

XI`AN TECHNOLOGICAL UNIVERSITY

实验报告

实验课程名称 软件算法设计

专 业： 软件工程

班 级： 18060213

姓 名： 张振龙

学 号： 18060213118

实验学时： 16

指导教师： 赵莉

成 绩：

2019年 11月18 日

西安工业大学实验报告

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 专业 | 软件工程 | 班级 | 18060213 | 姓名 | 张振龙 | 学号 | 18060213118 |
| 实验课程 | 16 | 指导教师 | 赵莉 | 实验日期 | 11.18 | 同实验者 |  |
| 实验项目 | 停车场管理 | | | | | | |
| 实验设备及器材 | 具有编写环境的电脑 | | | | | | |

**一、实验题目：**

设停车场内只有一个可停放n辆汽车的狭长通道，且只有一个大门可供汽车进出。汽车在停车场内按车辆到达时间的先后顺序，依次由北向南排列（大门在最南端，最先到达的第一辆车停放在车场的最北端），若车场内已停满n辆汽车，则后来的汽车只能在门外的便道上等候，一旦有车开走，则排在便道上的第一辆车即可开入；当停车场内某辆车要离开时，在它之后开入的车辆必须先退出车场为它让路，待该辆车开出大门外，其它车辆再按原次序进入车场，每辆停放在车场的车在它离开停车场时必须按它停留的时间长短交纳费用。试为停车场编制按上述要求进行管理的模拟程序。

**二、算法设计：**

**主要数据结构：**停车场采用栈模仿，便道采用队列模仿，事件表采用线性链表模仿

**实验遇到的难点：**

1、如何模拟汽车到停车场以及离开事件？

2、如何实现当汽车出停车场，排在其后的汽车要让其行后，在压回栈去？

3、如何模拟汽车在便道等待进入这一事件？

4、如过停车栈不满时（便道队列不空），遇到汽车到达事件，怎么控制汽车到便道而不是直接压入停车栈？

**遇到难题的解决：**

1、根据《数据结构》中“银行模拟”引入“事件表”这一概念，将汽车到达事件和离开事件记录在事件表中。

2、创建两个栈，将停车栈出栈的数据压入转换栈中，当停车栈中的要离开的车离开后，在进行相反操作，从而完成汽车让行事件

3、处理到达事件时，先判断便道队列是否为空，若不空则直接插入队列中，若空则先判断停车栈是否已满，若已满，则插入队列中，若停车栈还未满，则直接插入停车栈

4、其实这种情况并不存在，因为用事件表进行模拟时，遇到汽车到达事件，若栈满，插入队列，当事件表进行到汽车离开事件时，若队列不空则要将便道队列中第一个数据元素插入停车栈中，所以按照事件表进行下去不会出现停车栈未满，便道队列不为空的情况发生。

**三、算法实现：**

系统运行环境：Visual Studio 2017

**源代码：**

#include<stdio.h>

#include<stdlib.h>

#include<time.h>

#define CloseTime 1440

typedef int Status;

int n;

int CustomerNum;

//队列数据结构

typedef struct {

int ArrivalTime; //进入停车场的时间

int leavetime; //离开停车场的时间

}QElemType;

typedef struct QNode {

QElemType data;

struct QNode \*next;

}QNode, \*QueuePtr;

typedef struct {

QueuePtr front; //队头指针

QueuePtr rear; //队尾指针

}LinkQueue;

QElemType customer;

LinkQueue point3; //队列

//事件表数据结构

typedef struct {

int OccurTime; //当前事件发生时间

int NType; //只有两个值：0为到停车场，1为离开停车场

}Event, ElemType; //元素类型

typedef struct LNode {

ElemType data;

struct LNode \*next;

}LNode, \*LinkList; //线性链表结点

typedef LinkList EventList;

EventList ev; //事件表

Event en; //事件

//线性栈的数据结构

typedef struct {

QElemType \*base; //栈底指针，在栈构造之前和销毁之后值为：NULL

QElemType \*top; //栈顶指针

int stacksize; //当前已分配的储存空间，以使用的类型大小为单位

}L, \*SqStack;

SqStack point1, point2;

int cmp(int a, int b)

{

if (a > b)

return 1;

else if (a < b)

return -1;

else

return 0;

}

//插入事件表函数

Status OrderInsert(LinkList &L, ElemType e, int(\*compare)(int, int))

{

LinkList s;

LinkList p = L;

if (p->next == NULL) //没有元素则直接插入

{

s = (LinkList)malloc(sizeof(LNode));

s->next = NULL;

s->data = e;

p->next = s;

}

else

{

while (compare(e.OccurTime, p->next->data.OccurTime) != -1)//在a大于当前事件之前一直移向下一位

{

p = p->next;

if (p->next == NULL)

{

s = (LinkList)malloc(sizeof(LNode));

s->data = e; s->next = p->next;

p->next = s;

return 1;

}

}

s = (LinkList)malloc(sizeof(LNode));

s->data = e; s->next = p->next; //插入当前结点之后

p->next = s;

}

return 1;

}

//栈功能函数

Status InitStack(SqStack S) { //创建新的栈

S->base = (QElemType \*)malloc(n \* sizeof(QElemType)); //利用malloc函数来分配储存空间

if (!S->base) { //S->base为空，分配空间失败，发出提示

printf("存储分配失败");

}

S->top = S->base; //栈顶指针指向栈底

S->stacksize = n; //起始大小

return 0;

}

//判断栈是否已满，满则返回1

Status FullStack(SqStack S) {

if (S->top - S->base == S->stacksize) return 1;

else return 0;

}

Status Push(SqStack S, QElemType \*e) { //压栈函数

if (S->top - S->base >= S->stacksize) { //在输入元素前判断栈是否为满，防止栈溢出。

printf("栈已满"); return 1;

}

else \*(S->top++) = \*e; //存入值

return 0;

}

Status StackEmpty(SqStack S) { //判断栈是否为空的函数， 若空返回1

if (S->top == S->base) return 1;

else return 0;

}

Status Pop(SqStack S, QElemType \*e) //出栈函数

{

if (S->top == S->base) {

printf("栈为空"); return 1;

}

else \*e = \*--S->top; return 0; //将栈顶元素值赋给e,并将栈顶指向前一个元素

}

Status Push2(SqStack S, QElemType \*e)

{

if (S->top == S->base) {

printf("栈为空"); return 1;

}

else \*S->top++ = \*e; return 0;

}

Status LengthStack(SqStack S)

{

return S->top - S->base;

}

//在停车栈中寻找最小leavetime，并返回leavetime，并用e返回当前指针地址

Status FindMin(SqStack S, int \*e)

{

int i,i2=LengthStack(S);

SqStack L = S;

int t; \*e = 1;

t = S->top->leavetime;

for (i = 0; i < LengthStack(S); L->top--, i++) {

if (t > L->top->leavetime) {

t = L->top->leavetime;

\*e = L->top - L->base;

}

}

return t;

}

//事件链表相关功能函数

//创建一个新的空链表

Status ListCreate\_L(LinkList &L)

{

if (L == NULL)

{

L = (LinkList)malloc(sizeof(LNode));

L->next = NULL;

return 1;

}

else

return 0;

}

//销毁一个链表

Status ListDestroy\_L(LinkList &L)

{

LinkList p = L, s;

while (p->next != NULL)

{

s = p; p = p->next; free(s);

}

return 1;

}

//判断一个链表是否为空

int ListEmpty\_L(LinkList L)

{

if (L->next == NULL)

return 1;

else

return 0;

}

//删除链表的第一个元素，并返回e

Status ListDeleteFirst\_L(LinkList &L, ElemType &e)

{

LinkList s;

LinkList p = L;

s = p;

p = p->next;

e = p->data;

s->next = p->next;

//free(p);

return 1;

}

//队列相关函数

//链队列

Status InitQueue\_L(LinkQueue &Q)

{

//构造一个空队列Q

Q.front = Q.rear = (QueuePtr)malloc(sizeof(QNode));

if (Q.front == NULL) exit(0);

Q.front->next = NULL;

return 1;

}

Status DestroyQueue\_L(LinkQueue &Q)

{

//销毁一个队列

while (Q.front != NULL)

{

Q.rear = Q.front->next; //节省定义

free(Q.front);

Q.front = Q.rear;

}

return 1;

}

Status EnQueue\_L(LinkQueue &Q, QElemType e) {

//插入元素e为Q的新的队尾元素

QueuePtr p = (QueuePtr)malloc(sizeof(QNode));

if (!p) exit(0);

p->data = e; p->next = NULL;

Q.rear->next = p;

Q.rear = p;

return 1;

}

Status DeQueue\_L(LinkQueue &Q, QElemType &e)

{

//若队列不空 删除Q的队头元素，用e返回其值，并返回OK

//否则返回ERROR

QueuePtr p;

if (Q.rear == Q.front) return 0;

p = Q.front->next;

e = p->data;

Q.front->next = p->next;

if (Q.rear == p) Q.rear = Q.front;

free(p);

return 1;

}

//判断队列是否为空

int QueueEmpty\_L(LinkQueue Q)

{

if (Q.rear == Q.front)

return 1;

else

return 0;

}

Status GetHeadQueue\_L(LinkQueue Q, QElemType &e)

{

e = Q.front->next->data;

return 1;

}

//队列插入停车栈函数

//初始化函数

void OpenForDay()

{

CustomerNum=0;

ListCreate\_L(ev); //初始化事件链表为空表

en.OccurTime = 0; en.NType = 0; //设定第一个客户到达事件

OrderInsert(ev, en, cmp); //插入事件表

InitQueue\_L(point3); //置空队列

}//OpenForDay

void CustomerArrived(SqStack &S)

{

//处理客户到达事件 en.NType = 0;

int durtime, intertime, t, \*e, e1; e = &e1;

ElemType tmp;

QElemType \*Qtmp, Qtmp1,Qtmp2; Qtmp = &Qtmp1;

++CustomerNum;

durtime = 1 + (int)(360.0\*rand() / (RAND\_MAX + 1.0)); //处理时间,随机生成0到359的数字

intertime = 1 + (int)(60.0\*rand() / (RAND\_MAX + 1.0)); //下一辆车到达时间间隔

t = en.OccurTime + intertime; //下一辆车到达时间

if (!FullStack(S)) {

if (t < CloseTime) //停车场一天 插入事件表,停车场空且没车等待，直接插入停车栈，否则插入等候栈

{

if (QueueEmpty\_L(point3)) { //栈不满，队列空

tmp.OccurTime = t;

tmp.NType = 0;

OrderInsert(ev, tmp, cmp);

tmp.OccurTime = en.OccurTime + durtime;

tmp.NType = 1;

OrderInsert(ev, tmp, cmp); //进停车栈就要记录离开事件

Qtmp1.ArrivalTime = t; Qtmp1.leavetime = t + durtime;

Push(S, Qtmp); //进入停车栈

}

/\*Qtmp2.ArrivalTime = FindMin(point1, e);

Qtmp2.leavetime = FindMin(point1, e) + durtime;

EnQueue\_L(point3, Qtmp2);

}\*/

}

}

else { //栈满，队列记录到达停车场事件，插入队列

if (t < CloseTime) {

tmp.OccurTime = t;

tmp.NType = 0;

OrderInsert(ev, tmp, cmp);

Qtmp2.ArrivalTime = FindMin(point1,e);

Qtmp2.leavetime = FindMin(point1,e) + durtime;

EnQueue\_L(point3, Qtmp2); //栈满，队列不空，顺序排

}

}

}

void CustomerDeparture(SqStack S, SqStack S2)

{

//处理客户离开事件 ，en.Type>0;

ElemType tmp;

QElemType e, \*e1; e.ArrivalTime = 0; e.leavetime = 0;

e1 = &e;

int t2 = 1,durtime;

int \*t; int i;

t = &t2;

if (QueueEmpty\_L(point3)) { //队空

for (i = 0; i < LengthStack(S) - \*t; i++) {

Pop(S, e1);

Push(S2, e1);

}

Pop(S, e1);

t2 = 0.05\*(e.leavetime - e.ArrivalTime);

printf("进入停车场时间:%d，离开停车场时间:%d，应缴费:%d\n", e.ArrivalTime, e.leavetime, t2);

while (!StackEmpty(S2)) {

Pop(S2, e1); Push(S, e1);

}

}

else { //队列不空时将队列元素插入

FindMin(point1, t);

durtime = point3.front->data.leavetime - point3.front->data.ArrivalTime;

point3.front->data.ArrivalTime = FindMin(S, t); point3.front->data.leavetime = FindMin(S, t) + durtime;

for (i = 0; i < LengthStack(S) - \*t; i++) {

Pop(S, e1);

Push(S2, e1);

}

Pop(S, e1);

t2 = 0.05\*(e.leavetime - e.ArrivalTime);

printf("进入停车场时间:%d，离开停车场时间:%d，应缴费:%d\n", e.ArrivalTime, e.leavetime, t2);

while (!StackEmpty(S2)) {

Pop(S2, e1); Push(S, e1);}

DeQueue\_L(point3, customer); //队列的排头车进入栈，在此之前将阻碍的车移到另一个栈

e.ArrivalTime = customer.ArrivalTime; e.leavetime = customer.leavetime;

Push2(S, e1);

tmp.OccurTime = e.leavetime;

tmp.NType = 1;

OrderInsert(ev, tmp, cmp);

} //进停车栈就要记录离开事件}

void Bank\_Simulation()

{

ElemType tmp;

OpenForDay();

point1 = (SqStack)malloc(sizeof(L));

point2 = (SqStack)malloc(sizeof(L));

InitStack(point1); InitStack(point2);

while (!ListEmpty\_L(ev)) //事件表非空继续执行

{

ListDeleteFirst\_L(ev, tmp);

en = tmp;

if (en.NType == 0)

CustomerArrived(point1);

else

CustomerDeparture(point1, point2);

}}

int main()

{

printf("请输入停场车位数:");

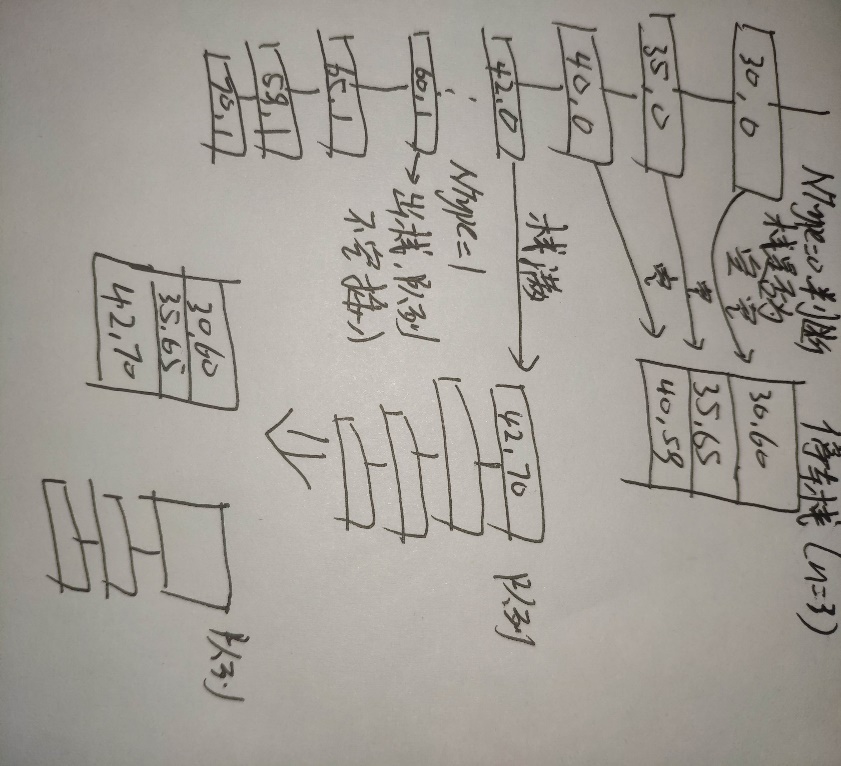
scanf\_s("%d", &n);

srand((int)time(0));

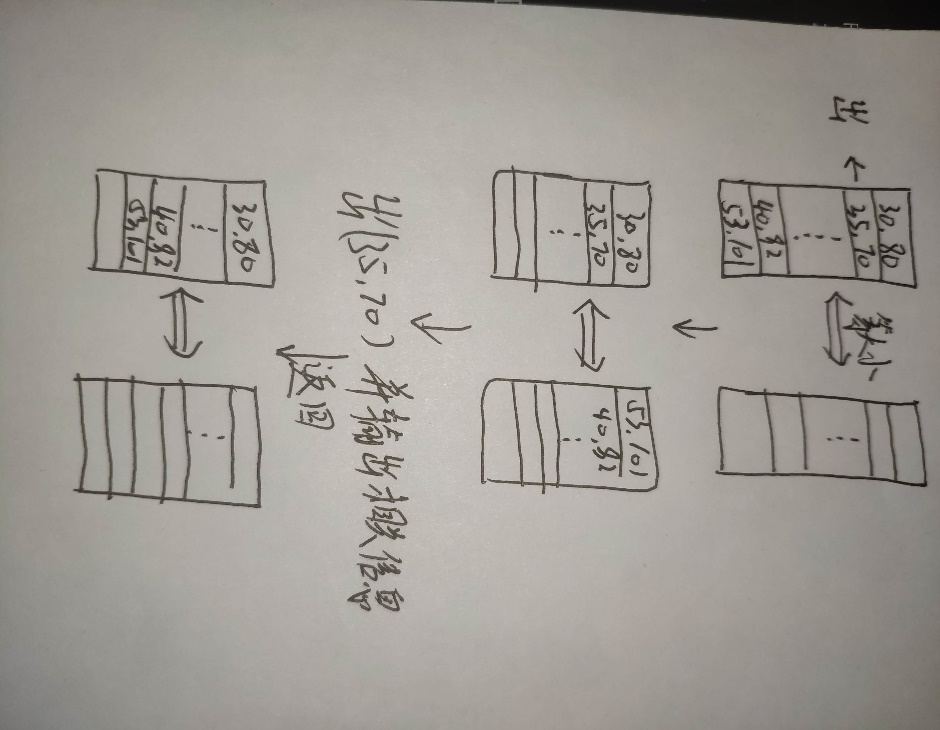
Bank\_Simulation();

system("pause");

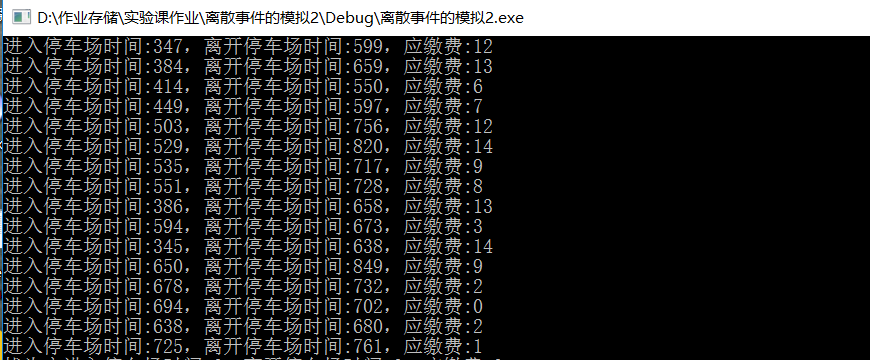
return 0;

}

主要函数流程图：



**实验结果截图：**

****

**四、心得体会：**

变量要尽量定义为看其名知其意的。

当处理一件事遇到问题时不要拘泥于一开始的思路，必要时一定要“跳出一开始的思路”，这样可能才能找到解决问题的方法。学算法不是学形式，而是学思维，数据结构只是工具。

**五、参考文献：**

**《数据结构》严蔚敏版**