实习报告：电梯调度模拟

问题简述：模拟某校五层楼层的电梯，有地下一层、地上一到四楼。乘客随机生成在任意楼层，目的地也是任意楼层，乘客有一个最大容忍时间，超过即放弃。每个动作都要花费一定时间，如进出电梯、开关电梯门等。当电梯空闲达到一定时间后，回到“本垒层”一楼。

关键代码简介：

1，电梯

本实验中，电梯的活动主要由状态机实现。其中电梯的状态有三种：上行、下行、等待，电梯的运动有开门中、门已打开、关门中、门已关上、乘客进出、电梯运动等状态，通过这两个状态机完整描述了电梯的所有运动。下面简单用语言描述每个地方要实现什么，具体代码参见附件。

```
Status ChangeMovement(Elevator *elevator)
{
    switch (elevator->movement)
    {
    case Opening:
        下一状态为 Opened

    case Opened:
       如果本层电梯栈不空
        如果有人要下楼
        {
            ClientOut(elevator);
            elevator->movementTim = PeopleInAndOutTIM;
        }
         乘客都出栈后/没有要出去的 进行下一个判断
         如果有人要在本楼层进入电梯并且电梯未满
                ClientIn(elevator);
                elevator->movementTim = PeopleInAndOutTIM;
            如果没人等待那么进入下一个状态：closing
                elevator->movement = Closing;
                printf("电梯门正在关闭\n");
                elevator->movementTim = doorTIM;
            }
        }
        break;

    case Closing:
        elevator->movement = Closed;
        break;

    case Closed:
        // 如果有请求,设置电梯状态为正在加速
```

```
        //当前方向没有请求，改变运动状态
        // 如果整个楼没有请求并且电梯内没有人那就等待
        break;

    case Moving:

            // 如果电梯正在上楼，就获取楼上的请求
            // 如果电梯正在下楼就获取楼下的请求
            // 如果应该继续往下那么更新当前状态的时间并下楼
            // 如果应该继续往上那么更新当前状态的时间并上楼

    case Waiting:
        给 back 计数器计时，检查是否到达 300s

    case Accelerating:
        elevator->movement = Moving;
        break;

    case SlowingDown:
       // 如果是正常减速则开门
       // 如果是回到本垒层则等待
        break;
}
```

2，时间设置

本组实验在开发过程中并未严格按照原题目时间。几个基本设置如下：

开关门：2s

乘客进出：2s

电梯速度：2s/层，为了模拟现实电梯，当电梯在起步/停止时，速度减为 3s/层

```
#define door_time 2  // 开关门所需时间
#define people_time 2  // 乘客进出时间
#define wait_time 300  // 最长等待时间
#define move_time 2  // 电梯运动
#define accele_time 3   // 加速时间
#define slow_time 3  // 减速时间
```

3，乘客的设置

乘客的生成是随机的，为了保证不会很密集或稀疏，每个乘客生成后有一个随机冷却时间。

乘客的放弃时间为 20-50s 随机生成

当乘客的进入/离开楼层相同，生成失败。

```
Status Newclient()
```

```
{
    Client NewClient;
    NewClient.Infloor = rand() % 5;
    NewClient.OutFloor = rand() % 5;
    InterTime = rand() % 30;   // 下一个乘客到达时间间隔
    if (NewClient.Infloor == NewClient.OutFloor)
        return False;
    NewClient.ClientID = ++ID;
    NewClient.GiveupTime = 20 + rand() % 30;
    if (NewClient.Infloor < NewClient.OutFloor)
    {
        Enqueue(&Queue[0][NewClient.Infloor], NewClient);
        printf("%d 号乘客在 %d 楼等待上楼\n", NewClient.ClientID,
NewClient.Infloor);
        Up[NewClient.Infloor] = 1;
    }
    else
    {
        Enqueue(&Queue[1][NewClient.Infloor], NewClient);
        printf("%d 号乘客在 %d 楼等待下楼\n", NewClient.ClientID,
NewClient.Infloor);
        Down[NewClient.Infloor] = 1;
    }
}
```

4，回归本垒层

设置一个计数器，每个 clock 后检查一次，如果电梯处于 Free 状态就加 1，当计数器到达等待上限执行回归本垒层的操作。当电梯不处于 Free 状态时将计数器清零。

```
Status BackToBase(Elevator *elevator)
{
    if(elevator->State == Free)
        elevator->back++;
    else
        elevator->back = 0;
    if(elevator->back < 300)
        return OK;

    if (elevator->curpos == 1)
    {
        elevator->movement = Waiting;
        printf("电梯在本垒层\n");
        elevator->State = Free;
    }
```

```
    else if (elevator->curpos < 1)
    {
        printf("电梯正在回归本垒层\n");
        elevator->movement = Accelerating;
        elevator->movementTim = accelerateTIM;
        elevator->State = GoingUp;
    }
    else
    {
        printf("电梯正在回归本垒层\n");
        elevator->movement = Accelerating;
        elevator->movementTim = accelerateTIM;
        elevator->State = GoingDown;
    }
    return OK;
}
```

5，存储结构体

整个运行过程中还利用到了几个简单的结构体：乘客栈、等待队列、申请数组等。对于这些结构体的操作都很基础，出栈、入栈、出队列、入队列、检查是否为空等，分别模拟了乘客出入电梯等动作。在这部分体会到了本学期数据结构课程学习的结构体具有很强大的实用性。

6，其他

在实际电梯运行中，有可能出现电梯在关门时有人要上电梯的情况，因此在每个 clock，如果当前如果是 closing 状态，检查是否有同向请求。同样这里也实现了电梯的变向：当电梯把当前方向上最后一个乘客送走后，先进入 closing 状态，此时若楼下有请求，改变运行方向。

```
        if (elevator.movement == Closing)
        {
            // 如果当前电梯不为空,如果有相同行进方向的乘客则入栈
            if (elevator.Passnum && elevator.Passnum < MaxNum)
            {
                // 如果当前电梯向上,有向上的乘客则切换到开门
                if (elevator.State == GoingUp)
                {
                    if (Up[elevator.curpos])
                    {
                        elevator.movement = Opening;
                        elevator.movementTim = doorTIM;
                    }
                }
                // 如果当前电梯向下,有向下的乘客则切换到开门
                else if (elevator.State == GoingDown)
```

```
                {
                    if (Down[elevator.curpos])
                    {
                        elevator.movement = Opening;
                        elevator.movementTim = doorTIM;
                    }
                }
            }
            // 如果当前电梯为空
            else if (!elevator.Passnum)
            {
                // 只要有请求则都切换到开门状态
                if (Up[elevator.curpos] || Down[elevator.curpos])
                {
                    elevator.movement = Opening;
                    elevator.movementTim = doorTIM;
                }
            }
        }
```

此外，为了更加生动形象地演示电梯的运行过程，本组尝试用 Eazy X 绘制了简易的动画，这部分比较像 html 网页编辑的过程。同样在这里用到 Sleep 函数，模拟了现实时间，电梯的时间和现实时间的比例为 1:0.3，通过修改 Sleep 函数的参数可以修改比例，这里是为了便于演示。具体代码如下

```
void makegraph(Elevator *elevator)
{
    int i,j;
    setbkcolor(WHITE);
    cleardevice();
    setlinecolor(BLACK);
    rectangle(150, 50, 350, 125);
    rectangle(150, 125, 350, 200);
    rectangle(150, 200, 350, 275);
    rectangle(150, 275, 350, 350);
    rectangle(150, 350, 350, 425);

    rectangle(350, 50, 450, 125);
    rectangle(350, 125, 450, 200);
    rectangle(350, 200, 450, 275);
    rectangle(350, 275, 450, 350);
    rectangle(350, 350, 450, 425);

    rectangle(550, 50, 750, 125);
```

```
    rectangle(550, 125, 750, 200);
    rectangle(550, 200, 750, 275);
    rectangle(550, 275, 750, 350);
    rectangle(550, 350, 750, 425);

    settextstyle(25, 0, "楷体");//设置字体高度，宽度，字型
    setbkmode(TRANSPARENT);//设置字体背景透明，默认不透明
    settextcolor(BLACK);
    char c;
    for(i=0;i<4;i++)
    {
        c = '4'- i;
        int y_pos = i * 75 + 70;
        outtextxy(70,y_pos,c);
    }
    char arr_0[] = "-1";
    outtextxy(70,370,arr_0);


    char arr_1[] = "电梯门与乘客";
    outtextxy(210,10, arr_1);
    char arr_2[] = "等待中的乘客";
    outtextxy(580,10, arr_2);

    //电梯的位置
    IMAGE door_img;
    int y_ele = 350 - elevator->curpos * 75;
    if(elevator->movement == Closing || elevator->movement == Opened)
        loadimage(&door_img,_T("opened.png"), 100, 75);
    else
        loadimage(&door_img,_T("closed.png"), 100, 75);
    putimage(350, y_ele, &door_img, SRCINVERT);


    //电梯中的乘客
    IMAGE people;
    loadimage(&people,_T("people.png"),50,50);
    for(i=0;i<5;i++)
    {
        int num = elevator->Stack[i].top;
        while(num > -1)
        {
            int x_people = 300 - 50 * num;
            putimage(x_people, y_ele+25 , &people, SRCINVERT);
```

```
            num --;
        }
    }


    //等待的乘客
    for(i=0;i<2;i++)
    {
        for(j=0;j<5;j++)
        {
            int d = Queue[i][j].WaitNum;
            int people_pos = 375 -  Queue[i][j].rear->passenger.Infloor
* 75;
            if(d)
            {
                putimage(570,people_pos,&people,SRCINVERT);
                d--;
            }
        }
    }

    Sleep(300);
}
```

实习报告：稀疏矩阵运算器

问题简述：实现稀疏矩阵的加减法，乘法，求转置，求行列式，求逆等一系列运算。

关键代码简介
本次代码为了丰富 UI 界面，实现更好的使用体验，将程序写成了网页的形式。
在代码简介中，css 部分不作解释，主要介绍 html 的设计以及 js 的核心逻辑。

1. html 的界面输入与输出部分
本实验中，第一步是选择功能，第二步是输入功能对应的稀疏矩阵，第三步是点击“solve”并输出结果。RESET 按钮用于选错功能的情况，是初始化界面的按钮。
根据功能不同，给出的矩阵框数也不同，加法，减法乘法为二元运算，需要给出两个输入框，而其余为一元运算，只需要给出一个输入框。

```
<div class="wrapper">
    <button class="but" onclick="init_table(2);opcode=1;">A+B</button>
    <button class="but" onclick="init_table(2);opcode=2;">A-B</button>
    <button class="but" onclick="init_table(2);opcode=3;mul_flag=getMatricesSize();">A*B</button>
    <button class="but" onclick="init_table(1);opcode=4;">A<sup>T</sup></button>
    <button class="but" onclick="init_table(1);opcode=5;inv_flag=getMatrixSize();">A<sup>-1</sup></button>
    <button class="but" onclick="init_table(1);opcode=6;inv_flag=getMatrixSize();">|A|</button>
    <button class="but" onclick="reset()" style="background-color: #fa6666;">RESET</button>
    <button class="but" onclick="display(calc());"  style="background-color: #60cd68;"">SOLVE</button>
</div>
```

2. Js 部分

1）控制信号设计与 RESET 函数

如注释所示，下面的信号用于控制整个程序的运行，opcode 采取了类似于选择器的想法，根据不同的功能设置不同的 opcode，随后进行计算。RESET 函数则用于恢复初始网页与控制信号的初始值。

```
var wrapper = document.getElementById("table_wrapper");
var flag = 0;//区分是否点击了功能的信号
var opcode = 0;//区分不同的功能选择信号
var mul_flag = 1;//标志是否选用的是乘法功能
var inv_flag = 1;//标志是否选用的是求逆或转置功能

function reset(){
    flag = 0;
    mul_flag = 1;
    inv_flag = 1;
    wrapper.innerHTML = '';
}
```

2）输入表格的删除与增添设计

在输入表格中，设计了“增加”与“删除”按钮用来增加或者减少行数和列数，满足稀疏矩阵输入数据的实时可变性，而不需要开始填写需要写入几个元素，增强了使用过程中的便捷。具体实现采用了创建一行表格，增添就是新加一行，删除就是去除一行的方法，以行为单位进行操作，即“appendrow”和“removerow”函数。

```
var table = document.createElement('table');
table.appendChild(tbody);
wrapper.appendChild(table);
var p = document.createElement('p');
p.classList.add('opration');
p.addEventListener('click', function(){
    appendRow(k);
});
p.innerHTML= '添加';
wrapper.appendChild(p);

var p = document.createElement('p');
p.classList.add('opration');
p.addEventListener('click', function(){
    removeRow(k);
});
p.innerHTML= '删除';
wrapper.appendChild(p);

appendRow(k);
appendRow(k);
appendRow(k);
```

3）代码选择逻辑部分的实现

Opcode1，2，3 的情况为二元运算，需要获取两个表格。Opcode>=4 的情况为一元运算，只需要获取一个表格即可。然后使用 switch-case 的形式实现对于不同状态的选取与操作。

```
//这是功能区分函数
function calc(){
    var arr1 = getArray(0);
    if(opcode<4)
        var arr2 = getArray(1);
    var arr3 = new Array();
    switch(opcode){
        case 1:
            arr3 = add(arr1, arr2);
            break;
        case 2:
            arr3 = sub(arr1, arr2);
            break;
        case 3:
            if(mul_flag==0){
                arr3 = mul(arr1, arr2);
                break;
            }else if(mul_flag==-1){
                alert("矩阵A的列数不等于矩阵B的行数");
                break;
            }

        case 4:
            arr3 = transpose(arr1);
            break;
        case 5:
```

4）关于乘法部分的条件限制的说明

矩阵乘法存在限制，需要第一个矩阵的列数等于第二个矩阵的行数，两个矩阵才可以相乘。因此先在网页上弹出输入框，输入矩阵 A,B 的行数与列数在进行运算。

```
function getMatricesSize(){
    window.alert("请先输入矩阵A的行数和列数");
    mul_A_row = Number(window.prompt("请输入矩阵A的行数"));
    mul_A_col = Number(window.prompt("请输入矩阵A的列数"));
    window.alert("请先输入矩阵B的行数和列数");
    mul_B_row = Number(window.prompt("请输入矩阵B的行数"));
    mul_B_col = Number(window.prompt("请输入矩阵B的列数"));
    if(mul_A_col != mul_B_row){
        return -1;
    }else{
        return 0;
```

5）将表格转化为三元组结构存储的函数

如下如所示，l 是表的长度，在每一个 arr【i】中，都有三个元素 arr[i][2],arr[i][1],,arr[i][0],用来记录数据，即将获取到的表格转化为一个每一个元素都是三元组的数组，即一个类似于二元数组的对象。

```
function getArray(n){
    var tab = document.getElementsByTagName("tbody");
    var l = tab[n].childNodes.length;
    var arr = new Array(l-1);
    for(let i = 1; i < l; i++){
        arr[i-1] = new Array(3);
        for(let j = 1; j <= 3; j++){
            arr[i-1][j-1] = Number(tab[n].childNodes[i].childNodes[j].childNodes[0].value);
        }
    }
    return arr;
}
```