BMI管理系统报告

姓名：郑子睿

学号：22920212204317

专业：计算机科学与技术

编程语言：C/C++，makefile

一、背景调查

1. 引言

在现代社会中，人们对健康和身体状况的关注日益增加。Body Mass Index（BMI）即身体质量指数，是一种常用的衡量人体肥胖程度的方法。BMI不仅可以用于个人健康管理，也广泛应用于医疗领域和公共卫生政策制定中。

BMI是评估个体体重状况的重要指标，具有以下作用： a. 健康风险评估：BMI可以作为判断个体患病风险的指标之一。高BMI与心血管疾病、糖尿病、高血压等健康问题的发生率增加相关联。 b. 肥胖管理：BMI可以帮助个体了解自己的体重状况，从而采取相应的措施进行肥胖管理。例如，如果BMI超过正常范围，个体可能需要调整饮食和增加运动量。 c. 公共卫生政策：BMI数据的收集和分析可以帮助制定公共卫生政策，如肥胖防控措施和营养教育活动等。

2. BMI定义和计算方法

BMI是根据人的身高和体重计算得出的一个数值，用于评估个体的体重状况。它是一个简单而有效的工具，可以帮助人们了解自己的体重是否在健康范围内。

BMI的计算方法非常简单，公式如下： BMI = 体重（公斤）/（身高（米）× 身高（米））

3. 本系统采用的BMI标准

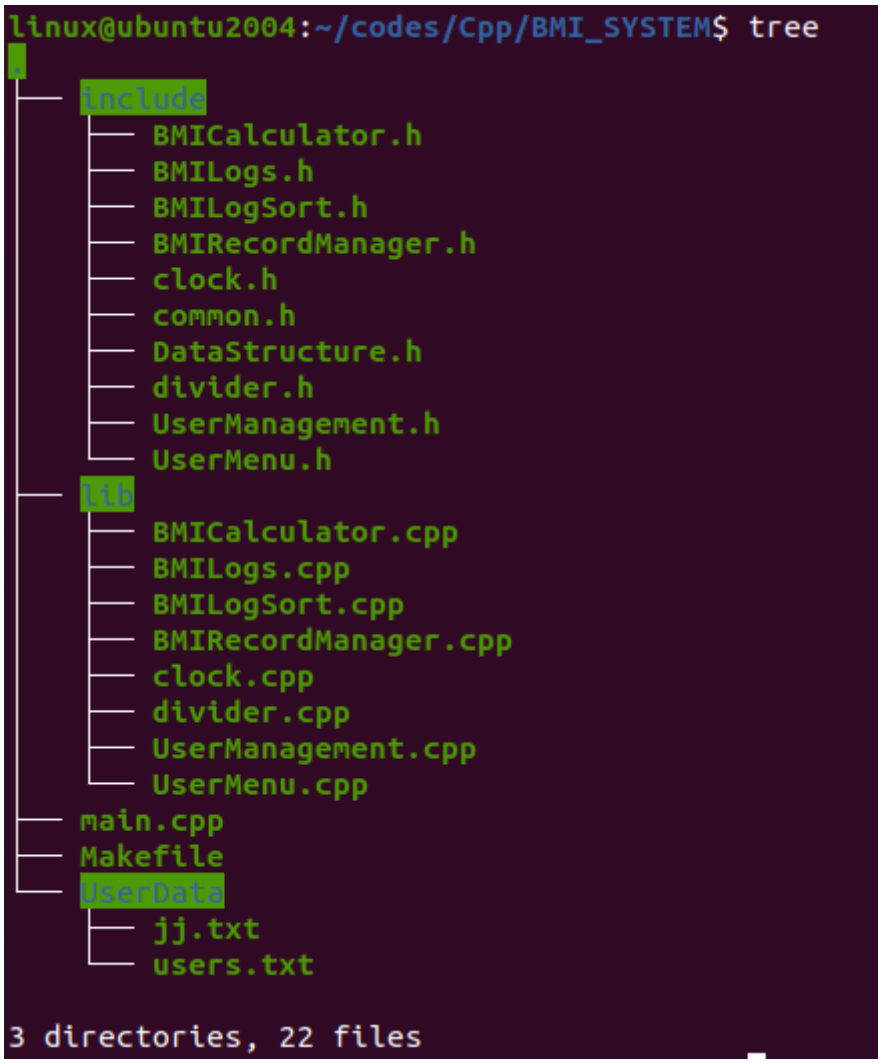
体重变化是判断一段时期内能量平衡与否最简便易行的指标，也是判断吃动是否平衡的指标。目前常用的判断健康体重的指标是体质指数（body mass index，BMI），它的计算方法是用体重（kg）除以身高（m）的平方。

状态	BMI (kg/m2)	
	最低	最高
体重过轻 （严重消瘦）		15
体重过轻 （中度消瘦）	15	16
体重过轻 （轻度消瘦）	16	18.5
体重正常	18.5	25
体重过重 （肥胖前期）	25	30
肥胖I级 （轻度肥胖）	30	35
肥胖II级 （中度肥胖）	35	40
肥胖III级 （严重肥胖）	40	

二、系统设计

本系统采用C/C++分模块编写，下面分别介绍这些模块。

文件结构如下图：



1. BMI计算及提示模块

1.1 设计思路

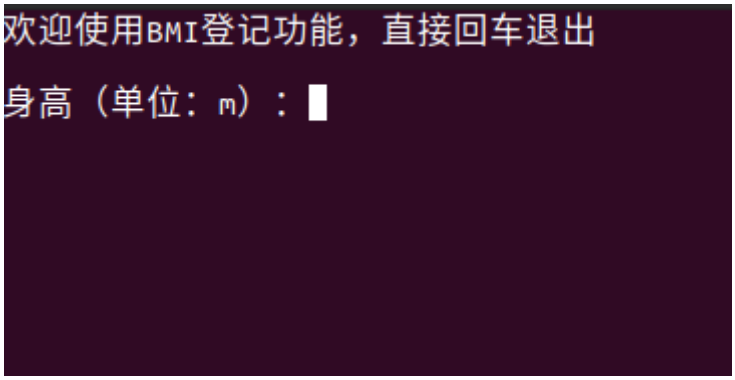
根据国际BMI标准计算公式，我们可以很简单地根据用户提供的身高和体重算出BMI，之后再根据他提供的年龄和计算出的BMI值，查表得到他的体重健康情况。最后将得到的结果存入他的个人数据文件中。

1.2 代码实现

具体的代码实现请见`lib/BMICalculator.cpp`，对应的头文件是`include/BMICalculator.h`

这里仅列出一些代码中的细节：

- 用户可以进入时的界面如下图



在这里我写了一段可以直接回车退出的代码，方便用户快速回到原来的界面选择其他功能，这种退出方式比选择功能数字退出更快捷流畅方便。

```
cout << "欢迎使用BMI登记功能，直接回车退出\n" << endl;
fflush(stdin);
cout << "身高（单位：m）：";
string height;
getline(cin, height);
if (height == "") // 输入为空时直接退出
{
    cout << "\n已退出BMI登记功能" << endl;
    break;
}
```

- 用`try-catch`处理异常输入，去除了身高或体重看错单位导致输入过大和恶意输入非数字的情况。

```
try ···
catch (const std::exception& e)
{
    system("clear");
    cout << RED << "输入错误，请重新输入\n" << RESET << endl;
    continue;
}
```

- 最终展示时粒度只到天，但是一天可以输入多次。我们的处理方式是更关注BMI最高的一次，因此在输入完成后，如果记录中已经有了这个日期的数据，则比较两个数据BMI的大小，保留大的

```cpp
if (records.count(now)) // 如果存在这个日期的数据，则比较大小
{
    if (records[now].bmi < bmi) // 如果现在算出的BMI大于数据文件里的，就覆盖
    {
        auto &t = records[now];
        t.bmi = bmi, t.height = h, t.weight = w, t.prompt = prompt;
    }
}
else // 如果不存在，则直接记录
{
    records[now].bmi = bmi;
    records[now].height = h;
    records[now].weight = w;
    records[now].time = now;
    records[now].prompt = prompt;
}
```

2. 乘法与除法模块

2.1 设计思路

观察BMI标准表可以发现里面的分界数字都是小数点后一位的，因此我们设计一个`divide`函数，它接收用户输入的身高和体重这两个参数，计算出BMI后四舍五入保留到小数点后一位。

2.2 代码实现

具体代码见`lib/divider.cpp`，对应的头文件是`include/divider.h`

```cpp
double divide(double h, double w)
{
    double bmi = w / (h * h);
    double roundedBMI = round(bmi * 10) / 10;  // 四舍五入并保留一位小数
    return roundedBMI;
}
```

3. 日期时间模块

3.1 设计思路

当我们录入BMI数据时，自然要记录它录入的日期，这既是为了更新数据库里数据，也是方便查看。

- `time()`函数，它会将当前日期以长度为8的`YYYYMMDD`形式的字符串返回。
- `setTime()`函数，用户可能并不喜欢系统默认的日期格式，因此这个函数的目的是展示出系统可以提供的所有日期格式，供用户按照功能数字的方式选择，界面如下：

```
--------------------------------------------------------------------------
本系统提供一下日期格式，直接回车退出：

1 YYYY年M月D日
2 YYYY-M-D
3 M/D/YYYY
4 YYYY年MM月DD日
5 YYYY-MM-DD
6 MM/DD/YYYY
--------------------------------------------------------------------------

请输入您想要的格式对应的数字：
```

选择的结果会传递到`parseTime()`函数

- `parseTime()`函数，它根据用户选择的格式，将数据结构中原始的日期字符串`YYYYMMDD`转化为用户所需要的格式。例如，用户选择的`YYYY年MM月DD日`格式的结果如下图：

```
--------------------------------------------------------------------------
BMI管理
version 1.0 by 郑子睿

n:下一页  p:上一页  s:排序  t:设置时间格式  q:退出  g:图
--------------------------------------------------------------------------
1. 2023年07月03日: 55公斤 (BMI: 18.4, 体重过低)
2. 2023年02月20日: 88公斤 (BMI: 23, 正常体重)
3. 2023年02月19日: 76公斤 (BMI: 28.3, 超重)
4. 2023年02月18日: 85.7公斤 (BMI: 33.7, 肥胖)
5. 2023年02月17日: 74.9公斤 (BMI: 30.7, 肥胖)
6. 2023年02月16日: 61.8公斤 (BMI: 17.5, 体重过低)
7. 2023年02月15日: 94.7公斤 (BMI: 25.8, 超重)
8. 2023年02月14日: 72.8公斤 (BMI: 18.8, 正常体重)
9. 2023年02月13日: 61.2公斤 (BMI: 24, 正常体重)
10. 2023年02月12日: 77公斤 (BMI: 24.2, 正常体重)

请输入功能对应的字母:
```

3.2 代码实现

具体代码见`lib/clock.cpp`，对应的头文件是`include/clock.h`

- `time()`函数

```
string time()
{
    // 获取当前系统时间
    auto currentTime = std::chrono::system_clock::now();

    // 将当前系统时间转换为时间点结构 tm
    std::time_t currentTimeT =
std::chrono::system_clock::to_time_t(currentTime);
    std::tm* localTime = std::localtime(&currentTimeT);

    // 从时间点结构 tm 中获取年、月、日
    int year = localTime->tm_year + 1900; // tm_year 是从1900年开始的偏移
量
    int month = localTime->tm_mon + 1; // tm_mon 的范围是 0-11，需要加 1
    int day = localTime->tm_mday;

    std::stringstream ss;
    ss << std::setw(4) << std::setfill('0') << year
```

```
        << std::setw(2) << std::setfill('0') << month
        << std::setw(2) << std::setfill('0') << day;
    return ss.str();
}
```

- `setTime()`函数
  - 展示功能目录

```
cout << "\n-------------------------------------------------------------\n";
cout << "本系统提供一下日期格式，直接回车退出：\n\n";
cout << "1 YYYY年M月D日" << endl;
cout << "2 YYYY-M-D" << endl;
cout << "3 M/D/YYYY" << endl;
cout << "4 YYYY年MM月DD日" << endl;
cout << "5 YYYY-MM-DD" << endl;
cout << "6 MM/DD/YYYY";
cout << "\n-------------------------------------------------------------\n";
cout << "\n请输入您想要的格式对应的数字：";
```

  - 异常处理，主要处理用户未选择而是直接回车以及恶意输入非功能数字的错误

```
string input;
getline(cin, input);
fflush(stdin);
if (input.empty()) return "Y-M-D";
try
{
    // 省略，具体见文件
}
catch (const invalid_argument& e)
{
    system("clear");
    cout << RED << "输入错误，请重新输入\n" << RESET;
    continue;
}
```

- `parseTime()`函数

  依赖C语言的`sprintf`实现

```
string parseTime(const string &time, const string &format)
{
    // 提取年月日
    int year = stoi(time.substr(0, 4));
    int month = stoi(time.substr(4, 2));
    int day = stoi(time.substr(6, 2));
    // 按照选择产生日期格式串
    char buffer[100]{};
    char* form = NULL;
    if (format == "Y年M月D日") sprintf(buffer, "%d年%d月%d日", year, month, day);
    else if (format == "Y-M-D") sprintf(buffer, "%d-%d-%d", year, month, day);
```

```
    else if (format == "M/D/Y") sprintf(buffer, "%d/%d/%d", month, day, year);
    else if (format == "YYYY年MM月DD日") sprintf(buffer, "%04d年%02d月%02d日", year, month, day);
    else if (format == "YYYY-MM-DD") sprintf(buffer, "%04d-%02d-%02d", year, month, day);
    else if (format == "MM/DD/YYYY") sprintf(buffer, "%02d/%02d/%04d", month, day, year);

    string formattedTime(buffer);
    return formattedTime;
}
```

4. BMI 记录存取模块

4.1 设计思路

这个模块我们主要实现了两个功能，一是从文件中读取该登录用户的所有BMI记录，存到数据结构里，二是将数据结构中的所有BMI记录写到文件里保存。

- `loadRecord()`函数，负责根据用户名找到储存该用户所有数据的文件，将里面的数据逐行读出并解析，存到数据结构里。
- `saveRecord()`函数，负责将数据结构中的所有BMI记录写到文件里。采用直接重写覆盖的方式。

  文件内容形式如下图：

4.2 代码实现

具体代码见`lib/BMIRecordManager.cpp`，对应的头文件是`include/BMIRecordManager.h`

- `loadRecord`函数采用C++的`ifstream`读取文件，每次用`getline`读一行，直到文件尾。做了文件打不开的异常处理。用`istringstream`解析读入的每行内容的信息，同样做了解析失败的异常处理。

```
bool loadRecord(string username, RECORD &records)
{
    ifstream iFile("UserData/" + username + ".txt");
    // 异常处理
    if (!iFile.is_open())
    {
```

```
        cerr << "打开用户文件失败！" << endl;
        return false;
    }
    string line;
    while (getline(iFile, line))
    {
        istringstream iss(line); // 解析字符串
        string time, prompt;
        double bmi, height, weight;
        try { // 异常处理
            iss >> time >> height >> weight >> bmi >> prompt;
            // 省略，这里是将变量存到数据结构里
        }
        catch (const invalid_argument& e) {
            break;
        }
    }
    iFile.close();
    return true;
}
```

- `saveRecord`函数，主要采用了`ofstream`的方式将数据结构里的BMI记录写到文件中

```
bool savaRecord(string username, RECORD &records)
{
    ofstream oFile("UserData/" + username + ".txt");
    if (!oFile.is_open())
    {
        cout << "\n保存用户数据失败！\n";
        return false;
    }

    for (auto &record : records)
    {
        auto &data = record.second;
        oFile << data.time << ' ' << data.height << ' ' << data.weight
<< ' ' << data.bmi << ' ' << data.prompt << endl;
    }
    oFile.flush();
    oFile.close();
    cout << "\n保存用户数据成功\n";
    return true;
}
```

5. 通用数据结构模块

5.1 设计思路

C++的标准模板库（STL）为我们实现了很多常用的数据结构，安全、强大、方便。本系统选用其中的`vector`和`map`作为底层容器，结合本系统对于数据组织形式的要求，实现了供其他模块使用的数据结构`USERDATA`、`USERINFO`、`RECORD`。

5.2 代码实现

具体代码见`include/DataStructure.h`

首先给出本系统的数据组织形式，即我们定义的两个结构体

```
struct Data // 存储每一条BMI记录的结构体
{
    double height, weight, bmi;
    string time, prompt;
};
struct BasicInfo // 存储用户基本信息的结构体
{
    string username, password, sex, age;
};
```

- `USERDATA`，底层用`vector<Data>`实现，主要作用是以数组的形式存储从文件中读取到的所有BMI记录，方便排序，它的常用API如下：
  - `push_back()`，将一条记录加入容器中
  - `begin()`，获取容器的第一个`iterator`，指向第一个元素的位置
  - `end()`，获取容器的最后一个`iterator`，指向最后一个元素的后一个位置

  代码实现

  ```
  using USERDATA = vector<Data>;
  ```

- `USERINFO`，底层用`map<string, BasicInfo>`实现，主要作用是以关联容器的方式储存用户的基本信息，即以用户名为`key`，用户信息为`value`，它的常用API如下：
  - `[]`，由`key`取出对应的`value`，不存在的映射会直接创建出来，复杂度`O(logn)`
  - `count()`， 用于查看某个`key`是否存在

  代码实现

  ```
  using USERINFO = map<string, BasicInfo>;
  ```

- `RECORD`，底层用`map<string, Data>`实现，主要作用是以关联容器的方式储存用户的BMI记录，即以记录的创建时间为`key`，记录的内容为`value`，它的常用API如下：
  - `[]`，由`key`取出对应的`value`，不存在的映射会直接创建出来，复杂度`O(logn)`
  - `count()`， 用于查看某个`key`是否存在

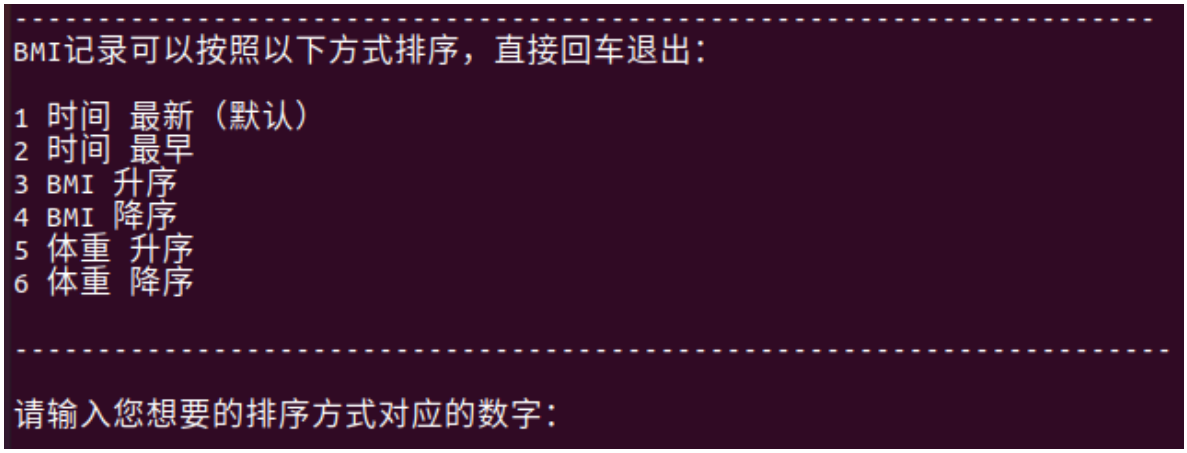
  代码实现

  ```
  using RECORD = map<string, Data>;
  ```

6. BMI记录排序模块

6.1 设计思路

系统默认以创建时间最新排序，但是也为用户提供了按照时间、体重、BMI值升序或降序排序共6种方式，不管列表还是图表形式都支持，旨在帮助用户把握体重健康变化情况，做好健康规划。排序功能界面如下图：



6.2 代码实现

具体代码见`lib/BMILogSort.cpp`，对应的头文件是`include/BMILogSort.h`

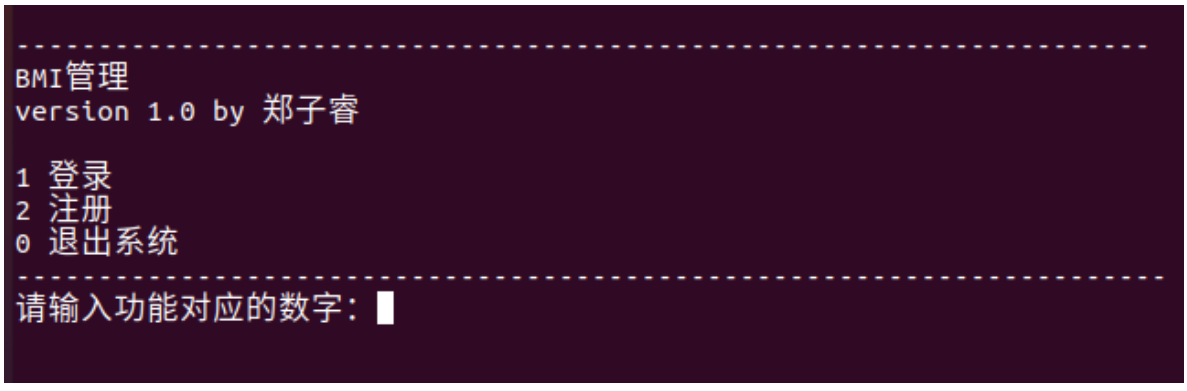
C++的`Algorithm`库中为我们实现了很多常用的算法，这里我们选择其中的`sort`函数，通过为不同的排序方式设计不同的比较函数的方式实现数据的排序。

```
bool cmp1(Data &a, Data &b) {
    return a.time > b.time;
}
bool cmp2(Data &a, Data &b) {
    return a.time < b.time;
}
bool cmp3(Data &a, Data &b) {
    return a.bmi < b.bmi;
}
bool cmp4(Data &a, Data &b) {
    return a.bmi > b.bmi;
}
bool cmp5(Data &a, Data &b) {
    return a.weight < b.weight;
}
bool cmp6(Data &a, Data &b) {
    return a.weight > b.weight;
}
```

7. 用户管理模块

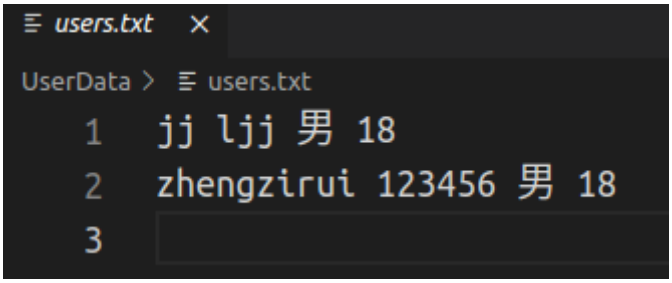
7.1 设计思路

用户管理界面如下图：

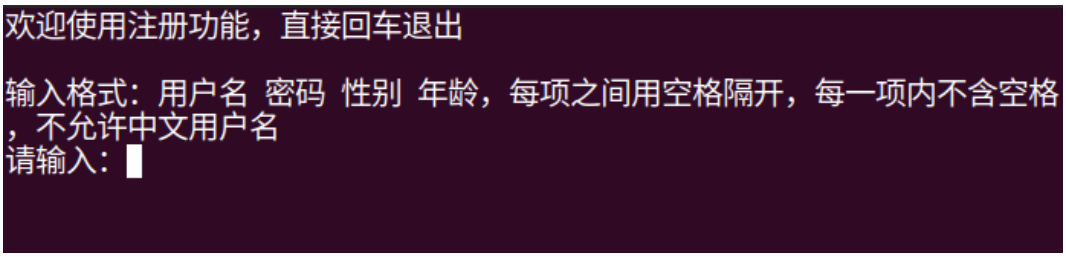


用纯C++实现复杂的用户管理功能是比较困难的，本系统采用的方式是：

1. 将所有用户的基本信息（用户名、密码、性别、年龄），每个用户一行写在文本文件`users.txt`里

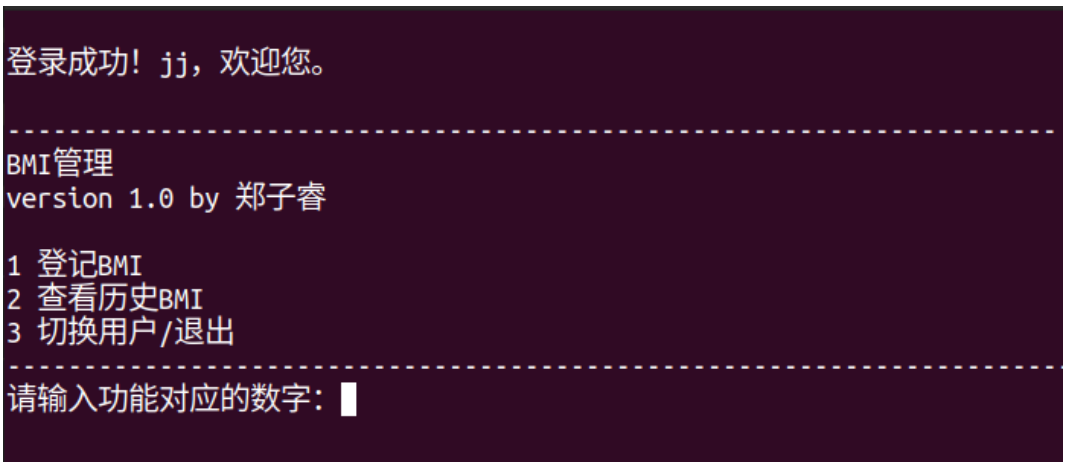


2. 注册功能的实现思路：每次要求用户输入用户名、密码、性别、年龄，判断输入没有问题后：
    1. 将这些信息存到`USERINFO`中，退出注册界面前写入`users.txt`中
    2. 在项目的`UserData`目录下创建一个与用户名同名的文本文件，用来存储该用户的所有BMI记录

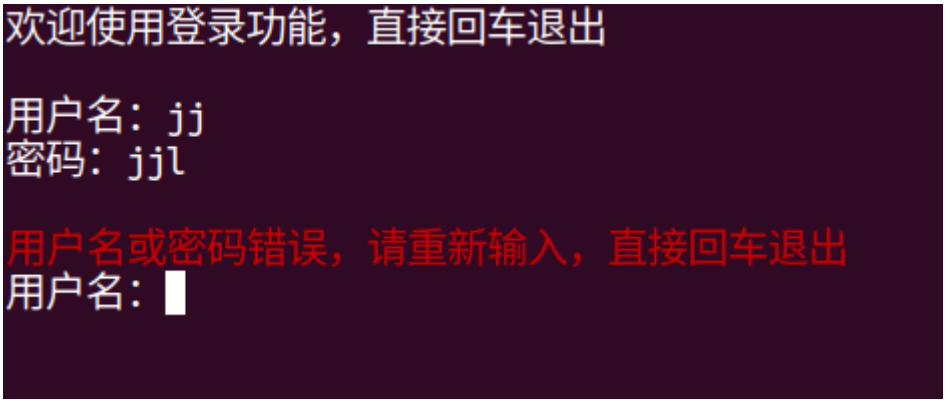


3. 登录功能的实现思路
    1. 每次进入用户管理界面时，从`users.txt`文件中读入所有已注册用户的用户名、密码、性别、年龄，存到`USERINFO`中。
    2. 要求用户输入用户名和密码，比较密码是否正确。

    3. 如果正确则进入用户主界面



    4. 否则提示密码错误请重新输入



7.2 代码实现

具体代码见`lib/UserManagement.cpp`，对应的头文件是`include/UserManagement.h`

1. 打印登录界面

```
void loginMenu()
{
    cout << "\n------------------------------------------------------------\n";
    cout << "BMI管理\n";
    cout << "version 1.0 by 郑子睿\n\n";
    cout << "1 登录\n";
    cout << "2 注册\n";
    cout << "0 退出系统";
    cout << "\n------------------------------------------------------------\n";
    cout << "请输入功能对应的数字：";
}
```

2. 载入用户数据

```
USERINFO loadUserInfo()
{
    ifstream iFile("UserData/users.txt");

    if (!iFile.is_open())
    {
        cerr << RED << "打开用户管理文件失败！" << RESET << endl;
        exit(1);
    }

    string line;
    USERINFO users;
    while (getline(iFile, line))
    {
        istringstream iss(line);
        string username, password, sex, age;
        iss >> username >> password >> sex >> age;
        users[username].username = username;
        users[username].password = password;
        users[username].sex = sex;
        users[username].age = age;
    }
    iFile.close();
```

```
    return users;
}
```

3. 登录功能

```
void login(USERINFO &users)
{
    cout << "欢迎使用登录功能，直接回车退出\n\n";
    fflush(stdin);

    while (1)
    {
        string username;
        cout << "用户名：";
        getline(cin, username);
        if (username == "") break;
        string password;
        cout << "密码：";
        getline(cin, password);
        auto info = users[username];
        if (info.password == password)
        {
            BasicInfo userInfo;
            userInfo.username = username;
            userInfo.password = password;
            userInfo.sex = info.sex;
            userInfo.age = info.age;
            system("clear");
            userMain(userInfo);
            break;
        }
        else
            cout << RED << "\n用户名或密码错误，请重新输入，直接回车退出\n" << RESET;
    }
}
```

4. 注册功能

```
void signup(USERINFO &users)
{
    while (1)
    {
        cout << "欢迎使用注册功能，直接回车退出\n\n";
        fflush(stdin);
        string input;
        cout << "输入格式：用户名 密码 性别 年龄，每项之间用空格隔开，每一项内不含空格，不允许中文用户名\n";
        cout << "请输入：";
        getline(cin, input);
        if (input == "") break;
        if (inputCheck(input) != 4)
        {
            system("clear");
            cout << RED << "\n输入错误，请重新输入\n\n" << RESET;
            continue;
```

```
        }
        else
        {
            string username, password, sex, age;
            istringstream iss(input);
            iss >> username >> password >> sex >> age;
            if (users.count(username))
            {
                system("clear");
                cout << RED << "用户名已存在\n" << RESET;
                continue;
            }
            ofstream oFile("UserData/" + username + ".txt");
            if (!oFile.is_open())
            {
                system("clear");
                cout << RED << "用户文件创建失败！\n" << RESET;
                continue;
            }
            oFile.close();
            cout << "用户文件创建成功\n";
            users[username].username = username;
            users[username].password = password;
            users[username].age = age;
            users[username].sex = sex;
            cout << "注册成功！\n";
            cout << "按任意键继续..." << endl;
            getchar();
            break;
        }
    }
}
```

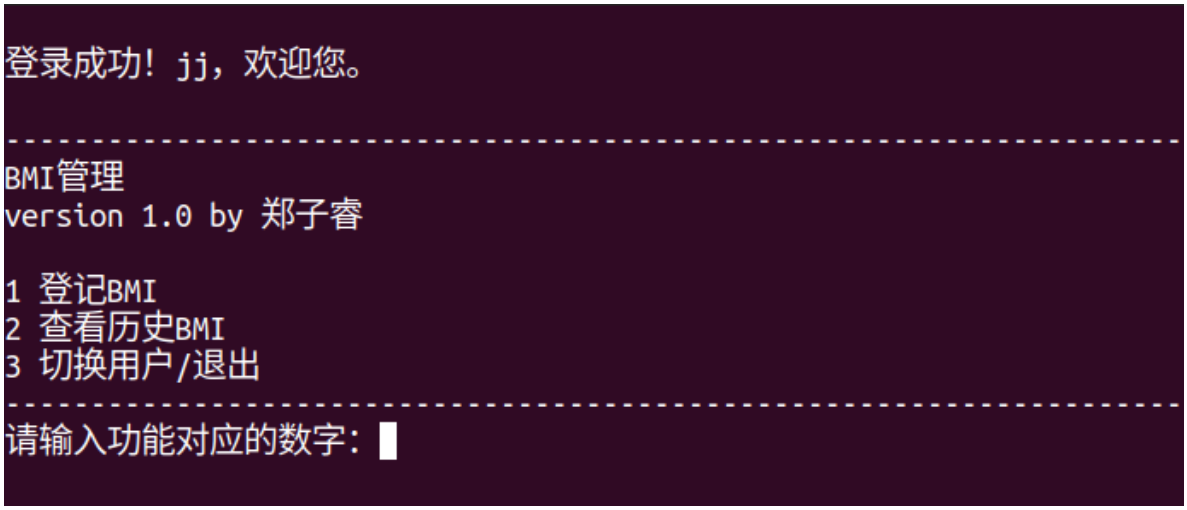
5. 维护登录页面的主函数

```
int loginMain(USERINFO &users)
{
    string input;
    int select;
    while (1)
    {
        system("clear");
        loginMenu();
        getline(cin, input);
        if (input == "")
        {
            system("clear");
            continue;
        }
        try { select = stoi(input); }
        catch (const invalid_argument& e)
        {
            cout << RED << "\n输入错误，请重新输入！\n" << RESET;
            continue;
        }

        switch (select)
```

```
        {
            case 1: system("clear"); login(users); break;
            case 2: system("clear"); signup(users); break;
            case 0: saveUserInfo(users); cout << "已安全退出\n"; return
0;
            default: cout << RED << "\n输入错误，请重新输入!\n" << RESET;
        }
    }
    return 0;
}
```

8. 用户主界面模块

8.1 设计思路



这是用户登录成功后的看到的界面，主要展示三个功能选项

- 登记BMI，这个功能由前面介绍过的BMI计算及提示模块实现
- 查看历史BMI记录，这个模块的实现后文介绍。
- 切换用户/退出，这个功能就是直接返回登录页面，由前面介绍的用户管理模块实现

8.2 代码实现

具体代码见`lib/UserMenu.cpp`，对应头文件是`include/UserMenu.h`

1. 打印主界面

```
void userMenu()
{
    cout << "\n------------------------------------------------------
-------------\n";
    cout << "BMI管理\n";
    cout << "version 1.0 by 郑子睿\n\n";
    cout << "1 登记BMI\n";
    cout << "2 查看历史BMI\n";
    cout << "3 切换用户/退出";
    cout << "\n------------------------------------------------------
--------------\n";
    cout << "请输入功能对应的数字: ";
}
```

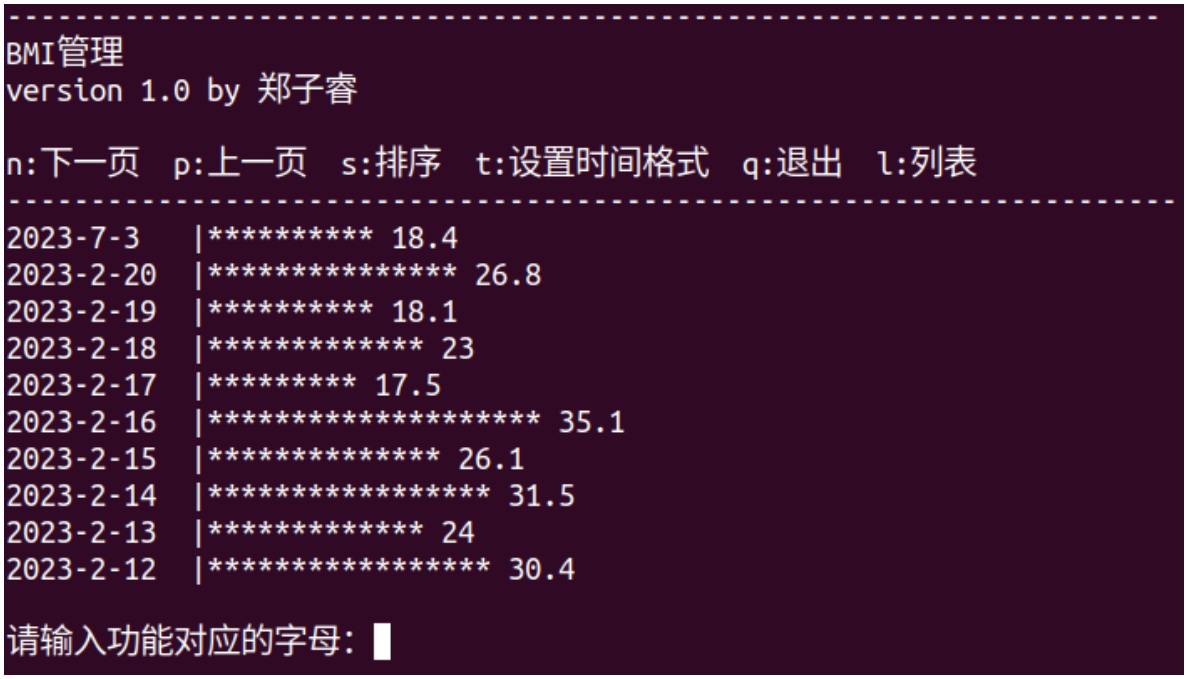
2. 三个功能模块的入口

```
void userMain(BasicInfo userInfo)
{
    RECORD records{};

    if (!loadRecord(userInfo.username, records)) return;

    string input;
    int select;
    while (1)
    {
        cout << "\n登录成功! ";
        cout << userInfo.username << "，欢迎您。\n";
        userMenu();
        getline(cin, input);
        if (input == "")
        {
            system("clear");
            continue;
        }
        try {
            select = stoi(input);
        }
        catch (const invalid_argument& e) {
            system("clear");
            cout << RED << "\n输入错误，请重新输入!\n" << RESET;
            continue;
        }
        system("clear");

        switch (select)
        {
            case 1: bmiCalc(records, userInfo.age); break;
            case 2: bmiHistory(records); break;
            case 3: savaRecord(userInfo.username, records); return;
            default:
            {
                system("clear");
                cout << RED << "\n输入错误，请重新输入!\n" << RESET;
                break;
            }
        }
    }
}
```

9. 柱状图模块

9.1 设计思路

图像的形式往往比列表更富有表现力，因此我们实现了根据用户的每一条BMI记录中的日期和BMI数值画出柱状图。



9.2 代码实现

具体代码见`lib/BMILogs.cpp`中的`bmiLogGraph`函数，对应头文件是`include/BMILogs.h`

首先设定最大的BMI宽度是20，计算出单位BMI的宽度，则其他的BMI值只需乘上单位BMI的宽度即可得到它应有的宽度。

```
void bmiLogGraph(USERDATA &records, int start, int increment, string format)
{
    USERDATA sub(records.begin() + start, records.begin() + min(start + increment, (int)records.size()));

    // 寻找最大的BMI值
    double maxBmi = 0.0;
    for (auto record : sub) {
        if (record.bmi > maxBmi) {
            maxBmi = record.bmi;
        }
    }

    // 定义长条的比例因子
    const int barLength = 20;
    const double scaleFactor = barLength / maxBmi;

    // 绘制条形图
    for (auto record : sub) {
        // 打印日期和BMI值
        cout << setw(10) << left << parseTime(record.time, format) << " ";
        cout << "|" << string(static_cast<int>(record.bmi * scaleFactor), '*') << " ";
        cout << record.bmi << endl;
    }
}
```

10. BMI历史记录查看模块

10.1 设计思路

BMI历史记录查看模块首先借助BMI记录存取模块得到用户的所有BMI记录，然后将这些数据储存在`USERDATA`结构中。

进入历史记录查看界面看到的是功能选项和列表形式展示的BMI记录。提供如下功能：

- `n下一页`：列表形式和柱状图形式一次都只展示十条记录，可以通过输入`n`（不区分大小写）回车查看更多的数据。如果没有更多数据了，会一直显示最后的数据。
- `p上一页`：通过输入`p`（不区分大小写）回车返回查看之前看过的数据，如果没有更多数据了会一直显示最初的数据。
- `s排序`：该功能由之前介绍过的BMI记录排序模块实现，具体见前文
- `t设置时间格式`：该功能由之前介绍过的日期时间模块实现，具体见前文
- `q退出`：该选项会退出历史记录查看功能，返回前面的用户主界面
- `g图`：该功能由前文介绍的柱状图模块实现

10.2 代码实现

具体代码见`lib/BMILogs.cpp`，对应头文件是`include/BMILogs.h`

1. 打印用户交互界面

```
1 void bmiLogMenu(bool state)
2 {
3     cout << "\n-------------------------------------------------------------\n";
4     cout << "BMI管理\n";
5     cout << "version 1.0 by 郑子睿\n\n";
6     cout << "n:下一页  p:上一页  s:排序  t:设置时间格式  q:退出  ";
7     cout << (state ? "g:图" : "l:列表");
8     cout << "\n-------------------------------------------------------------\n";
9 }
```

2. 用户交互函数

```cpp
void bmiHistory(RECORD &records)
{
    USERDATA all;
    for (auto &i : records)
    {
        auto &record = i.second;
        Data t;
        t.time = record.time;
        t.height = record.height;
        t.weight = record.weight;
        t.bmi = record.bmi;
        t.prompt = record.prompt;
        all.push_back(t);
    }
    bmiLogSort(all, 1);
    bool state = true;
    while (1)
    {
        int sl = 0, sg = 0, increment = 10;
        string format = "Y-M-D";
        while (state)
        {
            system("clear");
            bmiLogMenu(state);
            bmiLogList(all, sl, increment, format);
            cout << "\n请输入功能对应的字母：";
            string select;
            getline(cin, select);
            if (select == "n" or select == "N")
            {
                if (sl + increment < all.size())
                    sl += increment;
            }
            else if (select == "p" or select == "P")
            {
                if (sl - increment < 0) sl = 0;
                else sl -= increment;
            }
            else if (select == "s" or select == "S")
            {
                bmiSortMenu(all);
                sl = 0;
            }
            else if (select == "t" or select == "T") format =
setForm();
            else if (select == "q" or select == "Q")
            {
                system("clear");
                return;
            }
            else if (select == "g" or select == "G") state = false;
        }
        while (!state)
        {
            system("clear");
            bmiLogMenu(state);
            bmiLogGraph(all, sg, increment, format);
            cout << "\n请输入功能对应的字母：";
```

```
            string select;
            getline(cin, select);
            if (select == "n" or select == "N")
            {
                if (sg + increment < all.size())
                    sg += increment;
            }
            else if (select == "p" or select == "P")
            {
                if (sg - increment < 0) sg = 0;
                else sg -= increment;
            }
            else if (select == "s" or select == "S")
            {
                bmiSortMenu(all);
                sg = 0;
            }
            else if (select == "t" or select == "T") format =
setForm();
            else if (select == "q" or select == "Q")
            {
                system("clear");
                return;
            }
            else if (select == "l" or select == "L") state = true;
        }
    }
}
```

3. 列表形式显示用户BMI记录

```
void bmiLogList(USERDATA &records, int start, int increment, string
format)
{
    USERDATA sub(records.begin() + start, records.begin() + min(start +
increment, (int)records.size()));
    int index = start;
    for (auto &record : sub)
    {
        cout << index + 1 << ". ";
        cout << setw(10) << left << parseTime(record.time, format) <<
": " ; cout << record.weight << "公斤" << "（BMI: " << record.bmi << ",
" << record.prompt << "）" << endl;
        index ++ ;
    }
}
```

4. 柱状图形式显示用户BMI记录

```
void bmiLogGraph(USERDATA &records, int start, int increment, string
format)
{
    USERDATA sub(records.begin() + start, records.begin() + min(start +
increment, (int)records.size()));

    // 寻找最大的BMI值
```

```
    double maxBmi = 0.0;
    for (auto record : sub) {
        if (record.bmi > maxBmi) {
            maxBmi = record.bmi;
        }
    }

    // 定义长条的比例因子
    const int barLength = 20;
    const double scaleFactor = barLength / maxBmi;

    // 绘制条形图
    for (auto record : sub) {
        // 打印日期和BMI值
        cout << setw(10) << left << parseTime(record.time, format) << "
";
        cout << "|" << string(static_cast<int>(record.bmi *
scaleFactor), '*') << " ";
        cout << record.bmi << endl;
    }
}
```

三、Makefile

1. 设计思路

采用动态编译和静态编译结合的方式

- 动态编译的模块：BMI历史记录查看模块、BMI记录排序模块、用户管理模块、用户主界面模块
- 静态编译的模块：BMI计算与提示模块、BMI记录存取模块、日期时间模块、乘除法模块
- 安装：静态库不需要安装，将动态库安装到`usr/lib`目录下

2. 代码实现

```
vpath %.cpp lib:.
vpath %.o lib:.
CC = g++
CXXFLAGS = -Iinclude

static_ofiles := BMICalculator.o BMIRecordManager.o clock.o divider.o
dynamic_cpps := BMILogSort.cpp UserManagement.cpp UserMenu.cpp BMILogs.cpp
libs := libbmi.so libbmi.a
main: $(libs)
    $(CC) -o $@ main.cpp -L. $(libs) -Iinclude
libbmi.a: $(static_ofiles)
    ar rcs $@ $^
libbmi.so: $(dynamic_cpps)
    $(CC) -o $@ -fPIC -shared $^ -Iinclude
$(static_ofiles): %.o : %.cpp

install:
    sudo cp libbmi.so /usr/lib/

.PHONY: clean
```

```
clean:
    -sudo rm /usr/lib/libbmi.so
    -$(RM) main
    -$(RM) *.o
    -$(RM) lib/*.o
    -$(RM) *.a *.so
```

四、结语

BMI管理系统作为一种基于BMI指数的计算和管理工具，为个体健康管理和公共卫生政策制定提供了重要支持。然而，在系统的设计与实现中，我们也发现了一些可以改进的方面，特别是用户的交互和异常处理的完善，以及功能的进一步丰富化。

首先，用户的交互体验是一个关键因素，直接影响着系统的易用性和用户满意度。目前的BMI管理系统可能存在交互设计不够友好、操作流程不够直观等问题。为了改进用户的交互体验，我们可以考虑优化系统界面的设计，简化操作流程，并提供清晰明了的指导和反馈，使用户能够更轻松地使用系统进行BMI数据的管理和分析。

其次，异常处理是系统稳定性和可靠性的关键要素之一。在目前的BMI管理系统中，对于各种异常情况（如输入错误、无效数据等）的处理可能还不够全面。为了提高系统的健壮性，我们需要增加对异常情况的检测和处理机制，例如，对用户输入进行验证和校正，及时向用户提供错误提示和建议，确保系统能够正常运行并准确处理各种情况。

最后，功能的丰富化也是提升BMI管理系统的重要方面。目前的系统可能只提供了基本的BMI计算和数据管理功能，但在实际应用中，用户可能还有其他需求，如健康建议、趋势分析等。为了满足用户的多样化需求，我们可以考虑进一步拓展系统功能，增加更多实用的功能模块，使系统更加全面、实用和具有吸引力。