题目二：

一、总体设计

1．功能模块设计

总体功能上的实现大致按照以下顺序。

（1）构造全员信息表

（2）排队登记

（3）检测核酸

（4）查看排队情况

（5）等级检测结果

（6）各类人员查询

（7）个人查询

（8）构造全员信息表

（9）显示全员信息表

2．数据结构设计

首先介绍一些自定义的类。

- class Person_Information;

  人员信息类。类中私有成员变量有代表每个人的试管号以及两种状态。均为 string 类型。

- class Queue_Information;

  队列信息类。类内私有成员变量有代表每个人的人员代码以及一个 flag 标识，用于区分单检还是混检。

- class Single_tube;

  单人核酸检测管类，类内私有成员变量为代表每个人的试管号和人员代码。

- class Mixed_tube;

  混合核酸检测管类，类内成员私有变量为代表每十个人的试管号和十个人的人员代码。

- class Nucleic_acid_check_system;

  核酸检测系统类，类内私有成员变量如下图所示：

```
private:
    LinkQueue<Queue_Information> Mix_que;                    //排队队列
    LinkQueue<Queue_Information> Single_que;                 //排队队列
    vector<Single_tube> Single;                              //单人测试管vector容器
    vector<Mixed_tube> Mix;                                  //混合测试管vector容器
    map<string,Person_Information>  All_information;         //全员信息大表
    int single_tube_number = 0;                              //控制单管流水号
    int mixed_tube_number = 0;                               //控制混管流水号
    int count1 = 0;                                          //控制混检10人一组编号;
    string Current_Mix_number;                //记录当前分配的混检试管号
    string Current_Single_number;             //记录当前分配的单管号
    int count = 0;                            //控制混检核酸管的人员代码（10人一组）
```

图 2.1Nucleic_acid_check_system 类中私有成员变量图

可以看到其中有一些用到的数据结构，如队列、vector 容器、map 容器等，下面分别介绍：

（1）队列。

根据题目要求，需要模拟排队情况，进行排队登记，选择做混检还是单检。因此在自定义的核酸检测系统的类里加入两个队列，分别保存单检和混检的排队信息。队列中的元素是自定义的队列信息类，用于保存每个人的初始排队信息。

（2）vector 容器

当排队完成后，需要进行核酸检测，每检测完一个人，需要出队，但是此前的排队信息仍需保留，因为在接下来的可能发生的诸多的事件中，比如排队时已经有人患上新冠，之后才检测出来，仍需要获取当时的的排队信息，确定密接等信息。所以在人们做完检测后，出队的同时，将出队的信息放入 vector 容器中，里面的类型是自定义的管子类。检测的过程就是出队并且封装在 vector 管子容器里的过程。

（3）map 容器

一个设计良好的核酸检测系统，每个用户的查询耗时一定不能太长。因此在拥有 1e5 数据量的要求下，选择 map 容器。map 容器的键值映射使得个人查询耗时接近 logn，所以 map 容器里的 key 是人员代码，value 是自定义的人员信息类，里面保存有人员的基本信息，包括检测试管编号、状态等。

3．各个函数功能描述

从面向对象，即面向用户的角度出发，设计各个函数功能的实现。具体分为主要的功能实现函数和辅助函数。核酸检测系统类内部分公有函数图如下：

```
string Get_Single_tube_number();                                    //得到单管的序号
string Get_Mixed_tube_number();                                     //得到混管的序号
map<string, Person_Information>::iterator Find_through_personcode(string s);          //通过人员代码查询状态
void Show_STATUS(string STATUS);                                    //展示某种状态的人
void Line_up_in_file();                                             //排队登记(文件)
void Nucleic_acid_detection_in_file();                              //检测核酸(文件)
void Line_up();                                                     //排队登记
void Nucleic_acid_detection();                                      //检测核酸
void Show_que();
void Rejister_result();                                             //登记检测结果
void Status_query();                                                //各类人员查询
void Construct_ALL_Information();                                   //构建全员信息大表
void Show_map();                                                    //显示全员信息大表
```

图 3.1 核酸检测系统类内部分公有函数图

函数介绍：

string Get_Single_tube_number();	根据题目要求，要得到开头为 1，后面 4 位为流水号的单检试管号，因此该函数每次调用，会返回一个新的单检试管流水号，如第一次调用返回 10000，第二次调用返回 10001，之后类似。
string Get_Single_tube_number();	根据题目要求，要得到开头为 0，后面 4 位为流水号的混检试管号，因此该函数每次调用，会返回一个新的混检试管流水号，如第一次调用返回 00000，第二次调用返回 00001，之后类似。
map<string, Person_Information>::iterator Find_through_personcode(string s);	该函数需要一个字符串作为参数，实际给出的参数为人员代码，使得通过调用该函数可以从人员代码查询人状态。函数返回一个 map 类型的迭代器，使得可以返回指向该人员代码的指针，如果不存在该人员代码，则返回末尾键值对的指针。
void Show_STATUS(string STATUS);	该函数需要一个字符串作为参数，实际给出的参数为要查询的状态，例如：Negative，Diagnosis，Suspicious，Contact，Sub_Contact，Wait_for_upload_result，In_Line 等。调用该函数可以展示相应状态的全部人员及其信息。
void Line_up_in_file();	从文件读取要排队的人，分别读取人员代码，让单检和混检入队，便于后续的测试与处理。
void Nucleic_acid_detection_in_file();	从文件读取要进行核酸检测的人，分为混检人数读入和单检人数的读入，便于后续的测试与处理。
void Line_up();	从键盘输入数字，进行排队登记的函数，调用该函数可以进行手动的排队。同时相应人员状态更新为”In_Line”
void Nucleic_acid_detection();	从键盘输入数字，进行核酸检测的函数，调用该函数可以进行手动的

	控制要进行单检和混检的人数。同时相应人员状态更新为”Wait_for_upload_result”。
void Show_que();	展示当前的排队情况。
void Rejister_result();	登记核酸检测结果，调用函数后可以先输入核酸检测管号，然后输入核酸检测结果 Negative ，Diagnosis 等。登记试管检测结果的同时，也将相应人员的状态进行更新，如果有确诊出现，相关联的人员状态也会发生相应的更新。
void Status_query();	各类人员查询函数，通过调用此函数，输入相应数字选择状态，实现各类状态人员的查询
void Construct_ALL_Information();	初始化 map，将所有的人员代码，初始状态先行初始化，即生成一个拥有很多键值对的全员信息大表。
void Show_map();	显示全员信息大表。

二、详细设计

1．各个函数的调用关系图

由于函数进行了模块化设计，基本上一个功能对应一个函数，每个函数又或多或少调用了一些辅助函数（比如自定义的几个类里的公有函数，辅助实现其功能。大致的函数调用关系图如下：

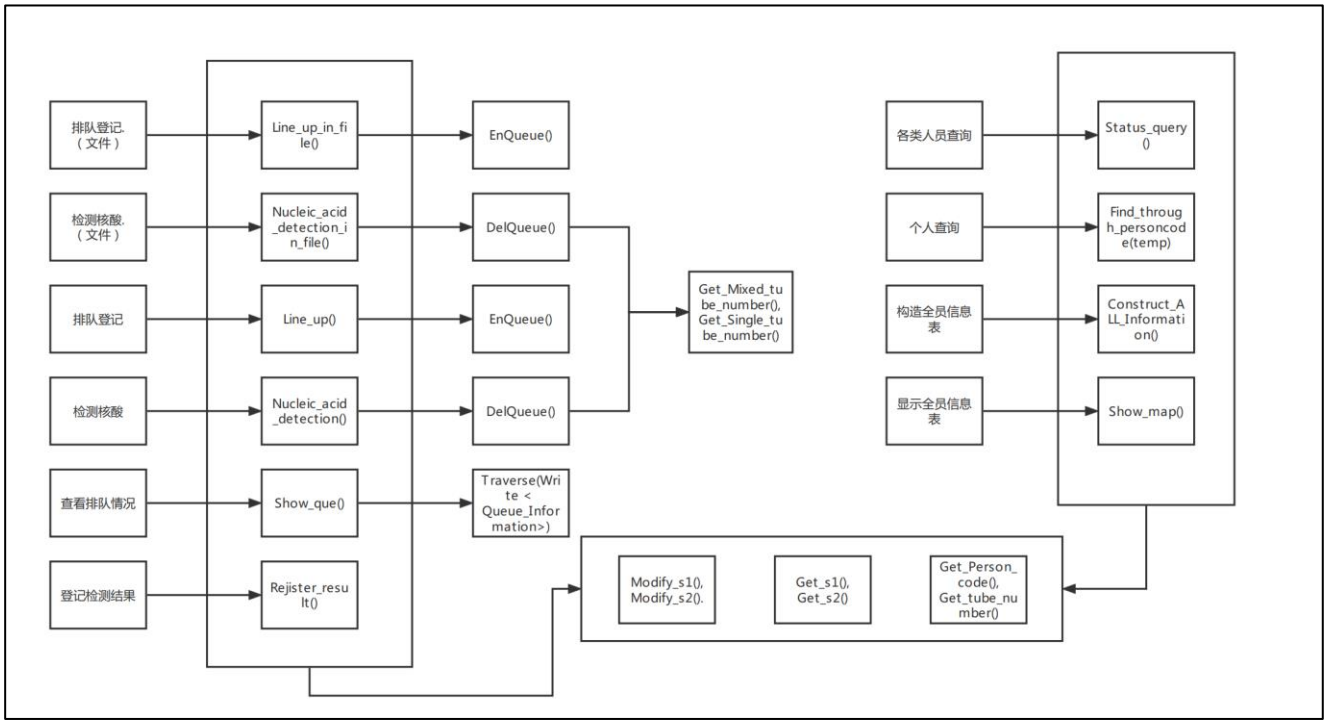

图 1.1 函数调用关系图

较为常见的一般函数如 string 库里的函数、还有一些简单的返回成员变量值的函数就不再赘述了，只表达体现功能性的重要的函数关系，以及它们之间的调用情况。

2. 函数设计

函数设计其实就是功能设计。函数是辅助实现功能的方法。分析题目需求，找出核酸检测系统应具备的一些功能，将功能细化为函数的组合，以达到预期的效果。

功能需求主要有：排队登记.（文件）、检测核酸.（文件）、排队登记、检测核酸、查看排队情况、登记检测结果、各类人员查询、个人查询、构造全员信息表、显示全员信息表。

对应的主要的函数分别为：Line_up_in_file()、Nucleic_acid_detection_in_file()、Line_up()、Nucleic_acid_detection()、Show_que()、Rejister_result()、Status_query()、Find_through_personcode()、Construct_ALL_Information()、Show_map()。

根据自定义的排队信息类、人员信息类、单管混管类，这些主要是保存单个实例的自定义类型，须与相应的数据结构配套使用。比如排队信息类与队列搭配实现排队功能，人员信息类与 map 容器搭配实现快速查询功能，并构造出全员信息的大表，单管混管类放进 vector 容器中，用于核酸检测保存记录和保存排队情况。

所以在这些简单的自定义类型中也需要有一些类内的成员函数，辅助实现实际功能。比如在人员信息类中，需要定义修改状态的函数。因为人员状态的变化在核酸检测过程中是很常见的，如果有确诊出现，可能会大量的修改人员的状态，所以定义修改状态的函数，完善人员信息类是非常有必要的。并且由于使用了 map 容器，以人员代码作为 key 值，人员信息作为 value 值，通过人员代码作为键，找到相应人员信息花费的时间是对数级别的，大大减少了查找时间，提

高了程序整体运行的速度。但是 map 里的 value 值的修改却是有些麻烦的。因为 map 里键必须是唯一的，即人员代码是唯一的，但 value 的修改必须重新覆盖键值，这样仍需要重新实例化一个自定义的类的临时变量，繁琐也浪费空间。最初的想法也是这样实现的，导致代码冗杂，可读性也不高，但是之后便发现了 value 的值可以通过完善自定义的类里的函数来实现，这样每个人员信息就都有了属于自己的修改状态函数，整体代码更加简洁，可读性增强。

在检测核酸功能的设计中，也遇到了一些困难。检测的过程可以理解为出队的过程，人员出队也就是做完了核酸检测，但是检测完之后代表检测过的人已经拥有了属于自己的试管号，并且试管号需要是流水号。考虑再三，决定将试管号作为字符串，控制流水的 int 类型的变量放在核酸检测系统类里的私有成员变量中，所以只需要实现一个函数具有如下描述的功能，每次调用该函数，会返回一个相应的 string 类型的流水号。单检比较简单，但是混检需要十人共同拥有一个相同的管号，所以需要新增加一个在类里的变量，控制十次调用才更新一次管号。并且其中有数字与字符串之间相互转化的部分，写成一个函数后，例如(string Get_Single_tube_number(),string Get_Mixed_tube_number()),后期的调用分配管号等功能的实现也就变得简单起来。即实现复杂功能，要将复杂的功能剖析为一些简单的函数，通过调用这些简单函数的组合达到目的。

三、功能测试

进行功能测试前，请先大致阅读该项目的 README.md，以便更好的理解程序的功能，和一些具体的操作细节。

测试步骤可以阅读 README.md 里的运行指导部分，这里仅展示 README.md 里的运行截图，并加以适当的说明。

首先按顺序选择 7，a,b 功能，从文件读取排队，和做部分人员的核酸检测检测。

将混检试管 00000 登记为 Diagnosis.即产生相应状态的 Suspicious 人员。选择 5 进行各类人员查询，选择 3 展示 Suspicious 人员的信息，运行情况截图如下：

```
请输入想要查询的状态
1.阴性 2.确诊 3.可疑 4.密接 5.次密接 6.待上传结果 7.在排队
3
05601011        00000   null    Suspicious

05701011        00000   null    Suspicious

05801011        00000   null    Suspicious

06001011        00000   null    Suspicious

06001012        00000   null    Suspicious

06001013        00000   null    Suspicious

06001014        00000   null    Suspicious

06001021        00000   null    Suspicious

06001022        00000   null    Suspicious

06001023        00000   null    Suspicious
```

图 3.1.1 运行结果一

选择 4，按下 enter;

输入 10004,按下 enter,即登记第五个单检管;他的人员代码从文件中不难看出是 05801012。

输入 Diagnosis,按下 enter；再展示确诊情况的运行截图如下：

```
请输入想要查询的状态
1.阴性 2.确诊 3.可疑 4.密接 5.次密接 6.待上传结果 7.在排队
2
05801012        10004   null    Diagnosis
```

图 3.1.2 运行结果二

选择 5，按下 enter;

再选择 4，按下 enter,展示出 Contact 状态下的人的信息。

```
请输入想要查询的状态
1.阴性 2.确诊 3.可疑 4.密接 5.次密接 6.待上传结果 7.在排队
4
05601012        10003   Wait_for_upload_result  Contact

05701012        10005   Wait_for_upload_result  Contact

05801013        null    Undetected      Contact

05801014        null    Undetected      Contact

05801021        null    Undetected      Contact

05801022        null    Undetected      Contact

05801023        null    Undetected      Contact

05801024        null    Undetected      Contact
```

```
05812044        null    Undetected      Contact

05901011        10000   Wait_for_upload_result  Contact

05901012        10001   Wait_for_upload_result  Contact

05901013        10002   Wait_for_upload_result  Contact
```

图 3.1.3 运行结果三

可以看到单检将前面 10 人（目前只有四个人，但运行仍然良好）和后面一个人修改为密接，同一栋楼的都改为密接。同一栋楼密接密接太多，就不进行截图展示了。

下面再展示次密接，根据题目要求，密接者的楼栋修改为次密接，058 号楼栋本就是确诊人员所在的楼栋，状态不变，仍未密接，而受其影响的 056、057、059 都应该是次密接。程序运行良好，符合预期，数据较多，下面仅展示部分结果：

05912032	null	Undetected	Sub_Contact
05912033	null	Undetected	Sub_Contact
05912034	null	Undetected	Sub_Contact
05912041	null	Undetected	Sub_Contact
05912042	null	Undetected	Sub_Contact
05912043	null	Undetected	Sub_Contact
05912044	null	Undetected	Sub_Contact

05712034	null	Undetected	Sub_Contact
05712041	null	Undetected	Sub_Contact
05712042	null	Undetected	Sub_Contact
05712043	null	Undetected	Sub_Contact
05712044	null	Undetected	Sub_Contact
05901014	10006	Wait_for_upload_result	Sub_Contact
05901021	10007	Wait_for_upload_result	Sub_Contact

05612032	null	Undetected	Sub_Contact
05612033	null	Undetected	Sub_Contact
05612034	null	Undetected	Sub_Contact
05612041	null	Undetected	Sub_Contact
05612042	null	Undetected	Sub_Contact
05612043	null	Undetected	Sub_Contact
05612044	null	Undetected	Sub_Contact

图 3.1.4 运行结果四

对于其他状态的查询、登记是类似的操作，详情请见 README.md;对于一些异常输入的判定情况由于太过繁琐，并没有写出提示性的话语，不过相关特殊情况的判定以及避免异常输入的

方法，写在了 README.md 中，通过仔细阅读，可避免错误操作。即使误操作，也不会有太大的影响，只是提示语未一一给出。

遇到的困难：

为解决题目二，给出一般性的解决方案其实并不复杂，但是怎样让程序的数据结构更加合理，逻辑更加清晰，运行情况更加迅速，尽量减少运行时间，才是解决问题的关键。

困难一：在思考核酸检测这个过程中，排队的情况是最容易实现的，只需两个队列就能实现。但是核酸检测又是如何进行的，经过分析，出队就是核酸检测完成时。但是需要为人员分配相应的试管号，单检的情况容易控制，但是混检的人数检测容易出现不为 10 的整数倍的情况，因此需要考虑多种情况，最后根据实际情况出发，通过在类里加新的成员变量的方法（初始值为零，当累计加到 10 之后置零，并考虑所有可能出现的异常情况，针对不同情况进行不同的处理），解决混合检测引发的一系列的问题，只要程序一直在运行着，这种异常情况就能够解决。通过这种类似的手段，也解决了混检流水管号的分配问题。同时提供这样一个可以查看这种值的接口，也能反过来避免程序的异常输入，提高了程序的鲁棒性，一定程度上减少了崩溃的风险。

困难二：近期健康云等核酸检测系统在访问量较多的时候出现并发等一系列的问题，如果程序某个功能运行的更快，耗时更少，服务器的压力就会小很多。所以一定要采用合理的数据结构，使得遍历的次数尽量少，遍历的速度尽量快。本来是采用线性表来做全员信息表的记录，但是查询时间太慢，复杂度 O（n），如果数据小，这样做并无不妥，但如果数据足够大，每个查询都需要耗费如此多的时间，程序运行会变的很慢。于是仔细分析问题需求，发现 stl 的库函数里的 map 容器，在查询时间方面有着质的飞跃，如果 key 有序，时间复杂度会接近 O(logn)。于是开始读相关的文档、资料，学习使用 map，同时也学习了 vector 的用法，使得程序运行速度得到迅速的提高。