计算机编程实践

日程管理软件
设计文档

2023.7.6

钟泓逸 522031910522

全诗琪 522031910573

谢宇开 522031910626

龚笑旸 522031910776

目录

一、 小组分工 2
二、 模块、类与设计思路 2
2.1 Schedule.h 2
2.1.1 ScheduleList 类 3
2.1.2 时间处理函数 4
2.1.3 引用库与被引用库 4
2.2 User.h 5
2.2.1 user 类 5
2.2.2 引用库与被引用库 5
2.3 scheduleManage.h 5
2.3.1 scheduleManage 类 5
2.3.2 格式判断函数 6
2.3.3 引用库和被引用库 6
2.4 login.h 6
2.4.1 login 类 6
2.4.2 引用库与被引用库 7
2.5 命令行交互（main 函数） 7
2.5.1 包括函数 7
三、 关键技术说明 7
3.1 数据类型的选取 7
3.1.1 底层数据容器的选取 7
3.1.2 时间数据类型的选取 8
3.2 业务逻辑类 9
3.2.1 业务逻辑类的作用 9
3.2.2 业务逻辑实现示例（以 createSchedule（）部分代码为例） 9
3.2.3 缓冲区的清除 10
3.3 文件的读取与保存 10
3.3.1 文件打开方式选择 10
3.3.2 fscanf 和 fprintf 10
3.4 密码的加密 11
3.5 命令行参数的判断 11
3.6 多线程与提醒函数 11
3.6.1 多线程设计（thread） 11
3.6.2 程序加锁协调（mutex） 12
3.6.3 提醒函数的阻塞与过期日程提醒 13
3.7 （图形化系统）通过 id 修改日程 14
四、 附录：程序流程图 15

一、小组分工

表 1.1 小组任务与分工表

Schedule.h/scheduleManage.h	钟泓逸、全诗琪
User.h/login.h	钟泓逸、全诗琪、谢宇开、龚笑旸
main	钟泓逸
图形化	钟泓逸
技术文档	钟泓逸、全诗琪、谢宇开
Shell 编写、程序调试	谢宇开、龚笑旸

二、模块、类与设计思路

共有五个头文件和命令行交互模块（main.cpp 文件）。其中 schedule.h 和 user.h 用于处理最底层数据的添加/删除/查找/遍历和文件数据的处理；scheduleManage.h 和 login.h 用于处理业务逻辑，包括参数传递的处理和处理数据逻辑的判断，实现了基本功能；md5.h 为加密算法的实现；main 函数用于处理命令行参数，并且组织各功能函数之间的关系。

四个自定义头文件又分为两个模块。schedule.h 和 scheduleManage.h 为日程管理模块，user.h 和 login.h 为用户登录模块

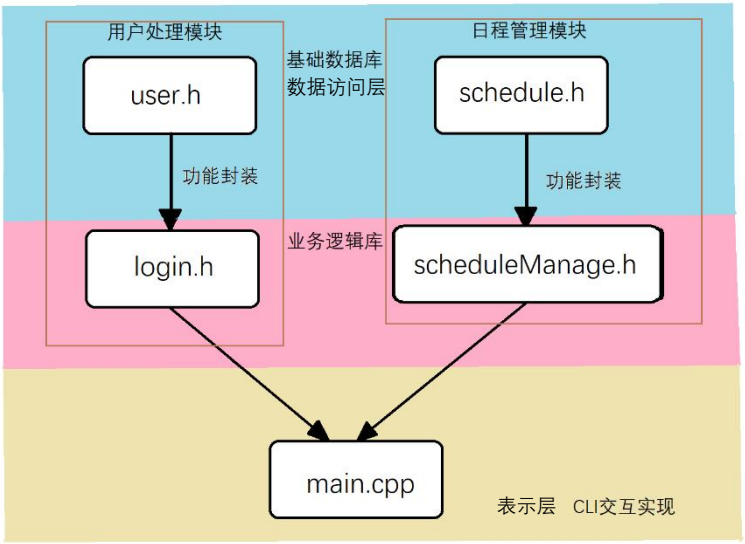

图 2.1 类模块的关系图

注：在下文函数表中，加粗函数为重要函数。

2.1 Schedule.h

用 schedule 结构体实现单个日程的储存。其中时间结构用 tm 结构体储存。

考虑到任务日程的唯一性和排序的需要，指定用户的日程用 map 容器储存，以 startTime 开始时间为 KEY 值，构造任务列表

同时，为了方便提醒函数获取最早的提醒时间，用以 alertTime 提醒时间为 KEY 值的 map 容器储存，构造提醒列表。

最后，为了分配唯一 id，用任务数量（只增不减）来分配 id。

2.1.1 ScheduleList 类

私有成员：

```
struct schedule
{
    char taskname[31]; // 储存的taskname中不能含有空格
    struct tm *alertTime;
    struct tm *startTime;
    int priority; // 用数字1-3描述优先级，3为最高，1为最低
    int id;
    char classification[22];

    schedule(char *name, int y1, int m1, int d1, int h1, int mi1, int y2, int m2,
    int d2, int h2, int mi2, int number, int prio, char *cla) : priority(prio)
    {
        strcpy(taskname, name);
        strcpy(classification, cla);
        alertTime = new tm;
        startTime = new tm;
        getTime(alertTime, y1, m1, d1, h1, mi1);
        getTime(startTime, y2, m2, d2, h2, mi2);
        id = number;
    }
```

图 2.2 schedule 结构体的定义

表 2.1 scheduleList 类成员表

名称	类型	作用
username[22]	char 数组	储存用户信息
scheduleFile[40]	char 数组	储存文件名称
taskList	map<long int, schedule>	任务列表； 储存的 pair 内容：<开始时间，日程>
alertList	map<long int, int>	提醒时间列表 储存的 pair 内容： <提醒时间，提醒时间对应任务数量>
taskNumber	int	对应的任务数量

类函数

表 2.2 scheduleList 类函数表

函数名	返回参数	作用
ScheduList	/	构造函数
日程列表处理函数		
addTask	void	增加日程
deleteTask	void	根据 id 删除日程以及对应提醒时间
findTaskById	long int	根据 id 查找，返回值为 id 对应任务 longint 类型
findTaskByStartTime	void	根据开始时间查找，并且打印查找到的日程
findTaskByName	void	通过名字查找，并且打印查找到的日程
findTaskByAlertTime	void	通过提醒时间查找
findTaskByPriority	void	根据优先级查找，并打印查到日程
findTaskByClass	void	根据分类查找，并打印查到日程
displayTask	void	遍历输出指定日期 min-max 之间的日程 （以开始时间升序）
isUniqueTime	bool	是否存在开始时间为 time 的日程，存在返回 false

提醒列表处理函数		
nextAlert	long int	返回下一个提醒时间
（接上表）		
displayAlert	void	输出提醒时间列表及对应日程数量
deleteAlertByAT	void	删除提醒时间
deleteAlert	void	按开始时间删除提醒时间
文件处理		
saveFile	void	保存文件
loadFile	void	加载文件,flag=true 时打印加载信息

2.1.2 时间处理函数

在 schedule.h 中，使用了**三种时间结构储存**格式：

1、普通 yyyy-mm-dd hh-mm 格式，即用**五个 int 类型变量**储存。

2、**Tm 结构体**格式。（time.h 中定义格式）

3、**Long int** 格式（自定义格式），长整型数字格式为 yyymmddhhmm（10-11 位整数）。其中所有时间都以 tm 结构体中储存时间相同。以下图为例。

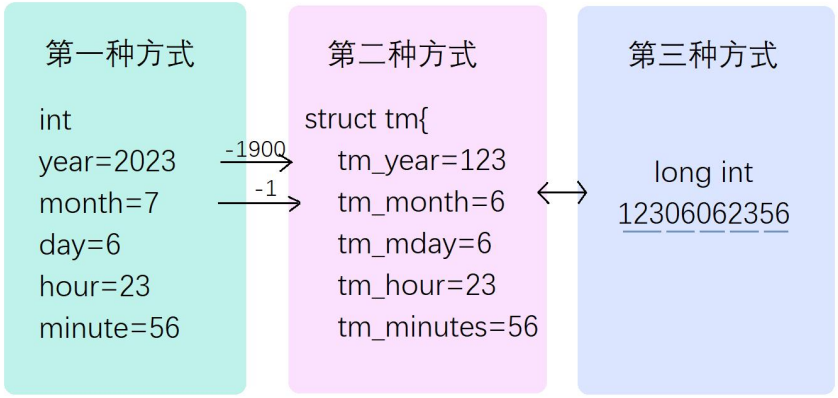

图 2.3 三种时间结构的转换图示（以 2023.7.6 23:56 为例）

表 2.3 schedule.h 类时间相关函数表

函数名	返回参数	作用
getTime	void	yyyy-mm-dd-hh-mm 格式转 tm 结构体
getLongIntTime	long int	Yyyy-mm-dd hh-mm 格式转为 long int
getLongIntTime	long int	tm 结构体转为 long int
getTmTime	tm 结构体	long int 转为 tm 结构体
compareTime	Bool	比较两个 tm 结构体时间大小
printTime	void	打印 tm 结构体储存的时间信息

2.1.3 引用库与被引用库

引用库：iostream , time.h , cstring , map , utility , iomanip

被引用至库：scheduleManage.h

2.2 User.h

用 userInfo 结构体储存单个用户数据，用 vector 容器储存所有用户列表。User.h 中所有储存密码皆为加密后密码。

2.2.1 user 类

私有成员

```
struct userInfo{
    char username[22];
    char password[40];
    userInfo():username(""),password(""){};
};
```

图 2.4 userInfo 结构体的定义

表 2.4 user 类成员表

名称	类型	作用
userList	vector<userInfo>	储存用户数据表（userInfo 结构体）

类函数

表 2.5 user 类函数表

函数名	返回参数	作用
user	/	构造函数
addUser	void	添加用户
deleteUser	bool	删除用户数据
checkPassword	bool	检查用户名密码是否正确，用户名不存在/密码错误返回 false
checkUser	bool	检查用户是否存在
displayUserList	void	遍历输出用户列表
modifyPassword	void	修改密码
findUser	int	返回 user 的迭代器位置
saveUserFile	void	保存用户数据文件

2.2.2 引用库与被引用库

引用库：iostream ， vector , cstring , iomanip , md5.h
被引用至库：login.h

2.3 scheduleManage.h

封装并组合 schedule.h，并且实现参数格式、参数长度、日期格式、日期正确性、日期逻辑等判断。

在安全方面，实现数组越界等的判断。

2.3.1 scheduleManage 类

私有成员

表 2.6 scheduleManage 类成员表

名称	类型	作用
s_list	scheduleList 类	封装储存日程信息表

类函数

表 2.7 scheduleManage 类函数表

函数名	返回参数	作用
scheduleManage	/	构造函数
m_addTask	void	（命令行模式）添加日程
m_display	void	（命令行模式）遍历输出 min-max 之间的日程
deleteSchedule	void	（命令行/run）删除日程
displaySchedule	void	（命令行模式）遍历输出指定时间段日程
display	void	（run 模式）遍历输出日程（按时间、分类、名称）
createSchedule	void	（run 模式）添加日程
menu	void	（run 模式）添加/删除/查找日程的循环进行
alert	void	（run 模式）提醒函数

2.3.2 格式判断函数

为了实现格式判断和逻辑判断，并且减少代码重用，设计了判断合理日期和判断字符串是否为数字的函数。

表 2.8 格式判断函数表

函数名	返回参数	作用
isCorreTime	bool	判断日期是否合理（年需≥1900）
isInteger	bool	输入字符串是否都是数字

2.3.3 引用库和被引用库

引用库：schedule.h , unistd.h , thread , mutex
被引用库：main.cpp

2.4 login.h

封装并组合 user.h，并且实现参数格式、参数长度等判断。
同时，密码加密过程由 login 类中的函数完成。
在安全方面，实现数组越界等的判断。

2.4.1 login 类

私有成员

表 2.9 login 类成员表

名称	类型	作用
list	user 类	封装储存用户信息表

类函数

表 2.9 login 类函数表

函数名	返回参数	作用
login	/	构造函数
m_login	bool	（命令行模式）判断用户是否登录/创建成功
m_deleteUser	void	（命令行模式）删除用户
m_modifyPassword	void	（命令行模式）修改密码
loginUser	char*	（run 模式）判断用户是否登录/创建成功，若成功返回用户名 char 指针，失败返回 NULL
loginModify	void	（run 模式）修改密码
loginDelete	void	（run 模式）删除日程

2.4.2 引用库与被引用库

引用库：user.h
被引用库：main.cpp

2.5 命令行交互（main 函数）

在 main 函数中实现：

1、 用户管理模块参数的判断，登录或命令读取对相应函数的调用，错误参数的判断与提示；

2、 日程管理模块参数的判断，读取命令后对相应函数的调用，错误参数的判断与提示；

3、 run 模式中各模块内关系、模块间关系的实现和错误提示；

2.5.1 包括函数

表 2.9 main.cpp 函数表

run	void	进入 run 模式，实现登录模块和日程模块的块内逻辑和块间逻辑
getHelp	void	获取函数帮助列表或指定函数的详细用法

注：getHelp 有两个重载函数，其中 getHelp 获取函数帮助列表，getHelp 函数 获取该函数的具体用法和注意事项。

三、关键技术说明

3.1 数据类型的选取

3.1.1 底层数据容器的选取

1. 选择 STL：STL 可以省去储存结构本身的析构，比较自身设计实现的数据结构，内存泄漏的风险更少
2. 对于用户列表而言，添加、查找操作较多，删除、排序操作较少；且在用户列表储存的时候，不用考虑成员顺序，则选择 vector 容器
3. 对于日程列表来说，删除、排序操作较多，且有开始时间唯一的需求，vector 作为数组/线性表类容器则不再适用，而 map 作为红黑树类容器，KEY 值唯一，更好地契合需求，则使用 map 容器

3.1.2 时间数据类型的选取

1. 在 schedule 结构体中储存 tm 结构体：tm 结构体作为系统库定义结构体，整个体系更加完善，且 tm 结构体中有 tm_wday，配合 mktime 函数可以直接获取星期数据，更加方便。

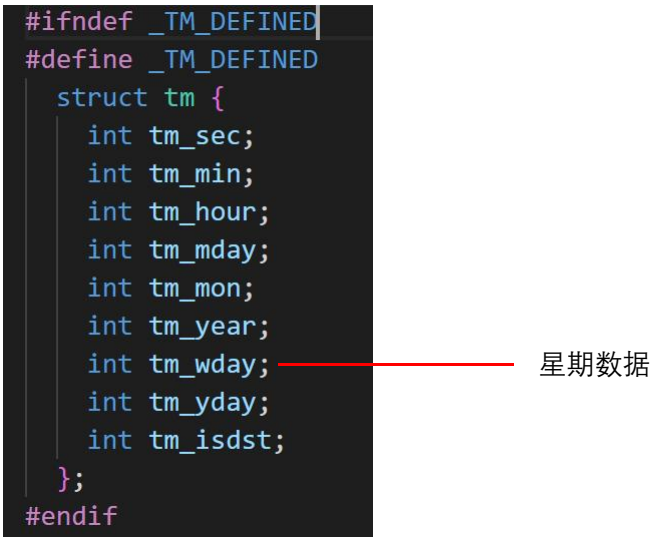

图 3.1 tm 结构体的定义及其成员

2. 在用户输入的时候，考虑到用户的输入便捷性和习惯，选择用五个 int 类型数据，即第一种时间储存方式，普通方式，来储存读取时间，后续通过时间类型转换函数再传入不同的下层函数。

```
cout << "请输入日程信息,输入格式：[任务名称] [提醒时间] [开始时间]，时间格式为:yyyy-mm-dd hh:mm,如果需要退出请输入quit" << endl;
char *taskname = new char[100];
char *priority = new char();
char *classification = new char[100];
char quit[5] = "quit";
char classfi[22] = "默认";
int y1, m1, d1, h1, mi1, y2, m2, d2, h2, mi2;
```

图 3.2 用户输入时时间结构的读取与储存（以 createSchedule（）函数为例）

```
aT = getLongIntTime(y1, m1, d1, h1, mi1);
sT = getLongIntTime(y2, m2, d2, h2, mi2);
time_t preTime;
struct tm *presentTime;
```

图 3.3 实际处理时时间的转换与储存（以 createSchedule（）函数为例）

3. 在 taskList 和 alertList 中，选取 long int 类型作为 map 容器的 KEY 值。
   a) 若将 tm 结构体作为 KEY 值，首先，由于 map 本身没有 tm 结构体的比较函数，则需要设计 tm 结构体的函数，并且重定义‘<’运算符，其次，由图 3.1 可见，tm 结构体本身储存占据空间较大，若用 tm 结构体作为 KEY，则会使空间成

本大大增加。

b) 若将 time_t 类型作为 KEY 值，虽然 time_t 同为 int（time_t 类型储存 1970 年到当前时间的秒数），且与 long int 一样，皆为 10-11 位整数。但是从下图可以看出，time_t 不如 long int 类直观，则 long int 更加适合调试。

long int类型	12400010000
time_t类型	20056896000

图 3.4 long int 类型和 time_t 类型时间比较（以 2024.1.1 0:0 为例）

3.2 业务逻辑类

3.2.1 业务逻辑类的作用

1. 单独封装对数据本身的处理操作，使得各种参数判断不会涉及内部数据，增加数据储存的安全性，提高程序健壮性。
2. 使得各种功能更加易于控制和延展，便于后期的维护

3.2.2 业务逻辑实现示例（以 createSchedule（）部分代码为例）

未截图的判断：
1、日程提醒时间大于开始时间的提醒
2、优先级和分类缺省值传参的判断
3、……

```
if (strlen(taskname) > 30)
{
    cout << "任务名称过长，任务名称应为小于等于30个英文字符或小于等于
         << endl;
    while ((clear = getchar()) != '\n')…
    cout << "请重新输入日程信息" << endl;
    continue;
}
pout = scanf("%d-%d-%d %d:%d %d-%d-%d %d:%d", &y1, &m1, &d1, &h1
if (pout != 10)
{
    cout << "日期格式错误" << endl
         << endl;
    while ((clear = getchar()) != '\n')…
    cout << "请重新输入日程信息" << endl;
    continue;
}
if (!isCorrectTime(y2, m2, d2, h2, mi2))
{
    cout << "开始时间不存在" << endl
         << endl;
    while ((clear = getchar()) != '\n')…
    cout << "请重新输入日程信息" << endl;
    continue;
}
if (!s_list.isUniqueTime(sT))
{
    cout << "已存在相同开始时间日程，日程添加失败" << endl;
    while ((clear = getchar()) != '\n')…
    continue;
}
if (sT < pT)
{
    cout << "日程开始时间小于当前时间，是否确定添加日程，输入y/n: ";
    cin >> Flag;
```

```
s_list.addTask(taskname, y1, m1, d1, h1, mi1, y2, m2, d2, h2, mi2, prio, classfi);
s_list.saveFile();
```

任务名称长度判断（防止底层长度为 31 的 char 数组越界）

日期格式判断，保证 scanf 读取到所有日期数据

日期合理性判断，转换到 tm 结构体时不会出错

相同开始时间判断，以防 map 容器无法加入该日程，并提醒用户

日程开始时间小于当前时间提醒

当所有参数检查无误时，调用底层函数

图 3.5 业务逻辑实现示例

注：遇到会破坏程序储存结构/导致程序运行的错误参数，程序会直接拒绝参数的向下

传递；遇到逻辑上有错误的参数，程序会询问用户意见之后再决定是否拒绝参数向下传递。

3.2.3 缓冲区的清除

由于程序中多处用到读取缓冲区输入数据的函数，所以需要及时清除用户输入的多余代码/残留在缓冲区的错误代码，防止这些代码被之后的函数读取。

```
while ((clear = getchar()) != '\n')
{
}
```

图 3.6 缓冲区数据的清除（直至换行符）

3.3 文件的读取与保存

3.3.1 文件打开方式选择

1. loadFile 时，由于只需要读取文件数据，选择以“r”方式打开。若以“r”方式打开失败（文件不存在），则先以“w”方式打开来创建文件，再以“r”方式打开。
2. saveFile 时，由于需要写入新数据，而新数据在 vector/map 容器中有完整储存，则选择以“w+”方式打开。

3.3.2 fscanf 和 fprintf

在储存类/结构体数据时，由于数据比较整齐，则考虑格式化输入/输出；又由于 fscanf 有按行读取（读到换行符停止档次读取）的特点，则考虑按行储存/读取数据。以 user.h 为例。

```
void user::saveUserFile()
{
    FILE *fp;
    char filename[45] = "./file/user";
    fp = fopen(filename, "w+");
    if (fp == NULL)…
    for (int i = 0; i < userList.size(); i++)
    {
        fprintf(fp, "%s %s\n", userList[i].username, userList[i].password);
    }
    fclose(fp);
    return;
}
```

图 3.7 user 类数据储存

```
    while (!feof(fp))
    {
        int pout = fscanf(fp, "%s %s", tmp.username, tmp.password);
        if (pout == 2)
        {
            userList.push_back(tmp);
        }
    }
    fclose(fp);
}
```

图 3.8 user 类构造函数中数据读取

由于 fprintf 时按行输出，导致最后一行也会输出‘\n’，这就使得在 fscanf 读取完最后一

行数据时，读取指针仍在换行符之前，feof（fp）返回为 false，则有可能造成最后一行数据的重复储存（fscanf 读取错误，不会进行赋值操作，tmp 中储存数据仍然为最后一行数据）。此时，用 fscanf 的返回值（fscanf 函数正确读取的参数个数）和 feof（fp）进行双重判断，则可保证最后一个换行符不会对数据读取造成影响。

3.4 密码的加密

我们保存某些密码信息以用于身份确认时，如果直接将密码信息以明码方式保存在数据库中，不使用任何保密措施，他人就很容易能得到原来的密码信息，这些信息一旦泄露， 密码也很容易被破译。为了增加安全性，有必要对数据库中需要保密的信息进行加密，这样，即使有人得到了整个数据库，如果没有解密算法，也不能得到原来的密码信息。

MD5 算法可以很好地解决这个问题，因为它可以将任意长度的输入串经过计算得到固定长度的输出，而且绝大多数情况下只有在明文相同时，才能等到相同的密文，并且 MD5 算法是不可逆的，即便得到了加密以后的密文，也不可能通过解密算法反算出明文。这样就可以把用户的密码以 MD5 值的方式保存起来，用户登录时，系统是把用户输入的密码计算成 MD5 值，然后再去和系统中保存的 MD5 值进行比较，如果密文相同，就可以认定密码是正确的，否则密码错误。通过这样的步骤，系统在并不知道用户密码明码的情况下就可以确定用户登录系统的合法性。这样不但可以避免用户的密码被具有系统管理员权限的用户知道，而且还在一定程度上增加了密码被破解的难度 。

MD5 信息摘要算法的原理可简要的叙述为：MD5 码以 512 位分组来处理输入的信息，且每一分组又被划分为 16 个 32 位子分组，经过了一系列的处理后，算法的输出由四个 32 位分组组成，将这四个 32 位分组级联后将生成一个 128 位散列值。

3.5 命令行参数的判断

1. 命令参数个数的判断：通过 argc 的比较实现

```
if (argc != (commandStart + 8))
```

图 3.9.1 argc 判断参数个数

2. 命令参数格式的判断：通过 isInterger 函数的返回值判断

```
if (!isInterger(argv[i]))
```

图 3.9.2 isInterger 判断参数是否为整数

3. 命令是否匹配的判断：通过 strcasecmp 比较目标命令与实际获取命令实现

```
if (!strcasecmp(userCommand, "deleteUser"))
```

图 3.9.3 strcasecmp 判断字符串一致性

3.6 多线程与提醒函数

3.6.1 多线程设计（thread）

在 run 模式中实现多线程。其中 menu 函数，即正常的添加/删除/查找显示日程操作为主线程，alert 函数，即日程提醒为子线程。其中日程提醒的不断循环体现在 alert 函数内部，且子线程以 detach 方式加入主线程，使主线程能在不被子线程阻塞的情况下一直运行。而在主线程结束时，子线程会随之一起结束。

```
scheduleManage list(user);

thread first(&scheduleManage::alert, &list);

first.detach();
list.menu();
```

图 3.10 子线程的创建及以 detach 方式加入

3.6.2 程序加锁协调（mutex）

由于 alert 和 menu 在运行时需要调用相同函数/修改相同数据成员，且防止 alert 提醒信息干扰用户已经开始的 addtask 等操作，因此考虑安全性和用户需求，需要给两个函数加上互斥锁 mutex。

```
mutex mtx;
```

图 3.11 mutex 的定义（全局变量）

```
void scheduleManage::alert()
{
    bool flag = true;
    bool isAlert=false;
    while (true)
    {
        mtx.lock();
        isAlert=false;
        time_t preTime;
        time(&preTime);
        struct tm presentTime;
        localtime_r(&preTime, &presentTime);
        long int pT, aT = 0;
        pT = getLongIntTime(&presentTime);
        aT = s_list.nextAlert();
        while ((aT < pT)&& (flag) )
        {
            if(aT==0){break;}
            s_list.deleteAlertByAT(aT);
            aT = s_list.nextAlert();
        }
        flag = false;
        while ((aT < pT) && (!flag))
        {
            cout << "过期日程提醒：  " << endl;
            s_list.findTaskByAlertTime(aT);
            s_list.deleteAlertByAT(aT);
            aT = s_list.nextAlert();
        }

        if (pT == aT)
        {
            cout << "有当前日程提醒：  ";
            isAlert=true;
            s_list.findTaskByAlertTime(aT);
            s_list.deleteAlertByAT(aT);
            aT = s_list.nextAlert();
        }
        mtx.unlock();
        if(isAlert){
            sleep(60);
        }
        else{
            sleep(5);
        }

    }
}
```

```
while (flag)
{
    cin >> command;
    isCorrectCommand = false;
    if (!strcasecmp(command, "quit"))
    {
        isCorrectCommand = true;
        flag = false;
    }
    if (!strcasecmp(command, "addTask"))
    {
        isCorrectCommand = true;
        mtx.lock();
        createSchedule();
        mtx.unlock();
    }
    if (!strcasecmp(command, "deleteTask"))
    {
        isCorrectCommand = true;
        mtx.lock();
        deleteSchedule();
        mtx.unlock();
    }
    if (!strcasecmp(command, "displayTask"))
    {
        isCorrectCommand = true;
        mtx.lock();
        displaySchedule();
        mtx.unlock();
    }
```

图 3.12 mutex 的 lock 与 unlock（左图为 alert，右图为 menu 的部分截图）

由于 mutex 会阻塞其他线程，因此在实现功能的情况下，mutex 上锁时间应该尽量少。对此我们在程序中做出以下设计：

（1）在 alert 循环时，在 mtx 的 unlock 和下一次 lock 之间，若本次没有日程提醒，则暂停 5 秒，使得程序能够在当前时间和提醒时间相同时，尽快地打印日程提醒，并且能保证不会一直占用锁；若本次有日程提醒，则暂停 60s，即打印日程提醒后直接跳过该分钟，防止出现重复打印的情况。

（2）在 menu 函数中，只有命令比对成功，即进入创建日程等流程（createSchedule 函数等）时，才会真正给 mtx 上锁，若只是简单的参数读取/判断，不涉及下一层数据的处理的情况下，不会对 mtx 进行上锁操作

（3）由于日程的提醒打印在命令行中，因此用户添加日程或者其他操作时，若正在进行输入，可能会收到提醒的干扰，因此在进行日程的操作时，当 menu 接收到正确指令，就会对 mtx 进行加锁，直至指令结束才会进行解锁。

3.6.3 提醒函数的阻塞与过期日程提醒

通过图 3.12 可得，当正确输入 addtask 等指令时，程序会对 mtx 加锁。但是 createSchedule 等函数本身需要用户的输入，若用户输入时间过长，则可能不能及时接收日程提醒。综合考虑用户的输入需求（输入不被打断/干扰）和日程的及时提醒，在程序中设计了过期日程提醒。

在程序打开时，会获取时间并且删除提醒列表中提醒时间早于当前时间的提醒，而在程序运行时，若因为用户输入等操作造成 alert 函数被阻塞，则被阻塞时的提醒日程会被保留，在用户结束输入操作时将会弹出过期日程提醒，打印被阻塞提醒日程信息。

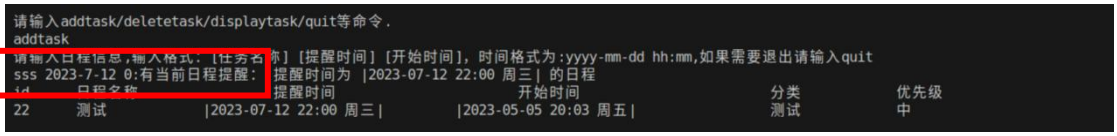

图 3.13 用户输入被阻塞示例

图 3.14 未被阻塞的日程提醒

图 3.15 过期日程提醒（提醒时间 16:32，在 16:31 进行日程添加，16:33 结束操作）

3.7 （图形化系统）通过 id 修改日程

图 3.16 日程修改界面

在图形化系统中，用户修改日程时只需要输入 id，随后当鼠标光标失去焦点或者回车后，如果日程存在，会自动出现对应的日程信息。这一功能由 lineEdit 控件的槽函数 editFinished 实现。当系统检测到输入完成，会自动查找 id 是否存在，调取 id 对应日程并显示。若 id 不存在，自动将信息栏显示为空。

四、 附录：程序流程图

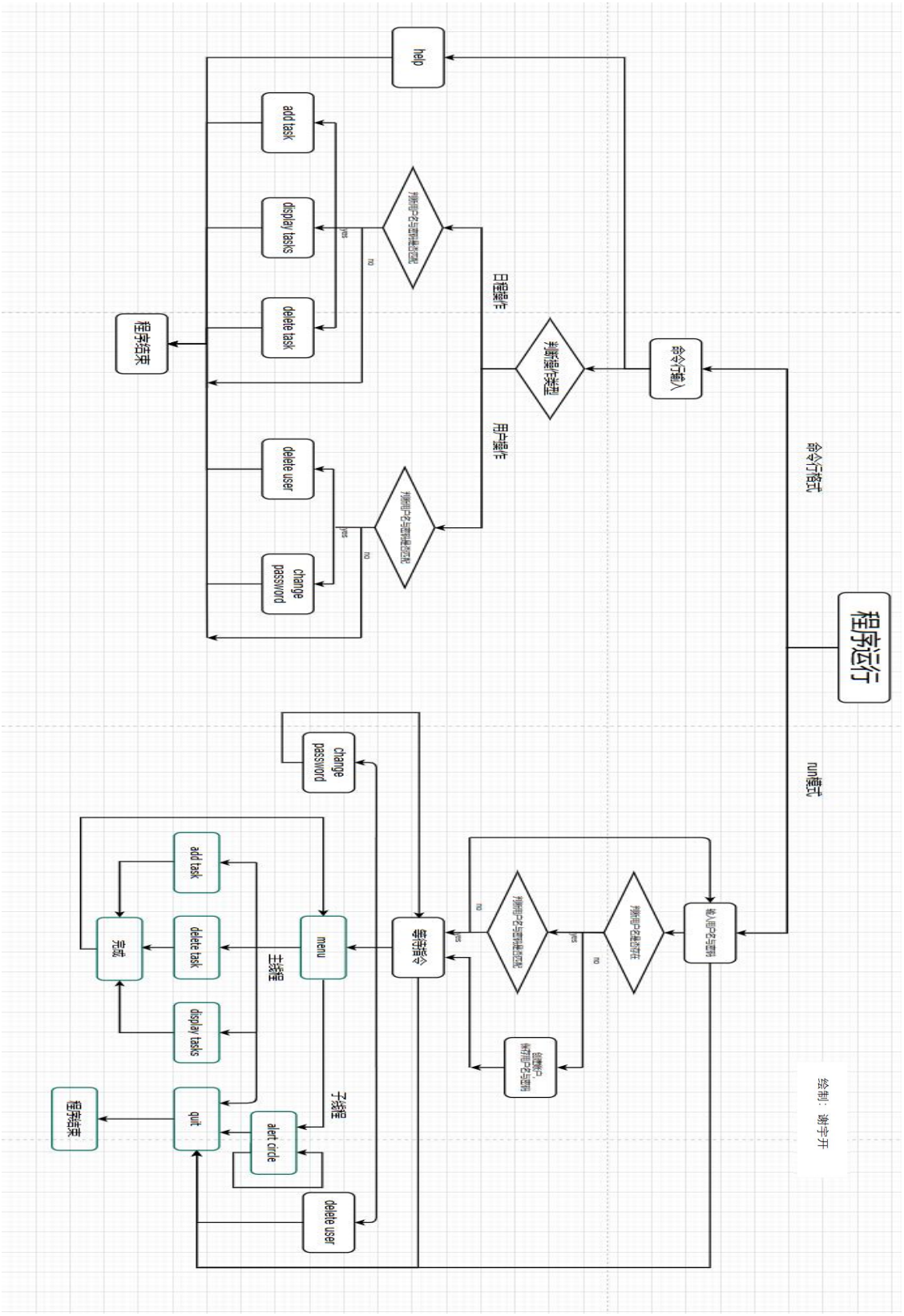

图 4.1 程序流程图