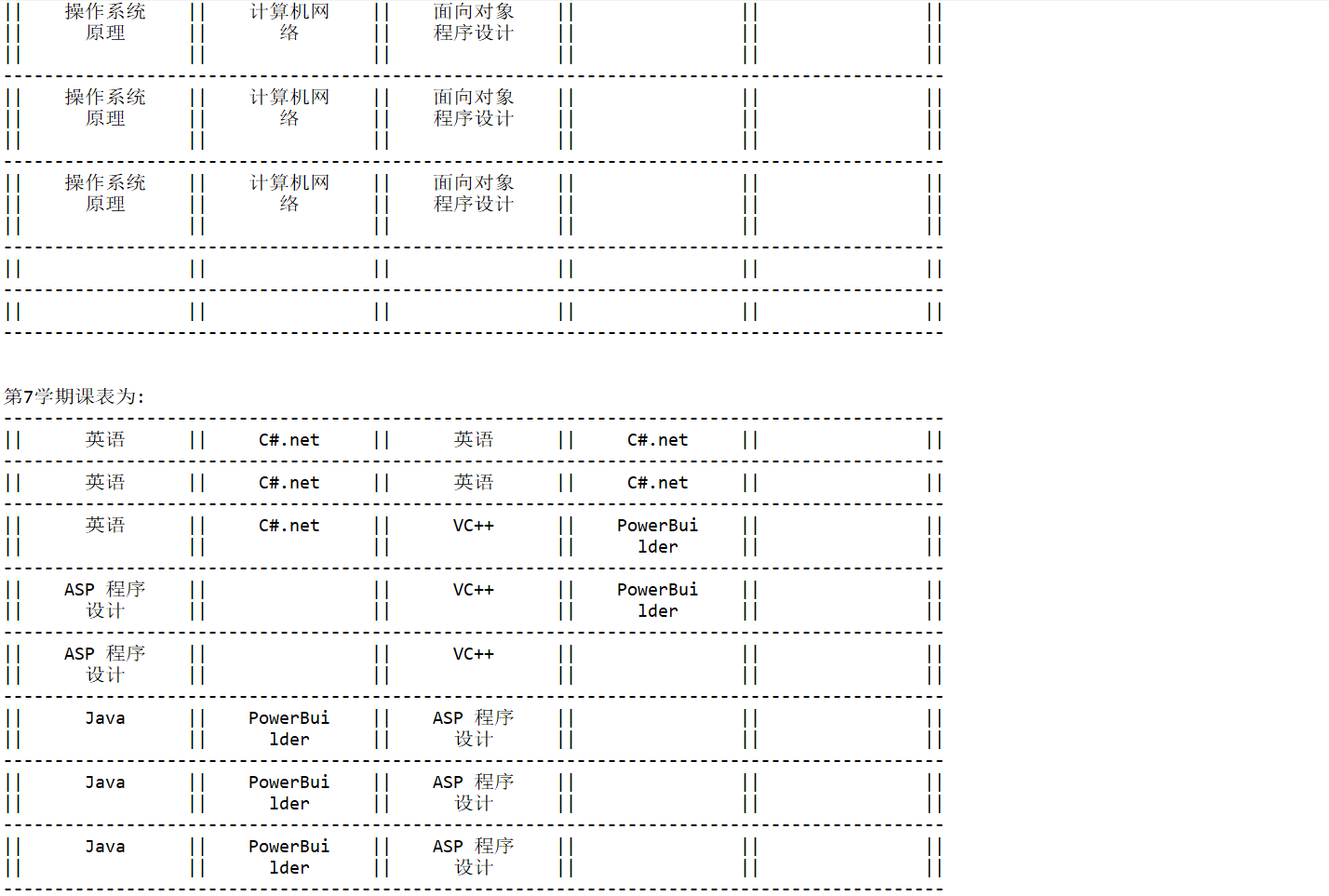
**输出文件结果：**

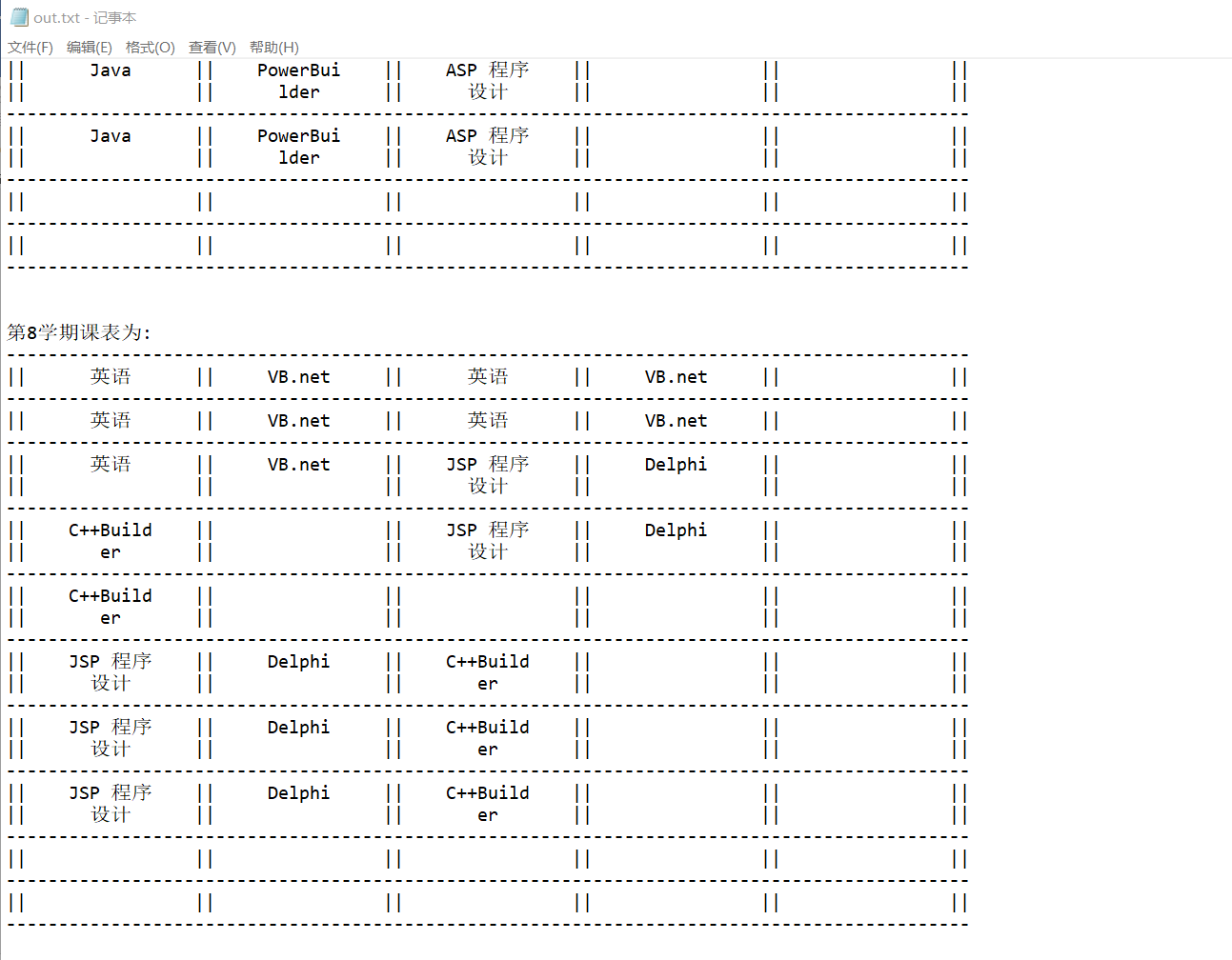












## 4.2 错误测试

### 4.2.1 每学期开课数之和不等于课程总数

**测试用例：**

7

1 1 1 1 1 1 1 1

c01 程序设计基础 5 0

c02 离散数学 6 0 c01

c03 数据结构算法 4 0 c01 c02

c04 汇编语言 5 0 c01

c05 算法设计 4 0 c03 c04

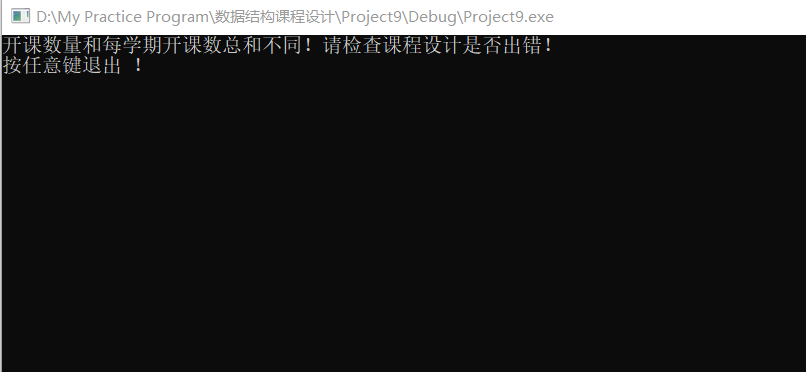
c06 计算机组成原理 6 0

c07 微机原理 4 0 c03

c08 单片机应用 3 0 c03

**预期结果：**给出错误提示，输出文件为空。

**实验结果：**



### 4.2.2 课程存在循环依赖

**测试用例：**

8

1 1 1 1 1 1 1 1

c01 程序设计基础 5 0 c02

c02 离散数学 6 0 c01v c03

c03 数据结构算法 4 0 c04

c04 汇编语言 5 0 c01 c05

c05 算法设计 4 0 c03 c06

c06 计算机组成原理 6 0 c07

c07 微机原理 4 0 c03 c08

c08 单片机应用 3 0 c01

**预期结果：**给出错误提示，输出文件为空

**实验结果：**

