



**数据结构课程设计报告**

**课程名称\_\_\_\_\_\_数据结构\_\_\_\_\_**

**题目名称 银行业务模拟**

**学生学院\_\_ 计算机学院\_\_\_\_\_\_**

**专业班级\_计算机科学与技术（5） \_**

**学 号\_\_\_ \_3117004622 \_\_**

**学生姓名\_\_\_\_ 刘友滨 \_\_**

**指导教师\_\_\_\_\_\_ 黄剑锋 \_\_**

**2018 年 12月 25日**

1. **需求分析**

【**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。

**【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。

**【设计任务】**

本程序是模拟银行存取款的流程。模拟客户在银行办理存取款的排队以及办理过程，并计算出客户总的逗留时间。

程序的设计是需要实现两个排队队列（**Q1、Q2**），一个事件队列（**Q\_en**）。队列1（**Q1**）和队列2（**Q2**）用于模拟客户的排队情况，客户从队尾开始排队，从队头出队。事件队列（**Q\_en**）用于记录客户的到达/离开事件，客户到达时记录到达事件，客户办理完存取款后离开时记录离开事件。到达事件记录客户的序列号、办理金额、到达时间；离开事件记录客户的序列号、办理金额、离开时间。

客户的到达时间是随机确定的（以客户最大到达时间**arriveMaxTime**为上界，客户最小到达时间**arriveMinTime**为下界）。客户若办理完存取款离开则离开时间由客户的到达时间加上交易时间（交易时间由**dealMaxTime和dealMinTime**随机确定）**。**客户的交易额由最大交易额上限（**MaxMoney**）随机确定，规定负数为取款，正数为存款。

开始时,客户到达银行进入队列1（**Q1**）排队同时记录客户的到达事件将事件存到事件队列（**Q\_en**）。如果客户办理存款，则更新银行的存款（**total\_money**）,然后将客户结点从队列1（**Q1**）删除，同时事件表（**Q\_en**）记录客户的离开事件。如果客户办理取款，当银行当前的金额（**total\_money**）能够满足客户的取款需求，则更新银行的存款（**total\_money**），然后将客户结点从队列1（**Q1**）删除，同时事件表（**Q\_en**）记录客户的离开事件。当银行当前的金额（**total\_money**）不能够满足客户的取款需求，则将客户移到队列2（**Q2**）进行等待。每当队列1（**Q1**）办理完一次存款操作的时候就顺序检查队列2（**Q2**），查找能够完成取款操作的客户，当队列2（**Q2**）当前客户完成取款则更新银行存款（**total\_money**），然后将客户从队列2（**Q2**）中删除，同时事件表（**Q\_en**）记录客户的离开事件，接着转到队列1（**Q1**）接待客户。如果队列2（**Q2**）中当前客户无法完成取款则将队客户移到队列2（**Q2**）的队尾继续排队，接着检查下一个客户。

当银行营业结束，检查队列1（**Q1**）和队列2（**Q2**）将还在排队的客户删除同时更新客户总逗留时间和接待客户总数。最后输出客户排队情况、接待客户总人数（**counter**）、需要存款的人数（**n\_p**）、成功存款的人数（**s\_p**）、需要取款的人数（**n\_g**）、成功取款的人数（**s\_g**）、客户逗留总时间（**totalTime**）、客户逗留平均时间（**totalTime/ counter**）以及银行现存金额（**total\_money**）。

**【输入的形式和输入值的范围】**

1. 银行模拟程序需要用户输入以下几个参量：

**银行初始存款（total\_money）**

**银行营业时间(closeTime)**

**客户最大到达时间间隔(arriveMaxTime)**

**客户最小到达时间间隔(arriveMinTime)**

**业务最大处理时间(dealMaxTime)**

**业务最小处理时间(dealMinTime)**

**最大交易额上限(MaxMoney)**

1. 参量输入要求用户从终端输入。

3.**total\_money、closeTime、arriveMaxTime、dealMaxTime、MaxMoney、arriveMinTime、dealMinTime**要求是一个大于零的整数。

同时**arriveMinTime**不能大于 **arriveMaxTime，dealMinTime**不能大于**dealMaxTime。**

以及**arriveMinTime、** **arriveMaxTime、dealMinTime、dealMaxTime**均不能大于**closeTime**

**【输出的形式】**

银行模拟程序通过dos界面输出银行营业过程的整个排队结果以及银行办理业务的结果。

**输出：**

**需要存款的客户人数(n\_p)**

**需要取款的客户人数(n\_g)**

**成功存款的客户人数(s\_p)**

**成功取款的客户人数(s\_g)**

**客户逗留总时间(totalTime)**

**接待客户总数(counter)**

**客户逗留平均时间(totalTime/counter)**

**银行结业余额(total\_money)**

**【程序所能达到的功能】**

程序可以根据用户自定输入的合法参量（银行初始存款（total\_money）、银行营业时间(closeTime)、客户最大到达时间间隔(arriveMaxTime)、客户最小到达时间间隔(arriveMinTime)、业务最大处理时间(dealMaxTime)、业务最小处理时间(dealMinTime)、最大交易额上限(MaxMoney)）来模拟银行的存取款业务。通过模拟输出结果包括客户排队情况、接待客户总人数（**counter**）、需要存款的人数（**n\_p**）、成功存款的人数（**s\_p**）、需要取款的人数（**n\_g**）、成功取款的人数（**s\_g**）、客户逗留总时间（**totalTime**）、客户逗留平均时（**totalTime/ counter**）以及银行现存金额（**total\_money**）。

**【测试数据】**

**1.第一次测试**

**银行初始存款（total\_money）：10000**

**银行营业时间(closeTime)：600**

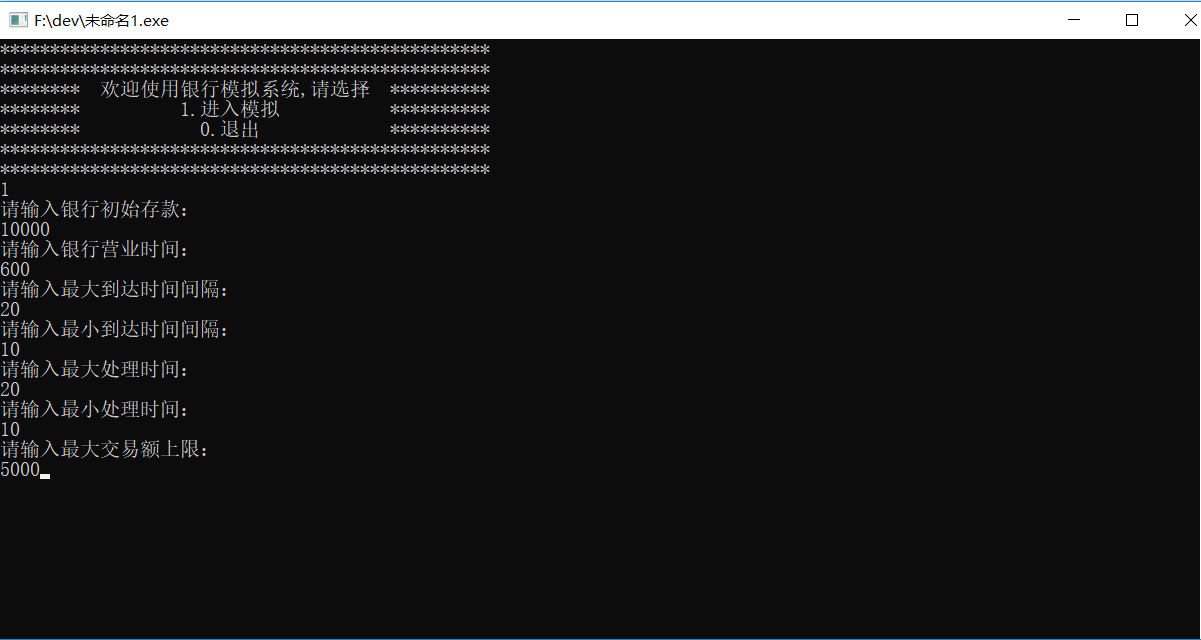
**客户最大到达时间间隔(arriveMaxTime)：20**

**客户最小到达时间间隔(arriveMinTime)：10**

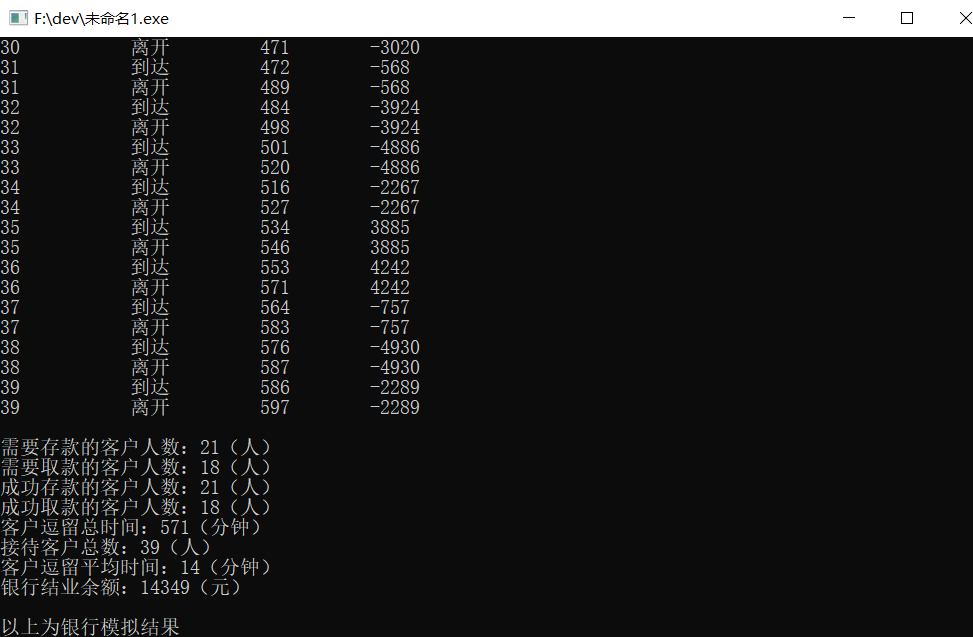
**业务最大处理时间(dealMaxTime)：20**

**业务最小处理时间(dealMinTime)：10**

**最大交易额上限(MaxMoney)：5000**

****

**输出结果：**

****

**2第二次测试让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**

**银行初始存款（total\_money）：10000**

**银行营业时间(closeTime)：600**

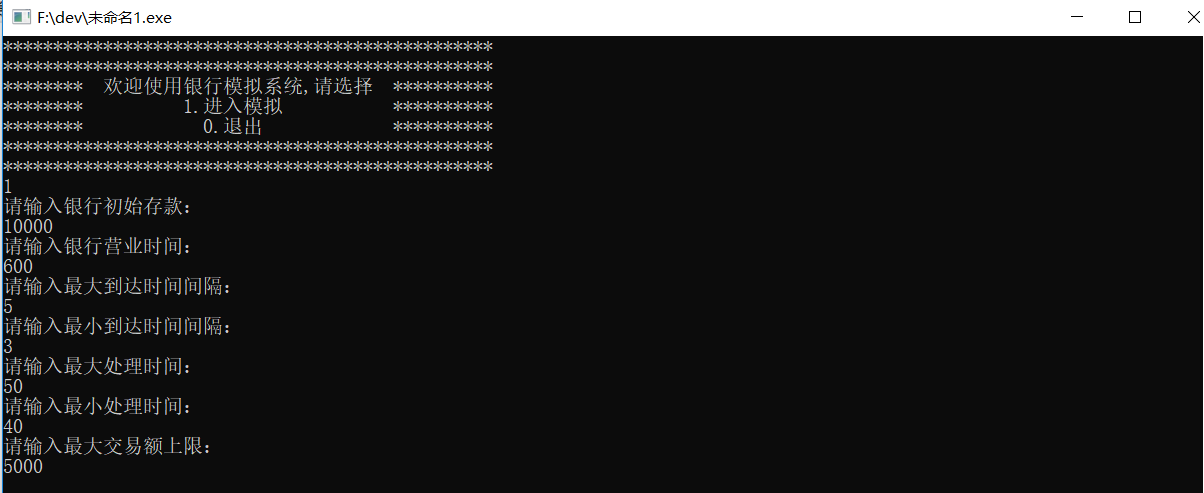
**客户最大到达时间间隔(arriveMaxTime)：5**

**客户最小到达时间间隔(arriveMinTime)：3**

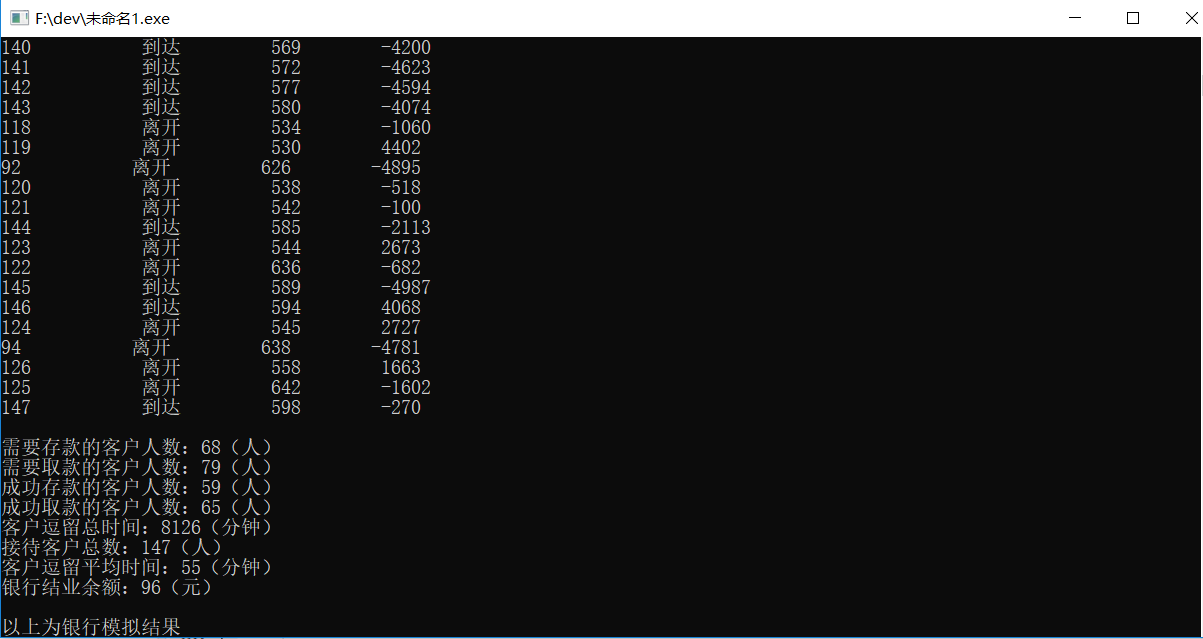
**业务最大处理时间(dealMaxTime)：50**

**业务最小处理时间(dealMinTime)：40**

**最大交易额上限(MaxMoney)：5000**

****

**输出结果：**

****

**3第三次测试让两个到达事件之间的间隔时间很长，而客户的交易时间很短**

**银行初始存款（total\_money）：10000**

**银行营业时间(closeTime)：600**

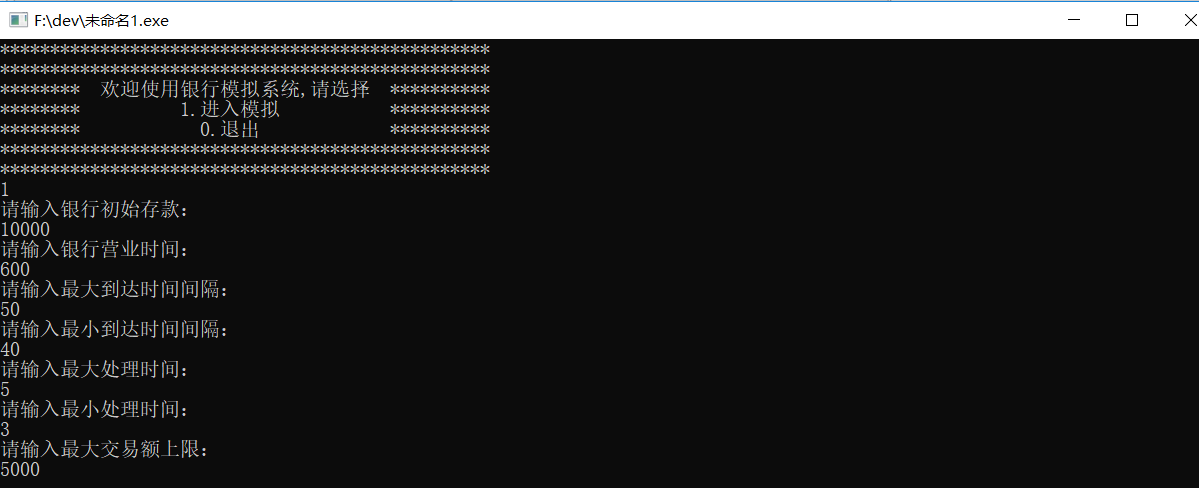
**客户最大到达时间间隔(arriveMaxTime)：50**

**客户最小到达时间间隔(arriveMinTime)：40**

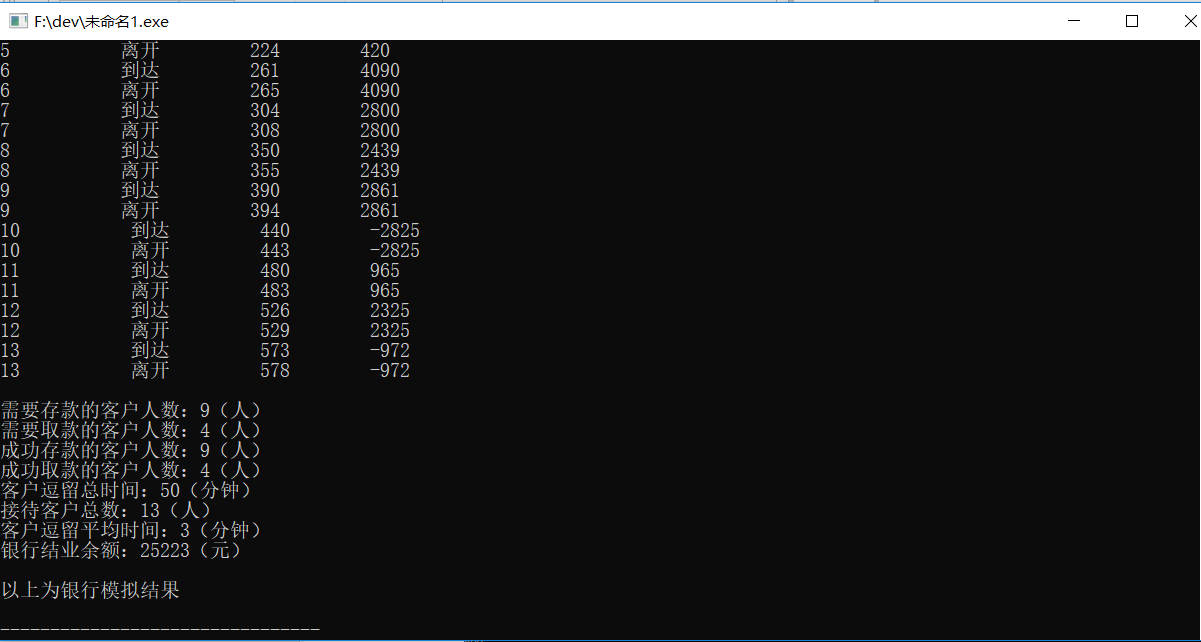
**业务最大处理时间(dealMaxTime)：5**

**业务最小处理时间(dealMinTime)：3**

**最大交易额上限(MaxMoney)：5000**

****

**输出结果：**

****

**4 第四次测试测试输入非法数据的情况**

****

1. **概要设计**

**【数据类型的定义】**

**结构体定义**

/\*客户/事件结构体\*/

typedef struct event{

int type;//事件类型到达或者离开 0为到达，1为离开

int arriveTime;//到达时间

int leaveTime;//离开时间

int money;//存款数，正数为存款，复数为取款

int num;//客户编号

event\* next;

}event,\*eventLink;

/\*队列结构体\*/

typedef struct QNode{

eventLink front;//队头指针

eventLink rear; //队尾指针

}QNode,\*Queue;

**队列抽象数据类型的定义**

**ADT Queue{**

**数据对象**：D＝{ ai | ai∈event, 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**：

eventLink get\_front(Queue &q)；

初始条件：队列q存在

操作结果：返回队头结点指针

void push(Queue &q,eventLink e)；

初始条件：队列q存在，结点指针e存在

操作结果：将e指向的结点插入到队列q的队尾

void destoryQueue(Queue &q)；

初始条件：队列q存在

操作结果：销毁队列q，释放空间

void pop(Queue &q)；

初始条件：队列q存在

操作结果：删除对头结点并释放结点的空间

**}ADT Queue**

**全局变量的定义**

int total\_money;//银行现存款

int closeTime;//银行结束营业的时间

int arriveMaxTime;//两个到达事件的时间间隔上限

int arriveMinTime; //两个到达事件的时间间隔下限

int dealMaxTime;//客户交易之间的时间上限

int dealMinTime; //客户交易之间的时间下限

int MaxMoney;//交易额上限

int currentTime=0;//当前时间

int totalTime;//客户逗留总时间

int counter=0;//客户总数

int number=1;//客户初始号码

int flag=1;//判断是否有窗口在处理

int TimeOfDeal;//交易时间

int MaxTime;//到达时间

Queue Q\_en;//事件队列

Queue Q1;//队列一

Queue Q2;//队列二

**函数类型定义**

/\*\*

结束函数

当银行营业结束的时候调用此函数，清理释放申请的空间

\*/

**void closeBank()**；

/\*\*

到达函数

当客户到达时调用此函数，随机生成客户信息以及相应的到达事件，分别加入队列1和事件队列。

\*/

**void arrive()；**

/\*\*

存款函数

当客户办理存款的时候调用此函数

\*/

**void putMoney()；**

/\*\*

取款函数

当客户办理取款的时候调用此函数

\*/

**void getMoney()；**

/\*\*

查找函数

查找队列q中第一个可以满足取款的结点并返回，若是查找失败则返回NULL

\*/

**eventLink search(Queue &q,int m)；**

/\*\*

处理函数

对通过【查找函数】找到的结点进行取款处理，生成相应的离开事件

\*/

**void findAndDeal()；**

/\*\*

初始化函数

初始化三个队列，为三个队列分配空间

\*/

**void initQueue()；**

/\*\*

主函数

\*/

**int main()；**

**【主程序的流程】**

主程序首先先接收用户的数据输入，并对用户输入的数据进行验证，如果输入的数据不合法则要求用户重新输入。

当用户输入的数据合法后开始主要程序的运行。首先先调用**initQueue()**函数对队列进行初始化，以及根据客户到达时间的上下界随机生成第一个客户的到达时间。当当前时间（**currentTime**）小于银行结业时间（**closeTime**）时进行循环。每循环一次当前时间加1。

在每一次循环当中进行四次判断。

1.当交易时间（**TimeOfDeal**）小于当前时间（**currentTime**）的时候，让交易时间等于当前时间。

2.当交易时间等于当前时间时，让flag为1表示当前有窗口在办理业务。

3.当到达时间（**MaxTime**）等于当前时间（**currentTime**）的时候调用**arrive()**函数让客户到达加入队列1（**Q1**）排队并在事件表（**Q\_en**）中加入到达事件，同时随机生成下一个客户到达时间（**MaxTime**）。

4.当窗口处于交易状态（flag=1）且队列1（**Q1**）不为空。则进行客户交易处理，当客户是存款类型的时候调用**putMoney()**进行存款操作，同时调用 **findAndDeal()**函数对队列2（**Q2**）进行检查是否有满足取款的客户并进行处理。当客户是取款类型的时候调用**getMoney()**进行取款处理。

每一次循环当前时间（**currentTime**）加1，直到银行营业结束退出循环，输出模拟结果，最后调用**closeBank()**函数结束程序。

**【各模块的调用关系】**

程序分成**主程序模块**、**存款模块**、**取款模块**、**检查处理模块**、**事件处理模块。主程序模块**负责调用**存款模块**、**取款模块**和**事件处理模块**。**存款模块**调用**事件处理模块、检查处理模块**。**取款模块**和**检查处理模块**调用**事件处理模块。**

1. **详细设计**

**【数据类型的实现】**

**队列抽象数据结构的实现**

//入队操作

void push(Queue &q,eventLink e){

if(e==NULL){//结点指针为空

return;

}

if(q->front==NULL){//队列为空

q->front=e;

q->rear=e;

}else{

q->rear->next=e;

q->rear=q->rear->next;

}

}

//销毁队列

void destoryQueue(Queue &q){

eventLink e,e1;

e=q->front;

if(q->front==q->rear==NULL){//队空

return;

}

while(e!=NULL){

e1=e->next;

free(e);

e=NULL;

e=e1;

}

}

//删除队首元素

void pop(Queue &q){

eventLink e;

e = q->front;

if(q->front==NULL){//队列为空

return;

}else if(q->front->next==NULL){//队列只有一个元素

q->front=q->rear=NULL;

}else{

q->front=q->front->next;

}

free(e);//释放空间

e=NULL;

}

//返回队首元素

eventLink get\_front(Queue &q){

return q->front;

}

**函数类型的实现**

/\*\*

结束函数

当银行营业结束的时候调用此函数，清理释放申请的空间

/

**void closeBank()**{

//调用destoryQueue()函数销毁三个队列Q1、Q2、Q\_en

**destoryQueue(Q1);**

**destoryQueue(Q2);**

**destoryQueue(Q\_en);**

}

/\*\*

到达函数

当客户到达时调用此函数，随机生成客户信息以及相应的到达事件，分别加入队列1和事件队列。

\*/

伪代码：

**void arrive(){**

**//**声明两个eventLink结点指针e、e1

**//**用malloc函数为两个结点指针分配空间

if(e为空或者e1为空){

//退出函数

}

//用(rand()%(2\*MaxMoney)-MaxMoney)随机生成e和e1的办理金额

//将当前时间（currentTime）作为e和e1的到达时间

//将（number）作为e和e1的序列号，number++

//将e1的类型设置为0（到达）

//将e加入队列1（Q1）

//将e1加入事件队列（Q\_en）

**}**

/\*\*

存款函数

当客户办理存款的时候调用此函数

\*/

伪代码：

**void putMoney(){**

**//**声明两个eventLink结点指针e、e1

**//**用malloc函数为结点指针e1分配空间

if(e1为空){

//退出函数

}

//用get\_front()函数获取队列1的队头元素

//更新银行金额total\_money

//利用获得到的对头元素信息设置离开事件e1

//e1的离开时间为客户的到达时间+随机的处理时间

//e1加入事件队列

//删除队列1对头元素

//增加客户的总逗留时间（客户的离开时间-客户的到达时间）

//接待客户人数加1

//客户的离开时间为交易时间

//将窗口设置为空闲状态

**}**

/\*\*

取款函数

当客户办理取款的时候调用此函数

\*/

伪代码：

**void getMoney(){**

**//**声明两个eventLink结点指针e、e1

**//**用malloc函数为结点指针e1分配空间

if(e1为空){

//退出函数

}

//用get\_front()函数获取队列1的队头元素

if(客户的取款金额大于银行现存金额){

//取款失败

//e1为队列2客户元素

//将客户从队列1删除，让客户进入队列2继续排队

}else{

//取款成功

//更新银行现存金额

//向事件队列添加相应的离开事件e1

//将客户从队列1删除

//接待客户人数加1

//增加客户的总逗留时间（客户的离开时间-客户的到达时间）

//客户的离开时间为交易时间

/ /将窗口设置为空闲状态

}

**}**

/\*\*

查找函数

查找队列q中第一个可以满足取款的结点并返回，若是查找失败则返回NULL

\*/

伪代码：

**eventLink search(Queue &q,int m){**

**//**声明两个eventLink结点指针e、e1

//取队列的队头元素e

while(队头元素不为空){//遍历队列

if(客户办理金额小于等于银行现存金额){

if(队列只有一个元素){

//队列头尾指针置空

//返回客户指针e1

}else{

//队头指针移动一位

//返回客户指针e1

}

}else{//资金不足，查找下一个客户

if(队列只有一个元素){

//不做操作

}else{

//将队头结点放到队尾

}

}

if(队列遍历了一遍){

//返回NULL

}

}

//返回NULL

**}**

/\*\*

处理函数

对通过【查找函数】找到的结点进行取款处理，生成相应的离开事件

\*/

伪代码：

**void findAndDeal(){**

**//**声明两个eventLink结点指针e、e1

**while(**调用search函数查找客户结点e**){**

**//**用malloc函数为结点指针e1分配空间

//更新银行现存款total\_money

//离开事件e1的离开时间为当前时间+随机处理时间

//设置离开事件e1加入事件表(Q\_en)

//接待客户人数增加

//交易时间加上处理时间

//增加客户总逗留时间（客户的离开时间-客户的到达时间）

//释放结点e的空间

**}**

**//**将交易状态设为空闲状态

**}**

/\*\*

初始化函数

初始化三个队列，为三个队列分配空间

\*/

**void initQueue(){**

//为三个队列Q1、Q2、Q\_en分配空间

if(Q1、Q2、Q\_en其中一个为空){

退出函数

}

//将三个队列的头尾指针置空

**}**

**主函数的实现**

伪代码：

**int main(){**

**while（**用户选择进入模拟程序**）{**

//用户输入模拟数据

//初始化三个队列

//随机产生到达时间（**MaxTime**）用来确定第一个到达的客户

**while(**当前时间<银行结业时间**){**

//当前时间加1

**if(**当交易时间<当前时间**){**

//让交易时间等于当前时间

**}**

**if(**交易时间==当前时间**){**

//让flag为1

**}**

**if(**到达时间==当前时间**){**

//调用到达函数arrive()

//随机生成下一个到达时间（**MaxTime**）

**}**

**if(**flag==1并且队列1不为空**){**

**if(**队列1队头客户办理存款**){**

//调用存款函数putMoney()

//调用查找函数findAndDeal()

**}else{**//办理取款

//调用取款函数getMoney()

**}**

**}**

**}**

//输出事件队列信息

**while(**事件队列队头不为空**){**//遍历事件队列

**if(**事件为离开类型**){**

//输出事件信息

**if(**事件为存款类型**){**

//成功存款人数加1

**}else{//**事件为取款类型

//成功取款人数加1

**}**

**}else{**//事件为到达类型

//输出事件信息

**if(**事件为存款类型**){**

//需要存款人数加1

**}else{//**事件为取款类型

//需要存款人数加1

**}**

**}**

//删除事件队列队头元素

**}**//事件队列遍历结束

**while(**队列1队头元素不为空**){**

//更新总逗留时间

//接待客户人数加一

//删除事件队列1队头元素

**}**

**while(**队列2队头元素不为空**){**

//更新总逗留时间

//接待客户人数加一

//删除事件队列2队头元素

**}**

//调用结束函数closeBank()

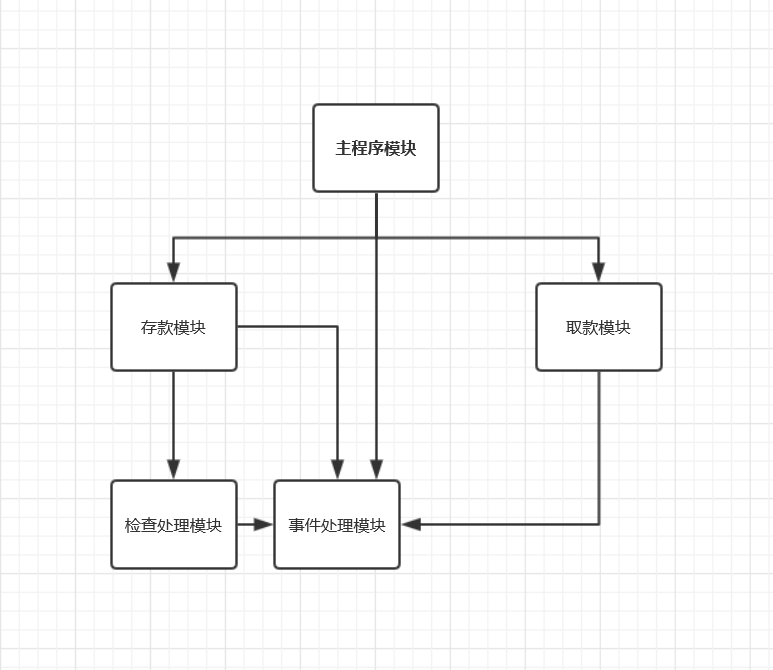
//输出最终模拟营业结果

**}**//退出循环

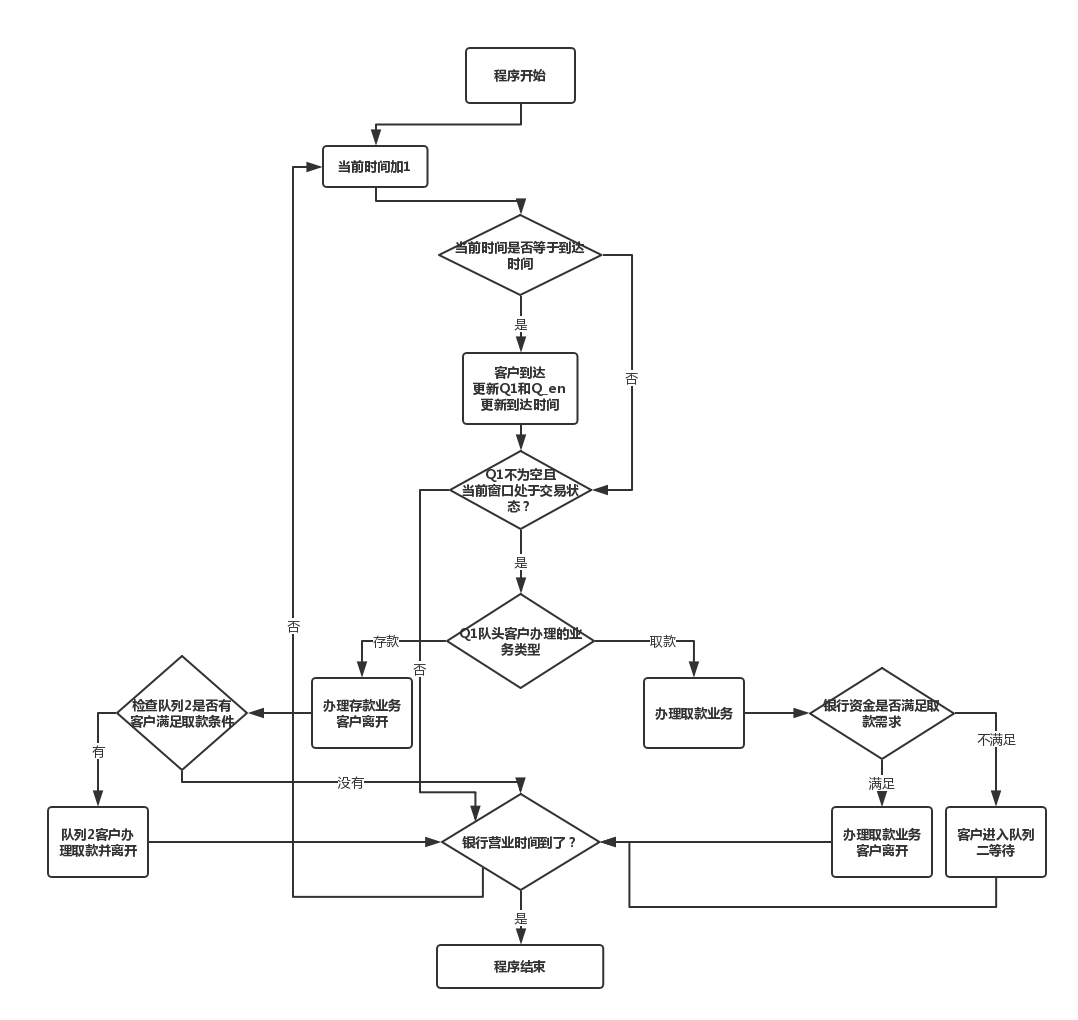
**}//main**

**【程序模块调用关系图】**

程序分成**主程序模块**、**存款模块**、**取款模块**、**检查处理模块**、**事件处理模块**。



**【银行业务模拟程序的流程图】**



1. **调试分析**

【调试过程中遇到的问题以及解决办法】

程序调试阶段中遇到了几个问题，下面将对遇到的问题进行描述以及给出解决办法

1. 问题：在运行程序的一开始就无法运行

解决过程：首先排除输入数据的错误，我取了几组正常的数据进行验证，排除是输入的数据的错误。

接着我在程序中设置调试输出语句进行调试，发现程序在调用存款函数的时候出错。经过检查发现是在使用节点的时候，忘记给结点指针分配空间。

解决方法：给待使用的结点指针分配空间

1. 问题： 在输出结果的时候发现需要存款和需要取款的人数之和有时候会小于总的客户接待数量

解决过程： 首先我是想到可能是在存款和取款函数里添加离开事件的时候忘记更新客户总数。在对存取款函数进行排查后发现并不是这个问题。接着我对更新需要存款和需要取款的人数进行检查，发现这两个变量的增加是符合逻辑的，并没有错误。最后我通过不断的调试和判断发现原来是在营业结束后没有对队列2里面还在排队的客户进行计算。

解决方法：在结束模拟营业后增加一个循环对队列2还存在的客户进行计算、更新总逗留时间并释放空间。

1. 问题：在程序运行到搜索函数的时候会出错，程序无法运行

解决过程：我对搜索函数进行调试发现函数陷入循环里面无法退出循环，在对程序的逻辑进行研究后我发现我的逻辑存在问题，在对队列2进行遍历的时候忘记在遍历一遍之后强制退出循环。

解决办法：在处理函数的循环语句中增加一个判断，当队列遍历一遍后退出寻环

**【算法的时空分析以及改进设想】**

下面给出程序中各个操作的时间复杂度和空间复杂度的分析

1. get\_front(Queue &q)//返回队首元素

由函数的操作可以看出此操作只需要返回队列头指针，所以时间复杂度为O(1)，空间复杂度也为O(1)

(2) push(Queue &q,eventLink e)//入队操作

由函数的操作可以看出来函数里没有循环语句只有简单的判断语句，没有额外的空间，只有指针的赋值操作。所以时间复杂度为O(1)，空间复杂度也为O(1)

（3） destoryQueue(Queue &q)// 销毁队列

在这个函数中存在循环用以遍历队列，所以这个操作的时间复杂度为O(n),函数 没有额外的空间,所以空间复杂度为O(1)

(4) pop(Queue &q)// 删除队首元素

通过分析可以知道函数的操作可以看出来函数里没有循环语句只有简单的判断语句，没有额外的空间，只有指针的操作。所以时间复杂度为O(1)，空间复杂度也为O(1)

(5) closeBank()//结束函数

函数只是调用了三个函数而调用的函数时间复杂度为O(n)，所以函数时间复杂度为O(n)，空间复杂度为O(1)

(6) arrive();//到达函数

通过对函数操作的分析可以知道，这个函数与问题规模无关所以时间复杂度为O(1)，空间复杂度也为O(1)

(7) putMoney()//存款函数

通过对函数操作的分析可以知道，这个函数与问题规模无关所以时间复杂度为O(1)，空间复杂度也为O(1)

(8) getMoney()//取款函数

通过对函数操作的分析可以知道，这个函数与问题规模无关所以时间复杂度为O(1)，空间复杂度也为O(1)

(9) search(Queue &q,int m)// 搜索函数

通过分析可以知道函数里面存在一个循环用来遍历队列，所以这个操作的时间复杂度为O(n),函数 没有额外的空间,所以空间复杂度为O(1)

(10) findAndDeal()//处理函数

通过分析可以知道函数里面存在一个循环用来调用查找函数，所以这个操作的时间复杂度为O(n),函数 没有额外的空间,所以空间复杂度为O(1)

(11) initQueue()//队列初始化

通过对函数操作的分析可以知道，这个函数与问题规模无关所以时间复杂度为O(1)，空间复杂度也为O(1)

**【经验和体会】**

一开始在选这个课题的时候其实心里挺没底的，因为没有完整的用c语言做过一个课题。在看完这个课题的要求之后发现这个题目主要是需要用队列进行实现。所以我先把队列这部分的知识认真的复习了一遍。之后通过反复看设计需求大致了解了题目的设计要求，在对要求存在疑问的地方我也通过和同学讨论逐渐解决了疑惑。

在对要求有了一个大致的了解后，我通过写伪代码和思维导图逐渐理清设计思路。接着就是通过题目具体的要求设计了数据结构，和一些基本模块。在经过认真编写后程序的架构基本完成。

在调试阶段遇到的问题比较多，当然这个阶段是必不可少的也是十分重要的。在调试的过程中我不仅发现的在设计程序的过程中出现的问题，同时也提高了自己发现问题解决问题的能力以及对代码的分析设计能力。在调试过程中我也曾经怀疑过自己，但是我不曾放弃。在一遍又一遍的调试修改过程中，代码功能逐渐完善，我也更加体会到编程的乐趣。

通过这次课程设计我提高了需求的分析能力、对程序的设计能力、对错误的发现修改能力。我觉得这次课程设计让我受益良多。

1. **用户使用说明**

本程序的功能是根据用户的输入参量进行银行存取款业务的模拟

在使用本程序的时候首先出现两个选项你进行选择：1.进入银行模拟系统 0.退出

你需要对此做出选择

如果你想退出银行模拟系统则按下0

如果你想进行模拟则需要按下1

在你进入模拟系统之后你需要根据提示输入模拟参量

本程序需要用户自行输入以下参量

**银行初始存款（total\_money）**

**银行营业时间(closeTime)**

**客户最大到达时间间隔(arriveMaxTime)**

**客户最小到达时间间隔(arriveMinTime)**

**业务最大处理时间(dealMaxTime)**

**业务最小处理时间(dealMinTime)**

**最大交易额上限(MaxMoney)**

接下来将对各个参量的作用和输入范围进行说明

---------------------------------------------------------------------------

**银行初始存款（total\_money）：total\_money是银行的初始存款金额，这个参量必须是一个大于零的整数。【单位为(元)】**

**银行营业时间(closeTime)：closeTime是银行的营业时间用来标识银行的关闭时间，但当前时间等于closeTime的时候，银行关闭。这个参量必须是一个大于零的整数。【单位为(分钟)】**

**客户最大到达时间间隔(arriveMaxTime)：arriveMaxTime是客户的到达时间距离上一个客户的到达时间间隔的上限。这个参量必须是一个大于零的整数且不能大于银行营业时间(closeTime)。【单位为(分钟)】**

**客户最小到达时间间隔(arriveMinTime)：arriveMinTime是客户的到达时间距离上一个客户的到达时间间隔的下限。这个参量必须是一个大于零的整数，要求不能大于客户最大到达时间间隔(arriveMaxTime)且不能大于银行营业时间(closeTime)。【单位为(分钟)】**

**业务最大处理时间(dealMaxTime)：dealMaxTime是客户办理业务所用时间的上限。这个参量必须是一个大于零的整数且不能大于银行营业时间(closeTime)。【单位为(分钟)】**

**业务最小处理时间(dealMinTime)：dealMinTime是客户办理业务所用时间的下限。这个参量必须是一个大于零的整数，要求不能大于业务最大处理时间(dealMaxTime)且不能大于银行营业时间(closeTime)。【单位为(分钟)】**

**最大交易额上限(MaxMoney)：MaxMoney是客户办理金额的上限。这个参量必须是一个大于零的整数。【单位为(元)】**

---------------------------------------------------------------------------

在输入合法的参量后程序就会执行，在执行成功后dos界面会输出银行模拟的结果。输出的结果包括客户到达的事件情况以及成功办理业务离开的事件情况；以及以下几个数据。

**输出：**

**需要存款的客户人数(n\_p)**

**需要取款的客户人数(n\_g)**

**成功存款的客户人数(s\_p)**

**成功取款的客户人数(s\_g)**

**客户逗留总时间(totalTime)**

**接待客户总数(counter)**

**客户逗留平均时间(totalTime/counter)**

**银行结业余额(total\_money)**

1. **测试结果**

**下面给出完整的测试方案包括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。**

**测试1**

**输入：**

银行初始存款（total\_money）：10000

银行营业时间(closeTime)：600

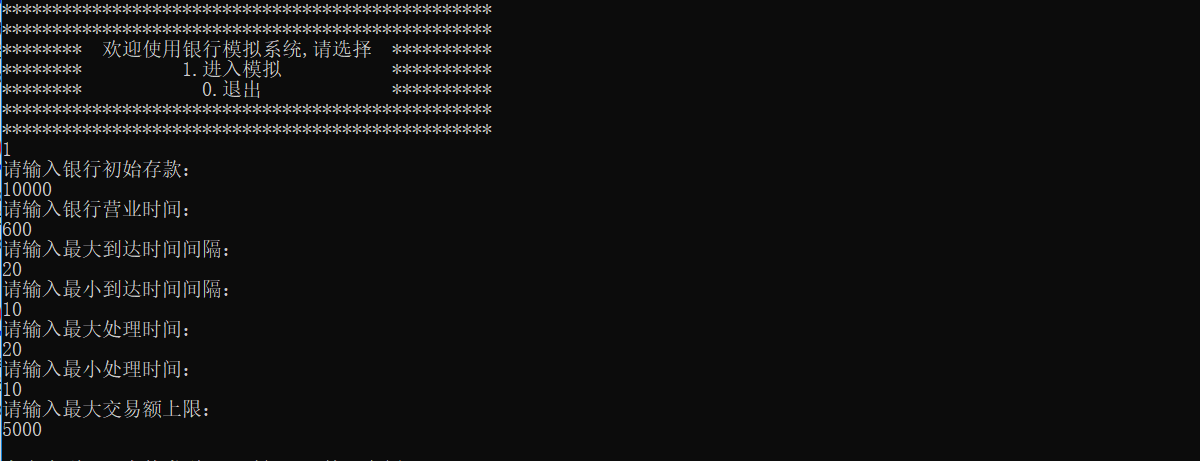
客户最大到达时间间隔(arriveMaxTime)：20

客户最小到达时间间隔(arriveMinTime)：10

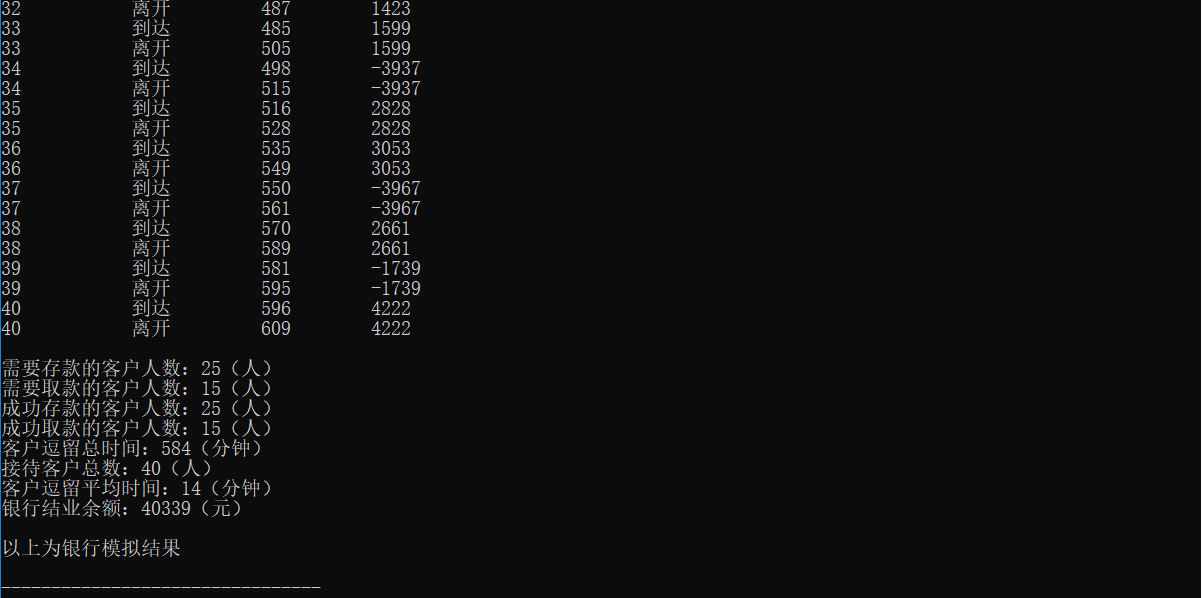
业务最大处理时间(dealMaxTime)：20

业务最小处理时间(dealMinTime)：10

最大交易额上限(MaxMoney)“5000

****

**输出：**

****

**测试2**

**输入：**

银行初始存款（total\_money）：10000

银行营业时间(closeTime)：600

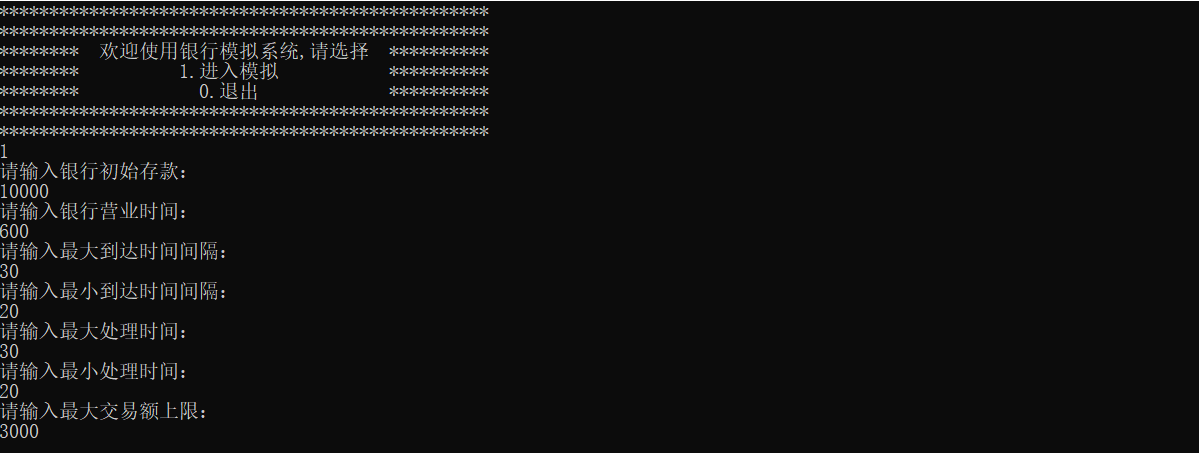
客户最大到达时间间隔(arriveMaxTime)：30

客户最小到达时间间隔(arriveMinTime)：20

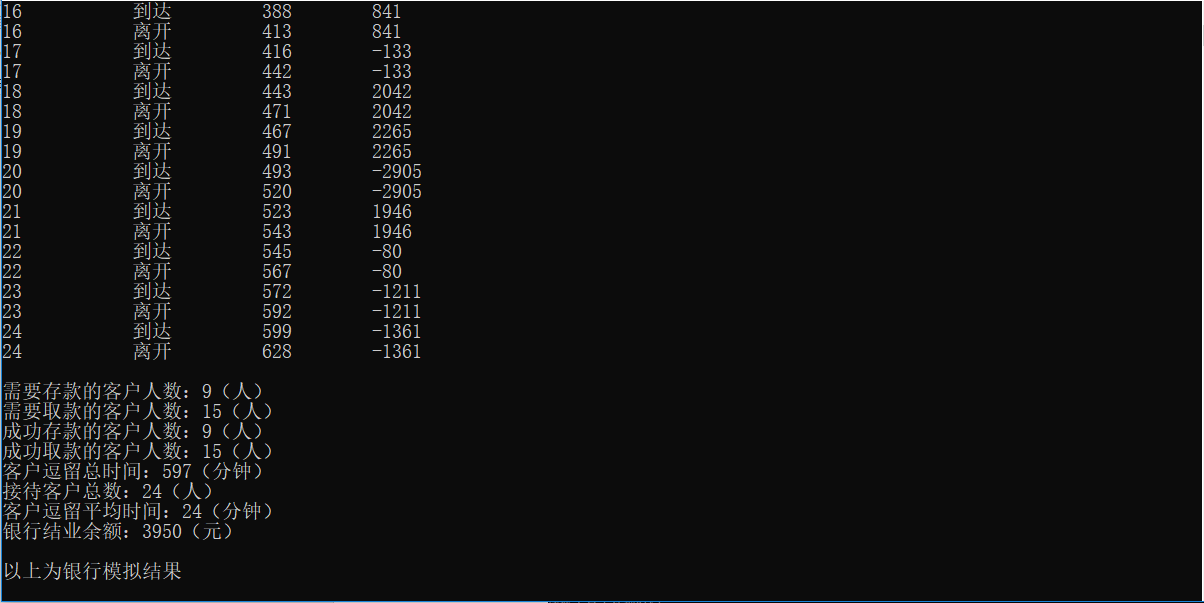
业务最大处理时间(dealMaxTime)：30

业务最小处理时间(dealMinTime)：20

最大交易额上限(MaxMoney)“3000

****

**输出：**

****

**测试3**

**（这一次测试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）**

**输入：**

银行初始存款（total\_money）：10000

银行营业时间(closeTime)：600

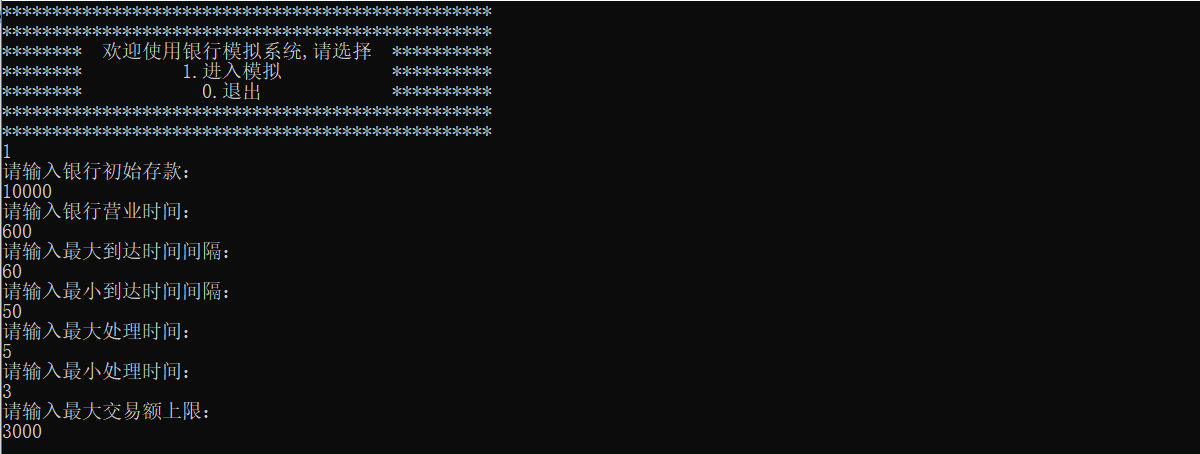
客户最大到达时间间隔(arriveMaxTime)：60

客户最小到达时间间隔(arriveMinTime)：50

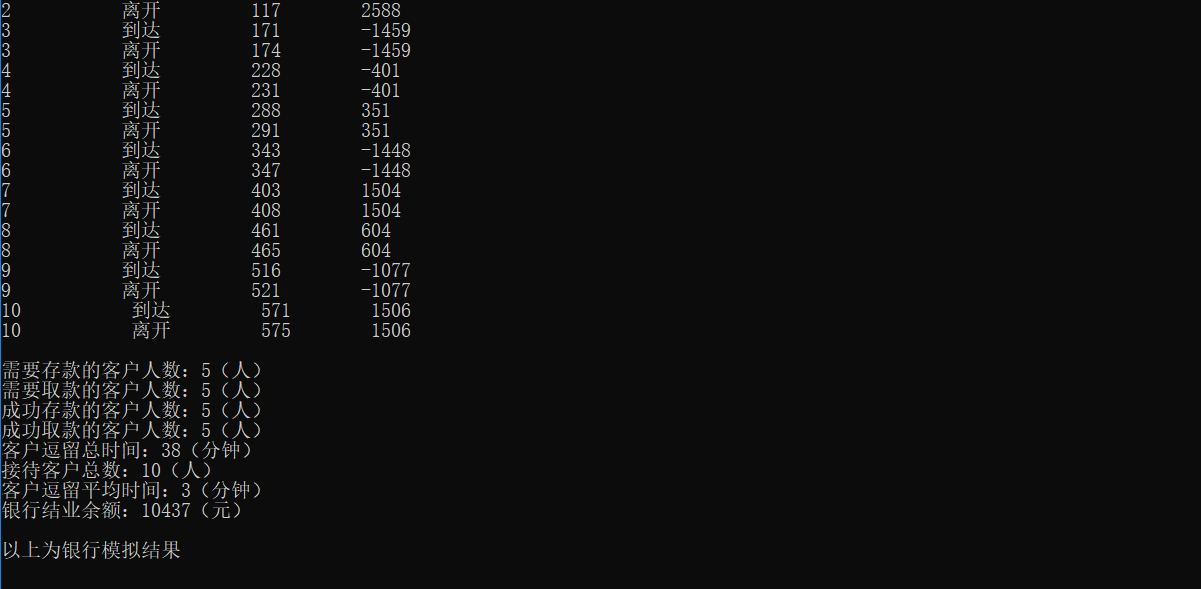
业务最大处理时间(dealMaxTime)：5

业务最小处理时间(dealMinTime)：3

最大交易额上限(MaxMoney)“3000

****

**输出：**

****

**测试4**

**（这一次同样测试两个到达事件的间隔时间很长，而客户的交易时间很短）**

**输入：**

银行初始存款（total\_money）：10000

银行营业时间(closeTime)：600

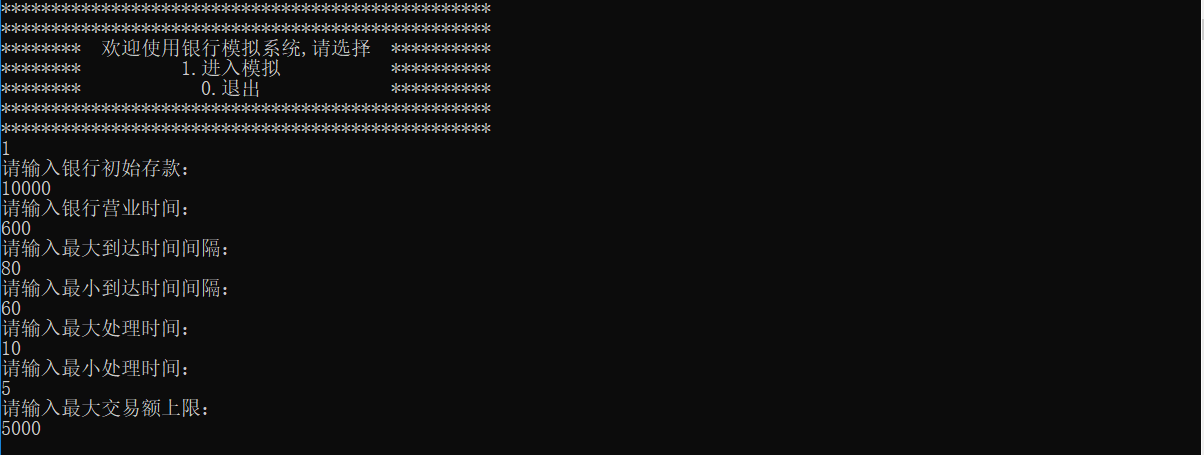
客户最大到达时间间隔(arriveMaxTime)：80

客户最小到达时间间隔(arriveMinTime)：60

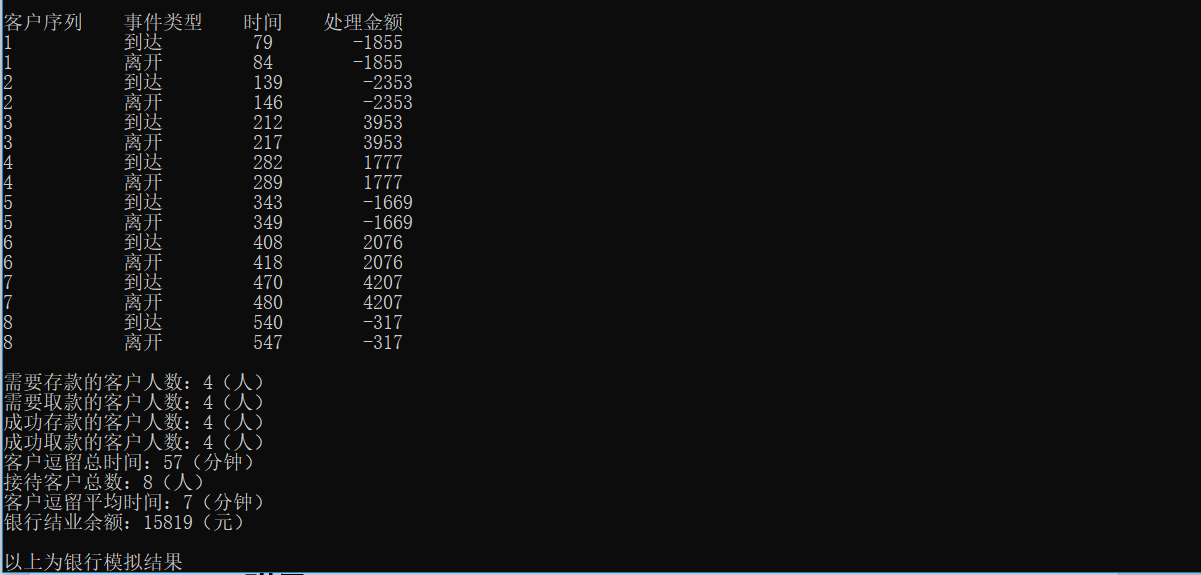
业务最大处理时间(dealMaxTime)：10

业务最小处理时间(dealMinTime)：5

最大交易额上限(MaxMoney)“5000

****

**输出：**

****

**测试5**

**（这一次测试两个到达事件的间隔时间很短，而客户的交易时间很长）**

**输入：**

银行初始存款（total\_money）：10000

银行营业时间(closeTime)：600

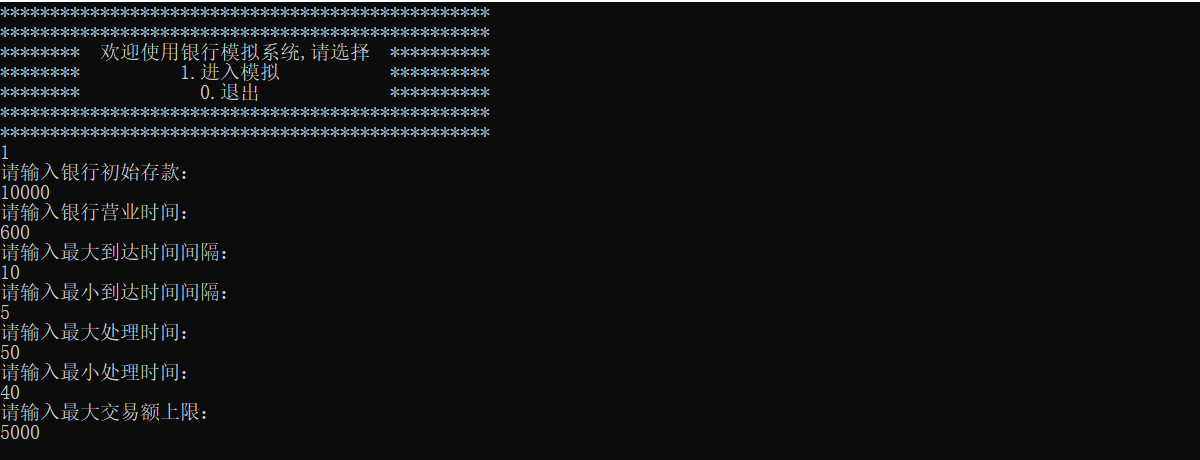
客户最大到达时间间隔(arriveMaxTime)：10

客户最小到达时间间隔(arriveMinTime)：5

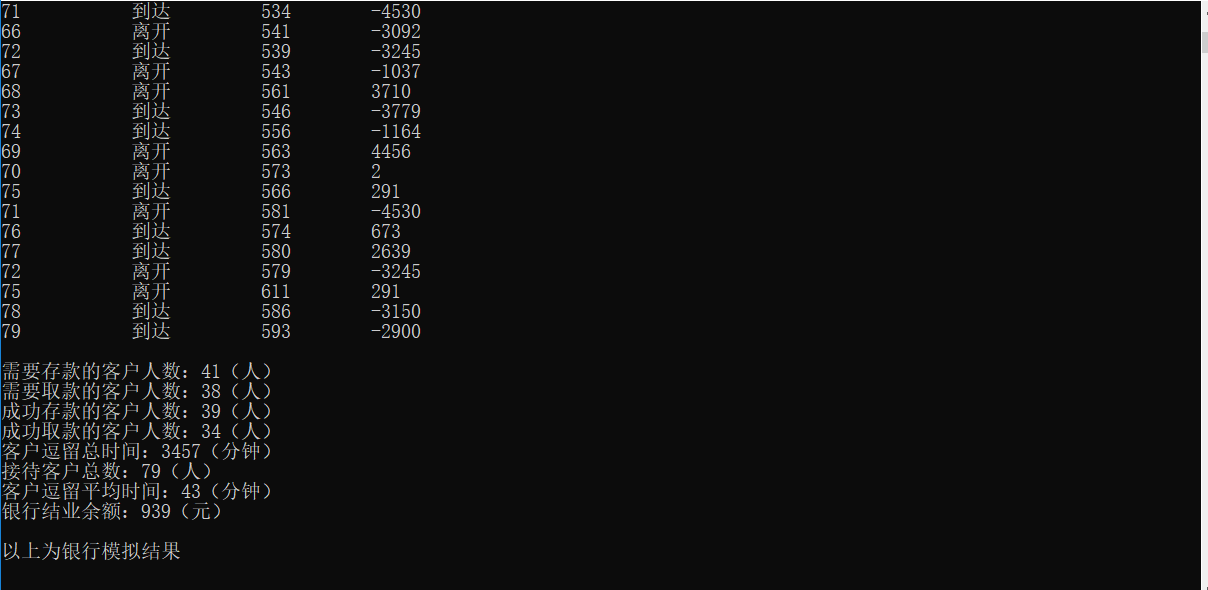
业务最大处理时间(dealMaxTime)：50

业务最小处理时间(dealMinTime)：40

最大交易额上限(MaxMoney)“5000

****

**输出：**

****

**测试6**

**（这一次同样测试两个到达事件的间隔时间很短，而客户的交易时间很长）**

**输入：**

银行初始存款（total\_money）：10000

银行营业时间(closeTime)：600

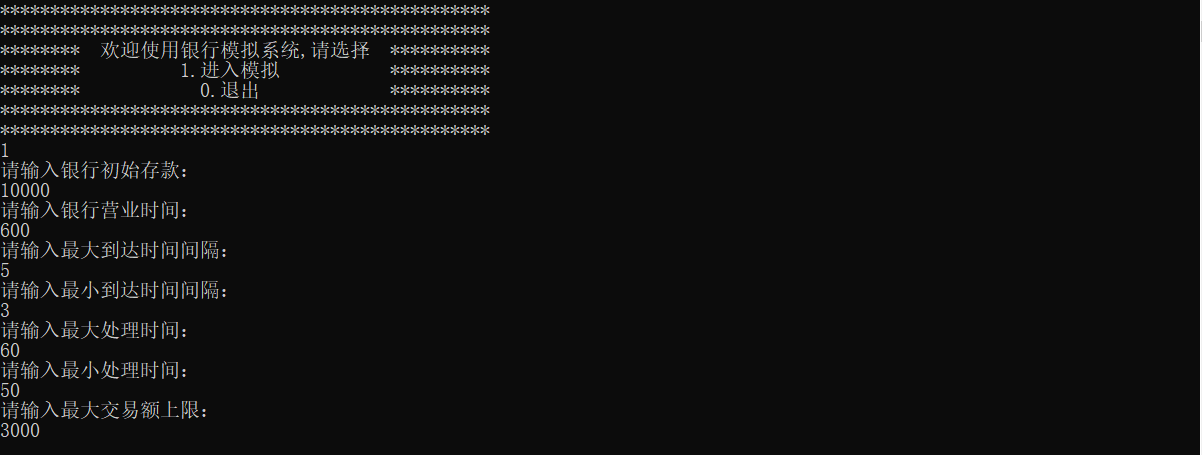
客户最大到达时间间隔(arriveMaxTime)：5

客户最小到达时间间隔(arriveMinTime)：3

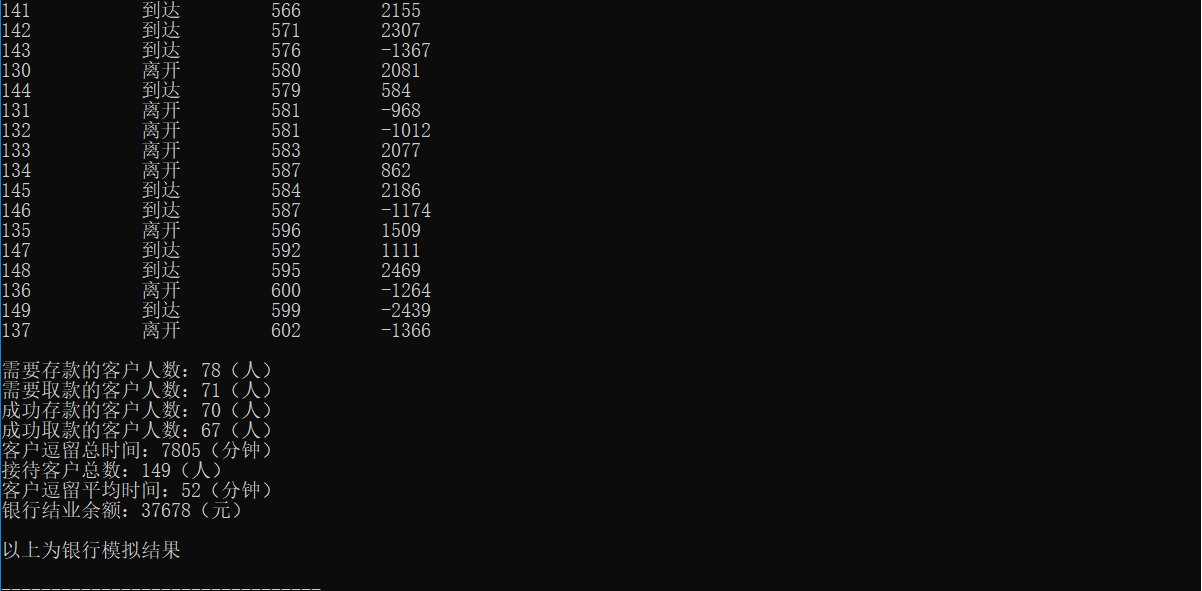
业务最大处理时间(dealMaxTime)：60

业务最小处理时间(dealMinTime)：50

最大交易额上限(MaxMoney)“3000

****

**输出：**

****

**测试7**

**（这一次测试非法数据的输入以及程序的反馈）**

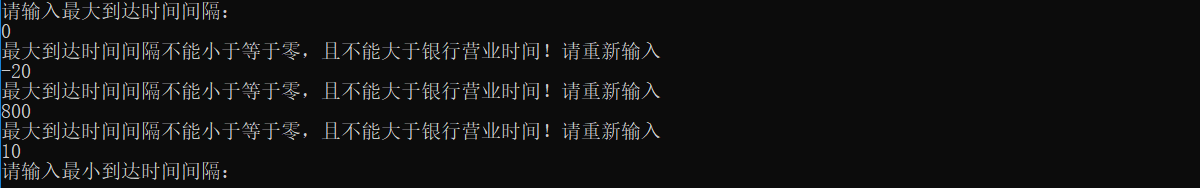
**测试银行初始存款（total\_money）的非法输入**

****

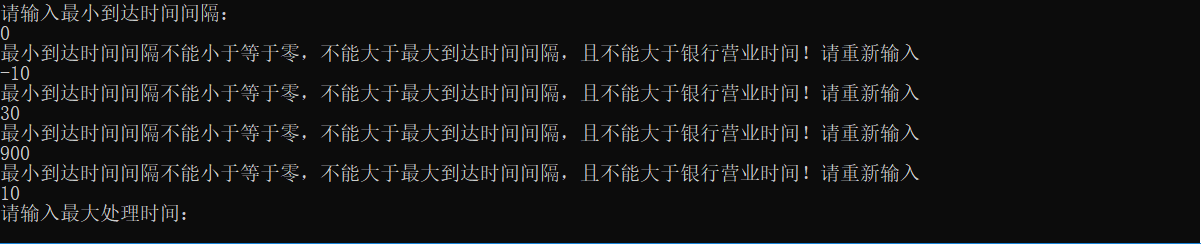
**测试银行营业时间(closeTime)的非法输入**

****

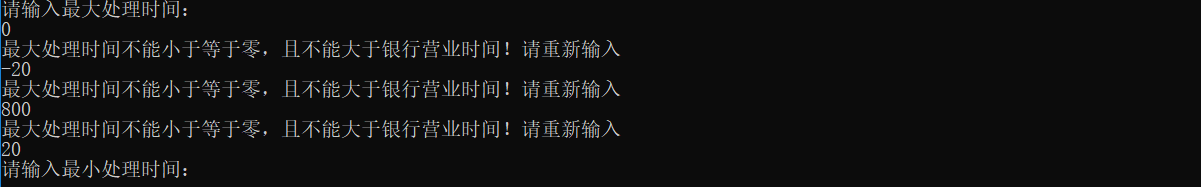
**测试客户最大到达时间间隔(arriveMaxTime)的非法输入**

****

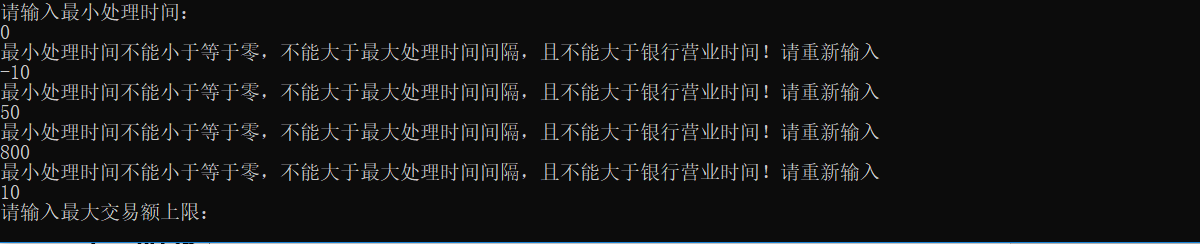
**测试客户最小到达时间间隔(arriveMinTime)的非法输入**

****

**测试业务最大处理时间(dealMaxTime)的非法输入**

****

**测试业务最小处理时间(dealMinTime)的非法输入**

****

**测试最大交易额上限(MaxMoney)的非法输入**

****

1. **附录**

//带注释的源程序

**#include<stdio.h>**

**#include<stdlib.h>**

**#include<time.h>**

**/\*\***

**客户\事件结构体**

**\*/**

**typedef struct event{**

**int type;//事件类型到达或者离开 0为到达，1为离开**

**int arriveTime;//到达时间**

**int leaveTime;//离开时间**

**int money;//存款数，正数为存款，复数为取款**

**int num;//客户编号**

**event\* next;**

**}event,\*eventLink;**

**/\*\***

**队列结构体**

**\*/**

**typedef struct QNode{**

**eventLink front;//队头指针**

**eventLink rear; //队尾指针**

**}QNode,\*Queue;**

**int total\_money;//银行现存款**

**int closeTime;//银行结束营业的时间**

**int arriveMaxTime;//两个到达事件的时间间隔上限**

**int arriveMinTime; //两个到达事件的时间间隔下限**

**int dealMaxTime;//客户交易之间的时间上限**

**int dealMinTime; //客户交易之间的时间下限**

**int MaxMoney;//交易额上限**

**int currentTime=0;//当前时间**

**int totalTime;//客户逗留总时间**

**int counter=0;//客户总数**

**int number=1;//客户初始号码**

**int flag=1;//判断是否有窗口在处理**

**int TimeOfDeal;//交易时间**

**int MaxTime;//到达时间**

**Queue Q\_en;//事件队列**

**Queue Q1;//队列一**

**Queue Q2;//队列二**

**//返回队首元素**

**eventLink get\_front(Queue &q){**

**return q->front;**

**}**

**//入队**

**void push(Queue &q,eventLink e){**

**if(e==NULL){//结点指针为空**

**return;**

**}**

**if(q->front==NULL){//队列为空**

**q->front=e;**

**q->rear=e;**

**}else{**

**q->rear->next=e;**

**q->rear=q->rear->next;**

**}**

**}**

**//销毁队列**

**void destoryQueue(Queue &q){**

**eventLink e,e1;**

**e=q->front;**

**if(q->front==q->rear==NULL){//队空**

**return;**

**}**

**while(e!=NULL){**

**e1=e->next;**

**free(e);**

**e=NULL;**

**e=e1;**

**}**

**}**

**//删除队首元素**

**void pop(Queue &q){**

**eventLink e;**

**e = q->front;**

**if(q->front==NULL){//队列为空**

**return;**

**}else if(q->front->next==NULL){//队列只有一个元素**

**q->front=q->rear=NULL;**

**}else{**

**q->front=q->front->next;**

**}**

**if(e!=NULL){**

**free(e);//释放空间**

**e=NULL;**

**}**

**}**

**//结束函数,营业时间到**

**void closeBank(){**

**destoryQueue(Q1);**

**destoryQueue(Q2);**

**destoryQueue(Q\_en);**

**}**

**//到达函数，随机产生顾客进入队列一同时，产生到达事件**

**void arrive(){**

**eventLink e,e1;**

**e = (eventLink)malloc(sizeof(event));//分配空间**

**e1 = (eventLink)malloc(sizeof(event));//分配间**

**if(e==NULL||e1==NULL){//分配空间失败**

**return;**

**}**

**e1->money=e->money = (rand()%(2\*MaxMoney)-MaxMoney);//随机交易金额**

**e1->arriveTime=e->arriveTime = currentTime;//当前时间为到达时间**

**e1->num=e->num = number;**

**e->next=NULL;**

**e1->next=NULL;**

**push(Q1,e);//加入第一队列**

**e1->type=0;//到达事件**

**push(Q\_en,e1);//加入事件队列**

**number++; //客户人数增加**

**}**

**//存款函数**

**void putMoney(){**

**eventLink e,e1;**

**int temptime;**

**e1 = (eventLink)malloc(sizeof(event));//分配空间**

**if(e1==NULL){//分配空间失败**

**return;**

**}**

**e = get\_front(Q1);//对头元素**

**total\_money += e->money;//存款**

**e1->type=1;//离开事件**

**e1->num=e->num;**

**e1->money=e->money;**

**temptime=rand()%(dealMaxTime-dealMinTime+1)+dealMinTime;//随机生成处理时间**

**e1->leaveTime = (e->arriveTime+temptime);//离开时间为到达时间+处理时间**

**e1->next=NULL;**

**push(Q\_en,e1);//离开事件加入队列**

**counter++;**

**totalTime+=e1->leaveTime-e->arriveTime;//刷新交易时间**

**pop(Q1);//Q1出队**

**TimeOfDeal=e1->leaveTime;//交易时间为客户的离开时间**

**flag=0;//窗口没有交易需要处理**

**}**

**void getMoney(){//取款函数**

**eventLink e,e1;**

**e1 = (eventLink)malloc(sizeof(event));//分配空间**

**if(e1==NULL){//分配空间失败**

**return;**

**}**

**e = get\_front(Q1);//取Q1对头元素**

**if((-(e->money))>total\_money){//资金不足,加入第二队列等待**

**e1->arriveTime = e->arriveTime;**

**e1->money = e->money;**

**e1->num = e->num;**

**e1->next=NULL;**

**push(Q2,e1);//客户加入第二队列**

**pop(Q1);//从Q1删除客户**

**}else{//资金足够，执行取款**

**total\_money +=e->money;//取款**

**e1->type=1;//离开事件**

**e1->num=e->num;**

**e1->money=e->money;**

**e1->leaveTime= (e->arriveTime+rand()%(dealMaxTime-dealMinTime+1)+dealMinTime);//离开时间为到达时间+随机处理时间**

**e1->next=NULL;**

**push(Q\_en,e1);//离开事件加入事件队列**

**TimeOfDeal=e1->leaveTime;//交易时间为客户的离开时间**

**counter++;//客户人数增加**

**totalTime+=e1->leaveTime-e->arriveTime;//刷新客户总逗留时间**

**pop(Q1);//客户离开**

**flag=0;**

**}**

**}**

**//搜索函数，搜索队列中是否有可以处理的元素**

**eventLink search(Queue &q,int m){**

**eventLink e,e1;**

**e = q->front;**

**while(q->front!=NULL){//遍历队列**

**if((-(q->front->money))<=total\_money){//资金充足，当前客户可取款**

**if(q->front==q->rear){**

**e1=q->front;**

**q->front=q->rear=NULL;**

**return e1;**

**}else{**

**e1=q->front;**

**q->front=q->front->next;**

**return e1;**

**}**

**}else{//资金不足，查找下一个客户**

**if(q->front==q->rear){**

**}else{**

**q->rear->next=q->front;**

**q->rear=q->rear->next;**

**q->front=q->rear->next;**

**q->rear->next=NULL;**

**}**

**}**

**if(q->front==e){**

**return NULL;**

**}**

**}**

**return NULL;**

**}**

**//处理函数，在队列中寻找可处理的元素并处理**

**void findAndDeal(){**

**eventLink e,e1;**

**int randTemp;**

**while((e=search(Q2,total\_money))!=NULL){**

**e1 = (eventLink)malloc(sizeof(event));//分配空间**

**if(e1==NULL){//分配空间失败**

**return;**

**}**

**total\_money+=e->money;//取款**

**e1->type=1;//离开事件**

**e1->num=e->num;**

**e1->money=e->money;**

**randTemp = rand()%(dealMaxTime-dealMinTime+1)+dealMinTime;//随机处理时间**

**e1->leaveTime=currentTime+ randTemp;//离开时间=当前时间+处理时间**

**e1->next=NULL;**

**push(Q\_en,e1);//加入事件队列**

**TimeOfDeal+=randTemp;//刷新交易时间**

**counter++;//客户人数增加**

**totalTime+=e1->leaveTime-e->arriveTime;//刷新总逗留时间**

**free(e);//释放空间**

**e=NULL;**

**}**

**flag=0;**

**}**

**//队列初始化**

**void initQueue(){**

**Q1 = (Queue)malloc(sizeof(QNode));**

**Q2 = (Queue)malloc(sizeof(QNode));**

**Q\_en = (Queue)malloc(sizeof(QNode));**

**if(Q1==NULL||Q2==NULL||Q\_en==NULL){**

**return;**

**}**

**Q1->front=Q1->rear=NULL;**

**Q2->front=Q2->rear=NULL;**

**Q\_en->front=Q\_en->rear=NULL;**

**}**

**int main(){**

**printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");**

**printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");**

**printf("\*\*\*\*\*\*\*\* 17级计算机科学与技术（5）班 \*\*\*\*\*\*\*\*\*\*\n");**

**printf("\*\*\*\*\*\*\*\* 学号：3117004622 \*\*\*\*\*\*\*\*\*\*\n");**

**printf("\*\*\*\*\*\*\*\* 刘友滨 \*\*\*\*\*\*\*\*\*\*\n");**

**printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");**

**printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");**

**printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");**

**printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");**

**printf("\*\*\*\*\*\*\*\* 欢迎使用银行模拟系统,请选择 \*\*\*\*\*\*\*\*\*\*\n");**

**printf("\*\*\*\*\*\*\*\* 1.进入模拟 \*\*\*\*\*\*\*\*\*\*\n");**

**printf("\*\*\*\*\*\*\*\* 0.退出 \*\*\*\*\*\*\*\*\*\*\n");**

**printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");**

**printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");**

**int n;**

**int s\_p,s\_g,n\_p,n\_g;**

**scanf("%d",&n);**

**while(n==1){**

**srand(time(NULL));**

**printf("请输入银行初始存款：\n");**

**scanf("%d",&total\_money);**

**while(total\_money<=0){**

**printf("银行初始金额不能小于等于零！请重新输入\n");**

**scanf("%d",&total\_money);**

**}**

**printf("请输入银行营业时间：\n");**

**scanf("%d",&closeTime);**

**while(closeTime<=0){**

**printf("银行营业时间不能小于等于零！请重新输入\n");**

**scanf("%d",&closeTime);**

**}**

**printf("请输入最大到达时间间隔：\n");**

**scanf("%d",&arriveMaxTime);**

**while(arriveMaxTime<=0||arriveMaxTime>closeTime){**

**printf("最大到达时间间隔不能小于等于零，且不能大于银行营业时间！请重新输入\n");**

**scanf("%d",&arriveMaxTime);**

**}**

**printf("请输入最小到达时间间隔：\n");**

**scanf("%d",&arriveMinTime);**

**while(arriveMinTime<=0||arriveMinTime>arriveMaxTime||arriveMinTime>closeTime){**

**printf("最小到达时间间隔不能小于等于零，不能大于最大到达时间间隔，且不能大于银行营业时间！请重新输入\n");**

**scanf("%d",&arriveMinTime);**

**}**

**printf("请输入最大处理时间：\n");**

**scanf("%d",&dealMaxTime);**

**while(dealMaxTime<=0||dealMaxTime>closeTime){**

**printf("最大处理时间不能小于等于零，且不能大于银行营业时间！请重新输入\n");**

**scanf("%d",&dealMaxTime);**

**}**

**printf("请输入最小处理时间：\n");**

**scanf("%d",&dealMinTime);**

**while(dealMinTime<=0||dealMinTime>dealMaxTime||dealMinTime>closeTime){**

**printf("最小处理时间不能小于等于零，不能大于最大处理时间间隔，且不能大于银行营业时间！请重新输入\n");**

**scanf("%d",&dealMinTime);**

**}**

**printf("请输入最大交易额上限：\n");**

**scanf("%d",&MaxMoney);**

**while(MaxMoney<=0){**

**printf("最大交易额上限不能小于等于零！请重新输入\n");**

**scanf("%d",&MaxMoney);**

**}**

**printf("\n");**

**MaxTime+=rand()%(arriveMaxTime-arriveMinTime+1)+arriveMinTime;**

**initQueue();//队列初始化**

**while(currentTime<closeTime){**

**++currentTime;**

**if(TimeOfDeal<currentTime){**

**TimeOfDeal=currentTime;**

**}**

**if(TimeOfDeal==currentTime){**

**flag=1;//有窗口在处理交易**

**}**

**if(MaxTime==currentTime) {**

**arrive();**

**MaxTime+=rand()%(arriveMaxTime-arriveMinTime+1)+arriveMinTime;**

**}**

**if(flag==1&&Q1->front!=NULL){**

**if(Q1->front->money>=0){**

**putMoney();//调用存款函数**

**findAndDeal();//每存完一次款就调用查找函数，检查队列2**

**}else{**

**getMoney();//调用取款函数**

**}**

**}**

**}**

**printf("客户序列 事件类型 时间 处理金额\n");**

**s\_p=0;**

**s\_g=0;**

**n\_p=0;**

**n\_g=0;**

**while(Q\_en->front!=NULL){**

**if(Q\_en->front->type==1){//离开**

**printf("%d %s %d %d\n",Q\_en->front->num,"离开",Q\_en->front->leaveTime,Q\_en->front->money);**

**if(Q\_en->front->money>=0){**

**s\_p++;//成功存款**

**}else{**

**s\_g++; //成功取款**

**}**

**}else{**

**printf("%d %s %d %d\n",Q\_en->front->num,"到达",Q\_en->front->arriveTime,Q\_en->front->money);**

**if(Q\_en->front->money>=0){**

**n\_p++;//需要存款**

**}else{**

**n\_g++; //需要取款**

**}**

**}**

**pop(Q\_en);**

**}**

**while(Q1->front!=NULL)**

**{ totalTime+=(closeTime-Q1->front->arriveTime);**

**counter++;**

**eventLink e;**

**e=get\_front(Q1);**

**pop(Q1);**

**}**

**while(Q2->front!=NULL)**

**{ totalTime+=(closeTime-Q2->front->arriveTime);**

**counter++;**

**eventLink e;**

**e=get\_front(Q1);**

**pop(Q2);**

**}**

**closeBank();//营业时间到调用结束函数**

**printf("\n");**

**printf("需要存款的客户人数：%d（人）\n",n\_p);**

**printf("需要取款的客户人数：%d（人）\n",n\_g);**

**printf("成功存款的客户人数：%d（人）\n",s\_p);**

**printf("成功取款的客户人数：%d（人）\n",s\_g);**

**printf("客户逗留总时间：%d（分钟）\n",totalTime);**

**printf("接待客户总数：%d（人）\n",counter);**

**printf("客户逗留平均时间：%d（分钟）\n",totalTime/counter);**

**printf("银行结业余额：%d（元）\n",total\_money);**

**printf("\n");**

**printf("以上为银行模拟结果\n");**

**break;**

**}**

**system("pause");**

**return 0;**

**}**