

**class Procedure set up count report tell**

**Topic : Class Schedule Evaluation and Analysis Assistant**

**Course Title: Data Structure Course Design**

**professional class: School cross 1 601**

**student ID: U201612696**

**Name: Chen Haorui**

**Instructor: Zhouquan**

**Date of report: 2018.6.4**

**School of Computer Science and Technology**

## mission statement

**design content**

Design an effective logical data structure and storage structure to represent schedule information, teacher information, course information, class information, student information, classroom information, etc. Classroom is composed of time, classroom, course name, class, teacher name, class period and class ID, etc. information. According to the classroom and its complex correlation, evaluate the rationality of teacher arrangement, curriculum arrangement, class and student course learning rationality, classroom arrangement rationality and utilization rate, energy efficiency, etc. of class arrangement schedule according to reasonable logic and criteria. This design only evaluates the existing course arrangement, and does not solve the optimal course arrangement plan.

**Design requirements**

(1) Interactive operation interface (not necessarily a graphical interface);

(2) Addition, deletion, modification, search and retrieval of teachers, classrooms, classes, courses, etc.;

(3) Adding, deleting, modifying, searching and retrieving classes, including retrieving classes by teacher, retrieving classes by class, retrieving classes by classroom, course, time, etc.;

(4) Search and retrieval of vacant classrooms, classroom utilization analysis, and energy efficiency analysis;

(5) Teachers undertake classroom rationality analysis;

(6) Reasonable analysis of class participation in class;

(7) Data file organization and storage of main data objects.

**references**

[1]Yan Weimin, Wu Weimin. Data Structure (C language version). Beijing: Tsinghua University Press, 1997

[2]Yan Weimin, Wu Weimin, Mining. Data Structure Problem Collection (C Language Version). Beijing: Tsinghua University Press, 1999

[3]Wang Xiaodong. Computer Algorithm Design and Analysis. Beijing : Electronics Industry Press , 2007

[4]Wang Xiuhuan . Research on Optimizing Course Scheduling System in Colleges and Universities Based on Graph Theory . Master Thesis of Chongqing University , 20 12

[5]Hong Wen, Zhu Guangbin. Course scheduling problem and its mathematical model. Journal of Anhui Electric Power Workers University, 2002, 7(3): 74-77

[6] Tao Tao, Xie Weixing. Class schedule model and application of class scheduling algorithm. Computer System Application, 2011, 20(2): 198-201

[7] Xu Xiao . Research on Course Scheduling Model Based on Ontology Mapping and Rule Reasoning. Software Guide, 2009 , 8 ( 2): 1-4

**head record**

[Mission](#_Toc515962734)  [I](#_Toc515962734)

[1 Introduction](#_Toc515962735)  [1](#_Toc515962735)

[1.1 Subject background and significance](#_Toc515962736)  [1](#_Toc515962736)

[1.1.1 Background](#_Toc515962737)  [1](#_Toc515962737)

[1.1.2 Significance](#_Toc515962738)  [1](#_Toc515962738)

[1.2 Research status at home and](#_Toc515962739)  [abroad1](#_Toc515962739)

[1.3 The main research work of curriculum design](#_Toc515962740)  [2](#_Toc515962740)

[2 System requirements analysis and overall design](#_Toc515962741)  [3](#_Toc515962741)

[2.1 System requirements analysis](#_Toc515962742)  [3](#_Toc515962742)

[2.1.1 System Objective](#_Toc515962743)  [3](#_Toc515962743)

[2.1.2 Transaction Process](#_Toc515962744)  [3](#_Toc515962744)

[2.2 Overall system design](#_Toc515962745)  [5](#_Toc515962745)

[2.2.1 Data Maintenance](#_Toc515962746)  [5](#_Toc515962746)

[2.2.2 Data query](#_Toc515962747)  [6](#_Toc515962747)

[2.2.3 Statistics](#_Toc515962748)  [6](#_Toc515962748)

[2.2.4 Help](#_Toc515962749)  [7](#_Toc515962749)

[3 System Detailed Design](#_Toc515962750)  [8](#_Toc515962750)

[3.1 Definition of data structure](#_Toc515962751)  [8](#_Toc515962751)

[3.1.1 Data list](#_Toc515962752)  [8](#_Toc515962752)

[3.1.2 Data Relationship](#_Toc515962753)  [9](#_Toc515962753)

[3.2 Main algorithm design](#_Toc515962754)  [10](#_Toc515962754)

[3.2.1 Basic function function](#_Toc515962755)  [10](#_Toc515962755)

[3.2.2 Data maintenance function](#_Toc515962756)  [12](#_Toc515962756)

[3.2.3 Data search and statistical functions](#_Toc515962757)  [21](#_Toc515962757)

[3.2.4 File access and other functions](#_Toc515962758)  [25](#_Toc515962758)

[3.2.5 Time and Space Complexity Analysis](#_Toc515962759)  [28](#_Toc515962759)

[4 System Implementation and Testing](#_Toc515962760)  [30](#_Toc515962760)

[4.1 System Implementation](#_Toc515962761)  [30](#_Toc515962761)

[4.1.1 Description of header files and predefined constants](#_Toc515962762)  [30](#_Toc515962762)

[4.1.2 Data structure definition](#_Toc515962763)  [30](#_Toc515962763)

[4.1.3 Function declaration](#_Toc515962764)  [34](#_Toc515962764)

[4.2 System Testing](#_Toc515962765)  [37](#_Toc515962765)

[4.2.1 Test methods and principles](#_Toc515962766)  [37](#_Toc515962766)

[4.2.2 Test plan](#_Toc515962767)  [37](#_Toc515962767)

[4.2.3 Testing](#_Toc515962768)  [43](#_Toc515962768)

[5 Summary and Outlook](#_Toc515962769)  [65](#_Toc515962769)

[5.1 Summary of the full text](#_Toc515962770)  [65](#_Toc515962770)

[5.2 Job Outlook](#_Toc515962771)  [65](#_Toc515962771)

[6 Experience](#_Toc515962772)  [66](#_Toc515962772)

[Reference](#_Toc515962773)  [68](#_Toc515962773)

[Appendix](#_Toc515962774)  [69](#_Toc515962774)

## 1 introduction

1.1 Subject Background and Significance

The class scheduling problem is a multi-factor optimization decision-making problem that solves the contradiction between time and space resources. It realizes the time-space arrangement of several basic factors that restrict each other, such as classes, teachers, time, courses, and classrooms (laboratories). This arrangement needs to satisfy Certain constraints, such as the capacity of the classroom, the teaching effect of teachers, etc.

1.1.1 background

With the continuous expansion of the enrollment scale of colleges and universities in recent years, the number of teachers and students has increased sharply, and the number of courses offered is large. If we can make full use of the limited teaching resources and efficiently arrange a high-quality class schedule, it will have a positive effect on stabilizing the teaching order and improving the quality of teaching. meaning. The problem of course scheduling is a difficult problem in teaching management. It is actually a kind of ad hoc resource scheduling problem. It is a multi-factor overall optimization problem, that is, the timetable problem, and has a wide range of application fields.

1.1.2 significance

In the teaching management work, class scheduling is a very complicated and tricky task. How to use limited teachers and limited teaching resources to arrange a reasonable course arrangement has positive significance for stabilizing teaching order and improving teaching quality.

1.2 Research status at home and abroad

1960s , some people abroad began to study the problem of curriculum arrangement. In 1962 , Gotlieb once proposed a mathematical model of the timetable problem. After that, people made a lot of in-depth discussions on the algorithm of the timetable problem and the existence of solutions. But the mathematical models used in most literatures are simplifications or supplements of Gotlieb 's mathematical models, and there is no feasible algorithm to solve the problem of curriculum. In the past 40 years, many attempts have been made to computer solutions to the timetable problem. Among them, the integer programming model of class schedule boils down the problem to find the solution of a set of 0-1 variables, but the calculation amount is very large. The branch-and-bound technique to solve the 0-1 linear optimization problem is only applicable to the small-scale schedule arrangement. Mihoc and Balas formulated the schedule as an optimization problem, and Krawczk proposed a linear programming method. Junginger simplifies the curriculum problem into a three-dimensional transportation problem, while Tripathy regards the curriculum problem as an integer linear programming problem and proposes a mathematical model of the curriculum

In addition, some literatures try to solve the problem of arranging class schedules from the perspective of graph theory, but the coloring problem of graphs is also an NP- complete problem. The model is too far from reality to be of practical value for most school schedule problems. At present, the methods to solve the timetable include: simulated manual scheduling method, graph theory method, Lagrangian method, secondary distribution method and other methods. Due to the complex constraints of the curriculum, the description of the problem with mathematical methods often leads to a dramatic increase in the scale of the problem, which has become a huge obstacle to the application of mathematical programming to solve the curriculum problem.

1.3 The main research work of curriculum design

Obtain the original classroom scheduling data from the EXCEL file of the school scheduling schedule, extract the classroom information in it, and obtain teachers, classrooms, classes, courses and other data from other resources. Classroom data can be entered manually, and automatic extraction using relevant programming interfaces is encouraged . Model the data being processed using structures such as linear tables, lookup tables, graphs, etc., and encourage the use of sorting algorithms to optimize operations if necessary. The test data covers at least all classes in the school for one semester. The logic and rules of rationality can be used to interview relevant personnel such as teachers, logistics, management, and classmates. The use of database systems to manage data is not permitted. Free play in interface design and other functions.

## 2 System requirements analysis and overall design

2 .1 System Requirements Analysis

2.1.1 \_ \_ system target

This system should first realize the database-related operations based on the 2017-18 second semester schedule of the School of Computer Science, Huazhong University of Science and Technology . These include : (1) Addition, deletion, modification, search and retrieval of teachers , classrooms, classes , courses, etc.; Retrieve classes by class, search classes by classroom, course, time, etc.

Secondly , this system should be able to carry out data statistics on the current existing course arrangement , including : searching and retrieval of vacant classrooms and classroom utilization analysis, and at the same time evaluate the existing course arrangement method with various standards . In this system , the evaluation methods specifically include: (1) the classroom energy efficiency ratio within a specific time range ; ( 2 ) the rationality of the time distribution of the courses that a specific teacher undertakes every day in a week ; The rationality of the time distribution of the courses undertaken; (4) The rationality of the floors of the classrooms for senior teachers (male : 55 years old ; female , 50 years old) ; ( 5 ) The time of the courses that a specific class takes every day within a week Rationality of distribution; (6 ) Time distribution and quantity rationality of weekly courses of a specific class; ( 7 ) Rationality of courses that a specific class is taking this semester (whether the prerequisite courses of each course have been studied before ) .

2.1.2 \_ \_ transaction process

interface similar to the windows folder window ( as shown in Figure 2-1 ) . Before entering the operation interface, the system first initializes various structures to be used . After this work is completed, the system will display as shown in Figure 2-1 .

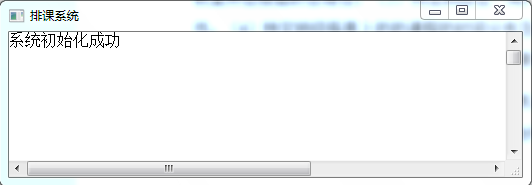


Figure 2-1 □System initialization interface

the system, the main interface is shown in Figure 2-2 . The top is the main menu bar , and the mouse or keyboard direction keys can be used to switch and select between the main menu and submenu options .



Figure 2-2 □ System main interface

Depending on the function , the system may enter the secondary menu, as shown in Figure 2-3 .

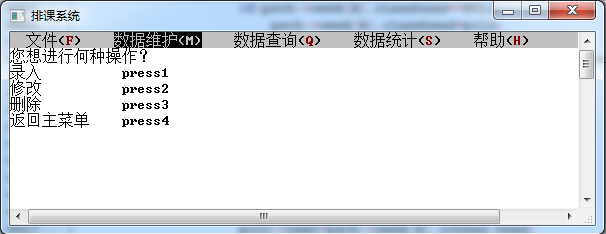


Figure 2-3 □Second-level menu interface of the system

After entering the function from the main menu or secondary menu and executing it (or directly returning to the main menu ) , there will always be a clear screen and return to the interface shown in Figure 2-2 , and other operations can continue . When the exit option is selected , the interface shown in Figure 2-4 is entered . Press any key again to actually exit the system .

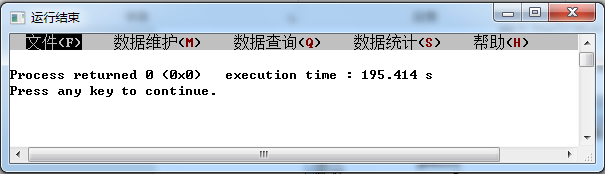


Figure 2-4 □Exit the system interface

2 .2 Overall System Design

The whole system is divided into 5 major functional modules : file , data maintenance , data query , data statistics and help.

There are three branches in the file part : data backup , data recovery and exit. Among them, the data backup saves the data in the current system into the disk file ; the data recovery can input the data saved by the data backup into the system at any time when the system is running and replace the current data ( if any ) ; exit means to end the operation and exit the system.

2. 2 .1 data maintenance

The data maintenance part has five branches: teacher information , classroom information, course information , class information and classroom information. The secondary menu under each branch provides three data maintenance methods: adding, modifying, and deleting data objects.

adding a teacher , you can add its name, gender and age; when modifying a teacher , you can modify its name, gender and age , and when modifying the teacher's name , it will also modify the corresponding information in all the classrooms taught by the teacher ; when deleting a teacher, Also deletes all classes taught by this teacher . When adding a classroom , you can add its number and capacity; when modifying a classroom , you can modify its number and capacity , and when modifying the classroom number, it will also modify the corresponding information in all the classrooms in the classroom ; All classes in this classroom . When adding a course, you can add its name and the name of each prerequisite course; when modifying a course, you can modify its name or add/delete its prerequisite courses , and modifying the course name will also modify the corresponding information in all classes; delete a course , all classes for that course are deleted. When adding a class , you can add its class number, number of people and each course you have taken; when modifying a class , you can add its class number, number of people or add/delete courses you have taken; when deleting a class , delete its subordinate class schedule structure. When adding a class , you can add its course name, class room, class teacher ( if these three information do not exist in the corresponding lookup table , it will be automatically created) , class time ( including the number of sessions, the day of the week and the number of weeks) and The class number of each class in the class ; when modifying a class , you can modify its class room, teacher and class time ( including the number of sessions, day of week and week number) ; when deleting a class , you can choose to remove a class from a class Cull or delete an entire class.

2. 2 .2 data query

The data query part has five branches: teacher information , classroom information, course information , class information and classroom information. In the teacher information, three search /retrieval methods are provided: teacher name , age interval, and gender ; in classroom information, there are two search /retrieval methods : classroom number and capacity interval ; in course information, provide The method of fuzzy search by course name substring ; in class information, provide two search /retrieval methods by class number and by number of people ; in class information, provide by teacher , class , classroom + course + class time , four search /retrieval methods by specific class time .

2. 2 .3 Statistics

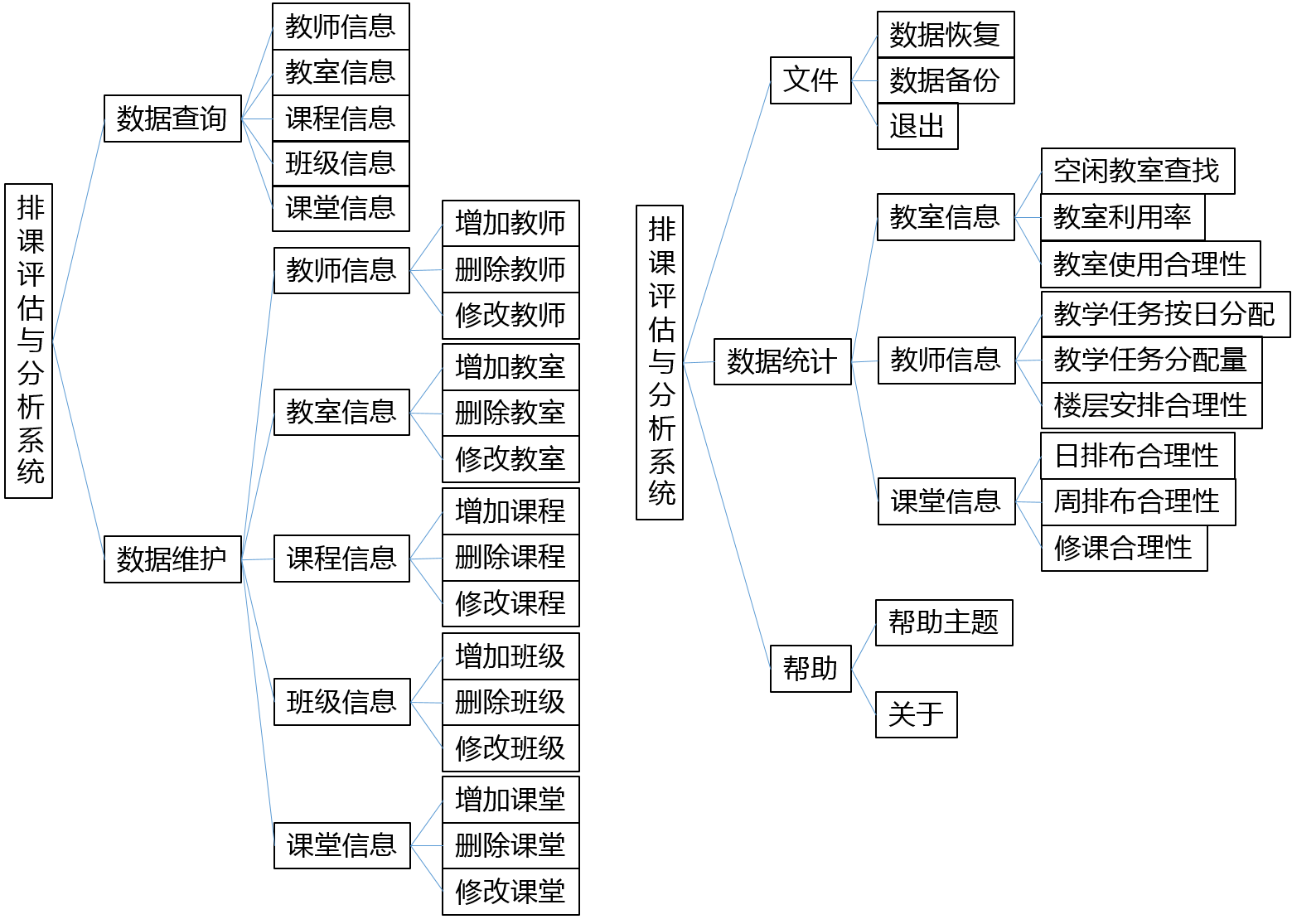
The data statistics part has three branches: teacher information , classroom information and classroom information. In the teacher information, three evaluation methods are provided: the rationality of the teacher's teaching tasks assigned by day , the rationality of the total amount of teaching tasks assigned to the teachers , and the rationality of the floor setting for senior teachers ; in the classroom information, Provides three statistical /evaluation methods for searching for free classrooms within a specific time period , teacher utilization rate within a specific time period, and the rationality of classroom -to-classroom adaptation ; in the classroom information, it provides the rationality of the course setting for a specific class in each day , the rationality of the course arrangement within a week and the rationality of taking courses in this semester (that is, whether the prerequisite courses of the courses taken by this class in this semester are included in the list of courses taken ) .

Figure 2-5 □System module structure diagram

2. 2 .4 help

are two branches in the help section : topic and about .... The help topic will open the system instruction document named "help.txt" in the directory , and about... will open a dialog box displaying the relevant properties of the system.

The system module structure diagram is shown in Figure 2-5 .

## 3 System detailed design

3.1 \_ Definitions about data structures

3.1.1 \_ \_ data list

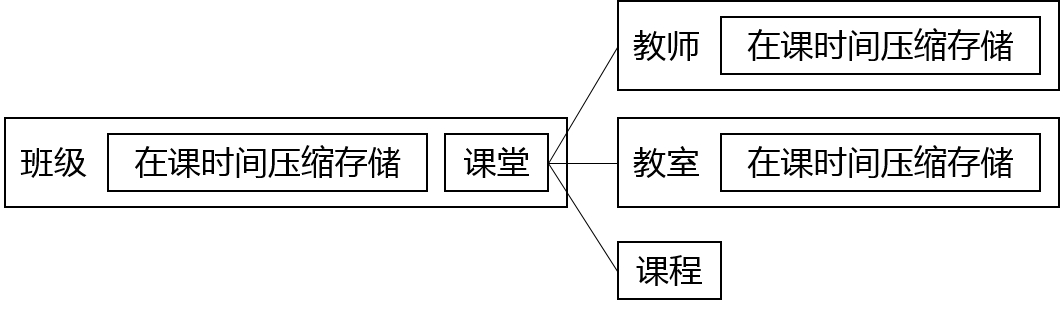
This system handles five data objects in total, which are teachers, classrooms , courses , classes, and classrooms . Among them, the classroom , teacher, and class structure also include the compressed storage table sub-object of the class time three - dimensional array . The teacher structure includes the teacher name , gender code, age, class time compressed storage structure header array; the teacher structure contains the classroom number, teacher capacity, and class time compressed storage structure header array; the course structure includes the course name , The number of pre-requisite courses, the head node pointer of pre-requisite courses; the class structure includes class number , number of people , number of courses taken, number of courses in progress, string group of courses taken, two-dimensional array of courses in progress, , pointing to the next Node pointer and date structure array , which contains the header of the class time compressed storage structure, the head node of the classroom structure, and the number of classrooms of the day; the classroom structure includes the course name , classroom number , teacher name , start section , end section, start Start week number array , end week number array ; class time compression storage structure includes class week number and class number. The specific data types are shown in Table 3-1 .

Table 3-1 □ List of data types

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| structure | variable name | | type | note |
| Teacher header structure  TeList | length | | unsigned short | lookup table length |
| list size | | unsigned short | Find the number of tablespaces |
| Elem | | Teacher\* | Head pointer |
| teacher structure  teacher | tname[20] | | char | Name |
| gender | | unsigned short | sex code |
| Age \_ | | unsigned short | age |
| O chead [6] | | Ochead | Compress and store headers during class time |
|  | | | | |
| Classroom Header Structure  R olist | length | | unsigned short | lookup table length |
| list size | | unsigned short | Find the number of tablespaces |
| Elem | | Room\* | Head pointer |
| Classroom Structure Room | code[20] | | char | classroom number |
| R size | | unsigned short | capacity |
| o chead [6] | | Ochead | Compress and store headers during class time |
|  | | | | |
| Course Topology Header Structure  Cougraph \_ | length | | unsigned short | Adjacency table length |
| list size | | unsigned short | Adjacency table space number |
| coulist | | CouHead\* | Head pointer |
| Course inverse adjacency header structure  CouHead | name [60] | | char | Course Title |
| arcnum | | unsigned short | Number of prerequisite courses |
| archhead | | CouArc\* | Prerequisite Head Pointer |
| Prerequisites \_  CouArc | prename | | char\* | Prerequisite course name |
| nextarc | | CouArc\* | Pointer to the next prerequisite |
|  | | | | |
| class header node  Grade \_ | gr\_num | | unsigned short | The number of nodes in the next class of the grade |
| schead | | Schedule\* | class node head pointer |
| class node  Schedule \_ | classname[20] | | char | shift number |
| cl size | | unsigned short | number of people |
| do\_num | | unsigned short | Number of courses taken |
| pr\_num | | unsigned short | Number of courses in progress |
| done [60][60] | | char | Courses taken |
| progress[15][60] | | char | Courses in progress |
| next | | Schedule\* | pointer to the next class |
| week[6] | | struct weekday | school day structure |
| school day structure  week | cl\_num | unsigned short | Number of classes on the day |
| classhead | Class \* | class head node |
| ochhead | Ochead | Compress and store headers during class time |
| class node  class \_ | coursename[60] | | char | Course Title |
| roomcode[20] | | char | classroom number |
| tname[20] | | char | teacher name |
| cou\_st | | unsigned short | start number |
| cou\_ed | | unsigned short | number of ending sessions |
| wk\_st[5] | | unsigned short | start week |
| wk\_ed[5] | | unsigned short | end weeks |
| next | | struct Class\* | pointer to next class |
|  | | | | |
| Compress and store header nodes during class time  Ochead | len | | unsigned short | Compress storage node length |
| the head | | Occupy\* | Compressed Storage Node Nod Pointer |
| Compress storage nodes during class time  Occupy | week | | unsigned short | Number of weeks |
| time | | unsigned short | Number of classes |
| next | | struct Occupy\* | pointer to the next node |

3.1.2 \_ \_ data relationship

Each class is organized according to the grade it is in, and the classroom creates a copy node for each class and attaches it to the corresponding time of the corresponding class. Corresponding classrooms , classrooms, and teachers are recorded in the classroom to achieve association . Classes , teachers and teachers each have six groups of class time compression storage lookup tables to record their respective class time from Monday to Saturday.

Figure 3-1 □Data relationship diagram

What to write in this part: (1) First describe the data to be processed in the system, which data items are included in each type of data, and the data type of each data item, and finally can be expressed in a table; (2) describe these various How the data is related in the system can be intuitively illustrated through the diagram.

3.2 Main Algorithm Design

3.2.1 \_ \_ basic functions

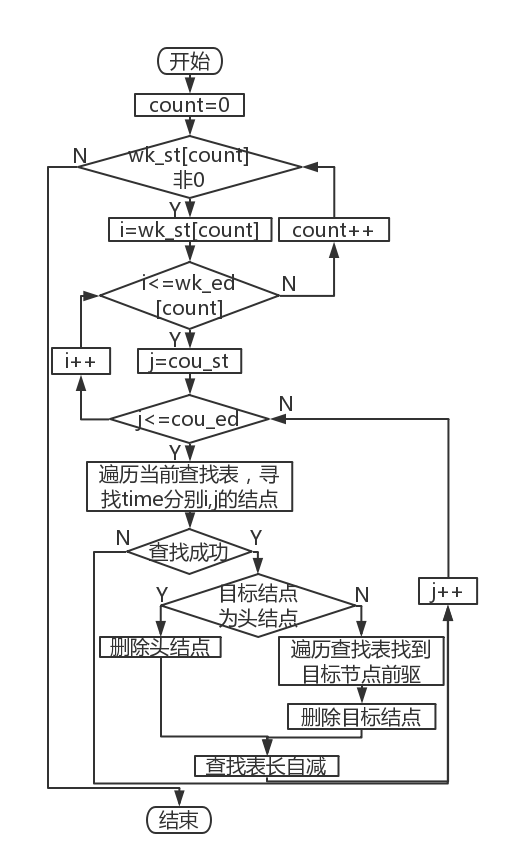
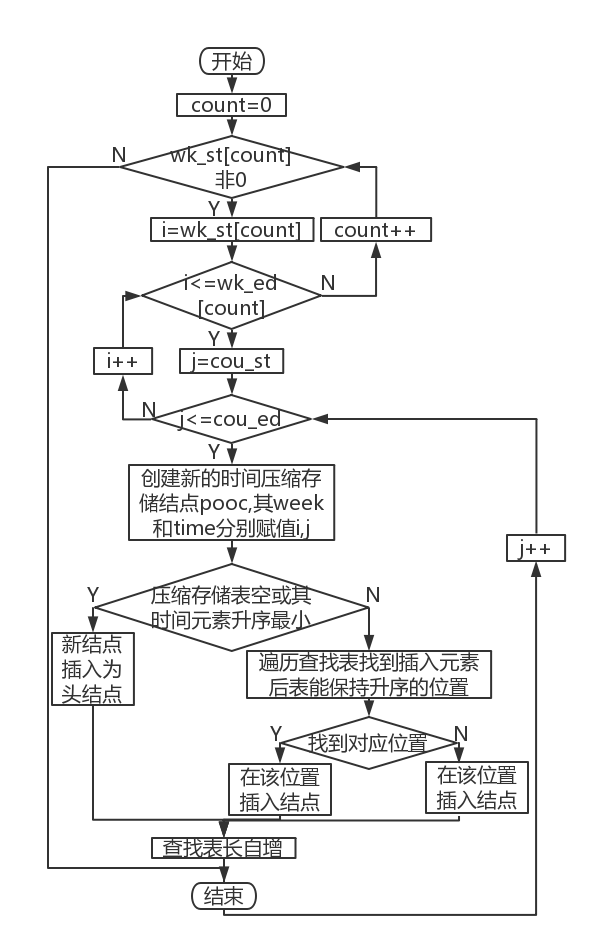
The code in this part is used repeatedly in each function, so it is extracted as an independent function to simplify the code and eliminate reuse .  

Figure 3-2 □Delete class time node flow chart Figure 3-3 □Flowchart of adding class time nodes

1. Delete the compressed storage node function DelTimeNode during class time : the function parameters are the start week number array wk\_st[] , the end week number array wk\_ed[] , the start session number cou\_st , the end session number cou\_ed and the class time compressed storage header pointer ochead . The first loop determines whether the start /end week number at the current position of the array is valid. Since the two arrays must have the same number of valid elements , it is only necessary to determine whether the current element of the start week number array is 0. When the above conditions are valid, use double loops to take over the current week number and node value, find and delete the eligible node in the compressed storage lookup table , and discuss whether the target node is the head node or not . The flowchart is shown in Figure 3-2 .

2. Add the AddTimeNode function of compressed storage node during class time : the function parameters are the start week number array wk\_st[] , the end week number array wk\_ed[] , the start section number cou\_st , the end section number cou\_ed and the class time compressed storage header pointer ochead . The first loop determines whether the start /end week number at the current position of the array is valid. Since the two arrays must have the same number of valid elements , it is only necessary to determine whether the current element of the start week number array is 0. When the above conditions are valid, use a double loop to take over the current week number and node value, traverse the existing compressed storage lookup table , if the table is empty, directly use the new node as the head node , otherwise search for the position that meets the requirements in ascending order of time , The standard is that the number of weeks succeeded by the current node is greater than the number of weeks of the new node, or the number of weeks succeeded by the current node is less than or equal to the number of weeks of the new node, and the number of nodes succeeded by the current node is greater than the number of nodes of the new node. If the corresponding position is found , insert the node to the target position, otherwise insert the end of the list. The flowchart is shown in Figure 3-3 .

3. Class name input function ClassInput : the parameter is the new class buffer string cbuf3. After inputting a new class into the string , determine the position of the head node according to its grade, find the existence of the class in the corresponding class list , and return the pointer to record the position of the node that meets the requirements, and return NULL if the search fails.

4. Insert new classroom function AddNewRoom : The parameter is the string cbuf2 storing the new classroom . If the length of the table is greater than or equal to the capacity of the table, newly allocate space with the number of LISTINCREMENT (the macro definition in the program is 10) for the lookup table , and output a prompt message and return if the allocation fails . Enter the teacher's information at the end of the table and initialize the six compressed storage table lengths and header pointers to 0 and NULL respectively .

5. Insert the new teacher function AddNewTeacher : the idea and implementation method are the same as the insert new classroom function.

6. Insert the new course function AddNewCourse : the idea and implementation method are the same as the insert new classroom function.

7. SeekRemClass function for querying remaining classes : the parameters are the class node pointer psch and the buffer string cbuffer[]. Find the course in the cbuffer in the class 's course list , if found, traverse all the classes of the class to find out whether there is such a course, if not, delete the course from the class's course list , in the course list The length is auto-decremented. The flowchart is shown in Figure 3-4 .

8. Delete class node function DelClassNode: the parameters are class node pointer pcls, class node pointer psch, week number subscript j. If pcls is the head node of the class list under psch- >week[j] , delete pcls, if psch->week[j].classhead is empty , then set the length of the class list psch->week[j].cl\_num to 0 and return directly . If pcls is not the head node , then traverse the class linked list to find the predecessor of pcls , and after deleting pcls, the length of the linked list psch->week[j].cl\_num will be decremented .

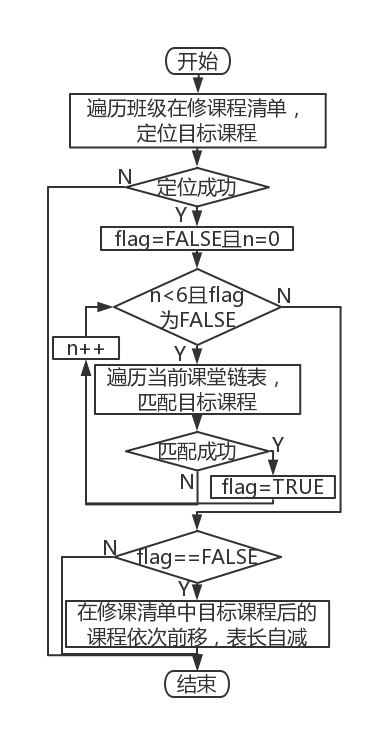
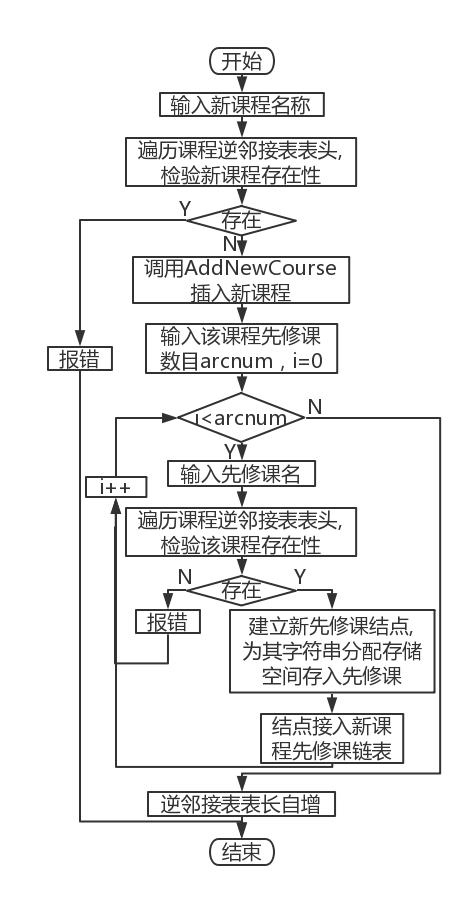
 

Figure 3-4 □ Flowchart of querying remaining classes Figure 3-5 □Course adding flow chart

9. Insert time function TimeInput : The parameters are start time pointer cou\_st, end time pointer cou\_ed , start week number array wk\_st[] , end week number array wk\_ed[]. Set all the elements of the two arrays to 0, input the values of cou\_st and cou\_ed , and then cyclically input the start and end weeks of each segment and judge whether to continue.

3.2.2 \_ \_ data maintenance function

This part of the function implements the addition , modification and deletion of various data objects (including classrooms, teachers, courses, classes, classrooms ) in the system.

1. Teacher addition function AddTeacher: Input a new teacher, find out whether the teacher already exists in the teacher linear table , if it exists, output a prompt message and return, otherwise call AddNewTeacher to insert a new teacher, input its gender code and age, and the linear table length will increase automatically . clear screen .

2. Classroom addition function AddRoom : Input a new classroom, find out whether the classroom already exists in the teacher linear table , if it exists, output a prompt message and return, otherwise call AddNewRoom to insert a new classroom , input its capacity , and the linear table length will increase automatically . clear screen .

3. Course addition function AddCourse : input a new course , find out whether the classroom already exists in the teacher linear table , if it exists, output a prompt message and return , otherwise call AddNewCourse to insert a new course , input the number of prerequisite courses and the corresponding number of prerequisites Take courses. For each prerequisite course , check its existence in the course linear table, if it exists, create a prerequisite course node for it and connect to the new course node, otherwise output an error message and return . The length of the linear table is auto-incremented . clear screen . The flowchart is shown in Figure 3-5 .

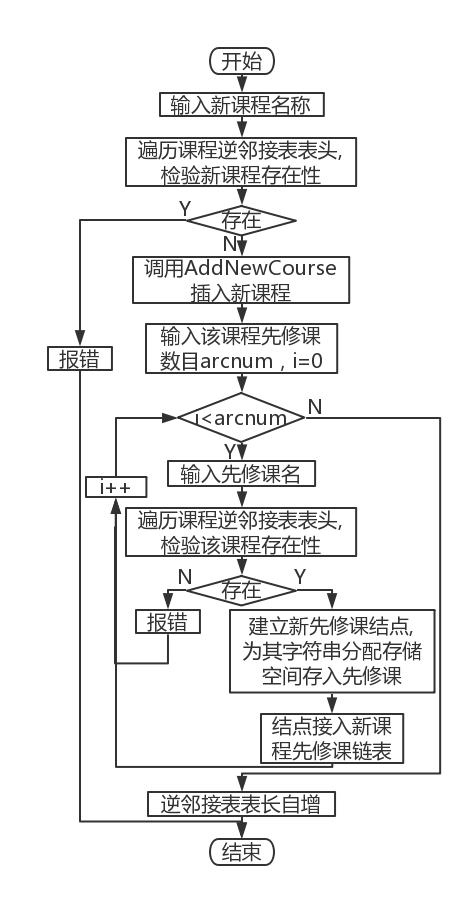


Figure 3-5 □ Add classroom flow chart

4. Class addition function AddGroup : Input a new class, call ClassInput to check the existence of the class , if it already exists , output an error message and return. Create a new class node psch , copy the class string, enter the number of people , the number of courses that have been taken and the corresponding courses that have been taken, and set the head pointer and class head pointer of the six week structures to NULL. The length of the time compression storage table and the length of the class linked list are set to 0 . According to the grade of the new class, insert the new node as the head pointer of the class list , the length of the corresponding course link list will increase automatically , and the screen will be cleared .

5. Teacher deletion function DelTeacher : Input the name of the teacher to be deleted, check its existence in the teacher linear table , if it does not exist, output an error message and return. The double cycle deletes all the class time compression storage nodes under the course . Find all classrooms where the teacher is the target teacher in the class linked list , and use variables to store all relevant information of the node , including classroom number , teacher name, array of starting weeks , array of ending weeks, starting sessions, and ending sessions. Call DelTimeNode to delete the time node corresponding to the class in the class time compression storage table under the class , call SeekRemClass to check whether there is a class with the same name, if not, remove the course from the class's current course list. Locate the classroom used by the classroom in the classroom lookup table , call DelTimeNode to delete the time node corresponding to the classroom in the class time compression storage table under the classroom , and call DelClassNode to delete the classroom. After all related classrooms are deleted, the target teacher is deleted , and the length of the linear table is reduced. If it is reduced to 0, a corresponding prompt message will be output . clear screen .

6. Classroom deletion function DelRoom : the idea and implementation method are the same as the teacher’s deletion function, the only difference is that when deleting all classrooms as the target classroom, DelTimeNode should be called to delete the time period corresponding to the class in the teacher’s subordinates’ class time compression storage table point.

7. Course deletion function DelCourse : Input the course to be deleted , check its existence in the course inverse adjacency list , if it does not exist, output an error message and return. A loop deletes all prerequisite course nodes of the target course . Find all the classes of this course in the class linked list , and use variables to store all relevant information of the node , including classroom number , teacher name, array of start weeks , array of end weeks, number of start sessions, and number of end sessions. Call DelTimeNode to delete the time node corresponding to the class in the class time compression storage table under the class , call SeekRemClass to check whether there is a class with the same name, if not, remove the course from the class's current course list. Locate the classroom used by the classroom in the classroom lookup table, and call DelTimeNode to delete the time node corresponding to the classroom in the class time compression storage table under the classroom . Locate the teacher of the class in the teacher lookup table , and call DelTimeNode to delete the time node corresponding to the class in the teacher's subordinate in-class time compression storage table . Call DelClassNode to delete the class. After all relevant courses are deleted, delete the target course , and the length of the inverse adjacency table will be reduced. If it is reduced to 0, the corresponding prompt message will be output . clear screen . The flowchart is shown in Figure 3-6 .

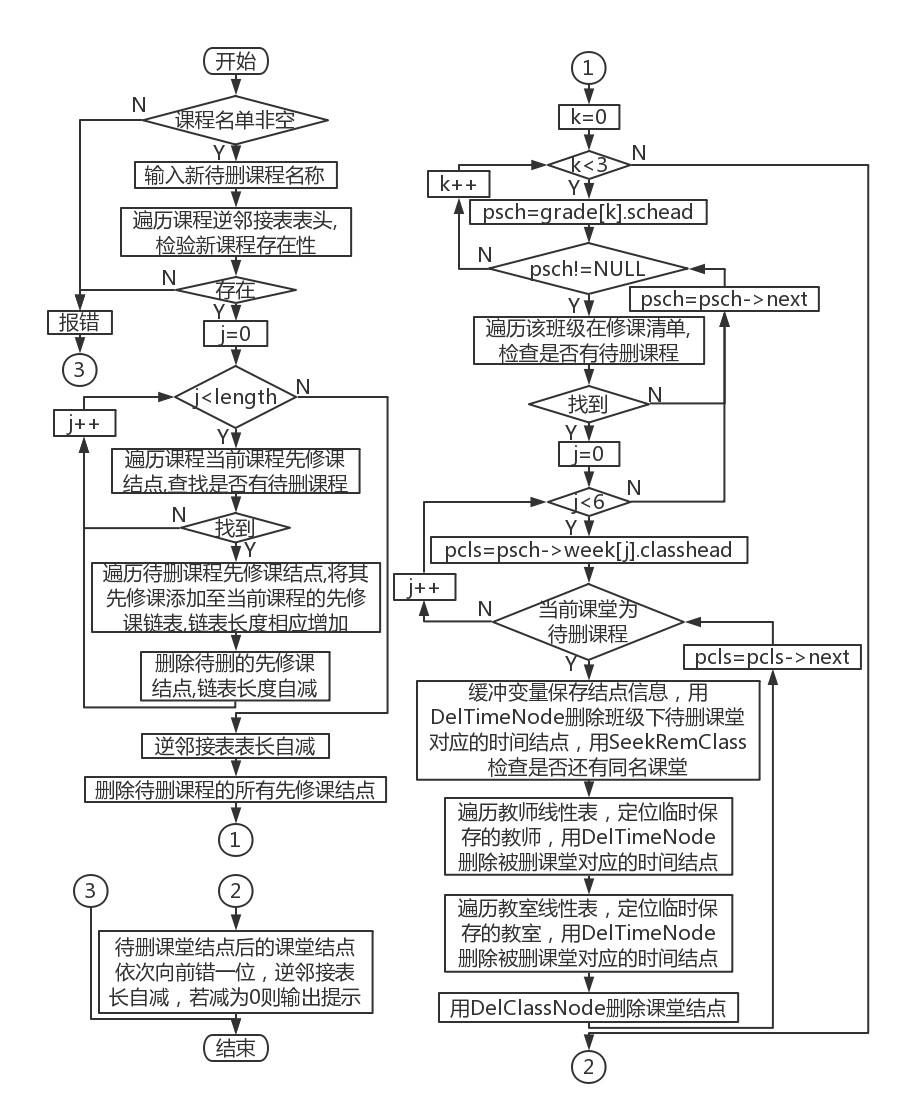


Figure 3-6 □Course deletion flow chart

8. Class deletion function DelGroup : Input the class to be deleted , enter the class number of the class to be deleted after confirming the deletion , call ClassInput to locate the corresponding class node, and delete all class time compression storage nodes and classroom nodes under the class in a double cycle . Delete the target class node . When the node is the head node of the grade, the head pointer directly points to its successor . Otherwise, it is necessary to traverse the class list to find its predecessor, and use the predecessor to point to the successor. Delete header stickers and clear the screen.

9. Teacher modification function AltTeacher: first determine whether the teacher linear list is not empty, and if it is empty, report an error and return. Enter the teacher's name , traverse the linear table to check its existence , and output an error message if it does not exist and return . Choose a modification method. Modify the teacher's name : Enter the new teacher's name and traverse the linear table to check its existence . If it already exists, report an error and return . Traverse the class list, find all the classrooms where the teacher is the teacher to be changed and write the name of the new teacher, and finally write the new name into the target teacher node . Modify gender : If the original gender code is not 0, change it to 0, if it is 0, change it to 1, and then output a prompt message. Modify age : Write the new age directly to the teacher node . clear screen .

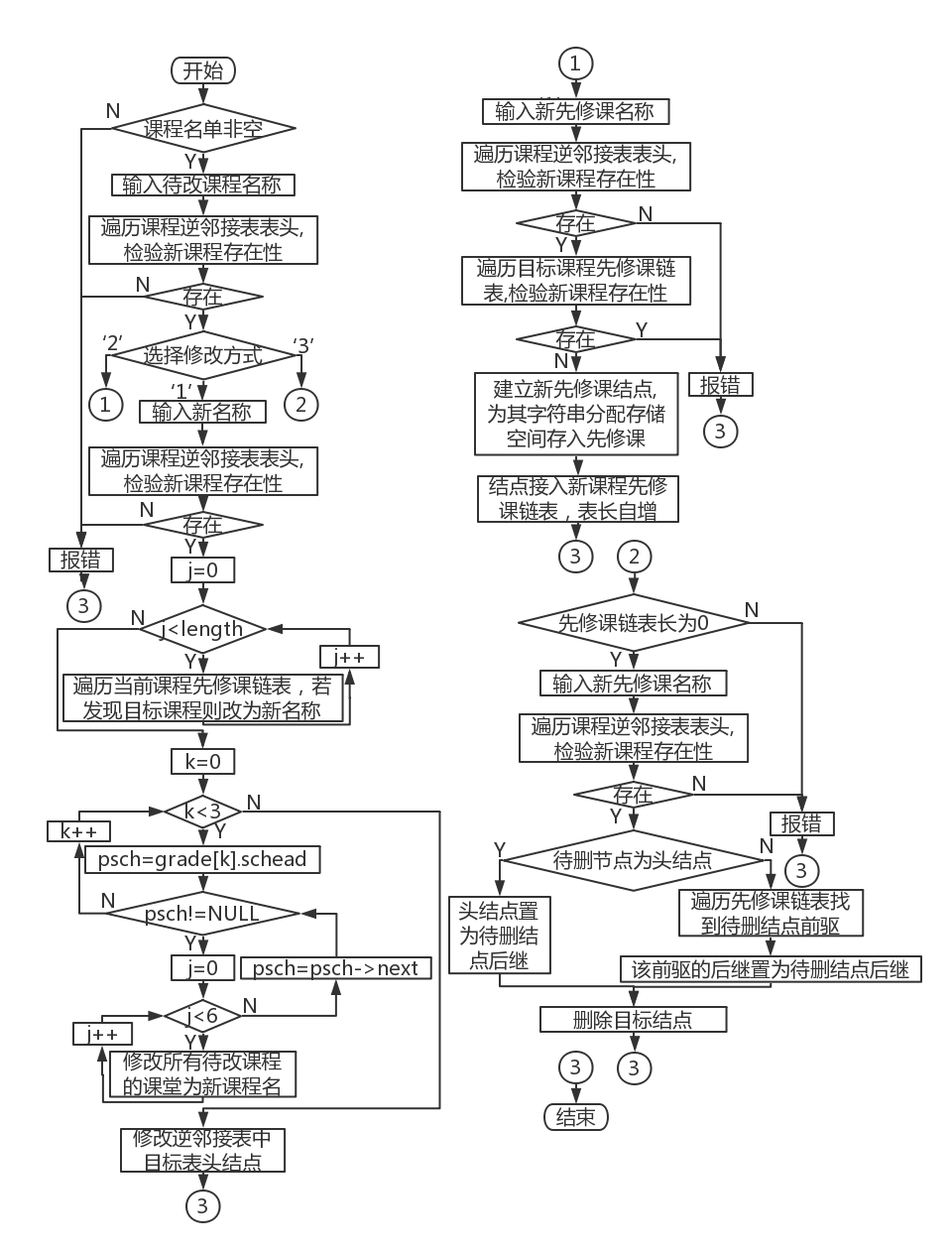


Figure 3-7 □Course deletion flow chart

10. Classroom modification function AltRoom : first determine whether the classroom linear table is not empty, and if it is empty, report an error and return. Enter the classroom number , traverse the linear table to check its existence , and output an error message and return if it does not exist . Choose a modification method. Modify classroom number: input a new number and traverse the linear table to check its existence . If it already exists, report an error and return . Traverse the class list, find all the classrooms where the classroom is the classroom to be changed and write the new number , and finally write the new number into the target classroom node . Modify capacity : write new capacity directly to the classroom node . clear screen .

11. Course modification function AltCourse : first determine whether the course inverse adjacency table is not empty, and if it is empty, report an error and return. Enter the course name , traverse the header of the inverse adjacency table to check its existence , and output an error message and return if it does not exist . Choose a modification method. Modify course name: Enter a new name and traverse the inverse adjacency list to check its existence . If it already exists, report an error and return . Traverse the inverse adjacency list , find all the prerequisite course nodes of the course and modify the course name. Traverse the class list, find all the classes of the course and write the new class name , and finally write the new name into the target course node . Add a prerequisite course : Enter the name of a new prerequisite course and traverse the inverse adjacency list to check its existence . If it already exists, report an error and return . Traverse the prerequisite course list of the target course , check the existence of the new prerequisite course, and report an error and return if it already exists . Create a prerequisite course node and an empty string for the new prerequisite course , write the prerequisite course into the empty string and insert the new node into the prerequisite course linked list. Return to the starting position of the function when you choose to continue input , and clear the screen when you exit . The flowchart is shown in Figure 3-7 .

12. Class modification function AltGroup : input the name of the class to be called , call ClassInput to check its existence and return the node address . If the node does not exist , report an error and return . Choose a modification method. Modify shift number: input a new shift number and check its legality . Traverse the class list according to the grade of the class . If the new class number is already occupied , an error will be reported and returned. Write the new class number to the class node . Add a course taken : Enter the name of a new course taken , traverse the list of courses taken to check its existence, and return an error if it exists. A new course is inserted at the end of the table , and the length of the list of courses taken is automatically incremented. Modify the course taken: input the name of the course to be taken , traverse the list of courses taken to check its existence, and return an error if it does not exist . Enter the new course name , traverse the list of courses taken to check its existence, and return an error if it exists . Overwrite the original class name with the new class name . Delete a course that has been taken: Enter the name of the course to be deleted, traverse the list of courses taken to check its existence, and return an error if it does not exist . All the courses to be deleted are shifted one place forward to achieve the deletion effect , and the length of the list of courses taken is reduced . Modify the number of people : directly write the new number of people to the class node. clear screen .

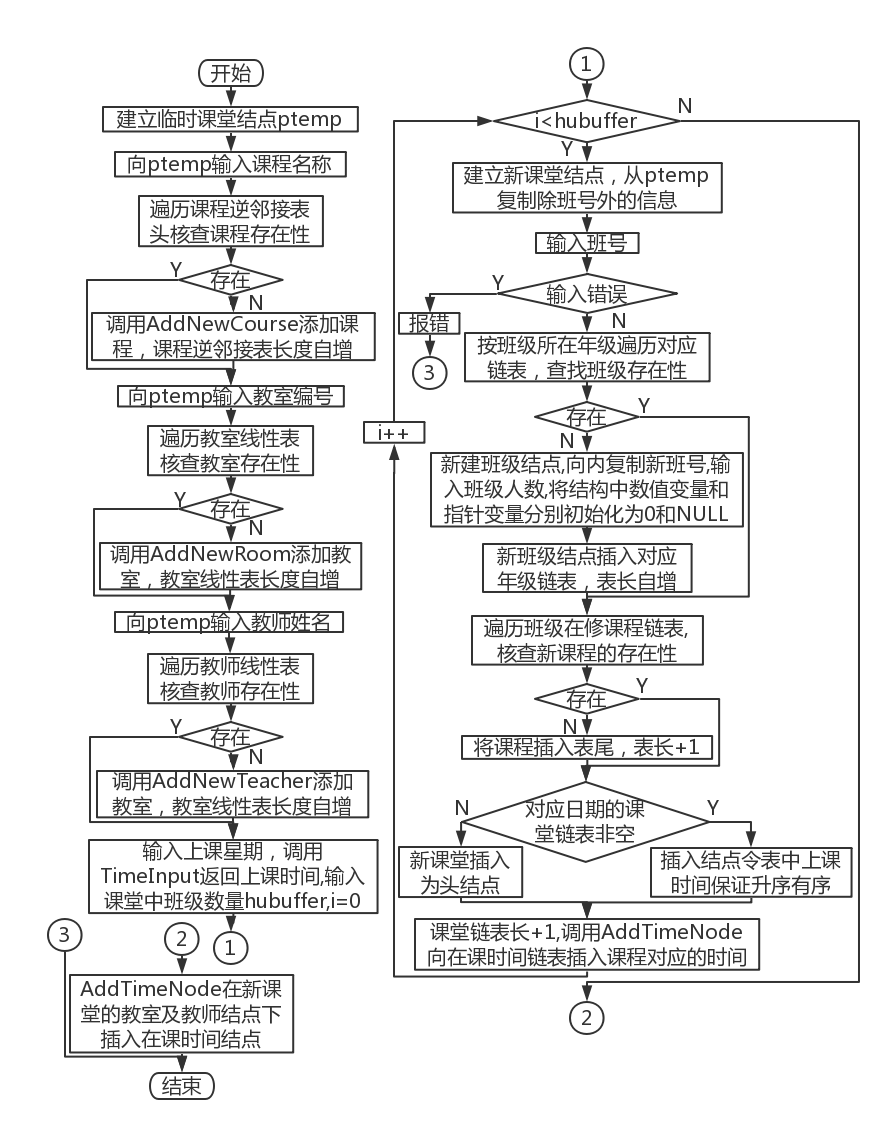


Figure 3-8 □Classroom Adding Flowchart

13. Classroom adding function AddClass : Create a temporary class node . Enter the course name into the temporary node , traverse the inverse adjacency list of the course to check the existence of the course, if it does not exist , call AddNewCourse to add a new course , and the length of the inverse adjacency list will increase automatically . Enter the teacher number and teacher name into the temporary node according to the same process . Call TimeInput to write class time to the node . Enter the number of classes, and in each cycle , create a new course node and copy the entered information in the temporary node to the new node. Enter the class number of the class you are attending , and search for the existence of the target class according to its grade . If the class number is entered incorrectly, an error will be reported and returned . If it does not exist, create a new class node , initialize the number of courses taken and the number of courses in progress to 0, enter the number of new classes , and initialize the lower nodes and values of the six subordinate date structures to NULL and 0 , Insert the new class node into the linked list. Insert the classroom nodes into the classroom linked list in ascending order according to the scheduled class date, and the length of the course linked list will increase automatically . At the same time, call AddTimeNode to insert the class time compression storage node under the class . After all classes are added , call AddTimeNode to insert class time compression storage nodes under the nodes corresponding to classrooms and teachers in this class . clear screen . The flowchart is shown in Figure 3-8 .

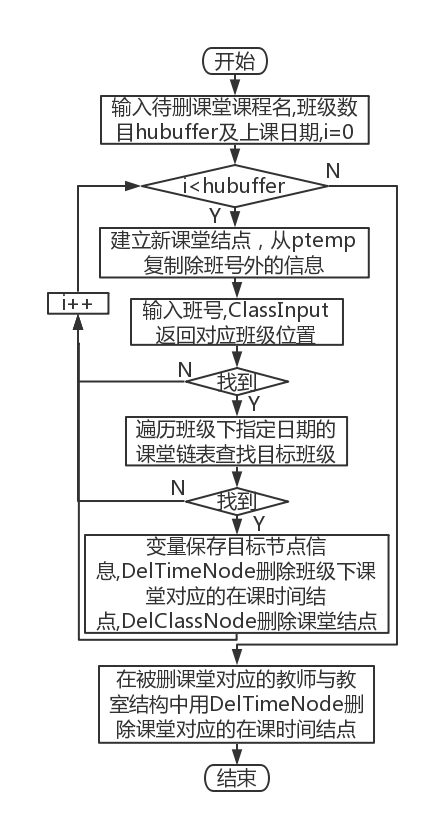


Figure 3-9 □Classroom deletion flow chart

14. Classroom deletion function DelClass : Input the course name, number of classes and class days of the class to be deleted. In each cycle , input the class number and call ClassInput to locate the target class node , if not found, report an error and enter the next cycle. Under the class, locate the class to be deleted in the class list according to the class date . If it is not found, it will report an error and enter the next cycle. Use variables to store the class information, including classroom number , teacher name, start week array , end week array, start section number, and end section number. Call DelTimeNode to delete the corresponding class time node, and call DelClassNode to delete the target class node. Locate the teacher and classroom of the deleted classroom in the teacher and classroom linear table , and call DelTimeNode to delete the corresponding class time node. clear screen . The flowchart is shown in Figure 3-9 .

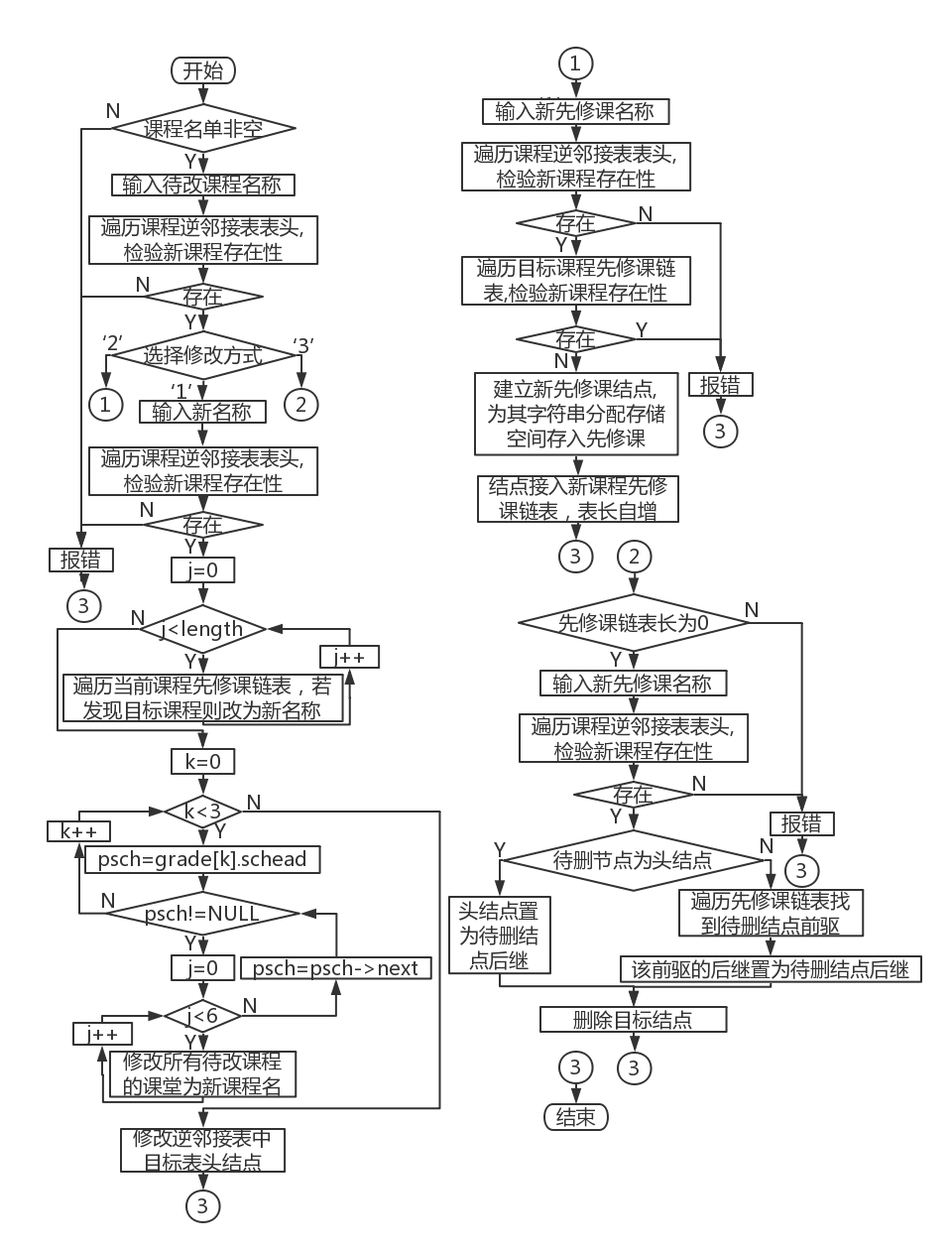


Figure 3-10 □ Flow chart of class modification

15. Class modification function AltClass : Input the course name, class number and class day of the class to be modified, and select the modification method. Modify the classroom : write a new classroom into the buffer string , traverse the linear table of classrooms to check its existence , if it does not exist, add it with AddNewRoom, and the length of the linear table is self-increased . Modify the teacher : write a new teacher into the buffer string , traverse the linear table of teachers to check its existence , if it does not exist, add it with AddNewTeacher , and the length of the linear table will increase automatically . Modify class time: use TimeInpu to input new class time. In each cycle , use ClassInput to locate the target class node, if not found, report an error and enter the next cycle. Under the class, locate the class to be deleted in the class list according to the class date . If it is not found, it will report an error and enter the next cycle. Use variables to store the class information, including classroom number , teacher name, start week array , end week array, start section number, and end section number. Modify classroom: Copy the new teacher ID from the buffer string into the classroom node. Modify teacher: Copy the new teacher's name from the buffer string into the class node. Modify the class time: use DelTimeNode to delete the original class time node, and use variables to copy the classroom and teacher occupied by the class. Assign the new class time to the class node , and use AddTimeNode to insert the new class time node . After the loop is over , modify the classroom: traverse the classroom linear table to find the original classroom , use DelTimeNode to delete the node of the current class time of the target course , and use AddTimeNode to add the node of the current class time under the new classroom . Modify the teacher: traverse the teacher linear table to find the original teacher , use DelTimeNode to delete the class time node of the target course , and use AddTimeNode to add the class time node under the new teacher . Modify the class time: Traverse the classroom linear table to find the original classroom , delete the original class time node with DelTimeNode , and add a new class time node with AddTimeNode . Traverse the teacher linear table to find the original teacher , use DelTimeNode to delete the original class time node , and use AddTimeNode to add a new class time node. clear screen . The flowchart is shown in Figure 3-10 .

3.2.3 \_ \_ Data search and statistical functions

1. Teacher lookup function SeekTeacher : first check whether the teacher linear list is empty, and return an error if it is empty. Choose a search method. Search by name : Enter the name of the teacher to be checked, traverse the teacher linear table to check the existence of the target teacher, return if not found , otherwise output gender and age information. Search by age range : input the upper and lower limits of age, traverse the linear table of teachers, and output the names , genders and ages of all teachers who meet the requirements . Search by gender : Enter the gender code , traverse the linear list of teachers, and output the names and ages of all teachers who meet the requirements . clear screen .

2. Classroom lookup function SeekRoom : first check whether the classroom linear list is empty, and return an error if it is empty. Choose a search method. Search by number : Enter the number of the classroom to be checked, traverse the linear table of classrooms to check the existence of the target classroom, return if not found , otherwise output capacity information . Search by capacity interval : input the upper and lower limits of capacity , traverse the linear table of classrooms, and output the numbers and capacities of all classrooms that meet the requirements . clear screen .

3. Course lookup function SeekCourse : first check whether the inverse adjacency list of courses is empty, and return an error if it is empty. Enter the number of the substring of the course to be checked , traverse the classroom linear table for fuzzy search , if a certain course contains the target substring , output the target information, traverse the adjacency list of its prerequisite courses and output each prerequisite course ; traverse the entire inverse adjacency list, if, a certain If the prerequisite course of the course is the target course, then output this course as the follow-up course , and output a prompt message if it is not found . clear screen .

4. Class lookup function SeekGroup : first check whether the class list divided by grade is empty, and return an error if it is empty. Choose a search method . Search by class number : Enter the class number of the class to be checked, use ClassInput to locate its node , and return an error if it is not found . Output the number of classes , the list of courses taken and the list of courses in progress. Search according to the number of people: input the upper and lower limits of the number of people, traverse the class list, and output the class number , number of people, the list of courses taken and the list of courses in progress for all classes that meet the requirements . clear screen .

5. Class search function SeekClass : Select the search method . Search by teacher : input the target teacher , and traverse the class list in a quadruple cycle . If the teacher of a certain course is the target teacher, output the class , classroom and class time of the course . Search by class: input the class number to be searched, use ClassInput to return its node position , if not found, report an error and return . The double cycle traverses the classroom linked list under the class, and outputs the teacher of each class , the teacher and the time of the class. Search by classroom + course + time : input the course name, classroom number, and class date of the class to be searched, and use TimeInput to return the number of weeks and sessions of the class. The triple cycle traverses the classroom list of the specified date for each class. If the class time, classroom and course are completely consistent, output the teacher of the class and classroom . Search by time: Output the number of beginning and end sections , class dates and beginning and ending weeks . The quadruple cycle traverses the classroom linked list of the specified date , visits the class time node under it , and if the time recorded by a node is found within the given target time, then output the course , classroom , class and teacher of the class .

6. Classroom scheduling evaluation and statistical function RoomUsage : input analysis method . Find free classrooms : Enter the target week number, date and number of sessions, double cycle through the class time list of the specified date under each classroom , if the time recorded by the current node is within the given time range, it means that the classroom is in If it is occupied within the target time period , skip the classroom, otherwise output the number of the classroom. Statistical classroom utilization rate : input the beginning and end weeks, beginning and ending dates, and beginning and ending sessions, and triple loop through the class time list of each classroom within a given date range. If the time information recorded by the node is within the given time range, then Indicates that the classroom is occupied. Use a counter to count such classes as the numerator, and take the product of the number of teachers and the given time range as the numerator, and the result is the classroom utilization rate within the target time range. Classroom usage rationality: Enter the classroom , traverse the classroom linear table to check the existence of the target classroom, and return an error if it does not exist. Output the capacity of the target classroom. Enter the course name and class date of the class to be analyzed , use TimeInput to return the class time, triple loop through the class list of each class under the specified date, if there is a class , record the number of people in this class, if no class that meets the requirements is found Report an error and return . Count the total number of classes in the target classroom and compare it with the classroom capacity . The evaluation is divided into four grades. If the seat occupancy ratio is less than 0.5, it is too sparse . It is recommended to change to a smaller classroom ; greater than 0.5 and less than 0.8 is appropriate ; greater than 0.8 and less than 1 If it is too crowded, it is recommended to change to a larger classroom ; if it is greater than 1 , it means the capacity is too small, and it is necessary to change to a larger classroom . clear screen . The flowchart is shown in Figure 3-11 .

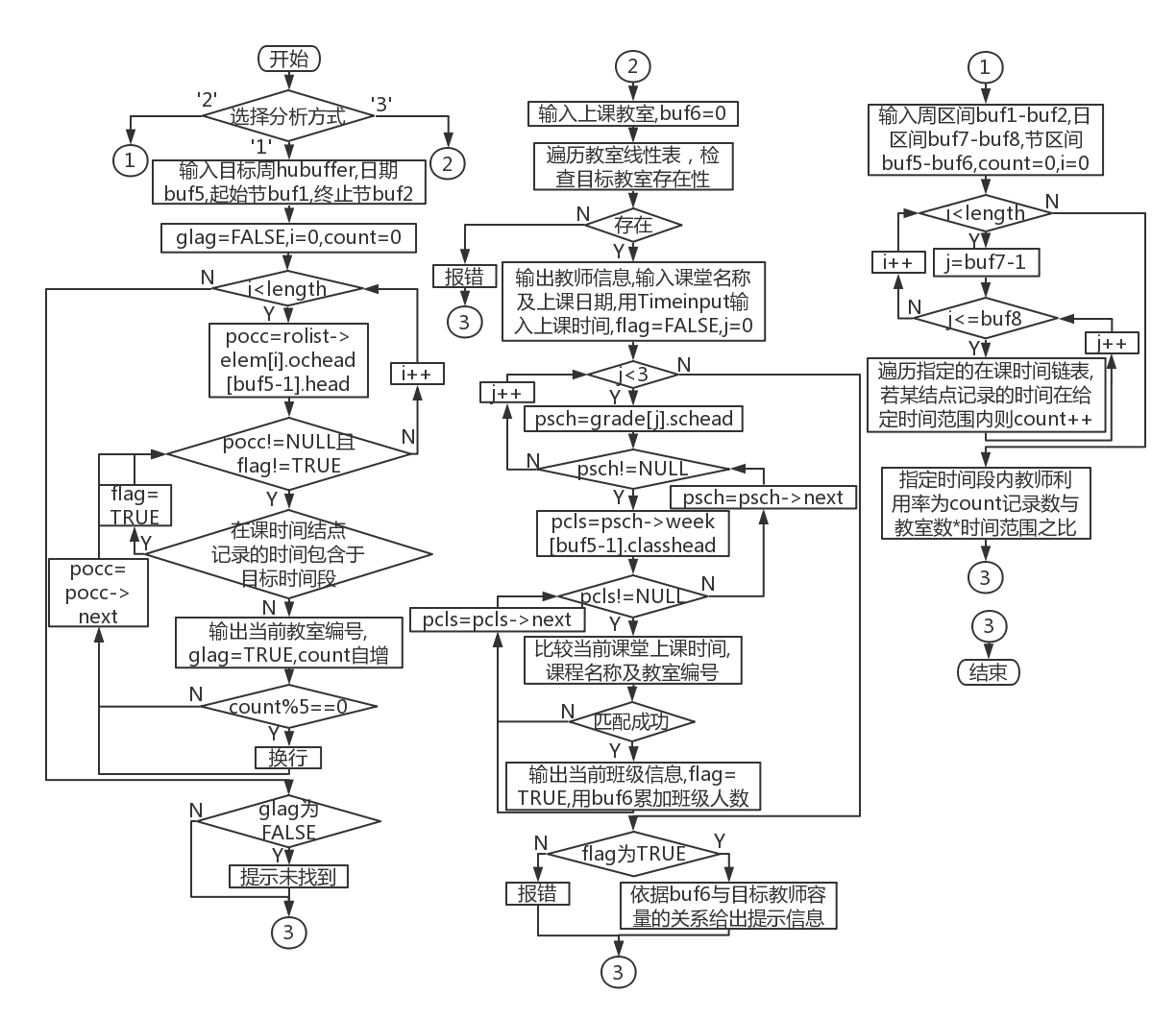


Figure 3-1 1 □ Flow chart of classroom scheduling evaluation and statistics

7. The evaluation function ClassAnalysis of class scheduling rationality : use ClassInput to return the position of the class node to be analyzed . Returns an error if the class is not found . Choose an analysis method . Analyze the rationality of the daily arrangement of courses: the judging criterion is whether to arrange all the courses during the day as much as possible . Triple cycle traverses the class time list under the class , respectively accumulating the number of nodes whose class time is 1-4 , 5-8 , and 9-12 every week . If the number of evening classes is greater than the number of morning classes or afternoon classes Half of the number of sessions , it is considered that there are too many evening classes this week , otherwise it is said that the proportion of evening classes this week is appropriate . Analyze the rationality of the daily arrangement of courses within a week: input the number of effective semesters in this semester, traverse the linked list of class hours under the class in a double loop , and use the occupied two-dimensional array with the element set to 0 in advance to count the course sessions every day of the week number . For each effective week of this semester , compare the number of courses per day in pairs . If the difference between the two days of courses is greater than or equal to 6 periods , it is judged that the daily arrangement of courses is too wide. If there is no such judgment in a week , it is considered that the weekly schedule of courses is more reasonable . Then count the number of courses in the first and second weeks and the penultimate first and second weeks according to the occupy matrix . Under normal circumstances , the number of courses in these four weeks should not be too many. If there are more than 20 classes in the first week , more than 22 classes in the second week , more than 10 classes in the penultimate week , and more than 4 classes in the last week , it will be considered If there are too many courses in the corresponding week , a prompt message will be output. Analyze the rationality of taking courses: For each course in the class’s course list, traverse its prerequisite courses in the course inverse adjacency table, for each prerequisite course, if the course does not appear in the class’s course list , it is considered that the class should take the course after taking the prerequisite course, and it is judged that the course should not be taken in this semester , and the prompt information is output . clear screen . The flowchart is shown in Figure 3-12 .

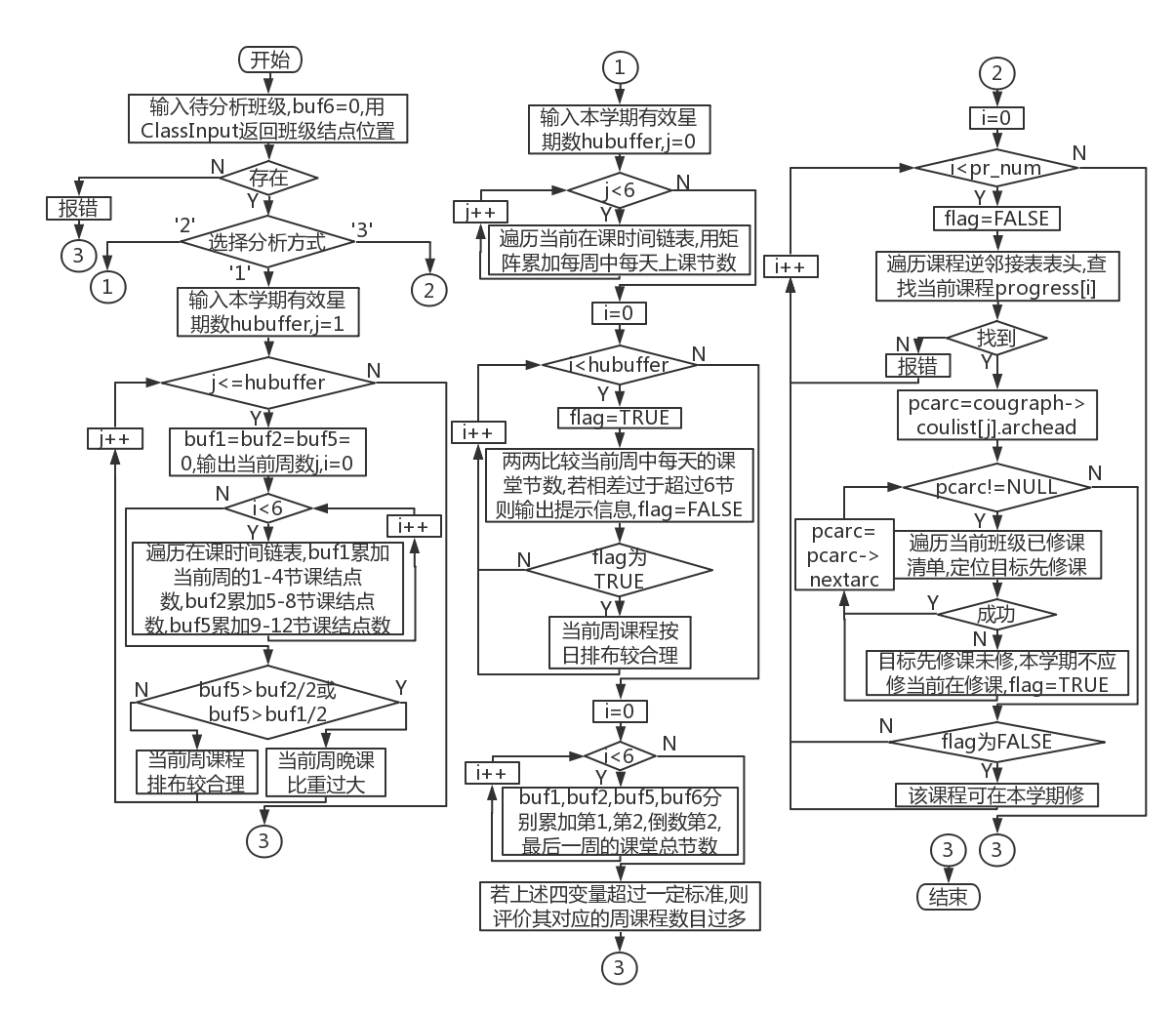


Figure 3-12 □ Flowchart of evaluating the rationality of class scheduling

8. TeacherAnalysis evaluation function for teacher arrangement rationality : Input the teacher to be analyzed , traverse the linear list of teachers to check their existence , and report an error and return if they do not exist. Choose an analysis method . Analyzing the rationality of the daily assignment of teaching tasks : the judging criterion is whether or not all classes are arranged during the day as much as possible . Triple cycle traverses the class time list under the class , respectively accumulating the number of nodes whose class time is 1-4 , 5-8 , and 9-12 every week . If the number of evening classes is greater than the number of morning classes or afternoon classes Half of the number of sessions , it is considered that there are too many evening classes this week , otherwise it is said that the proportion of evening classes this week is appropriate . Analysis of teaching assignments: The criterion is whether teachers over a certain age ( different standards for men and women) take on too many classes in a certain week. Enter the number of valid weeks in this semester , and double cycle through the linked list of each class time under the teacher node to count the number of teaching sessions per week . In this procedure , when it is judged that female teachers are younger than 50 years old and male teachers are younger than 55 years old , if they teach more than 14 lessons in a certain week , it is judged that their teaching tasks in that week are too heavy; If there are more than 10 lessons , it is judged that the teaching task of the week is too heavy . Analyzing the rationality of the teaching floor : traversing each class - classroom linked list in a quadruple loop . When the teacher in this class is the target teacher , if his age is higher than the above standard and the classroom is on the fourth or fifth floor , then it is judged that the classroom is suitable for the teacher. The words are too high, and the classroom arrangement is unreasonable . If there is still no similar judgment after the traversal is completed, it is said that the classrooms taught by the teacher are more reasonable. clear screen . The flowchart is shown in Figure 3-13 .

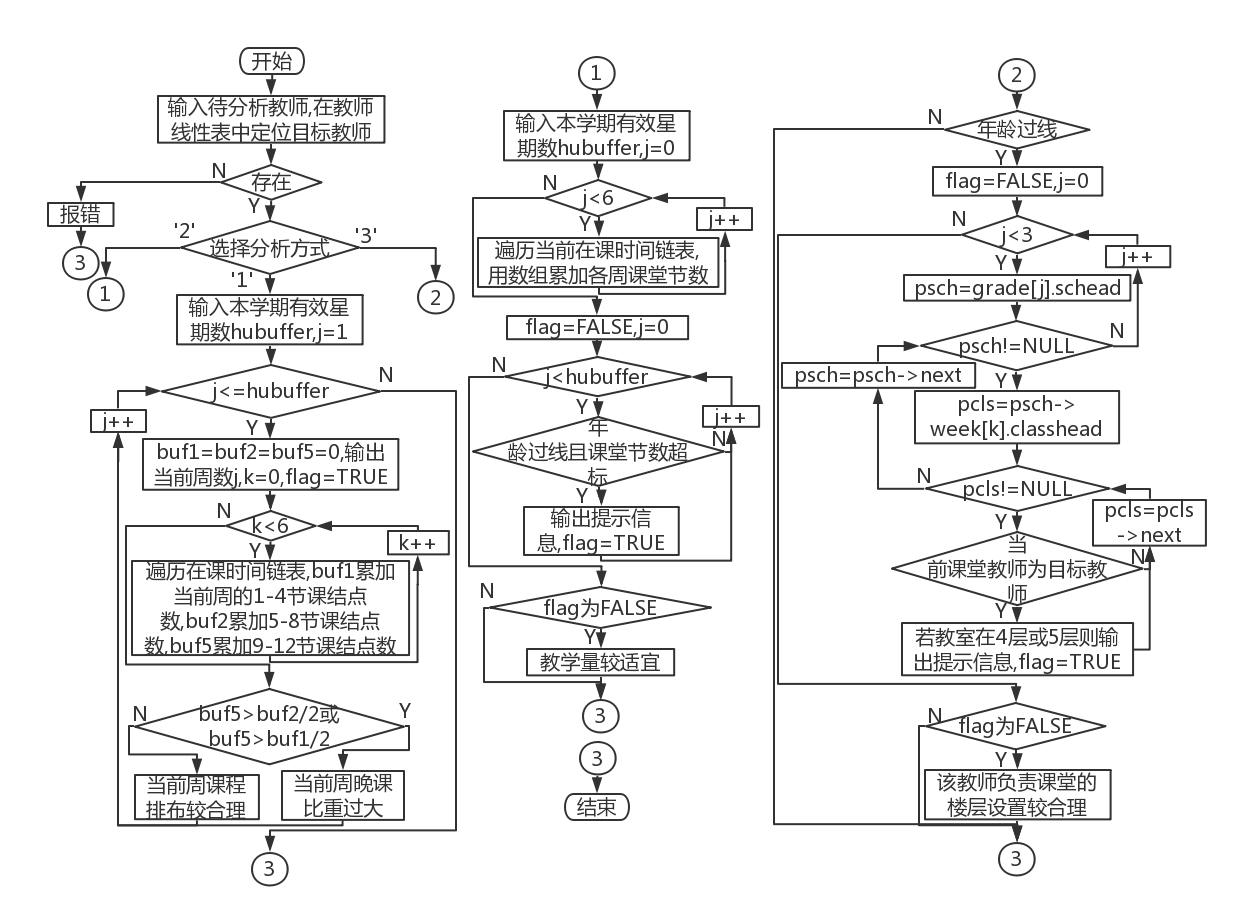


Figure 3-13 □ Flowchart for assessing the rationality of teacher scheduling

3. 2 .4 File access and other functions

1. File saving function FileOut : Automatically generate a text file name based on the system time when the operation is performed , and splicing with the suffix ".txt" to form a complete file name. Write the length of the teacher's linear table . Traverse the linear list of teachers , and for each teacher node , write its name, gender code and age into the file ; traverse the six linked lists of class time , write the length of each linked list, and traverse the current linked list to write each node The week number and knot value of the point . Write the length of the classroom linear table . Traverse the linear list of classrooms , write the number and capacity of each classroom node into the file ; traverse the six linked lists of class time , write the length of each linked list, and traverse the current linked list to write the value of each node Week number and festival value. Write course inverse adjacency header length . For each course node, write the name of the course and the number of prerequisite course nodes. Traverse the list of prerequisite courses and write the name of each prerequisite course. Traverse the three linked list groups of classes divided by grade . For each linked list, write the table length ( that is, the number of classes ) , and traverse the current linked list. For each class node , write its class number, number of people, number of courses it has taken and each course it has taken, number of courses it is taking and each course it is taking

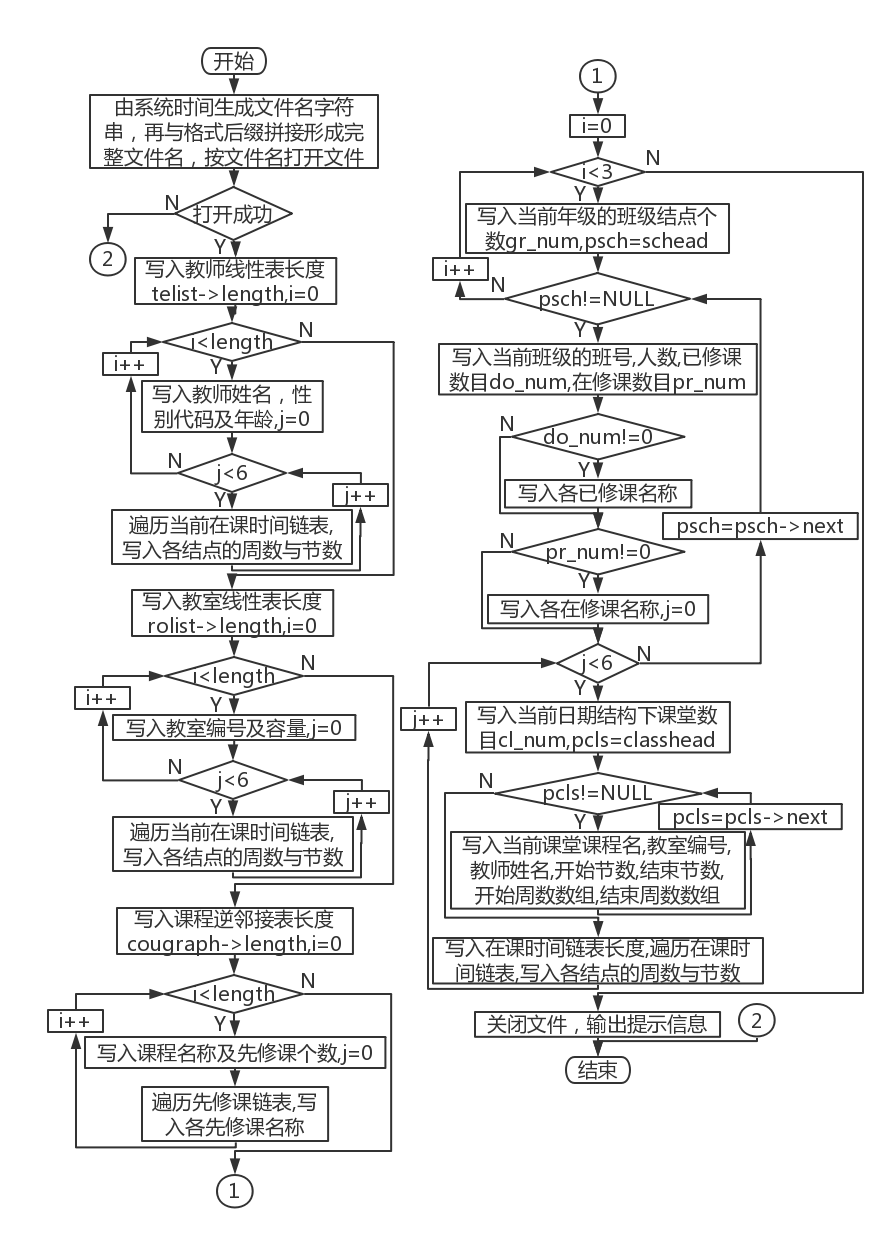


Figure 3-1 4 □File save flow chart

For the six date structures of its subordinates , write the length of the class time list in each structure and traverse the current class time list to write the week number and section value of each node ; write the length of the class list and traverse the current Classroom linked list , write the course name, classroom number, teacher name, start section number, end section number, start week number array and end week number array of each class. Close the file , output prompt information, and clear the screen. The flowchart is shown in Figure 3-14 .

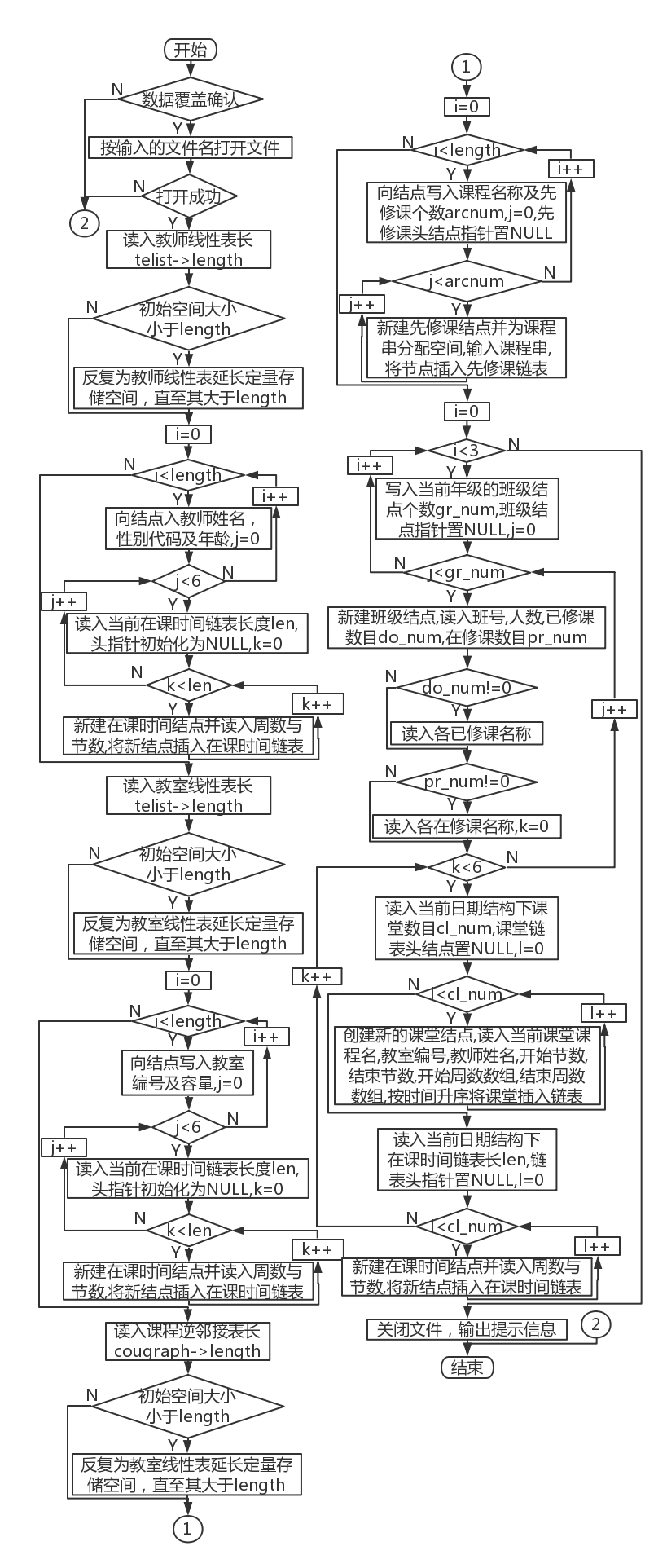


Figure 3-15 □File recovery flow chart

2. File recovery function FileIn : first confirm the operation of data overwriting. Do not overwrite (do not restore) to return to the main menu. When selecting Restore , open the corresponding file under the program directory according to the input file name , and start to restore data to the system . Read into the teacher linear table length . For each teacher node , read in its name, gender code, and age ; for the six class time lists , read in the length of each list, create class time nodes in a loop, and read in the number of weeks for each node and section values , and insert them into the linked list in ascending order of time , and complete after reading in the number of teacher data equal to the length of the teacher linear table . Write the length of the classroom linear table . For each classroom node , read in its serial number and capacity; for the six class time linked lists , read in the length of each linked list, create class time nodes in a loop, and read in the number of weeks and periods of each node Values are inserted into the linked list in ascending order of time , and it is completed after reading in the number of teacher data equal to the length of the linear table in the classroom . Read in course inverse adjacency header length . For each course node, read the name of the course and the number of prerequisite course nodes. Loop to create prerequisite course nodes and allocate space for prerequisite course strings , and read in the names of each prerequisite course. This is done after reading in the course data equal to the length of the course reverse adjacency table . Read the table length of the three class linked lists divided by grades ( that is, the number of classes ) , create class nodes in a loop , read their class number, number of people, number of courses taken, and the number of courses they are taking And each course in progress, for the six date structures under it , read the length of the class time list in each structure , create the class time node in a loop, and read in the week number and section value of each node and press Insert the linked list in ascending order of time ; read in the length of the classroom linked list, loop to create classroom nodes, read in the course name, classroom number, teacher name, start section number, end section number, start week number array and end week number array of each class and Insert into linked list in ascending order of time . After the above work is completed, the reading work of a class node is completed, and it is inserted into the corresponding linked list according to the grade. Close the file , output prompt information, and clear the screen. The flowchart is shown in Figure 3-15 .

3. System initialization function Initial : Create the teacher linear table, classroom linear table, and course inverse adjacency table header structure nodes and initialize the length to 0, and the initial space size is LIST\_INIT\_SIZE (the macro definition is 100). Allocate space with an initial size of LIST\_INIT\_SIZE for the teacher linear table, classroom linear table, and course inverse adjacency list . Set the head pointers of the three class linked lists to NULL, set the length to 0, output prompt information, and clear the screen.

3. 2 .5 Time and Space Complexity Analysis

It should be noted that many of the above functions seem to have a high degree of loop nesting, but the actual characteristics of the data determine that the upper limit of the number of loops is fixed in many cases. For example, there are three grades in the curriculum schedule of this semester , so when traversing the class list constructed based on grades, it loops up to three times; a class has up to six days of classes in a week, so traversing the class structure is based on the date of the week When constructing the linked list group , only six linked lists need to be traversed. However , although the specific issue of course evaluation determines the number of classes in a grade, the number of classes in a class must be limited , but this limitation is not very clear, that is to say, theoretically, as long as the data is legal Infinitely add classes to grades , and infinitely add classes to subclasses. If the evaluation object of class scheduling is expanded to the whole school or even multiple schools , the amount of the above data can indeed increase arbitrarily, so it is not suitable to be treated as a constant level .

Table 3-2 □ Time and Space Complexity Analysis

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Function name | time complexity | space complexity | Function name | time complexity | space complexity |
| 1. DelTimeNode | O( n 2 ) | O(1) | 19. AltRoom | O( n 2 ) | O( 1 ) |
| 2. AddTimeNode | O( n) | O( n ) | 20. Alt Course | O( n 2 ) | O( 1 ) |
| 3. ClassInput | O(1) | O(1) | 21. AltGroup | O(n) | O(1) |
| 4. Add New Room | O(1) | O(1) | 22. AddClass | O( n 2 ) | O( n ) |
| 5. Add New Teacher | O(1) | O(1) | 23. DelClass | O( n 2 ) | O( 1 ) |
| 6. Add New Course | O(1) | O(1) | 24. AltClass | O( n 2 ) | O( n ) |
| 7. SeekRemClass | O(n) | O(1) | 25. SeekTeacher | O(n) | O(1) |
| 8. DelClassNode | O(n) | O(1) | 26. SeekRoom | O(n) | O(1) |
| 9. Time Input | O(n) | O(1) | 27. Seek Course | O(n 3 ) | O(1) |
| 10. Add Teacher | O(n) | O(1) | 28. SeekGroup | O( n 2 ) | O( 1 ) |
| 11.AddRoom | O(n) | O(1) | 29. Seek Class | O( n 2 ) | O( 1 ) |
| 12. AddCourse | O( n 2 ) | O( n ) | 30. Room Usage | O( n 2 ) | O( 1 ) |
| 13.AddGroup | O(n) | O(1) | 31. Class Analysis | O( n 2 ) | O( 1 ) |
| 14. DelTeacher | O(n 3 ) | O(1) | 32. Teacher Analysis | O( n 2 ) | O( 1 ) |
| 15. DelRoom | O(n 3 ) | O(1) | 33. FileOut | O( n 2 ) | O( 1 ) |
| 16. Del Course | O(n 3 ) | O(1) | 34. FileIn | O( n 2 ) | O( n 2 ) |
| 17. DelGroup | O(n) | O(1) | 35. Initial | O(1) | O(1) |
| 18. Alt Teacher | O( n 2 ) | O( 1 ) |  |  |  |

## 4 System Implementation and Testing

4.1 \_ System implementation

This experiment is written with Codeblocks programming software, and compiled and run in W indows system with Codeblocks . The project name is DTCoursework .

4.1.1 \_ \_ Description of header files and predefined constants

1. Header file

#include <stdio.h>

#include <stdlib.h>

#include <windows.h>

#include <wincon.h>

#include <conio.h>

#include <string.h>

#include <io.h>

#include <fcntl.h>

#include <sys\stat.h>

#include <ctype.h>

#include <time.h>

2. Predefined constants

#define SCR\_ROW 50 /\*number of screen rows\*/

#define SCR\_COL 80 /\*Screen columns\*/

#define LIST\_INIT\_SIZE 100

#define LISTINCREMENT 10

4.1.2 \_ \_ Data structure definition

/\*\*

\* Compress and store the linked list node structure during class time

\*/

typedef struct Occupy{

unsigned short week; //class weeks

unsigned short time; //Number of lessons

struct Occupy \*next;

}Occupy;

/\*\*

\* Compress and store the linked list header structure during class time

\*/

typedef struct ochead{

unsigned short len;

Occupy \*head;

}Ochead;

/\*\*

\*Teacher node structure

\*/

typedef struct{

char tname[20]; //teacher name

unsigned short gender, age; //Gender: female is 0, male is 1

Ochead ochead[6]; //Teacher occupancy, three-dimensional: week-week-classroom

}Teacher;

/\*\*

\* Teacher linear header structure

\*/

typedef struct {

Teacher \*elem; //linear table

unsigned short length, listsize; //table length, list size

}TeList;

/\*\*

\* Classroom node structure

\*/

typedef struct{

char code[20]; //classroom number

unsigned short rsize; //teacher capacity

Ochead ochead[6]; //Classroom occupancy, three-dimensional: week-week-classroom

}Room;

/\*\*

\*Classroom linear header structure

\*/

typedef struct {

Room \*elem; //linear table

unsigned short length, listsize; //table length, list size

}RoList;

/\*\*

\*Prerequisite course node structure

\*/

typedef struct CouArc{

char \*prename; //prerequisite course name

struct CouArc \*nextarc; //pointer to the next arc

}CouArc;

/\*\*

\* course inverse adjacency list arc node structure

\*/

typedef struct {

char name[60]; //course name

unsigned short arcnum; // Number of prerequisite courses

CouArc \*archhead; // pointer to the head of the course

}CouHead;

/\*\*

\* course inverse adjacency header structure

\*/

typedef struct { //Course inverse adjacency table header structure

CouHead \*coulist; // reverse adjacency list

unsigned short length, listsize; //total number of courses

}CouGraph;

/\*\*

\*classroom node structure

\*/

typedef struct Class{

char coursename[60], roomcode[20], tname[20]; //Class name, classroom number, teacher name

unsigned short cou\_st, cou\_ed, wk\_st[5], wk\_ed[5]; //Start section number, end section number, start week number, continuous week number

struct Class \*next;

}Class;

/\*\*

\* class node structure

\*/

typedef struct Schedule{

char classname[20]; //class number

unsigned short do\_num, pr\_num, clsize; //Number of completed/current courses

char done[60][60], progress[15][60]; //Courses completed/in progress

struct weekday{

Class \*classhead; //class list head pointer

Ochead ochead; //head structure of class time chain list

unsigned short cl\_num; //class number

}week[6]; //There are no classes on Sunday, only six days a week

struct Schedule \*next;

}Schedule;

/\*\*

\* class list header structure

\*/

typedef struct {

unsigned short gr\_num; //class number

Schedule \*schead; //Schedule head pointer

}Grade;

4.1.3 \_ \_ function declaration

void InitInterface(void); /\*system interface initialization\*/

void ClearScreen(void); /\*clear screen\*/

void ShowMenu(void); /\* show menu bar \*/

void PopMenu(int num); /\*Display drop-down menu\*/

void PopPrompt(int num); /\*Display the pop-up window\*/

void PopUp(SMALL\_RECT \*, WORD, LABEL\_BUNDLE \*, HOT\_AREA \*); /\* pop-up window screen information maintenance \*/

void PopOff(void); /\*Close the top-level pop-up window\*/

void DrawBox(SMALL\_RECT \*parea); /\*draw border\*/

void LocSubMenu(int num, SMALL\_RECT \*parea); /\*Main menu drop-down menu location\*/

void ShowState(void); /\*Display status bar\*/

void TagMainMenu(int num); /\*mark the selected main menu item\*/

void TagSubMenu(int num); /\*mark the selected submenu item\*/

int DealConInput(HOT\_AREA \*phot\_area, int \*pihot\_num); /\*Console input processing\*/

void SetHotPoint(HOT\_AREA \*phot\_area, int hot\_num); /\*Set hot area\*/

BOOL ShowModule(char \*\*pString, int n); /\*popup information window\*/

BOOL Message(char \*\*); /\* pop-up window \*/

BOOL LoadData(void); /\*data loading\*/

void RunSys(void); /\*Selection and operation of system function modules\*/

BOOL ExeFunction(int main\_menu\_num, int sub\_menu\_num); /\* Function module call \*/

void CloseSys(void); /\* exit the system\*/

void ClearMemory(void); /\*clear memory data\*/

BOOL Initial(void); /\*data structure initialization\*/

BOOL AddTeacher(void); /\*insert teacher\*/

BOOL AddRoom(void); /\*insert classroom\*/

BOOL AddCourse(void); /\*insert course\*/

BOOL AddGroup(void); /\*insert class\*/

BOOL DelTeacher(void); /\*Delete teacher\*/

BOOL DelRoom(void); /\*Delete room\*/

BOOL DelCourse(void); /\*Delete course\*/

BOOL DelGroup(void); /\*Delete class\*/

BOOL AltTeacher(void); /\*modify teacher\*/

BOOL AltRoom(void); /\*Modify classroom\*/

BOOL AltCourse(void); /\*modify course\*/

BOOL AltGroup(void); /\*Modify class\*/

BOOL SeekTeacher(void); /\*find teacher\*/

BOOL SeekRoom(void); /\*find classroom\*/

BOOL SeekCourse(void); /\*find course\*/

BOOL SeekGroup(void); /\*find class\*/

BOOL MaintainTeacherInfo(void); /\*Maintain teacher information\*/

BOOL MaintainRoomInfo(void); /\*Maintain classroom information\*/

BOOL MaintainGroupInfo(void); /\*Maintain class information\*/

BOOL MaintainCourseInfo(void); /\*maintain course information\*/

BOOL MaintainClassInfo(void); /\*Maintain class information\*/

BOOL AddClass(void); /\*insert class\*/

BOOL DelClass(void); /\*Delete class\*/

BOOL AltClass(void); /\*modify class\*/

BOOL SeekClass(void); /\*find class\*/

BOOL RoomUsage(void); /\*room usage\*/

BOOL TeacherAnalysis(void); /\*Analysis of classroom scheduling\*/

BOOL ClassAnalysis(void); /\*class schedule analysis\*/

BOOL FileOut(void); /\*Array backup\*/

BOOL FileIn(void); /\*Array recovery\*/

BOOL ExitSys(void); /\*exit the system\*/

BOOL HelpTopic(void); /\*Help topic\*/

BOOL AboutDorm(void); /\*About the system\*/

BOOL DelTimeNode(unsigned short wk\_st[], unsigned short wk\_ed[], unsigned short cou\_st, unsigned short cou\_ed, Ochead \*ochead); /\*delete class time nodes\*/

BOOL AddTimeNode(unsigned short wk\_st[], unsigned short wk\_ed[], unsigned short cou\_st, unsigned short cou\_ed, Ochead \*ochead); /\*Add the class time node\*/

Schedule \*ClassInput(char cbuf[]); /\*class name input\*/

BOOL TimeInput(unsigned short \*cou\_st, unsigned short \*cou\_ed, unsigned short wk\_st[], unsigned short wk\_ed[]); /\*class name input\*/

BOOL AddNewRoom(char cbuf[]); /\*insert new room\*/

BOOL AddNewTeacher(char cbuf[]); /\*insert new teacher\*/

BOOL AddNewCourse(char cbuf[]); /\*insert a new course\*/

BOOL SeekRemClass(Schedule \*ps, char cbuf[]); /\*Find remaining classes\*/

BOOL DelClassNode(Class \*pc, Schedule \*ps, unsigned short j); /\*Delete class node\*/

The call relationship of each function function has been given in Figure 2-5 , and each function function calls ClearScreen to reset the interface when returning to the main menu . In addition , some functional functions also call the basic module functions in Section 3.2.1 , and the calling relationship is shown in Figure 4-1 .

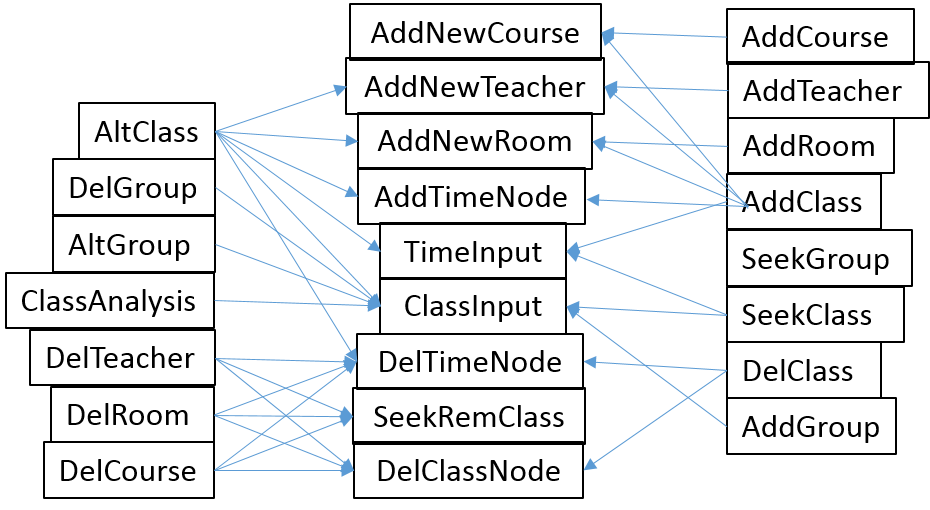


Figure 4-1 □ Function call relationship diagram

4.2 \_ System test

4.2.1 Test methods and principles

The correctness of the algorithm requires the carefully selected and difficult data to obtain the results that meet the specification requirements . Therefore , in addition to the correct data during the test , various input errors must be intentionally made to test the algorithm in the face of edge conditions. Can it return the correct value without crashing . At the same time , when selecting data, we should try our best to achieve the effect of " moving the whole body with one hair " , that is, when modifying or deleting data, let as many other data as possible be affected by it and change to the corresponding correct result , and at the same time, use multiple methods to check The correctness of the system operation after modifying the data .

4.2.2 \_ Test Plan

Table 4-1 □Outline of test plan

|  |  |  |  |
| --- | --- | --- | --- |
| function to be tested | test sequence | test input | expected output |
| 1. Add class | 1 | Course Name : Database System Architecture  Classroom : S209 , West 12th Floor  Teacher : Pan Peng  5 -16 weeks Monday 3 -4 classes  There are 2 classes in the classroom:  Computer 1501; Computer 1502 | Imported new courses , new classrooms and new teachers; created two new classes and filled in the number information |
| 2. Modify the course name | 2. | Formerly known as : Database System Architecture  New name : Database System Fundamentals | Successfully modified |
| 3. Find classes by course + time + classroom | 3 | Course : Fundamentals of Database Systems  Classroom : S209 , West 12th Floor  Class time : 5 -16 weeks, 3 -4 lessons on Monday | Classes : Computer 1501, Computer 1502; all teachers are Pan Peng |
| 4. Find free classrooms for a certain period of time | 4,9,10 \_ | 4 & 9 : 3 -4 lessons on Monday of the 7th week ( during the class time of the above courses )  10: Week 16, Monday 7-8 ( within the new class time period ) | 4: S209 on the west 12th floor of the occupied classroom does not appear;  the new classroom N511 on the west 12th floor appeared ;  1 0: N511 on the west 12th floor of the occupied classroom does not appear |
| 5. Find classes by teacher | 5,11,28 | 5 & 11 : Qin Leihua  28: Yan Shuai | All classrooms the teacher is in charge of ;  11 : Added " database system structure"  12 : Display the class information after modifying the time and classroom |
| 6. Modify the classroom used by the classroom | 6,27 | 6: Course : Database System Architecture  Class date: Monday  Classes attended : Computer 1501; Computer 1502  New Classroom : N511, West 12th Floor  7 : Course : Database System Architecture  Class date: Monday  Classes : Computer 1701; Computer 1702; Computer 1703; Computer 1704 ; Computer 1705  New Classroom : N511, West 12th Floor | 6 & 27: Modified successfully |
| 7. Modify the classroom teacher | 7. | Course : Database System Architecture  Class date: Monday  Classes attended : Computer 1501; Computer 1502  New teacher : Qin Leihua | Successfully modified |
| 8. Modify the class time of the class | 8 | Course : Database System Architecture  Class date: Monday  Classes attended : Computer 1501; Computer 1502  new sections: 7 -8 sections  New week numbers : 1-3 , 5-7 , 9-9 , 11-13 , 15-17 weeks | Successfully modified |
| 9. Find classes by class | 12 | COMPUTER 1502 | 12 : The information of other classes in this class and the new information after modification of the class " Database System Structure " |
| 10. Add prerequisites for courses | 17,19 | 17 & 19 : Target Course : Comprehensive English (2)  Target Course : Comprehensive English (1) | 17: There is no comprehensive English ( 1 ) course in the course list , and the addition fails  19: Successfully added |
| 11. Analyze the rationality of class teaching | 16,23 | 16&23 : Target class : Computer 1704 | 16: Calculus ( 1 ) (Part 2) and College Physics ( 1 ) should not be taken this semester , because there is no prerequisite course Calculus ( 1 ) (Part 1) in the list of courses taken in this class  23: The above two courses become available , but Comprehensive English ( 2 ) becomes unavailable |
| 12. Insert new class | 18 | New Course : Comprehensive English ( 1 )  Number of prerequisite courses : 0 | Inserted successfully |
| 13. Add a new taken class to the class | 20 | Target class : COMPUTER 1704  Add Course : Calculus ( 1 ) (Part 1) | Added successfully |
| 13. Fuzzy lookup courses | 15,21 \_ | 15: Course substring : University  21: Course Substring : Calculus ( 1 ) | 15: Course : University Physics ( 1 )  Prerequisite : Calculus ( 1 ) (Part 1)  Follow-up courses: none  21: Course : Calculus ( 1 ) (Part 1)  Prerequisites : none  Follow-up class : Calculus ( 1 ) (Part 2)  College Physics ( 1 )  Signal and Linear Systems  Course : Calculus ( 1 ) (2)  Prerequisite : Calculus ( 1 ) (Part 1)  Follow-up class : Calculus ( 1 ) (Part 2)  College Physics ( 1 )  Signal and Linear Systems |
| 14. Find class by class number | 22,43,49 \_ | 22: Computer 1704  43&49 : Computer 1711 | 2 2 : Display the number of people and courses in progress , and add the course "Calculus ( 1 ) (Part 1)" that has been completed  43: Display the revised number of classes and courses completed and in progress  49: Find failed |
| 15. Find teacher information by name | twenty four | Name: Shuai Yan | Gender : female  Age : 0 |
| 16. Modify the gender of the teacher | 25 | Name : Shuai Yan | Change gender to male |
| 17. Modify the teacher's age | 26,3 1 | 26: Name : Yan Shuai  new age : 57  31 : Name : Yan Shuai  new age : 61 | 26&31 : The modification is successful |
| 18. Assess the rationality of teachers' teaching tasks distributed in the morning and evening in a day | 29 | Name : Shuai Yan | The other weeks are reasonable ( no classes ) , the 15th week has no classes during the day but evening classes, the proportion of evening classes is too large |
| 19. Assess the rationality of the teacher 's class teacher arrangement | 30,32 \_ | 30 & 32 : Name : Yan Shuai | 30: The teacher has no special floor requirements ( evaluation criteria are given in 2.3.3 )  32: The floor of classroom N511 where the ideological and political courses are located is too high |
| 20. Assess the rationality of using classes in the classroom ( Energy Efficiency Ratio ) | 33,34,36,39 \_ \_ \_ | 33: Classroom : West Playground  34:  Classroom: S409 , West 12th Floor  Course : Assembly Language Programming  Class time : 3-8 weeks , 3-4 lessons on Monday | 33: The classroom was not found  The number of people in class 1601 is 26; the number of people in class 1602 is 27; the number of people in class 1603 is 26; the number of people in school handover is 38  34 : The capacity of teachers is 150, the seats are in short supply , and a larger classroom needs to be replaced  36: The capacity of teachers is 230, and the classroom is suitable  39: The capacity of teachers is 300, and there are too many vacant seats. Consider merging classrooms or changing to smaller classrooms |
| 21. Modify classroom capacity | 35,38 , | Classroom : S409 , West 12th Floor  35: Capacity : 230  38 : Capacity : 300 | Successfully modified |
| 22. Modify the classroom number | 37 | Classroom : S409 , West 12th Floor  New number : West 12th Floor N409 | Successfully modified |
| 23. Find the classes that exist in the target time period | 40 | Week 11 -15 Tuesday Session 3 -4 \_ | Course : Verilog language \_  Classroom : Room 803 , South 1st Floor  Teacher : Lu Ping  Classes : COMP 1601, COMP 1602, COMP 1603  Course : Fundamentals of Logic and Computer Design  Classroom : N203 , West 12th Floor  Teacher : Qin Leihua  Classes : Information Security 1601, Information Security 1602, Information Security 1603  Course : Discrete Mathematics (1)  Classroom : N 411 , West 12th Floor  Teacher : Zhang Aihua  Classes : COMP 1701, COMP 1702, COMP 1703 |
| 24. Modify the class number of the class | 41 | Waiting for this class : COMPUTER 1701  New class number : computer 1711 | Successfully modified |
| 25. Modify class size | 42 | Waiting for this class : COMPUTER 1711  Number of newcomers : 32 | Successfully modified |
| 26. Modify the name of the teacher | 44, 45 | 44: Teacher to be reformed : Zhang Aihua  New name : Qin Leihua  45: Teacher to be reformed : Zhang Aihua  New name : Wang Lijun | 44: New teacher already exists , modification failed  45: Modify successfully |
| 27. Delete teacher | 46 | Teacher Name : Wang Lijun | successfully deleted |
| 28. Delete classroom | 47 | Classroom No .: 505, West Fifth Floor | successfully deleted |
| 29. Delete class | 48 | Class number : Computer 1711 | successfully deleted |
| 30. Retrieve classrooms by capacity | 50 | Capacity range : 100 -150 | Show all classrooms with the required capacity and the specific capacity |
| 31. Retrieve classes by number of people | 51 | Number of people range : 25 -26 | Show the class numbers of all classes whose number of people meet the requirements, the specific number of people , the list of courses taken and the list of courses in progress |
| 32. Retrieve teachers by age | 52 | Age range : 34 -40 | Display the name , age and gender of all teachers who meet age requirements |
| 33. Retrieve teachers by gender | 53 | Gender : female | Display the names and ages of teachers who match the gender requirements |
| 34. Calculate classroom utilization for target time interval | 54 | Time interval : 2 -3 weeks Monday to Tuesday Sections 4-5 \_ \_ | Faculty utilization is 0.072368 |

4.2.3 \_ \_ test

First, describe the commonly used software testing methods, and describe the testing process after selecting several main functional modules (you can master the number yourself, the key is to reflect your level of some modules), (1) first clarify the functions and design goals of the modules. (2) To analyze and describe how to select test data, a complete test program is required. (3) Running results (screenshots are available at this time). (4) Analyze the operation results and confirm that the program meets the design goals of the module.

1. Insert new class and find by course +time+ room .

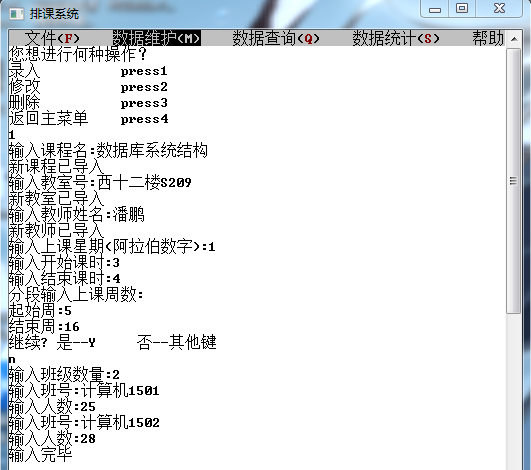
****

Figure 4- 2 □Insert a new class

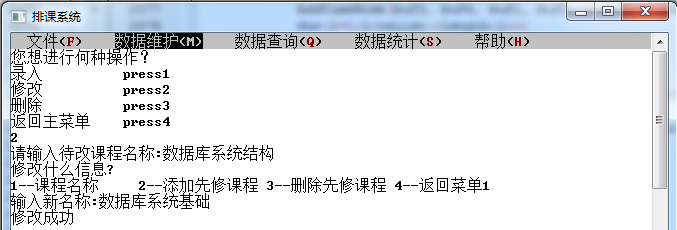


Figure 4- 3 □ Modify course name

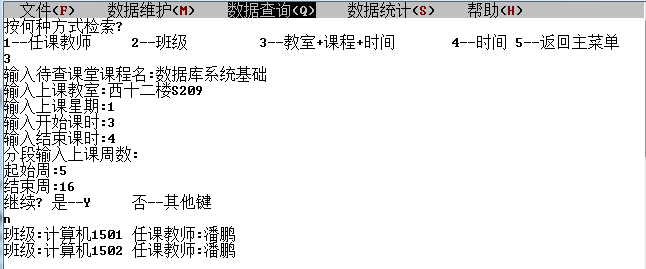


Figure 4- 4 □ Search for a new classroom by course + time + classroom

1. Find available classrooms for that time period .

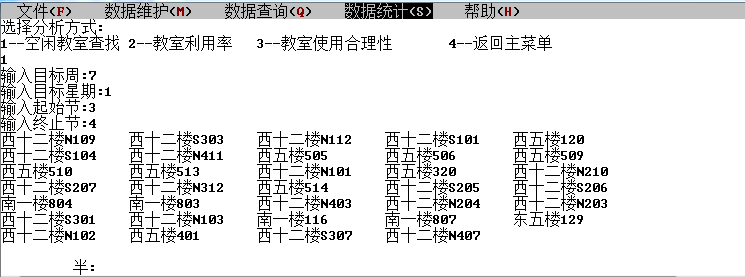


Figure 4-5 □Finding free classrooms

1. by teacher .

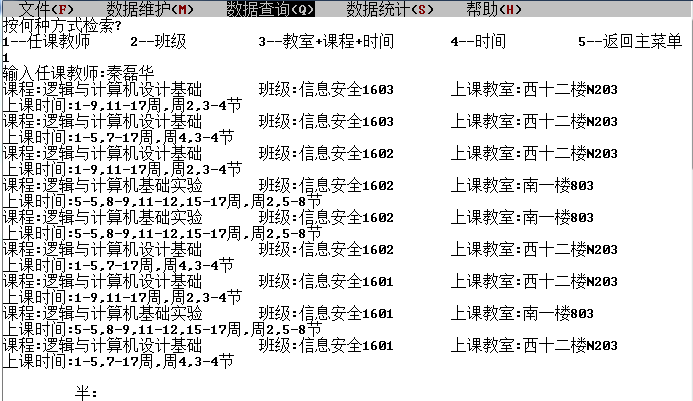


Figure 4-6 □Find classes by teacher

1. changing the course name back to " Database System Structure " , modify the classroom, teacher and time of this class.

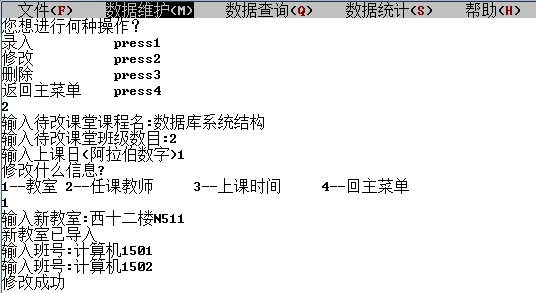


Figure 4-7 □Modify the classroom used in the classroom

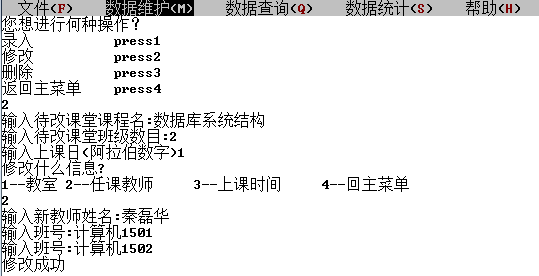


Figure 4-8 □Modify classroom teacher

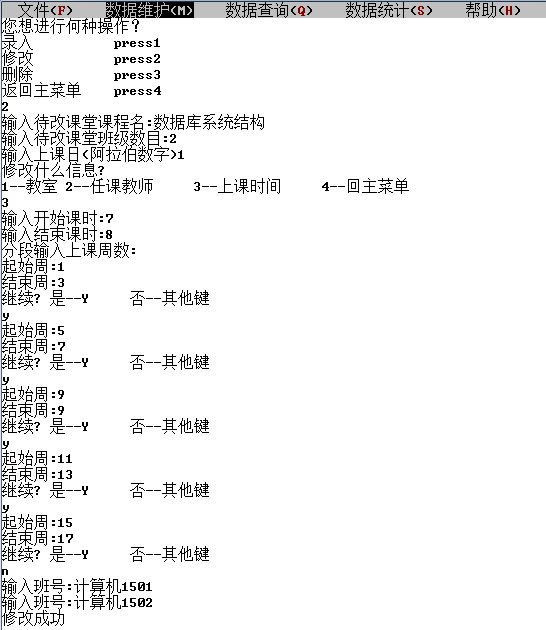


Figure 4-9 □Modify class time

1. Check the modification effect : Enter the relevant information according to the time period in step 3 , and you can see that the S209 on the West 12th Floor occupied before the modification time and the new classroom N511 on the West 12th Floor in the system entered when modifying the classroom appear ; In a new time period ( included in the new class time) , it is observed that N511 on the west 12th floor disappears , which proves that the time and classrooms have been modified successfully. Then search for the class by teacher, and find the target class when searching for the modified teacher , which proves that the teacher has successfully modified.



Figure 4- 10 □ Searching for vacant classrooms according to the time period before modifying the class time



Figure 4- 11 □ Search for free classrooms according to the time period after the class time is modified

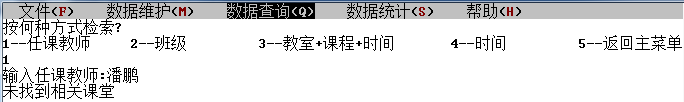


Figure 4- 12 □Search classes by original teacher



Figure 4- 13 □Search classes by new teachers

1. by class.

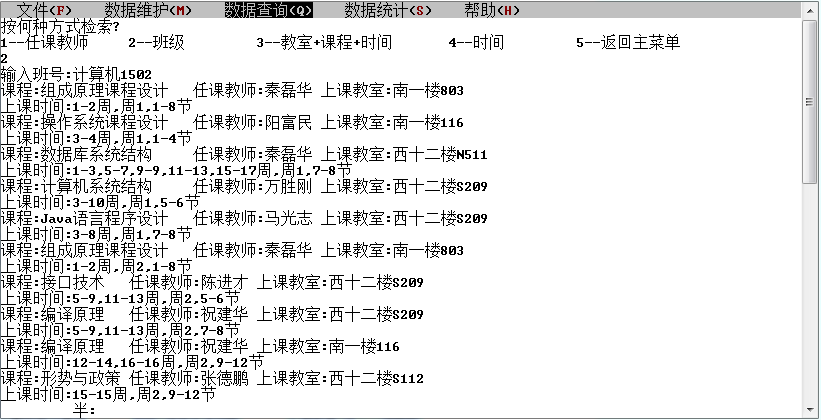


Figure 4- 14 □Search for classes by class

1. prerequisites for course Calculus ( I ) (below) and check Prerequisites for College Physics ( I ) .

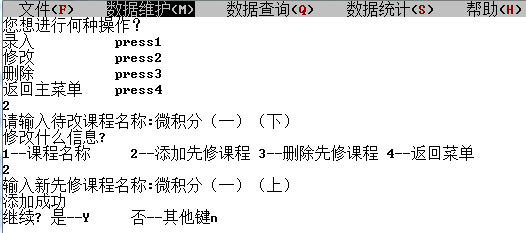


Figure 4- 15 □ Adding the pre-requisite course Calculus ( 1 ) (top) to the course Calculus ( 1 ) (bottom)



Figure 4- 16 □Looking up Classroom Physics ( 1 )

1. Analyze the rationality of taking courses in the 17th grade class.

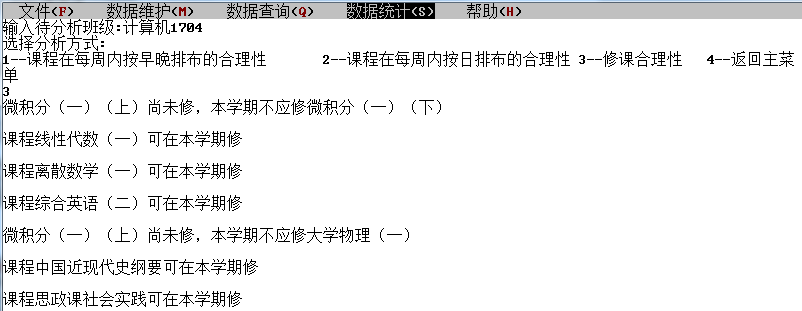


Figure 4- 17 □Analyzing the rationality of class 1704

1. Add a new course Integrated English ( 1 ) and add it as a prerequisite course for Comprehensive English (2) , and add the course Calculus ( 1 ) (Part 1) for Computer 1704 class .

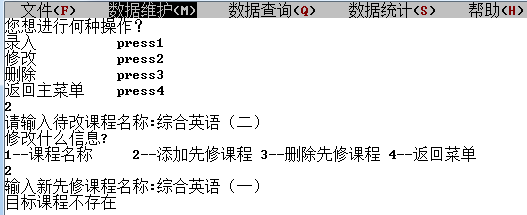


Figure 4-18 □An error is reported when adding a prerequisite course that is not in the course list to a course

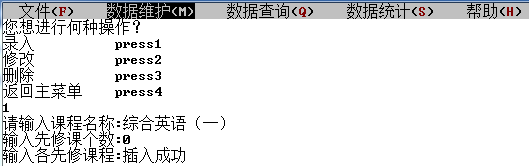


Figure 4- 19 □Add new course

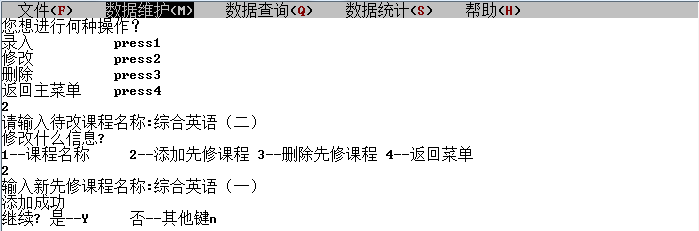


Figure 4- 20 □ Adding a new prerequisite course to the course

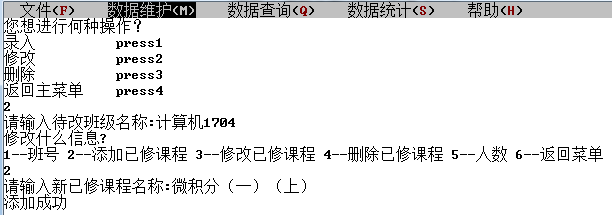


Figure 4- 21 □ Adding a new course to the class

1. After the above operations, check the course situation of the target class and re-evaluate the rationality of the class . It is found that this class should not take comprehensive English ( 1 ) before taking comprehensive English ( 2 ) this semester. Taking Comprehensive English ( II ) ; at the same time, Calculus ( I ) (Part I) has become a course it has taken, so Calculus ( I ) (Part II) and College Physics ( I ) as its successor courses have become available.

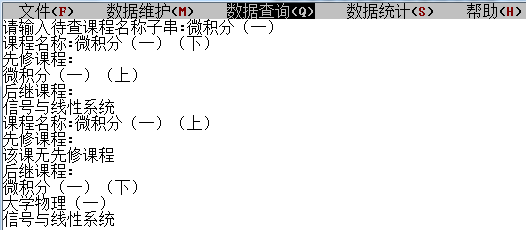


Figure 4- 22 □ Fuzzy search for course information

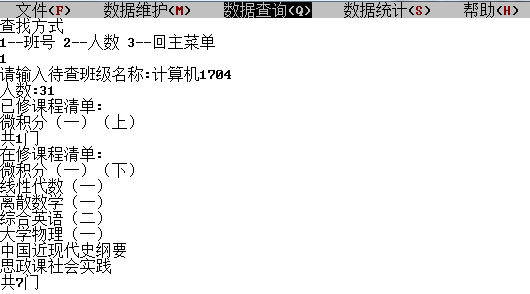


Figure 4- 23 □ Search class information by class number

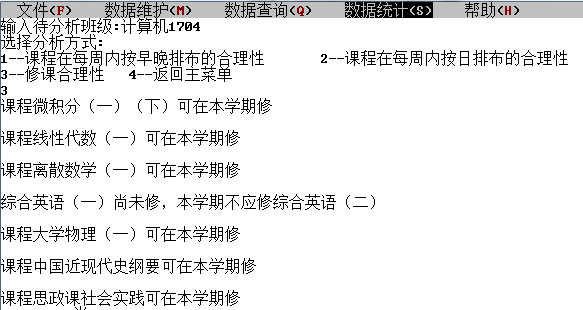


Figure 4- 24 □ Re-evaluate the rationality of class teaching

1. Modify the teacher's information , and evaluate the rationality of the teacher's teaching person 's time arrangement in a day and the floor arrangement of the classroom. See Section 3.2.3 for evaluation criteria .

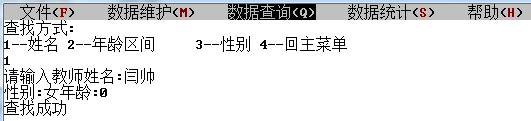


Figure 4- 25 □Find teacher information

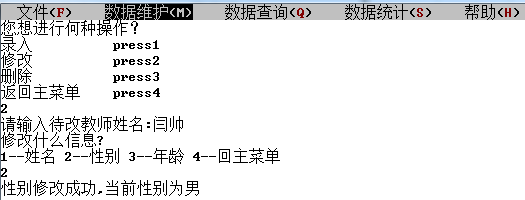


Figure 4- 26 □ Modify the gender of the teacher

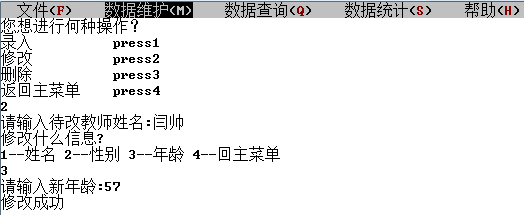


Figure 4- 27 □ Modify teacher's age

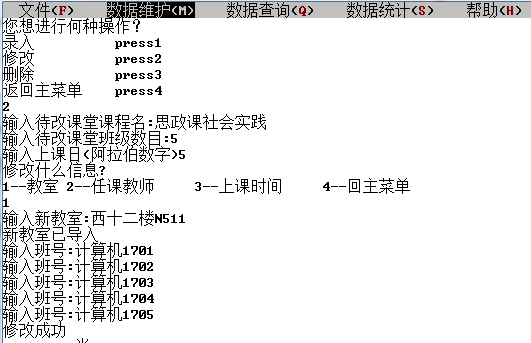


Figure 4- 28 □ Modify the classroom where the classroom is located

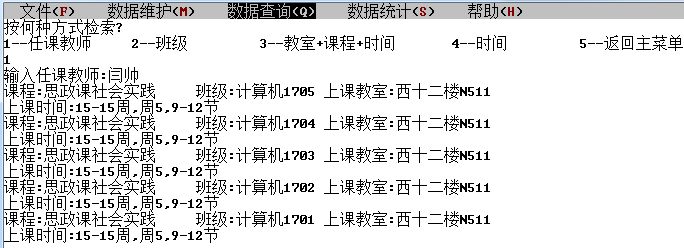


Figure 4- 29 □ Search classes by teacher

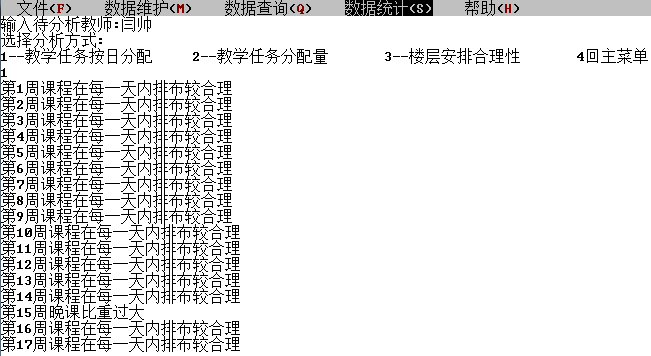


Figure 4-30 □Assess the rationality of the teacher 's teaching tasks distributed according to morning and evening in a day

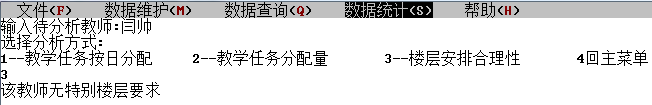
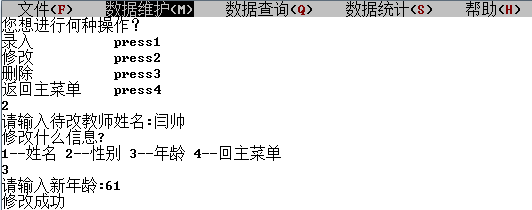


Figure 4- 31 □Assessing the rationality of the teacher 's classroom arrangement



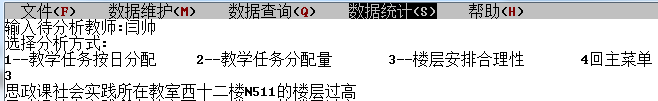
Figure 4- 32 □ Further modify teacher information

Figure 4- 33 □Reassess the rationality of the teacher 's classroom arrangement

1. Repeatedly modify the capacity of the target teacher and modify the classroom number, observe the adaptability change between the designated classroom and the target classroom and the effectiveness of the modification of the teacher number. See Section 3.2.3 for the evaluation criteria .

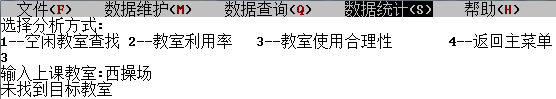


Figure 4- 33 □Find classroom errors

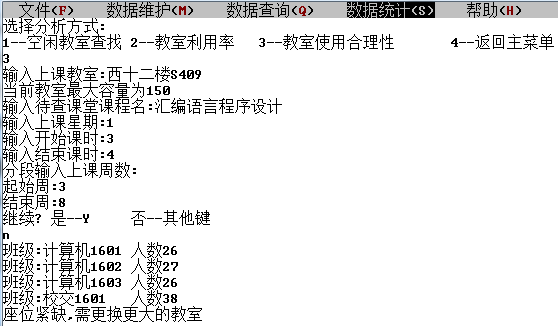


Figure 4- 34 □Assessing the adaptability of the classroom to the classroom

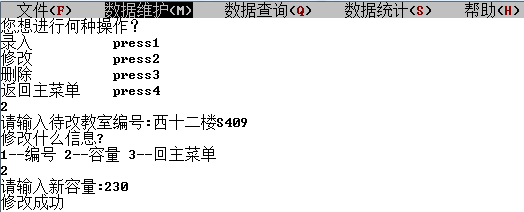


Figure 4- 3 5 □ Modify classroom capacity

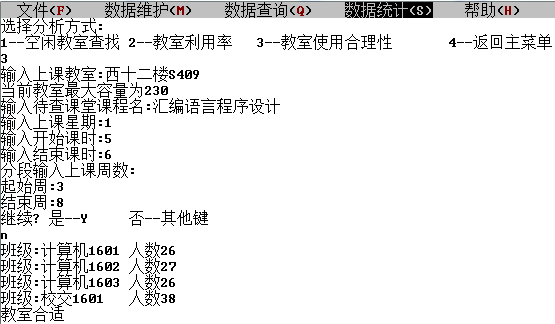


Figure 4- 3 6 □Assess classroom-to-class fit

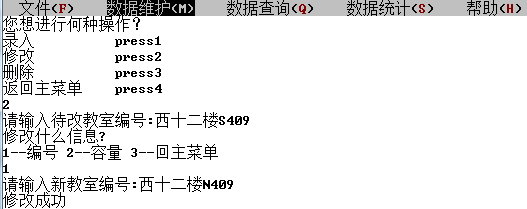


Figure 4- 3 7 □Modify classroom number

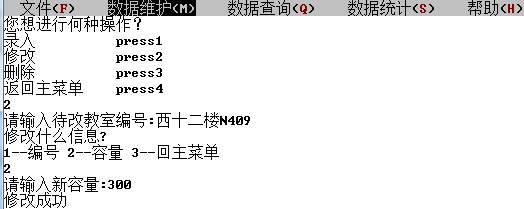


Figure 4- 3 8 □ Modify classroom capacity

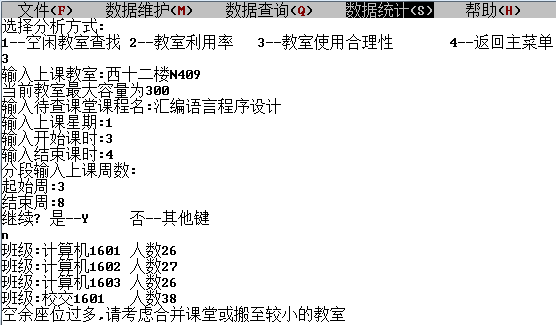


Figure 4- 3 9 □Assess classroom-to-class fit

1. Modify the class number and number of people in the class, and check whether the new information of the target class is correctly displayed or not through multiple search and analysis functions to check the accuracy of the modification .

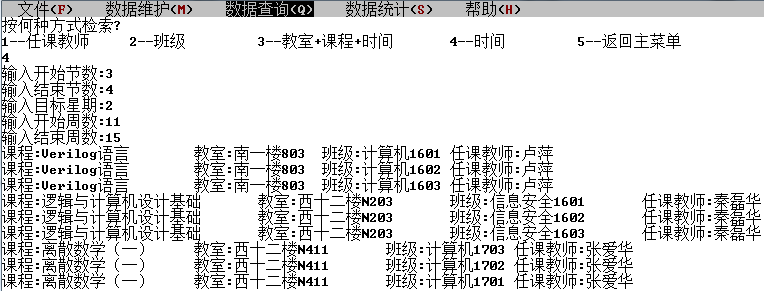


Figure 4- 40 □ Find the classes that exist in the target time period

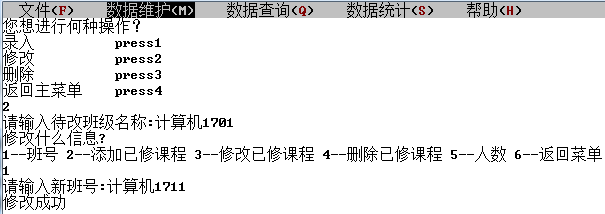


Figure 4- 41 □ Modify the class number of the class

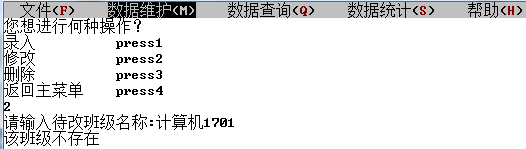


Figure 4- 42 □Failed to find the original class

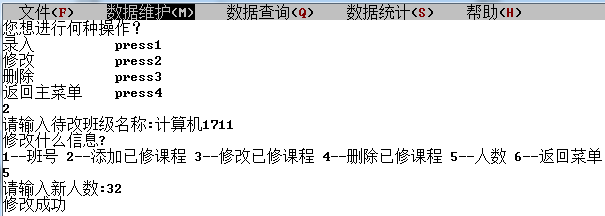


Figure 4- 43 □ Modify the number of people in the class



Figure 4- 44 □ Find class information by new class number

1. Modify the teacher's name, and check the validity of the modification by comparing the information displayed before and after the same search method.

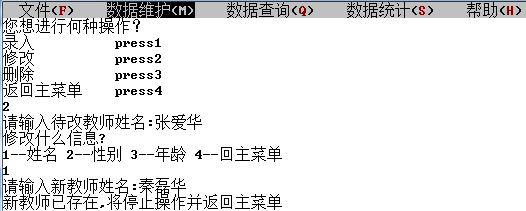


Figure 4- 45 □Failed to modify teacher name

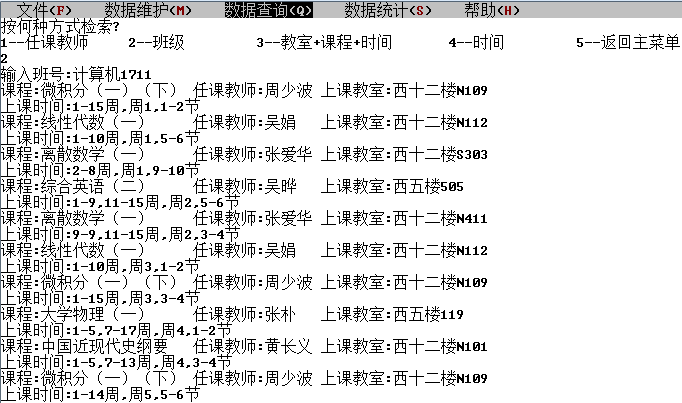


Figure 4- 46 □Find the class under the class

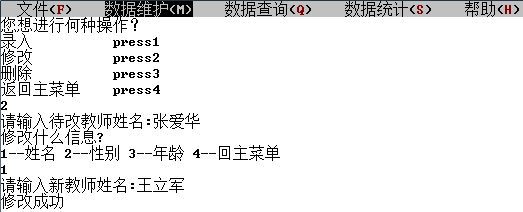


Figure 4- 47 □Modify teacher name

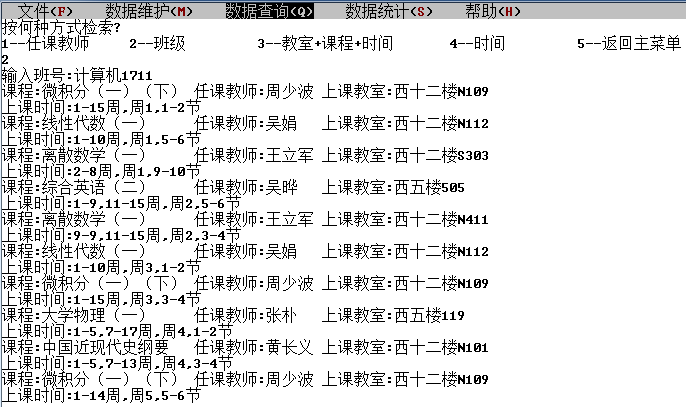


Figure 4- 48 □Find classes of the same class, focusing on the modified teacher

1. Use the information found in the previous step to delete one of the classrooms and teachers, and check the validity of the deletion by comparing the information displayed before and after the same search method .



Figure 4- 49 □ Delete teacher



Figure 4- 50 □ Delete classroom

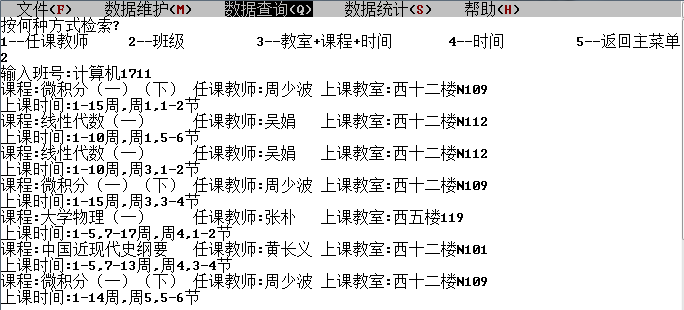


Figure 4-51 □Find the classrooms of the same class, focus on the classroom corresponding to the deleted teacher and classroom

1. Delete the above class , and then search for the class and search for the class by classroom + course + time , but the information of the target class is not found, which proves that the deletion is successful.

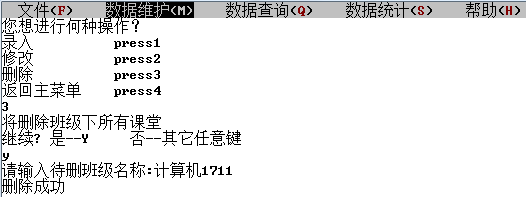


Figure 4- 52 □ Delete class

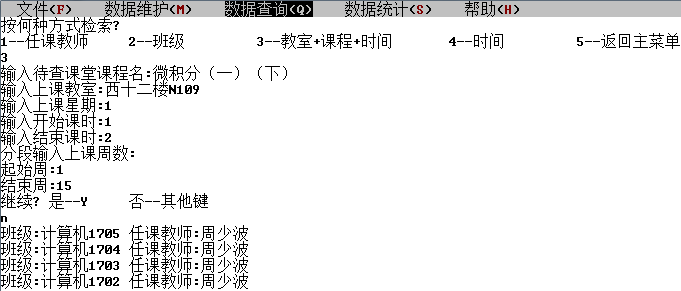


Figure 4- 53 □ Search by a class under the deleted class , but the original class is not found

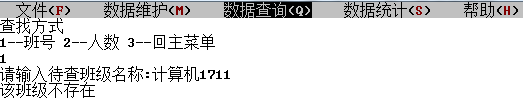


Figure 4- 54 □Find class failed

1. Retrieve classrooms by classroom capacity.

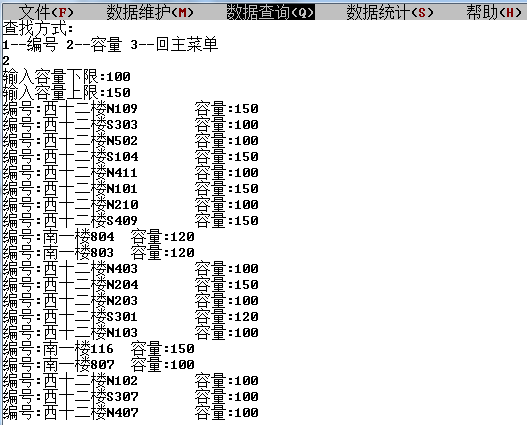


Figure 4- 55 □Retrieve classrooms by capacity

1. Retrieve classes by number of people.

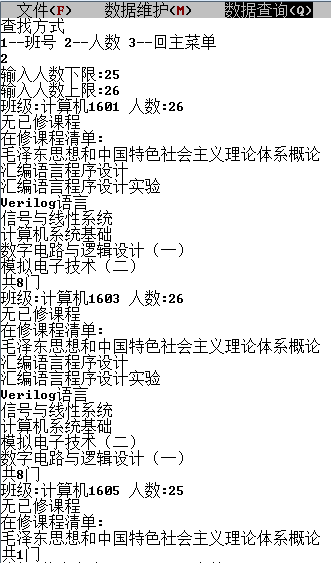
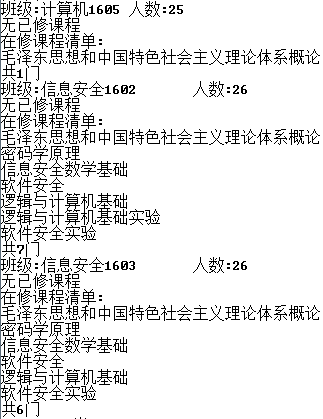
 

Figure 4- 56 □Retrieve classes by number of people

1. Retrieve teachers by age.

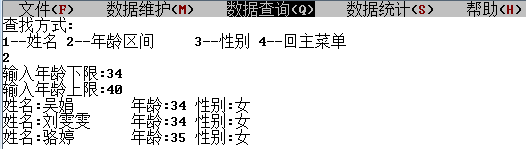


Figure 4- 57 □Retrieve teachers by age

1. Retrieve teachers by gender.

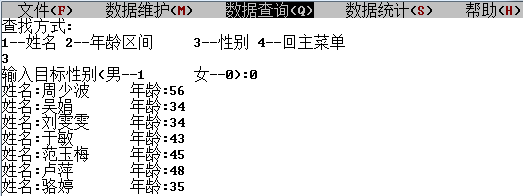


Figure 4- 58 □ Retrieve teachers by gender

1. Analyze teacher utilization for a specified time period .

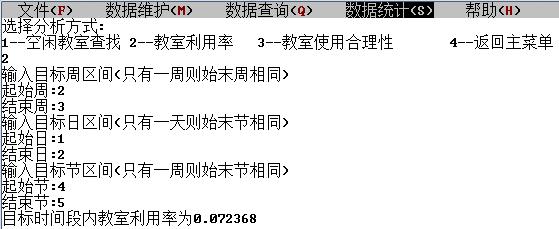


Figure 4- 59 □Analysis of teacher utilization in the target time period

1. Insert class.

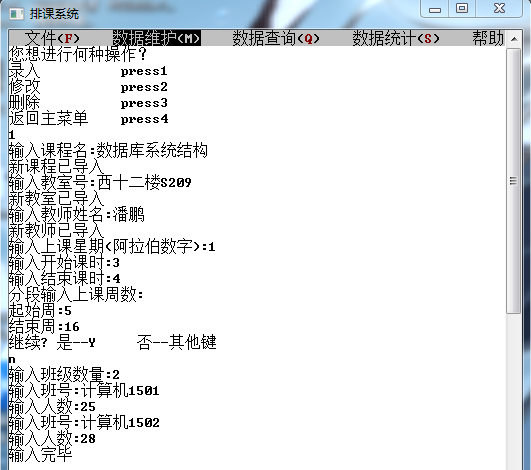


Figure 4- 60 □Insert class

## 5 Summary and Outlook

5 .1 Full text summary

To make a summary of my work, the main tasks are as follows:

(1) Analyze the design topic, design content and design requirements, consult relevant materials in the library and on the Internet, formulate a design plan, and prepare for the design work .

(2) Carry out design work according to design requirements, design appropriate data structures and algorithms, collect and process relevant data, and implement programming .

(3) Test the system according to the design requirements, debug program bugs , optimize and perfect the system.

(4) Write a curriculum design report.

5.2 \_ Job Prospects

In the future research, work will be carried out around the following aspects. . . . . . . .

(1) Try to use what you have learned in the data structure as much as possible in your studies . When encountering some data , think about what kind of structure to use to store and manage them .

(2) Practice more , bring the data structure to the professional courses to be learned , and constantly consolidate and improve .

(3) Read more algorithms to improve your programming efficiency.

(4) The designed system function is relatively single. Therefore, the system functions should be improved and expanded so that the system The system can provide users with more and better services.

## 6 body meeting

The first difficulty I encountered this time was the problem of data organization structure . Due to the foundation laid by the experimental class last semester , I am relatively familiar with the basic operations of various data structures even after the exam week and half a vacation; it is really difficult for me to solve the problem of data relationship for a long time , especially Classroom and classroom, teacher, and class issues associated with it. In the process of entering the general debugging for the first time , I was lazy to use high-dimensional arrays to save time information, and found that the storage of high-dimensional arrays could not be forcibly stored in the course scheduling problem, so I added another afternoon and one night . The high-order matrix is all changed to a compressed storage binary table.

File input and output has always been a shortcoming for me , but I want to simplify the operation of file input and output as much as possible. Combined with the data structure, I chose binary storage at the beginning , and I need to use two functions fread and fwrite. The way of writing should also be reviewed. This shows that people's memory ability is limited after all . If you want to master and maintain efficient programming ability, you need a programmer to pile up a lot of code operations on a daily basis. On the one hand, this is to keep memorizing some hard grammar knowledge , and on the other hand, it also helps to maintain the programming feel . At the same time, as a programmer , you must have a strong self-learning ability, not only to quickly learn new knowledge that you have never been exposed to before in the face of unfamiliar tasks, but also to keep up with the ever-changing computer field.

use binary files to save data in the C language course design last semester , the work of checking the data is not too much , and single-step debugging is enough; the data volume of the data structure course design can be said to have doubled. In this case, saving in text files is the only choice. To be honest, I am used to storing binary files . I am still very unfamiliar with the storage format of text files. When character strings and numbers are cross-saved, it is not possible to determine where the reading of a variable or string will stop , when the spaces between the data should be added , and when not. But this time, I forced myself to use various variables and strings to conduct mixed reading and writing experiments, which can be regarded as making up for the debt owed by the C language; the observation and analysis of the data in the text file also made me debug The speed has been increased.

In the process of debugging the program, I realized the hardships of being a programmer. Programmers need a cautious and serious attitude and style, and must pay great attention to every tiny detail without being careless. When I started writing programs, I saw other people's programs with very detailed functions and beautiful interfaces. I always hoped that my own programs would also be very perfect. However, I found that writing a good program is not a one-off thing. It takes a long time. accumulated programming experience.

the written function body with the given text interface at the end. It took me two days to understand the sample code of the text interface . While splicing, I gradually came up with some ideas to increase user-friendliness and enrich system functions, so I also came into contact with some new functions . In this process, program debugging really takes up a lot of links, some are due to omissions, and some are indeed caused by lack of knowledge: such as the inexplicable failure of mouse input after the execution of the sys tem ( "pause" ) statement I don't understand it at all, but before I know this feature, I can only guess that I made a misoperation on the mouse input related functions before, so I basically checked all the text interface related functions again.

In the process of repeated debugging, I finally made a simple course scheduling evaluation system. Although the functions of this program are still very basic, it fails to take care of all aspects of the rationality of course scheduling. Due to my shallow knowledge, lack of experience and experience, there are still many deficiencies in the design of this course scheduling evaluation system, such as the interface is not so beautiful, and the function is relatively single. I believe that in the future study, with the growth of knowledge and the accumulation of experience, I can make better results.

The experience given to me by this course is: when programming in the future , if you encounter a process that you can only use your brain to figure out, the best way is to confirm its execution on paper . One type of operation becomes more intuitive, and it is also more likely to save a lot of time in the debugging process later. But on the other hand , confirming the execution status over and over again is also a manifestation of my inability to abstract a piece of code smoothly , which reduces my programming efficiency to a certain extent. So then again , only a large amount of daily code stacking can be exchanged for high-quality and efficient programming in tasks .

## references

[1]Yan Weimin, Wu Weimin. Data Structure (C language version). Beijing: Tsinghua University Press, 1997

[2]Yan Weimin, Wu Weimin, Mining. Data Structure Problem Collection (C Language Version). Beijing: Tsinghua University Press, 1999

[3]Wang Xiaodong. Computer Algorithm Design and Analysis. Beijing : Electronics Industry Press , 2007

[4]Wang Xiuhuan . Research on Optimizing Course Scheduling System in Colleges and Universities Based on Graph Theory . Master Thesis of Chongqing University , 20 12

[5]Hong Wen, Zhu Guangbin. Course scheduling problem and its mathematical model. Journal of Anhui Electric Power Workers University, 2002, 7(3): 74-77

[6] Tao Tao, Xie Weixing. Class schedule model and application of class scheduling algorithm. Computer System Application, 2011, 20(2): 198-201

[7] Xu Xiao . Research on course scheduling model based on ontology mapping and rule reasoning. Software Guide, 2009 , 8 ( 2): 1-4

## appendix

1. head . h

#include <stdio.h>

#include <stdlib.h>

#include <windows.h>

#include <wincon.h>

#include <conio.h>

#include <string.h>

#include <io.h>

#include <fcntl.h>

#include <sys\stat.h>

#include <ctype.h>

#include <time.h>

#ifndef TEST\_H

#define TEST\_H

#endif

#ifndef TYPE\_H\_INCLUDED

#define TYPE\_H\_INCLUDED

#define SCR\_ROW 60 /\*number of screen rows\*/

#define SCR\_COL 100 /\*Screen columns\*/

#define LIST\_INIT\_SIZE 100

#define LISTINCREMENT 10

/\*\*

\* Compress and store the linked list node structure during class time

\*/

typedef struct Occupy{

unsigned short week; //class weeks

unsigned short time; //Number of lessons

struct Occupy \*next;

}Occupy;

/\*\*

\* Compress and store the linked list header structure during class time

\*/

typedef struct ochead{

unsigned short len;

Occupy \*head;

}Ochead;

/\*\*

\*Teacher node structure

\*/

typedef struct{

char tname[20]; //teacher name

unsigned short gender, age; //Gender: female is 0, male is 1

Ochead ochead[6]; //Teacher occupancy, three-dimensional: week-week-classroom

}Teacher;

/\*\*

\* Teacher linear header structure

\*/

typedef struct {

Teacher \*elem; //linear table

unsigned short length, listsize; //table length, list size

}TeList;

/\*\*

\* Classroom node structure

\*/

typedef struct{

char code[20]; //classroom number

unsigned short rsize; //teacher capacity

Ochead ochead[6]; //Classroom occupancy, three-dimensional: week-week-classroom

}Room;

/\*\*

\*Classroom linear header structure

\*/

typedef struct {

Room \*elem; //linear table

unsigned short length, listsize; //table length, list size

}RoList;

/\*\*

\*Course node structure

\*/

typedef struct CouArc{

char \*prename; // prerequisite course name

struct CouArc \*nextarc; // pointer to the next arc

}CouArc; //Repair relationship node structure first

/\*\*

\* course inverse adjacency list arc node structure

\*/

typedef struct {

char name[60]; //course name

unsigned short arcnum;

CouArc \*archhead; // pointer to the head of the course

}CouHead;

/\*\*

\* course inverse adjacency header structure

\*/

typedef struct { //Course inverse adjacency table header structure

CouHead \*coulist; // reverse adjacency list

unsigned short length, listsize; // total number of courses

}CouGraph;

/\*\*

\*classroom node structure

\*/

typedef struct Class{

char coursename[60], roomcode[20], tname[20]; //Class name, classroom number, teacher name

unsigned short cou\_st, cou\_ed, wk\_st[5], wk\_ed[5]; //Start section number, end section number, start week number, continuous week number

struct Class \*next;

}Class;

/\*\*

\* class node structure

\*/

typedef struct Schedule{

char classname[20]; //class number

unsigned short do\_num, pr\_num, clsize; // Number of completed/current courses

char done[60][60], progress[15][60]; //Courses completed/in progress

struct weekday{

Class \*classhead; //class list head pointer

Ochead ochead; //head structure of class time chain list

unsigned short cl\_num; //class number

}week[6]; //There are no classes on Sunday, only six days a week

struct Schedule \*next;

}Schedule;

/\*\*

\* class list header structure

\*/

typedef struct {

unsigned short gr\_num; //class number

Schedule \*schead; //Schedule head pointer

}Grade;

/\*\*

\*Screen window information link node structure

\*/

typedef struct layer\_node {

char LayerNo; /\*\*< popup window layer number\*/

SMALL\_RECT rcArea; /\*\*< popup window area coordinates\*/

CHAR\_INFO \*pContent; /\*\*< The original information storage buffer of the pop-up window area character unit\*/

char \*pScrAtt; /\*\*< The storage buffer of the original attribute value of the character unit in the pop-up window area\*/

struct layer\_node \*next; /\*\*< pointer to the next node\*/

} LAYER\_NODE;

/\*\*

\* Tag bundle structure

\*/

typedef struct label1\_bundle {

char \*\*ppLabel; /\*\*< the first address of the label string array\*/

COORD \*pLoc; /\*\*< The first address of the label positioning array \*/

int num; /\*\*< number of tags\*/

} LABEL\_BUNDLE;

/\*\*

\* Hot zone structure

\*/

typedef struct hot\_area {

SMALL\_RECT \*pArea; /\*\*< The first address of the hot area positioning array\*/

char \*pSort; /\*\*< The first address of the hot area category (button, text box, option box) array\*/

char \*pTag; /\*\*< the first address of the hot zone serial number array\*/

int num; /\*\*< number of hot zones\*/

} HOT\_AREA;

LAYER\_NODE \*gp\_top\_layer = NULL; /\* pop-up window information chain head \*/

char \*gp\_sys\_name = "Course Scheduling System"; /\*system name\*/

char \*ga\_main\_menu[] = {"File(F)", /\*system main menu name\*/

" Data Maintenance (M)",

"Data query (Q)",

"Data statistics (S)",

"Help (H)"

};

char \*ga\_sub\_menu[] = {"[S] Data Backup", /\*system submenu name\*/

"[R] Data Recovery",

"[X] Exit Alt+X",

"[S] Teacher Information",

"[T] Classroom Information",

"[P] Class information",

"[S] Course Information",

"[U] Classroom information",

"[S] Teacher Information",

"[T] Classroom Information",

"[P] Class information",

"[S] Course Information",

"[U] Classroom information",

"[G] Classroom Analysis",

"[T] Teacher Analysis",

"[C] Classroom Analysis",

"[T] Help topics",

"",

"[A] about..."

};

int ga\_sub\_menu\_count[] = {3, 5, 5, 3, 3}; /\*The number of submenus under each main menu item\*/

int gi\_sel\_menu = 1; /\*The selected main menu item number, the initial is 1\*/

int gi\_sel\_sub\_menu = 0; /\*selected submenu item number, initially 0, means unselected\*/

CHAR\_INFO \*gp\_buff\_menubar\_info = NULL; /\*The buffer for storing the character information of the menu bar screen area\*/

CHAR\_INFO \*gp\_buff\_stateBar\_info = NULL; /\*The buffer for storing the character information of the status bar screen area\*/

char \*gp\_scr\_att = NULL; /\* buffer for storing character unit attribute values on the screen \*/

char gc\_sys\_state = '\0'; /\*Character used to save system state\*/

HANDLE gh\_std\_out; /\*Standard output device handle\*/

HANDLE gh\_std\_in; /\* standard input device handle \*/

void InitInterface(void); /\*system interface initialization\*/

void ClearScreen(void); /\*clear screen\*/

void ShowMenu(void); /\*Display the menu bar\*/

void PopMenu(int num); /\*Display drop-down menu\*/

void PopPrompt(int num); /\*Display the pop-up window\*/

void PopUp(SMALL\_RECT \*, WORD, LABEL\_BUNDLE \*, HOT\_AREA \*); /\* pop-up window screen information maintenance \*/

void PopOff(void); /\*Close the top-level pop-up window\*/

void DrawBox(SMALL\_RECT \*parea); /\*draw border\*/

void LocSubMenu(int num, SMALL\_RECT \*parea); /\*Main menu drop-down menu location\*/

void ShowState(void); /\*Display status bar\*/

void TagMainMenu(int num); /\*mark the selected main menu item\*/

void TagSubMenu(int num); /\*mark the selected submenu item\*/

int DealConInput(HOT\_AREA \*phot\_area, int \*pihot\_num); /\*console input processing\*/

void SetHotPoint(HOT\_AREA \*phot\_area, int hot\_num); /\*Set hot area\*/

BOOL ShowModule(char \*\*pString, int n); /\*popup information window\*/

BOOL Message(char \*\*); /\* pop-up window \*/

BOOL LoadData(void); /\*data loading\*/

void RunSys(void); /\*Selection and operation of system function modules\*/

BOOL ExeFunction(int main\_menu\_num, int sub\_menu\_num); /\*call of function module\*/

void CloseSys(void); /\* exit the system\*/

void ClearMemory(void); /\*clear memory data\*/

BOOL Initial(void); /\*data structure initialization\*/

BOOL AddTeacher(void); /\*insert teacher\*/

BOOL AddRoom(void); /\*insert classroom\*/

BOOL AddCourse(void); /\*insert course\*/

BOOL AddGroup(void); /\*insert class\*/

BOOL DelTeacher(void); /\*Delete teacher\*/

BOOL DelRoom(void); /\*Delete room\*/

BOOL DelCourse(void); /\*Delete course\*/

BOOL DelGroup(void); /\*Delete class\*/

BOOL AltTeacher(void); /\*modify teacher\*/

BOOL AltRoom(void); /\*Modify classroom\*/

BOOL AltCourse(void); /\*modify course\*/

BOOL AltGroup(void); /\*Modify class\*/

BOOL SeekTeacher(void); /\*find teacher\*/

BOOL SeekRoom(void); /\*find classroom\*/

BOOL SeekCourse(void); /\*find course\*/

BOOL SeekGroup(void); /\*find class\*/

BOOL MaintainTeacherInfo(void); /\*Maintain teacher information\*/

BOOL MaintainRoomInfo(void); /\*Maintain classroom information\*/

BOOL MaintainGroupInfo(void); /\*Maintain class information\*/

BOOL MaintainCourseInfo(void); /\*maintain course information\*/

BOOL MaintainClassInfo(void); /\*Maintain class information\*/

BOOL AddClass(void); /\*insert class\*/

BOOL DelClass(void); /\*Delete class\*/

BOOL AltClass(void); /\*modify class\*/

BOOL SeekClass(void); /\* find class\*/

BOOL RoomUsage(void); /\*room usage\*/

BOOL TeacherAnalysis(void); /\*Analysis of classroom scheduling\*/

BOOL ClassAnalysis(void); /\*class schedule analysis\*/

BOOL FileOut(void); /\*Array backup\*/

BOOL FileIn(void); /\*Array recovery\*/

BOOL ExitSys(void); /\*exit the system\*/

BOOL HelpTopic(void); /\*Help topic\*/

BOOL AboutDorm(void); /\*About the system\*/

BOOL DelTimeNode(unsigned short wk\_st[], unsigned short wk\_ed[], unsigned short cou\_st, unsigned short cou\_ed, Ochead \*ochead); /\*delete class time nodes\*/

BOOL AddTimeNode(unsigned short wk\_st[], unsigned short wk\_ed[], unsigned short cou\_st, unsigned short cou\_ed, Ochead \*ochead); /\*Add the class time node\*/

Schedule \*ClassInput(char cbuf[]); /\*class name input\*/

BOOL TimeInput(unsigned short \*cou\_st, unsigned short \*cou\_ed, unsigned short wk\_st[], unsigned short wk\_ed[]); /\*class name input\*/

BOOL AddNewRoom(char cbuf[]); /\*insert new room\*/

BOOL AddNewTeacher(char cbuf[]); /\*insert new teacher\*/

BOOL AddNewCourse(char cbuf[]); /\*insert a new course\*/

BOOL SeekRemClass(Schedule \*ps, char cbuf[]); /\*Find remaining classes\*/

BOOL DelClassNode(Class \*pc, Schedule \*ps, unsigned short j); /\*Delete class node\*/

BOOL Add(void);

TeList \*telist=NULL;

RoList \*rolist=NULL;

CouGraph \*cougraph=NULL;

Grade grade[3];

CouArc \*pcarc=NULL, \*qcarc=NULL;

Occupy \*pocc=NULL, \*qocc=NULL;

Schedule \*psch=NULL, \*qsch=NULL;

Class \*pcls=NULL, \*qcls=NULL;

unsigned long ul;

unsigned short choice, hubuffer, buf1, buf2, buf3[5], buf4[5], buf5, buf6;

unsigned char i, j, k, l, m, n, b=' ', count;

char cbuffer[60], cbuf1[60], cbuf2[60];

BOOL flag;

char \*pError[]={" No target information found! Return to main menu?", "OK", "Cancel"};

char \*pContinue[]={"Back to the main menu?", "OK", "Cancel"};

#endif /\*\*< TYPE\_H\_INCLUDED\*/

1. main.c

#include "head.h"

int main()

{

COORD size = {SCR\_COL, SCR\_ROW}; /\*window buffer size\*/

gh\_std\_out = GetStdHandle(STD\_OUTPUT\_HANDLE); /\* Get standard output device handle\*/

gh\_std\_in = GetStdHandle(STD\_INPUT\_HANDLE); /\* Get standard input device handle\*/

SetConsoleTitle(gp\_sys\_name); /\*Set window title\*/

SetConsoleScreenBufferSize(gh\_std\_out, size); /\*Set the window buffer size to 80\*25\*/

//FileIn();

Initial(); /\*data loading\*/

InitInterface(); /\*Interface initialization\*/

RunSys(); /\*Selection and operation of system function modules\*/

CloseSys(); /\*Exit the system\*/

return 0;

}

/\*\*<The first part of text interface function\*/

/\*\*

\* Function name: InitInterface

\* Function function: Initialize the interface.

\* Input parameters: none

\* Output parameters: none

\* Return value: None

\*

\* Call description:

\*/

void InitInterface()

{

ClearScreen(); /\* clear screen\*/

/\* Create pop-up window information stack, treat the initialized screen window as the first layer of pop-up window\*/

gp\_scr\_att = (char \*)calloc(SCR\_COL \* SCR\_ROW, sizeof(char));/\*screen character attributes\*/

gp\_top\_layer = (LAYER\_NODE \*)malloc(sizeof(LAYER\_NODE));

gp\_top\_layer->LayerNo = 0; /\*The layer number of the pop-up window is 0\*/

gp\_top\_layer->rcArea.Left = 0; /\*The area of the pop-up window is the entire screen window\*/

gp\_top\_layer->rcArea.Top = 0;

gp\_top\_layer->rcArea. Right = SCR\_COL - 1;

gp\_top\_layer->rcArea.Bottom = SCR\_ROW - 1;

gp\_top\_layer->pContent = NULL;

gp\_top\_layer->pScrAtt = gp\_scr\_att;

gp\_top\_layer->next = NULL;

ShowMenu(); /\*Show menu bar\*/

ShowState(); /\*Display status bar\*/

return;

}

/\*\*

\* Function name: ClearScreen

\* Function: Clear screen information.

\* Input parameters: none

\* Output parameters: none

\* Return value: None

\*

\* Call description:

\*/

void ClearScreen(void)

{

CONSOLE\_SCREEN\_BUFFER\_INFO bInfo;

COORD home = {0, 1}; /\*Only clear the information starting from the first line, so that it can be called by each function\*/

unsigned long size;

GetConsoleScreenBufferInfo( gh\_std\_out, &bInfo );/\*Get screen buffer information\*/

size = bInfo.dwSize.X \* (bInfo.dwSize.Y); /\* Calculate the number of screen buffer character units\*/

/\*Set the character attributes of all units in the screen buffer to the character attributes of the current screen buffer\*/

FillConsoleOutputAttribute(gh\_std\_out, bInfo.wAttributes, size, home, &ul);

/\* Fill all cells of the screen buffer with space characters\*/

FillConsoleOutputCharacter(gh\_std\_out, ' ', size, home, &ul);

SetConsoleCursorPosition(gh\_std\_out, home); /\* The cursor is placed in the upper left corner of the window\*/

return;

}

/\*\*

\* Function name: ShowMenu

\* Function function: display the main menu on the screen, and set the hot zone, and put a check mark on the first item of the main menu.

\* Input parameters: none

\* Output parameters: None

\* Return value: None

\*

\* Call description:

\*/

void ShowMenu()

{

CONSOLE\_SCREEN\_BUFFER\_INFO bInfo;

CONSOLE\_CURSOR\_INFO lpCur;

COORD size, pos = {0, 0};

int PosA = 2, PosB, i, j;

char ch;

GetConsoleScreenBufferInfo( gh\_std\_out, &bInfo );

size.X = bInfo.dwSize.X; size.Y = 1;

SetConsoleCursorPosition(gh\_std\_out, pos);

for (i=0; i < 5; i++) /\*Output the main menu item at the first row and first column of the window\*/

{

printf(" %s ", ga\_main\_menu[i]);

}

GetConsoleCursorInfo(gh\_std\_out, &lpCur);

lpCur.bVisible = FALSE;

SetConsoleCursorInfo(gh\_std\_out, &lpCur); /\*hide cursor\*/

/\* Apply for a dynamic storage area as a buffer for storing character information in the menu bar screen area \*/

gp\_buff\_menubar\_info = (CHAR\_INFO \*)malloc(size. X \* size. Y \* sizeof(CHAR\_INFO));

SMALL\_RECT rcMenu ={0, 0, size.X-1, 0} ;

/\*Read the content of the first line of the window into the buffer that stores the character information of the menu bar screen area\*/

ReadConsoleOutput(gh\_std\_out, gp\_buff\_menubar\_info, size, pos, &rcMenu);

/\*Set the Chinese and English letters in this line to red, and set other character units to black on white background\*/

for (i=0; i<size.X; i++)

{

(gp\_buff\_menubar\_info+i)->Attributes = BACKGROUND\_BLUE | BACKGROUND\_GREEN

| BACKGROUND\_RED;

ch = (char)((gp\_buff\_menubar\_info+i)->Char.AsciiChar);

if ((ch >= 'A' && ch <= 'Z') || (ch >= 'a' && ch <= 'z'))

{

(gp\_buff\_menubar\_info+i)->Attributes |= FOREGROUND\_RED;

}

}

/\*The modified menu bar character information is written back to the first line of the window\*/

WriteConsoleOutput(gh\_std\_out, gp\_buff\_menubar\_info, size, pos, &rcMenu);

COORD endPos = {0, 1};

SetConsoleCursorPosition(gh\_std\_out, endPos); /\*Set the cursor position to row 2, column 1\*/

/\*Set the menu item as a hot zone, the hot zone number is the menu item number, and the hot zone type is 0 (button type)\*/

i = 0;

do

{

PosB = PosA + strlen(ga\_main\_menu[i]); /\*Locate the start and end positions of the i+1th menu item\*/

for (j=PosA; j<PosB; j++)

{

gp\_scr\_att[j] |= (i+1) << 2; /\*Set the attribute value of the character unit where the menu item is located\*/

}

PosA = PosB + 4; i++;

} while (i<5);

TagMainMenu(gi\_sel\_menu); /\*mark the selected main menu item, the initial value of gi\_sel\_menu is 1\*/

return;

}

/\*\*

\* Function name: ShowState

\* Function function: display the status bar.

\* Input parameters: None

\* Output parameters: None

\* return value: None

\*

\* Call description: The character attribute of the status bar is black on a white background, and there is no status information in the initial state.

\*/

void ShowState()

{

CONSOLE\_SCREEN\_BUFFER\_INFO bInfo;

COORD size, pos = {0, 0};

int i;

GetConsoleScreenBufferInfo( gh\_std\_out, &bInfo );

size.X = bInfo.dwSize.X; size.Y = 1;

SMALL\_RECT rcMenu ={0, bInfo.dwSize.Y-1, size.X-1, bInfo.dwSize.Y-1};

if (gp\_buff\_stateBar\_info == NULL)

{

gp\_buff\_stateBar\_info = (CHAR\_INFO \*)malloc(size. X \* size. Y \* sizeof(CHAR\_INFO));

ReadConsoleOutput(gh\_std\_out, gp\_buff\_stateBar\_info, size, pos, &rcMenu);

}

for (i=0; i<size.X; i++)

{

(gp\_buff\_stateBar\_info+i)->Attributes = BACKGROUND\_BLUE | BACKGROUND\_GREEN

| BACKGROUND\_RED;

}

WriteConsoleOutput(gh\_std\_out, gp\_buff\_stateBar\_info, size, pos, &rcMenu);

return;

}

/\*\*

\* Function name: TagMainMenu

\* Function: Set the check mark on the specified main menu item.

\* Input parameters: num selected main menu item number

\* Output parameters: none

\* Return value: None

\*

\* Call description:

\*/

void TagMainMenu(int num)

{

CONSOLE\_SCREEN\_BUFFER\_INFO bInfo;

COORD size, pos = {0, 0};

int PosA = 2, PosB, i;

char ch;

if (num == 0) /\*When num is 0, the check mark of the main menu item will be removed\*/

{ PosA = PosB = 0; }

else /\*Otherwise, locate the start and end positions of the selected main menu item: PosA is the start position, PosB is the end position\*/

{

for (i=1; i<num; i++)

{

PosA += strlen(ga\_main\_menu[i-1]) + 4;

}

PosB = PosA + strlen(ga\_main\_menu[num-1]);

}

GetConsoleScreenBufferInfo( gh\_std\_out, &bInfo );

size.X = bInfo.dwSize.X; size.Y = 1;

/\*Remove the menu item check mark in front of the selected menu item\*/

for (i=0; i<PosA; i++)

{

(gp\_buff\_menubar\_info+i)->Attributes = BACKGROUND\_BLUE | BACKGROUND\_GREEN

| BACKGROUND\_RED;

ch = (gp\_buff\_menubar\_info+i)->Char.AsciiChar;

if ((ch >= 'A' && ch <= 'Z') || (ch >= 'a' && ch <= 'z'))

{

(gp\_buff\_menubar\_info+i)->Attributes |= FOREGROUND\_RED;

}

}

/\*Mark on the selected menu item, black background and white text\*/

for (i=PosA; i<PosB; i++)

{

(gp\_buff\_menubar\_info+i)->Attributes = FOREGROUND\_BLUE | FOREGROUND\_GREEN

| FOREGROUND\_RED;

}

/\*Remove the menu item check mark behind the selected menu item\*/

for (i=PosB; i<bInfo.dwSize.X; i++)

{

(gp\_buff\_menubar\_info+i)->Attributes = BACKGROUND\_BLUE | BACKGROUND\_GREEN

| BACKGROUND\_RED;

ch = (char)((gp\_buff\_menubar\_info+i)->Char.AsciiChar);

if ((ch >= 'A' && ch <= 'Z') || (ch >= 'a' && ch <= 'z'))

{

(gp\_buff\_menubar\_info+i)->Attributes |= FOREGROUND\_RED;

}

}

/\*Write the marked menu bar information to the first line of the window\*/

SMALL\_RECT rcMenu ={0, 0, size.X-1, 0};

WriteConsoleOutput(gh\_std\_out, gp\_buff\_menubar\_info, size, pos, &rcMenu);

return;

}

/\*\*

\* Function name: PopMenu

\* Function function: In the submenu and set the hot zone, set the check mark with the cursor movement.

\* Input parameters: num submenu number of items from top to bottom

\* Output parameters: none

\* Return value: None

\*

\* Call description:

\*/

void PopMenu(int num)

{

LABEL\_BUNDLE labels;

HOT\_AREA areas;

SMALL\_RECT rcPop;

COORD pos;

WORD att;

char \*pCh;

int i, j, loc = 0;

if (num != gi\_sel\_menu) /\*If the specified main menu is not the selected menu\*/

{

if (gp\_top\_layer->LayerNo != 0) /\*If a submenu has popped up before\*/

{

PopOff(); gi\_sel\_sub\_menu = 0;

}

}

else if (gp\_top\_layer->LayerNo != 0) /\*If the submenu has popped up, return\*/

{ return; }

gi\_sel\_menu = num; /\*Set the selected main menu item to the specified main menu item\*/

TagMainMenu(gi\_sel\_menu); /\*mark the selected main menu item\*/

LocSubMenu(gi\_sel\_menu, &rcPop); /\*Calculate the area position of the pop-up submenu, store it in rcPop\*/

/\* Calculate the position (subscript) of the first item in the submenu in the submenu string array\*/

for (i=1; i<gi\_sel\_menu; i++)

{

loc += ga\_sub\_menu\_count[i-1];

}

/\* Store the item name of this group of submenu items in the tag bundle structure variable\*/

labels.ppLabel = ga\_sub\_menu + loc; /\*Address of the first label string in the label bundle\*/

labels.num = ga\_sub\_menu\_count[gi\_sel\_menu-1]; /\*The number of label strings in the label bundle\*/

COORD aLoc[labels.num];/\*Define a coordinate array to store the coordinates of the output position of each label string\*/

for (i=0; i<labels.num; i++) /\*Determine the output position of the label string and store it in the coordinate array\*/

{

aLoc[i].X = rcPop.Left + 2; aLoc[i].Y = rcPop.Top + i + 1;

}

labels.pLoc = aLoc; /\*Make the member pLoc of the label bundle structure variable labels point to the first element of the coordinate array\*/

/\*Set hot zone information\*/

areas.num = labels.num; /\*The number of hot areas is equal to the number of labels, that is, the number of items in the submenu\*/

SMALL\_RECT aArea[areas.num]; /\*Define an array to store all hot area locations\*/

char aSort[areas.num]; /\*Define an array to store the corresponding categories of all hot areas\*/

char aTag[areas.num]; /\*Define an array to store the number of each hot area\*/

for (i=0; i<areas. num; i++)

{

aArea[i].Left = rcPop.Left + 2; /\* Hot area positioning\*/

aArea[i].Top = rcPop.Top + i + 1;

aArea[i].Right = rcPop.Right - 2;

aArea[i].Bottom = aArea[i].Top;

aSort[i] = 0; /\* hot zone category is 0 (button type)\*/

aTag[i] = i + 1; /\*Hot zones are numbered sequentially\*/

}

areas.pArea = aArea;/\*Make the member pArea of the hot area structure variable areas point to the first element of the hot area position array\*/

areas.pSort = aSort;/\*Make the member pSort of the hot area structure variable areas point to the first element of the hot area category array\*/

areas.pTag = aTag; /\*Make the member pTag of the hot area structure variable areas point to the first element of the hot area number array\*/

att = BACKGROUND\_BLUE | BACKGROUND\_GREEN | BACKGROUND\_RED; /\*black text on white background\*/

PopUp(&rcPop, att, &labels, &areas);

DrawBox(&rcPop); /\* Draw a border for the pop-up window\*/

pos.X = rcPop.Left + 2;

for (pos.Y=rcPop.Top+1; pos.Y<rcPop.Bottom; pos.Y++)

{ /\*This cycle is used to draw a line at the position of the empty string sub-menu item to form a separation, and cancel the hot zone attribute of this menu item\*/

pCh = ga\_sub\_menu[loc+pos.Y-rcPop.Top-1];

if (strlen(pCh)==0) /\*The string length is 0, indicating that it is an empty string\*/

{ /\*First draw a horizontal line\*/

FillConsoleOutputCharacter(gh\_std\_out, '-', rcPop.Right-rcPop.Left-3, pos, &ul);

for (j=rcPop.Left+2; j<rcPop.Right-1; j++)

{ /\* cancel the hot area attribute of the character unit in this area \*/

gp\_scr\_att[pos.Y\*SCR\_COL+j] &= 3; /\*The result of bitwise AND retains the lower two bits\*/

}

}

}

/\*Set the function key of the submenu item as red letter on white background\*/

pos.X = rcPop.Left + 3;

att = FOREGROUND\_RED|BACKGROUND\_BLUE|BACKGROUND\_GREEN|BACKGROUND\_RED;

for (pos.Y=rcPop.Top+1; pos.Y<rcPop.Bottom; pos.Y++)

{

if (strlen(ga\_sub\_menu[loc+pos.Y-rcPop.Top-1])==0)

{ continue; /\*skip empty string\*/ }

FillConsoleOutputAttribute(gh\_std\_out, att, 1, pos, &ul);

}

return;

}

/\*\*

\* Function name: PopUp

\* Function: Output the pop-up window information in the specified area, set the hot zone at the same time, and push the pop-up window position information into the stack.

\* Input parameters: pRc The address where the location data of the pop-up window is stored

\* att popup window area character attribute

\* pLabel The address where the label bundle information is stored in the popup window

pHotArea The address where the hot area information is stored in the pop-up window

\* Output parameters: None

\* Return value: None

\*

\* Call description:

\*/

void PopUp(SMALL\_RECT \*pRc, WORD att, LABEL\_BUNDLE \*pLabel, HOT\_AREA \*pHotArea)

{

LAYER\_NODE \*nextLayer;

COORD size, pos = {0, 0};

char \*pCh;

int i, j, row;

/\* The character unit information of the location of the pop-up window is pushed into the stack\*/

size.X = pRc->Right - pRc->Left + 1; /\*The width of the pop-up window\*/

size.Y = pRc->Bottom - pRc->Top + 1; /\*The height of the pop-up window\*/

/\*Apply for a dynamic storage area for storing pop-up window related information\*/

nextLayer = (LAYER\_NODE \*)malloc(sizeof(LAYER\_NODE));

nextLayer->next = gp\_top\_layer;

nextLayer->LayerNo = gp\_top\_layer->LayerNo + 1;

nextLayer->rcArea = \*pRc;

nextLayer->pContent = (CHAR\_INFO \*)malloc(size.X\*size.Y\*sizeof(CHAR\_INFO));

nextLayer->pScrAtt = (char \*)malloc(size.X\*size.Y\*sizeof(char));

pCh = nextLayer->pScrAtt;

/\*Save the character information of the area covered by the pop-up window to restore it when the pop-up window is closed\*/

ReadConsoleOutput(gh\_std\_out, nextLayer->pContent, size, pos, pRc);

for (i=pRc->Top; i<=pRc->Bottom; i++)

{ /\*This double cycle stores the original attribute value of the covered character unit into the dynamic storage area, which is convenient for later restoration\*/

for (j=pRc->Left; j<=pRc->Right; j++)

{

\*pCh = gp\_scr\_att[i\*SCR\_COL+j]; pCh++;

}

}

gp\_top\_layer = nextLayer; /\*Complete pop-up window related information into the stack\*/

/\* Set the new attribute of the character of the popup window area \*/

pos.X = pRc->Left;

pos.Y = pRc->Top;

for (i=pRc->Top; i<=pRc->Bottom; i++)

{

FillConsoleOutputAttribute(gh\_std\_out, att, size.X, pos, &ul);

pos.Y++;

}

/\* Output the label string in the label bundle at the set position \*/

for (i=0; i<pLabel->num; i++)

{

pCh = pLabel->ppLabel[i];

if (strlen(pCh) != 0)

{

WriteConsoleOutputCharacter(gh\_std\_out, pCh, strlen(pCh),

pLabel->pLoc[i], &ul);

}

}

/\* Set the new attribute of the character cell of the popup window area \*/

for (i=pRc->Top; i<=pRc->Bottom; i++)

{ /\*This double loop sets the layer number of the character unit\*/

for (j=pRc->Left; j<=pRc->Right; j++)

{

gp\_scr\_att[i\*SCR\_COL+j] = gp\_top\_layer->LayerNo;

}

}

for (i=0; i<pHotArea->num; i++)

{ /\*This double loop sets the thermal zone type and thermal zone number of character cells in all thermal zones\*/

row = pHotArea->pArea[i].Top;

for (j=pHotArea->pArea[i].Left; j<=pHotArea->pArea[i].Right; j++)

{

gp\_scr\_att[row\*SCR\_COL+j] |= (pHotArea->pSort[i] << 6)

| (pHotArea->pTag[i] << 2);

}

}

return;

}

/\*\*

\* Function name: PopOff

\* Function function: close the top-level pop-up window, restore the original appearance of the covered area and the original properties of the character unit.

\* Input parameters: none

\* Output parameters: none

\* Return value: None

\*

\* Call description:

\*/

void PopOff(void)

{

LAYER\_NODE \*nextLayer;

COORD size, pos = {0, 0};

char \*pCh;

int i, j;

if ((gp\_top\_layer->next==NULL) || (gp\_top\_layer->pContent==NULL))

{ /\*The main interface screen information stored at the bottom of the stack, do not need to close\*/

return;

}

nextLayer = gp\_top\_layer->next;

/\*Restore the original appearance of the popup window area\*/

size.X = gp\_top\_layer->rcArea.Right - gp\_top\_layer->rcArea.Left + 1;

size.Y = gp\_top\_layer->rcArea.Bottom - gp\_top\_layer->rcArea.Top + 1;

WriteConsoleOutput(gh\_std\_out, gp\_top\_layer->pContent, size, pos, &(gp\_top\_layer->rcArea));

/\*Restore the original attribute of the character unit\*/

pCh = gp\_top\_layer->pScrAtt;

for (i=gp\_top\_layer->rcArea.Top; i<=gp\_top\_layer->rcArea.Bottom; i++)

{

for (j=gp\_top\_layer->rcArea.Left; j<=gp\_top\_layer->rcArea.Right; j++)

{

gp\_scr\_att[i\*SCR\_COL+j] = \*pCh; pCh++;

}

}

free(gp\_top\_layer->pContent); /\*Release dynamic storage area\*/

free(gp\_top\_layer->pScrAtt);

free(gp\_top\_layer);

gp\_top\_layer = nextLayer;

gi\_sel\_sub\_menu = 0;

return;

}

/\*\*

\* Function name: DrawBox

\* Function function: Draw a border in the specified area.

\* Input parameters: pRc stores the address of the location information of the region

\* Output parameters: none

\* Return value: None

\*

\* Call description:

\*/

void DrawBox(SMALL\_RECT \*pRc)

{

char chBox[] = {'+','-','|'}; /\*Character used for picture frame\*/

COORD pos = {pRc->Left, pRc->Top}; /\* positioned at the upper left corner of the area\*/

WriteConsoleOutputCharacter(gh\_std\_out, &chBox[0], 1, pos, &ul);/\*draw the upper left corner of the frame\*/

for (pos.X = pRc->Left + 1; pos.X < pRc->Right; pos.X++)

{ /\* This loop draws a horizontal line on the border \*/

WriteConsoleOutputCharacter(gh\_std\_out, &chBox[1], 1, pos, &ul);

}

pos.X = pRc->Right;

WriteConsoleOutputCharacter(gh\_std\_out, &chBox[0], 1, pos, &ul);/\*draw the upper right corner of the frame \*/

for (pos.Y = pRc->Top+1; pos.Y < pRc->Bottom; pos.Y++)

{ /\*This cycle draws the left and right lines of the border\*/

pos.X = pRc->Left;

WriteConsoleOutputCharacter(gh\_std\_out, &chBox[2], 1, pos, &ul);

pos.X = pRc->Right;

WriteConsoleOutputCharacter(gh\_std\_out, &chBox[2], 1, pos, &ul);

}

pos.X = pRc->Left; pos.Y = pRc->Bottom;

WriteConsoleOutputCharacter(gh\_std\_out, &chBox[0], 1, pos, &ul);/\*Draw the lower left corner of the frame\*/

for (pos.X = pRc->Left + 1; pos.X < pRc->Right; pos.X++)

{ /\*Draw the border horizontal line\*/

WriteConsoleOutputCharacter(gh\_std\_out, &chBox[1], 1, pos, &ul);

}

pos.X = pRc->Right;

WriteConsoleOutputCharacter(gh\_std\_out, &chBox[0], 1, pos, &ul);/\*Draw the lower right corner of the frame\*/

return;

}

/\*\*

\* Function name: TagSubMenu

\* Function function: Make a checkmark on the specified submenu item.

\* Input parameters: num selected submenu item number

\* Output parameters: none

\* Return value: None

\*

\* Call description:

\*/

void TagSubMenu(int num)

{

SMALL\_RECT rcPop;

COORD pos;

WORD att;

int width;

LocSubMenu(gi\_sel\_menu, &rcPop); /\*Calculate the area position of the pop-up submenu, store it in rcPop\*/

if ((num<1) || (num == gi\_sel\_sub\_menu) || (num>rcPop.Bottom-rcPop.Top-1))

{ /\* If the submenu item number is out of bounds, or the item submenu has been selected, return \*/

return;

}

pos.X = rcPop.Left + 2;

width = rcPop. Right - rcPop. Left - 3;

if (gi\_sel\_sub\_menu != 0) /\*First cancel the mark on the original selected submenu item\*/

{

pos.Y = rcPop.Top + gi\_sel\_sub\_menu;

att = BACKGROUND\_BLUE | BACKGROUND\_GREEN | BACKGROUND\_RED; /\*black text on white background\*/

FillConsoleOutputAttribute(gh\_std\_out, att, width, pos, &ul);

pos.X += 1;

att |= FOREGROUND\_RED;/\*red text on white background\*/

FillConsoleOutputAttribute(gh\_std\_out, att, 1, pos, &ul);

}

/\* Make a check mark on the specified submenu item \*/

pos.X = rcPop.Left + 2; pos.Y = rcPop.Top + num;

att = FOREGROUND\_BLUE | FOREGROUND\_GREEN | FOREGROUND\_RED; /\*white text on black background\*/

FillConsoleOutputAttribute(gh\_std\_out, att, width, pos, &ul);

gi\_sel\_sub\_menu = num; /\*Modify the selected submenu item number\*/

return;

}

/\*\*

\* Function name: LocSubMenu

\* Function: Calculate the position of the upper left corner and the lower right corner of the pop-up submenu area.

\* Input parameters: num selected main menu item number

\* Output parameter: rc is the address to store the location information of the region

\* Return value: None

\*

\* Call description:

\*/

void LocSubMenu(int num, SMALL\_RECT \*rc)

{

int i, len, loc = 0;

rc->Top = 1; /\*The top of the area is set on the second line, and the line number is 1\*/

rc->Left = 1;

for (i=1; i<num; i++)

{ /\* Calculate the position of the left border of the area, and calculate the position of the first submenu item in the submenu string array \*/

rc->Left += strlen(ga\_main\_menu[i-1]) + 4;

loc += ga\_sub\_menu\_count[i-1];

}

rc->Right = strlen(ga\_sub\_menu[loc]);/\*Temporarily store the string length of the first submenu item\*/

for (i=1; i<ga\_sub\_menu\_count[num-1]; i++)

{ /\*Find the longest submenu string and store its length in rc->Right\*/

len = strlen(ga\_sub\_menu[loc+i]);

if (rc->Right < len)

{ rc->Right = len; }

}

rc->Right += rc->Left + 3; /\*calculate the right boundary of the region\*/

rc->Bottom = rc->Top + ga\_sub\_menu\_count[num-1] + 1; /\* Calculate the line number below the area\*/

if (rc->Right >= SCR\_COL) /\*The processing of the right boundary crossing \*/

{

len = rc->Right - SCR\_COL + 1;

rc->Left -= len;

rc->Right = SCR\_COL - 1;

}

return;

}

/\*\*

\* Function name: DealInput

\* Function function: set the hot zone in the pop-up window area, wait for and respond to user input.

\* Input parameters: pHotArea hot area parameter pointer

\* piHot The storage address of the focus hot zone number, that is, the pointer to the focus hot zone number

\* Output parameter: pHotArea hot area parameter

\* piHot returns the current hot spot number when clicking with the mouse, pressing enter or space

\* Return value: iRet carriage return virtual key code

\*

\* Call description:

\*/

int DealInput(HOT\_AREA \*pHotArea, int \*piHot)

{

INPUT\_RECORD inRec;

DWORD res;

COORD pos = {0, 0};

int num, arrow, iRet = 0;

int cNo, cTag, cSort;/\*cNo: layer number, cTag: hot zone number, cSort: hot zone type\*/

char vkc, asc; /\*vkc: virtual key code, asc: character ASCII code value\*/

SetHotPoint(pHotArea, \*piHot);

while (TRUE)

{ /\*loop\*/

ReadConsoleInput(gh\_std\_in, &inRec, 1, &res);

if ((inRec. EventType == MOUSE\_EVENT) &&

(inRec.Event.MouseEvent.dwButtonState

== FROM\_LEFT\_1ST\_BUTTON\_PRESSED))

{

pos = inRec.Event.MouseEvent.dwMousePosition;

cNo = gp\_scr\_att[pos.Y \* SCR\_COL + pos.X] & 3;

cTag = (gp\_scr\_att[pos.Y \* SCR\_COL + pos.X] >> 2) & 15;

cSort = (gp\_scr\_att[pos.Y \* SCR\_COL + pos.X] >> 6) & 3;

if ((cNo == gp\_top\_layer->LayerNo) && cTag > 0)

{

\*piHot = cTag;

SetHotPoint(pHotArea, \*piHot);

if (cSort == 0)

{ iRet = 13; break; }

}

}

else if (inRec.EventType == KEY\_EVENT && inRec.Event.KeyEvent.bKeyDown)

{

vkc = inRec.Event.KeyEvent.wVirtualKeyCode;

asc = inRec.Event.KeyEvent.uChar.AsciiChar;;

if (asc == 0)

{

arrow = 0;

switch (vkc)

{ /\*Arrow keys (left, up, right, down) processing\*/

case 37: arrow = 1; break;

case 38: arrow = 2; break;

case 39: arrow = 3; break;

case 40: arrow = 4; break;

}

if (arrow > 0)

{

num = \*piHot;

while (TRUE)

{

if (arrow < 3) { num--; }

else { num++; }

if ((num < 1) || (num > pHotArea->num) ||

((arrow % 2) && (pHotArea->pArea[num-1].Top

== pHotArea->pArea[\*piHot-1].Top)) || ((!(arrow % 2))

&& (pHotArea->pArea[num-1].Top

!= pHotArea->pArea[\*piHot-1].Top)))

{ break; }

}

if (num > 0 && num <= pHotArea->num)

{

\*piHot = num;

SetHotPoint(pHotArea, \*piHot);

}

}

}

else if (vkc == 27)

{ /\*ESC key\*/ iRet = 27; break; }

else if (vkc == 13 || vkc == 32)

{ /\*Enter key or space means pressing the current button\*/ iRet = 13; break; }

}

}

return iRet;

}

/\*\*

\* Function name: SetHotPoint

\* Function function: set the hot zone in the pop-up window area, wait for and respond to user input.

\* Input parameters: pHotArea hot area parameter pointer

\* iHot focus hot zone number

\* Output parameter: pHotArea hot area parameter

\* Return value: None

\*

\* Call description:

\*/

void SetHotPoint(HOT\_AREA \*pHotArea, int iHot)

{

CONSOLE\_CURSOR\_INFO lpCur;

COORD pos = {0, 0};

WORD att1, att2;

int i, width;

att1 = FOREGROUND\_BLUE | FOREGROUND\_GREEN | FOREGROUND\_RED; /\*white text on black background\*/

att2 = BACKGROUND\_BLUE | BACKGROUND\_GREEN | BACKGROUND\_RED; /\*black text on white background\*/

for (i=0; i<pHotArea->num; i++)

{ /\*Set button hotspots to black text on a white background\*/

pos.X = pHotArea->pArea[i].Left;

pos.Y = pHotArea->pArea[i].Top;

width = pHotArea->pArea[i].Right - pHotArea->pArea[i].Left + 1;

if (pHotArea->pSort[i] == 0)

{ /\* hot zone is a button class \*/

FillConsoleOutputAttribute(gh\_std\_out, att2, width, pos, &ul);

}

}

pos.X = pHotArea->pArea[iHot-1].Left;

pos.Y = pHotArea->pArea[iHot-1].Top;

width = pHotArea->pArea[iHot-1].Right - pHotArea->pArea[iHot-1].Left + 1;

if (pHotArea->pSort[iHot-1] == 0)

{ /\*The activated hotspot is the button class\*/

FillConsoleOutputAttribute(gh\_std\_out, att1, width, pos, &ul);

}

else if (pHotArea->pSort[iHot-1] == 1)

{ /\*The activated hotspot is the text box class\*/

SetConsoleCursorPosition(gh\_std\_out, pos);

GetConsoleCursorInfo(gh\_std\_out, &lpCur);

lpCur.bVisible = TRUE;

SetConsoleCursorInfo(gh\_std\_out, &lpCur);

}

}

/\*\*

\* Function name: ShowModule

\* Function function: restore the system code table and three types of basic data from the specified data file.

\* Input parameters: pString The first address of the two-dimensional array of text information

\* n number of strings

\* Output parameters:

\* Return value: BOOL type, always TRUE

\*

\* Call description:

\*/

BOOL ShowModule(char \*\*pString, int n)

{

LABEL\_BUNDLE labels;

HOT\_AREA areas;

BOOL bRet = TRUE;

SMALL\_RECT rcPop;

COORD pos;

WORD att;

int i, maxlen, str\_len, iHot = 1;

for (i=0,maxlen=0; i<n; i++) {

str\_len = strlen(pString[i]);

if (maxlen < str\_len) {

maxlen = str\_len;

}

}

pos.X = maxlen + 16; pos.Y = n + 15;

rcPop.Left = (SCR\_COL - pos.X) / 2 - 10;

rcPop.Right = rcPop.Left + pos.X - 1;

rcPop.Top = (SCR\_ROW - pos.Y) / 2 - 12;

rcPop.Bottom = rcPop.Top + pos.Y - 1;

att = BACKGROUND\_BLUE | BACKGROUND\_GREEN | BACKGROUND\_RED; /\*black text on white background\*/

labels.num = n;

labels.ppLabel = pString;

COORD aLoc[n];

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

aLoc[i].X = rcPop.Left + 3;

aLoc[i].Y = rcPop.Top + (pos.Yn)/2 + i;

}

str\_len = strlen(pString[n-1]);

aLoc[n-1].X = rcPop.Left + 3 + (maxlen-str\_len)/2;

aLoc[n-1].Y = aLoc[n-1].Y + 2;

labels.pLoc = aLoc;

areas.num = 1;

SMALL\_RECT aArea[] = {{aLoc[n-1].X, aLoc[n-1].Y,

aLoc[n-1].X + 3, aLoc[n-1].Y}};

char aSort[] = {0}, aTag[] = {1};

areas.pArea = aArea;

areas.pSort = aSort;

areas.pTag = aTag;

PopUp(&rcPop, att, &labels, &areas);

pos.X = rcPop.Left + 1; pos.Y = rcPop.Top + (pos.Yn)/2 + n;

FillConsoleOutputCharacter(gh\_std\_out, '-', rcPop.Right-rcPop.Left-1, pos, &ul);

DealInput(&areas, &iHot);

PopOff();

return bRet;

}

/\*\*

\* Function name: Message

\* Function function: pop-up dialog box

\* Input parameters: \*\*pCh The first address of the dialog box information array

\* Output parameters: None

\* Return value: BOOL type, exit is TRUE, cancel is FALSE

\*

\* Call description:

\*/

BOOL Message(char \*\*pCh)

{

LABEL\_BUNDLE labels;

HOT\_AREA areas;

SMALL\_RECT rcPop;

COORD pos;

WORD att;

int iHot = 1;

char len[3] = { (char)strlen(pCh[0]), (char)strlen(pCh[1]), (char)strlen(pCh[2])};

char lmax = (len[0] > len[1]+len[2]+4) ? len[0] : len[1]+len[2]+4;

pos.X = lmax + 6;

pos.Y = 7;

char blanc = (pos.X - (len[1]+len[2]+4)) / 2;

rcPop.Left = (SCR\_COL - pos.X) / 2 - 10;

rcPop.Right = rcPop.Left + pos.X - 1;

rcPop.Top = (SCR\_ROW - pos.Y) / 2 - 12;

rcPop.Bottom = rcPop.Top + pos.Y - 1;

att = BACKGROUND\_BLUE | BACKGROUND\_GREEN | BACKGROUND\_RED; /\*black text on white background\*/

labels.num = 3;

labels.ppLabel = pCh;

COORD aLoc[] = {{rcPop.Left+3, rcPop.Top+2},

{rcPop.Left+blanc, rcPop.Top+5},

{rcPop.Right-blanc-(len[2]-1), rcPop.Top+5}};

labels.pLoc = aLoc;

areas.num = 2;

SMALL\_RECT aArea[] = {{rcPop.Left+blanc, rcPop.Top+5,

rcPop.Left+blanc+(len[1]-1), rcPop.Top + 5},

{rcPop.Right-blanc-(len[2]-1), rcPop.Top + 5,

rcPop.Right - blanc, rcPop.Top + 5}};

char aSort[] = {0, 0}, aTag[] = {1, 2};

areas.pArea = aArea;

areas.pSort = aSort;

areas.pTag = aTag;

PopUp(&rcPop, att, &labels, &areas);

pos.X = rcPop.Left + 1; pos.Y = rcPop.Top + 4;

FillConsoleOutputCharacter(gh\_std\_out, '-', rcPop.Right-rcPop.Left-1, pos, &ul);

if (DealInput(&areas, &iHot) == 13 && iHot == 1)

{ return FALSE; }

else{ return TRUE; }

}

/\*\*<The second part of the system skeleton function\*/

/\*\*

\* Function name: RunSys

\* Function function: Run the system, run the function module selected by the user under the main interface of the system.

\* Input parameters: None

\* Output parameters: None

\* Return value: None

\*

\* Call description:

\*/

void RunSys(void)

{

INPUT\_RECORD inRec;

DWORD res;

COORD pos = {0, 0};

BOOL bRet = TRUE;

int i, loc, num;

int cNo, cAtt; /\*cNo: character unit layer number, cAtt: character unit attribute\*/

char vkc, asc; /\*vkc: virtual key code, asc: character ASCII code value\*/

while (bRet)

{

/\* read a record from the console input buffer \*/

ReadConsoleInput(gh\_std\_in, &inRec, 1, &res);

if (inRec.EventType == MOUSE\_EVENT) /\*If the record is generated by mouse event\*/

{

pos = inRec.Event.MouseEvent.dwMousePosition; /\*Get mouse coordinate position\*/

cNo = gp\_scr\_att[pos.Y \* SCR\_COL + pos.X] & 3; /\*Get the layer number of this position\*/

cAtt = gp\_scr\_att[pos.Y \* SCR\_COL + pos.X] >> 2;/\*take the character unit attribute\*/

if (cNo == 0) /\*The layer number is 0, indicating that the position is not covered by the pop-up submenu\*/

{

/\* cAtt > 0 indicates that the position is in the hot zone (main menu item character unit)

\* cAtt != gi\_sel\_menu indicates that the main menu item at this position is not selected

\* gp\_top\_layer->LayerNo > 0 indicates that there is currently a submenu pop-up

\*/

if (cAtt > 0 && cAtt != gi\_sel\_menu && gp\_top\_layer->LayerNo > 0)

{

PopOff(); /\*Close the pop-up submenu\*/

gi\_sel\_sub\_menu = 0; /\*Set the item number of the selected submenu item to 0\*/

the submenu corresponding to the main menu item where the mouse is located \*/

}

}

else if (cAtt > 0) /\*The position of the mouse is the menu item character unit of the pop-up submenu\*/

{

TagSubMenu(cAtt); /\*Make a checkmark on the submenu item\*/

}

if (inRec.Event.MouseEvent.dwButtonState

== FROM\_LEFT\_1ST\_BUTTON\_PRESSED) /\*If the first button on the left side of the mouse is pressed\*/

{

if (cNo == 0) /\*The layer number is 0, indicating that the position is not covered by the pop-up submenu\*/

{

if (cAtt > 0) /\*If the position is in the hot zone (main menu item character unit)\*/

{

PopMenu(cAtt); /\*Pop up the submenu corresponding to the main menu item where the mouse is located\*/

}

/\*If the position does not belong to the character unit of the main menu item, and there is a submenu to pop up\*/

else if (gp\_top\_layer->LayerNo > 0)

{

PopOff(); /\*Close the pop-up submenu\*/

gi\_sel\_sub\_menu = 0; /\*Set the item number of the selected submenu item to 0\*/

}

}

else /\*The layer number is not 0, indicating that the position is covered by the pop-up submenu\*/

{

if (cAtt > 0) /\*if the position is in the hot zone (submenu item character unit)\*/

{

PopOff(); /\*Close the pop-up submenu\*/

gi\_sel\_sub\_menu = 0; /\*Set the item number of the selected submenu item to 0\*/

/\* Execute the corresponding function: gi\_sel\_menu main menu item number, cAtt submenu item number\*/

bRet = ExeFunction(gi\_sel\_menu, cAtt);

}

}

}

else if (inRec.Event.MouseEvent.dwButtonState

== RIGHTMOST\_BUTTON\_PRESSED) /\*If the right mouse button is pressed\*/

{

if (cNo == 0) /\*The layer number is 0, indicating that the position is not covered by the pop-up submenu\*/

{

PopOff(); /\*Close the pop-up submenu\*/

gi\_sel\_sub\_menu = 0; /\*Set the item number of the selected submenu item to 0\*/

}

}

}

else if (inRec.EventType == KEY\_EVENT /\*if the record is generated by a key\*/

&& inRec.Event.KeyEvent.bKeyDown) /\*and the key is pressed\*/

{

vkc = inRec.Event.KeyEvent.wVirtualKeyCode; /\*Get the virtual key code of the key\*/

asc = inRec.Event.KeyEvent.uChar.AsciiChar; /\*Get the ASC code of the key\*/

/\* Handling of system shortcut keys \*/

if (vkc == 112) /\*if the F1 key is pressed\*/

{

if (gp\_top\_layer->LayerNo != 0) /\*If a submenu pops up\*/

{

PopOff(); /\*Close the pop-up submenu\*/

gi\_sel\_sub\_menu = 0; /\*Set the item number of the selected submenu item to 0\*/

}

bRet = ExeFunction(5, 1); /\*run help topic function\*/

}

else if (inRec.Event.KeyEvent.dwControlKeyState

& (LEFT\_ALT\_PRESSED | RIGHT\_ALT\_PRESSED))

{ /\* If the left or right Alt key is pressed \*/

switch (vkc) /\*judgment key combination Alt+letter\*/

{

case 88: /\*Alt+X exit\*/

if (gp\_top\_layer->LayerNo != 0)

{ PopOff(); gi\_sel\_sub\_menu = 0; }

bRet = ExeFunction(1,4); break;

case 70: /\*Alt+F\*/ PopMenu(1); break;

case 77: /\*Alt+M\*/ PopMenu(2); break;

case 81: /\*Alt+Q\*/ PopMenu(3); break;

case 83: /\*Alt+S\*/ PopMenu(4); break;

case 72: /\*Alt+H\*/ PopMenu(5); break;

}

}

else if (asc == 0) /\*handling of other control keys\*/

{

if (gp\_top\_layer->LayerNo == 0) /\*If no submenu pops up\*/

{

switch (vkc) /\* handle the arrow keys (left, right, down), do not respond to other control keys \*/

{

case 37:

gi\_sel\_menu--;

if (gi\_sel\_menu == 0)

{ gi\_sel\_menu = 5; }

TagMainMenu(gi\_sel\_menu); break;

case 39:

gi\_sel\_menu++;

if (gi\_sel\_menu == 6)

{ gi\_sel\_menu = 1; }

TagMainMenu(gi\_sel\_menu); break;

case 40:

PopMenu(gi\_sel\_menu);

TagSubMenu(1); break;

}

}

else /\*When the submenu has popped up\*/

{

for (loc=0,i=1; i<gi\_sel\_menu; i++)

{

loc += ga\_sub\_menu\_count[i-1];

} /\* Calculate the position (subscript) of the first item in the submenu in the submenu string array\*/

switch (vkc) /\*Arrow keys (left, right, up, down) processing\*/

{

case 37:

gi\_sel\_menu--;

if (gi\_sel\_menu < 1)

{ gi\_sel\_menu = 5; }

TagMainMenu(gi\_sel\_menu);

PopOff(); PopMenu(gi\_sel\_menu);

TagSubMenu(1); break;

case 38:

num = gi\_sel\_sub\_menu - 1;

if (num < 1)

{

num = ga\_sub\_menu\_count[gi\_sel\_menu-1];

}

if (strlen(ga\_sub\_menu[loc+num-1]) == 0)

{ num--; }

TagSubMenu(num); break;

case 39:

gi\_sel\_menu++;

if (gi\_sel\_menu > 5)

{ gi\_sel\_menu = 1; }

TagMainMenu(gi\_sel\_menu);

PopOff(); PopMenu(gi\_sel\_menu);

TagSubMenu(1); break;

case 40:

num = gi\_sel\_sub\_menu + 1;

if (num > ga\_sub\_menu\_count[gi\_sel\_menu-1])

{ num = 1; }

if (strlen(ga\_sub\_menu[loc+num-1]) == 0)

{ num++; }

TagSubMenu(num); break;

}

}

}

else if ((asc-vkc == 0) || (asc-vkc == 32)){ /\*Press a normal key\*/

if (gp\_top\_layer->LayerNo == 0) /\*If no submenu pops up\*/

{

switch (vkc)

{

case 70: /\*f or F\*/ PopMenu(1); break;

case 77: /\*m or M\*/ PopMenu(2); break;

case 81: /\*q or Q\*/ PopMenu(3); break;

case 83: /\*s or S\*/ PopMenu(4); break;

case 72: /\*h or H\*/ PopMenu(5); break;

case 13: /\*Enter\*/

PopMenu(gi\_sel\_menu);

TagSubMenu(1); break;

}

}

else /\*Keyboard input processing when the submenu has popped up\*/

{

if (vkc == 27) /\*if the ESC key is pressed\*/

{

PopOff(); gi\_sel\_sub\_menu = 0;

}

else if(vkc == 13) /\*if enter is pressed\*/

{

num = gi\_sel\_sub\_menu;

PopOff(); gi\_sel\_sub\_menu = 0;

bRet = ExeFunction(gi\_sel\_menu, num);

}

else /\* other common key processing \*/

{

/\* Calculate the position (subscript) of the first item in the submenu in the submenu string array\*/

for (loc=0,i=1; i<gi\_sel\_menu; i++)

{

loc += ga\_sub\_menu\_count[i-1];

}

/\* Compare with the representative characters of each item in the current submenu in turn\*/

for (i=loc; i<loc+ga\_sub\_menu\_count[gi\_sel\_menu-1]; i++)

{

if (strlen(ga\_sub\_menu[i])>0 && vkc==ga\_sub\_menu[i][1])

{ /\* if the match is successful \*/

PopOff(); gi\_sel\_sub\_menu = 0;

bRet = ExeFunction(gi\_sel\_menu, i-loc+1);

}

}

}

}

}

}

}

}

/\*\*

\* Function name: ExeFunction

\* Function function: Execute the function determined by the main menu number and submenu number.

\* Input parameters: m main menu item number

\* s submenu item number

\* Output parameters: none

\* Return value: BOOL type, TRUE or FALSE

\*

\* Call description: Only when the function ExitSys is executed, it is possible to return FALSE, and it always returns TRUE in other cases

\*/

BOOL ExeFunction(int m, int s)

{

BOOL bRet = TRUE;

/\*Array of function pointers, used to store the entry addresses of all function functions\*/

BOOL (\*pFunction[ga\_sub\_menu\_count[0]+ga\_sub\_menu\_count[1]+ga\_sub\_menu\_count[2]+ga\_sub\_menu\_count[3]+ga\_sub\_menu\_count[4]])(void);

int i, loc;

/\*Save the entry address of the function function into the array element corresponding to the subscript of the main menu number and submenu number where the function function is located\*/

pFunction[0] = FileOut;

pFunction[1] = FileIn;

pFunction[2] = ExitSys;

pFunction[3] = MaintainTeacherInfo;

pFunction[4] = MaintainRoomInfo;

pFunction[5] = MaintainGroupInfo;

pFunction[6] = MaintainCourseInfo;

pFunction[7] = MaintainClassInfo;

pFunction[8] = SeekTeacher;

pFunction[9] = SeekRoom;

pFunction[10] = SeekGroup;

pFunction[11] = SeekCourse;

pFunction[12] = SeekClass;

pFunction[13] = RoomUsage;

pFunction[14] = TeacherAnalysis;

pFunction[15] = ClassAnalysis;

pFunction[16] = HelpTopic;

pFunction[17] = NULL;

pFunction[18] = AboutDorm;

for (i=1,loc=0; i<m; i++) /\*Calculate the corresponding subscript according to the main menu number and submenu number\*/

{ loc += ga\_sub\_menu\_count[i-1]; }

loc += s - 1;

if (pFunction[loc] != NULL)

{

bRet = (\*pFunction[loc])(); /\*use the function pointer to call the pointed function\*/

}

return bRet;

}

/\*\*

\* Function name: CloseSys

\* Function function: shut down the system.

\* Input parameters: hd main chain head pointer

\* Output parameters: none

\* Return value: None

\*

\* Call description:

\*/

void CloseSys(void)

{

ClearMemory(); ClearScreen(); /\*clear screen\*/

/\*Release the dynamic storage area for storing information such as menu bar, status bar, gender code and student category code\*/

free(gp\_buff\_menubar\_info);

free(gp\_buff\_stateBar\_info);

/\*Close standard input and output device handles\*/

CloseHandle(gh\_std\_out); CloseHandle(gh\_std\_in);

/\* Set the window title bar to the end of the run \*/

SetConsoleTitle(" End of operation");

return;

}

/\*\*

\* Function name: ExitSys

\* Function function: pop up the option to exit the system

\* Input parameters: None

\* Output parameters: none

\* Return value: BOOL type, exit is TRUE, cancel is FALSE

\*

\* Call description:

\*/

BOOL ExitSys(void)

{

char \*pCh[] = {"Are you sure to exit the system?", "OK", "Cancel"};

BOOL bRet = Message(pCh);

PopOff();

return bRet;

}

/\*\*

\* Function name: ClearMemory

\* Function function: clear the memory

\* Input parameters: None

\* Output parameters: none

\* Return value: None

\*

\* Call description:

\*/

void ClearMemory(void){

for(i=0;i<telist->length;i++){

for(j=0;j<6;j++)

for(pocc=telist->elem[i].ochead[j].head;pocc!=NULL;pocc=telist->elem[i].ochead[j].head){

telist->elem[i].ochead[j].head=pocc->next; free(pocc); pocc=NULL;

}

}

free(telist->elem); telist->elem=NULL; telist->length=0;

for(i=0;i<rolist->length;i++){

for(j=0;j<6;j++)

for(pocc=rolist->elem[i].ochead[j].head;pocc!=NULL;pocc=rolist->elem[i].ochead[j].head){

rolist->elem[i].ochead[j].head=pocc->next; free(pocc); pocc=NULL;

}

}

free(rolist->elem); rolist->elem=NULL; rolist->length=0;

for(i=0;i<cougraph->length;i++){ //Delete all prerequisite course nodes

for(pcarc=cougraph->coulist[i].archhead;pcarc!=NULL;pcarc=cougraph->coulist[i].archhead){

cougraph->coulist[i].archhead=pcarc->nextarc;

free(pcarc->prename); free(pcarc);

}

}

free(cougraph->coulist); cougraph->coulist=NULL; cougraph->length=0;

for(i=0;i<3;i++){

for(psch=grade[i].schead;psch!=NULL;psch=grade[i].schead){

for(j=0;j<6;j++){

for(pcls=psch->week[j].classhead;pcls!=NULL;pcls=psch->week[j].classhead){

psch->week[j].classhead=pcls->next; free(pcls); pcls=NULL;

}

for(pocc=psch->week[j].ochead.head;pocc!=NULL;pocc=psch->week[j].ochead.head){

psch->week[j].ochead.head=pocc->next; free(pocc); pcls=NULL;

}

}

grade[i].schead=psch->next; free(psch); psch=NULL;

}

}

}

BOOL SaveData(void)

{

BOOL bRet = TRUE;

char \*plabel\_name[] = {"Main menu item: File",

"Submenu item: Data Save",

"confirm"

};

ShowModule(plabel\_name, 3);

return bRet;

}

BOOL Initial(void){ //pass

//Initialize classroom, teacher, course list structure

telist=(TeList \*)malloc(sizeof(TeList));

rolist=(RoList \*)malloc(sizeof(RoList));

cougraph = (CouGraph \*)malloc(sizeof(CouGraph));

//Initialize classroom, teacher, course linear lookup table

telist->elem = (Teacher \*)malloc(LIST\_INIT\_SIZE\*sizeof(Teacher));

rolist->elem = (Room \*)malloc(LIST\_INIT\_SIZE\*sizeof(Room));

cougraph->coulist = (CouHead \*)malloc(LIST\_INIT\_SIZE\*sizeof(CouHead));

if(!(telist->elem && rolist->elem && cougraph->coulist)) return FALSE;

//The length of each table is set to 0, and the pointer is set to NULL

telist->length = rolist->length = cougraph->length = 0;

telist->listsize = rolist->listsize = cougraph->listsize = LIST\_INIT\_SIZE;

cougraph->length = 0;

grade[0].gr\_num=grade[1].gr\_num=grade[2].gr\_num=0;

grade[0].schead = grade[1].schead = grade[2].schead =NULL;

printf("System initialization successful");

getchar(); ClearScreen();

return TRUE;

}

BOOL AddTeacher(void){ //pass

fflush(stdin);

printf(" Enter the teacher's name:");

scanf("%s", cbuffer); getchar();

for(i=0;i<telist->length;i++)

if(!strcmp(telist->elem[i].tname,cbuffer)){

printf("The teacher already exists");

getchar(); ClearScreen(); return TRUE;

}

AddNewTeacher(cbuffer);

printf("Enter gender (male--1\tfemale--0):");

scanf("%hu",&telist->elem[telist->length].gender); getchar();

printf("Enter age:");

scanf("%hu",&telist->elem[telist->length].age); getchar();

telist->length++; printf("insert successfully");

getchar(); ClearScreen();

return TRUE;

}

BOOL AddRoom(void){ //pass

fflush(stdin);

printf("Please enter the classroom number:");

scanf("%s", cbuffer); getchar();

for(i=0;i<rolist->length;i++)

if(!strcmp(rolist->elem[i].code,cbuffer)){

printf("This classroom already exists");

getchar(); ClearScreen(); return TRUE;

}

AddNewRoom(cbuffer);

printf("Enter capacity:");

scanf("%hu",&rolist->elem[rolist->length].rsize); getchar();

rolist->length++; printf("insert successfully");

getchar(); ClearScreen();

return TRUE;

}

BOOL AddCourse(void){ //pass

fflush(stdin);

printf("Please enter the course name:");

scanf("%s", cbuffer); getchar();

for(i=0;i<cougraph->length;i++)

if(!strcmp(cougraph->coulist[i].name,cbuffer)){

printf("This class already exists");

getchar(); ClearScreen(); return TRUE;

}

AddNewCourse(cbuffer);

printf("Enter the number of prerequisite courses:");

scanf("%hu", &(cougraph->coulist[cougraph->length].arcnum)); getchar();

printf("Enter each prerequisite course:");

for(j=0;j<cougraph->coulist[cougraph->length].arcnum;j++){

scanf("%s", cbuffer); getchar();

for(i=0;i<cougraph->length;i++)

if(!strcmp(cougraph->coulist[i].name, cbuffer)){ //Find the course in the course list

pcarc = (CouArc \*)malloc(sizeof(CouArc));

pcarc->prename=(char \*)malloc(sizeof(cbuffer)\*sizeof(char));

strcpy(pcarc->prename, cbuffer);

pcarc->nextarc = cougraph->coulist[cougraph->length].archhead;

cougraph->coulist[cougraph->length]. archhead = pcarc;

}

else printf("There is no such course in the list, give up its operation");

}

cougraph->length++; printf("insert successfully");

getchar(); ClearScreen();

return TRUE;

}

BOOL AddGroup(void){ //pass

char cbuf3[20];

fflush(stdin);

printf("Enter class number:");

psch=ClassInput(cbuf3);

if(psch!=NULL){

printf("This class already exists");

getchar(); ClearScreen(); return TRUE;

}

//Create the class node if it does not exist

psch=(Schedule \*)malloc(sizeof(Schedule));

strcpy(psch->classname, cbuf3);

printf("Enter the number of courses taken:");

scanf("%hu",&psch->do\_num); getchar();

psch->pr\_num=0; //Courses in progress cannot be entered manually, and need to be imported together with new courses

printf("Enter the courses you have taken:\n");

for(i=0;i<psch->do\_num;i++){

scanf("%s",psch->done[i]); getchar();

}

printf("Enter the number of people:");

scanf("%hu",&psch->clsize); getchar();

for(i=0;i<6;i++){ //New class class structure initialization

psch->week[i].classhead=NULL;

psch->week[i].cl\_num=0;

psch->week[i].ochead.head=NULL;

psch->week[i].ochead.len=0;

}

// Insert new classes into the class-based curriculum by grade

if(strstr(cbuf3,"15")){

psch->next=grade[0].schead; grade[0].schead=psch;

grade[0].gr\_num++;

}

else if(strstr(cbuf3,"16")){

psch->next=grade[1].schead; grade[1].schead=psch;

grade[1].gr\_num++;

}

else if(strstr(cbuf3,"17")){

psch->next=grade[2].schead; grade[2].schead=psch;

grade[2].gr\_num++;

}

printf("insert successfully");

getchar(); ClearScreen();

return TRUE;

}

BOOL DelTeacher(void){

fflush(stdin);

if(telist->length==0){

printf("Teacher list is empty");

getchar(); ClearScreen(); return TRUE;

}

printf("Enter the name of the teacher to be deleted:");

scanf("%s", cbuffer); getchar();

// Locate the target teacher and confirm its existence

for(i=0;i<telist->length;i++){

if(!strcmp(telist->elem[i].tname,cbuffer)) break;

}

if(i==telist->length){

printf("No corresponding teacher found");

getchar(); ClearScreen(); return TRUE;

}

//Delete the time occupancy linked list under the target teacher structure

for(j=0;j<6;j++)

for(pocc=telist->elem[i].ochead[j].head;pocc!=NULL;pocc=telist->elem[i].ochead[j].head){

telist->elem[i].ochead[j].head = pocc->next; free(pocc);

}

//Delete all classes with corresponding teachers

for(k=0;k<3;k++)

for(psch=grade[k].schead;psch!=NULL;psch=psch->next)

for(j=0;j<6;j++)

for(pcls=psch->week[j].classhead;pcls!=NULL;pcls=pcls->next)

if(!strcmp(telist->elem[i].tname,pcls->tname)){ //Check whether the teacher's name matches

strcpy(cbuf1, pcls->roomcode);

memcpy(buf3, pcls->wk\_st, sizeof(buf3)); memcpy(buf4, pcls->wk\_ed, sizeof(buf4));

buf5=pcls->cou\_st; buf6=pcls->cou\_ed; //Copy the course node information into the temporarily created course node

//Delete the relevant time period in the subordinate array of the class

DelTimeNode(pcls->wk\_st, pcls->wk\_ed, pcls->cou\_st, pcls->cou\_ed, &(psch->week[j].ochead));

SeekRemClass(psch, cbuffer); //Check whether the target course still has a class under this class

//Remove seat time in classroom structure

for(l=0;l<rolist->length;l++)

if(!strcmp(rolist->elem[l].code, cbuf1)) break;

if(l!=rolist->length)

DelTimeNode(buf3,buf4,buf5,buf6,&(rolist->elem[l].ochead[j]));

DelClassNode(pcls, psch, (unsigned short)j); //delete course node

}

//delete target teacher

for(j=i;j<telist->length-1;j++)

telist->elem[j]=telist->elem[j+1];

telist->length--; printf("Deleted successfully");

if(telist->length==0) printf("There is no teacher in the table");

getchar(); ClearScreen();

return TRUE;

}

BOOL DelRoom(void){

fflush(stdin);

if(rolist->length==0){

printf("Classroom list is empty");

getchar(); ClearScreen(); return TRUE;

}

printf("Please enter the classroom number to be deleted:");

scanf("%s", cbuffer); getchar();

// Locate the target classroom and confirm its existence

for(i=0;i<rolist->length;i++){

if(!strcmp(rolist->elem[i].code,cbuffer)) break;

}

if(i==rolist->length){

printf("Could not find the corresponding classroom");

getchar(); ClearScreen(); return TRUE;

}

//Delete all prerequisite nodes

for(j=0;j<6;j++)

for(pocc=rolist->elem[i].ochead[j].head;pocc!=NULL;pocc=rolist->elem[i].ochead[j].head){

rolist->elem[i].ochead[j].head = pocc->next; free(pocc);

}

//Delete all classes that have corresponding classrooms

for(k=0;k<3;k++)

for(psch=grade[k].schead;psch!=NULL;psch=psch->next)

for(j=0;j<6;j++)

for(pcls=psch->week[j].classhead;pcls!=NULL;pcls=pcls->next)

if(!strcmp(rolist->elem[i].code,pcls->roomcode)){

strcpy(cbuf2, pcls->tname); //Copy the course node information into the temporarily created course node

memcpy(buf3, pcls->wk\_st, sizeof(buf3)); memcpy(buf4, pcls->wk\_ed, sizeof(buf4));

buf5=pcls->cou\_st; buf6=pcls->cou\_ed;

//Delete the relevant time period in the subordinate array of the class

DelTimeNode(pcls->wk\_st, pcls->wk\_ed, pcls->cou\_st, pcls->cou\_ed, &(psch->week[j].ochead));

SeekRemClass(psch, cbuffer); //Check whether the target course still has a class under this class

// remove seat time in teacher struct

for(l=0;l<telist->length;l++)

if(!strcmp(telist->elem[l].tname, cbuf2)) break;

if(l!=telist->length)

DelTimeNode(buf3,buf4,buf5,buf6,&(telist->elem[l].ochead[j]));

DelClassNode(pcls, psch, (unsigned short)j); //delete course node

}

for(j=i;j<rolist->length-1;j++) //Delete classroom structure

rolist->elem[j]=rolist->elem[j+1];

rolist->length--; printf("delete successfully");

if(rolist->length==0) printf("There is no classroom in the list");

getchar(); ClearScreen();

return TRUE;

}

BOOL DelCourse(void){

fflush(stdin);

if(cougraph->length==0){

printf("Course list is empty");

getchar(); ClearScreen(); return TRUE;

}

CouArc \*rcarc;

printf("Enter the name of the course to be deleted:");

scanf("%s", cbuffer); getchar();

// Locate the target course and confirm its existence

for(i=0;i<cougraph->length;i++)

if(!strcmp(cougraph->coulist[i].name,cbuffer)) break;

if(i==cougraph->length){

printf("Correct course not found");

getchar(); ClearScreen(); return TRUE;

}

//If the prerequisite course of a course in the course list is the target course, delete the target course node, and add the prerequisite course of the target course as the new prerequisite course

for(j=0; j<cougraph->length; j++){

if(j==i) continue;

qcarc = cougraph->coulist[j].archhead;

while(qcarc){ // There are arcs or edges

if(!strcmp(qcarc->prename, cbuffer)){ // The arc is pointed by the node to be deleted

pcarc = cougraph->coulist[i].archhead;

while(pcarc){ // Point the prerequisite course of the prerequisite course to the current course

rcarc = (CouArc \*)malloc(sizeof(CouArc));

strcpy(rcarc->prename, pcarc->prename);

rcarc->nextarc = cougraph->coulist[j].archhead;

cougraph->coulist[j].archhead = rcarc;

pcarc = pcarc->nextarc;

}

rcarc = cougraph->coulist[j].archhead;

while(rcarc->nextarc!=qcarc) rcarc=rcarc->nextarc; // delete the arc whose head is the specified course

rcarc->nextarc = qcarc->nextarc;

free(qcarc->prename); free(qcarc);

qcarc = rcarc->nextarc;

} } }

cougraph->length--; // The length of the course list is reduced

for(pcarc=cougraph->coulist[i].archhead;pcarc!=NULL;pcarc=cougraph->coulist[i].archhead){ // Delete all prerequisite courses of the target course

cougraph->coulist[i].archhead=pcarc->nextarc;

free(pcarc->prename); free(pcarc);

}

//Delete all classes with corresponding courses

for(k=0;k<3;k++)

for(psch=grade[k].schead;psch!=NULL;psch=psch->next){

for(j=0;j<psch->pr\_num;j++)

if(!strcmp(cougraph->coulist[i].name,psch->progress[j])) break;

if(j==psch->pr\_num) continue;

for(j=0;j<6;j++)

for(pcls=psch->week[j].classhead;pcls!=NULL;pcls=pcls->next)

if(!strcmp(cougraph->coulist[i].name,pcls->coursename)){

strcpy(cbuf1, pcls->roomcode); strcpy(cbuf2, pcls->tname); //Copy the course node information into the temporarily created course node

memcpy(buf3, pcls->wk\_st, sizeof(buf3)); memcpy(buf4, pcls->wk\_ed, sizeof(buf4));

buf5=pcls->cou\_st; buf6=pcls->cou\_ed;

//Delete the relevant time period in the subordinate array of the class

DelTimeNode(pcls->wk\_st, pcls->wk\_ed, pcls->cou\_st, pcls->cou\_ed, &(psch->week[j].ochead));

SeekRemClass(psch, cbuffer); //Check whether the target course still has a class under this class

//Delete the target course occupancy time in the classroom structure

for(l=0;l<rolist->length;l++)

if(!strcmp(rolist->elem[l].code, cbuf1)) break;

if(l!=rolist->length)

DelTimeNode(buf3,buf4,buf5,buf6,&(rolist->elem[l].ochead[j]));

// remove seat time in teacher struct

for(l=0;l<telist->length;l++)

if(!strcmp(telist->elem[l].tname, cbuf2)) break;

if(l!=telist->length)

DelTimeNode(buf3,buf4,buf5,buf6,&(telist->elem[l].ochead[j]));

//delete target course

DelClassNode(pcls, psch, (unsigned short)j); //delete course node

}

}

for(j=i; j<cougraph->length-1; j++) // The vertex behind vertex v moves forward

cougraph->coulist[j] = cougraph->coulist[j+1];

cougraph->length--; printf(" delete successfully");

if(cougraph->length==0) printf(" There is no vertex in the graph");

getchar(); ClearScreen();

return TRUE;

}

BOOL DelGroup(void){ //pass

char cbuf3[20];

fflush(stdin);

Schedule \*psch=NULL, \*qsch=NULL;

Class \*pcls=NULL;

printf("Will delete all classes under the class\nContinue? Yes--Y\tNo--any other key\n");

j=getchar(); getchar();

if(j!='y' && j!='Y'){

ClearScreen(); return TRUE;

}

printf("Please enter the name of the class to be deleted:");

psch=ClassInput(cbuf3);

if(psch==NULL){

printf("This class does not exist");

getchar(); ClearScreen(); return TRUE;

}

for(i=0;i<6;i++){

for(pcls=psch->week[i].classhead;pcls!=NULL;pcls=psch->week[i].classhead){ //Delete all classes under the class

psch->week[i].classhead=pcls->next; free(pcls);

}

for(pocc=psch->week[i].ochead.head;pocc!=NULL;pocc=psch->week[i].ochead.head){ //Delete all time-occupied nodes under the class

psch->week[i].ochead.head=pocc->next; free(pocc);

}

}

// Find the class node according to the grade and delete it

if(strstr(cbuf3,"15")){

if(psch==grade[0].schead){ //The target class is the head node

grade[0].schead=psch->next; free(psch);

}

else{ //target class non-head node

for(qsch=grade[0].schead;qsch->next!=psch;qsch=qsch->next) ;

qsch->next=psch->next; free(psch);

}

grade[0].gr\_num--;

if(grade[0].gr\_num==0) printf("2015 grade has no class");

}

else if(strstr(cbuf3,"16")){ //The target class is the head node

if(psch==grade[1].schead){

grade[1].schead=psch->next; free(psch);

}

else{ //target class non-head node

for(qsch=grade[1].schead;qsch->next!=psch;qsch=qsch->next) ;

qsch->next=psch->next; free(psch);

}

grade[1].gr\_num--;

if(grade[1].gr\_num==0) printf("2016 grade has no class");

}

else if(strstr(cbuf3,"17")){ //The target class is the head node

if(psch==grade[2].schead){

grade[2].schead=psch->next; free(psch);

}

else{ //target class non-head node

for(qsch=grade[2].schead;qsch->next!=psch;qsch=qsch->next) ;

qsch->next=psch->next; free(psch);

}

grade[2].gr\_num--;

if(grade[2].gr\_num==0) printf("2017 grade has no class");

}

printf("Deleted successfully");

getchar(); ClearScreen();

return TRUE;

}

BOOL AltTeacher(void){ //pass

fflush(stdin);

if(telist->length==0){

printf("Teacher list is empty");

getchar(); ClearScreen(); return TRUE;

}

printf("Please enter the name of the teacher to be changed:");

scanf("%s", cbuffer); getchar();

for(i=0;i<telist->length;i++)

if(!strcmp(telist->elem[i].tname,cbuffer)) break; //i saves the position of the target teacher in the list

if(i==telist->length){

printf("No corresponding teacher found");

getchar(); ClearScreen(); return TRUE;

}

count=0;

//Choose the modification method

do {

if(count++) printf("The option is out of bounds, re-enter");

printf("What information to modify?\n1--Name\t2--Gender\t3--Age\t4--Back to the main menu\n");

scanf("%hu", &choice); getchar();

if(choice==4){ ClearScreen(); return TRUE; }

}while(choice<1 || choice>4);

switch(choice){

case 1:

printf("Please enter the new teacher's name:");

scanf("%s", cbuffer); getchar();

//Verify the legitimacy of the new name

for(j=0;j<telist->length;j++){

if(j==i) continue;

if(!strcmp(telist->elem[j].tname,cbuffer)){

printf("The new teacher already exists, the operation will stop and return to the main menu");

getchar(); ClearScreen(); return TRUE;

}

}

for(k=0;k<3;k++) //Modify all classes with corresponding teachers

for(psch=grade[k].schead;psch!=NULL;psch=psch->next)

for(j=0;j<6;j++)

for(pcls=psch->week[j].classhead;pcls!=NULL;pcls=pcls->next) //Find the class and modify it

if(!strcmp(telist->elem[i].tname,pcls->tname)) strcpy(pcls->tname,cbuffer);

strcpy(telist->elem[i].tname, cbuffer);

printf("modified successfully");

break;

case 2:

// There are only two possibilities for gender, just change each other

if(telist->elem[i].gender==0) telist->elem[i].gender=1;

else telist->elem[i].gender=0;

printf("Gender modified successfully, the current gender is");

if(telist->elem[i].gender==0) printf("female");

else printf("male");

break;

case 3:

printf("Please enter the new age:");

scanf("%hu", &telist->elem[i].age); getchar();

printf("modified successfully");

case 4:

break;

}

getchar(); ClearScreen();

return TRUE;

}

BOOL AltRoom(void){ //pass

fflush(stdin);

if(rolist->length==0){

printf("Classroom list is empty");

getchar(); ClearScreen(); return TRUE;

}

printf("Please enter the classroom number to be changed:");

scanf("%s", cbuffer); getchar();

//Find the target classroom

for(i=0;i<rolist->length;i++)

if(!strcmp(rolist->elem[i].code,cbuffer)) break; //i saves the position of the target classroom in the list

if(i==rolist->length){

printf("Could not find the corresponding classroom");

getchar(); ClearScreen(); return TRUE;

}

//Choose the modification method

count = 0;

do {

if(count) printf("The option is out of bounds, re-enter");

printf("What information to modify?\n1--Number\t2--Capacity\t3--Back to the main menu\n");

scanf("%hu", &choice); getchar(); count++;

if(choice==3){ ClearScreen(); return TRUE; }

}while(choice<1 || choice>3);

switch(choice){

case 1:

printf("Please enter the new classroom number:");

scanf("%s", cbuffer); getchar();

// Check the validity of the new number

for(j=0;j<rolist->length;j++){

if(j==i) continue;

if(!strcmp(rolist->elem[j].code,cbuffer)){

printf("The target classroom already exists, the operation will stop and return to the main menu");

getchar(); ClearScreen(); return TRUE;

}

}

for(k=0;k<3;k++) //Modify all classes with corresponding teachers

for(psch=grade[k].schead;psch!=NULL;psch=psch->next)

for(j=0;j<6;j++)

for(pcls=psch->week[j].classhead;pcls!=NULL;pcls=pcls->next)

if(!strcmp(rolist->elem[i].code,pcls->roomcode)) strcpy(pcls->roomcode,cbuffer);

//Modify the classroom node

strcpy(rolist->elem[i].code, cbuffer);

printf("modified successfully");

break;

case 2:

printf("Please enter the new capacity:");

scanf("%hu", &rolist->elem[i].rsize); getchar();

printf("modified successfully");

case 3:

break;

}

getchar(); ClearScreen();

return TRUE;

}

BOOL AltCourse(void){ //pass

fflush(stdin);

if(cougraph->length==0){

printf("Course list is empty");

getchar(); ClearScreen(); return TRUE;

}

printf("Please enter the name of the course to be changed:");

scanf("%s", cbuffer); getchar();

//Find the target course

for(i=0;i<cougraph->length;i++)

if(!strcmp(cougraph->coulist[i].name,cbuffer)) break; //i saves the position of the target course in the list, and i cannot be used as an accumulator afterwards

if(i==cougraph->length){

printf("Correct course not found");

getchar(); ClearScreen(); return TRUE;

}

//Choose the modification method

count = 0;

do {

if(count++) printf("The option is out of bounds, re-enter");

printf("What information to modify?\n1--course name\t2--add prerequisite courses\t3--delete prerequisite courses\t4--return to menu\n");

scanf("%hu", &choice); getchar();

if(choice==4){ ClearScreen(); return TRUE; }

}while(choice<1 || choice>4);

switch(choice){

case 1:

printf("Enter a new name:");

scanf("%s", cbuffer); getchar();

// Check the validity of the new class name

for(j=0;j<cougraph->length;j++){

if(j==i) continue;

if(!strcmp(cougraph->coulist[j].name,cbuffer)){

printf("This class already exists");

getchar(); ClearScreen(); return TRUE;

}

}

for(j=0;j<cougraph->length;j++){

if(j==i) continue;

for(pcarc=cougraph->coulist[j].archhead;pcarc!=NULL;pcarc=pcarc->nextarc) //Modify all corresponding prerequisite course arc nodes

if(!strcmp(cougraph->coulist[i].name,pcarc->prename))

strcpy(pcarc->prename,cbuffer);

}

for(k=0;k<3;k++) //Modify all classes with corresponding courses

for(psch=grade[k].schead;psch!=NULL;psch=psch->next)

for(j=0;j<6;j++)

for(pcls=psch->week[j].classhead;pcls!=NULL;pcls=pcls->next)

if(!strcmp(cougraph->coulist[i].name,pcls->coursename)) strcpy(pcls->coursename,cbuffer);

//Modify the target course node

strcpy(cougraph->coulist[i].name, cbuffer);

printf("modified successfully");

break;

case 2:

do {

printf("Enter the name of the new prerequisite course:");

scanf("%s", cbuffer); getchar();

for(j=0;j<cougraph->length;j++){ //Find the corresponding course in the course lookup table

if(j==i) continue;

if(!strcmp(cougraph->coulist[j].name, cbuffer)) break;

}

if(j==cougraph->length){

printf("Target course does not exist");

getchar(); ClearScreen(); return TRUE;

}

pcarc = cougraph->coulist[i].archhead;

while(pcarc){ //Check if the course is an existing prerequisite course

if(!strcmp(pcarc->prename, cbuffer)){

printf("The prerequisite course already exists");

getchar(); ClearScreen(); return TRUE;

}

pcarc = pcarc->nextarc;

}

pcarc = (CouArc \*)malloc((sizeof(CouArc))); //Insert the pre-requisite course arc node

pcarc->prename=(char \*)malloc(sizeof(cbuffer));

strcpy(pcarc->prename, cbuffer);

//The new node is inserted into the prerequisite course linked list

pcarc->nextarc = cougraph->coulist[i].archhead;

cougraph->coulist[i].archhead = pcarc;

cougraph->coulist[i].arcnum++;

printf("Added successfully\nContinue? Yes--Y\tNo--other keys");

j=getchar(); getchar();

}while(j=='Y' || j=='y');

break;

case 3:

if(cougraph->coulist[i].arcnum==0){

printf("This course has no prerequisites");

getchar(); ClearScreen(); return TRUE;

}

printf("Enter the name of the pre-requisite course to be deleted:");

scanf("%s", cbuffer); getchar();

pcarc = cougraph->coulist[i].archhead;

while(pcarc){ //Check if the course is an existing prerequisite course

if(!strcmp(pcarc->prename, cbuffer)) break;

pcarc = pcarc->nextarc;

}

if(!pcarc){

printf("Correct course not found");

getchar(); ClearScreen(); return TRUE;

}

if(pcarc == cougraph->coulist[i].archhead){ //The node to be deleted is the head node

cougraph->coulist[i].archhead = pcarc->nextarc;

free(pcarc->prename); free(pcarc);

}

else{ //The node to be deleted is not the head node

for(qcarc=cougraph->coulist[i].archhead;qcarc->nextarc!=pcarc;qcarc=qcarc->nextarc) ; //q moves to the predecessor of p

qcarc->nextarc = pcarc->nextarc; free(pcarc->prename); free(pcarc);

}

cougraph->coulist[i].arcnum--;

printf("Deleted successfully");

break;

case 4:

return TRUE;

}

getchar(); ClearScreen();

return TRUE;

}

BOOL AltGroup(void){ //pass

char cbuf3[20];

fflush(stdin);

printf("Please enter the name of the class to be changed:");

psch=ClassInput(cbuf3);

if(psch==NULL){

printf("This class does not exist");

getchar(); ClearScreen(); return TRUE;

}

//Choose the modification method

count = 0;

do {

if(count) printf("The option is out of bounds, re-enter");

printf("What information to modify?\n1--class number\t2--add course taken\t3--modify course taken\t4--delete course taken\t5--number of people\t6--return menu\n ");

scanf("%hu", &hubuffer); getchar(); count++;

if(choice==6){ ClearScreen(); return TRUE; }

}while(hubuffer<1 || hubuff>6);

switch(hubuffer){

case 1:

flag=TRUE;

do {

printf("Please enter the new class number:");

scanf("%s", cbuf2); getchar();

//Verify the legitimacy of the new class number

if((strstr(cbuf3,"15") && !strstr(cbuf2,"15")) || (strstr(cbuf3,"16") && !strstr(cbuf2,"16")) || (strstr(cbuf3 ,"17") && !strstr(cbuf2,"17"))){

printf(" The class number cannot be modified across grades\n"); flag=FALSE;

}

}while(flag==FALSE);

//Determine the verification head node according to the class number

if(strstr(cbuf2,"15")) qsch=grade[0].schead;

else if(strstr(cbuf2,"16")) qsch=grade[1].schead;

else if(strstr(cbuf2,"17")) qsch=grade[2].schead;

// Verify the existence of the new class number

for(;qsch!=NULL;qsch=qsch->next){

if(!strcmp(qsch->classname,cbuf2)){

printf("The target class already exists");

getchar(); ClearScreen(); return TRUE;

}

}

//Modify class number

strcpy(psch->classname, cbuf2);

printf("modified successfully");

break;

case 2:

printf("Please enter the new course name:");

scanf("%s", cbuffer); getchar();

//Verify the existence of the new course

for(i=0; i<psch->do\_num; i++)

if(!strcmp(psch->done[i], cbuffer)){

printf("This class already exists");

getchar(); ClearScreen(); return TRUE;

}

//add course

strcpy(psch->done[psch->do\_num], cbuffer);

psch->do\_num++;

printf("Added successfully");

break;

case 3:

printf("Enter the name of the course to be modified:");

scanf("%s", cbuffer); getchar();

// Check the existence of the course to be modified

for(i=0; i<psch->do\_num; i++)

if(!strcmp(psch->done[i], cbuffer)) break;

if(j==psch->do\_num){

printf("Correct course not found");

getchar(); ClearScreen(); return TRUE;

}

printf("Please enter the new course name:");

scanf("%s", cbuffer); getchar();

//Verify the legitimacy of the course name stored in the new class number

for(j=0;j<psch->do\_num;j++){

if(!strcmp(psch->done[j], cbuffer)){

printf("This class already exists");

getchar(); ClearScreen(); return TRUE;

}

}

//Modify the course

strcpy(psch->done[i], cbuffer);

printf("modified successfully");

break;

case 4:

if(psch->do\_num==0){

printf("No courses taken"); break;

}

printf("Please enter the courses to be deleted:");

scanf("%c", cbuffer); getchar();

// Check the existence of courses to be deleted

for(i=0; i<psch->do\_num; i++)

if(!strcmp(psch->done[i], cbuffer)) break;

if(i==psch->do\_num){

printf("Correct course not found");

getchar(); ClearScreen(); return TRUE;

}

//delete course

for(j=i;j<psch->do\_num-1;j++)

strcpy(psch->done[j], psch->done[j+1]);

psch->do\_num--;

if(psch->do\_num==0) printf("There are no courses in the table");

printf("deleted successfully"); getchar();

break;

case 5:

printf(" Please enter the new number:");

scanf("%hu", &(psch->clsize)); getchar();

printf("modified successfully");

/\* Shield the interface of modifying the course in progress

case 5:

printf("Please enter the new course name:");

scanf("%s", cbuffer); getchar();

for(i=0; i<psch->pr\_num; i++){

if(!strcmp(psch->progress[i], cbuffer)){

printf("The class already exists"); return FALSE;

}

}

strcpy(psch->progress[psch->pr\_num], cbuffer);

psch->pr\_num++;

printf("Added successfully");

break;

case 6:

if(psch->pr\_num==0){

printf("There are no courses in progress"); break;

}

printf("Please enter the name of the course to be changed:");

scanf("%s", cbuffer); getchar();

for(i=0; i<psch->pr\_num; i++)

if(!strcmp(psch->progress[i],cbuffer)) break;

if(i==psch->pr\_num){

printf("Course not found"); return FALSE;

}

printf("Please enter the new course name:");

scanf("%s", cbuffer); getchar();

for(j=0;j<psch->pr\_num;j++){

if(!strcmp(psch->progress[j], cbuffer)){

printf("The class already exists"); return FALSE;

}

}

strcpy(psch->progress[i], cbuffer);

printf("modified successfully");

break;

case 7:

if(psch->pr\_num==0){

printf(" No courses in progress"); break;

}

printf("Please enter the name of the course to be deleted:");

scanf("%s", cbuffer); getchar();

for(i=0; i<psch->pr\_num; i++)

if(!strcmp(psch->progress[i],cbuffer)) break;

if(i==psch->pr\_num){

printf("Course not found"); return FALSE;

}

for(j=i;j<psch->pr\_num-1;j++)

strcpy(psch->progress[j], psch->progress[j+1]);

printf("Deleted successfully");

psch->pr\_num--;

if(psch->pr\_num==0) printf("There are no courses in the table");

break;

\*/

case 6:

break;

}

getchar(); ClearScreen();

return TRUE;

}

BOOL SeekTeacher(void){ //pass

fflush(stdin);

if(telist->length==0){

printf("Teacher list is empty");

getchar(); ClearScreen(); return TRUE;

}

count=0; flag=FALSE;

do {

if(count) printf("The option is out of bounds, re-enter");

printf("Search method:\n1--name\t2--age range\t3--gender\t4--return to main menu\n");

scanf("%hu", &choice); getchar(); count++;

if(choice==4){ ClearScreen(); return TRUE; }

}while(choice<1 || choice>4);

switch(choice){

case 1: //Search by name

printf("Please enter the teacher's name:");

scanf("%s", cbuffer); getchar();

for(i=0;i<telist->length;i++)

if(!strcmp(telist->elem[i].tname,cbuffer)) break;

if(i==telist->length){

printf("No corresponding teacher found");

getchar(); ClearScreen(); return TRUE;

}

printf("Gender:");

if(telist->elem[i].gender==0) printf("female");

else printf("male");

printf("Age: %hu\nfind success", telist->elem[i].age);

break;

case 2: //Search by age range

printf("Enter the minimum age limit:");

scanf("%hu", &buf1); getchar();

printf("Enter the upper limit of age:");

scanf("%hu", &buf2); getchar();

for(i=0;i<telist->length;i++)

if(telist->elem[i].age>=buf1 && telist->elem[i].age<=buf2){

printf("Name: %s\tAge: %hu\tGender:",telist->elem[i].tname,telist->elem[i].age);

flag=TRUE;

if(telist->elem[i].gender==0) printf("female\n");

else printf("Male\n");

}

if(flag==FALSE) printf("No qualified teacher"); getchar();

break;

case 3: //Search by gender

printf("Enter the target gender (male--1\tfemale--0):");

j=getchar(); getchar();

for(i=0;i<telist->length;i++){

if(j=='0'){

if(!telist->elem[i].gender){

printf("Name: %s\tAge:%hu\n",telist->elem[i].tname,telist->elem[i].age);

flag=TRUE; //flag mark output valid information

} }

else {

if(telist->elem[i].gender){

printf("Name: %s\tAge:%hu\n",telist->elem[i].tname,telist->elem[i].age);

flag=TRUE; //flag mark output valid information

} } }

if(flag==FALSE) printf("No qualified teachers"); getchar();

case 4:

break;

}

getchar(); ClearScreen();

return TRUE;

}

BOOL SeekRoom(void){ //pass

fflush(stdin);

if(rolist->length==0){

printf("Classroom list is empty");

getchar(); ClearScreen(); return TRUE;

}

count=0;

do {

if(count) printf("The option is out of bounds, re-enter");

printf("Search method:\n1--number\t2--capacity\t3--back to main menu\n");

scanf("%hu", &choice); getchar(); count++;

if(choice==3){ ClearScreen(); return TRUE; }

}while(choice<1 || choice>3);

switch(choice){

case 1: //Search by classroom number

printf("Please enter the classroom number:");

scanf("%s", cbuffer); getchar();

for(i=0;i<rolist->length;i++)

if(!strcmp(rolist->elem[i].code,cbuffer)) break;

if(i==rolist->length){

printf("Could not find the corresponding classroom");

getchar(); ClearScreen(); return TRUE;

}

printf("Classroom capacity: %hu\n searched successfully", rolist->elem[i].rsize);

break;

case 2: // Search by classroom capacity

flag=FALSE;

printf("Enter the lower limit of capacity:");

scanf("%hu", &buf1); getchar();

printf("Enter the upper limit of capacity:");

scanf("%hu", &buf2); getchar();

for(i=0;i<rolist->length;i++)

if(rolist->elem[i].rsize>=buf1 && rolist->elem[i].rsize<=buf2){

printf("Code: %s\tCapacity:%hu\n", rolist->elem[i].code, rolist->elem[i].rsize);

flag=TRUE; //flag mark output valid information

}

if(flag==FALSE) printf("There is no classroom that meets the requirements"); getchar();

case 3:

break;

}

getchar(); ClearScreen();

return TRUE;

}

BOOL SeekCourse(void){ //pass

fflush(stdin);

BOOL glag=FALSE;

if(cougraph->length==0){

printf("Course list is empty");

getchar(); ClearScreen(); return TRUE;

}

printf("Please enter the substring of the course name to be checked:");

scanf("%s", cbuffer); getchar();

for(i=0; i<cougraph->length; i++)

// fuzzy search

if(strstr(cougraph->coulist[i].name,cbuffer)){

printf("Course name: %s\n",cougraph->coulist[i].name);

flag = FALSE;

printf("Prerequisite courses:\n");

for(pcarc=cougraph->coulist[i].archhead;pcarc!=NULL;pcarc=pcarc->nextarc){

printf("%s\n", pcarc->prename);

flag = TRUE; //flag tag output valid information

}

if(flag==FALSE) printf("There is no prerequisite course for this course\n");

flag = FALSE;

printf("Successor course:\n");

for(j=0; j<cougraph->length; j++){

if(j==i) continue;

//Search method: find out whether the current course has a prerequisite course as the target course

for(pcarc=cougraph->coulist[j].archhead;pcarc!=NULL;pcarc = pcarc->nextarc)

if(!strcmp(pcarc->prename, cougraph->coulist[i].name)){

printf("%s\n", cougraph->coulist[j].name);

flag = TRUE; //flag tag output valid information

}

}

if(flag==FALSE) printf("There is no follow-up course for this course\n");

glag=TRUE;

}

if(glag==FALSE) printf("No corresponding course found");

getchar(); ClearScreen();

return TRUE;

}

BOOL SeekGroup(void){ //pass

char cbuf3[20];

fflush(stdin);

if(!grade[0].schead && !grade[1].schead && !grade[2].schead){

printf("Class list is empty"); return FALSE;

}

count=0;

do {

if(count) printf("The option is out of bounds, re-enter");

printf("Search method\n1--class number\t2--number of people\t3--return to main menu\n");

scanf("%hu", &choice); getchar(); count++;

if(choice==3){ ClearScreen(); return TRUE; }

}while(choice<1 || choice>3);

switch(choice){

case 1: //Search by class number

printf("Please enter the name of the class to be checked:");

psch=ClassInput(cbuf3);

if(psch==NULL){

printf("This class does not exist");

getchar(); ClearScreen(); return TRUE;

}

// output class related information

printf("Number of people:%hu\n", psch->clsize);

if(psch->do\_num!=0){

printf("List of courses taken:\n");

for(j=0; j<psch->do\_num; j++)

printf("%s\n", psch->done[j]);

printf("A total of %hu doors\n", psch->do\_num);

}

else printf("No courses taken\n");

if(psch->pr\_num!=0){

printf("Course list:\n");

for(j=0; j<psch->pr\_num; j++)

printf("%s\n", psch->progress[j]);

printf("A total of %hu doors\n", psch->pr\_num);

}

else printf("No current courses\n");

break;

case 2: //Search by class size

flag=FALSE;

printf("Enter the minimum number of people:");

scanf("%hu", &buf1); getchar();

printf("Enter the upper limit of the number of people:");

scanf("%hu", &buf2); getchar();

for(i=0;i<3;i++)

for(psch=grade[i].schead;psch!=NULL;psch=psch->next)

if(psch->clsize>=buf1 && psch->clsize<=buf2){

// output class related information

printf("Class: %s\tnumber of people:%hu\n", psch->classname, psch->clsize);

if(psch->do\_num!=0){

printf("List of courses taken:\n");

for(j=0; j<psch->do\_num; j++)

printf("%s\n", psch->done[j]);

printf("A total of %hu doors\n", psch->do\_num);

}

else printf("No courses taken\n");

if(psch->pr\_num!=0){

printf("Course list:\n");

for(j=0; j<psch->pr\_num; j++)

printf("%s\n", psch->progress[j]);

printf("A total of %hu doors\n", psch->pr\_num);

}

else printf("No current courses\n");

flag=TRUE; //flag mark output valid information

}

if(flag==FALSE) printf("No class that meets the requirements"); getchar();

case 3:

break;

}

getchar(); ClearScreen();

return TRUE;

}

BOOL AddClass(void){ //pass

fflush(stdin);

Class \*ptemp=(Class \*)malloc(sizeof(Class));

//The course name is written into the sample structure

printf("Enter course name:");

scanf("%s", ptemp->coursename); getchar();

for(i=0;i<cougraph->length;i++)

if(!strcmp(cougraph->coulist[i].name,ptemp->coursename)) break;

if(i==cougraph->length){ //Create a new course (the prerequisite course node is not added)

AddNewCourse(ptemp->coursename);

cougraph->length++;

printf("The new course has been imported\n");

}

//classroom number write sample structure

printf("Enter classroom number:");

scanf("%s", ptemp->roomcode); getchar(); //How to deal with multiple classrooms?

for(k=0; k<rolist->length; k++)

if(!strcmp(rolist->elem[k].code,ptemp->roomcode)) break; //k stores the position of the target classroom in the classroom list

if(k==rolist->length) AddNewRoom(ptemp->roomcode);// New classroom

rolist->length++;

// Write the teacher's name into the sample structure

printf("Enter the teacher's name:");

scanf("%s", ptemp->tname); getchar(); //How to deal with multiple teachers?

for(l=0; l<telist->length; l++)

if(!strcmp(telist->elem[l].tname,ptemp->tname)) break; //l stores the position of the target teacher in the teacher list

if(l==telist->length) AddNewTeacher(ptemp->tname);//New teacher

telist->length++;

//Classroom information is stored in a temporary node, and after the class is determined, use it as a template to copy the node to each class

printf("Enter class week (Arabic numerals):");

scanf("%hu", &buf1); getchar();

TimeInput(&(ptemp->cou\_st), &(ptemp->cou\_ed), ptemp->wk\_st, ptemp->wk\_ed);

count=0;

printf("Enter the number of classes:");

scanf("%hu", &hubuffer); getchar();

for(i=0;i<hubuffer;i++){

pcls=(Class \*)malloc(sizeof(Class));

//Copy the sample structure for accessing the linked list

strcpy(pcls->coursename, ptemp->coursename);

strcpy(pcls->roomcode, ptemp->roomcode);

strcpy(pcls->tname, ptemp->tname);

pcls->cou\_st=ptemp->cou\_st; pcls->cou\_ed=ptemp->cou\_ed;

memcpy(pcls->wk\_st, ptemp->wk\_st, sizeof(ptemp->wk\_st));

memcpy(pcls->wk\_ed, ptemp->wk\_ed, sizeof(ptemp->wk\_ed));

//Enter class information

printf("Enter class number:");

//Determine the search head node according to the class number

scanf("%s", cbuffer); getchar();

if(strstr(cbuffer,"15")){

// Find the existence of the target class

for(psch=grade[0].schead; psch!=NULL; psch=psch->next)

if(!strcmp(psch->classname,cbuffer)) break;

if(psch==NULL){ //Insert a new class in the class schedule structure

psch=(Schedule \*)malloc(sizeof(Schedule));

strcpy(psch->classname, cbuffer);

psch->do\_num=psch->pr\_num=0;

printf("Enter the number of people:");

scanf("%hu",&(psch->clsize)); getchar();

psch->next=grade[0].schead;

grade[0].schead=psch;

for(j=0;j<6;j++){ //Time occupancy and weekly schedule structure initialization

psch->week[j].classhead=NULL; psch->week[j].cl\_num=0;

psch->week[j].ochead.head=NULL; psch->week[j].ochead.len=0;

}

grade[0].gr\_num++;

}

}

else if(strstr(cbuffer,"16")){

// Find the existence of the target class

for(psch=grade[1].schead; psch!=NULL; psch=psch->next)

if(!strcmp(psch->classname,cbuffer)) break;

if(psch==NULL){ //Insert a new class in the class schedule structure

psch=(Schedule \*)malloc(sizeof(Schedule));

strcpy(psch->classname, cbuffer);

psch->do\_num=psch->pr\_num=0;

psch->next=grade[1].schead;

grade[1].schead=psch;

for(j=0;j<6;j++){ //Time occupancy and weekly schedule structure initialization

psch->week[j].classhead=NULL; psch->week[j].cl\_num=0;

psch->week[j].ochead.head=NULL; psch->week[j].ochead.len=0;

}

grade[1].gr\_num++;

}

}

// Find the existence of the target class

else if(strstr(cbuffer,"17")){

for(psch=grade[2].schead; psch!=NULL; psch=psch->next)

if(!strcmp(psch->classname,cbuffer)) break;

if(psch==NULL){ //Insert a new class in the class schedule structure

psch=(Schedule \*)malloc(sizeof(Schedule));

strcpy(psch->classname, cbuffer);

psch->do\_num=psch->pr\_num=0;

psch->next=grade[2].schead; grade[2].schead=psch;

for(j=0;j<6;j++){ //Time occupancy and weekly schedule structure initialization

psch->week[j].classhead=NULL; psch->week[j].cl\_num=0;

psch->week[j].ochead.head=NULL; psch->week[j].ochead.len=0;

}

grade[2].gr\_num++;

}

}

else{ // intercept garbled characters

printf("The class number input error");

getchar(); ClearScreen(); return TRUE;

}

for(j=0;j<psch->pr\_num;j++) //Find whether there is a target course in the course list

if(!strcmp(psch->progress[j],ptemp->coursename)) break;

if(j==psch->pr\_num){ //Add this class if there is no

strcpy(psch->progress[j],ptemp->coursename);

psch->pr\_num++;

}

//insert course node

//Special case: currently there is no class for this day or the new class is the first class of the day

if(psch->week[buf1-1].classhead==NULL || psch->week[buf1-1].classhead->cou\_st > pcls->cou\_st || psch->week[buf1-1].classhead ->wk\_st[0]>pcls->wk\_st[0]){

pcls->next=psch->week[buf1-1].classhead;

psch->week[buf1-1].classhead=pcls;

}

else{ //Insert in the middle of the course list

for(qcls=psch->week[buf1-1].classhead; qcls->next!=NULL; qcls=qcls->next)

if(qcls->next->cou\_st>pcls->cou\_st || (qcls->next->cou\_st==pcls->cou\_st && qcls->next->wk\_st[0]>pcls->wk\_st[0]) ){

pcls->next=qcls->next; qcls->next=pcls; break;

}

if(qcls->next==NULL){ //Insert at the end of the course list

pcls->next=qcls->next; qcls->next=pcls;

}

}

psch->week[buf1-1].cl\_num++;

//Occupy the relevant time period in the subordinate array of the class

AddTimeNode(pcls->wk\_st, pcls->wk\_ed, pcls->cou\_st, pcls->cou\_ed, &(psch->week[buf1-1].ochead));

}

//Insert a time-occupancy node under the corresponding teacher node

AddTimeNode(ptemp->wk\_st, ptemp->wk\_ed, ptemp->cou\_st, ptemp->cou\_ed, &(telist->elem[l].ochead[buf1-1]));

//Insert a time-occupancy node under the corresponding classroom node

AddTimeNode(ptemp->wk\_st, ptemp->wk\_ed, ptemp->cou\_st, ptemp->cou\_ed, &(rolist->elem[k].ochead[buf1-1]));

printf("Entered");

getchar(); ClearScreen();

return TRUE;

}

BOOL DelClass(void){ //pass

fflush(stdin);

char cbuf3[20];

unsigned short buf3[5], buf4[5], buf5=0, buf6=0;

printf("Enter the course name of the class to be deleted:");

scanf("%s", cbuffer); getchar();

printf("Enter the number of classes to be deleted:");

scanf("%hu", &hubuffer); getchar();

printf("Enter class date (Arabic numerals)");

scanf("%hu", &buf1); getchar();

for(i=0;i<hubuffer;i++){

count=0;

printf("Enter class number:");

psch=ClassInput(cbuf3);

scanf("%s", cbuf3); getchar();

if(psch==NULL){

printf("The class is not found, continue to operate the next class"); getchar(); continue;

}

//delete target class

for(pcls=psch->week[buf1-1].classhead;pcls!=NULL;pcls=pcls->next)

if(!strcmp(pcls->coursename, cbuffer)) break;

if(pcls==NULL){

printf("No corresponding course found under this date\n"); break;

}

//Copy node information

strcpy(cbuf1, pcls->roomcode); strcpy(cbuf2, pcls->tname);

memcpy(buf3, pcls->wk\_st, sizeof(buf3)); memcpy(buf4, pcls->wk\_ed, sizeof(buf4));

buf5=pcls->cou\_st; buf6=pcls->cou\_ed;

//Delete the relevant time period in the subordinate array of the class

DelTimeNode(pcls->wk\_st, pcls->wk\_ed, pcls->cou\_st, pcls->cou\_ed, &(psch->week[buf1-1].ochead));

DelClassNode(pcls, psch, buf1-1); //delete course node

}

//Delete the occupancy time node in the classroom structure

for(i=0;i<rolist->length;i++)

if(!strcmp(rolist->elem[i].code, cbuf1)) break;

if(i!=rolist->length)

DelTimeNode(buf3,buf4,buf5,buf6,&(rolist->elem[i].ochead[buf1-1]));

// remove seat time in teacher struct

for(i=0;i<telist->length;i++)

if(!strcmp(telist->elem[i].tname, cbuf2)) break;

if(i!=telist->length)

DelTimeNode(buf3,buf4,buf5,buf6,&(telist->elem[i].ochead[buf1-1]));

printf("Deleted successfully");

getchar(); ClearScreen();

return TRUE;

}

BOOL AltClass(void){ //pass

fflush(stdin);

unsigned short buf3[5], buf4[5], buf5, buf6, buf7, buf8[5], buf9[5];

//buf1-buf4 is new information, buf6-buf9 is old information

char cbuf3[20];

Class \*pcls=NULL;

Schedule \*psch=NULL;

printf("Enter the course name of the class to be changed:");

scanf("%s", cbuffer); getchar();

printf(" Enter the number of classes to be changed:");

scanf("%hu", &hubuffer); getchar();

printf("Enter school day (Arabic numerals)");

scanf("%hu", &buf5); getchar(); //The above information is used to locate the class in the class schedule

count=0;

do {

if(count) printf("The option is out of bounds, re-enter");

printf("What information to modify?\n1--classroom\t2--teacher\t3--class time\t4--return to main menu\n");

scanf("%hu", &choice); getchar(); count++;

if(choice==4){ ClearScreen(); return TRUE; }

}while(choice<1 || choice>4);

switch(choice){

case 1: //Change the classroom

printf("Enter new classroom:");

scanf("%s",cbuf2); getchar();

for(l=0;l<rolist->length;l++)

if(!strcmp(rolist->elem[l].code,cbuf2)) break;

if(l==rolist->length) AddNewRoom(cbuf2);//Change to a new classroom and expand the linear table when it crosses the boundary

rolist->length++;

//l record the new classroom position in the lookup table

break;

case 2: //change teacher

printf("Enter the new teacher's name:");

scanf("%s",cbuf2); getchar();

for(l=0; l<telist->length; l++)

if(!strcmp(telist->elem[l].tname,cbuf2)) break;

if(l==telist->length) AddNewTeacher(cbuf2);//New teacher

telist->length++;

//l record the new teacher's position in the lookup table

break;

case 3: //Change time

TimeInput(&buf1, &buf2, buf3, buf4);

break;

case 4:

return TRUE;

}

for(i=0;i<hubuffer;i++){

printf("Enter class number:");

psch=ClassInput(cbuf3);

if(psch==NULL){

printf("Class %s not found, continue to search for the next class\n", cbuf3); continue;

}

// check the existence of the input class

for(pcls=psch->week[buf5-1].classhead; pcls!=NULL; pcls=pcls->next)

if(!strcmp(pcls->coursename, cbuffer)) break;

if(pcls==NULL){

printf("The target class is not found, continue to search for the next class\n"); continue;

}

buf6=pcls->cou\_st; buf7=pcls->cou\_ed;

memcpy(buf8, pcls->wk\_st, sizeof(pcls->wk\_st));

memcpy(buf9, pcls->wk\_ed, sizeof(pcls->wk\_ed)); // Put the original class time into the cache

switch(choice){

case 1: //Modify classroom occupancy

strcpy(cbuf3, pcls->roomcode); //cbuf3 is old information, cbuf2 is new information

strcpy(pcls->roomcode,cbuf2); break;

case 2: //Modify teacher occupancy

strcpy(cbuf3, pcls->tname);

strcpy(pcls->tname,cbuf2); break;

case 3: //Modify time occupancy

//Delete the original time period occupied by the target classroom

DelTimeNode(pcls->wk\_st, pcls->wk\_ed, pcls->cou\_st, pcls->cou\_ed, &(psch->week[buf5-1].ochead));

strcpy(cbuf2, pcls->roomcode); strcpy(cbuf3, pcls->tname);

//Two buffer strings are used to modify the time occupancy in the corresponding classroom and teacher structure

pcls->cou\_st=buf1; pcls->cou\_ed=buf2;

memcpy(pcls->wk\_st, buf3, sizeof(buf3));

memcpy(pcls->wk\_ed, buf4, sizeof(buf4)); //Change the class time in the target class

AddTimeNode(pcls->wk\_st, pcls->wk\_ed, pcls->cou\_st, pcls->cou\_ed, &(psch->week[buf5-1].ochead));

}

}

flag=FALSE;

switch(choice){

case 1: //Modify classroom occupancy

for(j=0;j<rolist->length;j++)

if(!strcmp(rolist->elem[j].code, cbuf3)) break;

//Delete the occupancy of the original classroom by the target classroom

if(j!=rolist->length)

DelTimeNode(buf8,buf9,buf6,buf7,&(rolist->elem[j].ochead[buf5-1]));

//Add the occupancy of the target classroom to the new classroom

AddTimeNode(buf8, buf9, buf6, buf7, &(rolist->elem[l].ochead[buf5-1]));

break;

case 2: //Modify teacher occupancy

for(j=0;j<telist->length;j++)

if(!strcmp(telist->elem[j].tname, cbuf3)) break;

//Delete the occupancy of the original teacher by the target classroom

if(j!=telist->length)

DelTimeNode(buf8,buf9,buf6,buf7,&(telist->elem[j].ochead[buf5-1]));

//Add the occupancy of the target class to the new teacher

AddTimeNode(buf8, buf9, buf6, buf7, &(telist->elem[l].ochead[buf5-1]));

break;

case 3:

for(l=0;l<rolist->length;l++)

if(!strcmp(rolist->elem[l].code, cbuf2)) break;

//Delete the original occupied time of the target classroom

if(l!=rolist->length)

DelTimeNode(buf8,buf9,buf6,buf7,&(rolist->elem[l].ochead[buf5-1]));

//Add the new occupancy time for the target classroom

AddTimeNode(buf3, buf4, buf1, buf2, &(rolist->elem[l].ochead[buf5-1]));

for(l=0;l<telist->length;l++)

if(!strcmp(telist->elem[l].tname, cbuf3)) break;

//Delete the original occupied time of the target teacher

if(l!=telist->length)

DelTimeNode(buf8,buf9,buf6,buf7,&(telist->elem[l].ochead[buf5-1]));

//Add the new occupancy time for the target teacher

AddTimeNode(buf3, buf4, buf1, buf2, &(telist->elem[l].ochead[buf5-1]));

break;

}

printf("modified successfully");

getchar(); ClearScreen();

return TRUE;

}

BOOL SeekClass(void){ //pass

fflush(stdin);

char cbuf3[20];

unsigned short buf8;

BOOL glag;

count=0;

do {

if(count++) printf("The option is out of bounds, re-enter");

printf("How to search?\n1--teacher\t2--class\t\t3--classroom+course+time\t4--time\t\t5--return to main menu\n");

scanf("%hu", &hubuffer); getchar();

if(hubuffer==5){ ClearScreen(); return TRUE; }

}while(hubuffer<1 || hubuff>5);

flag=FALSE;

switch(hubuffer){

case 1:

printf("Enter the teacher:");

scanf("%s", cbuffer); getchar();

for(i=0;i<3;i++){ //three grades

for(psch=grade[i].schead; psch!=NULL; psch=psch->next)

for(j=0;j<6;j++) //six days a week with classes

for(pcls=psch->week[j].classhead;pcls!=NULL;pcls=pcls->next)

// find the corresponding course

if(!strcmp(pcls->tname, cbuffer)){

printf("Course: %s\tClass: %s\tClass Room: %s\nClass Time:", pcls->coursename, psch->classname, pcls->roomcode);

k=0; flag=TRUE; //flag mark output valid information

while(pcls->wk\_st[k]&&k<5){

if(k) putchar(',');

printf("%hu-%hu",pcls->wk\_st[k],pcls->wk\_ed[k]); k++;

}

printf("week, week %hu, %hu-%hu section\n", j+1, pcls->cou\_st, pcls->cou\_ed);

}

}

if(flag==FALSE) printf("No relevant class found");

break;

case 2:

printf("Enter class number:");

psch=ClassInput(cbuf3);

if(psch==NULL){

printf("The class was not found");

getchar(); ClearScreen(); return TRUE;

}

for(i=0;i<6;i++)

for(pcls=psch->week[i].classhead;pcls!=NULL;pcls=pcls->next){

printf("Course: %s\tTeacher: %s\tClass room: %s\nClass time:", pcls->coursename, pcls->tname, pcls->roomcode);

j=0; flag=TRUE; //flag mark output valid information

while(pcls->wk\_st[j]&&j<5){

if(j) putchar(',');

printf("%hu-%hu",pcls->wk\_st[j],pcls->wk\_ed[j]); j++;

}

printf("week, week %hu, %hu-%hu section\n", i+1, pcls->cou\_st, pcls->cou\_ed);

}

if(flag==FALSE) printf("No relevant class found");

break;

case 3:

printf("Enter the name of the class to be checked:");

scanf("%s", cbuffer); getchar();

printf("Enter classroom:");

scanf("%s",cbuf1); getchar();

printf(" Enter class week:");

scanf("%hu", &buf5); getchar();

TimeInput(&buf1, &buf2, buf3, buf4);

for(i=0;i<3;i++){

for(psch=grade[i].schead; psch!=NULL; psch=psch->next)

for(pcls=psch->week[buf5-1].classhead;pcls!=NULL;pcls=pcls->next){

hubuffer=0;//Check the corresponding relationship between class time

hubuffer=(pcls->cou\_st-buf1)+(pcls->cou\_ed-buf2);

count=0;

while((pcls->wk\_st[count]||buf3[count]||pcls->wk\_ed[count]||buf4[count])&&count<5){ //The array element of class week number is valid

hubuffer=(pcls->wk\_st[count]-buf3[count])+(pcls->wk\_ed[count]-buf4[count]);

count++;

}

//All information has been checked and completely overlapped

if(!strcmp(pcls->coursename, cbuffer)&&!strcmp(pcls->roomcode,cbuf1)&&!hubuffer){

printf("Class: %s\tTeacher: %s\n", psch->classname, pcls->tname);

flag=TRUE; //flag mark output valid information

}

}

}

if(flag==FALSE) printf("No relevant class found");

break;

case 4:

printf("Enter the starting section number:");

scanf("%hu", &buf1); getchar();

printf("Enter the end section number:");

scanf("%hu", &buf2); getchar();

printf("Enter target week:");

scanf("%hu", &buf8); getchar();

printf("Enter the starting week number:");

scanf("%hu", &buf5); getchar();

printf("Enter the end week number:");

scanf("%hu", &buf6); getchar();

flag=FALSE;

for(i=0;i<3;i++){

for(psch=grade[i].schead; psch!=NULL; psch=psch->next)

for(pcls=psch->week[buf8-1].classhead;pcls!=NULL;pcls=pcls->next){

glag=FALSE;

hubuffer=count=0;//Check the corresponding relationship between class time

if(pcls->cou\_st>=buf1&&pcls->cou\_ed<=buf2) hubuffer=1;

//The target class has classes within the specified time range

while(pcls->wk\_st[count]||pcls->wk\_ed[count]){

if(buf5 >= pcls->wk\_st[count] && buf6<= pcls->wk\_ed[count]){

glag=TRUE; break; //flag tag output valid information

}

count++;

}

//The time parameter meets the requirements

if(hubuffer&&glag){

printf("Course: %s\tClassroom: %s\tClass: %s\tTeacher:%s\n", pcls->coursename, pcls->roomcode, psch->classname, pcls->tname) ;

flag=TRUE; //flag mark output valid information

} }

}

if(flag==FALSE) printf("No relevant class found");

case 5:

break;

}

getchar(); ClearScreen();

return TRUE;

}

BOOL RoomUsage(void){

fflush(stdin);

unsigned short buf7, buf8;

BOOL glag;

count=0;

do {

if(count) printf(" Please re-enter\n");

printf("Choose the analysis method:\n1--search for free classrooms\t2--classroom utilization\t3--rationality of classroom use\t4--return to the main menu\n");

scanf("%hu", &choice); getchar(); count++;

if(choice==4){ ClearScreen(); return TRUE; }

}while(choice<1 || choice>4);

switch(choice){

case 1:

printf("Enter target week:");

scanf("%hu", &hubuffer); getchar();

printf("Enter target week:");

scanf("%hu", &buf5); getchar();

printf("Enter the start section:");

scanf("%hu", &buf1); getchar();

printf("Enter termination section:");

scanf("%hu", &buf2); getchar();

glag=FALSE;

count=0;

for(i=0;i<rolist->length;i++){

flag=FALSE;

for(pocc=rolist->elem[i].ochead[buf5-1].head;pocc!=NULL;pocc=pocc->next){

if(pocc->week==hubuffer && pocc->time>=buf1 && pocc->time<=buf2){

flag=TRUE; break;

}

}

if(flag==FALSE){

printf("%s\t", rolist->elem[i].code); glag=TRUE;

count++;

if(!(count%5)) putchar('\n');

}

}

if(glag==FALSE) printf("There are no classrooms that meet the requirements within this time period");

break;

case 2:

printf("Enter the target week interval (only one week, the beginning and end weeks are the same)\nStarting week:");

scanf("%hu", &buf1); getchar();

printf("End week:");

scanf("%hu", &buf2); getchar();

printf("Enter the target date interval (only one day, the beginning and end are the same)\nStart date:");

scanf("%hu", &buf7); getchar();

printf("End date:");

scanf("%hu", &buf8); getchar();

printf("Enter the target section interval (the beginning and end sections are the same only for one week)\nStart section:");

scanf("%hu", &buf5); getchar();

printf("end section:");

scanf("%hu", &buf6); getchar();

count=0;

for(i=0;i<rolist->length;i++)

for(j=buf7-1; j<buf8; j++)

for(pocc=rolist->elem[i].ochead[j].head;pocc!=NULL;pocc=pocc->next)

if(pocc->week>=buf1 && pocc->week<=buf2 && pocc->time>=buf5 && pocc->time<=buf6) count++;

printf("Classroom utilization rate in the target time period is %f", count\*1.0/(rolist->length\*(buf2-buf1+1)\*(buf8-buf7+1)\*(buf6-buf5+1))) ;

break;

case 3:

buf6=0;

printf("Enter classroom:");

scanf("%s",cbuf1); getchar();

for(i=0;i<rolist->length;i++)

if(!strcmp(cbuf1,rolist->elem[i].code)) break;

if(i==rolist->length){

printf("No target classroom found");

getchar(); ClearScreen(); return TRUE;

}

printf("The maximum capacity of the current classroom is %hu\n", rolist->elem[i].rsize);

printf("Enter the name of the class to be checked:");

scanf("%s", cbuffer); getchar();

printf("Enter class week:");

scanf("%hu", &buf5); getchar();

TimeInput(&buf1, &buf2, buf3, buf4);

flag=FALSE;

for(j=0;j<3;j++){

for(psch=grade[j].schead; psch!=NULL; psch=psch->next)

for(pcls=psch->week[buf5- 1].classhead;pcls!=NULL;pcls=pcls->next){

hubuffer=0;//Check the corresponding relationship between class time

hubuffer=(pcls->cou\_st-buf1)+(pcls->cou\_ed-buf2);

count=0;

while((pcls->wk\_st[count]||buf3[count]||pcls->wk\_ed[count]||buf4[count])&&count<5){

hubuffer=(pcls->wk\_st[count]-buf3[count])+(pcls->wk\_ed[count]-buf4[count]);

count++;

}

if(!strcmp(pcls->coursename, cbuffer)&&!strcmp(pcls->roomcode,cbuf1)&&!hubuffer){

printf("Class: %s\tnumber of people%hu\n", psch->classname, psch->clsize);

buf6+=psch->clsize;

flag=TRUE;

}

}

}

if(flag==FALSE){

printf("No related class found"); break;

}

else {

if(buf6 > rolist->elem[i].rsize)

printf("The classroom is overcrowded, a larger classroom needs to be replaced\n");

else if(buf6/rolist->elem[i].rsize<=1 && buf6\*1.0/rolist->elem[i].rsize>=0.6)

printf("There is a shortage of seats, a larger classroom needs to be replaced\n");

else if(buf6\*1.0/rolist->elem[i].rsize<=0.5)

printf("There are too many vacant seats, please consider merging classrooms or moving to smaller classrooms\n");

else

printf("The classroom is suitable\n");

}

case 4:

break;

}

getchar(); ClearScreen();

return TRUE;

}

BOOL ClassAnalysis(void){

fflush(stdin);

unsigned short occupied[22][6];

char cbuf3[20];

printf("Enter the class to be analyzed:");

psch=ClassInput(cbuf3);

if(psch==NULL){

printf("The class was not found");

getchar(); ClearScreen(); return TRUE;

}

count=0;

do {

if(count) printf("Please re-enter\n");

printf("Choose the analysis method:\n1--the rationality of the courses arranged in the morning and evening\t2--the rationality of the courses arranged by the day\n3--the rationality of the courses\t4- - Return to main menu\n");

scanf("%hu", &choice); getchar(); count++;

if(choice==4){ ClearScreen(); return TRUE; }

}while(choice<1 || choice>4);

switch(choice){

case 1: //The rationality of the course arrangement in each day

for(j=1;j<18;j++){

flag=TRUE;

buf1=buf2=buf5=0;

printf("Week %d",(int)j);

for(i=0;i<6;i++)

for(pocc=psch->week[i].ochead.head;pocc!=NULL;pocc=pocc->next)

if(pocc->week==j){

if(pocc->time>=1 && pocc->time<=4) buf1++;

else if(pocc->time>=5 && pocc->time<=8) buf2++;

else if(pocc->time>=9 && pocc->time<=12) buf5++;

}

if(buf5>buf1/2 || buf5>buf2/2){

printf("The proportion of evening classes is too large\n"); flag=FALSE;

}

if(flag==TRUE) printf("The courses are arranged more reasonably in each day\n");

}

break;

case 2:

printf("Enter the number of valid weeks in this semester:");

scanf("%hu", &hubuffer); getchar();

memset(occupy,0,sizeof(occupy));

for(j=0;j<6;j++)

for(pocc=psch->week[j].ochead.head;pocc!=NULL;pocc=pocc->next)

occupy[pocc->week-1][j]++;

for(i=0;i<hubuffer;i++){

flag=TRUE;

printf("Week %d",(int)i+1);

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

for(k=j;k<5;k++)

if(occupy[i][j]-occupy[i][k]>=6){

printf("The distribution of courses in week %d and week %d is too different\n",j+1,k+1); flag=FALSE;

}

if(flag==TRUE) printf("Course distribution is more reasonable\n");

}

buf1=buf2=buf5=buf6=0;

for(i=1;i<6;i++){

buf1+=occupy[0][i];buf2+=occupy[1][i];

buf5+=occupy[hubuffer-2][i]; buf6+=occupy[hubuffer-1][i];

}

if(buf1>=20) printf("There are too many classes in the first week\n");

if(buf2>=22) printf("There are too many classes in the second week\n");

if(buf5>=10) printf("There are too many classes in week %hu\n",hubuffer-1);

if(buf6>=4) printf("There are too many classes in week %hu\n",hubuffer);

break;

case 3:

for(i=0;i<psch->pr\_num;i++){

flag=FALSE;

for(j=0;j<cougraph->length;j++)

if(!strcmp(psch->progress[i],cougraph->coulist[j].name)) break;

if(j==cougraph->length){

printf("Course %s has not been entered\n",cougraph->coulist[j].name); return FALSE;

}

for(pcarc=cougraph->coulist[j].archhead;pcarc!=NULL;pcarc=pcarc->nextarc){

for(k=0;k<psch->do\_num;k++)

if(!strcmp(psch->done[k],pcarc->prename)) break;

if(k==psch->do\_num){

printf("%s has not been completed, %s should not be completed this semester\n",pcarc->prename,psch->progress[i]);

flag=TRUE;

}

}

if(flag==FALSE) printf("Course %s can be taken in this semester\n",cougraph->coulist[j].name);

putchar('\n');

}

case 4:

break;

}

//The rationality of the course arrangement within a week

getchar(); ClearScreen();

return TRUE;

}

BOOL Teacher Analysis(void){

fflush(stdin);

unsigned short occupied[22];

printf("Enter the teacher to be analyzed:");

scanf("%s",cbuffer); getchar();

for(i=0;i<telist->length;i++)

if(!strcmp(telist->elem[i].tname,cbuffer)) break;

if(i==telist->length){

printf("Target teacher not found");

getchar(); ClearScreen(); return TRUE;

}

count=0;

do {

if(count) printf(" The option is out of bounds, re-enter");

printf("Choose the analysis method:\n1--Teaching tasks are assigned by day\t2--Teaching task allocation\t3--Floor arrangement is reasonable\t4Return to the main menu\n");

scanf("%hu", &choice); getchar(); count++;

if(choice==4){ ClearScreen(); return TRUE; }

}while(choice<1 || choice>4);

switch(choice){

case 1:

for(j=1;j<18;j++){

flag=TRUE;

printf("Week %d",(int)j);

buf1=buf2=buf5=0;

for(k=0;k<6;k++)

for(pocc=telist->elem[i].ochead[k].head;pocc!=NULL;pocc=pocc->next)

if(pocc->week==j){

if(pocc->time>=1 && pocc->time<=4) buf1++;

else if(pocc->time>=5 && pocc->time<=8) buf2++;

else if(pocc->time>=9 && pocc->time<=12) buf5++;

}

if(buf5>buf1/2 || buf5>buf2/2){ //judgment based on class

printf("The proportion of evening classes is too large\n"); flag=FALSE;

}

if(flag==TRUE) printf("The courses are arranged more reasonably in each day\n");

}

break;

case 2:

printf("Enter the number of valid weeks in this semester:");

scanf("%hu", &hubuffer); getchar();

memset(occupy,0,sizeof(occupy));

for(j=0;j<6;j++)

for(pocc=psch->week[j].ochead.head;pocc!=NULL;pocc=pocc->next)

occupy[pocc->week-1]++;

flag=FALSE;

for(j=0;j<hubuffer;j++)

//Retirement age: male-55 female-60

if((telist->elem[i].gender==0 && telist->elem[i].age<50) || (telist->elem[i].gender==1 && telist->elem[i ].age<55)){

if(occupy[j]>14){

printf("There are too many classes in week %hu\n",j+1); flag=TRUE;

}

}

else {

if(occupy[j]>10){

printf("There are too many classes in week %hu\n",j+1); flag=TRUE;

}

}

if(flag==FALSE) printf("The teacher's teaching assignment is more appropriate");

break;

case 3:

if((telist->elem[i].gender==0 && telist->elem[i].age<55) || (telist->elem[i].gender==1 && telist->elem[i ].age<60)){

printf("The teacher has no special floor requirements\n"); break;

}

else {

flag=FALSE;

for(j=0;j<3;j++){

for(psch=grade[j].schead;psch!=NULL;psch=psch->next){

for(k=0;k<6;k++){

for(pcls=psch->week[k].classhead;pcls!=NULL;pcls=pcls->next){

if(!strstr(pcls->tname,telist->elem[i].tname)) continue;

if(strstr(pcls->roomcode,"4") || strstr(pcls->roomcode,"5")){

printf("The floor of the classroom %s where %s is located is too high\n", pcls->coursename, pcls->roomcode);

flag=TRUE;

} } } } }

if(flag==FALSE) printf("The teacher's floor is more reasonable\n");

}

case 4:

break;

}

getchar(); ClearScreen();

return TRUE;

}

BOOL FileOut(void){

fflush(stdin);

FILE \*out;

time\_t lt;

char \*filename, \*pc;

const char \*appe=".txt";

lt=time(NULL);

filename=ctime(&lt);

filename[strlen(filename)-1]='\0';

strcat(filename,appe);

pc=&filename[0];

while(\*pc){

if((\*pc==':' ) || (\*pc==' ')){ \*pc='\_'; }

pc++;

}

if((out=fopen(filename,"w"))==NULL){

printf("Failed to open!\n");

getchar(); ClearScreen(); return TRUE;

}

//teacher list

fprintf(out, "%hu ", telist->length);

for(i=0;i<telist->length;i++){

fprintf(out, "%s ", telist->elem[i].tname);

fprintf(out, "%hu ", telist->elem[i].gender);

fprintf(out, "%hu ", telist->elem[i].age);

for(j=0;j<6;j++){

fprintf(out, "%hu ", telist->elem[i].ochead[j].len);

for(pocc=telist->elem[i].ochead[j].head;pocc!=NULL;pocc=pocc->next){

fprintf(out, "%hu ", pocc->time);

fprintf(out, "%hu ", pocc->week);

}

}

}

//classroom list

fprintf(out, "%hu ", rolist->length);

for(i=0;i<rolist->length;i++){

fprintf(out, "%s ", rolist->elem[i].code);

fprintf(out, "%hu ", rolist->elem[i].rsize);

for(j=0;j<6;j++){

fprintf(out, "%hu ", rolist->elem[i].ochead[j].len);

for(pocc=rolist->elem[i].ochead[j].head;pocc!=NULL;pocc=pocc->next){

fprintf(out, "%hu ", pocc->time);

fprintf(out, "%hu ", pocc->week);

}

}

}

// course list

fprintf(out, "%hu ", cougraph->length);

for(i=0;i<cougraph->length;i++){

fprintf(out, "%s ",cougraph->coulist[i].name);

fprintf(out, "%hu " ,cougraph->coulist[i].arcnum);

for(pcarc=cougraph->coulist[i].archhead;pcarc!=NULL;pcarc=pcarc->nextarc)

fprintf(out, "%s ", pcarc->prename);

}

//class + schedule list

for(i=0;i<3;i++){

fprintf(out, "%hu ",grade[i].gr\_num);

for(psch=grade[i].schead;psch!=NULL;psch=psch->next){

fprintf(out, "%s ", psch->classname);

fprintf(out, "%hu ", psch->clsize);

fprintf(out, "%hu ", psch->do\_num);

fprintf(out, "%hu ", psch->pr\_num);

if(psch->do\_num)

for(j=0;j<psch->do\_num;j++)

fprintf(out, "%s ", psch->done[j]);

if(psch->pr\_num)

for(j=0;j<psch->pr\_num;j++)

fprintf(out, "%s ", psch->progress[j]);

for(j=0;j<6;j++){

fprintf(out, "%hu ", psch->week[j].cl\_num);

for(pcls=psch->week[j].classhead;pcls!=NULL;pcls=pcls->next){

fprintf(out, "%s ", pcls->coursename);

fprintf(out, "%s ", pcls->roomcode);

fprintf(out, "%s ",pcls->tname);

fprintf(out, "%hu ",pcls->cou\_st);

fprintf(out, "%hu ",pcls->cou\_ed);

for(k=0;k<5;k++){

fprintf(out, "%hu ",pcls->wk\_st[k]);

fprintf(out, "%hu ",pcls->wk\_ed[k]);

}

}

fprintf(out, "%hu ", psch->week[j].ochead.len);

for(pocc=psch->week[j].ochead.head;pocc!=NULL;pocc=pocc->next){

fprintf(out, "%hu ", pocc->time);

fprintf(out, "%hu ", pocc->week);

}

}

}

}

fclose(out);

printf("Backup succeeded! Press any key to continue...\n");

getchar(); ClearScreen();

return TRUE;

}

BOOL FileIn(void){

fflush(stdin);

FILE \*in;

printf("The imported data will overwrite the current data of the system! Continue——Y to return to the main menu——any other key\n");

fflush(stdin); j=getchar(); getchar();

if(j!='y'&& j!='Y'){

printf("Data import cancel");

getchar(); ClearScreen(); return TRUE;

}

char input[80], \*filename;

printf("Please enter the file name of the data you want to restore:");

fflush(stdin); scanf("%s",input); getchar();

filename=input;

//int handle, find;

if ((in = fopen(filename, "r+")) == NULL)

{

printf("The backup file you input does not exist, the system will keep the current data");

getchar(); ClearScreen(); return TRUE;

}

ClearMemory();

Initial();

fscanf(in, "%hu", &(telist->length));

if(telist->length >= telist->listsize)

buf1 = (telist->length-telist->listsize)/LISTINCREMENT+1;

while(telist->length >= telist->listsize){

Teacher \*te\_newbase = (Teacher \*)realloc(telist->elem, (telist->listsize+buf1\*LISTINCREMENT)\* sizeof(Teacher));

if(!te\_newbase){

printf("Memory allocation failed"); return FALSE;

}

telist->elem = te\_newbase;

telist->listsize += buf1\*LISTINCREMENT;

}

for(i=0;i<telist->length;i++){

fscanf(in, "%s", telist->elem[i].tname);

fscanf(in, "%hu ", &(telist->elem[i].gender));

fscanf(in, "%hu ", &(telist->elem[i].age));

for(j=0;j<6;j++){

fscanf(in, "%hu", &(telist->elem[i].ochead[j].len));

telist->elem[i].ochead[j].head=NULL;

for(k=0;k<telist->elem[i].ochead[j].len;k++){

pocc = (Occupy \*)malloc(sizeof(Occupy));

fscanf(in, "%hu", &(pocc->time));

fscanf(in, "%hu", &(pocc->week));

pocc->next=telist->elem[i].ochead[j].head;

telist->elem[i].ochead[j].head=pocc;

}

}

}

fscanf(in, "%hu", &(rolist->length));

if(rolist->length >= rolist->listsize)

buf1 = (rolist->length-rolist->listsize)/LISTINCREMENT+1;

while(rolist->length >= rolist->listsize){

Room \*ro\_newbase = (Room\*)realloc(rolist->elem, (rolist->listsize+buf1\*LISTINCREMENT)\* sizeof(Room));

if(!ro\_newbase){

printf("Memory allocation failed"); return FALSE;

}

rolist->elem = ro\_newbase;

rolist->listsize += buf1\*LISTINCREMENT;

}

for(i=0;i<rolist->length;i++){

fscanf(in, "%s", rolist->elem[i].code);

fscanf(in, "%hu ", &(rolist->elem[i].rsize));

for(j=0;j<6;j++){

fscanf(in, "%hu", &(rolist->elem[i].ochead[j].len));

rolist->elem[i].ochead[j].head=NULL;

for(k=0;k<rolist->elem[i].ochead[j].len;k++){

pocc = (Occupy \*)malloc(sizeof(Occupy));

fscanf(in, "%hu", &(pocc->time));

fscanf(in, "%hu", &(pocc->week));

pocc->next=rolist->elem[i].ochead[j].head; //The order of node insertion is reversed?

rolist->elem[i].ochead[j].head=pocc;

}

}

}

fscanf(in, "%hu", &(cougraph->length));

if(cougraph->length >= cougraph->listsize)

buf1 = (cougraph->length-cougraph->listsize)/LISTINCREMENT+1;

while(cougraph->length >= cougraph->listsize){

CouHead \*cou\_newbase = (CouHead\*)realloc(cougraph->coulist, (cougraph->listsize+buf1\*LISTINCREMENT)\* sizeof(CouHead));

if(!cou\_newbase){

printf("Memory allocation failed"); return FALSE;

}

cougraph->coulist = cou\_newbase;

cougraph->listsize += buf1\*LISTINCREMENT;

}

for(i=0;i<cougraph->length;i++){

fscanf(in, "%s", cougraph->coulist[i].name);

fscanf(in, "%hu", &(cougraph->coulist[i].arcnum));

cougraph->coulist[i].archhead=NULL;

for(j=0;j<cougraph->coulist[i].arcnum;j++){

pcarc=(CouArc \*)malloc(sizeof(CouArc));

pcarc->prename=(char \*)malloc(sizeof(cbuffer));

fscanf(in, "%s", pcarc->prename);

pcarc->nextarc=cougraph->coulist[i].archhead;

cougraph->coulist[i].archhead=pcarc;

}

}

for(i=0;i<3;i++){

fscanf(in, "%hu", &(grade[i].gr\_num));

grade[i].schead=NULL;

for(j=0;j<grade[i].gr\_num;j++){

psch=(Schedule \*)malloc(sizeof(Schedule));

fscanf(in, "%s", psch->classname);

fscanf(in, "%hu ",&(psch->clsize));

fscanf(in, "%hu", &(psch->do\_num));

fscanf(in, "%hu", &(psch->pr\_num));

if(psch->do\_num)

for(k=0;k<psch->do\_num;k++)

fscanf(in, "%s", psch->done[k]);

if(psch->pr\_num)

for(k=0;k<psch->pr\_num;k++)

fscanf(in, "%s", psch->progress[k]);

for(k=0;k<6;k++){

fscanf(in, "%hu", &(psch->week[k].cl\_num));

psch->week[k].classhead=NULL;

for(l=0;l<psch->week[k].cl\_num;l++){

pcls=(Class \*)malloc(sizeof(Class));

fscanf(in, "%s", pcls->coursename);

fscanf(in, "%s", pcls->roomcode);

fscanf(in, "%s", pcls->tname);

fscanf(in, "%hu", &(pcls->cou\_st));

fscanf(in, "%hu", &(pcls->cou\_ed));

for(m=0;m<5;m++){

fscanf(in, "%hu", &(pcls->wk\_st[m]));

fscanf(in, "%hu", &(pcls->wk\_ed[m]));

}

pcls->next=NULL;

if(psch->week[k].classhead==NULL)

psch->week[k].classhead=pcls;

else {

for(qcls=psch->week[k].classhead;qcls->next!=NULL;qcls=qcls->next) ;

qcls->next=pcls;

}

}

fscanf(in, "%hu", &(psch->week[k].ochead.len));

psch->week[k].ochead.head=NULL;

for(l=0;l<psch->week[k].ochead.len;l++){

pocc = (Occupy \*)malloc(sizeof(Occupy));

fscanf(in, "%hu", &(pocc->time));

fscanf(in, "%hu", &(pocc->week));

pocc->next=psch->week[k].ochead.head;

psch->week[k].ochead.head=pocc;

}

}

psch->next=grade[i].schead;

grade[i].schead=psch;

}

}

fclose(in);

printf("Data imported successfully, press any key to continue...\n");

getchar(); ClearScreen();

return TRUE;

}

/\*\*

\* Function name: MaintainTeacherlInfo

\* Function function: teacher information maintenance directory

\* Input parameters: none

\* Output parameters: None

\* Return value: BOOL type, always TRUE

\*

\* Call description:

\*/

BOOL MaintainTeacherInfo(void){

BOOL bRet = TRUE;

do {

printf(" What do you want to do?\n");

printf("Enter press1\nmodify press2\ndelete press3\nreturn to main menu press4\n");

fflush(stdin); k=getchar(); getchar();

if(k=='4') { ClearScreen(); return TRUE; }

}while(k<'1'||k>'4');

if(k=='1') bRet = AddTeacher();

else if(k=='2') bRet = AltTeacher();

else if(k=='3') bRet = DelTeacher();

return bRet;

}

/\*\*

\* Function name: MaintainRoomInfo

\* Function function: classroom information maintenance directory

\* Input parameters: none

\* Output parameters: None

\* Return value: BOOL type, always TRUE

\*

\* Call description:

\*/

BOOL MaintainRoomInfo(void){

BOOL bRet = TRUE;

do {

printf("What do you want to do?\n");

printf("Enter press1\nmodify press2\ndelete press3\nreturn to main menu press4\n");

fflush(stdin); k=getchar(); getchar();

if(k=='4') { ClearScreen(); return TRUE; }

}while(k<'1'||k>'4');

if(k=='1'){ bRet = AddRoom(); }

else if(k=='2'){ bRet = AltRoom(); }

else if(k=='3'){ bRet = DelRoom(); }

return bRet;

}

/\*\*

\* Function name: MaintainGroupInfo

\* Function function: class information maintenance directory

\* Input parameters: none

\* Output parameters: None

\* Return value: BOOL type, always TRUE

\*

\* Call description:

\*/

BOOL MaintainGroupInfo(void){

BOOL bRet = TRUE;

do {

printf("What do you want to do?\n");

printf("Enter press1\nmodify press2\ndelete press3\nreturn to main menu press4\n");

fflush(stdin); k=getchar(); getchar();

if(k=='4') { ClearScreen(); return TRUE; }

}while(k<'1'||k>'4');

if(k=='1'){ bRet = AddGroup(); }

else if(k=='2'){ bRet = AltGroup(); }

else if(k=='3'){ bRet = DelGroup(); }

return bRet;

}

/\*\*

\* Function name: MaintainCourseInfo

\* Function function: course information maintenance directory

\* Input parameters: none

\* Output parameters: None

\* Return value: BOOL type, always TRUE

\*

\* Call description:

\*/

BOOL MaintainCourseInfo(void){

BOOL bRet = TRUE;

do {

printf("What do you want to do?\n");

printf("Enter press1\nmodify press2\ndelete press3\nreturn to main menu press4\n");

fflush(stdin); k=getchar(); getchar();

if(k=='4') { ClearScreen(); return TRUE; }

}while(k<'1'||k>'4');

if(k=='1'){ bRet = AddCourse(); }

else if(k=='2'){ bRet = AltCourse(); }

else if(k=='3'){ bRet = DelCourse(); }

return bRet;

}

/\*\*

\* Function name: MaintainClassInfo

\* Function function: course information maintenance directory

\* Input parameters: none

\* Output parameters: None

\* Return value: BOOL type, always TRUE

\*

\* Call description:

\*/

BOOL MaintainClassInfo(void){

BOOL bRet = TRUE;

do {

printf("What do you want to do?\n");

printf("Enter press1\nmodify press2\ndelete press3\nreturn to main menu press4\n");

fflush(stdin); k=getchar(); getchar();

if(k=='4') { ClearScreen(); return TRUE; }

}while(k<'1'||k>'4');

if(k=='1'){ bRet = AddClass(); }

else if(k=='2'){ bRet = AltClass(); }

else if(k=='3'){ bRet = DelClass(); }

return bRet;

}

BOOL Add(void){

fflush(stdin);

for(i=0;i<telist->length;i++){

printf("Teacher's name: %s\nEnter gender (male--1\tfemale--0):",telist->elem[i].tname);

scanf("%hu",&telist->elem[i].gender); getchar();

printf("Enter age:");

scanf("%hu",&telist->elem[i].age); getchar();

}

for(i=0;i<rolist->length;i++){

printf("Classroom number: %s\nInput capacity:", rolist->elem[i].code);

scanf("%hu",&rolist->elem[i].rsize); getchar();

}

for(i=0;i<3;i++){

for(psch=grade[i].schead;psch!=NULL;psch=psch->next){

printf("Class number: %s\nEnter the number of people:", psch->classname);

scanf("%hu",&psch->clsize); getchar();

}

}

return TRUE;

}

BOOL HelpTopic(void)

{

ShellExecuteA(NULL, "open", "NOTEPAD.EXE", "help.txt", NULL, SW\_SHOWNORMAL);

printf("Press any key to continue...\n"); getch(); ClearScreen();

return TRUE;

}

BOOL AboutDorm(void)

{

char \*plabel\_name[] = {"System version: V1.0",

"\*\*\*\*\*\*\*\*\*\*\*\*\*\*",

"All Rights Reserved",

"confirm"

};

ShowModule(plabel\_name, 4);

return TRUE;

}

BOOL DelTimeNode(unsigned short wk\_st[], unsigned short wk\_ed[], unsigned short cou\_st, unsigned short cou\_ed, Ochead \*ochead){

count=0;

while(wk\_st[count] && count<5){ //traverse according to the number of weeks in each class

for(p=wk\_st[count]; p<=wk\_ed[count]; p++){ //Traverse by class week

for(n=cou\_st; n<=cou\_ed; n++){ // traverse according to the number of lessons

for(pocc=ochead->head;pocc!=NULL;pocc=pocc->next){

if(pocc->week==p && pocc->time==n){ //The class time matches

if(pocc==ochead->head){ //The time occupying node is the head node

ochead->head=pocc->next;

free(pocc); pocc=ochead->head;

if(ochead->head==NULL){

ochead->len=0; break;

} }

else{ //The time-occupied node is not the head node

for(qocc=ochead->head;qocc->next!=pocc;qocc=qoccc->next) ;

qocc->next = pocc->next; free(pocc); pocc=qocc;

}

ochead->len--; //The number of time-occupied nodes decreases by itself

break;

} } } }

count++;

}

return TRUE;

}

BOOL AddTimeNode(unsigned short wk\_st[], unsigned short wk\_ed[], unsigned short cou\_st, unsigned short cou\_ed, Ochead \*ochead){

unsigned short dwk, dtm;

count=0;

while(wk\_st[count]&&count<5){

for(p=wk\_st[count]; p<=wk\_ed[count]; p++)

for(n=cou\_st;n<=cou\_ed; n++){

//Insert the time-occupied nodes in ascending order of the number of weeks first and then the number of sections

pocc=(Occupy \*)malloc(sizeof(Occupy));

pocc->week=p; pocc->time=n;

if(ochead->len==0){ //Insert at the first place when there is no node

pocc->next=ochead->head; ochead->head=pocc;

}

else {

dwk=pocc->week-ochead->head->week;

dtm=pocc->time-ochead->head->time;

if(dwk<0 || (dwk>=0 && dtm<0)){ //The minimum value of the new node

pocc->next=ochead->head;

ochead->head=pocc;

}

else{ //Insert in the middle of the time-occupied linked list

for(qocc=ochead->head;qocc->next!=NULL;qocc=qoccc->next){

dwk=pocc->week-qocc->next->week;

dtm=pocc->time-qocc->next->time;

if(dwk<0 || (dwk>=0 && dtm<0)){

pocc->next=qocc->next; qocc->next=pocc;

} }

if(qocc->next==NULL){ //Insert at the end of the time-occupied linked list

pocc->next=qocc->next; qocc->next=pocc;

} } }

ochead->len++;

}

count++;

}

return TRUE;

}

Schedule \*ClassInput(char cbuf[]){

fflush(stdin);

Schedule \*ps=NULL;

scanf("%s", cbuf); getchar();

//Determine the search head node according to the class number

if(strstr(cbuf,"15")) ps=grade[0].schead;

else if(strstr(cbuf,"16")) ps=grade[1].schead;

else if(strstr(cbuf,"17")) ps=grade[2].schead;

else {

printf("The class number input error");

getchar(); ClearScreen(); return NULL;

}

// check the existence of the class number

for(;ps!=NULL;ps=ps->next)

if(!strcmp(ps->classname, cbuf)) break;

return ps;

}

BOOL AddNewRoom(char cbuf[]){

if(rolist->length >= rolist->listsize){

Room \*ro\_newbase = (Room \*)realloc(rolist->elem, (rolist->listsize+LISTINCREMENT)\* sizeof(Room));

if(!ro\_newbase){

printf("Memory allocation failed, return to main menu");

getchar(); ClearScreen(); return TRUE;

}

rolist->elem = ro\_newbase;

rolist->listsize += LISTINCREMENT;

}

//Add a new classroom

strcpy(rolist->elem[rolist->length].code, cbuf);

//time structure initialization

for(i=0;i<6;i++){

rolist->elem[rolist->length].ochead[i].head=NULL;

rolist->elem[rolist->length].ochead[i].len=0;

}

printf("The new classroom has been imported\n");

return TRUE;

}

BOOL AddNewTeacher(char cbuf[]){

if(telist->length >= telist->listsize){

Teacher \*te\_newbase = (Teacher \*)realloc(telist->elem, (telist->listsize+LISTINCREMENT)\* sizeof(Teacher));

if(!te\_newbase){

printf("Memory allocation failed, return to main menu");

getchar(); ClearScreen(); return TRUE;

}

telist->elem = te\_newbase;

telist->listsize += LISTINCREMENT;

}

//add new teacher

strcpy(telist->elem[telist->length].tname, cbuf);

//time structure initialization

for(i=0;i<6;i++){

telist->elem[telist->length].ochead[i].head=NULL;

telist->elem[telist->length].ochead[i].len=0;

}

printf("New teacher has been imported\n");

return TRUE;

}

BOOL AddNewCourse(char cbuf[]){

if(cougraph->length >= cougraph->listsize){ //Lookup table expansion

CouHead \*cou\_newbase = (CouHead \*)realloc(cougraph->coulist, (cougraph->listsize+LISTINCREMENT)\* sizeof(CouHead));

if(!cou\_newbase){

printf("Memory allocation failed, will return to the main menu");

getchar(); ClearScreen(); return TRUE;

}

cougraph->coulist = cou\_newbase;

cougraph->listsize += LISTINCREMENT;

}

// node initialization

strcpy(cougraph->coulist[cougraph->length].name,cbuf);

cougraph->coulist[cougraph->length].arcnum=0;

cougraph->coulist[cougraph->length].archhead=NULL;

return TRUE;

}

BOOL SeekRemClass(Schedule \*ps, char cbuf[]){

Class \*qc;

for(p=0;p<ps->pr\_num;p++)

if(!strcmp(ps->progress[p],cbuf)) break; //Look for the deleted courses in the courses in the class

flag=FALSE;

if(p!=ps->pr\_num){

for(n=0;n<6;n++) //Find out if this course still has classes

for(qc=ps->week[n].classhead;qc!=NULL;qc=qc->next){

if(flag==TRUE) break;

if(!strcmp(qc->coursename,ps->progress[p])){

flag=TRUE; break;

}

}

if(flag==FALSE){ //Delete the ongoing course when there is no class

for(n=p;n<ps->pr\_num-1;n++) strcpy(ps->progress[n],ps->progress[n+1]); //Delete the corresponding course in the class node

ps->pr\_num--;

}

}

return TRUE;

}

BOOL DelClassNode(Class \*pc, Schedule \*ps, unsigned short hu){\

Class \*qc;

if(pc==ps->week[hu].classhead){ //head node under the course chain

ps->week[hu].classhead=pc->next;

free(pc); pc=ps->week[j].classhead;

if(ps->week[hu].classhead==NULL){

ps->week[hu].cl\_num=0; return TRUE;

} }

else{ //non-head node under the course chain

for(qc=ps->week[hu].classhead;qc->next!=pc;qc=qc->next) ; //find the target node predecessor

qc->next=pc->next; free(pc); pc=qc;

}

psch->week[hu].cl\_num--;

return TRUE;

}

BOOL TimeInput(unsigned short \*cou\_st, unsigned short \*cou\_ed, unsigned short wk\_st[], unsigned short wk\_ed[]){

fflush(stdin);

memset(wk\_st,0,5);

memset(wk\_ed,0,5);

printf("Enter the start time:");

scanf("%hu", cou\_st); getchar();

printf("Enter the end of class time:");

scanf("%hu", cou\_ed); getchar();

printf("Enter the number of weeks in class:\n");

count = 0;

do {

printf(" Starting week:");

scanf("%hu", &wk\_st[count]); getchar();

printf("End week:");

scanf("%hu", &wk\_ed[count]); getchar();

printf("Continue? Yes--Y\tNo--other keys\n");

scanf("%c", &j); getchar(); count++;

}while(j=='Y' || j=='y');

return TRUE;

}