8;1+2+3+4;88-1*5;1024/4*8;1024/(4*8);(20+2)*(6/2);

3-3-3;8/(9-9);2*(6+2*(3+6*(6+6)));(((6+6)*6+3)*2+6)*2;

【实现提示】

(1) 设置运算符栈和运算数栈辅助分析算符优先关系。

(2) 在读入表达式的字符序列的同时,完成运算符和运算数(整数)的识别处理,以及相应的运算。

(3) 在识别出运算数的同时,要将其字符序列形式转换成整数形式。

(4) 在程序的适当位置输出运算符栈、运算数栈、输入字符和主要操作的内容。

【选作内容】

(1) 扩充运算符集,如增加乘方、单目减、赋值等运算。

(2) 运算量可以是变量。

(3) 运算量可以是实数类型。

(4) 计算器的功能和仿真界面。

◆2.6⑤ 银行业务模拟

【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。

【实现提示】

事件有两类:到达银行和离开银行。初始时银行现存资金总额为 total。开始营业后的第一个事件是客户到达,营业时间从 0 到 closetime。到达事件发生时随机地设置此客户

的交易时间和距下一到达事件之间的时间间隔。每个客户要办理的款额也是随机确定的，用负值和正值分别表示第一类和第二类业务。变量 total，closetime 以及上述两个随机量的上下界均交互地从终端读入，作为模拟参数。

两个队列和一个事件表均要用动态存储结构实现。注意弄清应该在什么条件下设置离开事件，以及第二个队列用怎样的存储结构实现时可以获得较高的效率。注意：事件表是按时间顺序有序的。

【选作内容】

自己实现动态数据类型。例如对于客户结点，定义 pool 为

```
CustNode pool[MAX];
// 结构类型 CustNode 含四个域：arrtime，durtime，amount，next
```

或者定义四个同样长的，以上述域名为名字的数组。初始时，将所有分量的 next 域链接起来，形成一个静态链栈，设置一个栈顶元素下标指示量 top，top＝0 表示栈空。动态存储分配函数可以取名为 myMalloc，其作用是出栈，将栈顶元素的下标返回。若返回的值为 0，则表示无空间可分配。归还函数可取名为 myFree，其作用是把该分量入栈。用 FORTRAN 和 BASIC 等语言实现时只能如此地自行组织。

2.7④ 航空客运订票系统

【问题描述】

航空客运订票的业务活动包括：查询航线、客票预订和办理退票等。试设计一个航空客运订票系统，以使上述业务可以借助计算机来完成。

【基本要求】

(1) 每条航线所涉及的信息有：终点站名、航班号、飞机号、飞行周日(星期几)、乘员定额、余票量、已订票的客户名单(包括姓名、订票量、舱位等级 1，2 或 3)以及等候替补的客户名单(包括姓名、所需票量)；

(2) 作为示意系统，全部数据可以只放在内存中；

(3) 系统能实现的操作和功能如下：

① 查询航线：根据旅客提出的终点站名输出下列信息：航班号、飞机号、星期几飞行，最近一天航班的日期和余票额；

② 承办订票业务：根据客户提出的要求(航班号、订票数额)查询该航班票额情况，若尚有余票，则为客户办理订票手续，输出座位号；若已满员或余票额少于订票额，则需重新询问客户要求。若需要，可登记排队候补；

③ 承办退票业务：根据客户提供的情况(日期、航班)，为客户办理退票手续，然后查询该航班是否有人排队候补，首先询问排在第一的客户，若所退票额能满足他的要求，则为他办理订票手续，否则依次询问其他排队候补的客户。

【测试数据】

由读者指定。

【实现提示】

两个客户名单可分别由线性表和队列实现。为查找方便，已订票客户的线性表应按客户姓名有序，并且，为插入和删除方便，应以链表作存储结构。由于预约人数无法预计，队列也应以链表作存储结构。整个系统需汇总各条航线的情况登录在一张线性表上，由于航线基本不变，可采用顺序存储结构，并按航班有序或按终点站名有序。每条航线是这张表上的一个记录，包含上述 8 个域、其中乘员名单域为指向乘员名单链表的头指针，等候替补的客户名单域为分别指向队头和队尾的指针。

【选作内容】

当客户订票要求不能满足时，系统可向客户提供到达同一目的地的其他航线情况。读者还可充分发挥自己的想象力，增加你的系统的功能和其他服务项目。

2.8⑤ 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

旦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。

【选作内容】

(1) 增加电梯数量，模拟多梯系统。

(2) 某高校的一座 30 层住宅楼有三部自动电梯，每梯最多载客 15 人。大楼每层 8 户，每户平均 3.5 人，每天早晨平均每户有 3 人必须在 7 时之前离开大楼去上班或上学。模拟该电梯系统，并分析分别在一梯、二梯和三梯运行情况下，下楼高峰期间各层的住户应提前多少时间候梯下楼？研究多梯运行最佳策略。