项目说明文档

数据结构课程设计

——排课系统

作 者 姓 名： 邓泉

学 号： 1953871

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**

[1 分析](#_Toc8268)

[1.1 背景分析](#_Toc27237)

[1.2 功能分析](#_Toc28991)

[2 设计](#_Toc19056)

[2.1 数据结构设计](#_Toc24080)

[2.2 类结构设计](#_Toc19529)

[2.3 成员与操作设计](#_Toc12865)

[2.4 排课算法设计](#_Toc10916)

[2.5 系统设计](#_Toc19976)

[3 实现](#_Toc23528)

[3.1 文件读入数据](#_Toc12561)

[3.2 各门课程排课](#_Toc16492)

[3.2.1 各课程排课算法说明](#_Toc1100)

[3.2.2 权值函数](#_Toc18311)

[3.4 各个学期排课](#_Toc25394)

[3.5 打印课表](#_Toc27210)

[3.6 系统整体](#_Toc26661)

[4 测试](#_Toc4316)

[4.1 功能测试](#_Toc23764)

[4.1.1 合法输入下输出完整课表](#_Toc24518)

[4.2 出错测试](#_Toc14031)

[4.2.1 每学期所开课程数之和与课程总数不相等](#_Toc5315)

[4.2.2输入所开课程数过大](#_Toc2480)

[4.2.3输入所开课程数过小，小于该学期必上课程数目](#_Toc5823)

# 1 分析

## 背景分析

大学的每个专业都要进行排课。假设任何专业都有固定的学习年限，每学年含两学期，每个专业开设的课程都是确定的，而且课程在开设时间的安排必须满足先修关系。每门课程有哪些先修课程是确定的。每门课恰好占一个学期，在这种情况下，一个功能齐全的排课软件有着十分重要的意义。

## 1.2 功能分析

本项目实现的是一个简单的排课软件，需要从文件中读取学生的所有课程，在给定每周上课时间的前提下，根据各个课程的先修关系及每学期的课程安排进行排课，并将其输出在另一文件中。

同时，排课需满足以下要求：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。

项目功能要求：

1输入数据包括：个学期所开的课程数（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 数据结构设计

课程与课程之间存在先修的关系，可用图中的有向边进行表示，因此项目中设计Graph类来存储各个课程的信息，其中图中的顶点表示课程，有向边表示课程之间的先修关系。

## 2.2 类结构设计

图类Graph：存储所有的课程信息，顶点表示课程，边表示课程之间的先修关系，使用邻接矩阵表示图的各条边。具有添加顶点、添加边、查看顶点个数、查看边的个数、查看各顶点入度等功能。

边结点类Edge：存储每条边的起点和终点，及边的权值。

序列类ClassList：存储图中所有顶点的信息。

课程类Course：存储课程的信息，包括课程序号、课程名称、周学时数、指定开课学期、先修课程。

## 2.3 成员与操作设计

**边结点类（Edge）**

公有成员

//无参数构造函数

Edge() :head(0), tail(0), value(0) {};

//含参构造函数

Edge(int h, int t, int v) :head(h), tail(t), value(v) {};

bool operator<(const Edge& p) { return this->value < p.value; }

bool operator>(const Edge& p) { return this->value > p.value; }

//起点

int head;

//终点

int tail; int value;//边的权值

**序列类（ClassList）**

**私有成员**

int maxsize;//最大数组大小

int size;//当前数组大小

**public:**

ClassList() :maxsize(MAX\_SIZE), size(0) { data = new Course[maxsize + 1]; }

//构造函数

~ClassList() { delete[] data; }

//析构函数

void insert(Course& t);

//插入元素

int find(Course& t)

//查找该元素存放在数组的位置

Course\* data;//存放数据的数组

**图类（Graph）**

私有成员**:**

int maxsize;//顶点个数上限

int verticenum;//当前的顶点数

int edgenum;//当前的边数

int \*\*edge;//邻接矩阵

ClassList\* List;//存放顶点的列表

Mintree mintree;//最小生成树

bool haveMT = 0;//是否已生成最小生成树

公有成员**:**

Graph():maxsize(MAX\_SIZE),verticenum(0),edgenum(0),mintree(MAX\_SIZE)

{

edge = new int\*[maxsize];

for (int i = 0;i < maxsize; i++)

{

edge[i] = new int[maxsize];

}

List = new ClassList<Nametype>;

for (int i = 0; i < maxsize; i++) {

for (int j = 0; j < maxsize; j++) {

edge[i][j] = INT\_MAX;

}

}

}

//构造函数

~Graph()

{

delete[]List;

for (int i = 0; i < maxsize; i++) delete[] edge[i];

delete[] edge;

}

//析构函数

void AddEdge(Course h, Course t, int v);

//添加边

void AddVertice(Course node);

//添加顶点

int EdgeNum() { return edgenum; }

//返回边的个数

int VerticeNum() { return verticenum; }

//返回顶点的个数

int size() { return verticenum; }

//返回顶点的个数

ClassList\* list() { return List; }

//返回顶点列表指针

int\* Count() const { return count; }

//返回count数组指针

## 2.4 排课算法设计

排课顺序采用了拓扑排序的思想，每个学期排课时程序对所有课程进行遍历，找出所有入度为零的课程，即代表其先决条件已全部达成，可在本学期安排课程；每学期课表安排完成后，若其他课程的先决条件包含本学期的课程，则该课程的入度减一。以此方法不断排出每学期的课表，直至所有课程都安排完成。

选取每节课最佳的上课时间则采用了权值函数的思想，通过深度优先遍历找出该课程所有的排课方式，并对每种排课方式进行赋权，找出权值最大的排课方式。权值函数的计算方法大致为：a)三节课连续上的权值大于两节课连续上的权值，排一节课的情况权值极低。b)同一节课如果要上两大节课，隔天上的权值最大，连续两天上的权值较低，排出同一天上两节相同课程的情况，即赋无穷小权值。

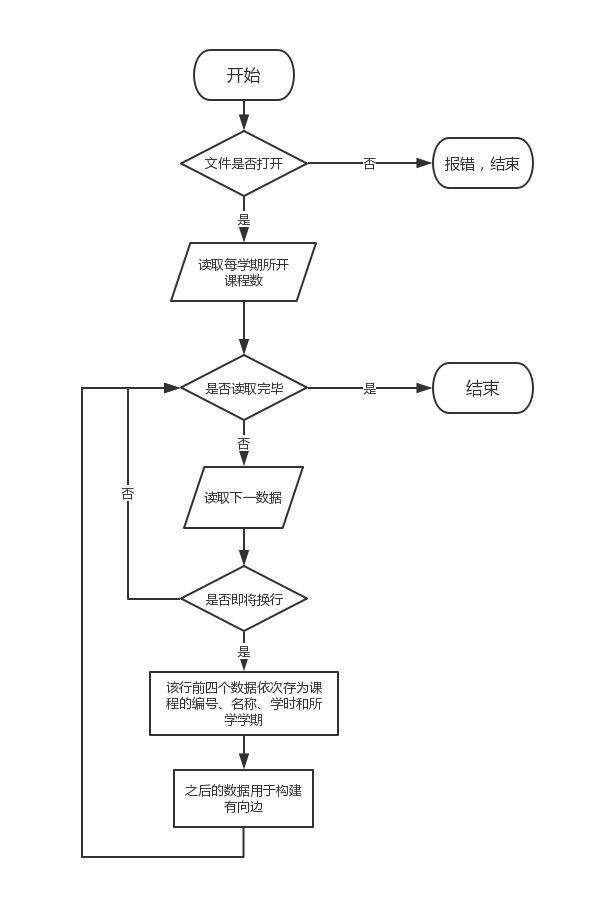
## 2.5 系统设计

系统首先调用函数GetData()，从文件读入数据，并进行相应处理构成课程的图，并判断输入数据是否有错误。接下来按顺序对每学期进行排课，具体则是按一节一节课程来排，如果找到合理的排课方式，则将该课程放入课程表Table中，其中Table是全局变量，用二维数组实现，直至排课数目等于本学期所开课程数，最后输出本学期的课表。对每一学期进行上述操作，直至所有课程均被排入课表为止。

# 3 实现

## 3.1 文件读入数据

流程图:



代码实现：

bool GetData(ifstream& infile, int& num)//从文件读入数据，并存入图中

{

if (!infile.is\_open())

{

cerr << "无法打开文件" << endl;

return false;

}

int k = 0, count = 0;

while (infile >> coursenum[k])//输入每学期上的课程数

{

count += coursenum[k];

k++;

if (infile.get() == '\n') break;

}

string data[10];

int i = 0;

while (infile >> data[i++])

{

if (infile.get() == '\n')//换行前，构建有向边

{

int period = StrToInt(data[2]), term = StrToInt(data[3]);

Course cur(data[0], data[1], period, term);

Courses.AddVertice(cur);

for (int j = 4; j < i; j++)

{

Course\* p = GetFromNum(data[j]);

Courses.AddEdge(\*p, cur);

}

num++;

i = 0;

}

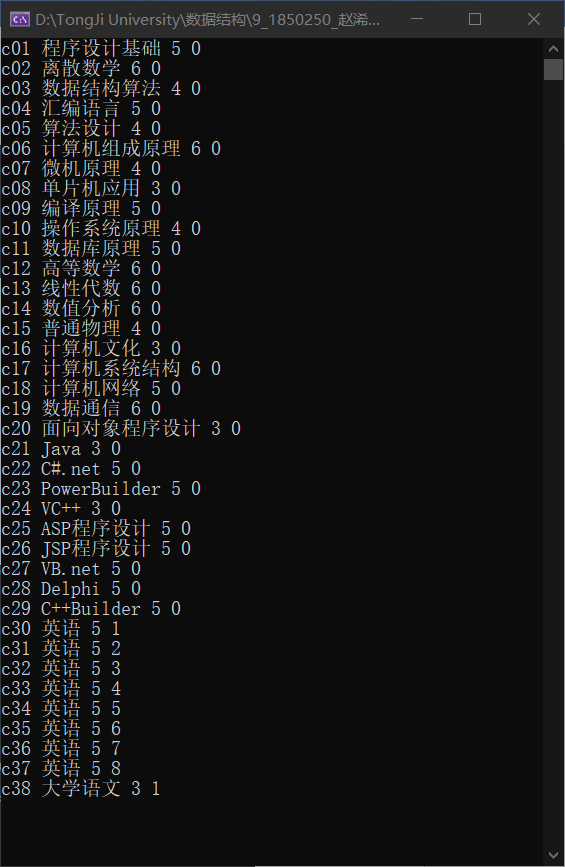
}

if (count != num) return false;

else return true;

}

程序示例：



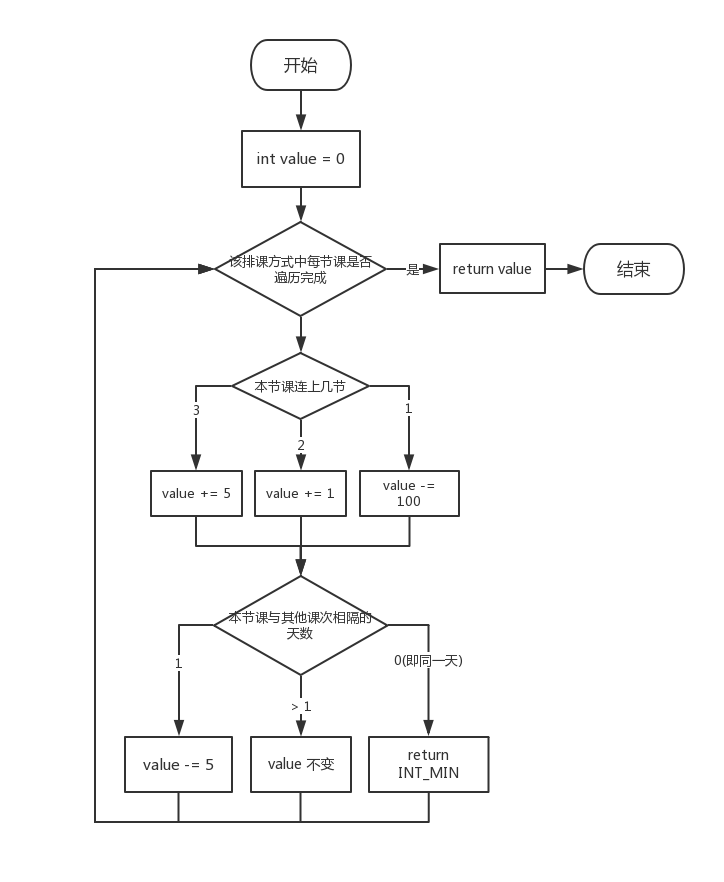
## 3.2 各门课程排课

### 3.2.1 各课程排课算法说明

先用深搜索遍历每一种可能的排课方式，将当前排课方式存于current数组中，并使用权值函数对每种方式进行赋权，权值越高代表这种排课方式越合理，如果当前的权值比最高权值还要大，则用current数组中的数据替代best数组，因此最终权值最高的排课方式即best数组就是我们所需要的。

## 3.2.2 权值函数

流程图



代码实现：

//权值函数

int Value(int n)

{

int value = 0;

for (int i = 0; i <= n; i++)

{

if (current[i][2] == 1) value -= 100;

if (current[i][0] == 0 || current[i][0] == 2) value += 1;

else value = value + 5;

for (int j = i + 1; j <= n; j++)

{

int p = current[i][1] > current[j][1] ? current[i][1] - current[j][1] : current[j][1] - current[i][1];

if (p == 0) return INT\_MIN;

if (p == 1) value -= 5;

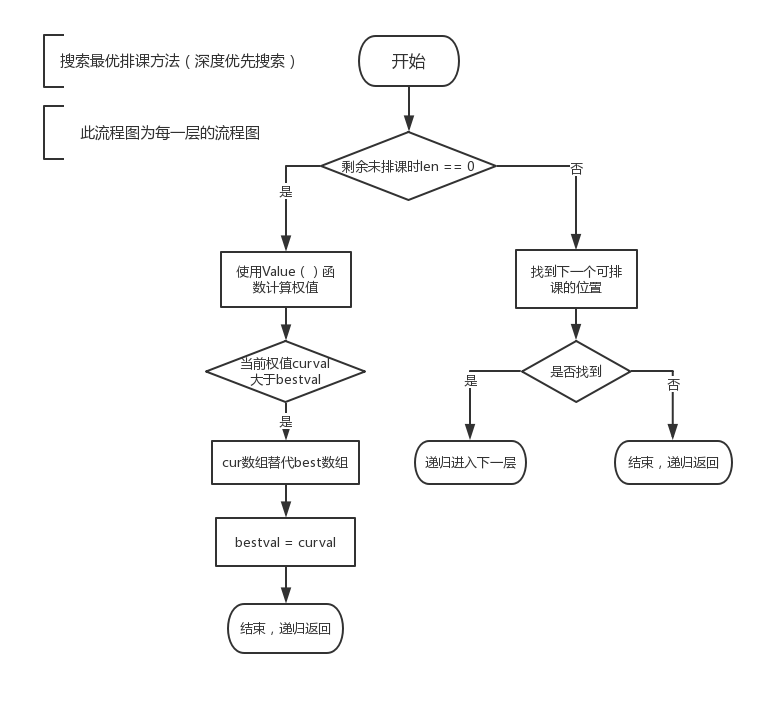
}

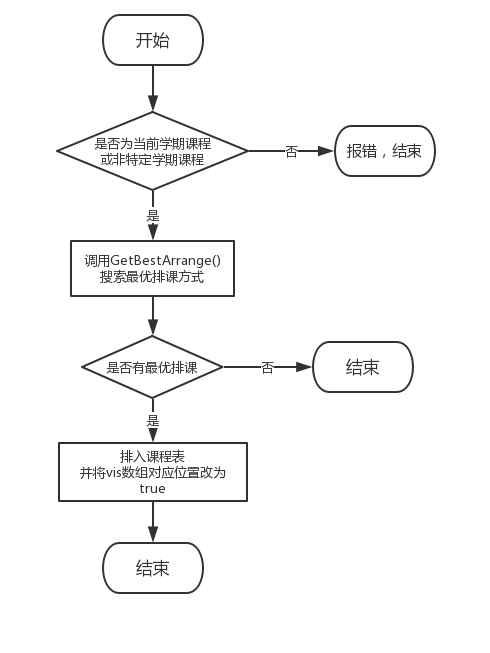
}

return value;

}

流程图：





代码实现：

void GetBestArrange(Course& course)

{

Dfs(course.period, 0);

}

void Dfs(int len, int depth)

{

if (len == 0)

{

currentval = Value(depth - 1);

if (currentval >= bestval)

{

bestval = currentval;

for (int i = 0; i < depth; i++)

{

best[i][0] = current[i][0];

best[i][1] = current[i][1];

best[i][2] = current[i][2];

}

for (int i = depth; i < 4; i++)

{

best[i][0] = -1;

best[i][1] = -1;

best[i][2] = -1;

}

}

return;

}

for (int i = 0; i < 4; i++)

{

for (int j = 0; j < 5; j++)

{

if (!ifcourse[i][j])

{

ifcourse[i][j] = 1;

int length = (i == 0 || i == 2 ? 2 : 3);

if (len < length)

{

ifcourse[i][j] = 0;

continue;

}

current[depth][0] = i;

current[depth][1] = j;

Dfs(len - length, depth + 1);

ifcourse[i][j] = 0;

}

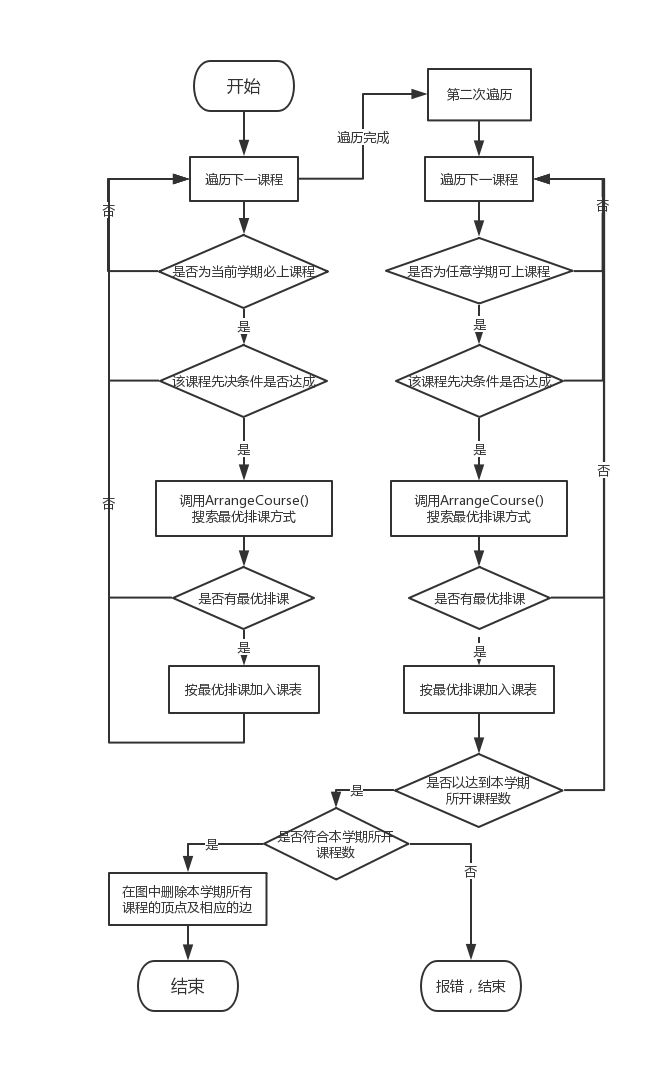
}

}

}

## 3.4 各个学期排课

流程图



代码实现：

bool ArrangeTerm(int& num, int term)

{

Course\* del = new Course[num];

int i = 0, kk = 0;

int\* p = Courses.Count(), n = Courses.size();

for (int j = 0; j < n; j++)

{

if (\*p == 0 && Courses.list()->data[kk].term == term)

{

ClearValue();

bool read = ArrangeCourse(Courses.list()->data[kk], term);

if (read)

{

del[i++] = Courses.list()->data[kk];

}

}

kk++;

p++;

}

kk = 0;

p = Courses.Count();

for (int j = 0; j < n; j++)

{

if (\*p == 0 && (i < coursenum[term - 1] && Courses.list()->data[kk].term == 0))

{

ClearValue();

bool read = ArrangeCourse(Courses.list()->data[kk], term);

if (read)

{

del[i++] = Courses.list()->data[kk];

}

}

kk++;

p++;

}

for (int j = 0; j < i; j++)

{

Courses.DeleteVertice(del[j]);

num--;

}

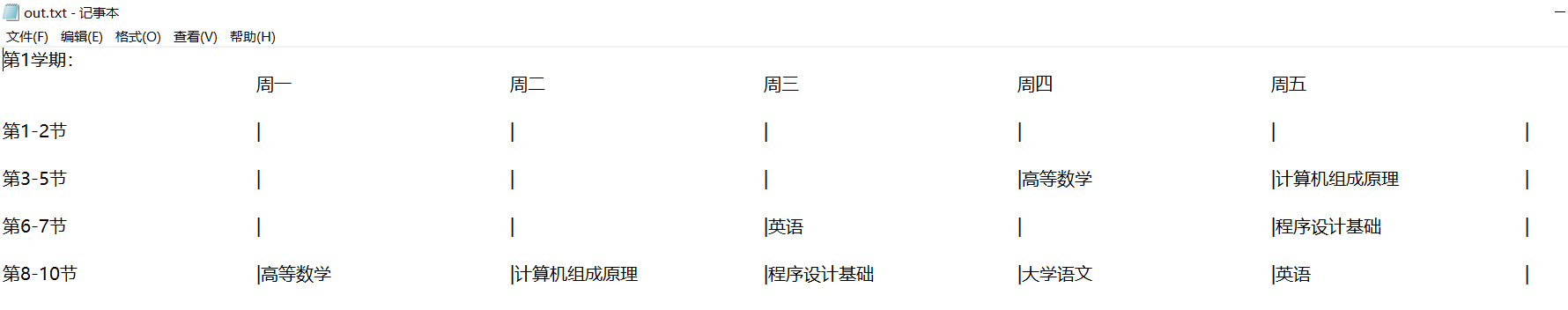
delete[]del;

if (i != coursenum[term - 1]) return false;

return true;

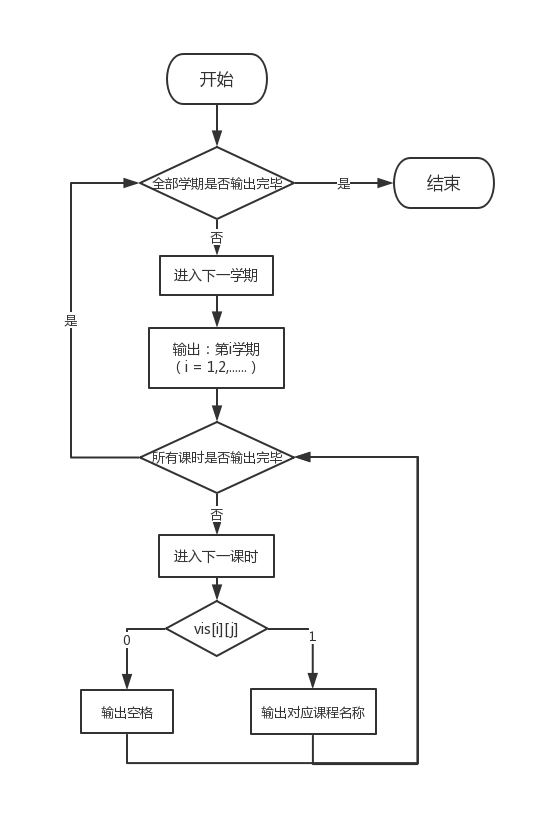
}

程序示例：



## 3.5 打印课表

流程图：



代码实现：

void PrintClass(ofstream& out, int term)

{

out << "第" << term << "学期：\n";

out << setw(32) << setiosflags(ios::left) << setfill(' ') << " " << "\t"

<< setw(32) << setiosflags(ios::left) << setfill(' ') << "周一" << "\t"

<< setw(32) << setiosflags(ios::left) << setfill(' ') << "周二" << "\t"

<< setw(32) << setiosflags(ios::left) << setfill(' ') << "周三" << "\t"

<< setw(32) << setiosflags(ios::left) << setfill(' ') << "周四" << "\t"

<< setw(32) << setiosflags(ios::left) << setfill(' ') << "周五" << "\t"

<< endl;

out << endl;

for (int i = 0; i < 4; i++)

{

switch (i)

{

case 0:out << setw(32) << setiosflags(ios::left) << setfill(' ') << "第1-2节" << "\t" << "|"; break;

case 1:out << setw(32) << setiosflags(ios::left) << setfill(' ') << "第3-5节" << "\t" << "|"; break;

case 2:out << setw(32) << setiosflags(ios::left) << setfill(' ') << "第6-7节" << "\t" << "|"; break;

case 3:out << setw(32) << setiosflags(ios::left) << setfill(' ') << "第8-10节" << "\t" << "|"; break;

}

for (int j = 0; j < 5; j++)

{

if (ifcourse[i][j]) out << setw(32) << setiosflags(ios::left) << setfill(' ') << Table[i][j].c\_str() << "\t" << "|";

else out << setw(32) << setiosflags(ios::left) << setfill(' ') << " " << "\t" << "|";

}

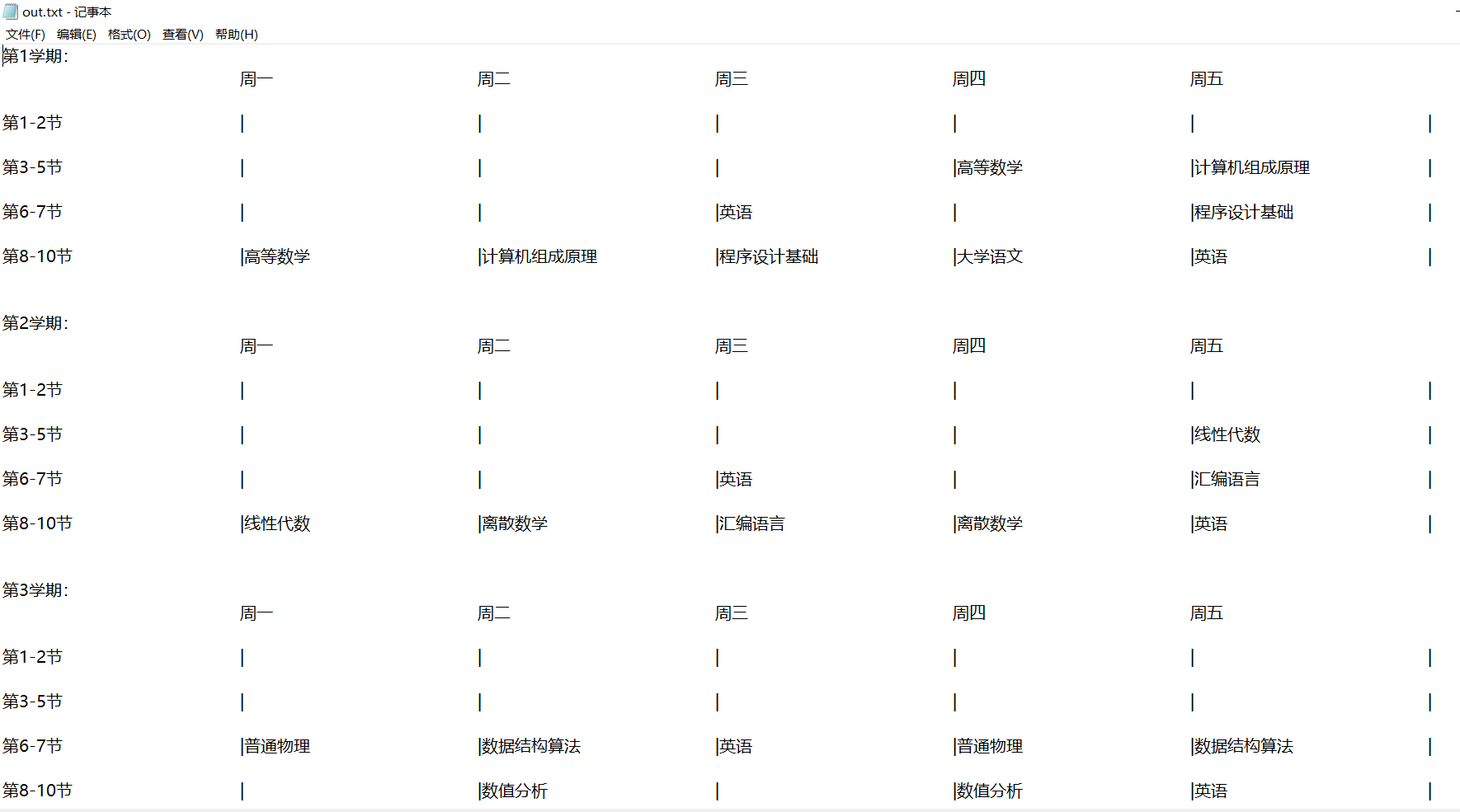
out << "\n\n";

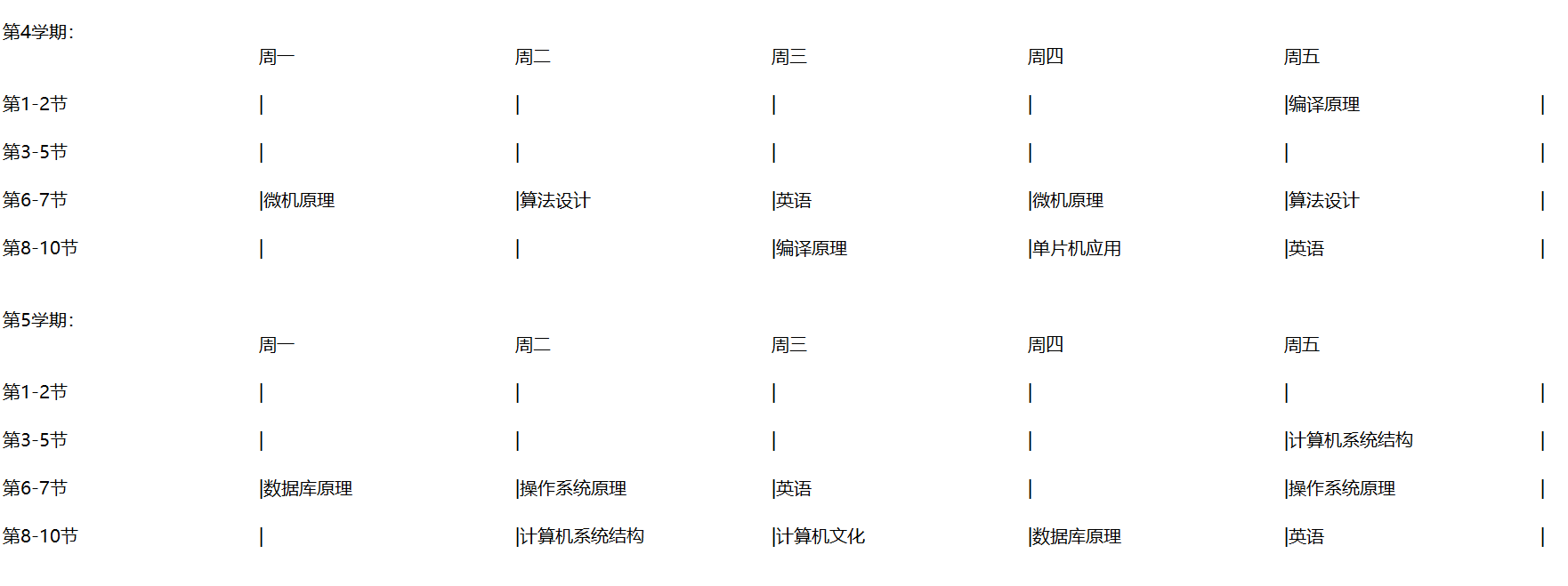
}

out << "\n";

}

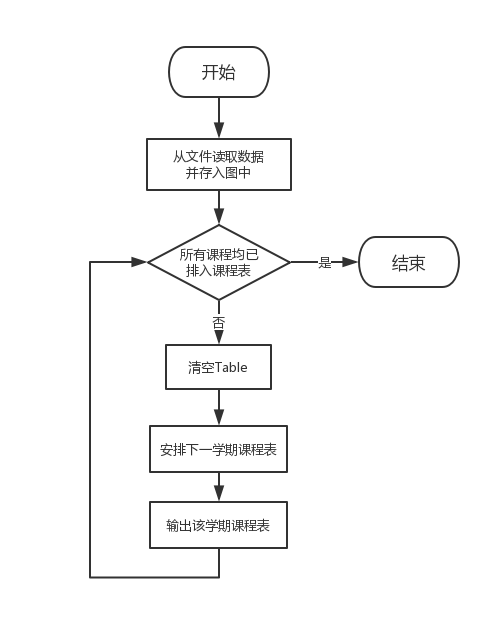
程序示例：





## 3.6 系统整体

流程图：



代码实现：

int main()

{

ifstream infile("9\_1953871\_邓泉.txt");

ofstream outfile("out.txt");

int classnumber = 0;//初始化课程数

bool read = GetData(infile, classnumber);//文件读入

if (!read)//error

{

outfile << "开设课程数与课程总数不相等" << endl;

cout << "开设课程数与课程总数不相等" << endl;

system("pause");

return 0;

}

int term = 1;//学期

while (classnumber > 0)

{

TableClear();

read = ArrangeTerm(classnumber, term);//安排当前学期课表

if (!read)//error

{

outfile << "无法匹配，程序中止" << endl;

cout << "无法匹配，程序中止" << endl;

system("pause");

return 0;

}

PrintClass(outfile, term);//打印课表

//PrintTable(term);

term++;//下一学期

}

cout << "课表已生成" << endl;

system("pause");

return 0;

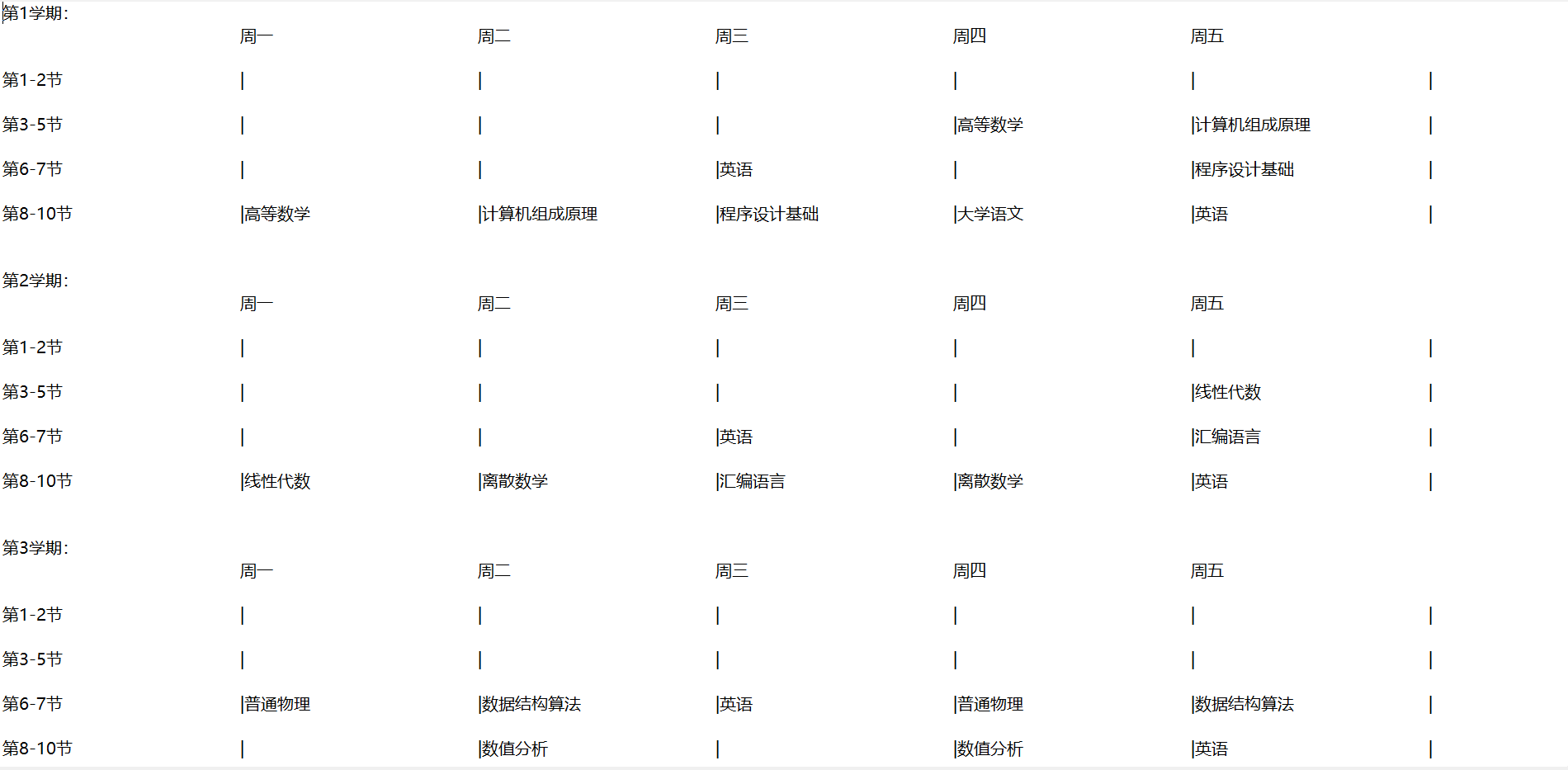
}

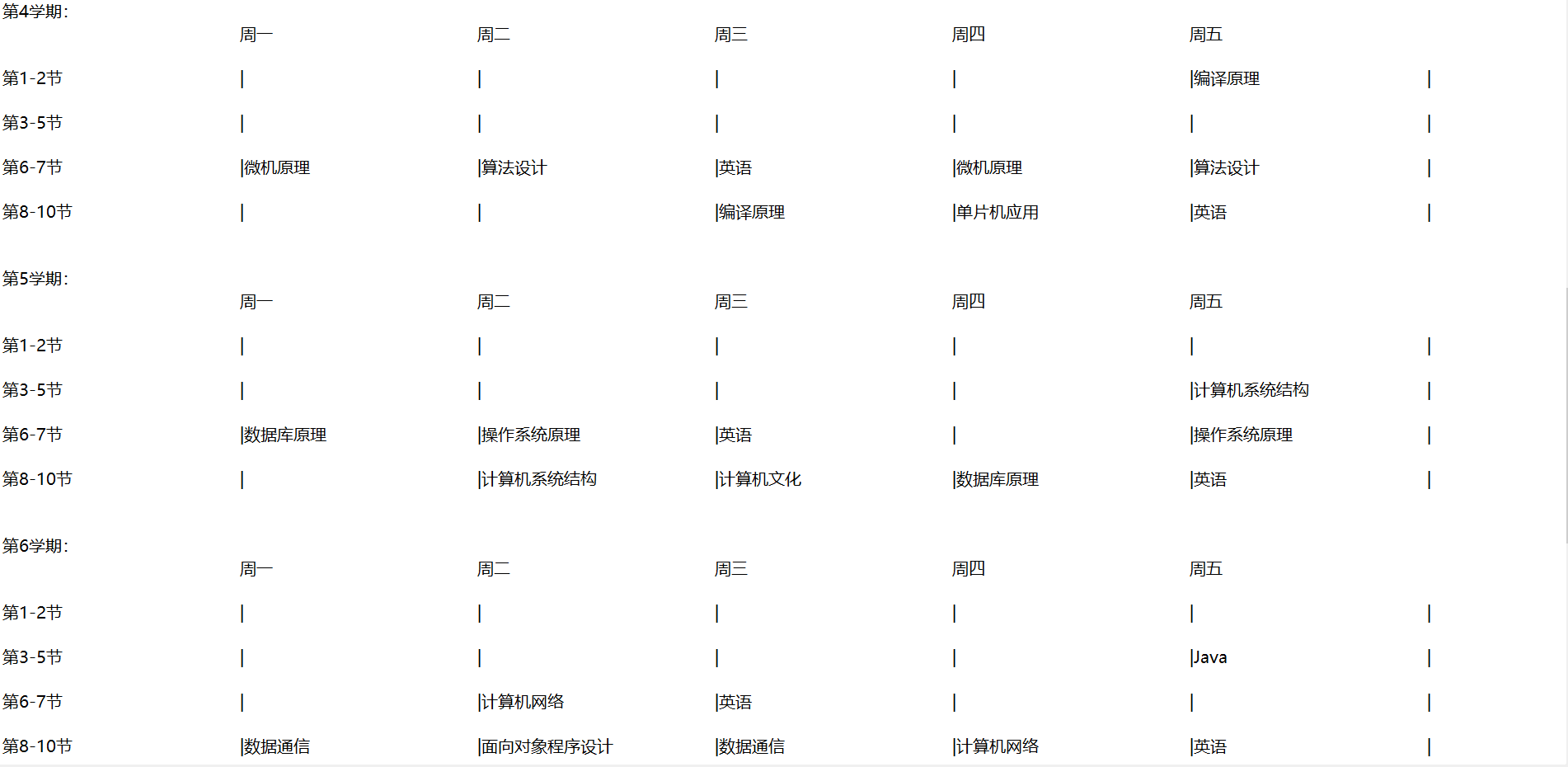
# 4 测试

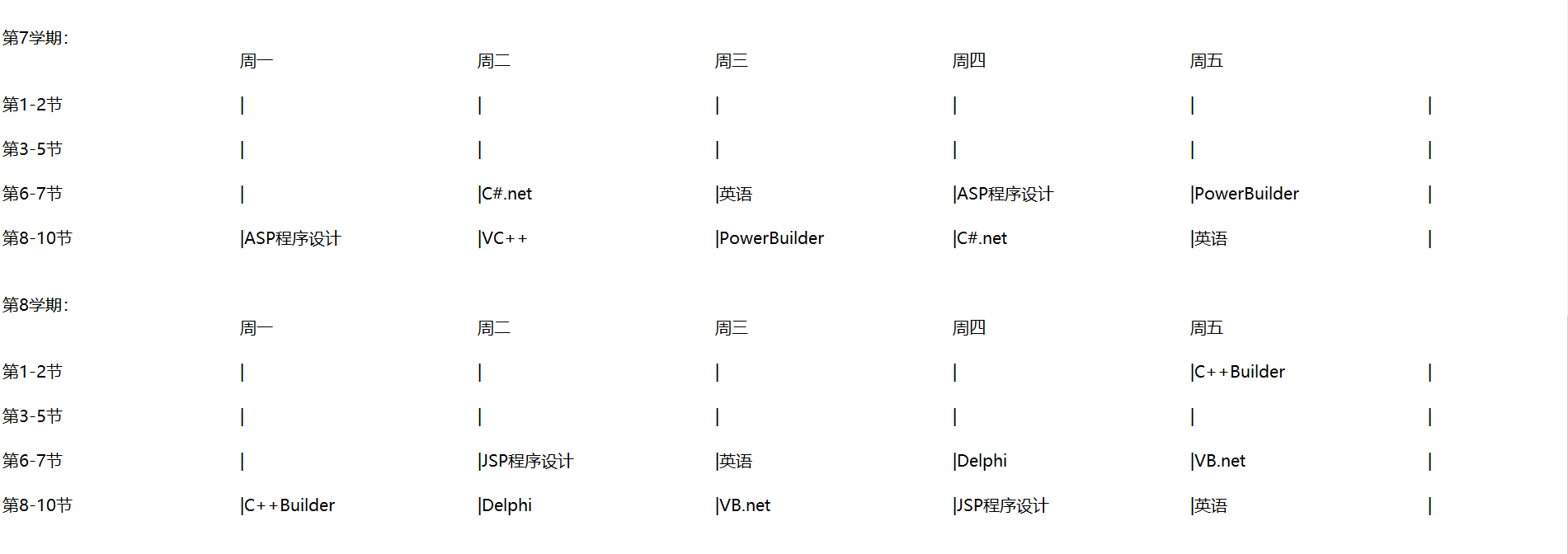
## 4.1 功能测试

## 4.1.1 合法输入下输出完整课表

实验结果：





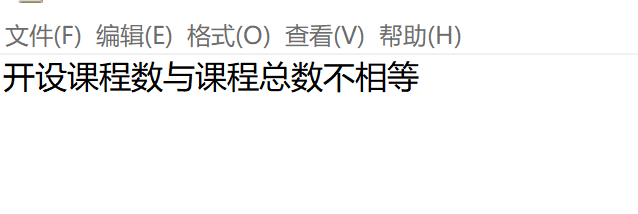


## 4.2 出错测试

### 4.2.1 每学期所开课程数之和与课程总数不相等

**预期结果：**程序报错并给予提示，程序正常运行

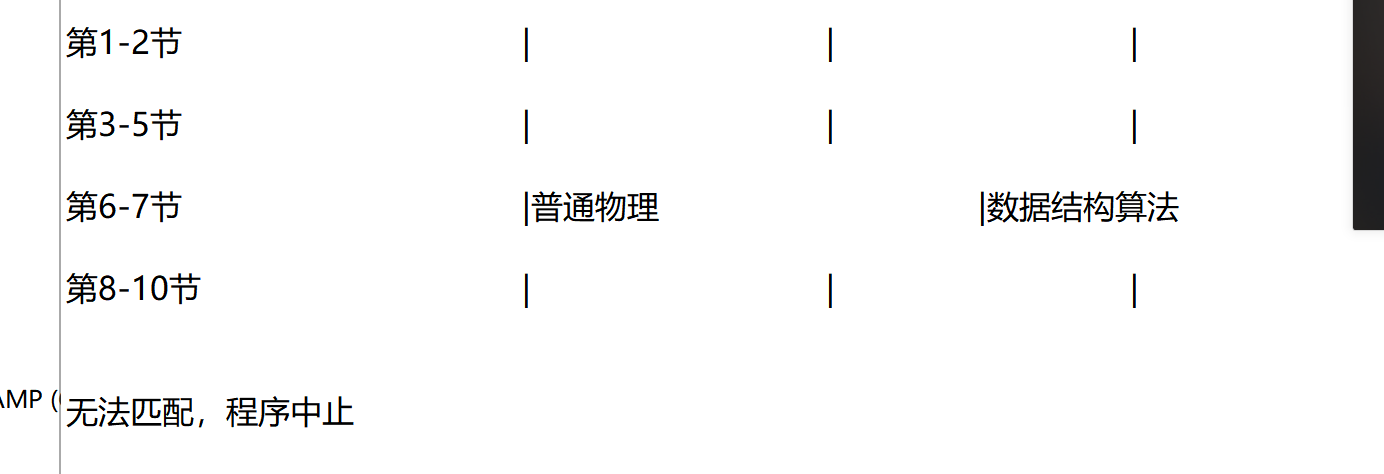
**实验结果：**

****

### 4.2.2输入所开课程数过大

**预期结果：**程序报错并给予提示，程序正常运行

**实验结果：**



### 4.2.3输入所开课程数过小，小于该学期必上课程数目

**测试用例**：每学期所开课程数1 6 6 5 5 5 5 5

**预期结果：**程序报错并给予提示，程序正常运行

**实验结果：**

