程序设计基础训练（80L878Q）

实验#7

交付物提交时间要求

- 详见《程序设计基础训练课程安排表》

相关知识点

- 文件读写、模块化程序设计
- 指针、链表、结构体

需自学的相关技术

- 定时器的使用

实验目的

- 训练学生掌握程序开发工程组织方式；
- 训练学生掌握外部程序调用技术；
- 训练学生掌握程序设计方法，了解程序设计与开发过程。

实验内容

- 程序设计：

1）回顾实验 5、6 中读取的数据记录文件，现将文件中每一条数据记录看作是一条电梯用户请求指令，即每条数据记录的三元组，分别代表“用户所处楼层”，“用户要到达的楼层”以及“用户发出这条请求的时间点”。举例来说，图 7-1 为数据记录文件内容组织形式效果图，图中数据表示了一部电梯在一定时间段内接收到的用户请求信息。在该记录文件中用户一共发出了 5 条请求指令：

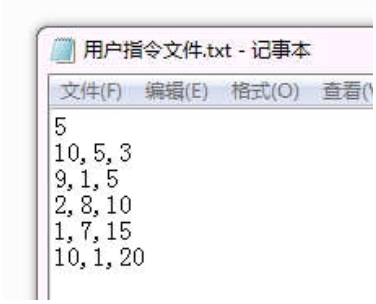

图 7-1 数据记录文件内容组织形式效果图

- 第一个指令在第 3 时刻发出，用户处于第 10 层，要去第 5 层；
- 第二个指令在第 5 时刻发出，用户处于第 9 层，要去第 1 层；
- 第三个指令在第 10 时刻发出，用户处于第 2 层，要去第 8 层；
- 第四个指令在第 15 时刻发出，用户处于第 1 层，要去第 7 层；
- 第五个指令在第 20 时刻发出，用户处于第 10 层，要去第 1 层；

以上述方式对数据记录文件进行解读，复用实验 5 程序中的部分函数，实现对电梯响应用户请求过程的仿真功能。在实现电梯仿真功能时，需遵循以下设计约束：

a) 电梯**运行状态**分为三个状态：
   - 停止状态：表示电梯当前停靠在某一楼层，用 **S** 表示；
   - 上行状态：表示电梯正在向上运行，用 **U** 表示；
   - 下行状态：表示电梯正在向下运行，用 **D** 表示；

b) 电梯三个状态之间的转换关系如图 7-2 所示，由停止状态转换为上升或下降状态，需要消耗 1 个单位时间，且楼层不变，而由上升或下降状态转换为停止状态，不需要消耗时间。举例来说：假设第 3 时刻电梯位于 3 层，当电梯需要上行时，在第 4 时刻，电梯由停止状态转换为上行状态，此时电梯仍处于 3 层，然后，在第 4 时刻，电梯为上行状态，所处楼层变为 4 层，以此类推，后续时刻中，电梯开始上行直到到达目的楼层。同样假设第 3 时刻 A 号电梯位于 3 层，电梯处于上行状态，并且每一个单位时间，电梯可以运动 1 层，那么第 4 时刻电梯应处于第 4 层，状态为上行状态，如果电梯目标是运动到第 5 层，则可以在第 5 时刻运动到第 5 层时直接转换为停止状态，而不需要等到第 6 时刻才转换为停止状态；

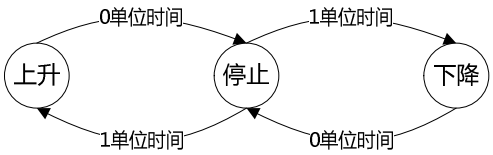

图 7-2 电梯状态转换图

c) 当电梯处于上行或下行状态时，表示电梯正在响应某条用户指令，此时电梯分为两种状态，我们将其称为服务状态，具体**服务状态**包括：
   - 提供服务前状态：在这一状态下，表示电梯正在向用户所在楼层运动，尚未接到用户并开始提供服务，用 **P** 表示；
   - 提供服务中状态：在这一状态下，表示电梯已接到用户并在向用户目标楼层方向移动，用 **E** 表示；

d) 电梯的服务状态与具体用户请求指令相关，当电梯同时服务多条用户请求时，电梯对不同的用户请求的服务状态可能不同；

e) 设置一个**待响应指令队列**，用于模拟用户请求指令依次由用户发出的情景。其保存**当前时刻已到来并尚未得到电梯响应的用户请求指令**，电梯需要不断检查这个队列中是否有用户请求指令并对已到来的指令进行响应；

f) 为电梯设置一个**当前服务指令队列**，存储**当前电梯正在响应的用户请求指令**，当电梯响应用户的某一请求指令时，系统将该指令由**待响应指令队列**中取出，加入到电梯的**当前服务指令队列**中去；

g) 在每一个时刻对当前电梯的状态、待响应的请求指令进行检查，根据当前电梯状态和用户请求指令计算下一个时刻电梯处于什么状态，将下一时刻的电梯状态输出；然后将电梯下一时刻的状态更新为当前状态（相当于时间轴向前走了一步）；重复上述步骤，一直到电梯对所有用户请求指令均完成响应为止，其运行流程如图 7-3 所示；

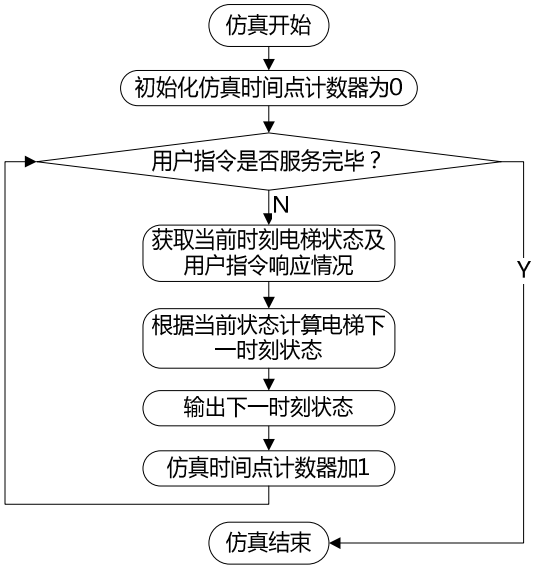

图 7-3 电梯仿真逻辑流程图

h) 默认仿真开始的 0 时刻，电梯处于第 1 层，状态为“S”，当前服务指令队列为空。

i) 每一时刻电梯根据当前状态确定下一时刻状态的具体判断逻辑如图 7-4、图 7-5、图 7-6 所示。

j) 根据上述业务流程，假设某一时刻电梯处于 m 层，电梯当前服务指令队列中有 i 条指令，每条指令对应的用户所在楼层为 f_i，用户目标楼层为 t_i，电梯对该条指令当前的服务状态为 p_i，则电梯在某时刻其状态与电梯响应的用户请求指令各参数间必须满足以下约束关系：

- 当电梯运行状态为“U”或“D”时，电梯服务指令队列肯定不为空；
- 当电梯运行状态为“U”时，若 p_i 为“P”，则必有 $m<f_i$，若 p_i 为“E”，则必有 $m<t_i$；
- 当电梯运行状态为“D”时，若 p_i 为“P”，则必有 $m>f_i$，若 pi 为“E”，则必有 $m>t_i$；
- 当电梯当前服务指令队列不为空时，对于电梯服务队列中的各个指令，当服务状态为“P”时，不存在 $m==f_i$ 的情况，当服务状态为“E”时，不存在 $m==t_i$ 的情况；

- 当电梯运行状态为“S”时，若电梯当前服务指令队列为空，表示此时电梯为空载、静止状态，除非后续有新指令到来，否则不做任何操作，状态也不做改变。若电梯当前服务指令队列不为空，则表示此时电梯仍在响应用户指令为用户提供服务，电梯临时停靠接用户进电梯或完成服务送客户出电梯。

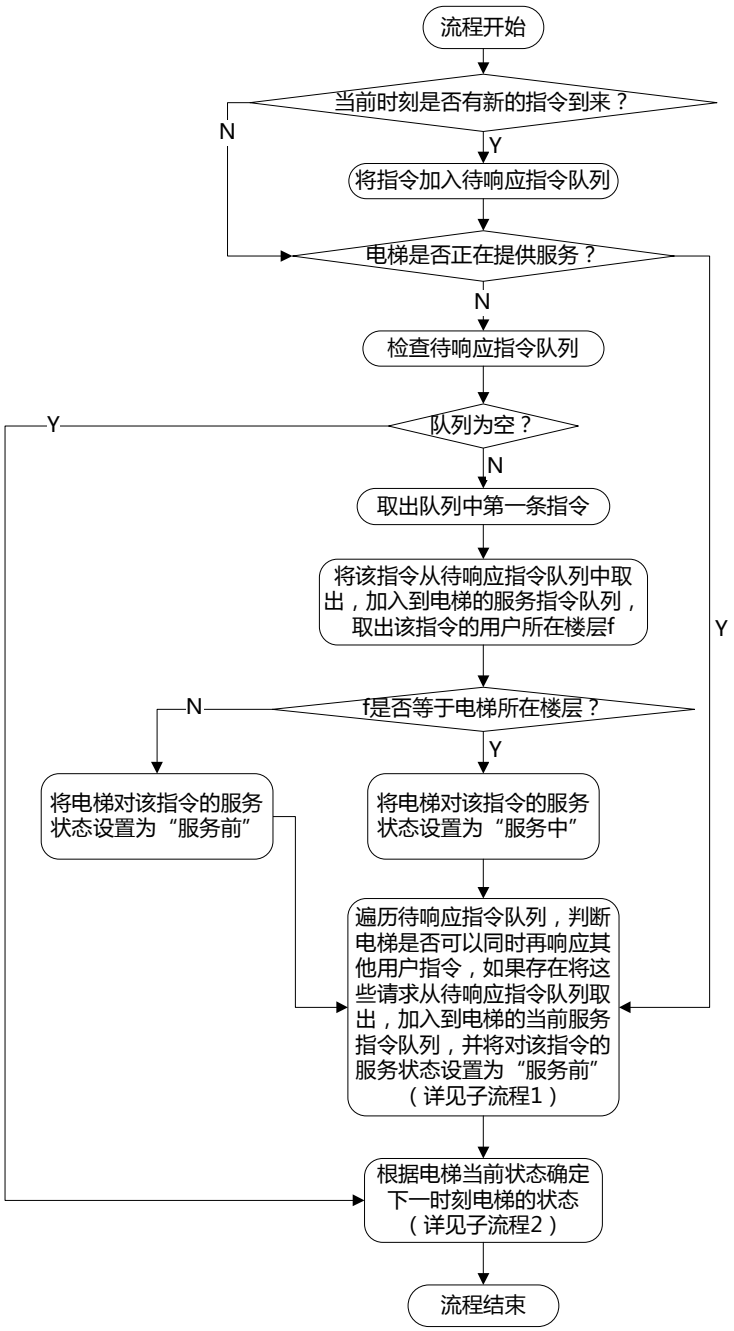

图 7-4 当前时刻电梯处理逻辑流程图

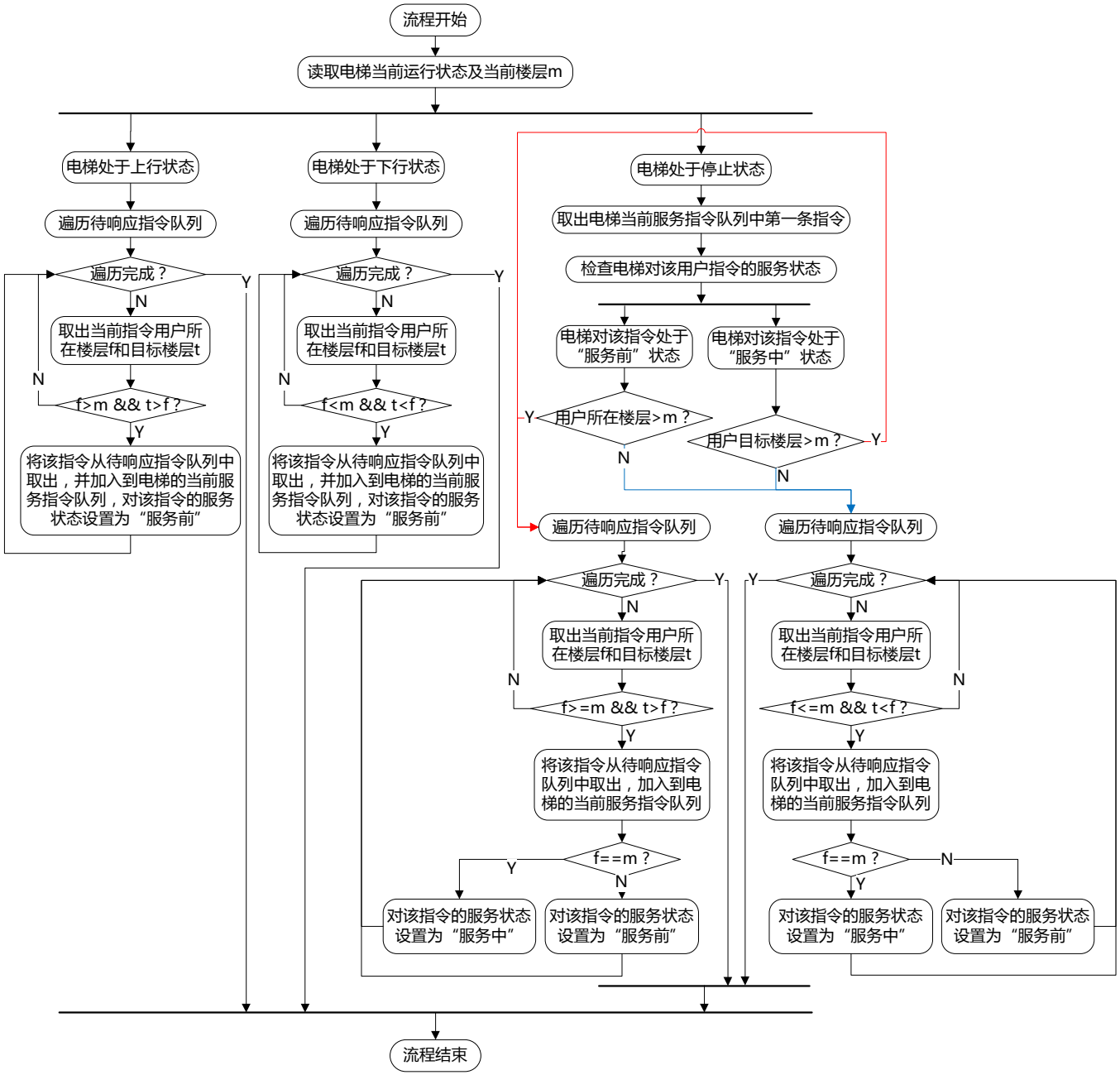

图 7-5 子流程 1：判断电梯是否可以同时响应其它用户指令的流程

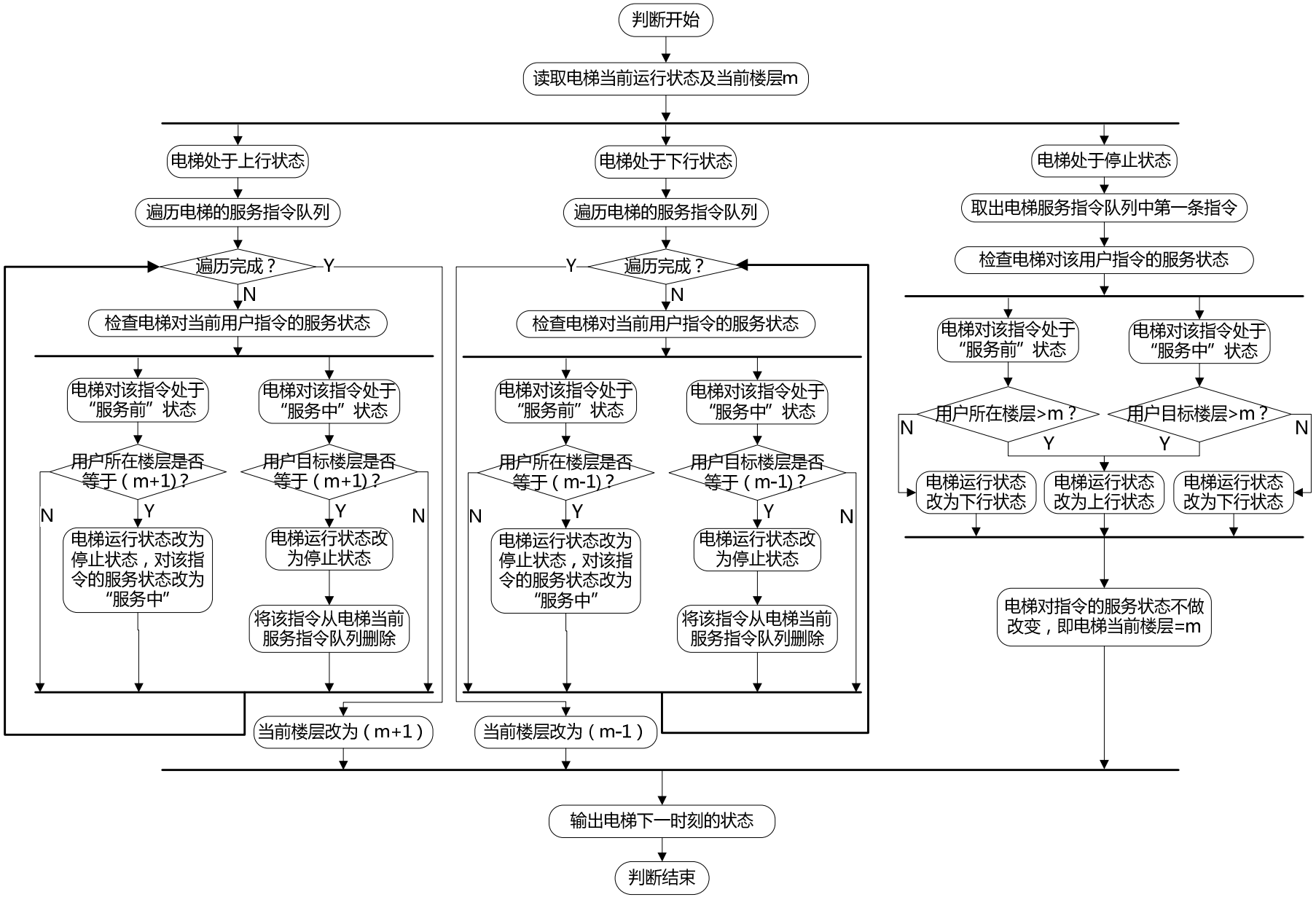

图 7-6 子流程 2：根据电梯当前状态确定下一时刻电梯状态的判断逻辑图

在进行程序设计时，需满足以下要求：

a) 采用外部调用的方式，调用实验 4 的 exe 程序以文本方式生成数据记录文件，即电梯用户请求指令数据文件；

b) 参考实验 5、6 中的相关函数，采用结构体数组的方式加载数据记录文件里的数据记录（即用户请求指令），加载后对用户请求指令按时间先后进行排序；

c) 电梯仿真功能以每隔 1 秒执行的功能函数方式实现（见下方函数示例），每调用一次函数相当于电梯运行了一个时间片长度，电梯根据当前电梯的运行状态及用户请求情况决定其下一时间周期运行状态（上行一层、下行一层或停在当前层），并将电梯状态修改到下一时间点的状态。以此类推，直到电梯仿真程序完成了所有用户请求的响应。请认真阅读实验 7 提供的《sleep 函数使用方法》参考资料；尝试思考如何设置 while 循环的条件保证电梯响应完所有用户指令后能够正常退出循环，并在实验报告中阐述你是如何设置 while 循环的退出条件的。

```
……
while(……)
{
    doyouwork(); //电梯仿真函数
    sleep(1000); //延迟 1 秒
}
……
```

d) 在每一时刻的电梯仿真函数中，将时刻序号、当前时刻的电梯状态和下一时刻的电梯状态打印输出到控制台，请设计合适的输出格式方便用户对仿真过程进行查看；

e) 构造以下几个内存容器用于支撑电梯仿真程序：

- 用户请求信息数组：用于存放从数据记录文件中读出的所有用户请求数据；
- 待响应指令队列：当当前仿真时刻等于用户发出请求的时间点时，将用户请求加入待响应指令队列；
- 电梯当前服务指令队列：当电梯响应一条用户请求时，将用户请求从待响应队列中取出，加入电梯当前服务指令队列；

f) 以单向链表的形式实现待响应指令队列及当前服务指令队列，用以下结构体声明实现两个链表：

- **用户请求信息结构体：**将用户请求信息封装为一个结构体，结构体中除了描述用户请求指令三元组的三个必要分量，还根据用户指令的特点，在结构体中增加了一个请求类型分量，当用户所在楼层大于用户目标楼层时，为“下行”类型的用户指令，反之为“上行”类型的用户指令，分别用“U”和“D”表示。用户指令结构体声明如下：

```
typedef struct UserCall{
        int user_floor;     //用户所在楼层
        int user_target;    //用户目标楼层
```

```
        int call_time;      //用户请求时刻
        char call_type;     //用户请求类型，´U´表示上行指令，´D´表示下行指令
    }USERCALL;
```

- **当前服务指令队列链表结点结构体：**采用无头结点的链表形式存储，其链表结点结构体声明如下：

```
    typedef struct ServeListNode{
        char serve_state;     //电梯服务状态
        USERCALL *user_call;//电梯当前响应用户指令时，指向指令数组的某一个元素
        struct ServeListNode *next_node;//存储下一个结点的地址
    }SERVELISTNODE
```

此外为了描述电梯的状态，需要构造一个电梯状态结构体，保存此电梯的各项状态信息，其中一个分量用于保存指向当前服务指令队列的头指针，其结构体声明如下：

```
    typedef struct elevatorstate{
        int current_floor;          //电梯当前所处楼层
        char run_state;       //电梯运行状态
        SERVELISTNODE *serve_list ;//电梯当前服务指令队列指针
    }ELEVATORSTATE;
```

- **待响应指令队列链表结点结构体：**采用带头结点的链表表示，其中链表的头结点保存队列中的结点个数及指向第一个和最后一个数据结点的指针，数据结点的数据域部分存放指向指令结构体的指针。链表数据结点结构体声明如下：

```
    typedef struct ResponseListNode{
        USERCALL *user_call;    //用户指令在指令数组中对应的序号
        struct ResponseListNode *next_node;//存储下一个结点的地址
    }RESPONSELISTNODE;
```

链表头结点数据结构体声明如下：

```
    typedef struct ResponseListHeadNode{
        int list_num;     //待响应用户指令链表中数据结点的个数
        RESPONSELISTNODE *head;//链表中第一个数据结点的指针
        RESPONSELISTNODE *tail;//链表中最后一个数据结点的指针
    }RESPONSELISTHEADNODE;
```

g) 参考实验5、6，以多模块的方式组织程序工程架构，对程序功能进行合理划分并设计合理的函数实现各个功能。

h) 请同学们参考实验5、6的设计文档大纲格式，编制电梯仿真程序的程序设计文档；

- 撰写实验报告：

请同学们在实验报告中阐述上一小节中提出的需要在实验报告中汇报的内容，并论述一下在设计实现实验 7 的过程中遇到的问题及解决方法，以及设计实现实验 7 过程中的心得体会。

结果提交

- 程序调试通过后，由授课教师课堂检查并记录成绩；
- 实验完毕后需提交程序工程源代码、release 版本程序和实验报告和程序设计说明书，以压缩包的形式提交给任课教师；
- 请使用本课程所要求的命名规范对压缩包及其内部文件、文件夹进行命名，详见《程序设计基础训练实验命名规范》；

成绩评定

- 采分点：
  - 程序是否独立调试通过并运行正常；
  - 是否掌握了外部程序调用的方法；
  - 是否掌握并理解了程序二进制存储方式和文本存储方式的区别；
  - 程序格式是否规范，程序是否易于阅读；
  - 实验报告内容是否详实、所反映出的学生对实验 6 开发的体会是否深刻。