# 银行业务模拟

## 一、**需求分析**

### 1.本程序主要有两个功能：记录事件和对客户的业务处理。

##### 1.1记录事件：

##### 每当有客户到达或者离开，记录下这个事件，把它添加到事件表中，事件表 相当于一个安装在银行门口的摄像头。

##### 1.2业务处理

客户到达时，先排第一个队列，对于银行可以满足的金额，马上处理，处理完后客户离开。如果银行满足不了（取款金额大于银行余额），就将客户排到第二个窗口的队列中，当有客户来存款后，马上关闭第一个窗口，打开第二个窗口，然后遍历第二个队列，对可以满足的客户马上处理业务，然后这个客户离开银行，如果不满足重新排到队尾。一直重复上述过程，直到时间达到银行的关门时间，所有还在排队的客户必须马上离开银行。

### 2.输入的形式

输入的元素为整型类型；

输入的银行初始存款BankBalance必须大于0；

输入的银行营业时间OpenTime必须大于0且必须小于1440（一天）；

输入的时间间隔上限IntervalTime必须大于0且必须小于银行营业时间；

输入的处理时间上限MaxDealTime必须大于0且必须小于银行营业时间；

输入的交易额上限MaxAmount必须大于0且必须小于银行初始存款。

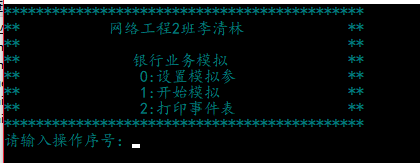
### 3.输出的形式

1. 在开始模拟的时候，每来一个用户就输出这个用户的信息，最后形成一个按时间顺序的客户列表，客户表的每一列包括：客户序号，到达事件，交易的时间，交易金额。
2. 在结束模拟后，以时间顺序输出事件表，事件表每一列包括：客户序号，事件类型，发生时间。
3. 模拟结束后，输出本次服务的客户数量、取款和取款成功的人数、存款和存款成功的人数、客户的平均逗留时间、银行的余额。

### 4.程序所能达到的功能

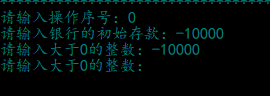
可以按时间顺序记录每一个事件，可以按照题目要求处理客户业务，题目给出的功能基本都完成了。

## 测试数据，包括正确的输入及其输出结果和含有错误的输入及其输出结果

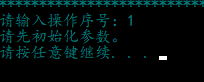


### 1.错误数据

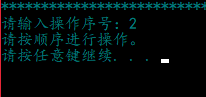
如果有错误的输入数据会及时提示，直到输入正确的数据才可以进行下一个操作，下面给出其中一个例子：



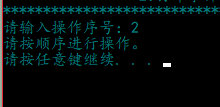
如果还没初始化模拟参数就想开始模拟：



如果还没初始化参数就想打印事件表：

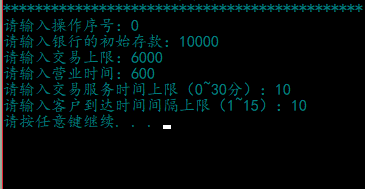


如果还没模拟就想打印事件表：

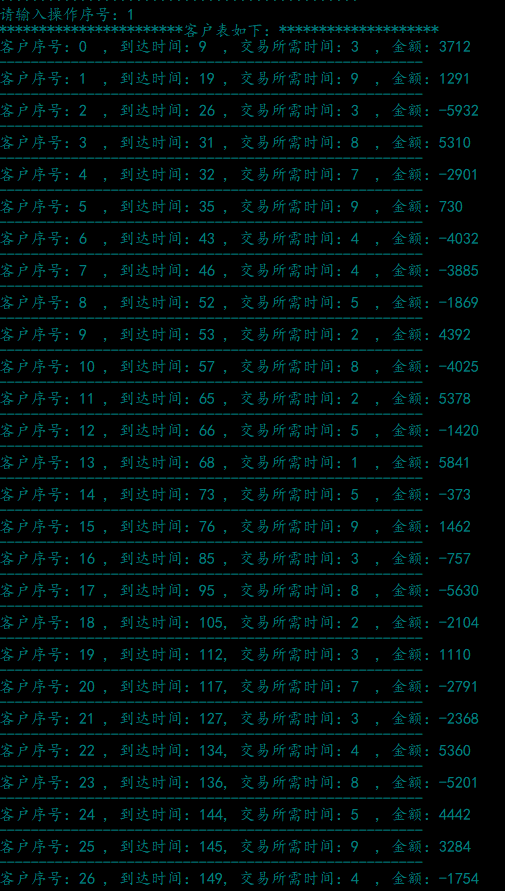


### 正确的数据

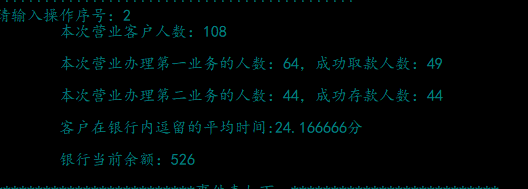
初始化参数：



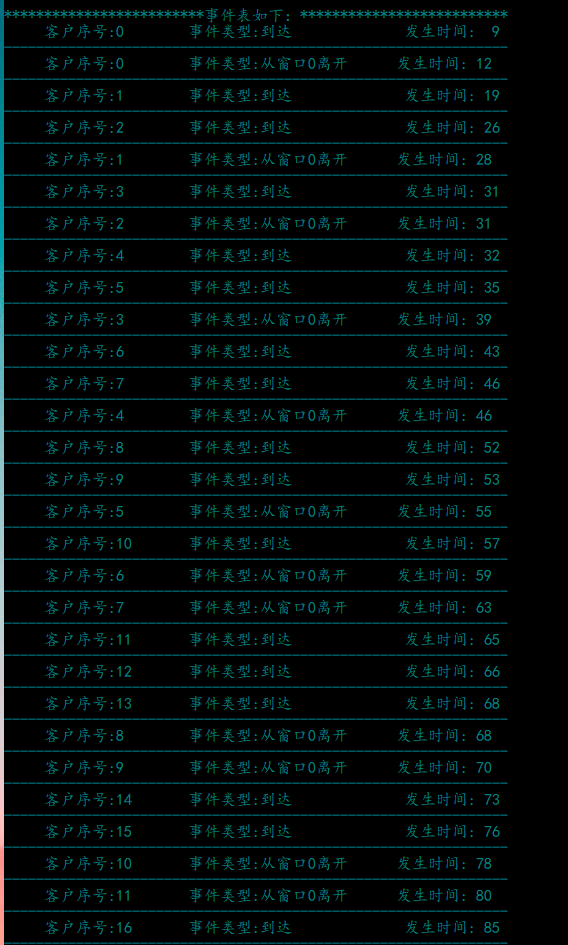
开始模拟：（节选数据）



查看最后结果：



事件表如下（节选数据）：



## 概要设计

##### 1.数据结构设计

设计中用到的数据结构ADT定义如下：

ADT LQueue{

数据对象：D＝{ ai | ai∈ElemSet, i=1,2,...,n, n≥0 }

数据关系：R1＝{ <ai-1, ai>|ai-1, ai∈D, i=2,...,n }

基本操作：

//销毁队列

void DestoryE(ELQueue &Q);

//销毁队列

void DestoryC(CLQueue &Q);

//链队列的出队

Status DeQueue\_ELQ(ELQueue &Q);

//链队列入队

Status EnQueue\_ELQ(ELQueue &Q);

//构造一个事件空队列

void InitQueue\_ELQ(ELQueue &Q);

//链队列的出队

Status DeQueue\_CLQ(CLQueue &Q);

//构造一个空队列

void InitQueue\_CLQ(CLQueue &Q);

//链队列入队

Status EnQueue\_CLQ(CLQueue &Q);。

}ADT Queue

##### 2.主程序的流程

int main(int argc, char\*\* argv) {

int num; //操作序号

srand((unsigned)time(NULL));

while(1){

indexPrint();

printf("请输入操作序号：");

scanf("%d", &num);

fflush(stdin); //清空输入缓冲区

if(num == 0) initial(); //初始化参数

if(num == 1) simulation(); //模拟

if(num == 2) result(); //查看结果

system("PAUSE"); //冻结屏幕

system("CLS"); //清屏操作

}

return 0;

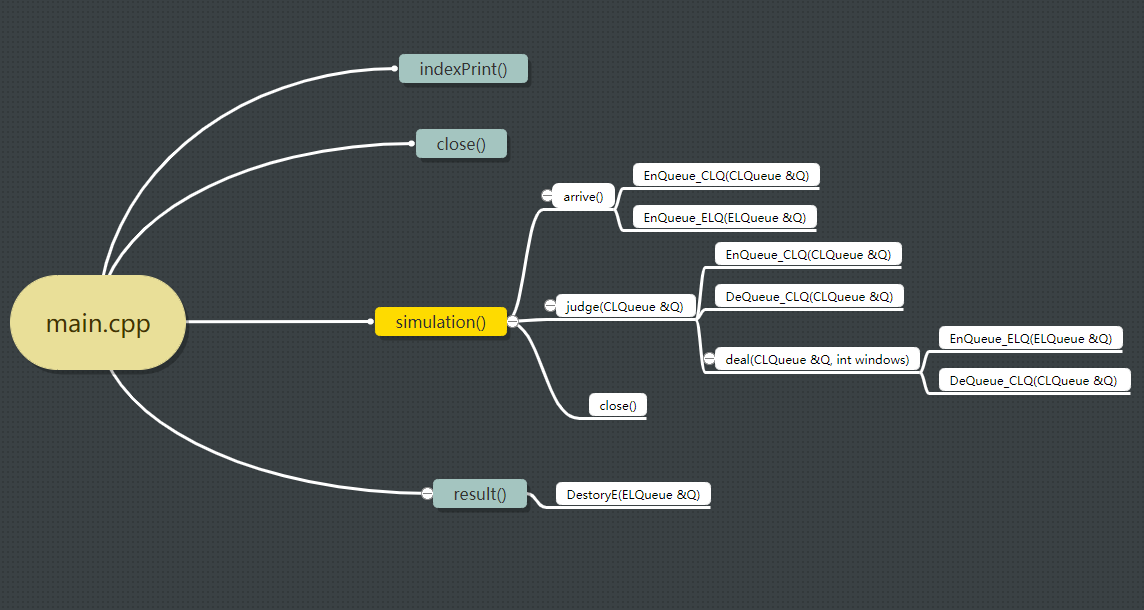
}

主程序主要就是调用方法接口，进入while循环indexPrint()打印界面；当输入为0时，调用initial()开始初始化参数；当输入为1时，开始调用simulation()开始模拟；当输入为2时，调用result()输出结果。

##### 各模块之间的调用关系

两个模块银行业务逻辑的模块、队列的操作：调用关系main程序调用bank\_controller.cpp

业务逻辑，bank\_controller.cpp调用lqueue.cpp对队列进行操作。



## 详细设计

typedef int Time; //时间

typedef int Money; //金额

typedef struct EventNode {

int num; //客户编号

int type; //事件类型，0/1从哪个窗口离开，2到达事件

Time occutime; //发生时间

struct EventNode \*next;

}EventNode, \*event;

typedef struct ELQueue{

event front; //队头指针

event rear; //队尾指针

}ELQueue;

typedef struct CustNode {

int num; //客户编号

Time arrtime; //到达时间

Time durtime; //逗留时间

Money amount; //金额大小

struct CustNode \*next;

} CustNode, \*Client;

typedef struct CLQueue{

Client front; //队头指针

Client rear; //队尾指针

}CLQueue;

设计两种链队列，一种是客户表，一种是事件表，避免耦合。

定义的常量如下：

CLQueue Q1; //队列1

CLQueue Q2; //队列2

ELQueue Event; //事件表

int FirstSuccess; //办理第一业务成功的人数

int SecondSuccess; //办理第二业务成功的人数

int FirstService; //办理第一业务的人数

int SecondService; //办理第二业务的人数

int LastBalance; //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

int OpenWindows = 0; //开启的是几号窗，默认0号

int FreeTime = 0; //服务空挡时间，标志什么时候窗口空闲

int IntervalTime; //到达间隔时间上限

int NextArriveTime = 0; //下一个到达时间

int OpenTime; //营业时间

int MaxAmount; //交易上限

int MaxDealTime; //交易时间上限

int CurrentTime = 0; //现在时间，初始为0

int Number = 0; //客户编码，初始为0

int BankBalance = 0; //银行余额

int TotalDurtime = 0; //总逗留时间

##### 1.因为两种结构的基本操作一样，下面分析客户表的操作：

1.//链队列入队

Status EnQueue\_CLQ(CLQueue &Q)

伪代码如下：

Begin

分配存储空间给结点p

结点p的金额置<-e

结点p的指针域<-空

IF队列<-空

Begin

头指针<-p

尾指针<-p

End

ELSE

Begin

头指针next域<-p

尾指针<-尾指针next域

End

End

2.//链队列的出队

Status DeQueue\_CLQ(CLQueue &Q);

伪码表示：

Begin

p<-头指针

IF p的next域<-空

Begin

头指针<-空

尾指针<-空

End

ELSE

头指针<-头指针的next域

DELETE p

End

3.//销毁队列

void DestoryC(CLQueue &Q);

伪代码表示：

Begin

WHILE 队列不为空

Begin

尾指针<-头指针的next

DELETE 头指针

头指针<-尾指针

END

End

##### 2.下面分析主程序算法：

Begin

WHILE 1

Begin

输出主页面

输入操作序号

IF 输入操作序号<-0

Begin

初始化参数

End

IF 输入操作序号<-1

Begin

开始模拟

End

IF 输入操作序号<-2

Begin

查看结果

End

End

##### 3.bank\_controller.cpp算法

1)、void indexPrint(); //打印开始界面

伪代码表示：

Begin

字体颜色选择

输出格式

End

2)、//初始化参数

void initial()

伪代码表示：

Begin

输入银行初始存款

WHILE 输入不符合标准

Begin

提示输入怎样的数据

继续输入

End

其它参数类似输入

End

3)、//模拟

void simulation()

Begin

IF 数据没有初始化

Begin

输出提示

End

ELSE

Begin

得到一个随机的客户到达时间

WHILE 现在时间 < 营业时间

Begin

现在时间++

IF 现在时间 == 下一个客户的到达时间

Begin

调用到达函数arrive()

打印客户信息

客户序号++

下个客户到达时间 <-随机间隔时间+这个客户的到达时间

End

IF 窗口空闲时间 <= 现在时间 && 开启的是0号窗口&& 队列1不为空

Begin

调用判断的方法judge(队列1)

End

IF 窗口空闲时间 <= 现在时间 && 开启的是1号窗口&& 队列2不为空

Begin

调用判断的方法judge(队列2)

End

End

IF 现在时间 == 结束营业时间

Begin

调用close()

End

End

End

4)、//客户到达

void arrive()

伪代码表示：

Begin

客户入队队列1

到达事件入队事件表

给队列1的队尾客户初始化参数：

到达时间<-事件发生时间<-现在时间

金额<-随机数

IF 金额 < 0

Begin

办理第一业务的人数++

End

ELSE

Begin

办理第二业务的人数++

End

业务处理时间<-随机时间

客户序号

IF 队列1只有一个人 && 开启的是0号窗

Begin

IF 这个人的业务金额 > 0 || 银行能满足其取款金额

Begin

空闲时间 <- 现在时间 + 业务处理时间

End

End

End

5)、//判断下一步怎么处理

void judge(CLQueue &Q)

Begin

IF 队头客户取款额大于银行存款

Begin

入队到队列2

队列1出队

IF 队列1不为空

Begin

IF 这个人的业务金额 > 0 || 银行能满足其取款金额

Begin

空闲时间 <- 现在时间 + 业务处理时间

End

End

退出这个方法

End

进行交易deal()

IF 交易金额>0 || 开启的是1号窗口

Begin

记录这个客户之前的银行存款

检查队列2check()

End

End

6)、//交易

void deal(CLQueue &Q, int windows)

伪代码表示:

Begin

进行交易更新银行存款

F 金额 < 0

Begin

办理第一业务的成功人数++

End

ELSE

Begin

办理第二业务的成功人数++

End

更新总的逗留时间

离开事件入队

客户队列出队

IF 队列1不为空 && 开启的是0号窗

Begin

IF 这个人的业务金额 > 0 || 银行能满足其取款金额

Begin

空闲时间 <- 现在时间 + 业务处理时间

End

End

End

7)、//检查第二个队列

void check()

Begin

IF 队列2为空 && 开启的是1号窗口

Begin

开启0号窗口

IF 队列1不为空

Begin

IF 这个人的业务金额 > 0 || 银行能满足其取款金额

Begin

空闲时间 <- 现在时间 + 业务处理时间

End

End

退出这个方法

End

IF 队列2为空 && 开启的是0号窗口

Begin

结束这个方法

End

p <- sign <-队列2队头

WHILE 银行存款 > 上个客户交易前余额

Begin

IF 银行能满足其取款金额

Begin

IF 队列1不为空

Begin

IF 这个人的业务金额 > 0 || 银行能满足其 取款金额

Begin

空闲时间 <- 现在时间 - 队列1的 队头业务处

End

End

IF 队列2不为空

Begin

更新空闲时间

End

开启1号窗口

BREAK;

End

ELSE

Begin

将队列2的队头重新排到队尾

End

p = 队列2队头

IF 循环过一遍

Begin

开启0号窗口

BREAK;

End

End

IF 银行存款 <= 上个客户交易前余额

开启0号窗口

Begin

IF 队列1不为空

Begin

IF 这个人的业务金额 > 0 || 银行能满足其取款金额

Begin

空闲时间 <- 现在时间 + 业务处理时间

End

End

End

End

8)、//停止营业

void close()

伪代码：

Begin

将队列1一个一个出队

离开事件一个一个入队

将队列2一个一个出队

离开事件一个一个入队

End

## 调试分析

##### 调试遇到的问题

主要问题是事件表的按顺序记录事件的需求，需要合理安排交易的发生时间，和检查队列2的时机，以及两个队列交替交易时，到达事件的处理顺序。

##### 算法分析

|  |  |  |
| --- | --- | --- |
| 接口 | 算法时间复杂度 | 算法空间复杂度 |
| EnQueue(Queue &Q) | O(1) | O(1) |
| DeQueue(Queue &Q) | O(1) | O(1) |
| arrive() | O(1) | O(1) |
| deal() | O(1) | O(1) |
| judge() | O(1) | O(1) |
| indexPrint() | O(1) | O(1) |
| initial() | 不可测 | O(1) |

##### 3.经验体会

体会到了用C语言也可以像java一样可以进行后台开发的大概流程，对C语言有了更深的理解。

##### 4.用户使用说明

在进入银行业务模拟界面时，选择操作编号：

0-初始化参数，1-开始进行模拟，2-查看结果

进入模拟系统后，输入的银行初始存款必须大于0；

输入的元素为整型类型；

输入的银行初始存款BankBalance必须大于0；

输入的银行营业时间OpenTime必须大于0且必须小于1440（一天）；

输入的时间间隔上限IntervalTime必须大于0且必须小于银行营业时间；

输入的处理时间上限MaxDealTime必须大于0且必须小于银行营业时间；

输入的交易额上限MaxAmount必须大于0且必须小于银行初始存款且必须

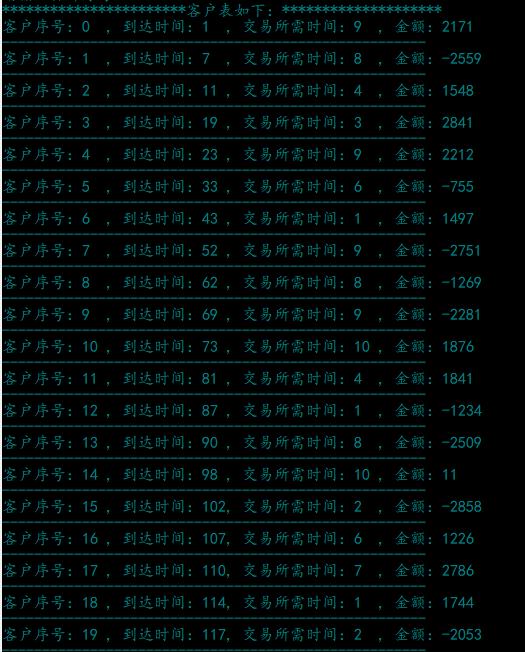
若输入无误，则开始进行输出，输出事件处理的列表信息，以及需要存款的客户人数，需要取款的客户人数，成功办理存款的客户人数，成功办理取款的客户人数，客户逗留平均时间，银行当前余额等信息。

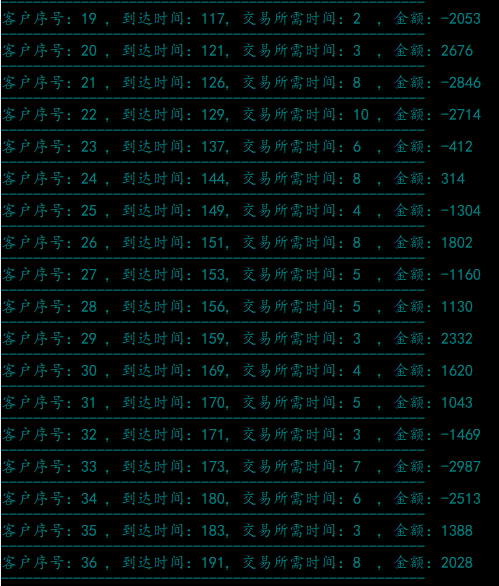
## **测试结果**

##### 1.输入：



##### 2.开始模拟打印客户表：





##### 打印事件表

