



**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（2）班**

**学 号 3122004665**

**学生姓名 王展锐**

**指导教师 苏 庆**

**2023 年 12 月 20 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。

**[基本要求]**

利用动态存储结构实现模拟，即利用**C**语言的动态分配函数malloc和free。

**[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。

**[选做内容]（已完成）**

自己实现动态数据类型。例如对于客户结点，定义pool为

CustNode pool[MAX];

// 结构类型CustNode含四个域: arrtime,durtime,amount,next

或者定义四个同样长的，以上述域名为名字的数组。初始时，将所有分量的next域链接起来，形成一个静态链栈，设置一个栈顶元素下标指示量top，top=0表示栈空。动态存储分配函数可以取名为myMalloc，其作用是出栈，将栈顶元素的下标返回。若返回的值为0，则表示无空间可分配。归还函数可取名为myFree，其作用是把该分量入栈。用**FORTRAN**和**BASIC**等语言实现时只能如此地自行组织。

**[输入的形式和输入值的范围]**

1、输入形式：

由用户在终端进行输入

int total;//银行余额

int max\_money;//最大交易金额

int RunTime； //银行营业时间

int MaxTimeGap; //客户到达的最大时间间隔

int MinTimeGap; //客户到达的最小时间间隔

int MaxTradeGap; //最大交易时间

int MinTradeGap; //最小交易时间

**2、输入值范围：**

要求所有输入值均大于0，且最小值不能大于最大值，所有时间值均不可大于银行营业时间

[输出的形式]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

当前时间是:0

客户序号：1 客户交易需要时间：14 客户需要处理的钱：2675 银行余额：8675

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

当前时间是:18

客户序号：2 客户交易需要时间：11 客户需要处理的钱：-1434 银行余额：7241

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

当前时间是:32

客户序号：3 客户交易需要时间：19 客户需要处理的钱：2228 银行余额：9469

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

当前时间是:36

客户序号：4 客户交易需要时间：6 客户需要处理的钱：918 银行余额：9469

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

当前时间是:42

客户序号：5 客户交易需要时间：13 客户需要处理的钱：401 银行余额：9469

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

当前时间是:48

客户序号：6 客户交易需要时间：3 客户需要处理的钱：2684 银行余额：9469

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

当前时间是:54

客户序号：7 客户交易需要时间：12 客户需要处理的钱：-2323 银行余额：10387

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\* 时间表信息： \*

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\* 全部用户数: 7 人 \*

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\* 总共消耗时间: 118.000000 分钟 \*

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\* 成功人数: 4 人 \*

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\* 取款/借款成功人数: 1 人 \*

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\* 存款/还款成功人数: 3 人 \*

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\* 客户平均逗留时间： 16.857143 分钟 \*

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

时间表一览：

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

| 客户序号： | 事件类型： | 事件触发时间： | 事件任务： | 事件处理金额： |

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

| 1 | 到达 | 0 | 存款/还款 | 2675 |

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

| 1 | 离开 | 14 | 存款/还款 | 2675 |

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

| 2 | 到达 | 18 | 取款/借款 | -1434 |

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

| 2 | 离开 | 29 | 取款/借款 | -1434 |

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

| 3 | 到达 | 32 | 存款/还款 | 2228 |

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

| 4 | 到达 | 36 | 存款/还款 | 918 |

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

| 5 | 到达 | 42 | 存款/还款 | 401 |

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

| 6 | 到达 | 48 | 存款/还款 | 2684 |

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

| 3 | 离开 | 51 | 存款/还款 | 2228 |

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

| 7 | 到达 | 54 | 取款/借款 | -2323 |

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

| 4 | 离开 | 57 | 存款/还款 | 918 |

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

| 5 | 离开 | 60 | 存款/还款 | 401 |

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

| 6 | 离开 | 60 | 存款/还款 | 2684 |

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

| 7 | 离开 | 60 | 取款/借款 | -2323 |

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

**[程序所能达到的功能]**

1. 模拟客户到达银行和离开银行事件。
2. 根据给定的数据计算客户在银行内逗留的平均时间
3. 使用动态存储结构来实现队列和事件表的存储。
4. 给出每个客户的到达/离开时间、交易金额、交易时间

**[测试数据]**

测试1：

银行初始存款（total）：10000

银行营业时间(Runtime)：600

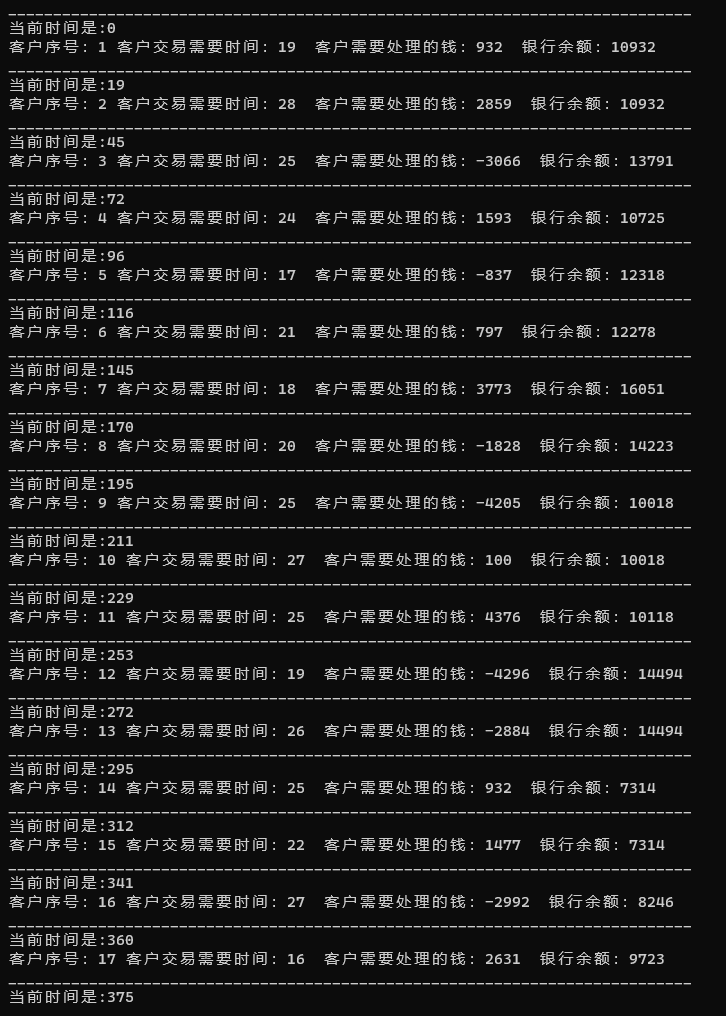
客户最大到达时间间隔(MaxTimeGap)：30

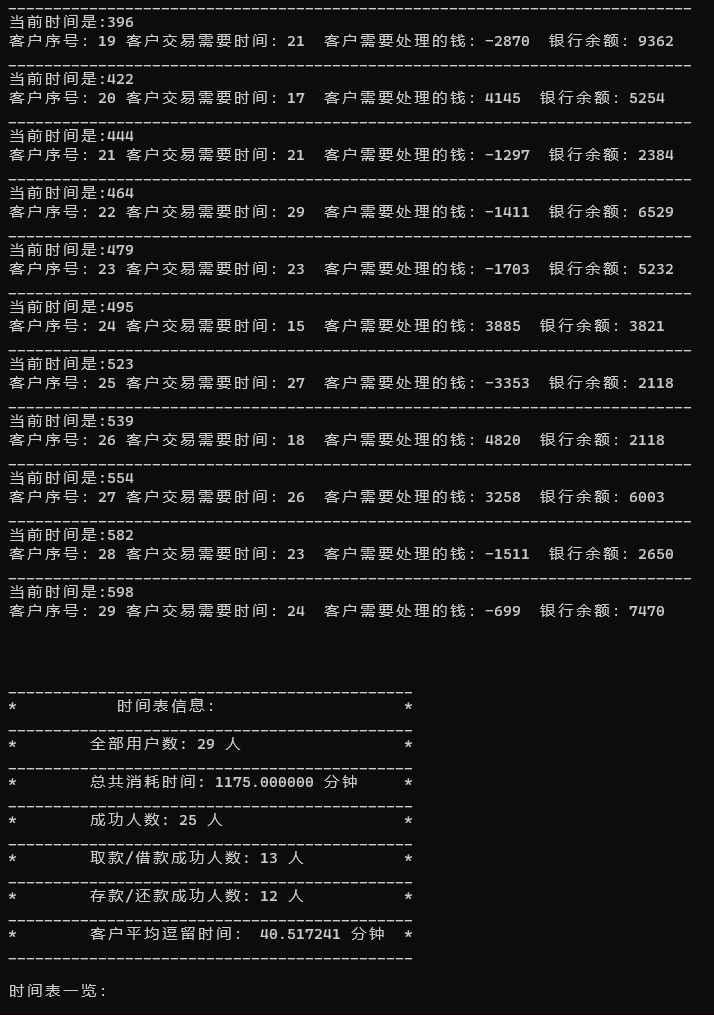
客户最小到达时间间隔(MinTimeGap)：15

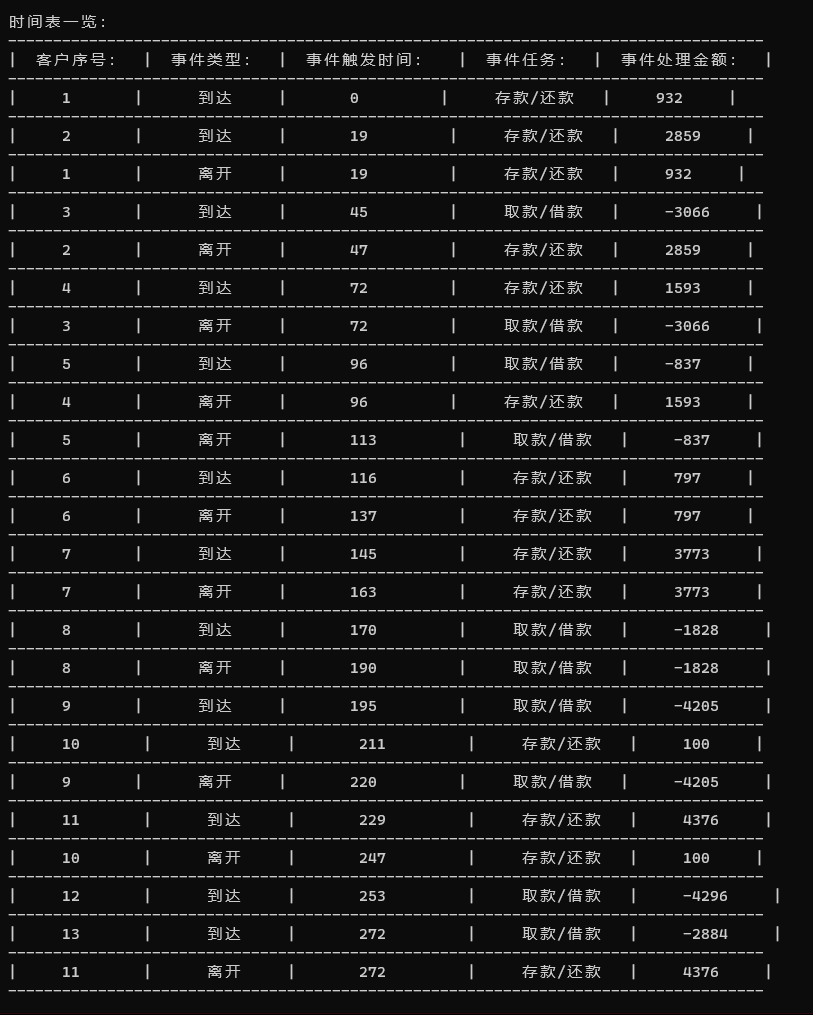
业务最大处理时间(MaxTradeTime)：30

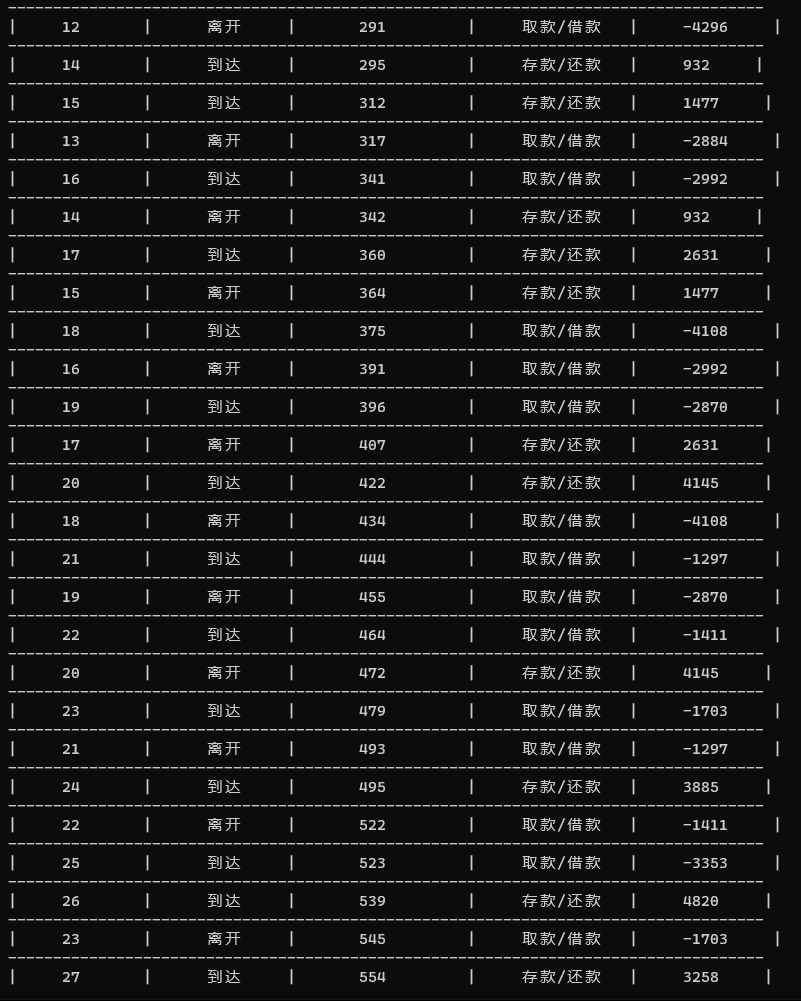
业务最小处理时间(MinTradeTime)：15

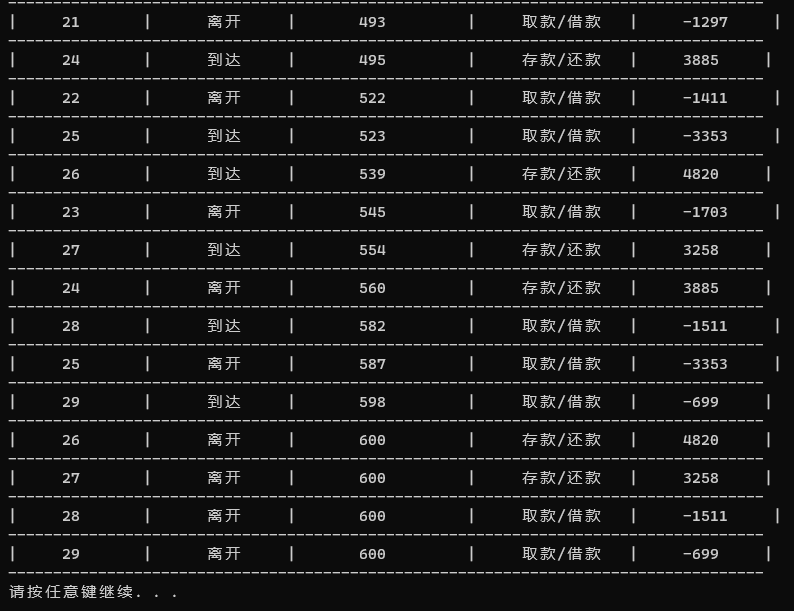
最大交易额上限(max\_money)：6000











**2、概要设计**

**[数据类型的定义]**

typedef int Status;//定义Status为整形

typedef struct Custom//定义客户结点结构体

{

int dealtime;//处理客户时间变量

int amonut;//处理客户交易金额变量（正为存款/还款，负为取款/借款）

int num;//定义客户序号

}ElemType;

typedef struct event

{

int type;//事件类型，1为到达，0为离开

int time;//事件发生时间

int num;//事件触发的客户序号

int index;//事件操作类型，1为存款/还款，0为取款/借款

int money;//客户在该事件中的交易金额

}event;

typedef struct//定义事件表结点

{

event data;

struct LNode\* next;

}LNode, \* Link;

typedef struct//定义事件表为链表的数据结构

{

Link head;

Link tail;

int length;

}\*LinkList;

typedef struct LQNode

{

ElemType data;

struct LQNode\* next;

}LQNode, \* QueuePtr;

typedef struct

{

QueuePtr front;

QueuePtr rear;

int length;

}LQueue;

**Queue基本操作**：

LQueue\* InitQueue()；

初始条件：无

操作结果：初始化队列

Status EnQueue(LQueue \*Q, ElemType 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

Status DeQueue(LQueue \*Q, ElemType \*e)；

初始条件：队列Q存在，结点指针e存在

操作结果：结点e从队列Q中出队，并返回e

Status DestroyQueue(LQueue \*Q)；

初始条件：队列Q存在

操作结果：销毁队列Q并释放空间

Status CombineQueue(LQueue \*Q,LQueue \*E)；

初始条件：队列Q和队列E同时存在

操作结果：将队列E插到队列Q的后面

**LinkList基本操作**：

LinkList InitLinkList()；

初始条件：无

操作结果：初始化事件表结点，返回表L

Status ClearLinkList(LinkList L)；

初始条件：事件表L存在

操作结果：清空事件表L

Status DestroyLinkList(LinkList L)；

初始条件：事件表L存在

操作结果：销毁事件表L并释放空间

Status InsertLinkList(LinkList L, event e)；

初始条件：事件表L存在

操作结果：在链表尾部插入e

Status PrintLink(LinkList L)；

初始条件：事件表L存在

操作结果：遍历事件表L

**ElemType基本操作**：

ElemType\* myMalloc(MaxTradeTime,MinTradeTime)；

初始条件：存在MaxTradeTime和MinTradeTime

操作结果：生成客户结点

Status myFree(ElemType \*e)；

初始条件：存在e

操作结果：释放结点e

**需要使用的全局变量：**

int total;//银行余额

int max\_money;//最大交易金额

LQueue\* deal\_queue = NULL;//客户处理队列

LQueue\* wait\_queue = NULL;//客户等待队列

LinkList event\_link = NULL;//一个事件表（包含到达和离开）

int nextTime = 0;//下一个事件的发生时间

int NextSolveTime = 0;//下一个事件的处理时间

int CustomNumber = 1;//客户的服务序号

ElemType\* leave\_Custom = NULL;//准备离开的客户变量

ElemType\* wait\_Custom = NULL;//待处理的客户变量

**[主程序的流程]**

主程序在运行时会显示选择菜单界面，用户通过输入选项来进行操作。如果用户输入的选项不正确，程序会重新显示界面并要求重新选择。

在开始模拟之前，程序会申请两个空间用于存储客户信息的结点：一个是准备离开客户的结点（leave\_Custom），另一个是等待处理客户的结点（wait\_Custom）。这两个结点是使用自定义的客户结点结构体（ElemType）来表示的。然后，对等待队列（wait\_queue）、处理队列（deal\_queue）和事件表（event\_link）进行初始化操作。

程序会使用随机函数生成随机值，并将其与电脑时间匹配，以确保每次的随机值不同。

接下来，用户需要输入一系列参数，包括银行的运营时间（Runtime）、初始运营金额（total）、客户最大交易金额（max\_money）、客户最大/最小到达时间间隔（MaxTimeGap/MinTimeGap）、客户最大/最小交易时间（MaxTradeTime/MinTradeTime）。

其中，要求银行的运营时间必须大于0，初始运营金额必须大于0，客户最大交易金额必须小于银行初始运营金额，客户最大到达时间间隔必须大于0且小于运营时间，客户最小到达时间间隔必须大于0且小于最大到达时间间隔且小于运营时间，客户最大交易时间必须大于0且小于运营时间，客户最小交易时间必须大于0且小于最大交易时间且小于运营时间。

然后，设定初始时间（i）为0，并且进行一个循环，当i小于运营时间（Runtime）时继续循环。在每次循环中，运行arrival\_event函数来生成到达事件，以及deal\_event函数来生成处理事件。

循环结束后，程序会运行average\_StayTime函数，计算总到达人数、总等待时间、成功处理的人数、成功存款/还款的人数、成功取款/借款的人数，并计算客户的平均逗留时间。

在运营时间到达后，程序会检查处理队列中已到达但尚未处理的客户，并将这些客户移到准备离开的结点中，同时调用leave\_event函数将其写入离开事件中。最后，程序会让所有未处理的客户离开。

最后，调用PrintLink函数对事件表（event\_link）进行遍历，并打印客户序号、事件类型、触发事件的时间、事件任务以及事件处理金额。然后，调用DestroyQueue和DestroyLinList函数销毁处理队列（deal\_queue）、等待队列（wait\_queue）和事件表（event\_link）。最后，调用initial函数对全局变量进行初始化，以方便下次重新进行模拟。

这样，程序就完成了银行业务模拟的整个过程。

**[各程序模块的调用关系]**

该程序主要由主程序模块、到达事件模块、离开事件模块、检查处理模块构成。而主程序模块负责调用到达事件以及检查处理模块。检查处理模块调用离开事件模块。

**3、详细设计**

**Queue基本操作**：

LQueue\* InitQueue()

{

LQueue\* Q = (LQueue\*)malloc(sizeof(LQueue));

if (Q == NULL)return ERROR;

Q->front = NULL;

Q->rear = NULL;

Q->length = 0;

return Q;

}

Status EnQueue(LQueue\* Q, ElemType temp )

{

LQNode\* p;

p = (LQNode\*)malloc(sizeof(LQNode));

if (p == NULL)return ERROR;

p->data = temp;

p->next = NULL;

if (NULL == Q->front)

{

Q->front = p;

Q->rear = p;

Q->length = 1;

}

else

{

Q->rear->next = p;

Q->rear = p;

Q->length++;

}

return OK;

}

Status DeQueue(LQueue\* Q, ElemType\* temp)

{

LQNode\* p;

if (NULL == Q->front)return ERROR;

p = Q->front;

\*temp = p->data;

Q->front = p->next;

if (Q->rear == p)Q->rear = NULL;

free(p);

Q->length--;

return OK;

}

Status DestroyQueue(LQueue\* Q)

{

if (Q == NULL)return ERROR;

QueuePtr p = NULL, q = NULL;

p = Q->front;

while (p != Q->rear)

{

q = p->next;

myFree(p);

p = q;

}

free(Q->rear);

return OK;

}

Status CombineQueue(LQueue\* Q, LQueue\* E)

{

if (E == NULL || E->front == NULL)return ERROR;

if (Q == NULL || Q->front == NULL || Q->rear == NULL)

{

Q->front = E->front;

Q->rear = E->rear;

}

else

{

E->rear->next = Q->front;

Q->front = E->front;

}

return OK;

}

**LinkList基本操作**：

LinkList InitLinkList()

{

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

if (L == NULL)return NULL;

L->head = NULL;

L->tail = NULL;

L->length = 0;

return L;

}

Status ClearLinkList(LinkList L)

{

if (L == NULL)return ERROR;

Link p = L->head;

Link k = L->head->next;

while (p != L->tail)

{

free(p);

p = k;

if (k != L->tail)k = k->next;

}

free(p);

L->head = NULL;

L->tail = NULL;

L->length = 0;

return OK;

}

Status DestroyLinkList(LinkList L)

{

if (L == NULL)return ERROR;

ClearLinkList(L);

free(L);

return OK;

}

Status InsertLinkList(LinkList L, event e)

{

if (L == NULL)return ERROR;

Link p = (Link)malloc(sizeof(LNode));

if (p == NULL)return ERROR;

p->data = e;

p->next = NULL;

if (L->head == NULL)

{

L->head = p;

L->tail = p;

L->length = 1;

}

else

{

L->tail->next = p;

L->tail = p;

L->length++;

}

return OK;

}

Status PrintLinkList(LinkList L)

{

if (L == NULL) return ERROR;

Link p = L->head;

printf("\n时间表一览：\n");

printf("------------------------------------------------------------------------------------");

printf("\n| 客户序号： | 事件类型： | 事件触发时间： | 事件任务： | 事件处理金额： |");

printf("\n------------------------------------------------------------------------------------");

while (p != NULL)

{

printf("\n| %d |", p->data.num);

if (p->data.type == 1)

{

printf(" 到达 | %d |", p->data.time);

if (p->data.index == 1)printf(" 存款/还款 | %d |", p->data.money);

else printf(" 取款/借款 | %d |", p->data.money);

printf("\n------------------------------------------------------------------------------------");

}

else

{

if (p->data.type == 0)

{

printf(" 离开 | %d |", p->data.time);

if (p->data.index == 1)printf(" 存款/还款 | %d |", p->data.money);

else printf(" 取款/借款 | %d |", p->data.money);

printf("\n------------------------------------------------------------------------------------");

}

}

p = p->next;

}

printf("\n");

return OK;

}

**ElemType基本操作**：

ElemType\* myMalloc(int MaxDealTime, int MinDealTime)

{

ElemType\* e;

e = (ElemType\*)malloc(sizeof(ElemType));

if (e == NULL)return NULL;

e->dealtime = (rand() % (MaxDealTime - MinDealTime) + MinDealTime);//处理用户事件事件在最小交易时间和最大交易时间之间的随机数

e->amonut = (rand() % (2 \* max\_money) - max\_money);//用户交易金额的随机数

e->num = CustomNumber;//用户序号

CustomNumber++;

return e;

}

Status myFree(ElemType\* e)

{

if (e == NULL)return ERROR;

free(e);

return OK;

}

**event基本操作**：

ElemType\* arrival\_event(int i, int MaxTimeGap, int MinTimeGap, int MaxTradeTime, int MinTradeTime)

{

if (nextTime == i)//如果当前时间等于下次客户到达时间

{

printf("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n");

printf("当前时间是:%d\n", i);

nextTime = nextTime + (rand() % (MaxTimeGap - MinTimeGap) + MinTimeGap);//下个用户到达时间为当前时间加上时间间隔范围的随机数

ElemType\* p = NULL;

p = myMalloc(MaxTradeTime, MinTradeTime);

event\* e = (event\*)malloc(sizeof(event));

e->type = 1;

e->num = p->num;

e->time = i;

e->money = p->amonut;

if (p->amonut < 0)e->index = 0;//如果随机交易金额为负，则定义为取款/借款

else e->index = 1;//反之则定义为存款/还款

InsertLinkList(event\_link, \*e);//将到达事件插入到事件表的尾部

if (NextSolveTime < i)

{

if (-(p->amonut) <= total)//当事件为存款/（取款/借款）金额小于银行余额

{

NextSolveTime = p->dealtime+ i;//下一事件的处理时间为该用户交易时间的随机数+当前时间

total = total + p->amonut;

EnQueue(deal\_queue, \*p);//将p入队

}

else EnQueue(wait\_queue, \*p);

}

else

{

if (NextSolveTime == 0)

{

if (-(p->amonut) <= total)

{

NextSolveTime = p->dealtime + i;

total = total + p->amonut;

EnQueue(deal\_queue, \*p);

}

else EnQueue(wait\_queue, \*p);

}

else EnQueue(deal\_queue, \*p);

}

printf("客户序号：%d 客户交易需要时间：%d 客户需要处理的钱：%d 银行余额：%d\n", p->num, p->dealtime, p->amonut, total);

return p;

}

}

Status leave\_event(ElemType\* e, int i)

{

event\* p = (event\*)malloc(sizeof(event));

p->type = 0;

p->num = e->num;

p->time = i;

p->money = e->amonut;

if (e->amonut < 0)p->index = 0;

else p->index = 1;

InsertLinkList(event\_link, \*p);

return OK;

}

Status deal\_event(int i)

{

if (NextSolveTime == i)//当前时间等于交易完成的时间

{

DeQueue(deal\_queue, leave\_Custom);//将该客户移出处理队列

int last\_money = total - leave\_Custom->amonut;//last\_money为用户移出后的银行余额

leave\_event(leave\_Custom, i);//将移出的客户和时间信息放进事件表

if (((leave\_Custom->amonut > 0) && wait\_queue->front != NULL))//准备离开用户是存钱的

{

int index = 0;

LQueue\* e = InitQueue();

int TempMoney = total;

while (TempMoney > last\_money && wait\_queue->front != NULL)

{

DeQueue(wait\_queue, wait\_Custom);//尝试将准备解决的用户移出等待队列

if (-(wait\_Custom->amonut) <= TempMoney)

{

EnQueue(e, \*wait\_Custom);//将准备解决的用户入队列e

TempMoney = TempMoney + wait\_Custom->amonut;//计算暂时的银行余额

}

else EnQueue(wait\_queue, \*wait\_Custom);//如果失败就将该用户重新移入等待队列的尾部

index++;

if (index >= wait\_queue->length)break;

}

CombineQueue(deal\_queue, e);//将队列e顺序接在处理队列的尾部

if (deal\_queue->front != NULL)

{

while (deal\_queue->front != NULL && -(deal\_queue->front->data.amonut) > total)//无法处理

{

DeQueue(deal\_queue, leave\_Custom);//从处理队列移出

EnQueue(wait\_queue, \*leave\_Custom);//加入等待队列

}

if (deal\_queue->front != NULL)

{

NextSolveTime = deal\_queue->front->data.dealtime+ NextSolveTime;//更新下个用户处理时间

total = total + deal\_queue->front->data.amonut;//更新银行余额

}

}

}

else//为取款/借款类型，其余同上

{

if (deal\_queue->front != NULL)

{

while (deal\_queue->front != NULL && -(deal\_queue->front->data.amonut) > total)

{

DeQueue(deal\_queue, leave\_Custom);

EnQueue(wait\_queue, \*leave\_Custom);

}

if (deal\_queue->front != NULL)

{

NextSolveTime = deal\_queue->front->data.dealtime + NextSolveTime;

total = total + deal\_queue->front->data.amonut;

}

}

}

}

}

**其他：**

int main()

{

int choice=0;

while (1)

{

system("CLS");

menu();

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

if (choice >= 1 && choice <= 2)

{

switch (choice)

{

case 1:

{

leave\_Custom = (ElemType\*)malloc(sizeof(ElemType));

wait\_Custom = (ElemType\*)malloc(sizeof(ElemType));

deal\_queue = InitQueue();

wait\_queue = InitQueue();

event\_link = InitLinkList();

srand((unsigned int)time(NULL));//随机函数随电脑时间变化,确保不会随机数一样

int Runtime, MaxTimeGap, MinTimeGap, MaxTradeTime, MinTradeTime, i;

printf("请输入银行的运营时间（分钟）：");

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

while (Runtime <= 0)

{

printf("运营时间必须大于0\n");

printf("请输入银行的运营时间（分钟）：");

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

}

printf("请输入银行的运营金额（元）：");

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

while (total <= 0)

{

printf("初始运营金额必须大于0\n");

printf("请输入银行的运营金额（元）：");

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

}

printf("请输入客户最大交易金额（元）：");

scanf\_s("%d", &max\_money);

while (max\_money > total || max\_money <= 0)

{

printf("客户最大交易金额必须小于银行初始运营金额且大于0\n");

printf("请输入客户最大交易金额（元）：");

scanf\_s("%d", &max\_money);

}

printf("请输入客户最大到达时间间隔（分钟）：");

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

while (MaxTimeGap <= 0 || MaxTimeGap > Runtime)

{

printf("客户最大到达时间间隔必须大于0且小于运营时间\n");

printf("请输入客户最大到达时间间隔（分钟）：");

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

}

printf("请输入客户最小时间间隔（分钟）：");

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

while (MinTimeGap <= 0 || MinTimeGap > MaxTimeGap || MinTimeGap > Runtime)

{

printf("客户最小到达时间间隔必须大于0且小于最大到达时间间隔且小于运营时间\n");

printf("请输入客户最小时间间隔（分钟）：");

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

}

printf("请输入客户最大交易时间（分钟）：");

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

while (MaxTradeTime <= 0 || MaxTradeTime > Runtime)

{

printf("客户最大交易时间必须大于0且小于运营时间\n");

printf("请输入客户最大交易时间（分钟）：");

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

}

printf("请输入客户最小交易时间（分钟）：");

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

while (MinTradeTime <= 0 || MinTradeTime > MaxTradeTime || MinTradeTime > Runtime)

{

printf("客户最小交易时间必须大于0且小于最大交易时间且小于运营时间\n");

printf("请输入客户最小交易时间（分钟）：");

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

}

for (i = 0; i < Runtime; i++)

{

arrival\_event(i, MaxTimeGap, MinTimeGap, MaxTradeTime, MinTradeTime);

deal\_event(i);

}

printf("\n");

average\_StayTime(event\_link, Runtime - 1);

while (deal\_queue->front != NULL)

{

DeQueue(deal\_queue, leave\_Custom);

leave\_event(leave\_Custom, i);

}

while (wait\_queue->front != NULL)

{

DeQueue(wait\_queue, leave\_Custom);

leave\_event(leave\_Custom, i);

}

PrintLinkList(event\_link);

DestroyQueue(deal\_queue);

DestroyQueue(wait\_queue);

DestroyLinkList(event\_link);

system("pause");

break;

}

case 2:

{

printf("感谢您的使用！\n");

system("PAUSE");

goto end;

}

}

}

initial();

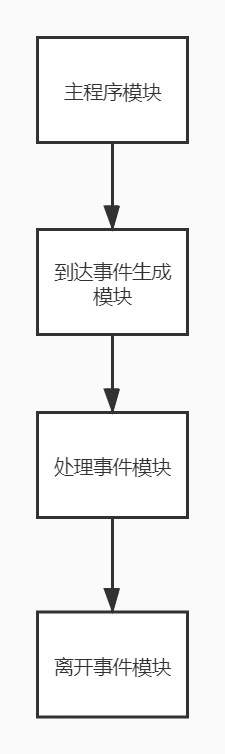
}

end:

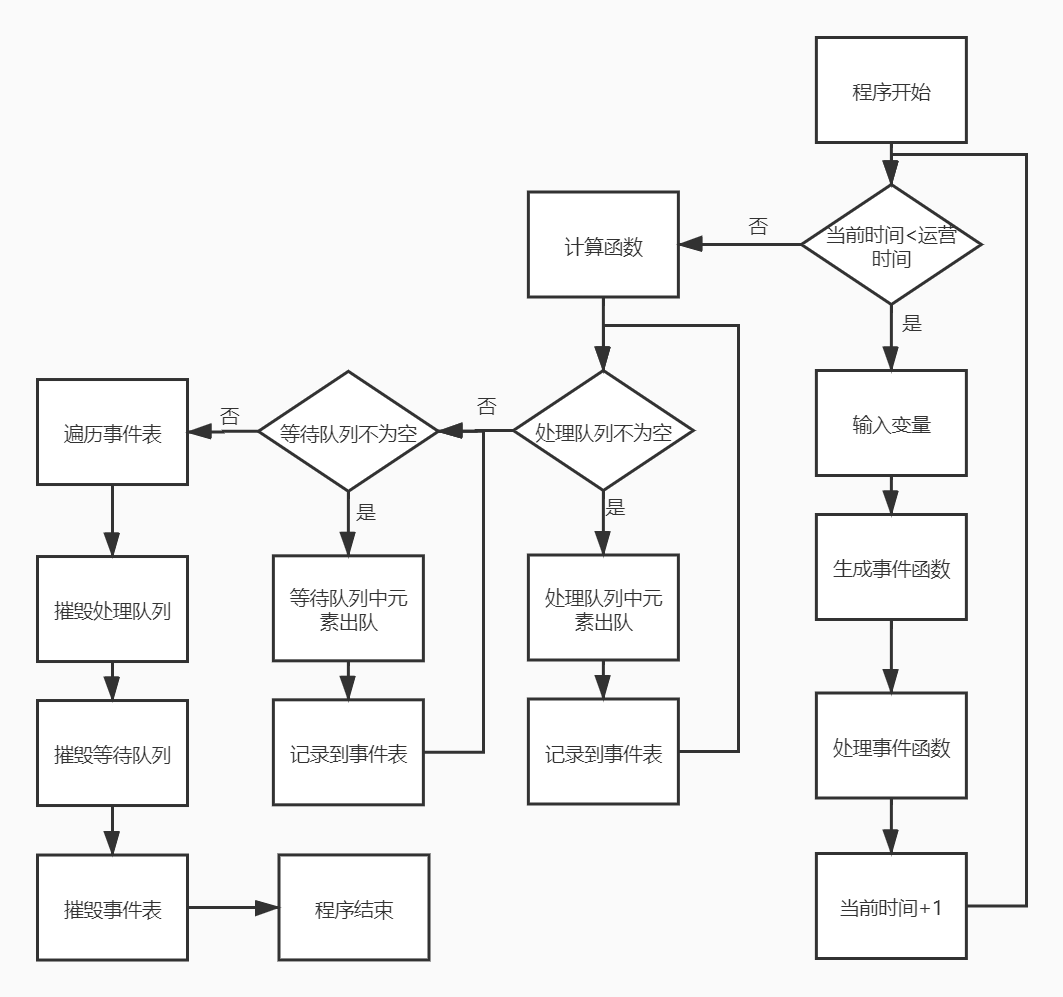
return 0;

}

[函数调用关系图]

****

**[过程调用关系图]**

****

**4、调试分析**

[调试过程中遇到的问题以及对设计与实现的回顾讨论和分析]

问题1：使用动态分配函数malloc和free时，由于对内存的管理不善，出现了内存泄 漏和悬空指针的问题

解决方法：重新检查代码，确保在适当的时候释放已经分配的内存，使用合适的算法 管理队列和事件表。

问题2：银行达到最终运营时间时仍有客户到达但未离开

解决方法：通过不断的debug发现客户仍旧在deal\_custom队列中未出队，所以在循 环结束后将队列中的结点全部出队。

问题3：编码过程中将myFree函数写在了DestoryQueue函数后，导致程序运行失败

解决方法：将myFree函数写在DestoryQueue函数前。

[算法的时空分析]

1. LinkList InitLinkList()

此操作为初始化链表，时间复杂度为O(1),空间复杂度为O(1)

1. Status ClearLinkList(LinkList L)

此操作为清空链表，时间复杂度为O(n)，空间复杂度为O(1)

1. Status DestroyLinkList(LinkList L)

此操作为摧毁链表，时间复杂度为O(n)，空间复杂度为O(1)

1. Status InsertLinkList(LinkList L, event e)

此操作为链表的尾插，时间复杂度为O(n)，空间复杂度为O(1)

1. Status PrintLinkList(LinkList L)

此操作为链表的遍历，时间复杂度为O(n)，空间复杂度为O(1)

1. LQueue\* InitQueue()

此操作为初始化队列，时间复杂度为O(1),空间复杂度为O(1)

1. Status EnQueue(LQueue \*Q, ElemType e)

此操作为入队，时间复杂度为O(n),空间复杂度为O(1)

1. Status DeQueue(LQueue \*Q, ElemType \*e)

此操作为出队，时间复杂度为O(1),空间复杂度为O(1)

1. Status myFree(ElemType \*e)

此操作为释放客户结点，时间复杂度为O(1),空间复杂度为O(1)

1. Status DestroyQueue(LQueue \*Q)

此操作为队列的删除，时间复杂度为O(n),空间复杂度为O(1)

1. Status CombineQueue(LQueue \*Q,LQueue \*E)

此操作为将一个队列插入另一个队列，时间复杂度为O(n),空间复杂度为O(1)

1. ElemType\* myMalloc(MaxTradeTime,MinTradeTime)

此操作为生成客户结点，时间复杂度为O(1),空间复杂度为O(1)

1. ElemType\*arrival\_event(i,MaxTimeGap,MinTimeGap,MaxTradeTime,MinTradeTime)

该操作为生成到达事件，时间复杂度为O(1),空间复杂度为O(1)

1. Status leave\_event(ElemType \*e ,i)

该操作为生成离开事件，时间复杂度为O(1),空间复杂度为O(1)

1. Status deal\_event(i)

该操作为处理客户是否移到等待队列或等到完成时间后离开，时间复杂度为O(1),空间复杂度为O(1)

1. void average\_StayTime(LinkList e,double final)

此操作为计算平均等待时间等，等同遍历链表，时间复杂度为O(n)，空间复杂度为O(1)

1. void menu()

此操作为打印目录，时间复杂度为O(1),空间复杂度为O(1)

1. void initial()

此操作为初始化全局变量，时间复杂度为O(1),空间复杂度为O(1)

[改进设想]

1. 用户界面优化：改进用户界面，使其更加美观。增加图形化元素、交互式图标，以更好地展示模拟结果和系统运行状态。
2. 多样化业务场景：扩展模拟系统的业务场景，包括更多类型的银行业务，如贷款、投资、保险等。

[经验和体会]

1. 良好的设计：编码前要充分考虑业务逻辑和需求的复杂性。良好的设计能够降低后期维护的成本
2. 模块化和可复用性：将系统模块化，每个模块负责一个特定的功能，使得代码更容易理解和维护。
3. 边界条件的考虑：在银行业务模拟系统中，各种边界条件可能对模拟结果产生重要影响。要仔细考虑边界的设置，确保系统能够正确应对各种极端情况。
4. 性能优化：若模拟的规模过大，性能将成为重要的考虑因素。对代码进行合理的优化，提高模拟系统的运行效率。

**5、使用说明**

运行程序，主界面出现两个选项

1. 程序模拟
2. 退出

输入1进入模拟。根据提示依次输入下列参数

银行营业时间(Runtime)（需>0）

银行初始存款（total）（需>0）

最大交易额上限(max\_money)(需>0且<total)

客户最大到达时间间隔(MaxTimeGap)（需>0且>MinTimeGap）

客户最小到达时间间隔(MinTimeGap)（需>0）

交易最大处理时间(MaxTradeTime)（需>0且>MinTradeGap）

交易最小处理时间(MinTradeTime)（需>0）

**6、测试结果**

测试2：

银行初始存款（total）：10000

银行营业时间(Runtime)：600

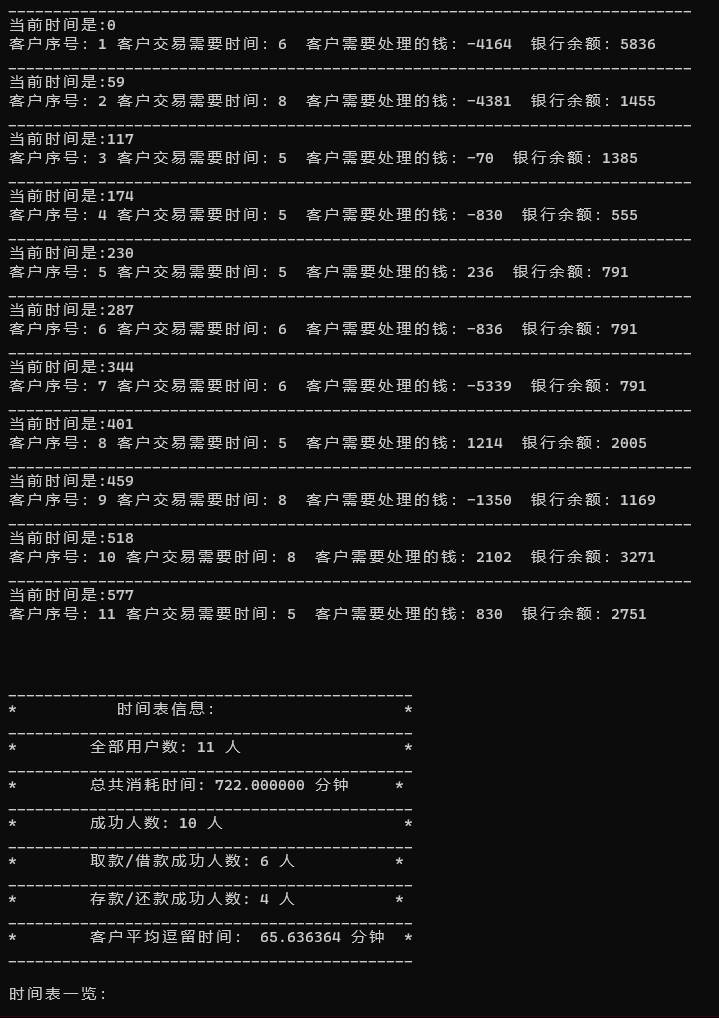
客户最大到达时间间隔(MaxTimeGap)：60

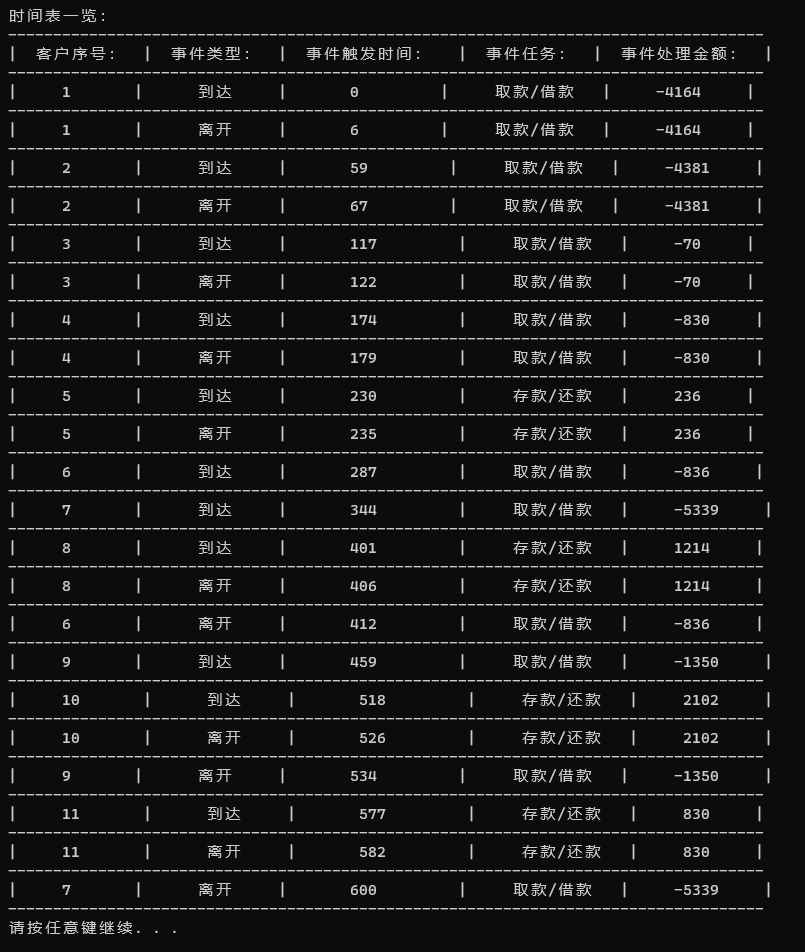
客户最小到达时间间隔(MinTimeGap)：55

业务最大处理时间(MaxTradeTime)：10

业务最小处理时间(MinTradeTime)：5

最大交易额上限(max\_money)：6000





测试3：

银行初始存款（total）：10000

银行营业时间(Runtime)：600

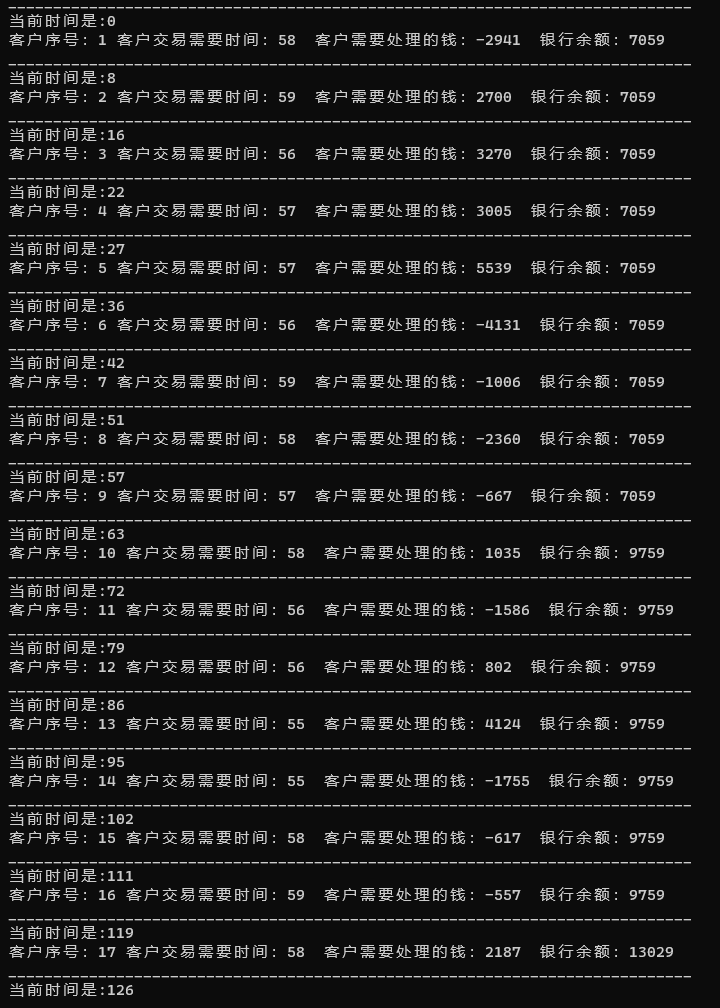
客户最大到达时间间隔(MaxTimeGap)：10

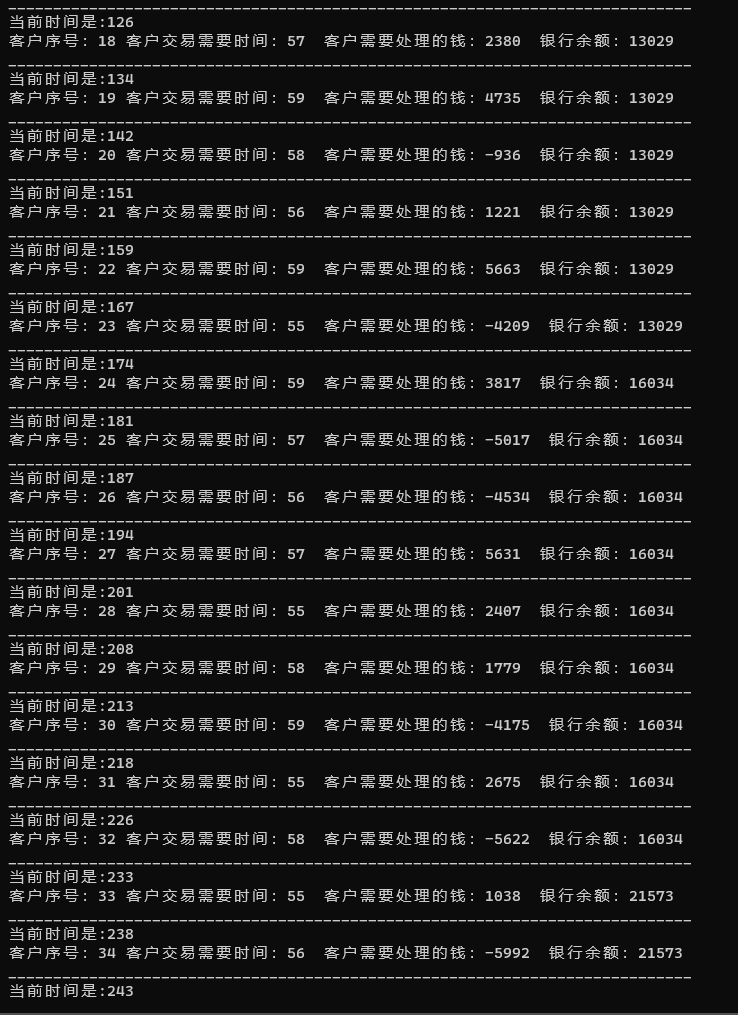
客户最小到达时间间隔(MinTimeGap)：5

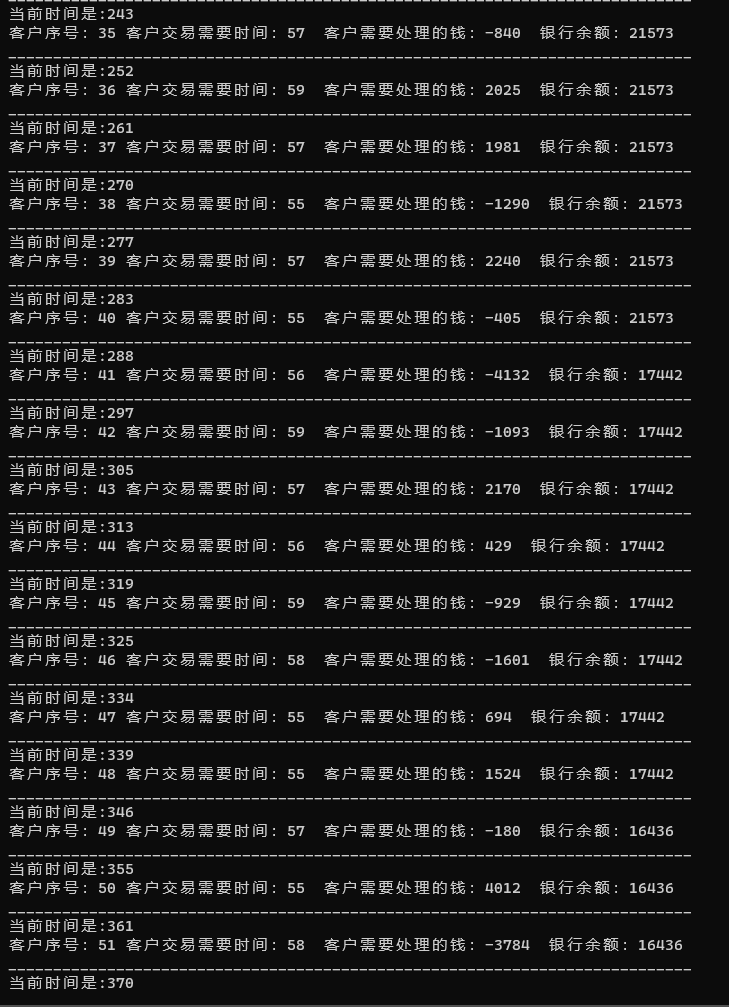
业务最大处理时间(MaxTradeTime)：60

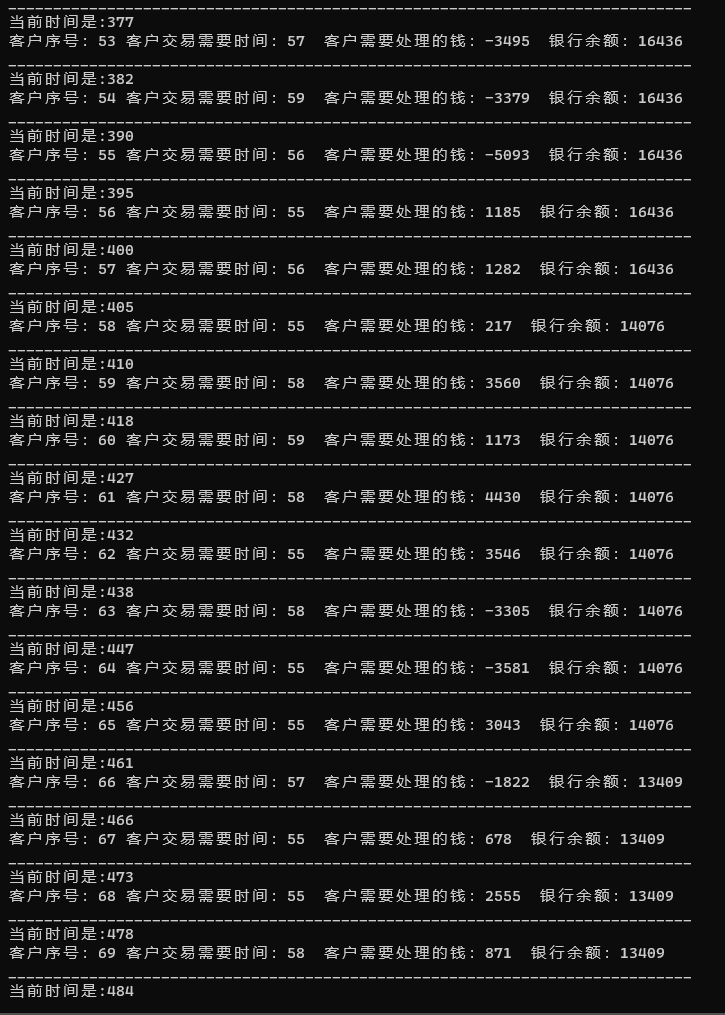
业务最小处理时间(MinTradeTime)：55

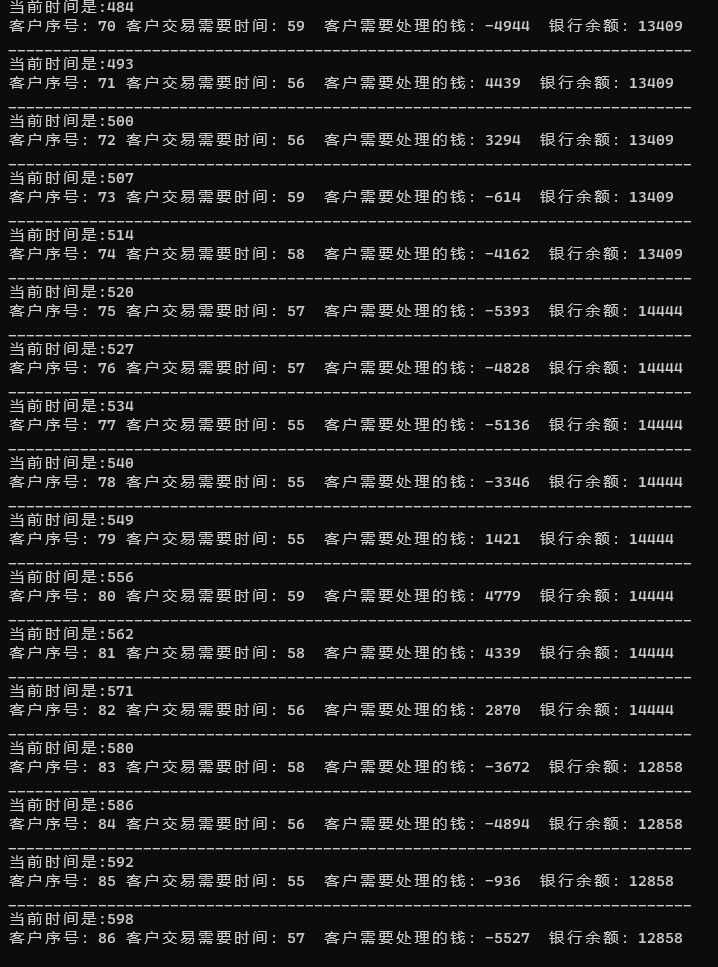
最大交易额上限(max\_money)：6000

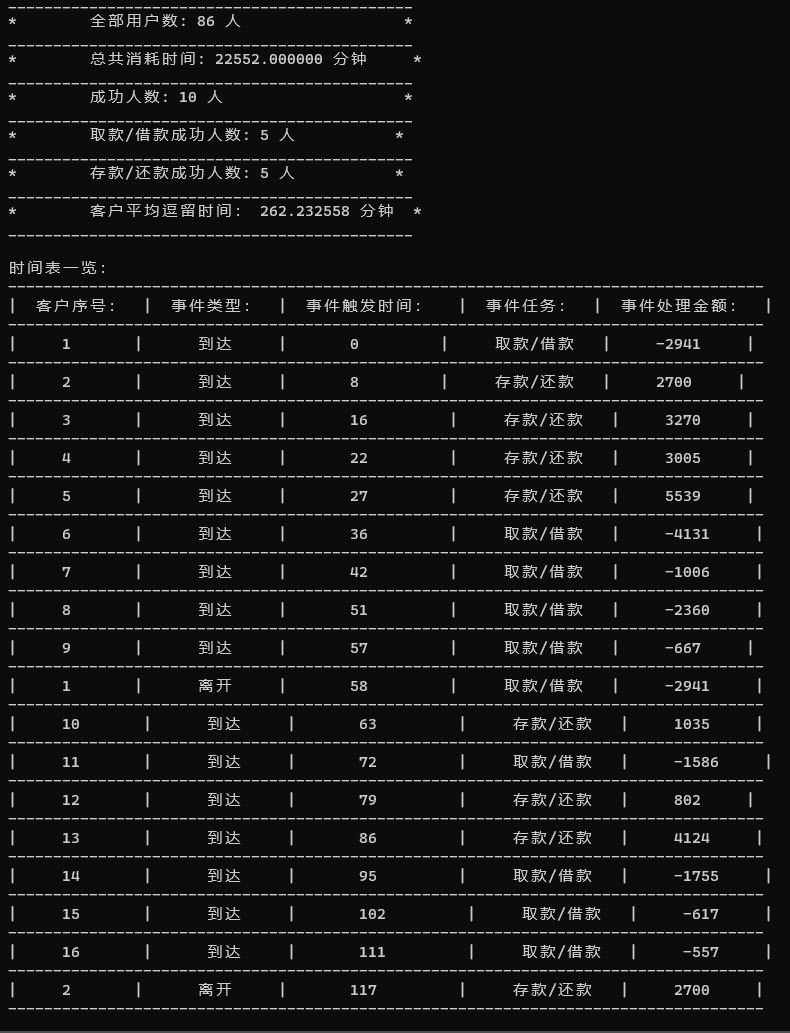


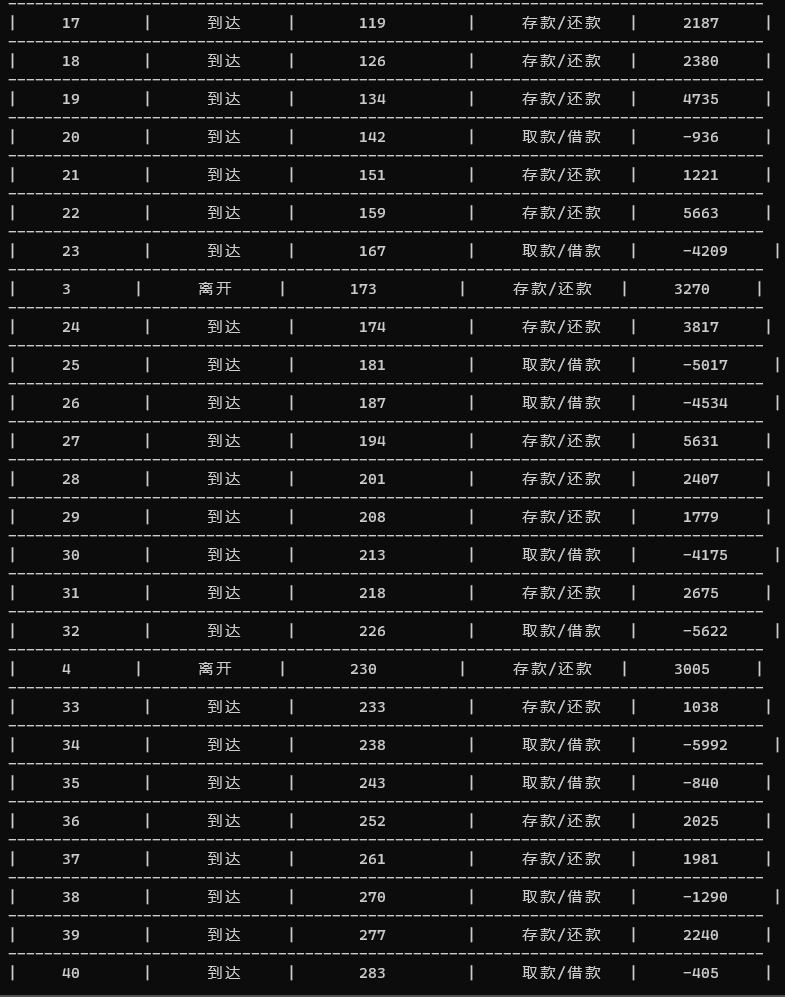


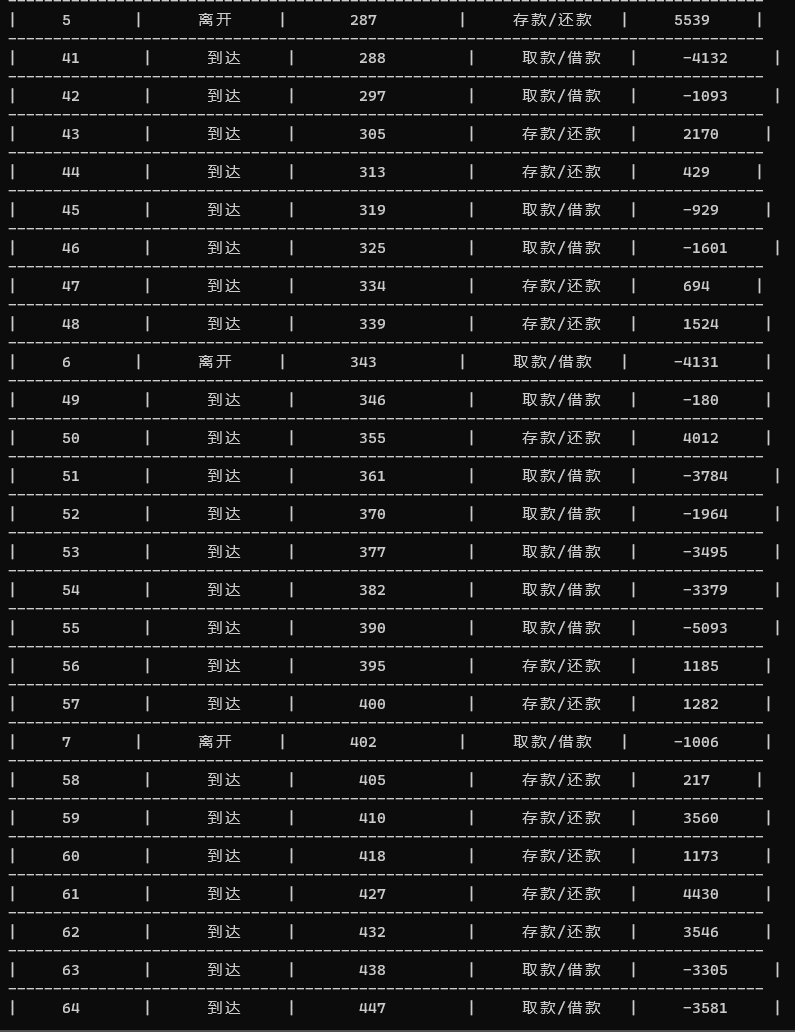


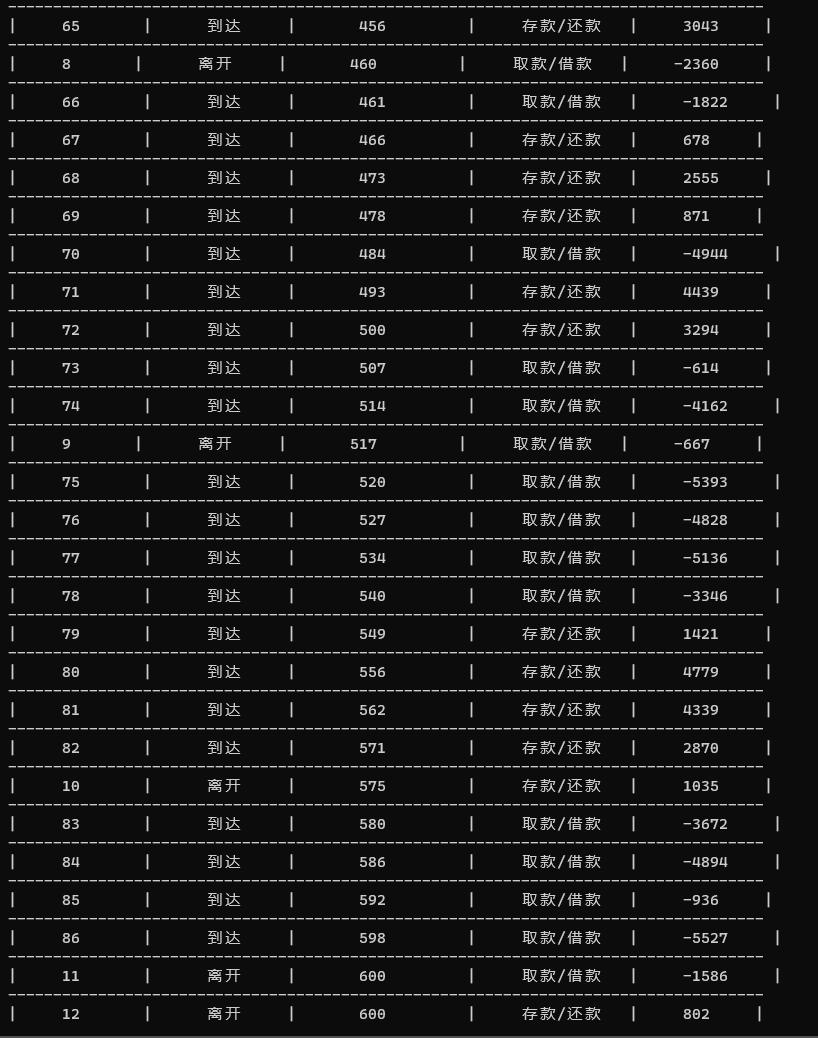


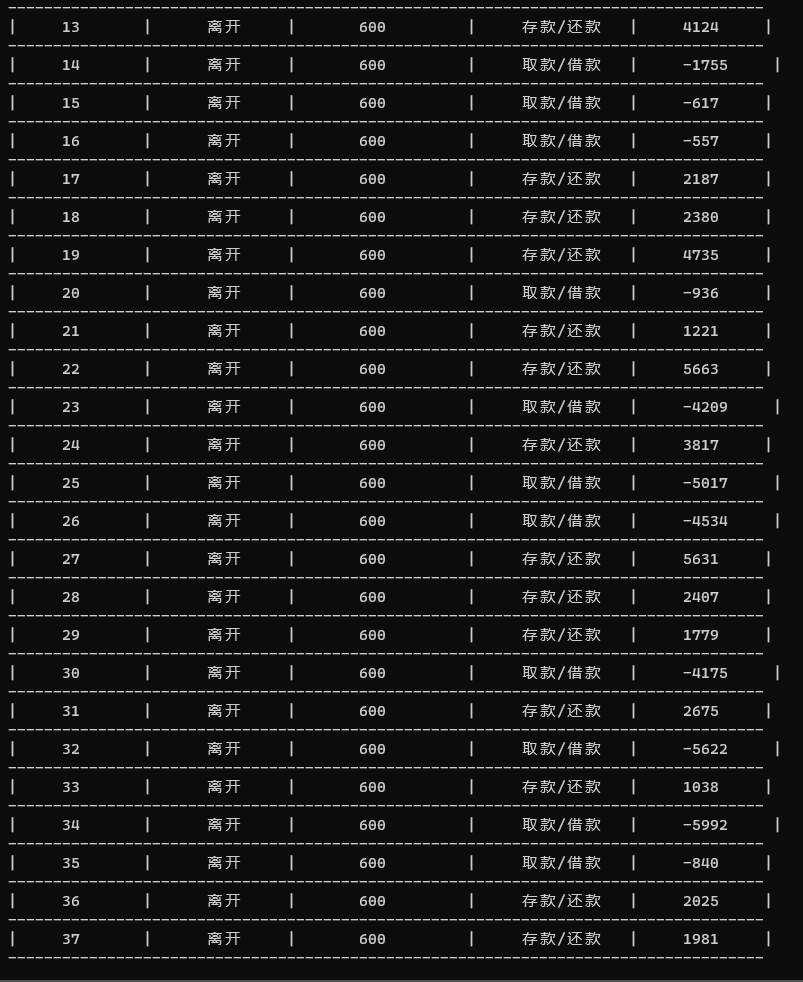


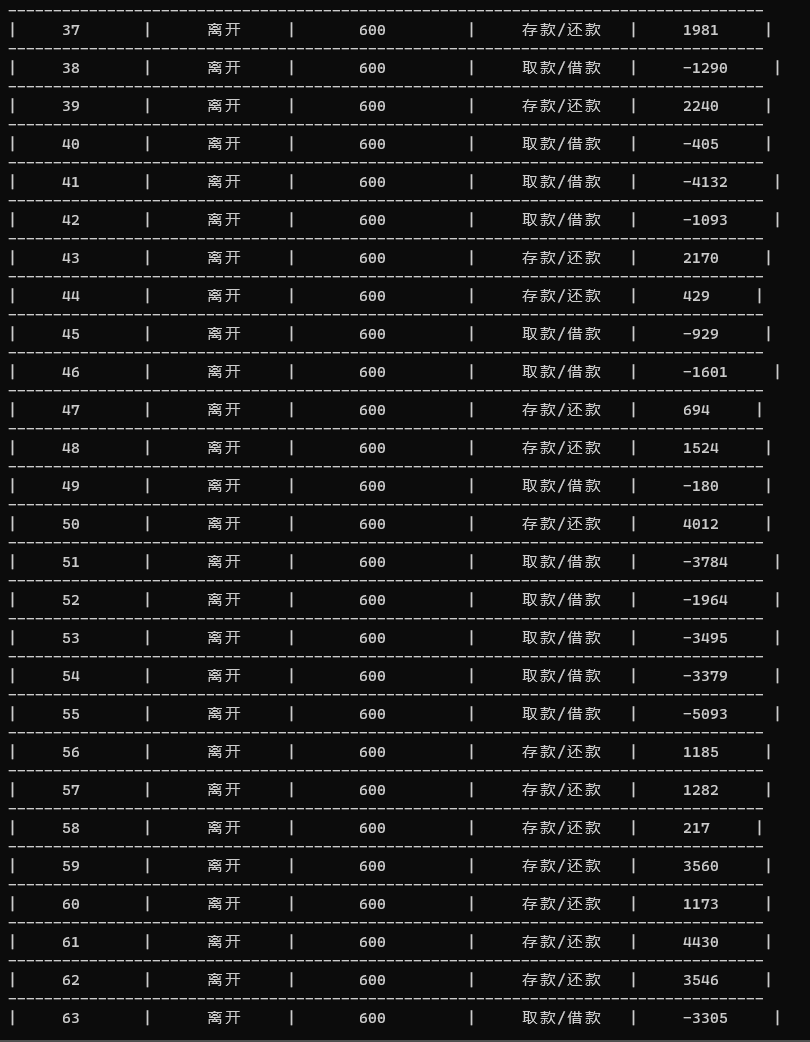


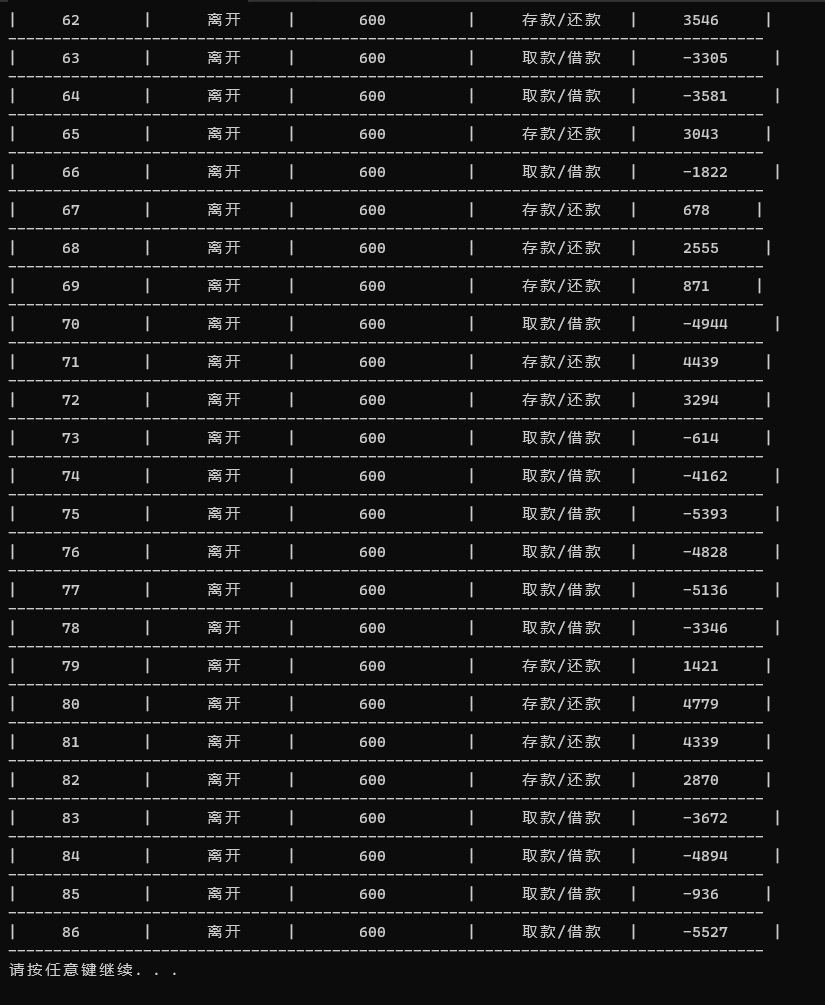






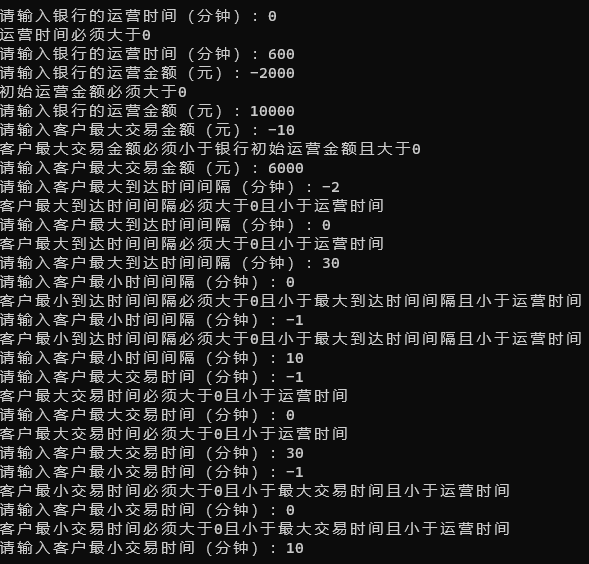


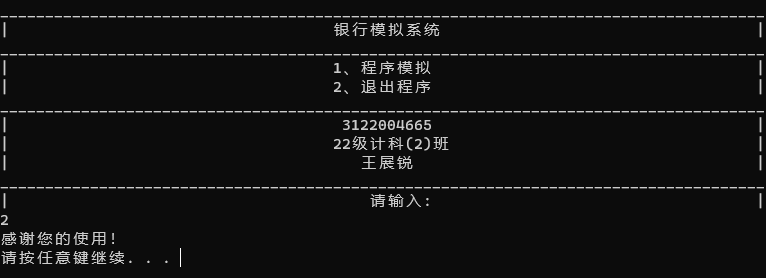




测试4：

非法输入及退出程序





**7、附录**

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#include<Windows.h>

#define OK 1

typedef int Status;//定义Status为整形

typedef struct Custom//定义客户结点结构体

{

int dealtime;//处理客户时间变量

int amonut;//处理客户交易金额变量（正为存款/还款，负为取款/借款）

int num;//定义客户序号

}ElemType;

typedef struct event

{

int type;//事件类型，1为到达，0为离开

int time;//事件发生时间

int num;//事件触发的客户序号

int index;//事件操作类型，1为存款/还款，0为取款/借款

int money;//客户在该事件中的交易金额

}event;

typedef struct//定义事件表结点

{

event data;

struct LNode\* next;

}LNode, \* Link;

typedef struct//定义事件表为链表的数据结构

{

Link head;

Link tail;

int length;

}\*LinkList;

LinkList InitLinkList()

{

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

if (L == NULL)return NULL;

L->head = NULL;

L->tail = NULL;

L->length = 0;

return L;

}

Status ClearLinkList(LinkList L)

{

if (L == NULL)return ERROR;

Link p = L->head;

Link k = L->head->next;

while (p != L->tail)

{

free(p);

p = k;

if (k != L->tail)k = k->next;

}

free(p);

L->head = NULL;

L->tail = NULL;

L->length = 0;

return OK;

}

Status DestroyLinkList(LinkList L)

{

if (L == NULL)return ERROR;

ClearLinkList(L);

free(L);

return OK;

}

Status InsertLinkList(LinkList L, event e)

{

if (L == NULL)return ERROR;

Link p = (Link)malloc(sizeof(LNode));

if (p == NULL)return ERROR;

p->data = e;

p->next = NULL;

if (L->head == NULL)

{

L->head = p;

L->tail = p;

L->length = 1;

}

else

{

L->tail->next = p;

L->tail = p;

L->length++;

}

return OK;

}

Status PrintLinkList(LinkList L)

{

if (L == NULL) return ERROR;

Link p = L->head;

printf("\n时间表一览：\n");

printf("------------------------------------------------------------------------------------");

printf("\n| 客户序号： | 事件类型： | 事件触发时间： | 事件任务： | 事件处理金额： |");

printf("\n------------------------------------------------------------------------------------");

while (p != NULL)

{

printf("\n| %d |", p->data.num);

if (p->data.type == 1)

{

printf(" 到达 | %d |", p->data.time);

if (p->data.index == 1)printf(" 存款/还款 | %d |", p->data.money);

else printf(" 取款/借款 | %d |", p->data.money);

printf("\n------------------------------------------------------------------------------------");

}

else

{

if (p->data.type == 0)

{

printf(" 离开 | %d |", p->data.time);

if (p->data.index == 1)printf(" 存款/还款 | %d |", p->data.money);

else printf(" 取款/借款 | %d |", p->data.money);

printf("\n------------------------------------------------------------------------------------");

}

}

p = p->next;

}

printf("\n");

return OK;

}

typedef struct LQNode

{

ElemType data;

struct LQNode\* next;

}LQNode, \* QueuePtr;

typedef struct

{

QueuePtr front;

QueuePtr rear;

int length;

}LQueue;

Status EnQueue(LQueue\* Q, ElemType temp )

{

LQNode\* p;

p = (LQNode\*)malloc(sizeof(LQNode));

if (p == NULL)return ERROR;

p->data = temp;

p->next = NULL;

if (NULL == Q->front)

{

Q->front = p;

Q->rear = p;

Q->length = 1;

}

else

{

Q->rear->next = p;

Q->rear = p;

Q->length++;

}

return OK;

}

Status DeQueue(LQueue\* Q, ElemType\* temp)

{

LQNode\* p;

if (NULL == Q->front)return ERROR;

p = Q->front;

\*temp = p->data;

Q->front = p->next;

if (Q->rear == p)Q->rear = NULL;

free(p);

Q->length--;

return OK;

}

LQueue\* InitQueue()

{

LQueue\* Q = (LQueue\*)malloc(sizeof(LQueue));

if (Q == NULL)return ERROR;

Q->front = NULL;

Q->rear = NULL;

Q->length = 0;

return Q;

}

Status myFree(ElemType\* e)

{

if (e == NULL)return ERROR;

free(e);

return OK;

}

Status DestroyQueue(LQueue\* Q)

{

if (Q == NULL)return ERROR;

QueuePtr p = NULL, q = NULL;

p = Q->front;

while (p != Q->rear)

{

q = p->next;

myFree(p);

p = q;

}

free(Q->rear);

return OK;

}

Status CombineQueue(LQueue\* Q, LQueue\* E)

{

if (E == NULL || E->front == NULL)return ERROR;

if (Q == NULL || Q->front == NULL || Q->rear == NULL)

{

Q->front = E->front;

Q->rear = E->rear;

}

else

{

E->rear->next = Q->front;

Q->front = E->front;

}

return OK;

}

/\*-------------------------------------定义需要使用的全局变量--------------------------------------\*/

int total;//银行余额

int max\_money;//最大交易金额

LQueue\* deal\_queue = NULL;//客户处理队列

LQueue\* wait\_queue = NULL;//客户等待队列

LinkList event\_link = NULL;//一个事件表（包含到达和离开）

int nextTime = 0;//下一个事件的发生时间

int NextSolveTime = 0;//下一个事件的处理时间

int CustomNumber = 1;//客户的服务序号

ElemType\* leave\_Custom = NULL;//准备离开的客户变量

ElemType\* wait\_Custom = NULL;//待处理的客户变量

/\*-------------------------------------定义需要使用的全局变量--------------------------------------\*/

ElemType\* myMalloc(int MaxDealTime, int MinDealTime)

{

ElemType\* e;

e = (ElemType\*)malloc(sizeof(ElemType));

if (e == NULL)return NULL;

e->dealtime = (rand() % (MaxDealTime - MinDealTime) + MinDealTime);//处理用户事件事件在最小交易时间和最大交易时间之间的随机数

e->amonut = (rand() % (2 \* max\_money) - max\_money);//用户交易金额的随机数

e->num = CustomNumber;//用户序号

CustomNumber++;

return e;

}

ElemType\* arrival\_event(int i, int MaxTimeGap, int MinTimeGap, int MaxTradeTime, int MinTradeTime)

{

if (nextTime == i)//如果当前时间等于下次客户到达时间

{

printf("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n");

printf("当前时间是:%d\n", i);

nextTime = nextTime + (rand() % (MaxTimeGap - MinTimeGap) + MinTimeGap);//下个用户到达时间为当前时间加上时间间隔范围的随机数

ElemType\* p = NULL;

p = myMalloc(MaxTradeTime, MinTradeTime);

event\* e = (event\*)malloc(sizeof(event));

e->type = 1;

e->num = p->num;

e->time = i;

e->money = p->amonut;

if (p->amonut < 0)e->index = 0;//如果随机交易金额为负，则定义为取款/借款

else e->index = 1;//反之则定义为存款/还款

InsertLinkList(event\_link, \*e);//将到达事件插入到事件表的尾部

if (NextSolveTime < i)

{

if (-(p->amonut) <= total)//当事件为存款/（取款/借款）金额小于银行余额

{

NextSolveTime = p->dealtime+ i;//下一事件的处理时间为该用户交易时间的随机数+当前时间

total = total + p->amonut;

EnQueue(deal\_queue, \*p);//将p入队

}

else EnQueue(wait\_queue, \*p);

}

else

{

if (NextSolveTime == 0)

{

if (-(p->amonut) <= total)

{

NextSolveTime = p->dealtime + i;

total = total + p->amonut;

EnQueue(deal\_queue, \*p);

}

else EnQueue(wait\_queue, \*p);

}

else EnQueue(deal\_queue, \*p);

}

printf("客户序号：%d 客户交易需要时间：%d 客户需要处理的钱：%d 银行余额：%d\n", p->num, p->dealtime, p->amonut, total);

return p;

}

}

Status leave\_event(ElemType\* e, int i)

{

event\* p = (event\*)malloc(sizeof(event));

p->type = 0;

p->num = e->num;

p->time = i;

p->money = e->amonut;

if (e->amonut < 0)p->index = 0;

else p->index = 1;

InsertLinkList(event\_link, \*p);

return OK;

}

Status deal\_event(int i)

{

if (NextSolveTime == i)//当前时间等于交易完成的时间

{

DeQueue(deal\_queue, leave\_Custom);//将该客户移出处理队列

int last\_money = total - leave\_Custom->amonut;//last\_money为用户移出后的银行余额

leave\_event(leave\_Custom, i);//将移出的客户和时间信息放进事件表

if (((leave\_Custom->amonut > 0) && wait\_queue->front != NULL))//准备离开用户是存钱的

{

int index = 0;

LQueue\* e = InitQueue();

int TempMoney = total;

while (TempMoney > last\_money && wait\_queue->front != NULL)

{

DeQueue(wait\_queue, wait\_Custom);//尝试将准备解决的用户移出等待队列

if (-(wait\_Custom->amonut) <= TempMoney)

{

EnQueue(e, \*wait\_Custom);//将准备解决的用户入队列e

TempMoney = TempMoney + wait\_Custom->amonut;//计算暂时的银行余额

}

else EnQueue(wait\_queue, \*wait\_Custom);//如果失败就将该用户重新移入等待队列的尾部

index++;

if (index >= wait\_queue->length)break;

}

CombineQueue(deal\_queue, e);//将队列e顺序接在处理队列的尾部

if (deal\_queue->front != NULL)

{

while (deal\_queue->front != NULL && -(deal\_queue->front->data.amonut) > total)//无法处理

{

DeQueue(deal\_queue, leave\_Custom);//从处理队列移出

EnQueue(wait\_queue, \*leave\_Custom);//加入等待队列

}

if (deal\_queue->front != NULL)

{

NextSolveTime = deal\_queue->front->data.dealtime+ NextSolveTime;//更新下个用户处理时间

total = total + deal\_queue->front->data.amonut;//更新银行余额

}

}

}

else//为取款/借款类型，其余同上

{

if (deal\_queue->front != NULL)

{

while (deal\_queue->front != NULL && -(deal\_queue->front->data.amonut) > total)

{

DeQueue(deal\_queue, leave\_Custom);

EnQueue(wait\_queue, \*leave\_Custom);

}

if (deal\_queue->front != NULL)

{

NextSolveTime = deal\_queue->front->data.dealtime + NextSolveTime;

total = total + deal\_queue->front->data.amonut;

}

}

}

}

}

void average\_StayTime(LinkList e, double last)

{

double sum = 0, come, leave;//sum为总等待时间，a为到达时间，b为离开时间

int AllCustom = 0;//总到达人数

int success = 0, deposit = 0, withdrawal = 0, DpAmount = 0, WdAmount = 0;//设置index来作为标记，为交易失败人数

LNode\* p = e->head;

for (int i = 1; i <= e->length; i++)

{

while (p != NULL)

{

if (p->data.num == i)

{

if (p->data.type == 1)

{

come = p->data.time;

success += 1;

AllCustom = AllCustom + 1;

if (p->data.index == 1)

{

DpAmount += 1;

deposit += 1;

}

if (p->data.index == 0)

{

WdAmount += 1;

withdrawal += 1;

}

}

if (p->data.type == 0)

{

leave = p->data.time;

success -= 1;

if (p->data.index == 1)deposit -= 1;

if (p->data.index == 0)withdrawal -= 1;

}

}

p = p->next;

}

p = e->head;

if (leave == -1.0)

leave = last;

sum = sum + leave - come;

leave = -1.0;

}

printf("\n\n");

printf("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n");

printf("\* 时间表信息： \*\n");

printf("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n");

printf("\* 全部用户数: %d 人 \*\n", AllCustom);

printf("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n");

printf("\* 总共消耗时间: %f 分钟 \*\n", sum);

printf("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n");

printf("\* 成功人数: %d 人 \*\n", AllCustom - success);

printf("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n");

printf("\* 取款/借款成功人数: %d 人 \*\n",WdAmount - withdrawal);

printf("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n");

printf("\* 存款/还款成功人数: %d 人 \*\n", DpAmount - deposit);

printf("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n");

printf("\* 客户平均逗留时间： %f 分钟 \*\n", sum / AllCustom);

printf("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n");

}

void menu()

{

printf("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n");

printf("| 银行模拟系统 |\n");

printf("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n");

printf("| 1、程序模拟 |\n");

printf("| 2、退出程序 |\n");

printf("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n");

printf("| 3122004665 |\n");

printf("| 22级计科(2)班 |\n");

printf("| 王展锐 |\n");

printf("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n");

printf("| 请输入: |\n");

}

void initial()

{

deal\_queue = NULL;

wait\_queue = NULL;

event\_link = NULL;

nextTime = 0;

NextSolveTime = 0;

CustomNumber = 1;

leave\_Custom = NULL;

wait\_Custom = NULL;

}

int main()

{

int choice=0;

while (1)

{

system("CLS");

menu();

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

if (choice >= 1 && choice <= 2)

{

switch (choice)

{

case 1:

{

leave\_Custom = (ElemType\*)malloc(sizeof(ElemType));

wait\_Custom = (ElemType\*)malloc(sizeof(ElemType));

deal\_queue = InitQueue();

wait\_queue = InitQueue();

event\_link = InitLinkList();

srand((unsigned int)time(NULL));//随机函数随电脑时间变化,确保不会随机数一样

int Runtime, MaxTimeGap, MinTimeGap, MaxTradeTime, MinTradeTime, i;

printf("请输入银行的运营时间（分钟）：");

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

while (Runtime <= 0)

{

printf("运营时间必须大于0\n");

printf("请输入银行的运营时间（分钟）：");

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

}

printf("请输入银行的运营金额（元）：");

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

while (total <= 0)

{

printf("初始运营金额必须大于0\n");

printf("请输入银行的运营金额（元）：");

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

}

printf("请输入客户最大交易金额（元）：");

scanf\_s("%d", &max\_money);

while (max\_money > total || max\_money <= 0)

{

printf("客户最大交易金额必须小于银行初始运营金额且大于0\n");

printf("请输入客户最大交易金额（元）：");

scanf\_s("%d", &max\_money);

}

printf("请输入客户最大到达时间间隔（分钟）：");

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

while (MaxTimeGap <= 0 || MaxTimeGap > Runtime)

{

printf("客户最大到达时间间隔必须大于0且小于运营时间\n");

printf("请输入客户最大到达时间间隔（分钟）：");

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

}

printf("请输入客户最小时间间隔（分钟）：");

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

while (MinTimeGap <= 0 || MinTimeGap > MaxTimeGap || MinTimeGap > Runtime)

{

printf("客户最小到达时间间隔必须大于0且小于最大到达时间间隔且小于运营时间\n");

printf("请输入客户最小时间间隔（分钟）：");

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

}

printf("请输入客户最大交易时间（分钟）：");

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

while (MaxTradeTime <= 0 || MaxTradeTime > Runtime)

{

printf("客户最大交易时间必须大于0且小于运营时间\n");

printf("请输入客户最大交易时间（分钟）：");

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

}

printf("请输入客户最小交易时间（分钟）：");

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

while (MinTradeTime <= 0 || MinTradeTime > MaxTradeTime || MinTradeTime > Runtime)

{

printf("客户最小交易时间必须大于0且小于最大交易时间且小于运营时间\n");

printf("请输入客户最小交易时间（分钟）：");

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

}

for (i = 0; i < Runtime; i++)

{

arrival\_event(i, MaxTimeGap, MinTimeGap, MaxTradeTime, MinTradeTime);

deal\_event(i);

}

printf("\n");

average\_StayTime(event\_link, Runtime - 1);

while (deal\_queue->front != NULL)

{

DeQueue(deal\_queue, leave\_Custom);

leave\_event(leave\_Custom, i);

}

while (wait\_queue->front != NULL)

{

DeQueue(wait\_queue, leave\_Custom);

leave\_event(leave\_Custom, i);

}

PrintLinkList(event\_link);

DestroyQueue(deal\_queue);

DestroyQueue(wait\_queue);

DestroyLinkList(event\_link);

system("pause");

break;

}

case 2:

{

printf("感谢您的使用！\n");

system("PAUSE");

goto end;

}

}

}

initial();

}

end:

return 0;

}