lab2 --离散事件模拟

上机检查：

`2024/10/20` 和 `2024/10/27` 的晚上 6:30 到 9:00。地点在 `电三楼 406室` 。

实验的ddl设置为 `2024/10/27` 。

如果提前写完了的话也可以在 `2024/10/13` 来检查。

说明

- 本次实验需要在两个题目中完成一个，每个题目的基础部分均为 10 分。
- 每个题目均有两个选做的部分，每个选做部分都为 1 分。本实验上限为 12 分。
- 我们会有多次实验，你只需选择其中的一次实验写报告即可。实验报告的详细写法可以参考习题集的 P83的实习报告示例。 报告内容应具有可读性，前后应具有逻辑性。 报告包含的有关程序代码/执行结果的图片需要保证是真实的代码/执行结果。实验报告无额外分数，但如果发现报告中有客观问题或可读性过低等过于离谱的错误，将从实验总分中酌情扣分。
- 若发现有同学的报告或代码严重雷同，将取消对应实验成绩 。bb系统实验报告和代码提交入口将会在后期开放

题目一：银行业务管理系统

基础部分

【问题描述】

客户业务分为两种。第一种是申请从银行得到一笔资金，即取款或借款。第二种是向银行投入一笔资金，即存款或还款。银行有两个服务窗口，相应地有两个队列。客户到达银行后先排第一个队。处理每个客户业务时，如果属于第一种，且申请额超出银行现存资金总额而得不到满足，则立刻排入第二个队等候，直至满足时才离开银行；否则业务处理完后立刻离开银行。每接待完一个第二种业务的客户，则顺序检查和处理（如果可能）第二个队列中的客户，对能满足的申请者予以满足，不能满足者重新排到第二个队列的队尾。注意，在此检查过程中，一旦银行资金总额少于或等于刚才第一个队列中最后一个客户（第二种业务）被接待之前的数额，或者本次已将第二个队列检查或处理了一遍，就停止检查（因为此时已不可能还有能满足者）转而继续接待第一个队列的客户。任何时刻都只开一个窗口。假设检查不需要时间。营业时间结束时所有客户立即离开银行。

写一个上述银行业务的事件驱动模拟系统，通过模拟方法求出客户在银行内逗留的平均时间。

【基本要求】

利用动态存储结构实现模拟，即利用 C 语言的动态分配函数 malloc 和 free。

【测试数据】

一天营业开始时银行拥有的款额为 10000（元），营业时间为 600（分钟）。其他模拟参量自定，注意测定两种极端的情况：一是两个到达事件之间的间隔时间很短，而客户的交易时间很长，另一个恰好相反，设置两个到达事件的间隔时间很长，而客户的交易时间很短。

其他说明和要求：

- 初始时银行现存总资金为 `total`，营业时间从 0 到 `closetime`。事件在 `arrtime` 到达（第一个事件的 `arrtime` 为0），办理需要 `durtime` 的时间。办理的金额为 `amount`，其负值表示第一类业务，正值表示第二类业务。前一个事件到达后，经过 `interval` 个单位时间，后一个事件到达。`durtime`、`amount` 和 `interval` 为随机确定的，对于每个事件都可能不同，但我们可以指定两者的上下界。`total`，`closetime`，`durtime` 的上下界，`amount` 的上下界，`interval` 的上下界都由终端输入。
- 根据银行队列的排队情况，确定每个事件的离开时间 `leavetime`，用于计算客户在银行逗留的时间。初始置 `leavetime` 为 -1
- 在一天的开始，根据上面的规则确定一天内所有到达事件信息。每当时间到了其中一个事件的 `arrtime` 时，将该事件放到银行服务窗口队列里排队。
- 事件表 `eventlist` 的定义：`CustNode eventlist[MAX]`。`CustNode` 类型需要包括 `arrtime`，`durtime`，`leavetime`，`amount` 四个域。注意 `eventlist` 内的事件需要按照 `arrtime` 升序排列。
- 两个队列的定义不做要求，需要同学们自己思考应该包含哪些属性。

选做部分

- **选做一**：

  使用**循环队列**作为银行中两个排队队列的底层实现形式。要求如下：

  - 初始使用动态内存分配创建长度为 `CHUNKSIZE` 的空数组， `CHUNKSIZE` 的大小可由终端读入。
  - 维护已分配的数组长度、数组基址、队列元素个数、队头下标、队尾下标等属性。
  - 实现队列的基础功能：判断队空/队满、入队操作、出队操作、队列的创建和销毁操作。
  - 为了使队满时队列仍能正常工作，要求实现队列的**扩容**。每次队满时，数组长度增加 `CHUNCKSIZE` 。

- **选做二**：

  - 银行长期运营，每天刚开始运营时，都将当天会到达的所有事件加入到 `eventlist` 中。对于当天营业时间内没有处理完事务的客户，我们假定他们会在第二天的同一时间到达，并且 `durtime` 、 `amount` 等信息维持不变。（提示：每天营业时间结束只在 `eventlist` 中清空掉当天完成已经完成的事件，第二天到来时将新增添的事件插入到 `eventlist` 中）。
  - 给 `CustNode` 新添加一个域 `next` ，用以指示下一个非空闲的位置。银行营业的第一天，除了最后一个事件外，所有事件的 `next` 都指向下一个位置。注意每天运营结束时由于 `eventlist` 中部分事件被清除，可能会出现“碎片化”（即 `next` 不一定指向下一个位置）。
  - 要求新来的事件尽可能填充事件表中的“碎片”，并且维护事件表内按 `arrtime` **升序排列**的性质。
  - （此条**不作得分要求**）事实上，上面事件表的内存管理方式和操作系统里的内存管理方式有许多相似之处，有兴趣的同学可以搜索相关内容，也可以了解操作系统内存碎片化的危害和碎片的消除方式。

题目二：电梯模拟

【问题描述】

设计一个电梯模拟系统。这是一个离散的模拟程序，因为电梯系统是乘客和电梯等“活动体”构成的集合，虽然他们彼此交互作用，但他们的行为是基本独立的。在离散的模拟中，以模拟时钟决定每个活动体的动作发生的时刻和顺序，系统在某个模拟瞬间处理有待完成的各种事情，然后把模拟时钟推进到某个动作预定要发生的下一个时刻。

【基本要求】

（1）模拟某校五层教学楼的电梯系统。该楼有一个自动电梯，能在每层停留。五个楼层由下至上依次称为地下层、第一层、第二层、第三层和第四层，其中第一层是大楼的进出层，即是电梯的“本垒层”，电梯“空闲”时，将来到该层候命。

（2）乘客可随机地进出于任何层。对每个人来说，他有一个能容忍的最长等待时间，一旦等候电梯时间过长，他将放弃。

（3）模拟时钟从 0 开始，时间单位为 0.1 秒。人和电梯的各种动作均要耗费一定的时间单位（简记为 t），比如：

有人进出时，电梯每隔 $40t$ 测试一次，若无人进出，则关门；

关门和开门各需要 $20t$；

每个人进出电梯均需要 $25t$；

如果电梯在某层静止时间超过 $300t$，则驶回 1 层候命。

（4）按时序显示系统状态的变化过程：发生的全部人和电梯的动作序列。

【测试数据】

模拟时钟 Time 的初值为 0，终值可在 500～10000 范围内逐步增加。

【实现提示】

（1）楼层由下至上依次编号为 0，1，2，3，4。每层有要求 Up（上）和 Down（下）的两个按钮，对应 10 个变量 CallUp[0..4]和 CallDown[0..4]。电梯内 5 个目标层按钮对应变量 CallCar[0..4]。有人按下某个按钮时，相应的变量就置为 1，一旦要求满足后，电梯就把该变量清为 0。

（2）电梯处于三种状态之一：GoingUp（上行）、GoingDown（下行）和 Idle（停候）。如果电梯处于 Idle 状态且不在 1 层，则关门并驶回 1 层。在 1 层停候时，电梯是闭门候命。一

且收到往另一层的命令，就转入 GoingUp 或 GoingDown 状态，执行相应的操作。

(3) 用变量 Time 表示模拟时钟，初值为 0，时间单位(t)为 0.1 秒。其他重要的变量有：

Floor——电梯的当前位置(楼层)；

D1——值为 0，除非人们正在进入和离开电梯；

D2——值为 0，如果电梯已经在某层停候 300t 以上；

D3——值为 0，除非电梯门正开着又无人进出电梯；

State——电梯的当前状态(GoingUp，GoingDown，Idle)。

系统初始时，Floor=1，D1=D2=D3=0，State=Idle。

(4) 每个人从进入系统到离开称为该人在系统中的存在周期。在此周期内，他有 6 种可能发生的动作：

M1. [进入系统，为下一人的出现作准备]产生以下数值：

InFloor——该人进入哪层楼；

OutFloor——他要去哪层楼；

GiveupTime——他能容忍的等候时间；

InterTime——下一人出现的时间间隔，据此系统预置下一人进入系统的时刻。

M2. [按电钮并等候]此时应对以下不同情况作不同的处理：

① Floor=InFloor 且电梯的下一个活动是 E6(电梯在本层，但正在关门)；

② Floor=InFloor 且 D3≠0(电梯在本层，正有人进出)；

③ 其他情况，可能 D2=0 或电梯处于活动 E1(在 1 层停候)。

M3. [进入排队]在等候队列 Queue[InFloor]末尾插入该人，并预置在 GiveupTime 个 t 之后，他若仍在队列中将实施动作 M4。

M4. [放弃]如果 Floor≠InFloor 或 D1=0，则从 Queue[InFloor]和系统删除该人。如果 Floor=InFloor 且 D1≠0，他就继续等候(他知道马上就可进入电梯)。

M5. [进入电梯]从 Queue[InFloor]删除该人，并把他插入到 Elevator(电梯)栈中。置 CallCar[OutFloor]为 1。

M6. [离去]从 Elevator 和系统删除该人。

(5) 电梯的活动有 9 种：

E1. [在 1 层停候]若有人按下一个按钮，则调用 Controler 将电梯转入活动 E3 或 E6。

E2. [要改变状态?]如果电梯处于 GoingUp(或 GoingDown)状态，但该方向的楼层却无人等待，则要看反方向楼层是否有人等候，而决定置 State 为 GoingDown(或 GoingUp)还是 Idle。

E3. [开门]置 D1 和 D2 为非 0 值，预置 300 个 t 后启动活动 E9 和 76 个 t 后启动 E5，然后预置 20 个 t 后转到 E4。

E4. [让人出入]如果 Elevator 不空且有人的 OutFloor=Floor，则按进入的倒序每隔 25 个 t 让这类人立即转到他们的动作 M6。Elevator 中不再有要离开的人时，如果 Queue[Floor]不空，则以 25 个 t 的速度让他们依次转到 M5。Queue[Floor]空时，置 D1 为 0，D3

≠0，而且等候某个其他活动的到来。

E5. [关门]每隔 40 个 t 检查 D1，直到是 D1＝0(若 D1≠0，则仍有人出入)。置 D3 为 0 并预置电梯再 20 个 t 后启动活动 E6(再关门期间，若有人到来，则如 M2 所述，门再次打开)。

E6. [准备移动]置 CallCar[Floor]为 0，而且若 State≠GoingDown，则置 CallUp[Floor]为 0；若 State≠GoingUp，则置 CallDown[Floor]为 0。调用 Controler 函数。

如果 State＝Idle，则即使已经执行了 Controler，也转到 E1。否则，如果 D2≠0，则取消电梯活动 E9。最后，如果 State＝GoingUp，则预置 15 个 t 后(电梯加速)转到 E7；如果 State＝GoingDown，则预置 15 个 t 后(电梯加速)转到 E8。

E7. [上升一层]置 Floor 加 1 并等候 51 个 t。如果现在 CallCar[Floor]＝1 或 CallUp[Floor]＝1，或者如果((Floor＝1 或 CallDown[Floor]＝1)且 CallUp[j]＝CallDown[j]＝ CallCar[j]＝0对于所有 j>Floor)，则预置 14 个 t 后(减速)转到 E2；否则重复 E7。

E8. [下降一层]除了方向相反之外，与 E7 类似，但那里的 51 和 14 个 t，此时分别改为 61 和 23 个 t(电梯下降比上升慢)。

E9. [置不活动指示器]置 D2 为 0 并调用 Controler 函数(E9 是由 E3 预置的，但几乎总是被 E6 取消了)。

(6) 当电梯须对下一个方向作出判定时，便在若干临界时刻调用 Controler 函数。该函数有以下要点：

C1. [需要判断?]若 State≠Idle，则返回。

C2. [应该开门?]如果电梯处于 E1 且 CallUp[1]，CallDown[1]或 CallCar[1]非 0，则预置 20 个 t 后启动 E3，并返回。

C3. [有按钮按下?]找最小的 j≠Floor，使得 CallUp[j]，CallDown[j]或 CallCar[j]非 0，并转到 C4。但如果不存在这样的 j，那么，如果 Controler 正为 E6 所调用，则置 j 为 1，否则返回。

C4. [置 State]如果 Floor＞j，则置 State 为 GoingDown；如果 Floor＜j，则置 State 为 GoingUp。

C5. [电梯静止?]如果电梯处于 E1 而且 j≠1，则预置 20 个 t 后启动 E6。返回。

(7) 由上可见，关键是按时序管理系统中所有乘客和电梯的动作设计合适的数据结构。

选做部分：

- 选做一：

实现简易的多梯系统。在有乘客按下按钮时按照如下规则，将任务委派给最合适的电梯：

- 优先委派给所有静止的电梯中离乘客当前楼层最近的那一个。
- 如果没有静止的电梯，委派给和乘客目标方向一致的、并且之后会经过乘客当前楼层的电梯。
- 如果所有电梯里都没有满足前两个条件之一的，那么随机委派给其中一个电梯。

（你需要自行添加状态，实现的方式不做要求）

- **选做二**：

  题目的基础部分假定乘客会随机从一个楼层去往另一个楼层，但这并不符合实际场景。我们假定这栋楼里没有楼梯，所有人都只能通过电梯在层间移动。具体要求如下：

  - 假定模拟时钟为 0 时系统内没有乘客，每隔一段时间随机生成一位乘客从 1 层前往随机的目标层。楼内的人达到某个上限后，禁止乘客进入楼层。
  - 乘客在目标层里经过随机的一段时间后，会从该层出发，随机再选一层作为新的目标层。当乘客选择目标层为一层并到达一层后，将乘客从系统中删去。不考虑基础部分的 M4 状态，即乘客等待的容忍时间 `Giveuptime` 为无限长。
  - 到了预定的时间后，楼里内清场，所有的人都会选择一层作为目标层，离开这栋楼。楼内的人都离开后，模拟终止。

  （你需要自行添加状态，实现的方式不做要求）