



**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 计算机科学与技术伏羲班**

**学 号 3222004895**

**学生姓名 钟雯敏**

**指导教师 苏 庆**

**2022 年 12 月 28 日**

**目录**

[**1. 需求分析 1**](#_Toc154932094)

[**2. 概要设计 2**](#_Toc154932095)

[**3. 详细设计 9**](#_Toc154932098)

[**4. 调试分析 23**](#_Toc154932099)

[**5. 用户使用说明 24**](#_Toc154932100)

[**6. 测试结果 24**](#_Toc154932101)

1. 需求分析

以无歧义的陈述说明程序设计的任务，强调的是程序要做什么？

**[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。

明确规定：

1. **输入的形式和输入值的范围；**

形式：从终端输入，依次输入银行拥有的款额（元）、营业时间（分钟）、相邻到达事件间隔时间的下界和上界、交易时间的下界和上界。

输入值范围：1.银行拥有的款额：正整数2.营业时间：正整数。3.相邻到达事件的间隔时间的下界和上界：正整数，上界大于下界。4.交易时间的下界和上界：正整数，上界大于下界。

1. **输出的形式；**

形式：在终端输出，输出平均逗留时间的数值。（可以在终端输出模拟过程的相关数据以及最后的事件表，也可以记录系统的运行状态、错误、警告等信息作为日志写入文件中）

1. **程序所能达到的功能；**

银行业务模拟系统的主要功能是通过事件驱动的方式模拟银行的日常运营情况，以便评估客户在银行内逗留的平均时间。

1.银行业务模拟：

根据输入的参数进行银行业务模拟，包括客户到达、处理业务、客户离开等过程。

处理队列中的客户，确保业务能够满足：如果客户取款或借款的金额超过银行的总款额，将客户加入第二队列等待。在处理完一项存款业务后，检查第二队列，看是否有客户可以满足其取款或借款需求。对每个客户，记录其到达时间和离开时间，计算总逗留时间。

2.计算平均逗留时间：

在模拟结束后，计算并输出平均逗留时间。

3.输出模拟过程的部分相关信息

4.输出模拟过程的所有事件或所有客户相关业务信息

**⑷ 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。**

**正确的输入及其输出结果：**

初始资金总额：10000元，

营业时间为600分钟，

客户到达时间间隔和交易时间的分布为合理的随机分布。

预期输出：正确的平均逗留时间等统计信息。

**正常业务流程**

初始资金总额：10000元

营业时间：600分钟

相邻到达事件的间隔时间下界：5分钟

相邻到达事件的间隔时间上界：30分钟

交易时间下界：5分钟

交易时间上界：30分钟

最大交易金额：3000元

平均逗留时间：36.000000

**含有错误的输入及其输出结果：**

初始资金总额为负数、营业时间为零、非法的时间间隔和交易时间的分布等。

预期输出：相应的错误提示信息，说明输入不符合要求。

**边界条件测试：**

测试初始资金总额为零、客户到达时间间隔和交易时间的范围极端情况等。

预期输出：系统能够在边界条件下正常运行，不出现错误或异常。

**1.（极端情况）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**

初始资金总额：10000元

营业时间：600分钟

相邻到达事件的间隔时间下界：5分钟

相邻到达事件的间隔时间上界：15分钟

交易时间下界：200分钟

交易时间上界：300分钟

最大交易金额：3000元

平均逗留时间：168.000000

**2. （极端情况）两个到达事件的间隔时间很长，而客户的交易时间很短**

初始资金总额：10000元

营业时间：600分钟

相邻到达事件的间隔时间下界：200分钟

相邻到达事件的间隔时间上界：300分钟

交易时间下界：5分钟

交易时间上界：15分钟

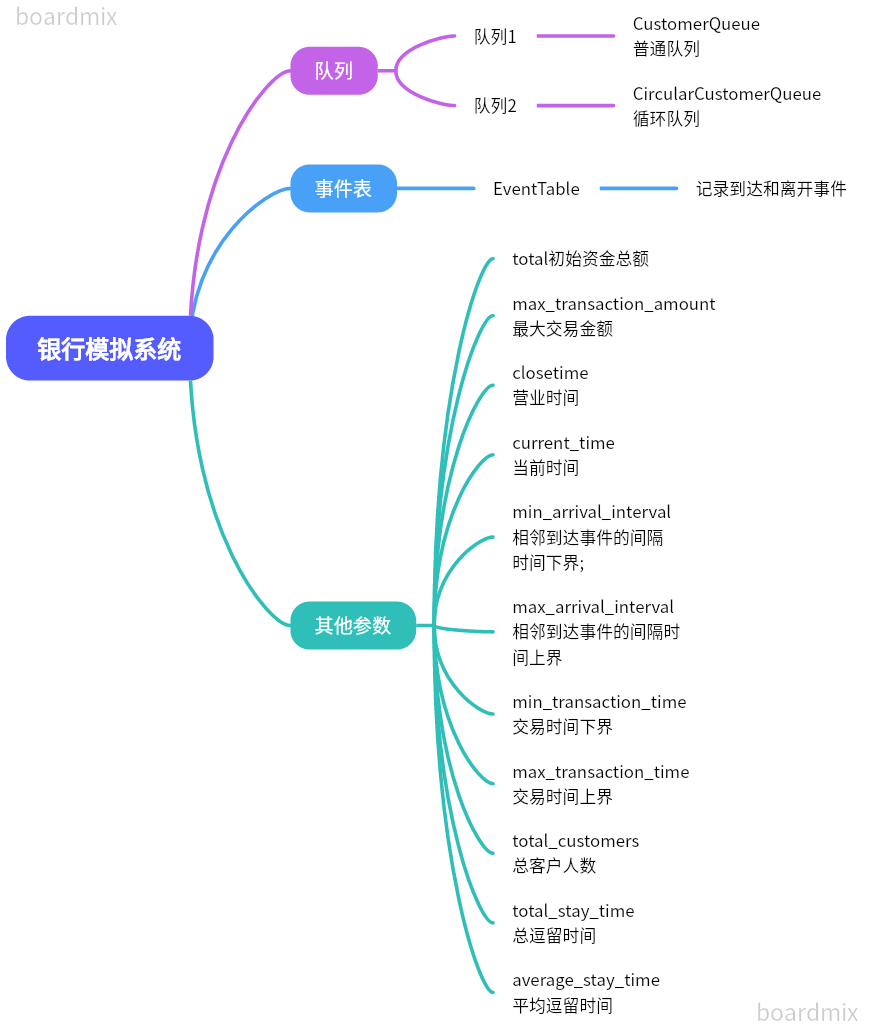
最大交易金额：3000元

平均逗留时间：6.000000

2. 概要设计

说明本程序中用到的所有数据类型的定义、主程序的流程以及各程序模块之间的调用关系。

所有数据类型的定义



// 定义银行模拟系统数据类型

typedef struct BankSimulation {

int total; // 初始资金总额

int closetime; // 营业时间

int current\_time; // 当前时间

int max\_transaction\_amount; // 最大交易金额

int min\_arrival\_interval; // 相邻到达事件的间隔时间下界

int max\_arrival\_interval; // 相邻到达事件的间隔时间上界

int min\_transaction\_time; // 交易时间下界

int max\_transaction\_time; // 交易时间上界

int total\_customers; // 总客户人数

CustomerQueue queue1; // 第一个队列

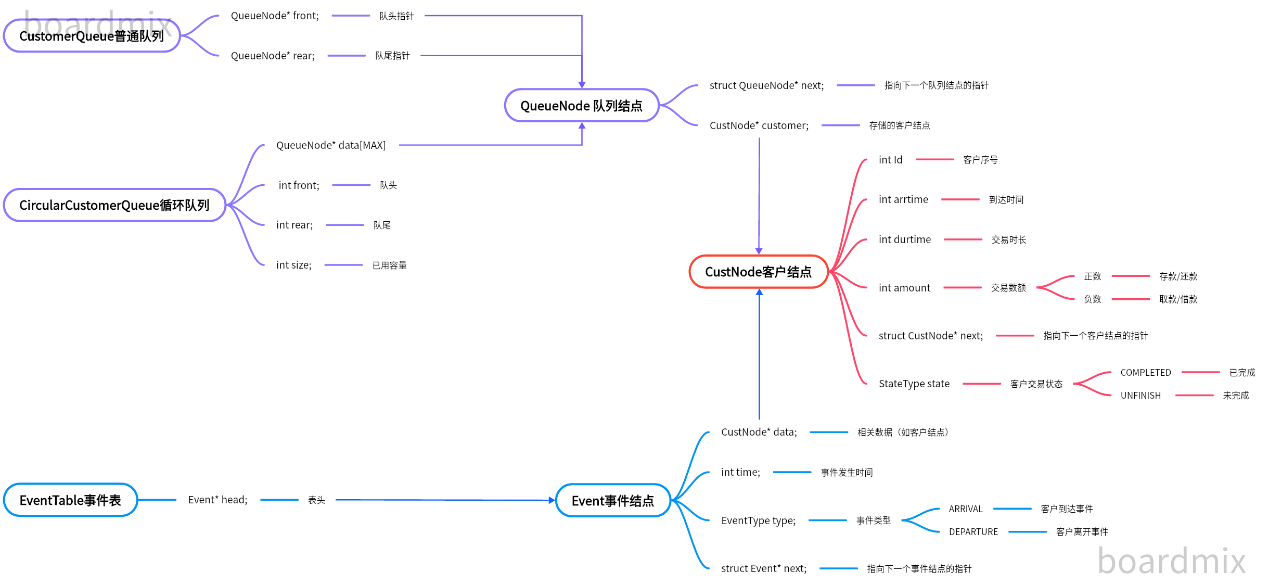
CircularCustomerQueue queue2; // 第二个队列

EventTable events; // 事件表

int total\_stay\_time; // 总逗留时间

double average\_stay\_time; // 平均逗留时间

} BankSimulation;

****BankSimulation 数据结构定义了一个银行模拟系统的各个参数和状态。total 表示银行初始资金总额，即模拟开始时银行拥有的款项。closetime 是银行的营业时间，规定了模拟的时间范围。current\_time 记录当前模拟的时间，随着模拟的进行而更新。max\_transaction\_amount用于规定交易的最大数额。min\_arrival\_interval 和 max\_arrival\_interval 规定了相邻客户到达事件的间隔时间范围，影响客户到达的时间分布。min\_transaction\_time 和 max\_transaction\_time 规定了客户办理业务所需的时间范围。total\_customers 记录总客户人数。queue1 和 queue2 分别表示两个服务窗口对应的队列，用于存储客户到达银行的顺序和无法暂时满足取款/借款的客户。events是事件表，按时间顺序存储不同类型的事件. total\_stay\_time 记录总逗留时间，用于计算平均逗留时间。average\_stay\_time存储最终的平均逗留时间结果。这些字段共同构成了一个全面的银行模拟系统，通过模拟客户到达和离开的事件，最终得出平均逗留时间的统计信息。

**1.客户结点**

// 定义客户交易状态

typedef enum {

COMPLETED,

UNFINISH

} StateType;

// 客户结点

typedef struct CustNode {

int Id;

int arrtime; // 到达时间

int durtime; // 交易时长

int amount; // 交易数额(+代表存款/还款，-代表取款/借款)

StateType state; // 交易状态

struct CustNode\* next;

} CustNode;

**2.队列**

//队列结点

typedef struct QueueNode {

CustNode\* customer; // 客户结点

struct QueueNode\* next; // 指向下一个队列结点的指针

} QueueNode;

//普通队列

typedef struct {

QueueNode\* front; // 队头指针

QueueNode\* rear; // 队尾指针

} CustomerQueue;

//循环队列

typedef struct {

QueueNode\*\* data; // 队列数据数组的指针

int front; // 队头

int rear; // 队尾

int size; // 队列大小

} CircularCustomerQueue;

**3.事件表**

// 定义事件类型枚举

typedef enum {

ARRIVAL, // 客户到达事件

DEPARTURE // 客户离开事件

} EventType;

// 定义事件结点数据类型

typedef struct Event {

int time; // 事件发生时间

EventType type; // 事件类型

CustNode\* data; // 相关数据（如客户结点）

struct Event\* next; // 指向下一个事件结点的指针

} Event;

// 定义事件表数据类型

typedef struct EventTable {

Event\* head;

} EventTable;

**4.动态数据类型**

// 定义动态数据类型

typedef struct DynamicDataType {

CustNode pool[MAX];

int top;

} DynamicDataType;

**5.其他定义**

// ANSI escape codes for terminal colors

#define ANSI\_COLOR\_RESET "\x1b[0m"

#define ANSI\_COLOR\_GREEN "\x1b[32m"

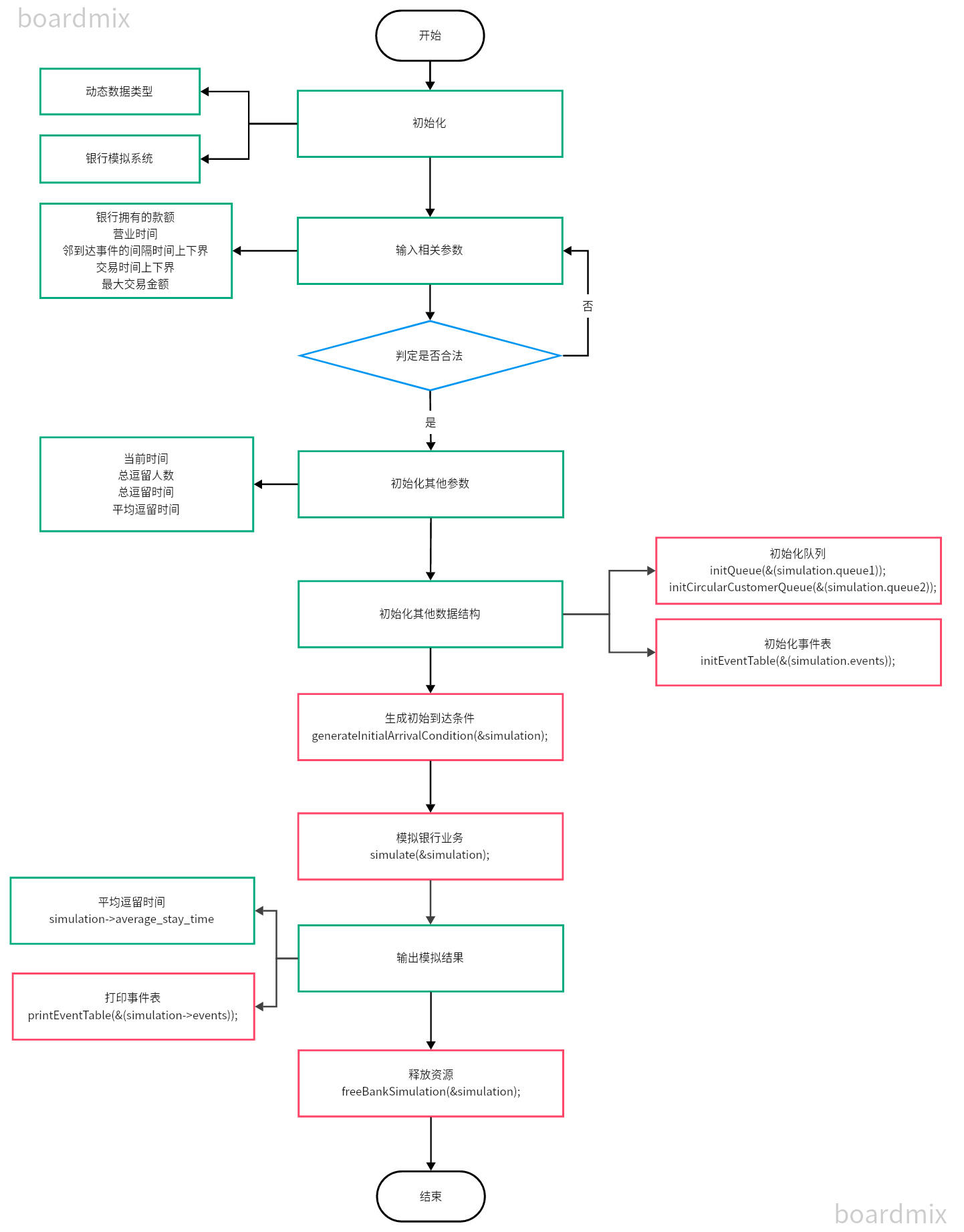
#define ANSI\_COLOR\_YELLOW "\x1b[33m"

#define ANSI\_COLOR\_RED "\x1b[31m"

#define MAX 600

#define SIZE 50

主程序的流程以及各程序模块之间的调用关系

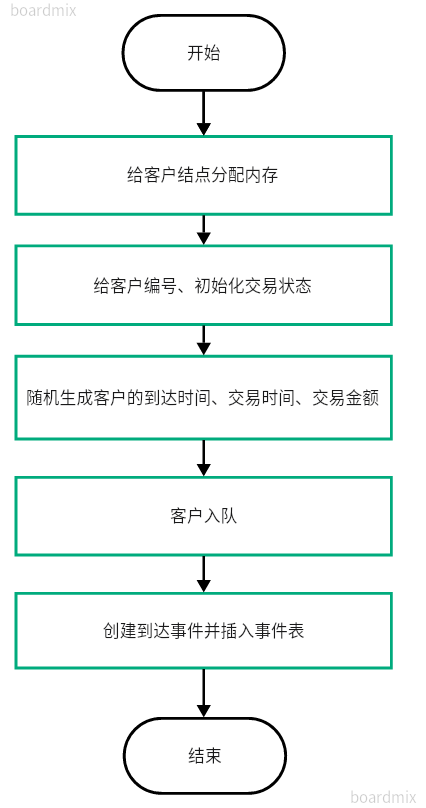
**主程序**

**主程序**是一个模拟银行业务的程序。程序一开始会先初始化动态数据类型结构体，接着进入程序目录。在每次循环中，用户会被提示输入银行的初始款项、营业时间、相邻到达事件的时间间隔、交易时间的范围、最大交易金额等模拟参数。用户输入后，程序初始化银行模拟系统，包括队列、事件表等数据结构。接着，程序生成初始到达条件，随机生成客户的到达时间、交易时间和金额，并插入到事件表和队列1中，统计总逗留人数并存储到银行模拟结构中。

模拟过程中，程序会处理事件表中的事件，包括客户到达和离开。在模拟过程中，依次处理队列1中客户的相关业务，根据客户的交易类型和金额，进行相应的操作。如果客户是**取款/借款**，检查银行资金总额能否满足需求，如果能满足则进行相关业务操作并生成离开事件，如果无法满足则安排客户在队列2中排队。如果客户是**存款/还款**，则进行相关业务操作，完成业务后检查银行资金总额是否能满足第二个队列客户的取款/借款需求，依次检查队列2中的客户，若可以满足则进行相关业务操作，若无法满足则安排客户再次排到队列2的队尾。（在生成离开事件时，对客户总逗留时间进行更新）

模拟结束后，程序输出模拟结果，包括平均逗留时间，打印事件表等等。最后，释放银行模拟系统占用的资源。

**生成初始到达条件void generateInitialArrivalCondition(BankSimulation\* simulation)**

****

1. 首次到达客户的生成：

通过调用 getRandomInt 函数生成第一个客户的到达时间、交易时间和交易金额，模拟客户到达的不确定性，并确保到达时间在营业时间内。

为第一个客户创建 CustNode 结点，包括到达时间、交易时间、交易金额等信息。将第一个客户入队，表示客户到达银行。创建到达事件，将其插入事件表中，并更新累计客户人数。

1. 生成后续到达事件：

利用循环，根据前一个到达事件的时间，结合相邻到达事件的间隔时间范围，生成后续到达事件。

为每个到达事件创建对应的客户结点，包括到达时间、交易时间、交易金额等信息。将客户入队，表示新客户到达银行。创建到达事件，将其插入事件表中，并更新累计客户人数。

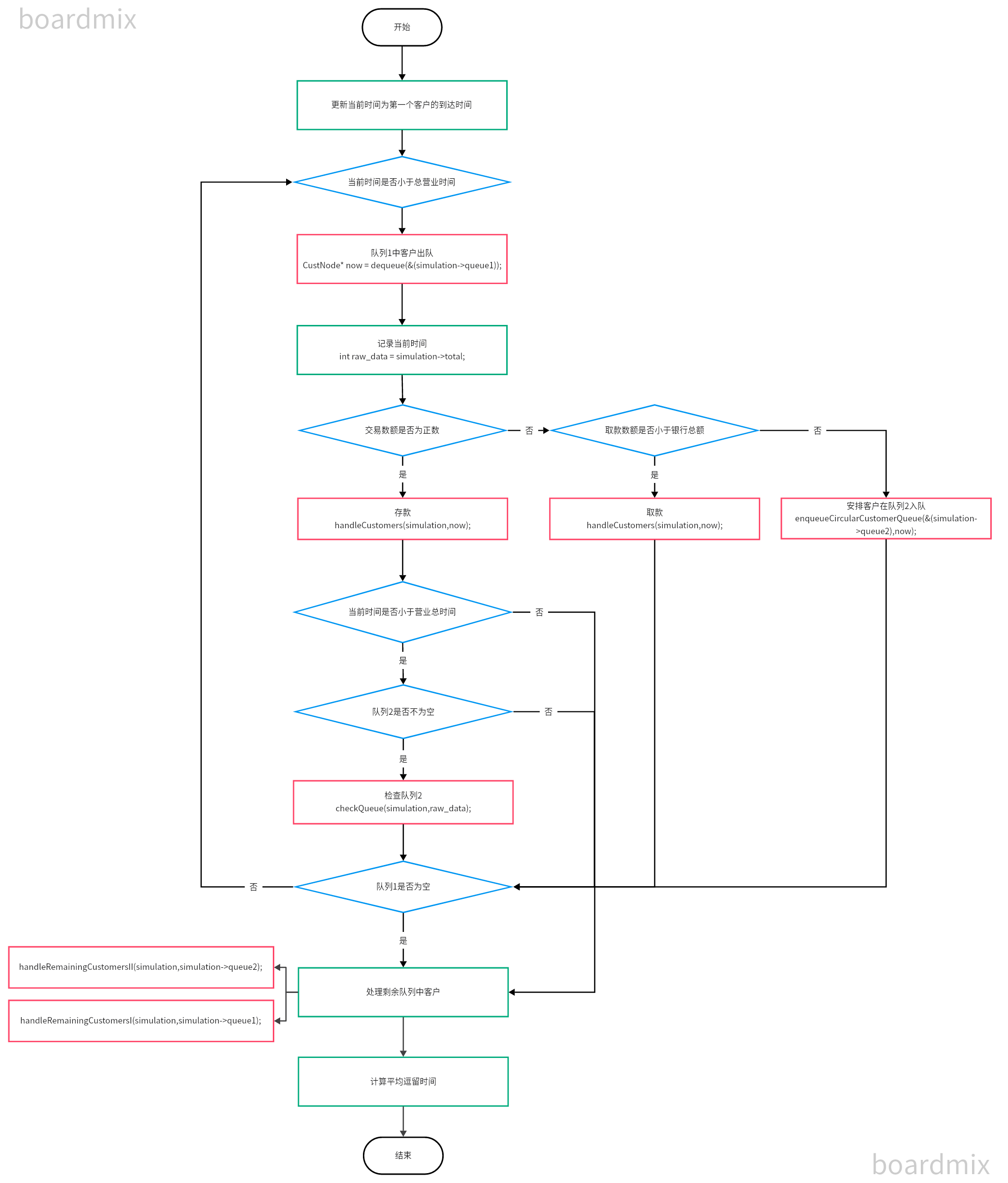
1. 数据结构的协同工作：

利用队列 (CustomerQueue) 存储客户到达的顺序，以便后续按照先进先出的原则处理客户。

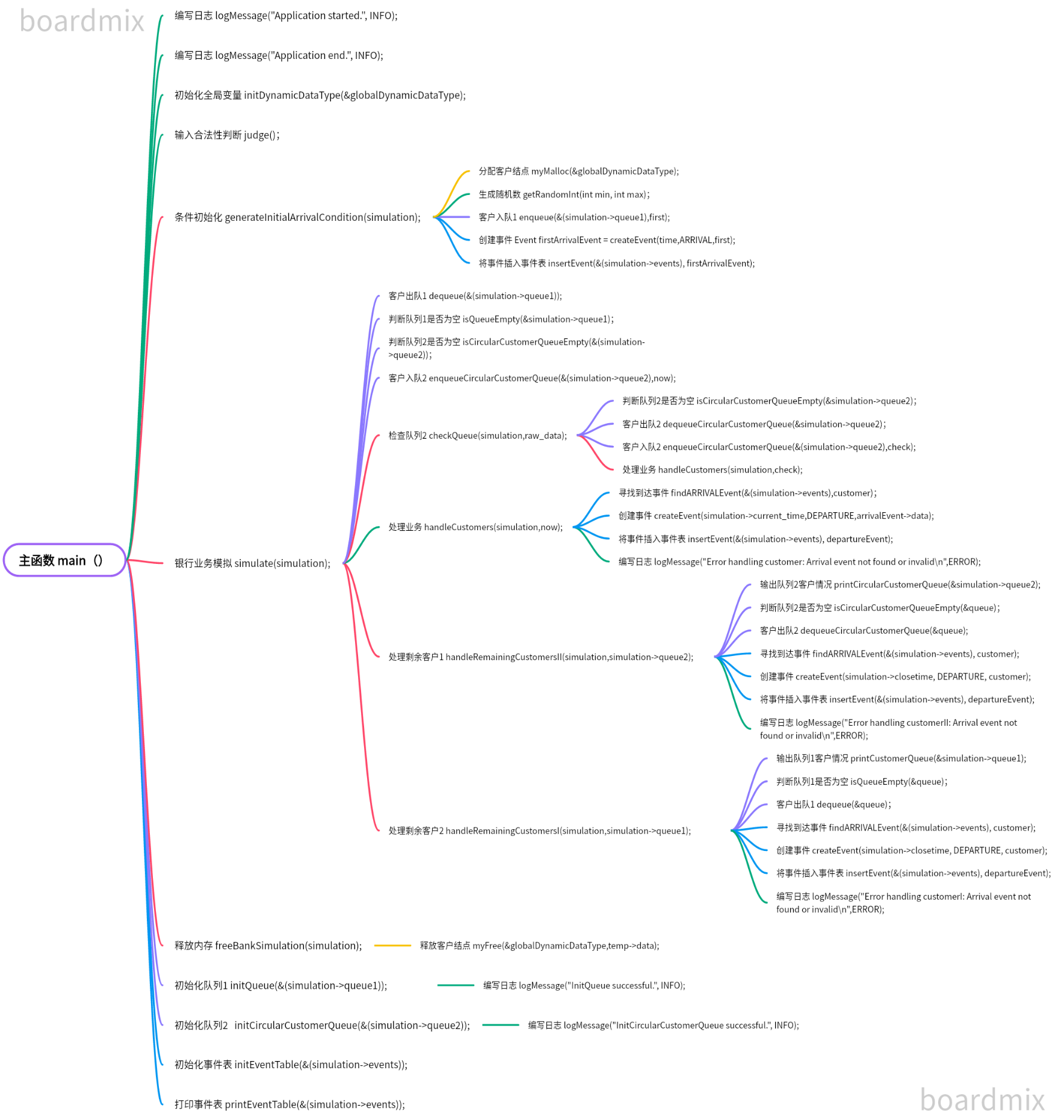
利用事件表 (EventTable) 存储所有事件，按照时间顺序组织。

**释放资源内存void freeBankSimulation(BankSimulation\* simulation)**

1.遍历事件表中的每个事件节点，释放其中的客户结点和事件节点本身。循环直到事件表为空。2.释放银行模拟系统结构体。

**主模拟程序**

3. 详细设计

实现概要设计中定义的所有数据类型，对每个操作只需要写出伪码算法；对主程序和其他模块也都需要写出伪码算法(伪码算法需要达到的详细程度为：按照伪码算法可以在计算机键盘直接输入高级程序设计语言程序)；画出函数和过程的调用关系图。

1. **队列**

// 初始化队列

void initQueue(CustomerQueue\* queue) {

queue->front = NULL;

queue->rear = NULL;

logMessage("InitQueue successful.", INFO);

}

// 入队操作

void enqueue(CustomerQueue\* queue, CustNode\* customer) {

QueueNode\* newNode = (QueueNode\*)malloc(sizeof(QueueNode));

if (newNode == NULL) {

logMessage("EnQueue memory allocation failed.", ERROR);

return;

}

newNode->customer = customer;

//printf("Enqueue amount:%d\t",newNode->customer->amount);

newNode->next = NULL;

if (isQueueEmpty(queue)) {

queue->front = newNode;

queue->rear = newNode;

} else {

queue->rear->next = newNode;

queue->rear = newNode;

}

logMessage("EnQueue successful.", INFO);

}

// 出队操作

CustNode\* dequeue(CustomerQueue\* queue) {

if (isQueueEmpty(queue)) {

logMessage("Dequeue operation on an empty queue.", WARNING);

return NULL;

}

QueueNode\* frontNode = queue->front;

CustNode\* customer = frontNode->customer;

queue->front = frontNode->next;

if (queue->front == NULL) {

queue->rear = NULL;

}

frontNode->next = NULL;

free(frontNode);

frontNode = NULL;

logMessage("Dequeue successful.", INFO);

return customer;

}

// 检查队列是否为空

int isQueueEmpty(CustomerQueue\* queue) {

return queue->front == NULL;

}

// 初始化循环队列

void initCircularCustomerQueue(CircularCustomerQueue\* queue) {

queue->data = (QueueNode\*\*)malloc(sizeof(QueueNode\*) \* MAX);

if (queue->data == NULL) {

logMessage("InitCircularCustomerQueue memory allocation failed.", ERROR);

return;

}

queue->front = 0;

queue->rear = 0;

queue->size = MAX;

logMessage("InitCircularCustomerQueue successful.", INFO);

}

// 判断循环队列是否为空

int isCircularCustomerQueueEmpty(const CircularCustomerQueue\* queue) {

return queue->front == queue->rear;

}

// 判断循环队列是否已满

int isCircularCustomerQueueFull(const CircularCustomerQueue\* queue) {

return (queue->rear + 1) % queue->size == queue->front;

}

// 入队

void enqueueCircularCustomerQueue(CircularCustomerQueue\* queue, CustNode\* element) {

if ((queue->rear+1)%queue->size == queue->front) {

logMessage("CirQueue is full. Enqueue operation failed.",ERROR);

return;

}

QueueNode\* newNode = (QueueNode\*)malloc(sizeof(QueueNode));

if (newNode == NULL) {

logMessage("CirEnQueue memory allocation failed.", ERROR);

return;

}

newNode->customer = element;

newNode->next = NULL;

queue->data[queue->rear] = newNode;

queue->rear = (queue->rear + 1) % queue->size;

logMessage("CirEnQueue successful.", INFO);

}

//出队

CustNode\* dequeueCircularCustomerQueue(CircularCustomerQueue\* queue) {

if (isCircularCustomerQueueEmpty(queue)) {

logMessage("CirQueue is empty. Dequeue operation failed.",ERROR);

return NULL;

}

QueueNode\* nodeToRemove = queue->data[queue->front];

queue->front = (queue->front + 1) % queue->size;

CustNode\* element = nodeToRemove->customer;

free(nodeToRemove);

nodeToRemove = NULL;

logMessage("CirDequeue successful.", INFO);

return element;

}

// 打印普通队列中的顾客情况

void printCustomerQueue(CustomerQueue\* queue) {

if (isQueueEmpty(queue)) {

printf("Customer Queue is empty.\n");

return;

}

printf("Customer Queue:\n");

QueueNode\* current = queue->front;

while (current != NULL) {

CustNode\* customer = current->customer;

printf("Customer ID: %d, Arrival Time: %d, Duration: %d, Amount: %d\n",

customer->Id, customer->arrtime, customer->durtime, customer->amount);

current = current->next;

}

}

// 打印循环队列中的顾客情况

void printCircularCustomerQueue(CircularCustomerQueue\* queue) {

if (isCircularCustomerQueueEmpty(queue)) {

printf("Circular Customer Queue is empty.\n");

return;

}

printf("Circular Customer Queue:\n");

int current = queue->front;

while (current != queue->rear) {

CustNode\* customer = queue->data[current]->customer;

printf("Customer ID: %d, Arrival Time: %d, Duration: %d, Amount: %d\n",

customer->Id, customer->arrtime, customer->durtime, customer->amount);

current = (current + 1) % queue->size;

}

}

1. **事件表**

// 初始化事件表

void initEventTable(EventTable\* et) {

et->head = NULL;

}

// 创建事件

Event createEvent(int time, EventType type, CustNode\* data) {

Event event;

event.time = time;

event.type = type;

event.data = myMalloc(&globalDynamicDataType);

//event.data = (CustNode\*)malloc(sizeof(CustNode));

if (event.data == NULL) {

logMessage("CreateEvent memory allocation failed.", ERROR);

}

memcpy(event.data, data, sizeof(CustNode));

event.next = NULL;

return event;

}

// 将事件按照发生时间插入事件表，保持事件表的有序性

void insertEvent(EventTable\* et, Event event) {

Event\* newEvent = (Event\*)malloc(sizeof(Event));

\*newEvent = event;

newEvent->next = NULL;

if (et->head == NULL || newEvent->time < et->head->time) {

newEvent->next = et->head;

et->head = newEvent;

return;

}

Event\* currentEvent = et->head;

while (currentEvent->next != NULL && currentEvent->next->time <= newEvent->time) {

currentEvent = currentEvent->next;

}

newEvent->next = currentEvent->next;

currentEvent->next = newEvent;

}

// 寻找到达事件

Event\* findARRIVALEvent(EventTable\* et, CustNode\* data) {

Event\* head = et->head;

while (head != NULL) {

if (head->type == ARRIVAL && memcmp(head->data, data, sizeof(CustNode)) == 0) {

return head;

}

head = head->next;

}

return NULL;

}

1. **动态数据类型**

// 初始化动态数据类型

void initDynamicDataType(DynamicDataType\* dynamicDataType) {

dynamicDataType->top = 0;

for (int i = 0; i < MAX - 1; ++i) {

dynamicDataType->pool[i].next = &(dynamicDataType->pool[i + 1]);

}

dynamicDataType->pool[MAX - 1].next = NULL;

}

// 分配内存

CustNode\* myMalloc(DynamicDataType\* dynamicDataType) {

if (dynamicDataType->top == MAX) {

return NULL;

}

CustNode\* allocatedNode = &(dynamicDataType->pool[dynamicDataType->top]);

dynamicDataType->top++;

return allocatedNode;

}

// 释放内存

void myFree(DynamicDataType\* dynamicDataType, CustNode\* node) {

if (dynamicDataType->top > 0) {

dynamicDataType->top--;

node->next = &(dynamicDataType->pool[dynamicDataType->top]);

}

}

1. **银行模拟系统**

// 生成初始到达条件

void generateInitialArrivalCondition(BankSimulation\* simulation) {

srand(time(NULL));

CustNode\* first = myMalloc(&globalDynamicDataType);

int time = getRandomInt(0, simulation->closetime/2);

first->Id = 1;

first->arrtime = time;

first->durtime=getRandomInt(simulation->min\_transaction\_time, simulation->max\_transaction\_time);

first->amount=getRandomInt(-simulation->max\_transaction\_amount,

simulation->max\_transaction\_amount);

first->state = UNFINISH;

printf(ANSI\_COLOR\_YELLOW "Generated random value: %d", first->amount);

printf(ANSI\_COLOR\_RESET "\n");

first->next = NULL;

enqueue(&(simulation->queue1),first);

Event firstArrivalEvent = createEvent(time,ARRIVAL,first);

printf(ANSI\_COLOR\_GREEN"arrivalEventAmount: %d\n",firstArrivalEvent.data->amount);

insertEvent(&(simulation->events), firstArrivalEvent);

simulation->total\_customers = 1;

first = NULL;

while (firstArrivalEvent.time < simulation->closetime) {

simulation->total\_customers++;

int time = firstArrivalEvent.time + getRandomInt(simulation->min\_arrival\_interval, simulation->max\_arrival\_interval);

if( time > simulation->closetime) break;

CustNode\* next = myMalloc(&globalDynamicDataType);

next->Id = simulation->total\_customers;

next->arrtime = time;

next->durtime=getRandomInt(simulation->min\_transaction\_time, simulation->max\_transaction\_time);

next->amount=getRandomInt(-simulation->max\_transaction\_amount, simulation->max\_transaction\_amount);

next->state = UNFINISH;

printf(ANSI\_COLOR\_YELLOW "Generated random value: %d", next->amount);

printf(ANSI\_COLOR\_RESET "\n");

next->next = NULL;

enqueue(&(simulation->queue1),next);

Event nextArrivalEvent = createEvent(time,ARRIVAL,next);

printf(ANSI\_COLOR\_GREEN"arrivalEventAmount: %d\n",nextArrivalEvent.data->amount);

insertEvent(&(simulation->events), nextArrivalEvent);

next = NULL;

firstArrivalEvent = nextArrivalEvent;

}

}

// 模拟银行业务

void simulate(BankSimulation\* simulation) {

simulation->current\_time = simulation->queue1.front->customer->arrtime;

// 处理事件直到银行关门

while (simulation->current\_time < simulation->closetime) {

CustNode\* now = dequeue(&(simulation->queue1));

if(simulation->current\_time<now->arrtime) simulation->current\_time = now->arrtime;

int raw\_data = simulation->total;

if(now->amount>=0){

handleCustomers(simulation,now);

if(simulation->current\_time > simulation->closetime) break;

if(!isCircularCustomerQueueEmpty(&(simulation->queue2))){

checkQueue(simulation,raw\_data);

}

}else{

int amount = (-1)\*now->amount;

if(amount<=simulation->total){

handleCustomers(simulation,now);

}else{

enqueueCircularCustomerQueue(&(simulation->queue2),now);

}

}

if(isQueueEmpty(&simulation->queue1)) break;

now = NULL;

}

handleRemainingCustomersII(simulation,simulation->queue2);

handleRemainingCustomersI(simulation,simulation->queue1);

if (simulation->total\_customers > 0) {

simulation->average\_stay\_time=(double)(simulation->total\_stay\_time / simulation->total\_customers);

}

}

//检查第二个队列

void checkQueue(BankSimulation\* simulation,int raw\_data){

CircularCustomerQueue queue = simulation->queue2;

int Id = queue.data[queue.front]->customer->Id;

while(!isCircularCustomerQueueEmpty(&simulation->queue2)){

CustNode\* check = dequeueCircularCustomerQueue(&simulation->queue2);

int Idd = queue.data[queue.front]->customer->Id;

int amount = (-1)\*check->amount;

if(amount<=simulation->total){

handleCustomers(simulation,check);

}else{

enqueueCircularCustomerQueue(&(simulation->queue2),check);

}

if(simulation->total<=raw\_data) break;

if(Id == Idd) break;

check = NULL;

}

}

//处理业务

void handleCustomers(BankSimulation\* simulation,CustNode\* customer){

simulation->total += customer->amount;

simulation->current\_time += customer->durtime;

Event\* arrivalEvent = findARRIVALEvent(&(simulation->events),customer);

arrivalEvent->data->state = COMPLETED;

Event departureEvent;

if(simulation->current\_time>simulation->closetime)

departureEvent = createEvent(simulation->closetime,DEPARTURE,arrivalEvent->data);

else{

departureEvent= createEvent(simulation->current\_time,DEPARTURE,arrivalEvent->data);

}

if (arrivalEvent != NULL && arrivalEvent->data != NULL) {

int stay\_time = departureEvent.time - arrivalEvent->time;

simulation->total\_stay\_time += stay\_time;

printf(ANSI\_COLOR\_GREEN"departureEventAmount: %d\n",departureEvent.data->amount);

insertEvent(&(simulation->events), departureEvent);

} else {

logMessage("Error handling customer: Arrival event not found or invalid\n",ERROR);

}

}

//处理剩余顾客

void handleRemainingCustomersI(BankSimulation\* simulation,CustomerQueue queue){

printCustomerQueue(&simulation->queue1);

while (!isQueueEmpty(&queue)) {

CustNode\* customer = dequeue(&queue);

Event\* arrivalEvent = findARRIVALEvent(&(simulation->events), customer);

if (arrivalEvent != NULL && arrivalEvent->data != NULL) {

EventdepartureEvent=createEvent(simulation->closetime,DEPARTURE, customer);

insertEvent(&(simulation->events), departureEvent);

printf(ANSI\_COLOR\_GREEN"departureEventAmount: %d",departureEvent.data->amount);

printf(ANSI\_COLOR\_RESET "\n");

int stay\_time = departureEvent.time - arrivalEvent->time;

simulation->total\_stay\_time += stay\_time;

free(arrivalEvent->data);

free(customer);

} else {

logMessage("Error handling customerI: Arrival event not found or invalid\n",ERROR);

}

customer = NULL;

arrivalEvent = NULL;

}

printf(ANSI\_COLOR\_RESET "end~\n");

}

//处理剩余顾客

void handleRemainingCustomersII(BankSimulation\* simulation,CircularCustomerQueue queue){

printCircularCustomerQueue(&simulation->queue2);

while (!isCircularCustomerQueueEmpty(&queue)) {

CustNode\* customer = dequeueCircularCustomerQueue(&queue);

Event\* arrivalEvent = findARRIVALEvent(&(simulation->events), customer);

if (arrivalEvent != NULL && arrivalEvent->data != NULL) {

EventdepartureEvent=createEvent(simulation->closetime,DEPARTURE, customer);

insertEvent(&(simulation->events), departureEvent);

printf(ANSI\_COLOR\_GREEN"departureEventAmount: %d\n",departureEvent.data->amount);

int stay\_time = departureEvent.time - arrivalEvent->time;

simulation->total\_stay\_time += stay\_time;

free(arrivalEvent->data);

free(customer);

} else {

logMessage("Error handling customerII: Arrival event not found or invalid\n",ERROR);

}

customer = NULL;

arrivalEvent = NULL;

}

}

//释放资源

void freeBankSimulation(BankSimulation\* simulation) {

Event\* currentEvent = simulation->events.head;

while (currentEvent != NULL) {

Event\* temp = currentEvent;

currentEvent = currentEvent->next;

myFree(temp->data);

free(temp);

}

simulation->events.head = NULL;

free(simulation);

}

//打印事件表

void printEventTable(const EventTable\* et) {

if (et == NULL || et->head == NULL) {

printf("Event table is empty.\n");

return;

}

printf("Event Table:\n");

printf("Time\tType\t\tData\t\t\tId\tDeal\n");

Event\* currentEvent = et->head;

while (currentEvent != NULL) {

printf("%d\t", currentEvent->time);

// Set color based on event type

if (currentEvent->type == ARRIVAL) {

printf(ANSI\_COLOR\_YELLOW "ARRIVAL\t\tCustomer Arrival\t");

} else if(currentEvent->data->state == COMPLETED){

printf(ANSI\_COLOR\_GREEN "DEPARTURE\tCustomer Departure\t");

} else {

printf(ANSI\_COLOR\_RED "DEPARTURE\tCustomer Departure\t");

}

CustNode\* now = currentEvent->data;

printf("%d\t",now->Id);

printf("%d\t" ANSI\_COLOR\_RESET "\n", now->amount);

currentEvent = currentEvent->next;

}

}

1. **其他函数**

int getRandomInt(int min, int max) {

int randomValue = rand() % (max - min + 1) + min;

return randomValue;

}

// 判断输入合法性，确保输入的整数是非负数

int judge() {

int num = 0;

char input[10];

while (1) {

scanf("%s", input);

int result = sscanf(input, "%d", &num);

if (result == 1 && num >= 0) {

return num;

}

printf("输入有误，请重新输入（要求非负整数）：");

}

}

// 输出银行当前状态

void displayBankSimulationProcess(const BankSimulation\* simulation) {

printf("当前时间：%d\n", simulation->current\_time);

printf("总款额：%d\n", simulation->total);

printf("总逗留时间：%d\n",simulation->total\_stay\_time);

printf("逗留总人数：%d\n",simulation->total\_customers);

}

// 记录日志的函数

void logMessage(const char\* message, enum LogLevel log\_level) {

FILE\* logFile = fopen("app.log", "a");

if (logFile == NULL) {

printf("Error opening log file.\n");

return;

}

// 获取当前时间

time\_t rawtime;

struct tm\* timeinfo;

char buffer[80];

time(&rawtime);

timeinfo = localtime(&rawtime);

strftime(buffer, sizeof(buffer), "%Y-%m-%d %H:%M:%S", timeinfo);

// 输出时间和日志消息到文件

fprintf(logFile, "[%s] ", buffer);

switch (log\_level) {

case DEBUG:

fprintf(logFile, "[DEBUG] ");

break;

case INFO:

fprintf(logFile, "[INFO] ");

break;

case WARNING:

fprintf(logFile, "[WARNING] ");

break;

case ERROR:

fprintf(logFile, "[ERROR] ");

break;

}

fprintf(logFile, "%s\n", message);

// 关闭日志文件

fclose(logFile);

}

// 清屏函数

void clearScreen() {

#ifdef \_WIN32

system("cls");

#else

system("clear");

#endif

}

1. **主函数**

int main() {

logMessage("Application started.", INFO);

BankSimulation\* simulation = malloc(sizeof(BankSimulation));

simulation->total = -1;

if (simulation == NULL) {

printf("BANKOVERFLOW.\n");

return 0;

}

initDynamicDataType(&globalDynamicDataType);

int flag = 0;

int choice;

do {

clearScreen();

displayMenu();

choice = judge();

switch (choice) {

case 1:

flag = 1;

printf("请输入银行拥有的款额(元)：");

int total = judge();

printf("请输入营业时间(分钟)：");

int closetime = judge();

printf("请输入相邻到达事件的间隔时间的下界(分钟)：");

int min\_arrival = 0;

while (min\_arrival == 0 || min\_arrival > closetime) {

if (min\_arrival > closetime) printf("超出营业时间，请重新输入:");

min\_arrival = judge();

}

printf("请输入相邻到达事件的间隔时间的上界(分钟)：");

int max\_arrival = 0;

while (max\_arrival == 0 || max\_arrival > closetime || max\_arrival < min\_arrival) {

if (max\_arrival > closetime) printf("超出营业时间，请重新输入:");

if (max\_arrival != 0 && max\_arrival < min\_arrival) printf("上界小于下界，请重新输入:");

max\_arrival = judge();

}

printf("请输入交易时间的下界(分钟)：");

int min\_transaction\_time = 0;

while (min\_transaction\_time == 0 || min\_transaction\_time > closetime) {

if (min\_transaction\_time > closetime) printf("超出营业时间，请重新输入:");

min\_transaction\_time = judge();

}

printf("请输入交易时间的上界(分钟)：");

int max\_transaction\_time = 0;

while (max\_transaction\_time == 0 || max\_transaction\_time > closetime || max\_transaction\_time < min\_transaction\_time) {

if (max\_transaction\_time > closetime) printf("超出营业时间，请重新输入:");

if (max\_transaction\_time != 0 && max\_transaction\_time < min\_transaction\_time) printf("上界小于下界，请重新输入:");

max\_transaction\_time = judge();

}

printf("请输入最大交易金额(元)：");

int amount = judge();

break;

case 2:

if(flag == 0) {

printf("您还未设置参数，请先进行设置\n");

break;

}

if(simulation == NULL) simulation = malloc(sizeof(BankSimulation));

simulation->total = total;

simulation->closetime = closetime;

simulation->min\_arrival\_interval = min\_arrival;

simulation->max\_arrival\_interval = max\_arrival;

simulation->min\_transaction\_time = min\_transaction\_time;

simulation->max\_transaction\_time = max\_transaction\_time;

simulation->max\_transaction\_amount = amount;

simulation->current\_time = 0;

simulation->total\_customers = 0;

simulation->total\_stay\_time = 0.0;

simulation->average\_stay\_time = 0.0;

initQueue(&(simulation->queue1));

initCircularCustomerQueue(&(simulation->queue2));

initEventTable(&(simulation->events));

generateInitialArrivalCondition(simulation);

simulate(simulation);

break;

case 3:

if(simulation->total == -1) {

printf("您还未设置参数，请先进行设置\n");

break;

}

if(simulation->events.head == NULL) {

printf("您还未进行初始化，请先进行设置\n");

break;

}

printf("平均逗留时间：%lf\n", simulation->average\_stay\_time);

printEventTable(&(simulation->events));

freeBankSimulation(simulation);

simulation = NULL;

break;

case 4:

break;

default:

printf("无效的选择，请重新输入。\n");

}

// 等待用户输入后继续

printf("按 Enter 键继续...");

getchar();

getchar();

} while (choice != 4);

logMessage("Application end.", INFO);

return 0;

}

4. 调试分析

内容包括：

1. **调试过程中遇到的问题是如何解决的以及对设计与实现的回顾讨论和分析；**

**调试过程中遇到的问题及解决方法：**

1.内存泄漏问题： 在程序中使用了动态内存分配（malloc）来创建结点，但在释放资源的函数中未正确释放这些内存，导致内存泄漏。解决方法是在释放资源的函数中进行遍历，逐个释放动态分配的内存。

2.队列相关问题： 在队列的实现中，需要确保队列的入队和出队操作正确无误。在测试时，发现有时候客户结点未能正确入队或出队，导致队列状态不一致。通过仔细检查入队和出队的逻辑，确保指针的正确移动，解决了这个问题。

3.循环队列大小问题： 在初始化循环队列时，未正确设置队列的大小，导致在判断队列是否已满时出现错误。解决方法是在初始化循环队列时设置正确的大小。

4.逻辑错误： 在模拟银行业务过程中，需要确保每个事件按时间顺序被正确处理。在事件处理时，需要注意事件表的有序性。通过仔细检查事件处理逻辑，确保按照时间顺序处理事件，解决了这个问题。

**对设计与实现的回顾讨论和分析：**

1.数据结构的选择： 使用队列和循环队列来管理客户的到达和业务处理是合适的选择。队列结构符合银行业务中先来先服务的特点，而循环队列则能够有效管理第二队列的客户。

2.事件驱动模拟系统设计： 通过事件表管理到达和离开事件，使得系统能够按照事件发生的时间顺序进行模拟。这种设计能够有效模拟实际业务中的时间流逝。

3.随机性的引入： 通过使用随机数生成客户的到达时间、交易时间和金额，增加了模拟的真实性。在测试时需要确保随机数生成的范围和分布是符合实际情况的。

4.模拟参数的输入： 在主程序中，通过与用户的交互方式输入模拟参数，使得程序更加灵活。用户可以根据具体情况输入不同的参数，进行多种场景的模拟。

**⑵ 算法的时空分析（包括基本操作和其他算法的时间复杂度和空间复杂度的分析）和改进设想；**

|  |  |  |  |
| --- | --- | --- | --- |
| 数据类型 | 操作 | 时间复杂度 | 空间复杂度 |
| 队列 | 入队 | O(1) | O(n) |
| 出队 | O(1) | O(n) |
| 事件表 | 插入 | O(n) | O(n) |

改进设想：

1.优化事件表操作： 考虑使用更高效的数据结构，如平衡二叉搜索树（BST）或堆，以提高事件表的插入和删除效率。这样可以将事件表的插入操作从 O(n) 降低到 O(log n)。

2.代码结构优化： 考虑将一些功能相近的代码封装成函数，提高代码的可读性和可维护性。

3.内存管理优化： 在释放资源的函数中，可以考虑采用更灵活的内存管理方式，例如对象池，减少频繁的内存分配和释放。

4.更多的异常处理： 考虑在关键操作中添加更多的异常处理，以应对一些意外情况，提高程序的健壮性。

**⑶ 经验和体会等（可选）。**

在整个银行业务模拟系统的开发过程中，我积累了一些宝贵的经验。首先，模块化设计对于系统的可维护性和可扩展性至关重要。将系统划分为多个模块，每个负责特定功能，有助于提高代码的可读性和理解性。抽象数据类型的运用，如队列、事件表、静态链栈等，进一步促进了模块化设计的实现。

动态内存管理在整个开发过程中显得至关重要。正确的内存分配和释放是确保程序正常运行的基本要素。使用malloc和free函数时，必须小心防止内存泄漏，合理地管理内存空间。

随机数的生成对于模拟系统尤为关键。它直接影响客户到达时间、交易金额等的随机性，从而影响了模拟系统的真实性。选择合适的随机数生成函数并妥善处理随机性是系统设计的关键一环。

在编码过程中，我注重异常处理机制。对于一些可能导致系统异常的情况，及时报错并退出程序是保证系统稳定性的有效手段。此外，通过在模拟系统的不同阶段输出相关信息，或者将信息或异常写入日志中，有助于调试和验证系统的正确性。

在完成基本功能后，我思考如何验证结果，最后想到可以实现打印队列中客户的信息、打印事件表来展现整个模拟过程。实现后，我又进一步思考如何将结果更加可视化。通过改变信息输出的颜色，能够区分不同情况下的事件相关数据进而验证模拟过程的可靠性。

整个开发过程中，不断进行调试和优化是一项必不可少的工作。通过发现问题、改进代码，逐步提高系统性能和可靠性。最终，这个模拟系统的开发过程让我更深刻地认识了数据结构和算法在实际应用中的重要性，同时也提高了我对软件开发中最佳实践的理解。

5. 用户使用说明

说明如何使用你编写的程序，详细列出每一步的操作步骤；

1.输入模拟参数： 程序会提示用户输入银行的初始资金总额、营业时间、相邻到达事件的间隔时间下界和上界、交易时间的下界和上界等参数。按照提示输入相关数值并按 Enter 键确认。

2.观察模拟过程： 程序会开始模拟银行业务，显示模拟的各个阶段的信息，包括客户到达、处理业务、离开银行等过程。这些信息将在终端中展示。（相同的参数可以多次模拟）

3.查看结果： 模拟结束后，程序会输出平均逗留时间等结果。可以在终端中查看这些输出信息。

4.关闭程序： 在程序完成模拟后，按照提示，可以按任意键关闭程序。

6. 测试结果

列出你的测试结果，包括输入和输出。这里的测试数据应该完整和严格，最好多于需求分析中所列。

**情景一：正常业务流程**

初始资金总额：10000元

营业时间：600分钟

相邻到达事件的间隔时间下界：5分钟

相邻到达事件的间隔时间上界：30分钟

交易时间下界：5分钟

交易时间上界：30分钟

最大交易金额：3000元

平均逗留时间：36.000000

Event Table:

Time Type Data Id Deal

90 ARRIVAL Customer Arrival 1 -2359

106 DEPARTURE Customer Departure 1 -2359

117 ARRIVAL Customer Arrival 2 1834

131 ARRIVAL Customer Arrival 3 2910

136 ARRIVAL Customer Arrival 4 -1051

137 DEPARTURE Customer Departure 2 1834

163 ARRIVAL Customer Arrival 5 -453

165 DEPARTURE Customer Departure 3 2910

185 ARRIVAL Customer Arrival 6 213

194 DEPARTURE Customer Departure 4 -1051

195 ARRIVAL Customer Arrival 7 -2148

206 DEPARTURE Customer Departure 5 -453

214 ARRIVAL Customer Arrival 8 2965

229 DEPARTURE Customer Departure 6 213

237 DEPARTURE Customer Departure 7 -2148

242 ARRIVAL Customer Arrival 9 -999

249 ARRIVAL Customer Arrival 10 -2332

265 DEPARTURE Customer Departure 8 2965

272 ARRIVAL Customer Arrival 11 -2806

276 DEPARTURE Customer Departure 9 -999

286 ARRIVAL Customer Arrival 12 181

296 DEPARTURE Customer Departure 10 -2332

314 ARRIVAL Customer Arrival 13 -1601

318 DEPARTURE Customer Departure 11 -2806

324 DEPARTURE Customer Departure 12 181

331 DEPARTURE Customer Departure 13 -1601

343 ARRIVAL Customer Arrival 14 -437

348 DEPARTURE Customer Departure 14 -437

362 ARRIVAL Customer Arrival 15 -1170

372 ARRIVAL Customer Arrival 16 -1425

384 ARRIVAL Customer Arrival 17 -240

384 DEPARTURE Customer Departure 15 -1170

401 DEPARTURE Customer Departure 16 -1425

407 ARRIVAL Customer Arrival 18 2332

427 ARRIVAL Customer Arrival 19 -2137

431 DEPARTURE Customer Departure 17 -240

438 DEPARTURE Customer Departure 18 2332

442 ARRIVAL Customer Arrival 20 -2196

467 DEPARTURE Customer Departure 19 -2137

470 ARRIVAL Customer Arrival 21 2479

490 ARRIVAL Customer Arrival 22 -1924

498 DEPARTURE Customer Departure 21 2479

502 ARRIVAL Customer Arrival 23 341

510 ARRIVAL Customer Arrival 24 99

518 DEPARTURE Customer Departure 20 -2196

526 ARRIVAL Customer Arrival 25 185

540 DEPARTURE Customer Departure 23 341

552 ARRIVAL Customer Arrival 26 729

553 DEPARTURE Customer Departure 24 99

565 ARRIVAL Customer Arrival 27 -995

569 DEPARTURE Customer Departure 22 -1924

592 DEPARTURE Customer Departure 25 185

593 ARRIVAL Customer Arrival 28 -2024

598 ARRIVAL Customer Arrival 29 353

600 DEPARTURE Customer Departure 26 729

600 DEPARTURE Customer Departure 27 -995

600 DEPARTURE Customer Departure 28 -2024

600 DEPARTURE Customer Departure 29 353

结果分析：输出符合预期。输入在合理范围内，按照银行流程正常运转，先到达的顾客先进行业务的处理，取款/借款和存款/还款都能较好地满足。客户平均逗留时间合理。

**情景二：短时间内较高频到达**

初始资金总额：10000元

营业时间：600分钟

相邻到达事件的间隔时间下界：5分钟

相邻到达事件的间隔时间上界：15分钟

交易时间下界：5分钟

交易时间上界：30分钟

最大交易金额：3000元

平均逗留时间：118.000000

Event Table:

Time Type Data Id Deal

121 ARRIVAL Customer Arrival 1 987

129 ARRIVAL Customer Arrival 2 -1235

133 DEPARTURE Customer Departure 1 987

141 ARRIVAL Customer Arrival 3 632

143 DEPARTURE Customer Departure 2 -1235

156 ARRIVAL Customer Arrival 4 -455

165 ARRIVAL Customer Arrival 5 1515

171 DEPARTURE Customer Departure 3 632

172 ARRIVAL Customer Arrival 6 180

179 ARRIVAL Customer Arrival 7 -2112

183 DEPARTURE Customer Departure 4 -455

184 ARRIVAL Customer Arrival 8 1528

190 ARRIVAL Customer Arrival 9 -1603

202 DEPARTURE Customer Departure 5 1515

203 ARRIVAL Customer Arrival 10 -2144

215 ARRIVAL Customer Arrival 11 -1462

222 ARRIVAL Customer Arrival 12 -149

223 DEPARTURE Customer Departure 6 180

227 ARRIVAL Customer Arrival 13 -2723

233 ARRIVAL Customer Arrival 14 1413

238 ARRIVAL Customer Arrival 15 -399

239 DEPARTURE Customer Departure 7 -2112

250 ARRIVAL Customer Arrival 16 -401

262 DEPARTURE Customer Departure 8 1528

263 ARRIVAL Customer Arrival 17 -2193

272 ARRIVAL Customer Arrival 18 -2752

274 DEPARTURE Customer Departure 9 -1603

281 DEPARTURE Customer Departure 10 -2144

284 ARRIVAL Customer Arrival 19 552

292 ARRIVAL Customer Arrival 20 1120

292 DEPARTURE Customer Departure 11 -1462

301 ARRIVAL Customer Arrival 21 2886

309 ARRIVAL Customer Arrival 22 -292

310 DEPARTURE Customer Departure 12 -149

323 ARRIVAL Customer Arrival 23 -2528

327 DEPARTURE Customer Departure 13 -2723

336 ARRIVAL Customer Arrival 24 2636

346 DEPARTURE Customer Departure 14 1413

348 ARRIVAL Customer Arrival 25 -805

358 ARRIVAL Customer Arrival 26 -311

366 ARRIVAL Customer Arrival 27 -1538

373 DEPARTURE Customer Departure 15 -399

374 ARRIVAL Customer Arrival 28 -1596

389 ARRIVAL Customer Arrival 29 2224

395 ARRIVAL Customer Arrival 30 151

396 DEPARTURE Customer Departure 16 -401

404 ARRIVAL Customer Arrival 31 -685

413 ARRIVAL Customer Arrival 32 -1633

423 DEPARTURE Customer Departure 17 -2193

424 ARRIVAL Customer Arrival 33 -2318

438 ARRIVAL Customer Arrival 34 2644

450 DEPARTURE Customer Departure 19 552

453 ARRIVAL Customer Arrival 35 -451

465 ARRIVAL Customer Arrival 36 -799

474 DEPARTURE Customer Departure 20 1120

475 ARRIVAL Customer Arrival 37 2779

486 ARRIVAL Customer Arrival 38 580

495 ARRIVAL Customer Arrival 39 1975

496 DEPARTURE Customer Departure 18 -2752

508 ARRIVAL Customer Arrival 40 -2914

522 ARRIVAL Customer Arrival 41 -723

522 DEPARTURE Customer Departure 21 2886

527 ARRIVAL Customer Arrival 42 -415

533 ARRIVAL Customer Arrival 43 1446

540 DEPARTURE Customer Departure 22 -292

544 ARRIVAL Customer Arrival 44 -2722

553 ARRIVAL Customer Arrival 45 1609

558 ARRIVAL Customer Arrival 46 -2417

563 DEPARTURE Customer Departure 23 -2528

567 ARRIVAL Customer Arrival 47 -2054

572 ARRIVAL Customer Arrival 48 -1244

580 DEPARTURE Customer Departure 24 2636

584 ARRIVAL Customer Arrival 49 328

590 DEPARTURE Customer Departure 25 -805

596 ARRIVAL Customer Arrival 50 2593

600 DEPARTURE Customer Departure 26 -311

600 DEPARTURE Customer Departure 27 -1538

600 DEPARTURE Customer Departure 28 -1596

600 DEPARTURE Customer Departure 29 2224

600 DEPARTURE Customer Departure 30 151

600 DEPARTURE Customer Departure 31 -685

600 DEPARTURE Customer Departure 32 -1633

600 DEPARTURE Customer Departure 33 -2318

600 DEPARTURE Customer Departure 34 2644

600 DEPARTURE Customer Departure 35 -451

600 DEPARTURE Customer Departure 36 -799

600 DEPARTURE Customer Departure 37 2779

600 DEPARTURE Customer Departure 38 580

600 DEPARTURE Customer Departure 39 1975

600 DEPARTURE Customer Departure 40 -2914

600 DEPARTURE Customer Departure 41 -723

600 DEPARTURE Customer Departure 42 -415

600 DEPARTURE Customer Departure 43 1446

600 DEPARTURE Customer Departure 44 -2722

600 DEPARTURE Customer Departure 45 1609

600 DEPARTURE Customer Departure 46 -2417

600 DEPARTURE Customer Departure 47 -2054

600 DEPARTURE Customer Departure 48 -1244

600 DEPARTURE Customer Departure 49 328

600 DEPARTURE Customer Departure 50 2593

结果分析：由于最大交易额和交易时间在正常范围，客户较高频到达，营业时间内无法实现部分客户的业务处理，客户平均逗留时间较长。

**情景三：长时间内较低频到达**

初始资金总额：10000元

营业时间：600分钟

相邻到达事件的间隔时间下界：30分钟

相邻到达事件的间隔时间上界：60分钟

交易时间下界：5分钟

交易时间上界：30分钟

最大交易金额：3000元

平均逗留时间：15.000000

Event Table:

Time Type Data Id Deal

211 ARRIVAL Customer Arrival 1 972

220 DEPARTURE Customer Departure 1 972

263 ARRIVAL Customer Arrival 2 -1923

290 DEPARTURE Customer Departure 2 -1923

300 ARRIVAL Customer Arrival 3 651

320 DEPARTURE Customer Departure 3 651

354 ARRIVAL Customer Arrival 4 -1830

364 DEPARTURE Customer Departure 4 -1830

385 ARRIVAL Customer Arrival 5 -2276

413 DEPARTURE Customer Departure 5 -2276

445 ARRIVAL Customer Arrival 6 -1880

464 DEPARTURE Customer Departure 6 -1880

483 ARRIVAL Customer Arrival 7 795

489 DEPARTURE Customer Departure 7 795

520 ARRIVAL Customer Arrival 8 -1956

548 DEPARTURE Customer Departure 8 -1956

550 ARRIVAL Customer Arrival 9 134

561 DEPARTURE Customer Departure 9 134

583 ARRIVAL Customer Arrival 10 -999

600 DEPARTURE Customer Departure 10 -999

结果分析：由于最大交易额和交易时间在正常范围，客户较低频地到达，所以几乎满足所有的客户都能完成业务处理并离开银行，客户平均逗留时间合理。

**情景四：交易金额较大**

初始资金总额：10000元

营业时间：600分钟

相邻到达事件的间隔时间下界：5分钟

相邻到达事件的间隔时间上界：30分钟

交易时间下界：5分钟

交易时间上界：30分钟

最大交易金额：7000元

平均逗留时间：26.000000

Event Table:

Time Type Data Id Deal

143 ARRIVAL Customer Arrival 1 -2214

152 ARRIVAL Customer Arrival 2 -3675

158 DEPARTURE Customer Departure 1 -2214

173 ARRIVAL Customer Arrival 3 -480

179 DEPARTURE Customer Departure 2 -3675

199 ARRIVAL Customer Arrival 4 -4503

202 DEPARTURE Customer Departure 3 -480

219 ARRIVAL Customer Arrival 5 2116

225 ARRIVAL Customer Arrival 6 -5663

231 DEPARTURE Customer Departure 5 2116

240 DEPARTURE Customer Departure 4 -4503

245 ARRIVAL Customer Arrival 7 2463

270 DEPARTURE Customer Departure 7 2463

272 ARRIVAL Customer Arrival 8 2292

286 DEPARTURE Customer Departure 8 2292

293 ARRIVAL Customer Arrival 9 5941

300 DEPARTURE Customer Departure 6 -5663

320 ARRIVAL Customer Arrival 10 -2886

325 DEPARTURE Customer Departure 9 5941

327 ARRIVAL Customer Arrival 11 -3162

332 ARRIVAL Customer Arrival 12 3070

354 DEPARTURE Customer Departure 10 -2886

358 ARRIVAL Customer Arrival 13 -5856

364 ARRIVAL Customer Arrival 14 4715

364 DEPARTURE Customer Departure 11 -3162

388 DEPARTURE Customer Departure 12 3070

393 ARRIVAL Customer Arrival 15 6127

412 DEPARTURE Customer Departure 14 4715

418 ARRIVAL Customer Arrival 16 5898

427 DEPARTURE Customer Departure 13 -5856

432 ARRIVAL Customer Arrival 17 -4900

435 DEPARTURE Customer Departure 15 6127

450 DEPARTURE Customer Departure 16 5898

451 ARRIVAL Customer Arrival 18 1110

455 DEPARTURE Customer Departure 17 -4900

467 DEPARTURE Customer Departure 18 1110

481 ARRIVAL Customer Arrival 19 5179

492 ARRIVAL Customer Arrival 20 6644

493 DEPARTURE Customer Departure 19 5179

503 ARRIVAL Customer Arrival 21 1869

504 DEPARTURE Customer Departure 20 6644

512 DEPARTURE Customer Departure 21 1869

516 ARRIVAL Customer Arrival 22 5674

525 DEPARTURE Customer Departure 22 5674

545 ARRIVAL Customer Arrival 23 -5078

558 DEPARTURE Customer Departure 23 -5078

574 ARRIVAL Customer Arrival 24 306

592 DEPARTURE Customer Departure 24 306

599 ARRIVAL Customer Arrival 25 -5580

600 DEPARTURE Customer Departure 25 -5580

结果分析：由于到达时间间隔和交易时间在正常范围，最大交易金额较高，所以有时候暂时不能满足部分取款/借款客户的业务要求，需要让他们先到队列2中排队，故客户平均逗留时间稍长。

**情景五：交易金额较小**

初始资金总额：10000元

营业时间：600分钟

相邻到达事件的间隔时间下界：5分钟

相邻到达事件的间隔时间上界：30分钟

交易时间下界：5分钟

交易时间上界：30分钟

最大交易金额：1000元

平均逗留时间：35.000000

Event Table:

Time Type Data Id Deal

176 ARRIVAL Customer Arrival 1 431

186 DEPARTURE Customer Departure 1 431

204 ARRIVAL Customer Arrival 2 799

209 ARRIVAL Customer Arrival 3 455

223 DEPARTURE Customer Departure 2 799

229 ARRIVAL Customer Arrival 4 -526

251 DEPARTURE Customer Departure 3 455

252 ARRIVAL Customer Arrival 5 -378

266 DEPARTURE Customer Departure 4 -526

272 ARRIVAL Customer Arrival 6 -913

273 DEPARTURE Customer Departure 5 -378

297 ARRIVAL Customer Arrival 7 -798

300 DEPARTURE Customer Departure 6 -913

311 DEPARTURE Customer Departure 7 -798

320 ARRIVAL Customer Arrival 8 592

332 DEPARTURE Customer Departure 8 592

338 ARRIVAL Customer Arrival 9 -752

350 ARRIVAL Customer Arrival 10 -552

358 ARRIVAL Customer Arrival 11 -15

367 DEPARTURE Customer Departure 9 -752

379 ARRIVAL Customer Arrival 12 -146

397 DEPARTURE Customer Departure 10 -552

398 ARRIVAL Customer Arrival 13 928

408 ARRIVAL Customer Arrival 14 620

423 ARRIVAL Customer Arrival 15 -310

425 DEPARTURE Customer Departure 11 -15

443 DEPARTURE Customer Departure 12 -146

451 ARRIVAL Customer Arrival 16 527

451 DEPARTURE Customer Departure 13 928

467 ARRIVAL Customer Arrival 17 448

477 DEPARTURE Customer Departure 14 620

483 ARRIVAL Customer Arrival 18 603

486 DEPARTURE Customer Departure 15 -310

495 ARRIVAL Customer Arrival 19 -1000

496 DEPARTURE Customer Departure 16 527

511 DEPARTURE Customer Departure 17 448

523 ARRIVAL Customer Arrival 20 -909

529 DEPARTURE Customer Departure 18 603

536 ARRIVAL Customer Arrival 21 468

539 DEPARTURE Customer Departure 19 -1000

554 DEPARTURE Customer Departure 20 -909

563 ARRIVAL Customer Arrival 22 -287

575 DEPARTURE Customer Departure 21 468

591 ARRIVAL Customer Arrival 23 -853

592 DEPARTURE Customer Departure 22 -287

600 DEPARTURE Customer Departure 23 -853

结果分析：由于各项参数正常，最大交易额偏小，故平均逗留时间正常。

**情景六：（极端情况）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**

初始资金总额：10000元

营业时间：600分钟

相邻到达事件的间隔时间下界：5分钟

相邻到达事件的间隔时间上界：15分钟

交易时间下界：200分钟

交易时间上界：300分钟

最大交易金额：3000元

平均逗留时间：168.000000

Event Table:

Time Type Data Id Deal

226 ARRIVAL Customer Arrival 1 -245

241 ARRIVAL Customer Arrival 2 -666

252 ARRIVAL Customer Arrival 3 -78

267 ARRIVAL Customer Arrival 4 380

282 ARRIVAL Customer Arrival 5 -268

288 ARRIVAL Customer Arrival 6 487

297 ARRIVAL Customer Arrival 7 -152

305 ARRIVAL Customer Arrival 8 -782

316 ARRIVAL Customer Arrival 9 -265

326 ARRIVAL Customer Arrival 10 -237

338 ARRIVAL Customer Arrival 11 314

353 ARRIVAL Customer Arrival 12 809

360 ARRIVAL Customer Arrival 13 -197

367 ARRIVAL Customer Arrival 14 703

378 ARRIVAL Customer Arrival 15 200

392 ARRIVAL Customer Arrival 16 99

397 ARRIVAL Customer Arrival 17 -223

410 ARRIVAL Customer Arrival 18 -367

423 ARRIVAL Customer Arrival 19 818

430 ARRIVAL Customer Arrival 20 982

441 ARRIVAL Customer Arrival 21 55

451 DEPARTURE Customer Departure 1 -245

452 ARRIVAL Customer Arrival 22 -529

462 ARRIVAL Customer Arrival 23 -677

469 ARRIVAL Customer Arrival 24 -621

479 ARRIVAL Customer Arrival 25 -651

485 ARRIVAL Customer Arrival 26 159

493 ARRIVAL Customer Arrival 27 746

498 ARRIVAL Customer Arrival 28 -689

509 ARRIVAL Customer Arrival 29 1000

517 ARRIVAL Customer Arrival 30 -562

525 ARRIVAL Customer Arrival 31 -161

540 ARRIVAL Customer Arrival 32 -395

545 ARRIVAL Customer Arrival 33 250

552 ARRIVAL Customer Arrival 34 537

559 ARRIVAL Customer Arrival 35 -360

568 ARRIVAL Customer Arrival 36 -733

574 ARRIVAL Customer Arrival 37 287

587 ARRIVAL Customer Arrival 38 968

598 ARRIVAL Customer Arrival 39 -316

600 DEPARTURE Customer Departure 2 -666

600 DEPARTURE Customer Departure 3 -78

600 DEPARTURE Customer Departure 4 380

600 DEPARTURE Customer Departure 5 -268

600 DEPARTURE Customer Departure 6 487

600 DEPARTURE Customer Departure 7 -152

600 DEPARTURE Customer Departure 8 -782

600 DEPARTURE Customer Departure 9 -265

600 DEPARTURE Customer Departure 10 -237

600 DEPARTURE Customer Departure 11 314

600 DEPARTURE Customer Departure 12 809

600 DEPARTURE Customer Departure 13 -197

600 DEPARTURE Customer Departure 14 703

600 DEPARTURE Customer Departure 15 200

600 DEPARTURE Customer Departure 16 99

600 DEPARTURE Customer Departure 17 -223

600 DEPARTURE Customer Departure 18 -367

600 DEPARTURE Customer Departure 19 818

600 DEPARTURE Customer Departure 20 982

600 DEPARTURE Customer Departure 21 55

600 DEPARTURE Customer Departure 22 -529

600 DEPARTURE Customer Departure 23 -677

600 DEPARTURE Customer Departure 24 -621

600 DEPARTURE Customer Departure 25 -651

600 DEPARTURE Customer Departure 26 159

600 DEPARTURE Customer Departure 27 746

600 DEPARTURE Customer Departure 28 -689

600 DEPARTURE Customer Departure 29 1000

600 DEPARTURE Customer Departure 30 -562

600 DEPARTURE Customer Departure 31 -161

结果分析：由于其他参数正常，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，故只能处理极少客户的业务要求，银行很忙，大量客户到营业时间结束还没有机会处理业务，客户平均逗留时间很长。

**情景七：（极端情况）****两个到达事件的间隔时间很长，而客户的交易时间很短**

初始资金总额：10000元

营业时间：600分钟

相邻到达事件的间隔时间下界：200分钟

相邻到达事件的间隔时间上界：300分钟

交易时间下界：5分钟

交易时间上界：15分钟

最大交易金额：3000元

平均逗留时间：6.000000

Event Table:

Time Type Data Id Deal

100 ARRIVAL Customer Arrival 1 -408

107 DEPARTURE Customer Departure 1 -408

327 ARRIVAL Customer Arrival 2 -368

339 DEPARTURE Customer Departure 2 -368

581 ARRIVAL Customer Arrival 3 273

587 DEPARTURE Customer Departure 3 273

结果分析：由于其他参数正常，两个到达事件的间隔时间很长，而客户的交易时间很短，故只有极少的客户到达银行，且能够迅速处理完业务，银行很闲，客户平均逗留时间很短。