电梯实习报告

题目：设计一个电梯模拟系统，模拟某校五层教学楼的电梯系统

班级：02

成员：张磊（2017K8009922027） 赵鑫浩（2017K8009922032）姜小平（2017K8009922014）

完成日期：2019.06.29

一. 需求分析

1. 楼层数量为 5 层，包含一层地下层（0 层），且地上 1 层为本垒层；

二. 概要设计

1. 定义“用户”类型：

```
typedef struct Person{
    int PersonID;
    int InFloor, OutFloor;
    int GiveUpTime, InterTime,AppearTime;
}Person;
int personnum = 0;
Person person[MaxPerson];
```

基本操作：输入用户的 ID，起始楼层，目的楼层，出现时间，最长忍受时间；

涉及函数：input();

2. 定义“等候队列”类型

```
typedef struct FloorPersonIn{
    int PersonID;
    int InFloor, OutFloor;
    int AppearTime, GiveUpTime;
    struct FloorPersonIn *next;
}FloorPersonIn;
FloorPersonIn personin[5];
```

基本操作：按照用户出现的时间先后顺序，将用户插入到对应起始楼层的等候队列中；

剔除超过忍受时间的等候用户；、

涉及函数：AddQueue();

3. 定义“等候数组”类型

```
int FloorPersonOut[5][MaxPerson+1];
int outpos[5];  //电梯中去第i层的人数
```

基本操作：将去往目的楼层的用户放入等候数组，并用 outpos 指示去往每层楼的人数；

涉及函数：PeopleIn();

4. 定义“活动链表”类型

```
typedef struct Activity{
        int time;
        void(*fn)(void);
        struct Activity* next;
}Activity;
```

基本操作：通过判断，将电梯即将进行的活动按时间顺序插入到活动链表中等待执行；

涉及函数：AddAct(int time, void(*fn)(void));

5. 本程序包含五个模块：

1). 主程序模块：

```
int main(void)
{
    //  Init();      //初始化
        Input();     //输入起始用户信息
        Clock();     //模拟时钟运行，并随时间调用相关函数
        return 0;
}
```

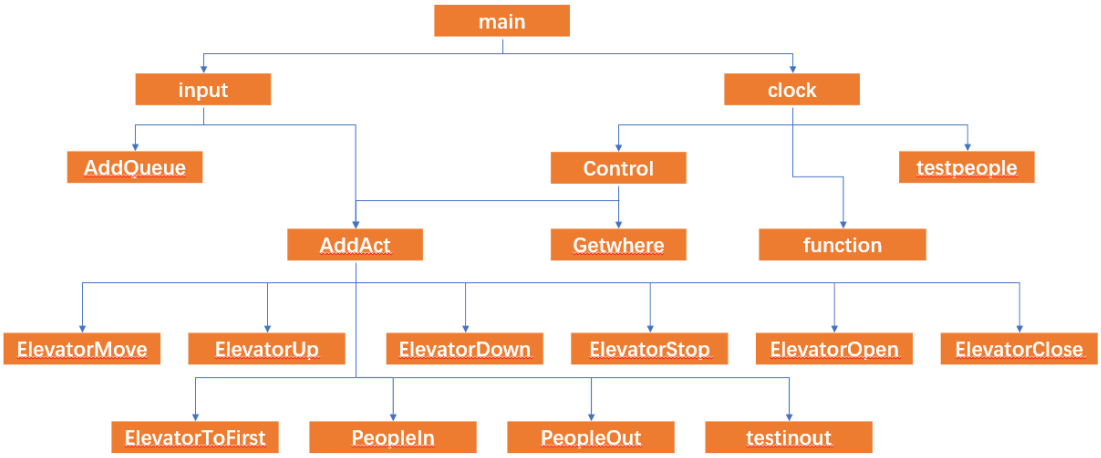
2). 输入模块——输入用户信息

3). 时钟模块——模拟时钟运行

4). 电梯控制模块——控制电梯运行

5). 用户控制模块——控制用户状态

各模块调用关系如下：



三. 详细设计

1. 主程序需要的全程量

```
// 时间
#define DoorTestTime 40         //有人进出测试时间
#define InOutDoorTime 20        //电梯开门关门时间
#define InOutPersonTime 25      //人进出电梯时间
#define OverTime 300            //电梯停候超时时间
#define UpAccelerateTime 15     //电梯上升的加速时间
#define UpTime 51               //电梯上升的匀速时间
#define UpDecelerateTime 14     //电梯上升的减速时间
#define DownAccelerateTime 15   //电梯下降的加速时间
#define DownTime 61             //电梯下降的匀速时间
#define DownDecelerateTime 23   //电梯下降的减速时间
#define MaxTime 10000           //最长时间

//电梯状态
#define GoingUp 1       //匀速上升
#define GoingDown 2     //匀速下降
#define SpeedUp 3       //加速上升
#define SpeedDown 4     //加速下降
#define SlowUp 5        //减速上升
#define SlowDown 6      //减速下降
#define Idle 7          //空闲
#define Stop 8          //停止且已关门
#define DoorOpen 9      //停止且门已打开
#define DoorOpening 10
#define DoorClosing 11


#define MaxPerson 50
#define MaxFloor 5
#define BaseFloor 1
```

2. 对于对象用户和电梯所需要的数据结构和参数

```
//输入
typedef struct Person{
    int PersonID;
    int InFloor, OutFloor;
    int GiveUpTime, InterTime,AppearTime;
}Person;
int personnum = 0;
Person person[MaxPerson];

//每一层的进入链表
typedef struct FloorPersonIn{
    int PersonID;
    int InFloor, OutFloor;
    int AppearTime, GiveUpTime;
    struct FloorPersonIn *next;
}FloorPersonIn;
FloorPersonIn personin[5];

//电梯中的人
int FloorPersonOut[5][MaxPerson+1];
int outpos[5];  //电梯中去第i层的人数


//电梯中的人
int FloorPersonOut[5][MaxPerson+1];
int outpos[5];  //电梯中去第i层的人数

//Activity
typedef struct Activity{
        int time;
        void(*fn)(void);
        struct Activity* next;
}Activity;

Activity activity={0,NULL,NULL};

int Time = 0;
//电梯
int CallUp[MaxFloor];
int CallDown[MaxFloor];
int CallCar[MaxFloor];
int Floor = BaseFloor;
int State = Idle;
int D1=0, D2=0, D3=0;
```

3. 需要调用的一些函数

```
int AddQueue(int infloor, Person personi);
void AddAct(int time, void(*fn)(void));
void TestPeople();
void Input(void);
void testinout(void);
void ElevatorClose(void);
void ElevatorOpen(void);
void PeopleOut(void);
void P2eopleIn(void);
int GetWhere(void);
void tofirst(void);
void ElevatorStop(void);
void ElevatorUp(void);
void ElevatorDown(void);
void ElevatorMove(void);
void Controler(void);
```

4. 主程序模块

```
int main(void){
//  Init();
    Input();
    Clock();
    return 0;
}
```

5. 输入部分：（将输入转化为数据结构并将用户加入电梯该层的等待链表）

```
void Input(void){
    int infloor,outfloor,giveuptime,intertime;

    while(1){//依次输入用户的起始楼层，目标楼层，容忍时间和到来时间
        printf("请输入用户的起始楼层:");
        scanf("%d", &infloor);
        printf("请输入用户的目标的楼层:");
        scanf("%d", &outfloor);
        printf("请输入用户的最长容忍时间:");
        scanf("%d", &giveuptime);
        printf("请输入下一个用户的到来时间:");
        scanf("%d", &intertime);
//      if(infloor==0 && outfloor==0 && giveuptime==0 && intertime==0){
//           printf("Input Finished.\n");
//            return;
//        }
        if(!(infloor<0||infloor>MaxFloor-1||outfloor<0||outfloor>MaxFloor-1)&&(infloor!=outfloor))
            break;//对错误输入的处理
        printf("Wrong Input.\n");
        return;
    }

    personnum++;//将输入赋值给数据结构
    person[personnum].PersonID = personnum;
    person[personnum].InFloor = infloor;
    person[personnum].OutFloor = outfloor;
    person[personnum].AppearTime = Time ;
    person[personnum].GiveUpTime = giveuptime + Time;
    person[personnum].InterTime = intertime;

    if(outfloor>infloor)    CallUp[infloor] = 1;//给电梯信号
    else                    CallDown[infloor] = 1;

    AddQueue(infloor, person[personnum]);
    AddAct(intertime, Input);

    return;
}
```

6. 时间模块（每秒检测等待队列和电梯的状态）

```
void Clock(void){//模拟时间
    while(1){
        if(Time>MaxTime){//超出约定时间
            return;
        }
        TestPeople();//检查链表中人的状态
        Controler();
        struct Activity* p = activity.next;
        if(p==NULL){//若活动链表为空，直接结束
            Time=MaxTime;
        }
        if(p&&Time>=p->time){//若不为空，且到达时间点，执行p
            activity.next=p->next;
            p->fn();
            free(p);
        }
        Time++;
    }
}
```

7. 测试模块（通过链表的相关操作实现对超过容忍时间的用户的删除操作）

```
void TestPeople(){
    int i;
    for(i=0; i<MaxFloor; i++){
        FloorPersonIn *pre = &personin[i];
        FloorPersonIn *p = personin[i].next;
        if(p == NULL)
            continue;
        while(p != NULL){
            if(p->GiveUpTime <= Time){
                if(Floor==i && (State>=Idle) )
                    break;
                pre->next = p->next;
                printf("Time: %d\t",Time);
                printf("用户%d放弃了等待.\n", p->PersonID);
                free(p);
                p = pre->next;
                continue;
            }
            pre = p;
            p = p->next;
        }
    }
    return;
}
```

8. 控制模块（控制电梯状态，处理电梯需要转变状态的情况）

```
void Controler(void){//电梯状态控制
    if(State==Idle||State==Stop){
        if(CallDown[Floor]||CallUp[Floor]||CallCar[Floor]){//当前层有请求
            if(CallCar[BaseFloor]==2){
                CallCar[BaseFloor]=0;
                State=Idle;
                printf("Time: %d\t",Time);
                printf("现在在%d层，无人请求电梯\n",BaseFloor);
                return;
            }
            State=DoorOpening;//将电梯门打开
            AddAct(InOutDoorTime,ElevatorOpen);
        }

        else{//当前层无请求，考虑其他层请求
            int direct=GetWhere();
            if(direct==GoingUp){//电梯启动加速
                State=SpeedUp;
                AddAct(UpAccelerateTime,ElevatorMove);
            }else if(direct==GoingDown){
                State=SpeedDown;
                AddAct(DownAccelerateTime,ElevatorMove);
            }else{
                State=Idle;
                if(Floor!=BaseFloor)
                    AddAct(OverTime,ElevatorToFirst);
            }
        }
    }//电梯正在运行，不改变电梯状态
    return;
}
```

9. 关于用户和活动加入的操作（用链表实现）

```
int AddQueue(int infloor, Person personi){
    FloorPersonIn *t = &personin[infloor];

    while( t->next!=NULL ){
        t = t->next;
    }

    FloorPersonIn *s;
    s = (FloorPersonIn *)malloc(sizeof(FloorPersonIn));
    s->PersonID = personi.PersonID;
    s->InFloor = personi.InFloor;
    s->OutFloor = personi.OutFloor;
    s->AppearTime = personi.AppearTime;
    s->GiveUpTime = personi.GiveUpTime;
    s->next = NULL;

    t->next = s;
    return 0;
}

void AddAct(int time, void(*fn)(void)){
    time = Time + time;

    Activity *act;
    act=(Activity *)malloc(sizeof(Activity));
    act->next = NULL;
    act->fn = fn;
    act->time = time;

    Activity* p = &activity;
    while(p->next!=NULL){
        if(p->next->time > time)
            break;
        p = p->next;
    }
    act->next = p->next;
    p->next = act;
}
```

10. 电梯模块（有关电梯的一系列操作（包括电梯上升，下降，停靠，开关门等））

```
void ElevatorClose(void)//电梯关门
{
    printf("Time: %d\t",Time);
    printf("电梯门关了！\n");
    State=Stop;
}

void ElevatorOpen(void)//电梯开门
{
    printf("Time: %d\t",Time);
    printf("电梯门开了！\n");
    State=DoorOpen;
    AddAct(DoorTestTime,testinout);//检查有无人进出电梯

    if(outpos[Floor])//如果有人要出电梯
        AddAct(InOutPersonTime,PeopleOut);
    else
    {
        if(personin[Floor].next)//如果有人要进电梯
            AddAct(InOutPersonTime,PeopleIn);
    }
}
```

```
void testinout(void)//检测有无人进出
{
    if(personin[Floor].next || outpos[Floor])
        AddAct(DoorTestTime,testinout);
    else
    {
        State=DoorCloseing;
        CallUp[Floor]=0;
        CallDown[Floor]=0;
        CallCar[Floor]=0;
        AddAct(InOutDoorTime,ElevatorClose);
    }
}

 void PeopleOut(void)// 电梯里的人出
 {
    if(outpos[Floor])//如果有人要出电梯
    {
        printf("Time: %d\t",Time);
        printf("用户%d走出了电梯\n",FloorPersonOut[Floor][outpos[Floor]]);
        outpos[Floor]--;
    }

    if(outpos[Floor])//如果还有人要出电梯
    {
        AddAct(InOutPersonTime,PeopleOut);
    }
    else if(personin[Floor].next)//如果没有人出，看有没有人进
    {
        AddAct(InOutPersonTime,PeopleIn);
    }
 }

void PeopleIn(void)//电梯进人
{
    if(personin[Floor].next)//如果有人在等电梯
    {
        FloorPersonIn *p=personin[Floor].next;
        personin[Floor].next=p->next;

        int in=p->OutFloor;
        CallCar[in]=1;      //位置请求

        outpos[in]++;
        FloorPersonOut[in][outpos[in]]=p->PersonID;//进入电梯

        printf("Time: %d\t",Time);
        printf("用户%d走入了电梯\n",p->PersonID);

    }

    if(personin[Floor].next)//如果还有人在电梯外边
    {
        AddAct(InOutPersonTime,PeopleIn);
    }
}
```

```c
int GetWhere(void)//判断电梯运行的最优方向及楼层
{
    static int old=0;
    int isup=0,isdown=0;
    int i;

    for(i=Floor+1;i<MaxFloor;i++)//检查楼上有没有人按电梯
    {
        if(CallCar[i] || CallUp[i] || CallDown[i])
            isup=1;
    }

    for(i=Floor-1;i>=0;i--)//检查楼下有没有人按电梯
    {
        if(CallDown[i] || CallUp[i] || CallCar[i])
            isdown=1;
    }

    if(isup==0 && isdown==0)//如果没有上行也没有下行
        return 0;

    if(old==0)//如果停止
    {
        if(isdown)//有上行信号
            old=GoingDown;
        if(isup)//有下行信号
            old=GoingUp;
        return old;
    }

    if(old == GoingUp && isup)//如果在上行且收到上行信号
        return old;
    else if(old == GoingDown && isdown)//如果如果在下行且收到下行信号
        return old;
    else if(isup)//如果收到上行信号
        old=GoingUp;
    else if(isdown)//如果收到下行信号
        old=GoingDown;
    else
        printf("在选择方向时发生错误\n");

    return old;
}

void ElevatorToFirst(void){//当长时间无人进入时，电梯回到1楼
    if(State!=Idle||Floor==BaseFloor)
        return;
    printf("Time: %d\t",Time);
    printf("长时间无人出入，电梯回到1楼\n");
    CallCar[BaseFloor]=2;//给电梯一个去1层的初始请求
}
```

```
void ElevatorStop(void){//电梯停
    printf("Time: %d\t",Time);
    printf("电梯停了，当前为%d层\n",Floor);
    State=Stop;
}

void ElevatorUp(void){//电梯上升
    Floor++;
    printf("Time: %d\t",Time);
    printf("电梯正在上升，当前为%d层\n",Floor);
    if(CallDown[Floor]||CallUp[Floor]||CallCar[Floor]){//如果有人上下电梯，则电梯停靠
        State=SlowUp;
        AddAct(UpDecelerateTime,ElevatorStop);
    }else{
        if(Floor==MaxFloor-1){//如果电梯到顶楼，则电梯停靠
            State=SlowUp;
            AddAct(UpDecelerateTime,ElevatorStop);
        }else{//如果不为上述情况，则继续上升
            AddAct(UpTime,ElevatorUp);
        }
    }
}

void ElevatorDown(void){//电梯下降
    Floor--;
    printf("Time: %d\t",Time);
    printf("电梯正在下降，当前为%d层\n",Floor);
    if(CallDown[Floor]||CallUp[Floor]||CallCar[Floor]){//如果有人上下电梯，则电梯停靠
        State=SlowDown;
        AddAct(DownDecelerateTime,ElevatorStop);
    }else{
        if(Floor==0){//如果电梯到底楼，则电梯停靠
            State=SlowDown;
            AddAct(DownDecelerateTime,ElevatorStop);
        }else{//如果不为上述情况，则继续下降
            AddAct(DownTime,ElevatorDown);
        }
    }
}

void ElevatorMove(void){//用于电梯刚开始完成加速
    if(State==SpeedUp){
        printf("Time: %d\t",Time);
        printf("电梯已完成加速上升\n");
        State=GoingUp;
        AddAct(UpTime,ElevatorUp);
    }else{
        printf("Time: %d\t",Time);
        printf("电梯已完成加速下降\n");
        State=GoingDown;
        AddAct(DownTime,ElevatorDown);
    }
}
```

四．调试分析

1. 从本程序实习题的编制过程中容易看出，线性表运用广泛。本体中涉及的元素类型（数组，链表）均为线性表结构。

2. 算法时空分析：

1). 由于本程序所有操作均根据时钟判断进行，没有用到循环算法，每次时钟判断的时间复杂度为 O(n)，所以，整个程序的时间复杂度大致为 O(n*T)；

2). 记录电梯状态和状态转移仅需常数个空间，若每个用户信息占用常数 c 个空间，则输入 n 个用户信息，整个电梯活动所需的　　空间复杂度为 O(n)；

五. 用户手册

1. 本程序运行环境为 Ubuntu16.04 操作系统，执行文件为：Elevator
2. 进入程序后即显示提示信息：请输入用起始楼层：等待用户输入起始楼层；请输入用户目的楼层：等待用户输入目的楼层；请输入用户最长忍受时间：等待用户输入最长忍受时间；请输入下一个用户到来时间：等待用户输入下一个用户到来时间；
3. 待用户输入完成后，程序模拟电梯运行，并在电梯每次变更状态时打印出相关信息；
4. 时间超过预先定义的最长时间后，程序停止运行，退出；

六. 测试结果

```
请输入用户的起始楼层:1
请输入用户的目标的楼层:4
请输入用户的最长容忍时间:200
请输入下一个用户的到来时间:100
Time: 20        电梯门开了!
Time: 45        用户1走入了电梯
Time: 80        电梯门关了!
Time: 96        电梯已完成加速上升
请输入用户的起始楼层:1
请输入用户的目标的楼层:0
请输入用户的最长容忍时间:400
请输入下一个用户的到来时间:500
Time: 147       电梯正在上升，当前为2层
Time: 198       电梯正在上升，当前为3层
Time: 249       电梯正在上升，当前为4层
Time: 263       电梯停了，当前为4层
Time: 284       电梯门开了!
Time: 309       用户1走出了电梯
Time: 344       电梯门关了!
Time: 360       电梯已完成加速下降
Time: 421       电梯正在下降，当前为3层
Time: 482       电梯正在下降，当前为2层
Time: 500       用户2放弃了等待.
Time: 543       电梯正在下降，当前为1层
Time: 566       电梯停了，当前为1层
Time: 587       电梯门开了!
```

```
Time: 543       电梯正在下降，当前为1层
Time: 566       电梯停了，当前为1层
Time: 587       电梯门开了！
请输入用户的起始楼层:0
请输入用户的目标的楼层:3
请输入用户的最长容忍时间:600
请输入下一个用户的到来时间:100000
Time: 647       电梯门关了！
Time: 663       电梯已完成加速下降
Time: 724       电梯正在下降，当前为0层
Time: 747       电梯停了，当前为0层
Time: 768       电梯门开了！
Time: 793       用户3走入了电梯
Time: 828       电梯门关了！
Time: 844       电梯已完成加速上升
Time: 895       电梯正在上升，当前为1层
Time: 946       电梯正在上升，当前为2层
Time: 997       电梯正在上升，当前为3层
Time: 1011      电梯停了，当前为3层
Time: 1032      电梯门开了！
Time: 1057      用户3走出了电梯
Time: 1092      电梯门关了！
Time: 1393      长时间无人出入，电梯回到1楼
Time: 1410      电梯已完成加速下降
Time: 1473      电梯正在下降，当前为2层
Time: 1537      电梯正在下降，当前为1层
Time: 1564      电梯停了，当前为1层
Time: 1565      现在在1层，无人请求电梯
```

七. 附录

源程序文件名清单：

Head.h　　//本程序头文件，包含数据结构定义个函数声明；

Elevator.c　//主程序文件