****

项目说明文档

**数据结构课程设计**

**——排课软件**

**培养单位：软件学院**

**本 科 生：蓝 笙 聆**

**学 号：1951096**

**指导老师：张 颖**

二○二○年十二月

目录

[第1章 功能分析 1](#_Toc59750202)

[1.1 背景分析 1](#_Toc59750203)

[1.2 功能分析 1](#_Toc59750204)

[第2章 设计 2](#_Toc59750205)

[2.1 数据结构设计 2](#_Toc59750206)

[2.2 类结构设计 2](#_Toc59750207)

[2.3 成员与操作设计 2](#_Toc59750208)

[2.4 系统设计 4](#_Toc59750209)

[第3章 实现 5](#_Toc59750210)

[3.1 从文件读取数据的实现 5](#_Toc59750211)

[3.1.1 从文件读取数据功能流程图 5](#_Toc59750212)

[3.1.2 从文件读取数据的核心代码 6](#_Toc59750213)

[3.2 拓扑排序的实现 7](#_Toc59750214)

[3.2.1 拓扑排序功能流程图 7](#_Toc59750215)

[3.2.2 拓扑排序核心代码 8](#_Toc59750216)

[3.3 排课表功能的实现 9](#_Toc59750217)

[3.3.1 排课表功能流程图 9](#_Toc59750218)

[3.3.2 排课表功能核心代码 10](#_Toc59750219)

[3.4 总体系统的实现 13](#_Toc59750220)

[3.4.1 总体系统流程图 13](#_Toc59750221)

[3.4.2 总体系统代码实现 13](#_Toc59750222)

[3.4.3 总体功能截屏示例 14](#_Toc59750223)

[第4章 测试 15](#_Toc59750224)

[4.1 通常功能测试 15](#_Toc59750225)

[4.2 出错测试 15](#_Toc59750226)

[4.2.1 输入文件路径不存在 15](#_Toc59750227)

# 第1章 功能分析

## 1.1 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，假定每天上午与下午各有5节课。是在这样的前提下设计一个教学计划编制程序。

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，设weekday表示当前安排上课的工作日期，下一次排课的工作日是：weekday=（weekday+2-5）？（weekday+2-5）：（weekday+2）；

## 1.2 功能分析

对项目需求进行分析，发现部分课程具有先修课程，需要在前一个或几个学期中修读完所有的先修课程才能修读该课程。也有部分课程是指明一定要在某一学期内开课。要从文本文件中录入课程，又要将排课结果输出到文本文件当中。

# 第2章 设计

## 2.1 数据结构设计

如上所述，根据先修课程之间的关系，考虑使用有向图存储，并使用拓扑排序生成课程修读顺序。

## 2.2 类结构设计

经典的图一般包括三个抽象数据类型（ADT）——图类（Graph <T>）、点类（Vertex <T>）与边类（Edge <T>），而三个类之间的耦合关系可以采用嵌套、继承等多种关系。为方便处理，本系统采用模板struct结构体描述树点类（Vertex <T>）与边类（Edge <T>），这样使得图类（Graph <T>）可以访问链表结点。而每一个课程可以封装为一个结构体（Course），更加方便进行操作。

## 2.3 成员与操作设计

**点结构体（Vertex）**

template <class T>

struct vertex {

    T val;

    vertex(const T& val) : val(val){};

    vertex() : val(){};

    bool operator==(const vertex& v2) const { return (this->val == v2.val); };

    bool operator==(const T& v2) const { return (this->val == v2); };

    bool operator!=(const vertex& v2) { return !(\*this == v2); };

    bool operator!=(const T& v2) { return !(\*this == v2); };

    bool operator<(const vertex& v2) const { return (this->val < v2.val); };

    bool operator>(const vertex& v2) const { return (this->val > v2.val); };

    void operator=(const vertex& v2) { this->val = v2.val; };

};

**边结构体（Edge）**

template <class T, class E>

struct edge {

    E val;

    vertex<T> v1;

    vertex<T> v2;

    edge(const vertex<T> v1, const vertex<T> v2, const E val)

        : v1(v1), v2(v2), val(val){};

    edge() : v1(), v2(), val(){};

    bool operator==(const edge& e2) {

        return (this->v1 == e2.v1 && this->v2 == e2.v2);

    };

    bool operator!=(const edge& e2) { return !(\*this == e2); };

    bool operator<(const edge& e2) const { return (this->val < e2.val); };

    bool operator>(const edge& e2) const { return (this->val > e2.val); };

};

**图类（Graph）**

template <class T, class E>

class Graph {

   public:

    //声明友元函数

    friend vector<vector<vertex<Course>>> Topological(Graph<Course, int> g);

    void InsertVertex(vertex<T>& v);

    void InsertVertex(const T val);

    void DelVertex(vertex<T>& v);

    void DelVertex(const T val);

    void InsertEdge(vertex<T> v1, vertex<T> v2, E cost);

    void InsertEdge(T v1, T v2, E cost);

    void InsertEdge(edge<T, E> edge);

    void RemoveEdge(vertex<T> v1, vertex<T> v2);

    void RemoveEdge(T v1, T v2);

    void RemoveEdge(edge<T, E> edge);

    void PrintGraph(ostream& os);

    int NumVertices() const { return \_vertices.size(); };

    int NumEdges() const { return \_edges.size(); }

    bool InGraph(T v);

    bool InGraph(vertex<T> v);

    bool InGraph(T v1, T v2);

    bool InGraph(vertex<T> v1, vertex<T> v2);

    bool InGraph(edge<T, E> edge);

   private:

    vector<vertex<T>> \_vertices;

    map<pair<vertex<T>, vertex<T>>, edge<T, E>> \_edges;

};

**课程结构体（Course）**

struct Course {

    string id;           //课程编号

    string name;         //课程名字

    int lesson;          //课时

    int term;            //开课学期（0为不限

    vector<string> pre;  //先修课程

    void input(string s);

    friend std::ostream& operator<<(std::ostream& output, const Course& D);

    bool operator==(const Course& v2) const { return (this->id == v2.id); };

    bool operator==(const string& v2) const { return (this->id == v2); };

    bool operator!=(const Course& v2) { return !(\*this == v2); };

    bool operator!=(const string& v2) { return !(\*this == v2); };

    bool operator<(const Course& v2) const { return (this->id < v2.id); };

};

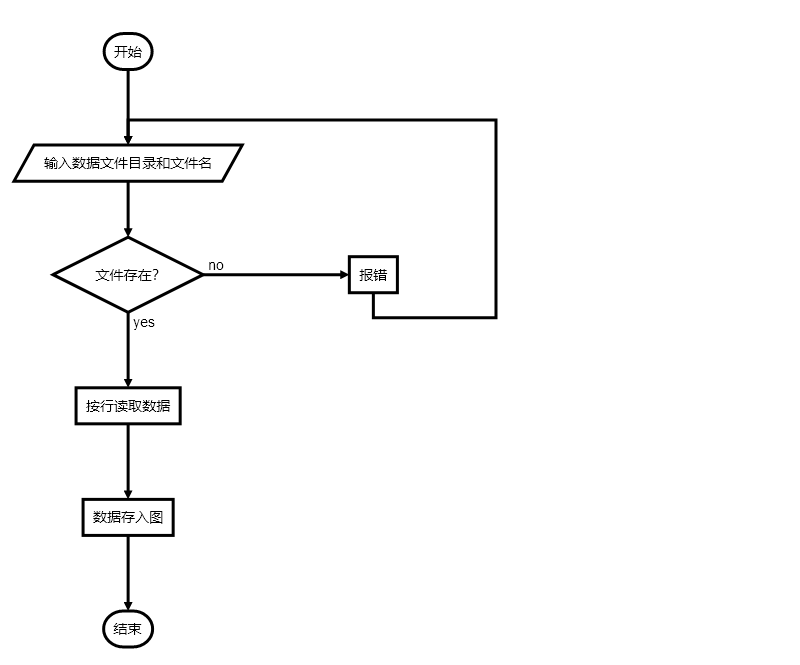
## 2.4 系统设计

系统首先调用opening ()函数实现对屏幕的初始化，输入课程信息所在的文件路径和文件名读取数据并录入，录入的同时开始构造有向图。录入无误后有向图构造完成，输入保存课程表的路径，通过拓扑排序和学时得出每个学期需修读的课程，再使用枚举方法将课程塞入课程表当中。

# 第3章 实现

## 3.1 从文件读取数据的实现

### 3.1.1 从文件读取数据功能流程图



### 3.1.2 从文件读取数据功能核心代码

void reading(Graph<Course, int>& g) {

    cout << "Enter the location of the timetable file." << endl;

    string temp;

    ifstream in;

    while (true) {

        cin >> temp;

        in = ifstream(temp);

        if (in) break;                                       // 有该文件

        cout << "No such file, please try again. " << endl;  // 无该文件

    }

    string line;

    while (getline(in, line)) {  // line中不包括每行的换行符

        map<string, Course>

            course;  //使用map暂存每门课程id与课程的对应关系以便画图

        if (line.size() == 0) continue;

        Course c;

        c.input(line);

        cout << c;

        course[c.id] = c;

        g.InsertVertex(c);

        if (c.pre.size() != 0)

            for (auto&& i : c.pre) {

                auto it = course.find(i);

                if (it != course.end()) g.InsertEdge(course[i], c, 0);

            }

    }

}

void Course::input(string s) {

    vector<string> text;

    SplitString(s, text, " ");

    if (text.size() < 4) throw "Error: invalue input detected. ";

    if (!isNum(text[2]) || !isNum(text[3]))

        throw "Error: invalue input detected. ";

    this->id = text[0];

    this->name = text[1];

    this->lesson = (int)str2num(text[2]);

    this->term = (int)str2num(text[3]);

    for (int i = 4; i < text.size(); i++) pre.push\_back(text[i]);

}

## 3.2 拓扑排序的实现

### 3.2.1 拓扑排序功能流程图

### 3.2.2 拓扑排序功能核心代码

vector<vector<vertex<Course>>> Topological(Graph<Course, int> g) {

    Graph<Course, int> a = g;  //复制图

    vector<vector<vertex<Course>>>

        v;  //用二维数组存课程表，二维分别代表学期，课程

    int t = 1;  //当前学期

    while (a.NumVertices() != 0) {

        vector<vertex<Course>> temp;  //存储当前出度为0且限定该学期开课的课程

        for (auto&& j : a.\_vertices) {

            if (j.val.term != 0 && j.val.term != t)

                continue;  //如果该课程是限定学期课程且不是该学期的课程则忽略

            bool bflag = false;

            for (auto&& i : a.\_vertices) {

                if (i == j) continue;

                auto it = a.\_edges.find({i, j});

                if (it == a.\_edges.end()) continue;  //如果不存在该边

                bflag = true;  //如果存在该边，则j点入度不为0，忽略

                break;

            }

            if (bflag) continue;

            temp.push\_back(j);  // j点入度为0，入队

        }

        int time = 0;

        vector<vertex<Course>>

            ntemp;  //存储在不超学时的前提下当前出度为0且限定该学期开课的课程

        for (auto&& i : temp) {

            if (i.val.term == 0) {  //不是限定课程

                time += i.val.lesson;

                if (time > 20) continue;  //学时超了就拖到下个学期再学

                ntemp.push\_back(i);

                a.DelVertex(i);

            } else {  //是限定课程，就算学时超了也得学

                ntemp.push\_back(i);

                a.DelVertex(i);

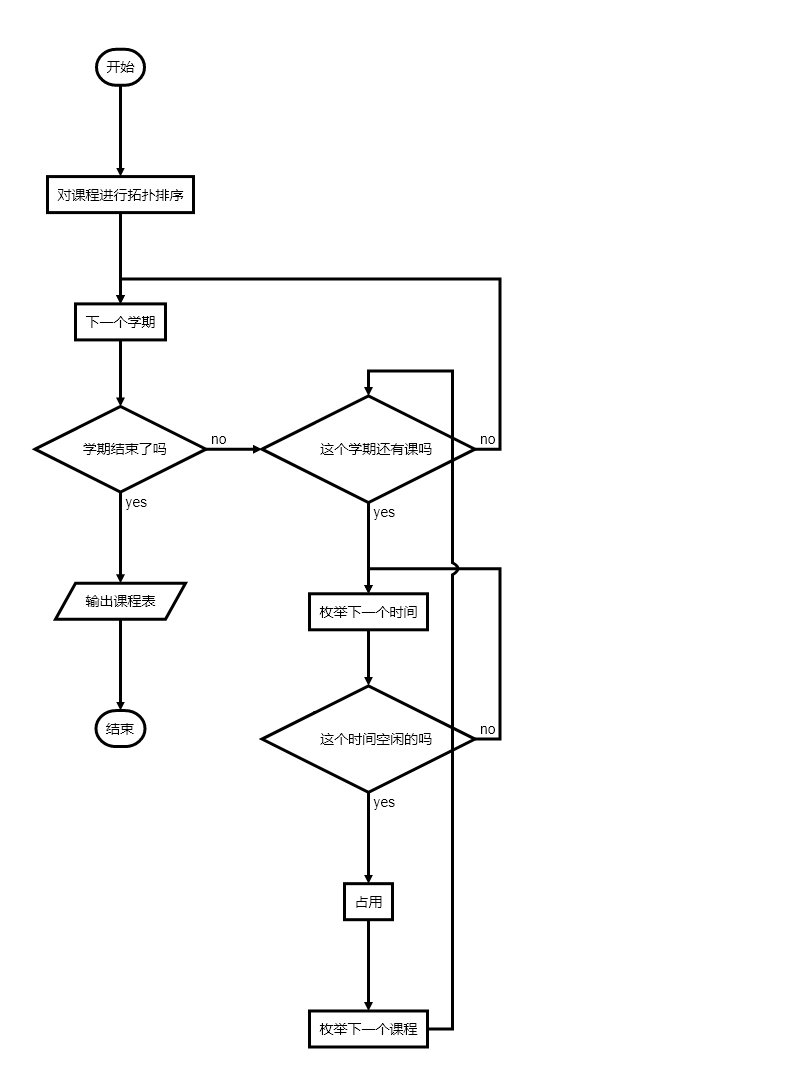
            }

        }

        v.push\_back(ntemp);  //将该学期课程推入

        t++;                 //学期数++

    }

    return v;

}

## 3.3 排课表功能的实现

### 3.3.1 排课表功能流程图

### 3.3.2 排课表功能核心代码

void scheduling(Graph<Course, int>& g) {

    cout << "Enter the path and name to save the timetable." << endl;

    string temp;

    cin >> temp;

    ofstream out(temp);

    auto v = Topological(g);

    //初始化三维数组作为课程表，其中三维分别表示学期、星期、第几节课

    vector<vector<vector<string>>> schedule(

        v.size(), vector<vector<string>>(5, vector<string>(4, "无")));

    for (int i = 0; i < v.size(); i++) {  //对每个学期

        for (auto&& k : v[i]) {           //对每个学期的每门课程

            for (int j = 0; j < 2; j++) {  //对所有的课时枚举合理的课时安排

                if (k.val.lesson == 6) {

                    if (schedule[i][j][0] == "无" &&

                        schedule[i][j + 2][2] == "无") {

                        schedule[i][j][0] = k.val.name;

                        schedule[i][j + 2][2] = k.val.name;

                        break;

                    } else if (schedule[i][j][2] == "无" &&

                               schedule[i][j + 3][0] == "无") {

                        schedule[i][j][2] = k.val.name;

                        schedule[i][j + 3][0] = k.val.name;

                        break;

                    }

                } else if (k.val.lesson == 5) {

                    if (schedule[i][j][0] == "无" &&

                        schedule[i][j + 2][3] == "无") {

                        schedule[i][j][0] = k.val.name;

                        schedule[i][j + 2][3] = k.val.name;

                        break;

                    } else if (schedule[i][j][1] == "无" &&

                               schedule[i][j + 2][2] == "无") {

                        schedule[i][j][1] = k.val.name;

                        schedule[i][j + 2][2] = k.val.name;

                        break;

                    } else if (schedule[i][j][2] == "无" &&

                               schedule[i][j + 3][1] == "无") {

                        schedule[i][j][2] = k.val.name;

                        schedule[i][j + 3][1] = k.val.name;

                        break;

                    } else if (schedule[i][j][3] == "无" &&

                               schedule[i][j + 3][0] == "无") {

                        schedule[i][j][3] = k.val.name;

                        schedule[i][j + 3][0] = k.val.name;

                        break;

                    }

                } else if (k.val.lesson == 4) {

                    if (schedule[i][j][1] == "无" &&

                        schedule[i][j + 2][3] == "无") {

                        schedule[i][j][1] = k.val.name;

                        schedule[i][j + 2][3] = k.val.name;

                        break;

                    } else if (schedule[i][j][3] == "无" &&

                               schedule[i][j + 3][1] == "无") {

                        schedule[i][j][3] = k.val.name;

                        schedule[i][j + 3][1] = k.val.name;

                        break;

                    }

                } else if (k.val.lesson == 3) {

                    if (schedule[i][j][0] == "无") {

                        schedule[i][j][0] = k.val.name;

                        break;

                    } else if (schedule[i][j][2] == "无") {

                        schedule[i][j][2] = k.val.name;

                        break;

                    }

                } else if (k.val.lesson == 2) {

                    if (schedule[i][j][1] == "无") {

                        schedule[i][j][1] = k.val.name;

                        break;

                    } else if (schedule[i][j][3] == "无") {

                        schedule[i][j][3] = k.val.name;

                        break;

                    }

                }

            }

        }

    }

    for (int i = 0; i < schedule.size(); i++) {

        cout << endl

             << "第" << i + 1 << "学期课表：" << endl

             << "星期一              星期二              星期三              "

                "星期四              星期五"

             << endl;

        out << endl

            << "第" << i + 1 << "学期课表：" << endl

            << "星期一              星期二              星期三              "

               "星期四              星期五"

            << endl;

        for (int k = 0; k < 4; k++) {

            if (k == 0 || k == 2)

                for (int num = 0; num < 3; num++) {

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

                        cout << schedule[i][j][k], out << schedule[i][j][k];

                        for (int u = schedule[i][j][k].size() - 1; u < 19; u++)

                            cout << ' ', out << ' ';

                    }

                    cout << endl, out << endl;

                }

            else

                for (int num = 0; num < 2; num++) {

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

                        cout << schedule[i][j][k], out << schedule[i][j][k];

                        for (int u = schedule[i][j][k].size() - 1; u < 19; u++)

                            cout << ' ', out << ' ';

                    }

                    cout << endl, out << endl;

                }

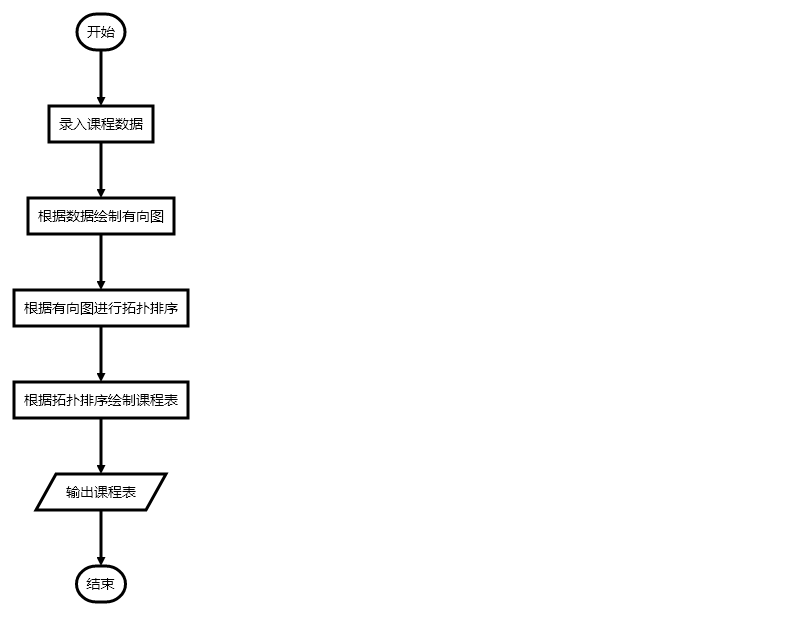
        }

    }

}

## 3.4 总体系统的实现

### 3.4.1 总体系统流程图



### 3.4.2 总体系统代码实现

void opening() {

    cout << "Course Scheduling Software" << endl;

    return;

}

int main() {

    opening();

    Graph<Course, int> g;

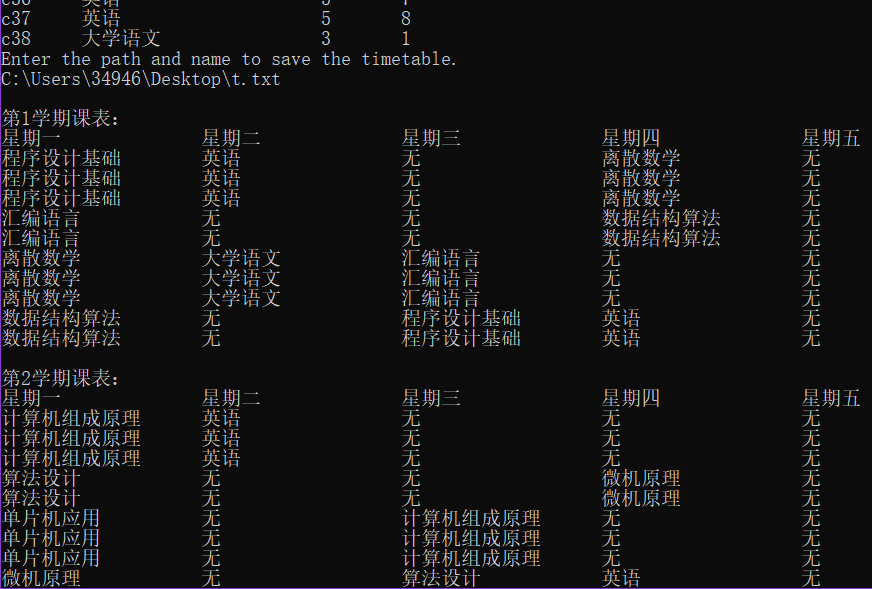
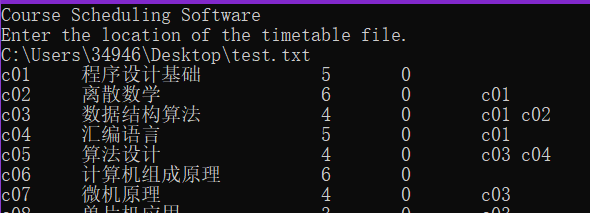
    reading(g);

    scheduling(g);

    return 0;

}

### 3.4.3 总体功能截屏示例



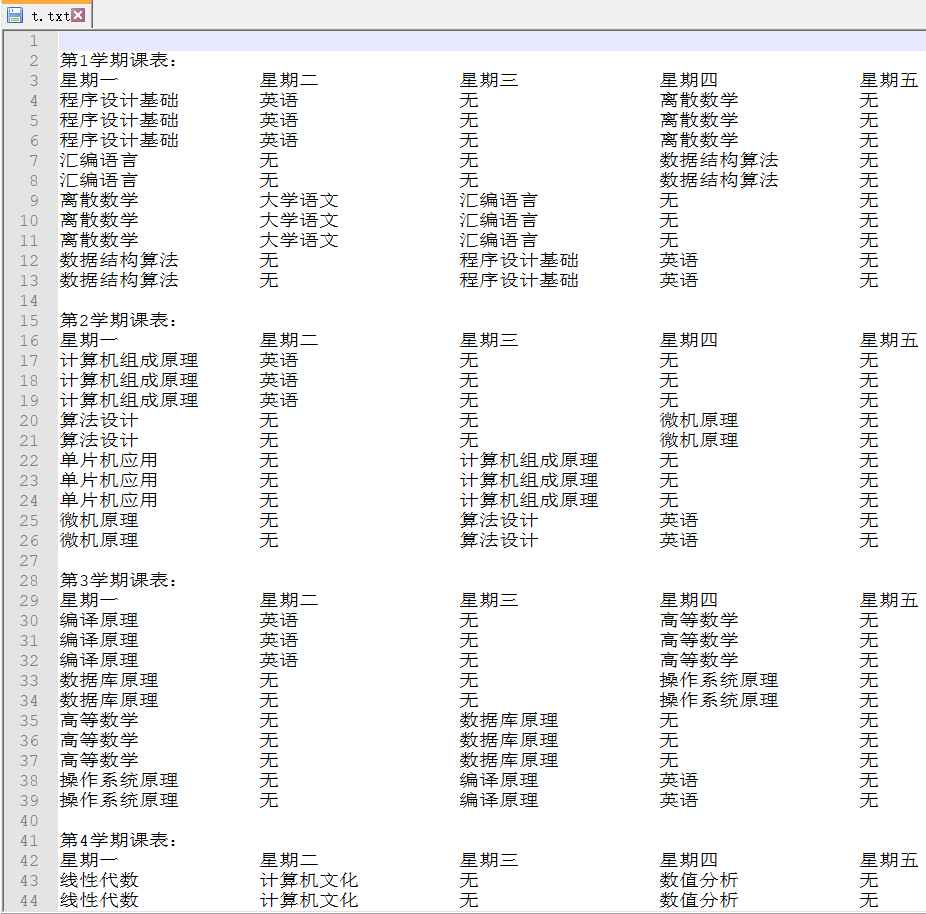
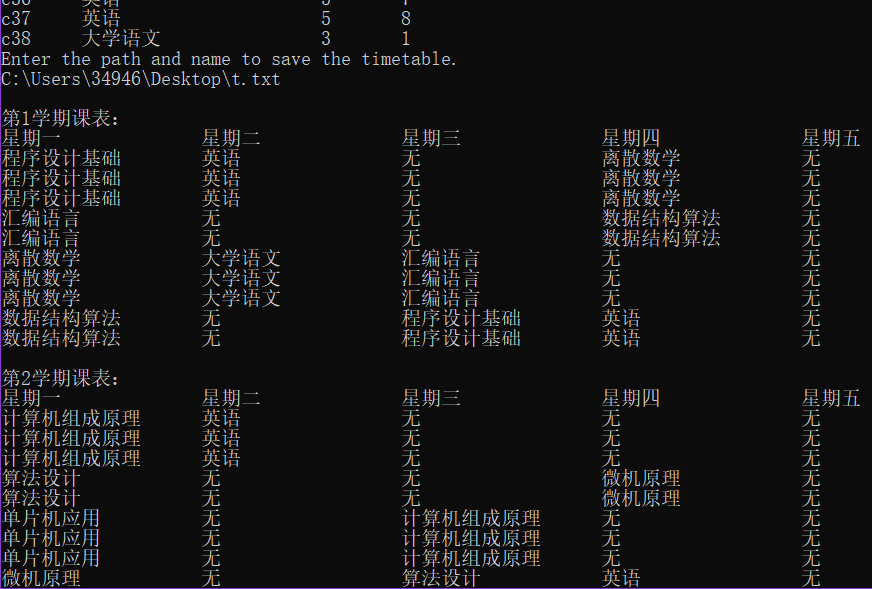
# 第4章 测试

## 4.1 通常功能测试

**测试用例**：

test.txt，文件使用GB3212简体中文编码

**实验结果**



## 4.2 出错测试

### 4.2.1 输入文件路径不存在

**测试用例：**输入不存在文件路径

**预期结果：**抛出异常，程序要求重新输入，不崩溃。

**实验结果：**

