

程序设计报告

（2023/2024学年 第一学期）

题 目：**实验室预约程序**

|  |  |
| --- | --- |
| **专 业** |  |
| **组长 学号姓名** |  |
| **组员 学号姓名** |  |
|  |  |
| **指 导 教 师** |  |
| **指 导 单 位** | **计算机学院、软件学院、网络空间安全学院** |
| **日 期** | **2023年 12月 15日** |

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **成员分工** | | | | | | | | | |
|  | | 1. 主编程手，负责设计实验室信息导入，输出实验室信息，将预约信息写到文件，读取预约记录，取消预约，输出课表等模块。 2. 整合组员思路及代码，美化程序，调试运行。 3. 撰写程序设计报告，绘制流程图。 | | | | | | | |
|  | | 1. 辅助编程，负责预约模块，包括预约，预约输入信息的合法性，冲突性检查，预约逻辑判断等模块。 2. 修复组长在调试时发现的错误，对程序进行完善。 3. 提供代码测试数据，提出程序部分模块的重要想法。 | | | | | | | |
|  | | 1. 辅助编程，负责预约查询模块，包括按日期/实验室编号输出预约信息，负责主菜单界面设计。 2. 整理代码，完善部分注释，增加代码可读性。 3. 撰写部分程序设计报告，提供程序运行截图。 | | | | | | | |
| **教师评价** | | | | | | | | | |
| 评价准则 | | | | 优秀 | 良好 | | 中等 | 合格 | 不合格 |
| 该课题组掌握程序设计开发的相关工程基础知识，并能够针对求解的工程问题，进行合理的分析与综合。 | | | |  |  | |  |  |  |
| 该课题组能够结合计算机软硬件资源，合理选用算法、数据结构、数据存储方式等技术手段，对求解的工程问题进行系统设计、建模及预测。 | | | |  |  | |  |  |  |
| 该课题组能够选择合适的程序设计语言与编程开发平台，对求解的工程问题进行系统模拟、仿真与预测。 | | | |  |  | |  |  |  |
| 该课题组熟练掌握调试方法与工具，对程序开发过程中出现的问题进行分析、跟踪与调试，并在开发平台上对最终作品进行充分测试。 | | | |  |  | |  |  |  |
| 该课题组具备初步的开发能力，完成了课题要求的各项开发任务，功能实现全面。 | | | |  |  | |  |  |  |
| 该课题组具备初步的界面设计意识。用户使用界面布局合理，交互友好，操作简便。 | | | |  |  | |  |  |  |
| 该课题组正确、完整的回答了指导教师关于课题作品的提问，反映出其对课题内容，以及相关的工程基础知识具有较好的理解和掌握。 | | | |  |  | |  |  |  |
| 该课题组具备一定的表达能力与文字处理能力，能够熟练使用文字处理软件完成课题报告一份。报告完整记录了课题组的工作及总结，内容详实，格式规范。 | | | |  |  | |  |  |  |
| 该课题组具有较好的团队协作精神，能够努力沟通解决遇到的各种开发、工程问题。 | | | |  |  | |  |  |  |
| 该课题组具备一定自学能力与探索创新意识，能够充分利用各种网络与图书馆资源自学新知识与新技能。 | | | |  |  | |  |  |  |
| 该课题组在上机过程中遵守机房规章制度，出勤与平时表现规范。 | | | | √ |  | |  |  |  |
| 最终作品和报告反映出该课题组具备较好的解决工程问题的能力。 | | | |  |  | |  |  |  |
| **本次程序设计能力达成评价** | | | | | | | | | |
| 学号姓名 |  | |  | | |  | | | |
| 贡献度及个人表现 |  | |  | | |  | | | |
| 最终成绩 |  | |  | | |  | | | |
| **备注：**  贡献度及个人表现：好、较好、中等、较差、差  最终成绩：优秀、良好、中等、及格、不及格 | | | | | | | | | |
| **指导教师： 年 月 日** | | | | | | | | | |

**实验室预约系统**

1. **课题内容和要求**

实验室预约系统是在MSVC2022编译器下，通过C++语言实现的，以Windows控制台为主要运行界面，帮助高校老师管理和预约学校实验室的便利型管理系统。用户可以自定义输入8个实验室的相关信息，系统将自动检测并导入这些信息。实验室分硬件实验室和软件实验室两个类型。每个实验室的机位有 40、60、80 台三种规模。每天可预约的时段是 1-9 节。预约天数以一周为期限。

1. 课题基本要求：

课题基本要求①：从文件导入实验室基本信息（包括实验室类型和规模以及一周的起止日期）。【实现。小组创新功能：检查导入信息的合理性】

课题基本要求②：实验室预约。输入预约的日期、时段（可以是连续的 2 节课或 3 节课）、所需机器数、实验室类型。程序筛选出符合条件的实验室，让用户选择预约。预约时，需填入课程名和学生班级。【实现。小组创新功能：显示所有预约信息，预约信息写入配置文件，增加取消预约功能】

课题基本要求③：实验室预约情况查询。输入日期，显示所有实验室可预约的时段。输入实验室编号，显示该实验室预约周期内每一天可预约的时段。【实现】

课题基本要求④：实验室课表导出。根据预约情况，自动生成每个实验室每一天的课表，导出到文件。【实现】

1. 课题选做要求：

课题选做要求①：对实验室基本信息的导入进行检查：若找不到导入文件，则为用户新建；若导入信息不合理，则提示用户修改。【实现】

课题选做要求②：大量数据以表格形式输出，并修改相关提示词的颜色，增加用户体验感。【实现】

课题选做要求③：显示所有预约信息，增加取消预约模块【实现】

课题选做要求④：在重要信息输出前进行一次清屏。【实现】

课题选做要求⑤：使用配置文件保存预约信息。【实现】

**二、需求分析**

实验室预约系统的功能框架图如图1所示。

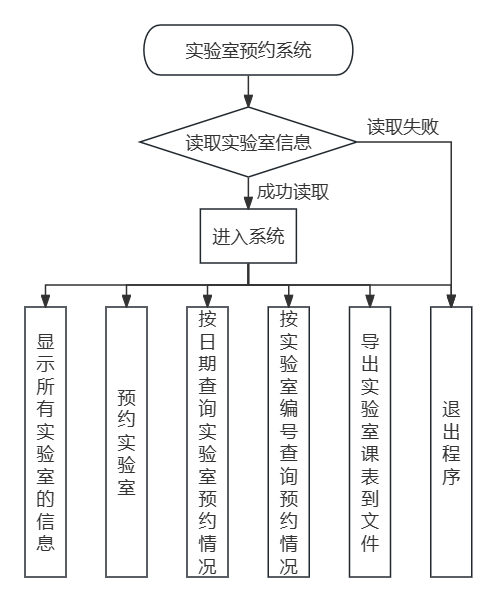


图1 实验室预约系统功能框架图

（1）提供可操作的主菜单：实验室预约系统主菜单，用于显示该系统的若干个可选功能选项。根据用户输入的选项来运行不同的功能，运行不同的函数以实现不同的功能。

**三、概要设计**

1. 主要存储结构

enum LabType // 实验室类型枚举

{

HARDWARE = 0, SOFTWARE = 1

};

enum LabSize // 实验室规模枚举

{

SMALL = 40,

MEDIUM = 60,

LARGE = 80

};

// 实验室信息结构体，用于储存从配置文件中读取的实验室信息数据

struct Lab

{

int labNumber; // 实验室编号

LabType labType; // 实验室类型

LabSize labSize; // 实验室规模

bool schedule[7][9]; // 二维数组表示每天每时段的预约情况

int startDate; // 开始时间

int endDate; // 结束时间

};

// 添加预约信息结构体

struct Reservation

{

string courseName; // 课程名称

string studentClass; // 班级名称

int date; // 日期

int startTime; // 开始时间

int endTime; // 结束时间

int requiredMachines; // 机器需求量

LabType labType; // 实验室类型

};

// 用于存储预约信息结构体

struct Appointment

{

int date; // 日期

int startTime; // 开始时间

int endTime; // 结束时间

int labNumber; // 实验室编号

int labType; // 实验室类型

int labSize; // 实验室规模

string lessonName; // 课程名称

string studentClass; // 班级名称

};

2. 主要函数流程

（1）预约实验室：在成功导入实验室信息后进入主菜单，选择2.预约实验室，进入预约实验室界面，用户输入课程名称，班级，预约日期(YYYYMMDD)，开始时间(1-9)，结束时间(1-9)，机器数量，实验室类型(0: 硬件, 1: 软件)，系统自动进行判断输入合法性和该时间段能否预约，并且自动筛选出第一个可供预约的实验室，写入配置文件appointment.txt，并提示用户预约成功。该函数的基本流程如下图2.

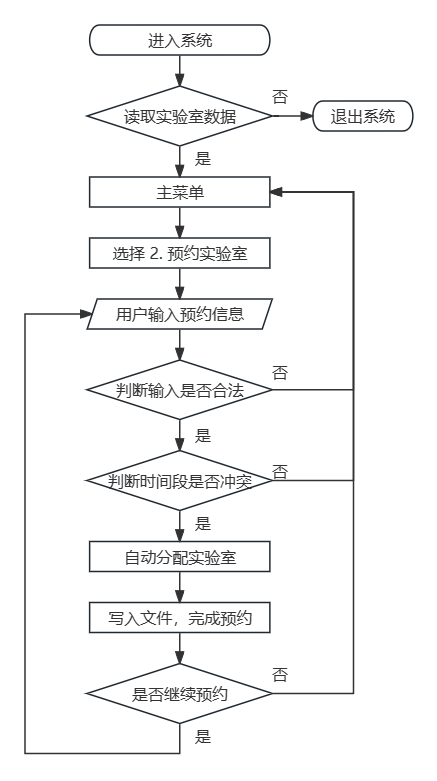


图2 预约实验室流程图

（2）按实验室编号查询预约情况：在成功导入实验室信息后进入主菜单，选择6.按实验室编号查询预约情况，进入查询界面。系统提示用户输入要查询的实验室编号(1-8)，用户完成输入后，系统首先会读取同一级目录下的appointment.txt获取所有预约信息。然后，系统会根据用户输入的实验室编号筛选出符合条件的预约数据，并且以表格的形式打印输出到屏幕上。然后系统会询问用户是否继续查询，“是”则清屏后继续查询，“否”则返回主界面。该函数的基本流程如下图3.

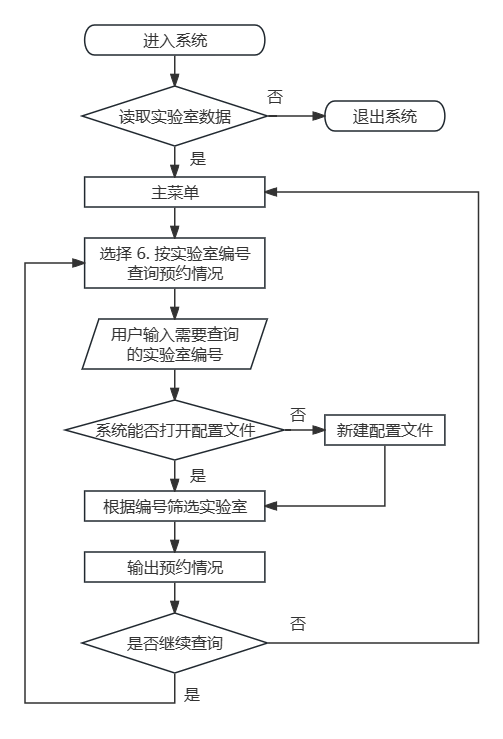


图3 按实验室编号查询预约情况流程图

（3）取消预约：在成功导入实验室信息后进入主菜单，选择3.取消预约，进入取消界面，系统提示用户输入取消预约的日期(YYYYMMDD)，课程开始时间(1-9)，课程结束时间(1-9)。用户完成输入后，系统首先会读取同一级目录下的appointment.txt获取所有预约信息。然后根据用户输入的信息进行筛选，若找不到预约记录，则打印错误信息“取消预约失败，未找到匹配的预约信息!”。若能找到预约记录，则先将该条记录移动到appointment.txt的末尾（与最后一行记录交换位置），即最后一行，然后删除最后一行，实现取消预约的操作。该函数的基本流程如下图4.

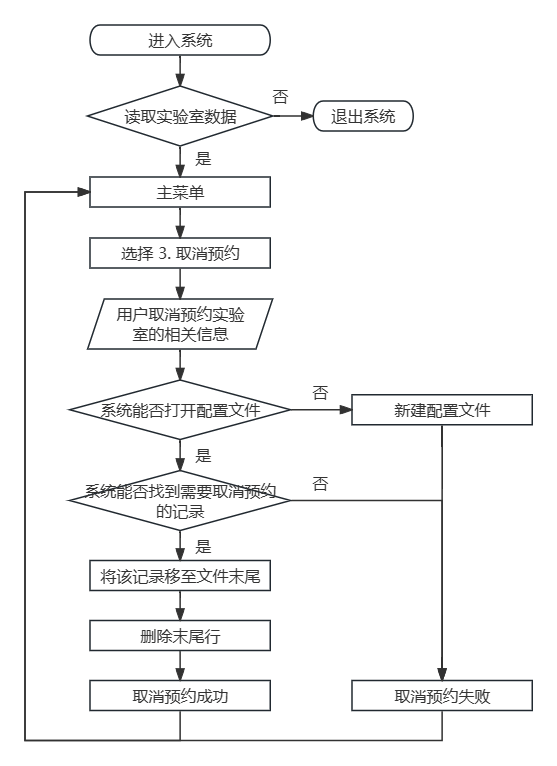


图4 取消预约流程图

**四、源程序代码**

#include <iostream> //输入输出头文件

#include <cstdlib> //标准库头文件

#include <fstream> //文件操作头文件

#include <string> //字符串操作头文件

#include <sstream> //字符串流头文件

#include <Windows.h> //windows头文件

#define MAX\_LAB 8 // 实验室最大数量

#define MAX\_APPOINTMENTS 200 // 预约最大条数

// 定义输出颜色

#define INFO\_BLUE "\033[94m"

#define WARNING\_YELLOW "\033[93m"

#define ERROR\_RED "\033[91m"

#define RESET "\033[0m"

using namespace std;

// 实验室类型枚举

enum LabType

{

HARDWARE = 0, // 硬件

SOFTWARE = 1 // 软件

};

// 实验室规模枚举

enum LabSize

{

SMALL = 40,

MEDIUM = 60,

LARGE = 80

};

// 实验室信息结构体

struct Lab

{

int labNumber; // 实验室编号

LabType labType; // 实验室类型

LabSize labSize; // 实验室规模

bool schedule[7][9]; // 二维数组表示每天每时段的预约情况

int startDate; // 开始时间

int endDate; // 结束时间

};

// 添加预约信息结构体

struct Reservation

{

string courseName; // 课程名称

string studentClass; // 班级名称

int date; // 日期

int startTime; // 开始时间

int endTime; // 结束时间

int requiredMachines; // 机器需求量

LabType labType; // 实验室类型

};

// 存储预约信息结构体

struct Appointment

{

int date; // 日期

int startTime; // 开始时间

int endTime; // 结束时间

int labNumber; // 实验室编号

int labType; // 实验室类型

int labSize; // 实验室规模

string lessonName; // 课程名称

string studentClass; // 班级名称

};

// 函数声明

int importLabInfo(Lab labs[], int labCount);

void showLabInfo(const Lab labs[], int labCount);

void makeReservation(Lab labs[], int labCount, Reservation& reservation);

void writeAppointmentToFile(const Reservation& reservation, Lab labs[], int labCount);

bool makeReservationLogic(const Lab labs[], int labCount, int date, int startTime, int endTime, int requiredMachines);

bool makeReservationValuable(const Lab labs[], int labCount, const Reservation& reservation);

int readAppointment(Appointment appointments[], int maxAppointments);

void showAppointment(const Appointment appointments[], int count);

void cancelAppointment(Appointment appointments[], int& count);

void displayLabScheduleDate(const Appointment appointments[], int appointmentCount, int date);

void displayLabScheduleLab(const Appointment appointments[], int appointmentCount, int labNumber);

void exportLabSchedule(const Appointment appointments[], int appointmentCount, const Lab labs[], int labCount);

void showmenu();

void choosechoice(int choice, Lab labs[], Appointment appointments[]);

// 导入实验室信息，成功导入则返回实验室总数，导入失败返回0

int importLabInfo(Lab labs[], int labCount)

{

// 打开文件

ifstream inputFile("input.txt");

// 如果文件不存在，则创建文件并提示用户输入

if (!inputFile.is\_open())

{

ofstream outputFile("input.txt");

outputFile << "LabNumber,LabType,LabSize,StartDate,EndDate" << endl;

cerr << ERROR\_RED << "[ERROR]:文件 input.txt 不存在，已为您创建，请将实验室信息输入该文件中后重试" << RESET << endl;

cout << endl;

cout << INFO\_BLUE << "[INFO]:按任意键关闭程序" << RESET << endl;

cin.get();

return 0;

}

// 读取标题行，但不使用

string line;

getline(inputFile, line);

int count = 0;

int lineCount = 0; // 用于计数文件的行数

// 读取文件中的每一行，并将其转换为Lab对象

while (getline(inputFile, line) && count < labCount)

{

lineCount++; // 每成功读取一行，行数加一

stringstream ss(line);

string token;

// 逐个读取CSV格式的值

getline(ss, token, ','); // LabNumber

labs[count].labNumber = stoi(token);

getline(ss, token, ','); // LabType

labs[count].labType = static\_cast<LabType>(stoi(token));

getline(ss, token, ','); // LabSize

labs[count].labSize = static\_cast<LabSize>(stoi(token));

getline(ss, token, ','); // StartDate

labs[count].startDate = stoi(token);

getline(ss, token, ','); // EndDate

labs[count].endDate = stoi(token);

// 将每个实验室的每个时间段都初始化为false

for (int i = 0; i < 7; ++i)

{

for (int j = 0; j < 9; ++j)

{

labs[count].schedule[i][j] = false;

}

}

count++;

}

// 关闭文件

inputFile.close();

// 行数检查

if (lineCount < 8 || lineCount > 9)

{

cerr << ERROR\_RED << "[ERROR]:文件行数不正确，请确保完整输入了8个实验室的数据" << RESET << endl;

system("pause");

return false;

}

cout << INFO\_BLUE << "[INFO]:已成功读取到实验室信息..." << RESET << endl;

return labCount;

}

// 输出实验室信息

void showLabInfo(const Lab labs[], int labCount)

{

// 输出表头

cout << WARNING\_YELLOW << "编号\t类型\t容量\t开始时间\t结束时间" << RESET << endl;

cout << "────────────────────────────────────────────────────" << endl;

// 输出实验室信息

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

{

cout << labs[i].labNumber << '\t'

<< (labs[i].labType == HARDWARE ? "硬件" : "软件") << '\t'

<< labs[i].labSize << '\t'

<< labs[i].startDate << '\t'

<< labs[i].endDate << endl;

}

cout << "────────────────────────────────────────────────────" << endl;

}

// 预约实验室

void makeReservation(Lab labs[], int labCount, Reservation& reservation)

{

// 声明一个变量，用于存储用户输入的实验室类型

int labTypeInput;

cout << INFO\_BLUE << "[INFO]:请按顺序输入下述内容，并使用空格隔开" << RESET << endl;

cout << endl;

cout << "课程名称 班级 预约日期(YYYYMMDD) 开始时间(1-9) 结束时间(1-9) 机器数量 实验室类型(0: 硬件, 1: 软件)" << endl;

// 获取用户输入的信息

cin >> reservation.courseName >> reservation.studentClass >> reservation.date >> reservation.startTime >> reservation.endTime >> reservation.requiredMachines >> labTypeInput;

// 检查用户输入的机器数量是否合理并按需求赋值(40,60,80)

if (reservation.requiredMachines > 0 && reservation.requiredMachines <= 40)

{

reservation.requiredMachines = 40;

}

if (reservation.requiredMachines > 40 && reservation.requiredMachines <= 60)

{

reservation.requiredMachines = 60;

}

if (reservation.requiredMachines > 60 && reservation.requiredMachines <= 80)

{

reservation.requiredMachines = 80;

}

if (reservation.requiredMachines <= 0 && reservation.requiredMachines >= 80)

{

cerr << ERROR\_RED << "[ERROR]:机器数量输入不合理，请输入(0-80)" << endl;

return;

}

// 检查用户输入的实验室类型是否合理

if (labTypeInput == 0 || labTypeInput == 1)

{

reservation.labType = static\_cast<LabType>(labTypeInput);

}

else

{

cerr << ERROR\_RED << "[ERROR]:实验室类型输入不合理，请输入(0: 硬件, 1: 软件)" << endl;

return;

}

// 调用 makeReservationLogic 函数，检查预约是否合理

if (makeReservationLogic(labs, labCount, reservation.date, reservation.startTime, reservation.endTime, reservation.requiredMachines) && makeReservationValuable(labs, labCount, reservation))

{

// 调用 writeAppointmentToFile 函数，将预约信息写入文件

writeAppointmentToFile(reservation, labs, labCount);

}

}

// 将预约信息写到文件

void writeAppointmentToFile(const Reservation& reservation, Lab labs[], int labCount)

{

ofstream outFile("appointment.txt", ios::app); // 打开文件，追加写入

if (!outFile.is\_open())

{

// 如果文件不存在，则创建文件

ofstream outputFile("appointment.txt");

outputFile << "Date,StartTime,EndTime,LabNumber,LabType,LabSize,LessonName,Class" << endl;

cerr << WARNING\_YELLOW << "[WARNING]:未找到 appointment.txt，已重新创建" << RESET << endl;

}

// 查找符合条件的实验室，选择第一个符合条件的实验室

int selectedLabIndex = -1;

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

{

if (labs[i].labType == reservation.labType && labs[i].labSize >= reservation.requiredMachines)

{

selectedLabIndex = i;

break;

}

}

if (selectedLabIndex != -1)

{

// 写入预约信息到文件

outFile << reservation.date << "," << reservation.startTime << ","

<< reservation.endTime << "," << labs[selectedLabIndex].labNumber << ","

<< static\_cast<int>(labs[selectedLabIndex].labType) << "," << static\_cast<int>(labs[selectedLabIndex].labSize) << ","

<< reservation.courseName << "," << reservation.studentClass << "\n";

// 更新实验室的预约情况，将对应时段标记为已预约

for (int i = reservation.startTime - 1; i < reservation.endTime; ++i)

{

labs[selectedLabIndex].schedule[reservation.date % 7][i] = true;

}

cout << INFO\_BLUE << "[INFO]:预约成功！" << RESET << endl;

}

else

{

cerr << ERROR\_RED << "[ERROR]:没有符合条件的实验室可供预约" << endl;

}

outFile.close(); // 关闭文件

}

// 判断预约信息逻辑

bool makeReservationLogic(const Lab labs[], int labCount, int date, int startTime, int endTime, int requiredMachines)

{

// 遍历所有实验室

for (int labIndex = 0; labIndex < labCount; ++labIndex)

{

const Lab& lab = labs[labIndex];

int reservationDate = date; // 预约日期的格式:YYYMMDD

// 检查预约日期是否在实验室有效范围内

if (reservationDate < lab.startDate || reservationDate > lab.endDate)

{

cerr << ERROR\_RED << "[ERROR]:无效的预约日期" << endl;

return false;

}

// 检查预约时间是否在有效范围内

if (startTime < 1 || startTime > 9 || endTime < 1 || endTime > 9 || endTime <= startTime)

{

cerr << ERROR\_RED << "[ERROR]:无效的预约时间" << endl;

return false;

}

}

// 如果通过了所有实验室的检查，表示预约成功

return true;

}

// 判断预约输入是否有效

bool makeReservationValuable(const Lab labs[], int labCount, const Reservation& reservation)

{

// 读取 "appointment.txt" 文件

ifstream inputFile("appointment.txt");

if (!inputFile.is\_open())

{

// 如果文件不存在，则创建文件

ofstream outputFile("appointment.txt");

outputFile << "Date,StartTime,EndTime,LabNumber,LabType,LabSize,LessonName,Class" << endl;

cerr << WARNING\_YELLOW << "[WARNING]:未找到 appointment.txt，已重新创建" << RESET << endl;

}

string line;

while (getline(inputFile, line))

{

stringstream ss(line);

char comma;

// 逐个读取CSV格式的值

int date, startTime, endTime, labNumber, labType, labSize;

string lessonName, studentClass;

ss >> date >> comma >> startTime >> comma >> endTime >> comma >> labNumber >> comma >> labType >> comma >> labSize >> comma >> lessonName >> comma >> studentClass;

// 检查实验室、日期和时间是否与现有的预约冲突

if (labType == static\_cast<int>(reservation.labType) &&

date == reservation.date &&

startTime <= reservation.endTime &&

endTime >= reservation.startTime)

{

cerr << ERROR\_RED << "[ERROR]:预约时间冲突，无法完成预约" << endl;

inputFile.close();

return false;

}

}

inputFile.close();

// 如果所有检查都通过，预约就是有效的

return true;

}

// 读取预约记录文件，返回记录条数

int readAppointment(Appointment appointments[], int maxAppointments)

{

// 打开文件

ifstream inputFile("appointment.txt");

// 如果文件不存在，则创建文件

if (!inputFile.is\_open())

{

// 如果文件不存在，则创建文件

ofstream outputFile("appointment.txt");

outputFile << "Date,StartTime,EndTime,LabNumber,LabType,LabSize,LessonName,Class" << endl;

cerr << WARNING\_YELLOW << "[WARNING]:未找到 appointment.txt，已重新创建" << RESET << endl;

return 0;

}

// 读取文件中的记录

int count = 0;

string line;

getline(inputFile, line); // 读取标题行，但不使用

while (count < maxAppointments && getline(inputFile, line))

{

stringstream ss(line);

char comma; // 用于读取逗号分隔符

// 逐个读取CSV格式的值

ss >> appointments[count].date >> comma >> appointments[count].startTime >> comma >> appointments[count].endTime >> comma >> appointments[count].labNumber >> comma >> appointments[count].labType >> comma >> appointments[count].labSize >> comma >> appointments[count].lessonName >> comma >> appointments[count].studentClass;

count++;

}

inputFile.close();

return count; // 返回记录条数

}

// 打印所有预约信息

void showAppointment(const Appointment appointments[], int count)

{

// 打印预约信息表格的表头

cout << WARNING\_YELLOW << "日期\t\t时间\t实验室编号\t实验室类型\t实验室容量\t课程名称及班级" << RESET << endl;

cout << "─────────────────────────────────────────────────────────────────────────────────────────" << endl;

// 遍历所有预约信息，并打印

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

{

cout << appointments[i].date << '\t' << appointments[i].startTime << ' ' << '-' << ' ' << appointments[i].endTime

<< '\t' << appointments[i].labNumber << '\t' << '\t' << (appointments[i].labType == 0 ? "硬件" : "软件")

<< '\t' << '\t' << appointments[i].labSize << '\t' << '\t' << appointments[i].lessonName << appointments[i].studentClass << endl;

}

}

// 取消预约

void cancelAppointment(Appointment appointments[], int& count)

{

// 输入要取消的预约信息

int cancelDate, cancelStartTime, cancelEndTime;

cout << INFO\_BLUE << "[INFO]:请按顺序输入下述内容，并使用空格隔开" << RESET << endl;

cout << endl;

cout << "取消预约的日期(YYYYMMDD) 课程开始时间(1-9) 课程结束时间(1-9)" << endl;

cin >> cancelDate >> cancelStartTime >> cancelEndTime;

// 使用 remove\_if 移动匹配的元素到末尾

auto it = remove\_if(appointments, appointments + count,

[cancelDate, cancelStartTime, cancelEndTime](const Appointment& app)

{

return app.date == cancelDate && app.startTime == cancelStartTime && app.endTime == cancelEndTime;

});

// 根据 remove\_if 的返回值计算新的有效元素数量

int newCount = distance(appointments, it);

if (newCount < count)

{

cout << endl;

cout << INFO\_BLUE << "[INFO]:取消预约成功！" << RESET << endl;

// 将预约信息写入文件

ofstream outFile("appointment.txt");

if (outFile.is\_open())

{

outFile << "Date,StartTime,EndTime,LabNumber,LabType,LabSize,LessonName,Class" << endl;

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

{

outFile << appointments[i].date << ","

<< appointments[i].startTime << ","

<< appointments[i].endTime << ","

<< appointments[i].labNumber << ","

<< appointments[i].labType << ","

<< appointments[i].labSize << ","

<< appointments[i].lessonName << ","

<< appointments[i].studentClass << endl;

}

outFile.close();

}

count = newCount;

}

else

{

cout << endl;

cerr << ERROR\_RED << "[ERROR]:取消预约失败，未找到匹配的预约信息!" << RESET << endl;

}

}

// 按日期输出预约信息

void displayLabScheduleDate(const Appointment appointments[], int appointmentCount, int date)

{

// 创建二维数组来表示预约情况

const int MAX\_LABS = 8;

const int MAX\_TIMESLOTS = 9;

// 创建二维数组来表示预约情况

char schedule[MAX\_LABS][MAX\_TIMESLOTS];

// 初始化预约情况数组

for (int i = 0; i < MAX\_LABS; ++i)

{

for (int j = 0; j < MAX\_TIMESLOTS; ++j)

{

schedule[i][j] = ' ';

}

}

// 根据预约信息更新预约情况数组

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

{

if (appointments[i].date == date)

{

int labIndex = appointments[i].labNumber - 1;

int startTime = appointments[i].startTime - 1;

int endTime = appointments[i].endTime;

for (int j = startTime; j < endTime; ++j)

{

schedule[labIndex][j] = 'O';

}

}

}

// 输出日期和预约情况

cout << endl;

cout << WARNING\_YELLOW << "日期 " << INFO\_BLUE << date << WARNING\_YELLOW << " 预约信息如下：" << RESET << endl;

cout << endl;

cout << WARNING\_YELLOW << "实验室\t\t1\t2\t3\t4\t5\t6\t7\t8" << RESET << endl;

cout << "────────────────────────────────────────────────────────────────────────────────" << endl;

for (int i = 0; i < MAX\_TIMESLOTS; ++i)

{

cout << "时间段" << i + 1 << "\t\t";

for (int j = 0; j < MAX\_LABS; ++j)

{

if (schedule[j][i] == ' ')

{

cout << "可预约\t";

}

else

{

cout << schedule[j][i] << "\t";

}

}

cout << endl;

}

cout << "────────────────────────────────────────────────────────────────────────────────" << endl;

}

// 按实验室编号输出预约信息

void displayLabScheduleLab(const Appointment appointments[], int appointmentCount, int labNumber)

{

const int MAX\_TIMESLOTS = 9;

// 创建二维数组来表示预约情况

char schedule[MAX\_TIMESLOTS][7];

// 初始化预约情况数组

for (int i = 0; i < MAX\_TIMESLOTS; ++i)

{

for (int j = 0; j < 7; ++j)

{

schedule[i][j] = ' ';

}

}

// 根据预约信息更新预约情况数组

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

{

if (appointments[i].labNumber == labNumber)

{

int startTime = appointments[i].startTime - 1;

int endTime = appointments[i].endTime;

for (int j = startTime; j < endTime; ++j)

{

int dayOfWeek = appointments[i].date % 7;

schedule[j][dayOfWeek] = 'O';

}

}

}

// 输出实验室编号和预约情况

cout << endl;

cout << WARNING\_YELLOW << "实验室 " << INFO\_BLUE << labNumber << WARNING\_YELLOW << " 的预约信息如下：" << RESET << endl;

cout << endl;

cout << "星期\t\t1\t2\t3\t4\t5\t6\t7" << endl;

cout << "────────────────────────────────────────────────────────────────────────" << endl;

for (int i = 0; i < MAX\_TIMESLOTS; ++i)

{

cout << "时间段" << i + 1 << "\t\t";

for (int j = 0; j < 7; ++j)

{

if (schedule[i][j] == ' ')

{

cout << "可预约\t";

}

else

{

cout << schedule[i][j] << "\t";

}

}

cout << endl;

}

cout << "────────────────────────────────────────────────────────────────────────" << endl;

}

// 导出实验室课表

void exportLabSchedule(const Appointment appointments[], int appointmentCount, const Lab labs[], int labCount)

{

// 每个时间段有9个时间段

const int MAX\_TIMESLOTS = 9;

// 每个星期有7个时间段

const int MAX\_DAYS = 7;

// 创建文件输出流

ofstream outputFile("output.txt");

// 遍历每个实验室

for (int labIndex = 0; labIndex < labCount; ++labIndex)

{

const Lab& lab = labs[labIndex];

outputFile << " ";

outputFile << "实验室" << lab.labNumber << "，类型：";

outputFile << (lab.labType == HARDWARE ? "硬件" : "软件");

outputFile << "，机位总数：" << lab.labSize << "，课表" << endl;

outputFile << endl;

// 创建二维数组来表示课表

string schedule[MAX\_TIMESLOTS][MAX\_DAYS];

// 初始化课表为空

for (int i = 0; i < MAX\_TIMESLOTS; ++i)

{

for (int j = 0; j < MAX\_DAYS; ++j)

{

schedule[i][j] = "";

}

}

// 填充课表

for (int appointmentIndex = 0; appointmentIndex < appointmentCount; ++appointmentIndex)

{

const Appointment& appointment = appointments[appointmentIndex];

// 如果预约的实验室与当前实验室相同

if (appointment.labNumber == lab.labNumber)

{

int startTime = appointment.startTime - 1;

int endTime = appointment.endTime;

// 遍历预约的时间段

for (int j = startTime; j < endTime; ++j)

{

int dayOfWeek = appointment.date % 7;

schedule[j][dayOfWeek] = appointment.lessonName;

}

}

}

// 输出课表到文件

outputFile << "─────────────────────────────────────────────────────────────────────────────────────────────" << endl;

outputFile << "星期\t\t1\t\t2\t\t3\t\t4\t\t5\t\t6\t\t7" << endl;

for (int i = 0; i < MAX\_TIMESLOTS; ++i)

{

outputFile << "时间段" << i + 1 << "\t\t";

for (int j = 0; j < MAX\_DAYS; ++j)

{

if (schedule[i][j] != "")

{

outputFile << schedule[i][j];

}

outputFile << "\t\t";

}

outputFile << endl;

}

outputFile << "================================================================================================================================" << endl;

outputFile << endl;

outputFile << endl;

}

outputFile.close();

}

// 打印主菜单

void showmenu()

{

cout << "──────────────────────────────" << WARNING\_YELLOW << " 欢迎使用实验室预约系统! " << RESET << "───────────────────────────────" << endl;

cout << " 当前版本:v1.2.1214 " << endl;

cout << " 1. 显示所有实验室的信息" << endl;

cout << " 2. 预约实验室" << endl;

cout << " 3. 取消预约" << endl;

cout << " 4. 显示所有预约信息" << endl;

cout << " 5. 按日期查询实验室预约情况" << endl;

cout << " 6. 按实验室编号查询预约情况" << endl;

cout << " 7. 导出实验室课表到文件" << endl;

cout << " 0. 退出程序" << endl;

cout << endl;

cout << "──────────────────────────────────────────────────────────────────────────────────────" << endl;

cout << endl;

cout << INFO\_BLUE << "[INFO]:请输入你的选择[0-4]:" << RESET;

return;

}

// 处理用户选择

void choosechoice(int choice, Lab labs[], Appointment appointments[])

{

switch (choice)

{

case 1:

{

cout << WARNING\_YELLOW << "──────────────── 当前位于：1.显示所有实验室的信息 ────────────────" << RESET << endl;

cout << endl;

int labCount;

labCount = importLabInfo(labs, MAX\_LAB);

cout << endl;

showLabInfo(labs, labCount);

cout << endl;

cout << INFO\_BLUE << "[INFO]:按任意键返回主界面..." << RESET << endl;

cin.get();

cin.get();

system("cls");

break;

}

case 2:

{

char temp = ' ';

do

{

cout << WARNING\_YELLOW << "──────────────── 当前位于：2.预约实验室 ────────────────" << RESET << endl;

cout << endl;

Reservation reservation;

makeReservation(labs, MAX\_LAB, reservation);

cout << endl;

cout << INFO\_BLUE << "[INFO]:继续预约?(Y/N)" << RESET;

cin >> temp;

system("cls");

} while (temp == 'Y' || temp == 'y');

break;

}

case 3:

{

cout << WARNING\_YELLOW << "──────────────── 当前位于：3.取消预约 ────────────────" << RESET << endl;

cout << endl;

int count = readAppointment(appointments, MAX\_APPOINTMENTS);

cancelAppointment(appointments, count);

cout << endl;

cout << INFO\_BLUE << "[INFO]:按任意键返回主界面..." << RESET << endl;

cin.get();

cin.get();

system("cls");

break;

}

case 4:

{

cout << WARNING\_YELLOW << "──────────────── 当前位于：4.显示所有预约信息 ────────────────" << RESET << endl;

cout << endl;

int count = readAppointment(appointments, MAX\_APPOINTMENTS);

showAppointment(appointments, count);

cout << endl;

cout << INFO\_BLUE << "[INFO]:按任意键返回主界面..." << RESET << endl;

cin.get();

cin.get();

system("cls");

break;

}

case 5:

{

char temp = ' ';

do

{

cout << WARNING\_YELLOW << "──────────────── 当前位于：5. 按日期查询实验室预约情况 ────────────────" << RESET << endl;

int date;

cout << endl;

cout << "[INFO]:请输入要查询的日期(YYYMMDD): ";

cin >> date;

int appointmentCount = readAppointment(appointments, MAX\_APPOINTMENTS);

displayLabScheduleDate(appointments, appointmentCount, date);

cout << endl;

cout << INFO\_BLUE << "[INFO]:继续查询?(Y/N)" << RESET;

cin >> temp;

system("cls");

} while (temp == 'Y' || temp == 'y');

break;

}

case 6:

{

char temp = ' ';

do

{

cout << WARNING\_YELLOW << "──────────────── 当前位于：6. 按实验室编号查询预约情况 ────────────────" << RESET << endl;

cout << endl;

int labindex;

cout << "[INFO]:请输入要查询的实验室编号(1-8): ";

cin >> labindex;

int appointmentCount = readAppointment(appointments, MAX\_APPOINTMENTS);

displayLabScheduleLab(appointments, appointmentCount, labindex);

cout << endl;

cout << INFO\_BLUE << "[INFO]:继续查询?(Y/N)" << RESET;

cin >> temp;

system("cls");

} while (temp == 'Y' || temp == 'y');

break;

}

case 7:

{

cout << WARNING\_YELLOW << "──────────────── 当前位于：7. 导出实验室课表到文件 ────────────────" << RESET << endl;

cout << endl;

int appointmentCount = readAppointment(appointments, MAX\_APPOINTMENTS);

exportLabSchedule(appointments, appointmentCount, labs, MAX\_LAB);

cout << "已成功输出课表到output.txt中";

cout << endl;

cout << INFO\_BLUE << "[INFO]:按任意键返回主界面..." << RESET << endl;

cin.get();

cin.get();

system("cls");

break;

}

case 0:

{

cout << INFO\_BLUE << "[INFO]:欢迎下次使用，再见！" << endl;

MessageBox(NULL, L"欢迎下次使用，再见！", L"退出", MB\_OK);

return;

}

}

}

int main()

{

// 定义数组，用于存储实验室信息

Lab labs[MAX\_LAB];

// 定义数组，用于存储预约信息

Appointment appointments[MAX\_APPOINTMENTS];

// 读取实验室信息

if (!importLabInfo(labs, MAX\_LAB))

{

return 0;

}

Sleep(500);

system("cls");

// 定义一个变量，用于存储用户输入的选项

int choice;

// 循环执行，直到用户输入0

do

{

// 显示菜单

showmenu();

// 获取用户输入的选项

cin >> choice;

system("cls");

// 根据用户输入的选项，调用choosechoice函数

choosechoice(choice, labs, appointments);

} while (choice != 0);

return 0;

}

**五、测试数据及其结果分析**

1. 用户在input.txt中输入8个实验室相关的数据后，查看当前预约情况，再进行预约。

步骤一：用户在input.txt中输入以下实验室数据，如图4所示：

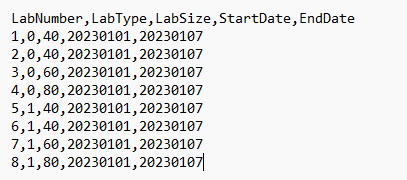


图4 input.txt中的信息

步骤二：程序成功读取到实验室信息后，向用户展示功能选择界面，如图5上半所示。此时，用户选择1.显示所有实验室信息，则系统将显示从input.txt中读取到的所有实验室信息，如图5下半所示。

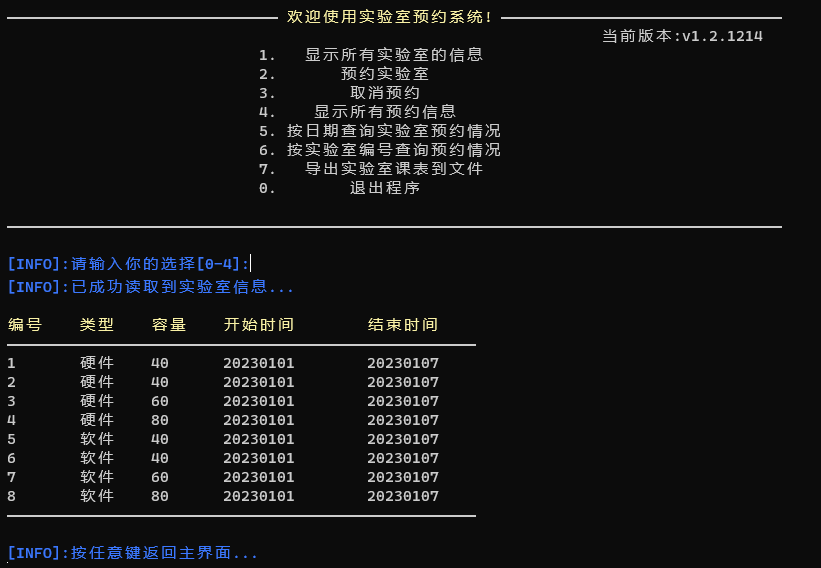


图5 功能选择界面及实验室信息的显示

步骤三：用户选择 4.显示所有预约信息 查看一下当前所有预约情况。此时系统会读取appointment.txt里面存储的预约信息，并且以表格的形式显示到频幕上来，如图6所示。



图6 显示所有预约信息

步骤四：查看完所有预约信息后，用户按任意键返回主菜单，然后选择 5.按日期查询预约情况。在下一级菜单中，用户输入需要查询的日期。以2023年1月1日为例，用户输入数字20230101，然后按回车，如图7上半所示。然后，系统将以表格的形式向用户展示出20230101这一天所有的预约情况（此处被预约的实验室和时间段会以O填充，可预约的实验室和时间段将显示“可预约”）如图7下半所示。

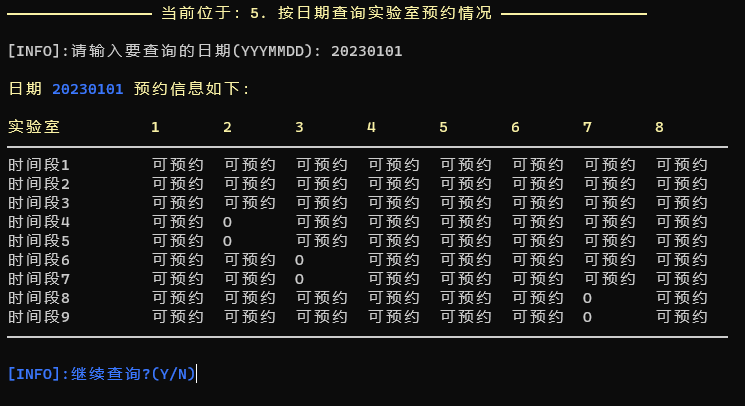


图7 查询20230101日实验室预约情况

步骤五：了解到相关预约信息后，用户首先输入“N”返回主菜单，然后选择 2.预约实验室，进入预约实验室界面。此时系统提示用户输入预约的相关信息，例如课程名称，班级，预约时间等。假设用户在经过上述步骤四的查询之后，想预约20230101这一天，时间段为1-5，实验室类型为1.软件，所需机器数量为40，并且课程名称和班级分别为LessonTest和1234。那么，用户只需要输入“LessonTset 1234 20230101 1 5 40 1”，然后按回车，系统提示“预约成功！”。整个预约过程如下图8所示。

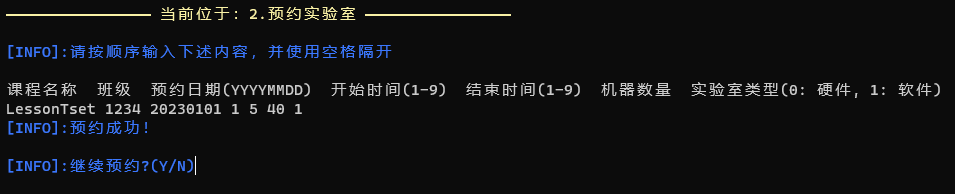


图8 预约实验室

步骤六：为了模拟用户预约失败的场景，假设用户继续预约，输入“Y”。此时，用户输入一个已经被预约的时间段，例如时间段8-9，实验室类型为1.软件，所需机器数量为60。根据步骤四可以得知，此预约是不成立的。我们假设用户输入了“TsetError 1234 20230101 8 9 60 1”然后按回车，会发现系统提示“预约时间冲突，无法完成预约”，如图9所示。

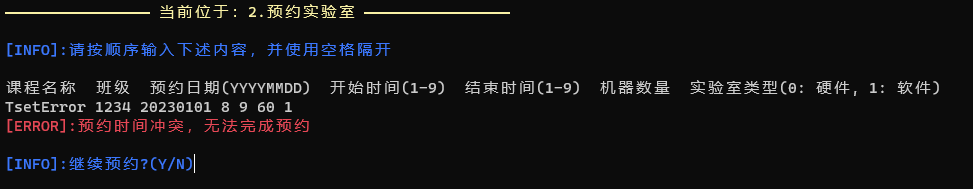


图9 时间冲突预约失败

步骤七：回到上述的步骤五，用户在成功预约2023年1月1日，时间段为1-5的课程后，系统自动为用户安排了实验室。此时用户返回主菜单，选择 4.显示所有预约信息 即可查看到新增的预约信息，如图10所示。

结果分析：与上述图6相比，我们可以看到，预约信息的最后一行已经成功加入了该用户在步骤五中添加的预约，并且系统也自动为用户分配了实验室5。

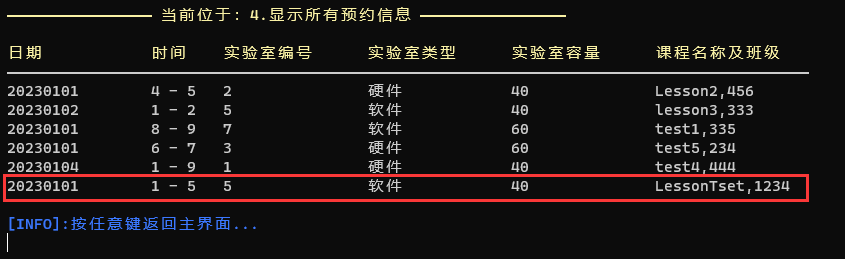


图10 预约成功后的所有预约信息

步骤八：之后，用户又返回到主菜单，选择 5.按日期查询预约情况 并输入20230101以查看该时间段是否已经变成“已预约”状态，如图11所示。

结果分析：对比下图11和图7，可以看到，系统已经成功读取到了用户的预约信息，并且在20230101这一天的预约查询情况表中用“O”标记出了用户预约的时间段（1-5）和系统自动分配的实验室5。

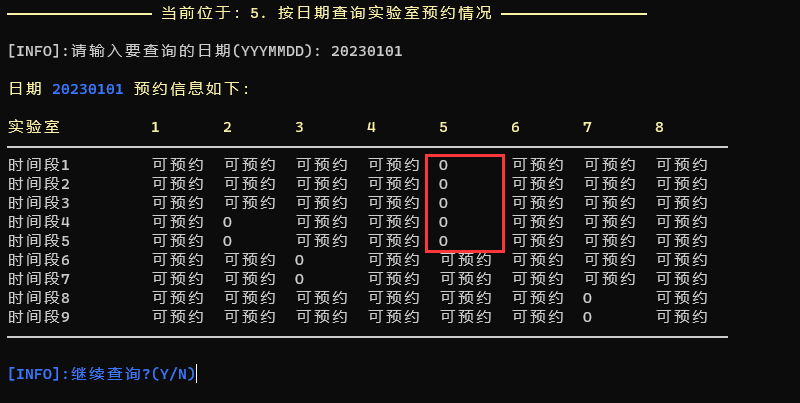


图11 预约成功后20230101日实验室预约情况

2. 用户取消预约实验室。

步骤一：用户打开程序并进入主界面。此时用户选择 3.取消预约 进入取消预约界面。如图12所示。

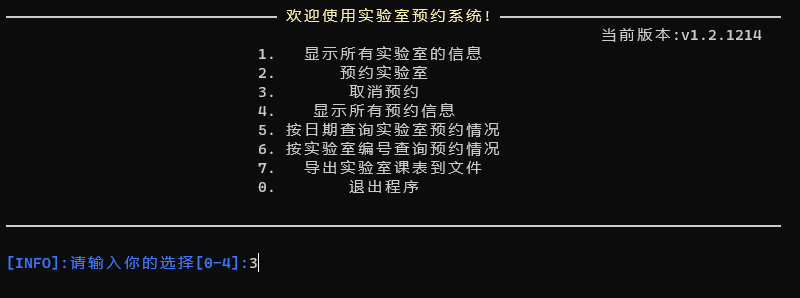


图12 功能选择界面

步骤二：我们假设用户想要取消测试数据1中步骤五预约的2023年1月1日，时间段为1-5的课程。根据系统提示信息，用户输入需要取消预约的日期，开始时间和结束时间，即“20230101 1 5”然后按回车，系统提示“取消预约成功！”如图13所示。

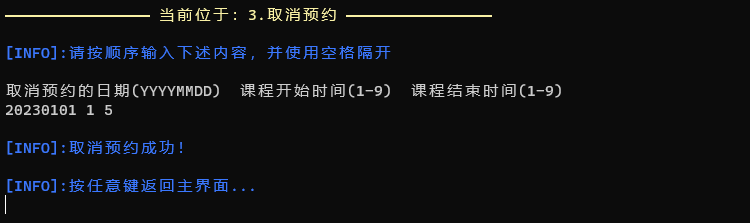


图13 取消预约

步骤三：此时，用户按任意键返回主界面后，选择 4.显示所有预约信息 系统显示当前所有预约信息（如图14所示）。与上面的图10相比，已经没有了日期为20230101，时间段1-5，实验室5的预约信息，说明取消预约成功。

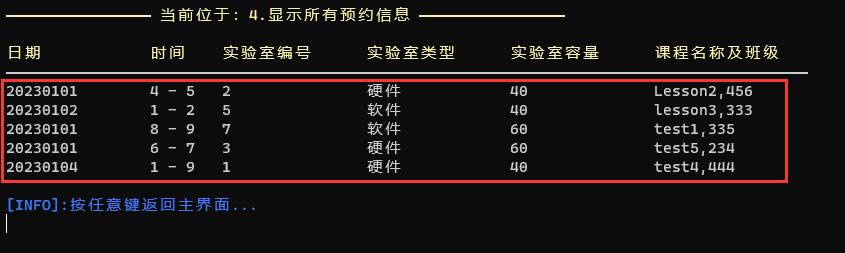


图14 取消预约后的所有预约信息

步骤四：用户继续返回主界面，选择 5.按日期查询实验室预约情况，并输入20230101以查看该时间段是否成功取消预约。系统显示取消预约后20230101当日的预约信息，如图15所示。与上面的图11相比，发现实验室5时间段1-5已由“O”变为“可预约”。说明取消预约成功。

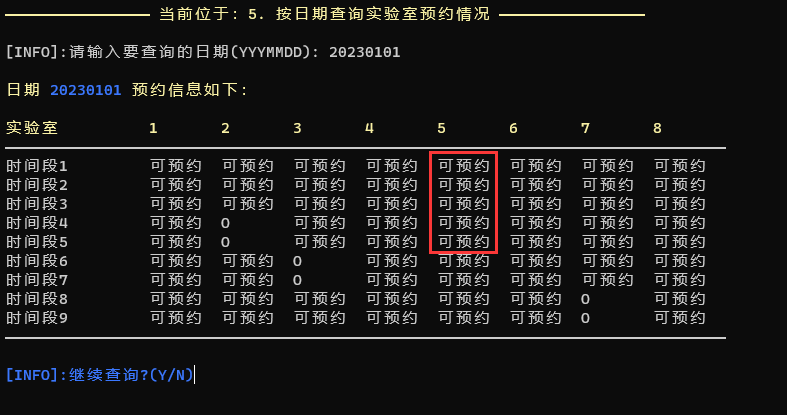


图15 取消预约后20230101日实验室预约情况

步骤五：为了模拟用户取消预约失败的场景。假设用户想要取消一个不存在的预约：20230103日时间段1-9的预约。用户进入取消预约界面，根据系统提示输入“20230103 1 9”，然后按回车确定，系统给出错误提示“取消预约失败，未找到匹配的预约信息!”

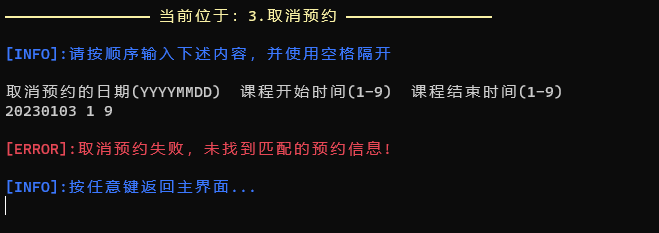


图16 取消预约失败

3. 用户导出实验室课表。

步骤一：用户打开程序并进入主界面，在确认完本周预约信息无误后，用户选择 7.导出实验室课表到文件，进入导出界面。如图17所示。

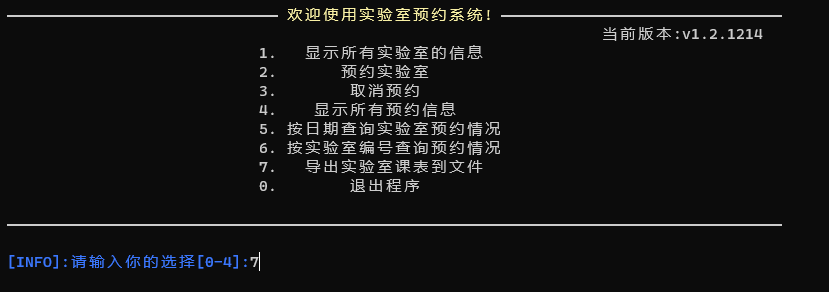


图17 功能选择界面

步骤二：进入导出界面后，系统提示“已成功输出课表到output.txt中”，如下图18。此时用户按任意键可以返回到主界面，并且程序会在当前文件夹目录下生成一个output.txt的文件，即导出的课表。

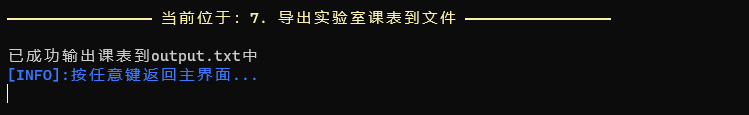


图18 导出实验室课表到文件

步骤三：用户打开output.txt阅读程序生成的课表。如图19所示为部分课表。

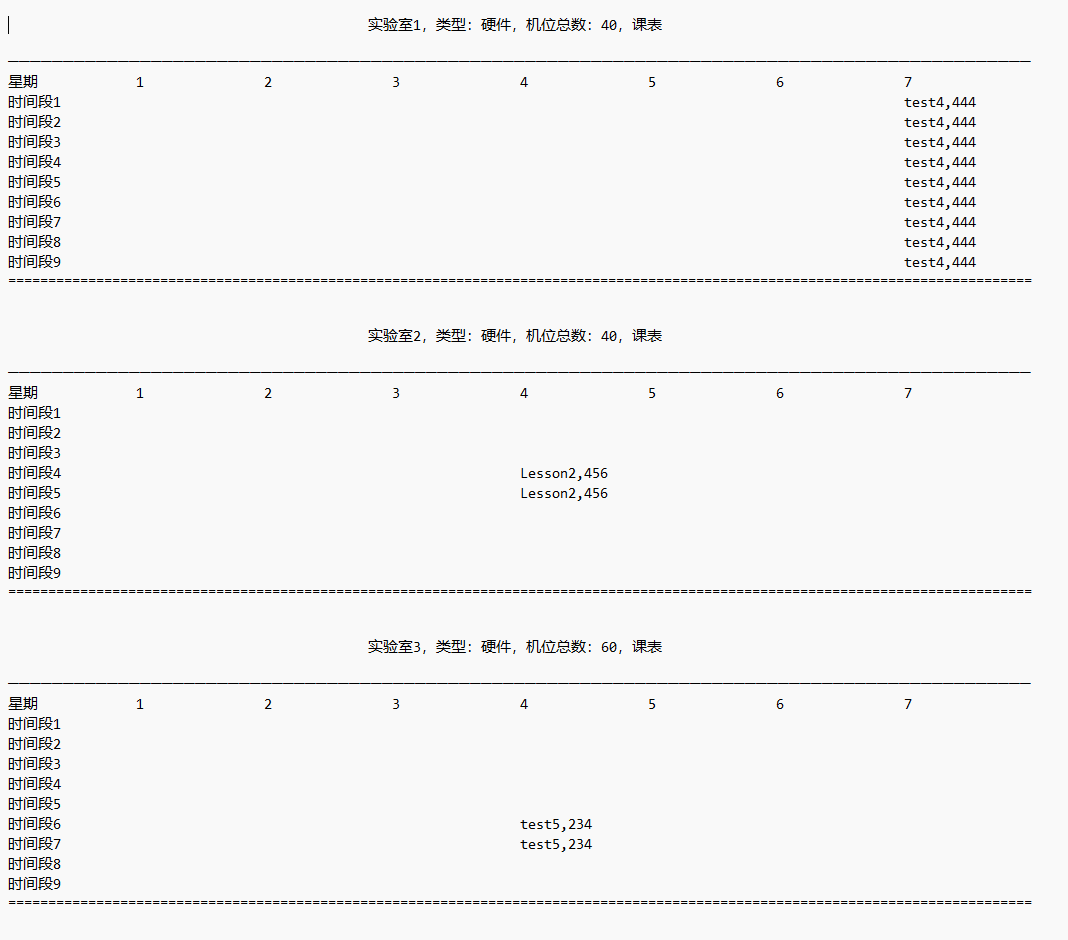


图19 output.txt文件中的内容（部分）

结果分析：系统读取appointment.txt中的预约信息（appointment.txt中的信息如图20所示），输出每个实验室对应的课表。其中，有课的时间段将展示出课程名称和班级，没课的时间段留空白。

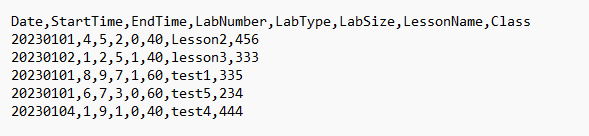


图20 appointment.txt中的内容

**六、调试过程中的问题**

（1）在进行实验室预约输入是否合法时发现，当输入合法时，无论怎么输入，系统都只会出现“预约成功”；问题原因是我们只设计了输入的逻辑判断，并未设计判断输入是否为有效输入；解决方法为新增一个自定义函数makeReservationValuable用于判断是否为有效输入：

bool makeReservationValuable(const Lab labs[], int labCount, const Reservation& reservation)

{

// 读取 "appointment.txt" 文件

ifstream inputFile("appointment.txt");

string line;

while (getline(inputFile, line))

{

stringstream ss(line);

char comma;

// 逐个读取CSV格式的值

int date, startTime, endTime, labNumber, labType, labSize;

string lessonName, studentClass;

ss >> date >> comma >> startTime >> comma >> endTime >> comma >> labNumber >> comma >> labType >> comma >> labSize >> comma >> lessonName >> comma >> studentClass;

// 检查实验室、日期和时间是否与现有的预约冲突

if (labType == static\_cast<int>(reservation.labType) &&

date == reservation.date &&

startTime <= reservation.endTime &&

endTime >= reservation.startTime)

{

cerr << ERROR\_RED << "[ERROR]:预约时间冲突，无法完成预约" << endl;

inputFile.close();

return false;

}

}

inputFile.close();

// 如果所有检查都通过，预约就是有效的

return true;

}

（2）在对新增的预约输入有效判断功能的测试时，我们发现无论如何输入预约信息，得到的依旧是“预约成功”，原因是在实现判断的代码中，我们采用的判断方式出现了问题，并非为逐一比对，导致次次都为“预约成功”，我们最后修改了对比方式的代码：

if (labType == static\_cast<int>(reservation.labType) &&

date == reservation.date &&

startTime <= reservation.endTime &&

endTime >= reservation.startTime)

通过逐项对比实现了该功能。

（3）当进行整体运行的时候，总会出现未找到文件导致报错；原因是调试的时候同路径下我们未添加同名txt文件；为了方便，不用每次都单独添加文件来防止报错，我们增加了自动创建文件的功能：

if (!inputFile.is\_open()) // 如果文件不存在，则创建文件并提示用户输入

{

ofstream outputFile("input.txt");

outputFile << "LabNumber,LabType,LabSize,StartDate,EndDate" << endl;

cerr << ERROR\_RED << "[ERROR]:文件 input.txt 不存在，已为您创建，请将实验室信息输入该文件中后重试" << RESET << endl;

cout << endl;

cout << INFO\_BLUE << "[INFO]:按任意键关闭程序" << RESET << endl;

cin.get();

return 0;

}

（4）在对程序进行调试的时候，在预约实验室测试环节，我们经常会出现输入输错的情况，但因为是逐个数据输入，导致输错的内容无法更改；因此我们更改了输入的代码，使得输入的容错更高，用户体验更好。并且，由于用户对实验室容量不知，我们并没有限制用户输入实验室容量的具体大小，仅要求用户输入所需机器数量，系统通过if语句，根据输入的机器数量自动分配合理容量的实验室：

int labTypeInput;

cout << INFO\_BLUE << "[INFO]:请按顺序输入下述内容，并使用空格隔开" << RESET << endl;

cout << endl;

cout << "课程名称 班级 预约日期(YYYYMMDD) 开始时间(1-9) 结束时间(1-9) 机器数量 实验室类型(0: 硬件, 1: 软件)" << endl;

// 获取用户输入的信息

cin >> reservation.courseName >> reservation.studentClass >> reservation.date >> reservation.startTime >> reservation.endTime >> reservation.requiredMachines >> labTypeInput;

// 检查用户输入的机器数量是否合理并按需求赋值(40,60,80)

if (reservation.requiredMachines > 0 && reservation.requiredMachines <= 40)

{

reservation.requiredMachines = 40;

}

if (reservation.requiredMachines > 40 && reservation.requiredMachines <= 60)

{

reservation.requiredMachines = 60;

}

if (reservation.requiredMachines > 60 && reservation.requiredMachines <= 80)

{

reservation.requiredMachines = 80;

}

if (reservation.requiredMachines <= 0 && reservation.requiredMachines >= 80)

{

cerr << ERROR\_RED << "[ERROR]:机器数量输入不合理，请输入(0-80)" << endl;

return;

}

（5）在对程序进行调试的时候，我们难免会出现输入测试数据错误的情况，纵使有上方优化过的输入代码，也无法更改我们已经输入保存的信息；问题原因是我们没有编写能够删除信息的代码，导致这种情况出现；为此我们新增了新的代码函数cancelAppointment，不仅使得我们能够及时更改测试的原数据，还使得用户能有新的功能（取消预约）可以使用：

void cancelAppointment(Appointment appointments[], int& count)

{

// 输入要取消的预约信息

int cancelDate, cancelStartTime, cancelEndTime;

cout << INFO\_BLUE << "[INFO]:请按顺序输入下述内容，并使用空格隔开" << RESET << endl;

cout << endl;

cout << "取消预约的日期(YYYYMMDD) 课程开始时间(1-9) 课程结束时间(1-9)" << endl;

cin >> cancelDate >> cancelStartTime >> cancelEndTime;

// 使用 remove\_if 移动匹配的元素到末尾

auto it = remove\_if(appointments, appointments + count,

[cancelDate, cancelStartTime, cancelEndTime](const Appointment& app)

{

return app.date == cancelDate && app.startTime == cancelStartTime && app.endTime == cancelEndTime;

});

// 根据 remove\_if 的返回值计算新的有效元素数量

int newCount = distance(appointments, it);

if (newCount < count)

{

cout << endl;

cout << INFO\_BLUE << "[INFO]:取消预约成功！" << RESET << endl;

// 将预约信息写入文件

ofstream outFile("appointment.txt");

if (outFile.is\_open())

{

outFile << "Date,StartTime,EndTime,LabNumber,LabType,LabSize,LessonName,Class" << endl;

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

{

outFile << appointments[i].date << ","

<< appointments[i].startTime << ","

<< appointments[i].endTime << ","

<< appointments[i].labNumber << ","

<< appointments[i].labType << ","

<< appointments[i].labSize << ","

<< appointments[i].lessonName << ","

<< appointments[i].studentClass << endl;

}

outFile.close();

}

count = newCount;

}

else

{

cout << endl;

cerr << ERROR\_RED << "[ERROR]:取消预约失败，未找到匹配的预约信息!" << RESET << endl;

}

}

**七、课程总结**

程序设计是一门既深邃又富有创造性的艺术，通过这一周的实践上机活动，我们积累了一些编写代码的经验，以及我们小组三个人的共同感悟：

首先，深刻理解问题是高效解决问题的关键。在着手编写代码之前，要对问题进行仔细分析和理解。这包括理解用户需求、系统架构和各个组件之间的关系。只有深刻理解问题，才能够提出合理的解决方案，避免不必要的错误和重构，就比如这次的实验室预约程序，我们动手之前先将程序充分理解，然后将其分成了好几个部分，不同部分实现不同功能。调理清晰，这样我们三人合作起来才更快更高效。

其次，注重代码的可读性和可维护性。良好的代码不仅仅是计算机可以理解和执行的，更是我们写程序的人可以阅读和进行维护的。给变量、函数和类选择有意义的命名，编写清晰的注释，采用一致的代码风格，都是我们提高代码可读性的有效手段。这有助于我们三人团队协作，降低后续维护的难度。这次我们在进行程序设计的时候统一采用了C++语言编写，对于一些虽然在C++环境下也能实现的C语言代码进行了修改，使其更加的符合整体的一个语言逻辑。

第三，善于利用工具和框架。现代程序设计离不开各种工具和框架，它们能够提高开发效率、降低错误率。熟练掌握版本控制工具，使用调试工具和性能分析工具，能够让程序开发过程更加高效和顺畅。

第四，保持学习的心态。程序设计是一个不断演进的领域，新的语言和技术不断涌现。要时刻保持学习的心态，不断提升自己的技术水平。积极参与开源项目、阅读优秀的代码和技术博客，都是我们提高编程技能的有效途径。

此外，及时的代码测试是确保代码质量的有效手段。测试不仅能够及早发现和修复bug，还能够确保代码在修改后不会对系统的其他部分产生不良影响。同时，测试代码不能等到程序全部写完后再进行统一测试，而是应该每写一个模块就测试一遍，这样的测试更加高效，并且更加容易找到问题所在。

最后，善于沟通和合作。在实际项目中，很少有一个人独自完成整个系统的开发。良好的沟通和合作是团队取得成功的关键。能够清晰地表达自己的想法，理解他人的观点，协调团队成员的工作，都是成为优秀程序员的重要素质。我们三人在编写过程中都有及时的交流，这样才使得我们代码在合并的时候没有出现大规模的报错，有的也只是一些很快就能修改过来的小bug。

总结：程序设计是一门需要我们时时保持学习和不断去实践的技能，我们不仅要深刻理解问题、提高并保持我们写的代码质量，还要善于利用科学工具、保持自身学习的心态，在多人合作的时候还要及时进行编写测试，同时不要忘了和合作的朋友保持良好的沟通协调，这样我们才可以更好地应对复杂的软件开发任务，提高工作效率和代码质量。也希望未来的我们在编写代码的时候也能拥有保持这些良好素质。