廣東工業大學

课 程 设 计

课程名称 数据结构

题目名称 银行业务模拟（难度系数：1.3）

学生学院 计算机学院

专业班级 网络工程 1602

学　　号 3116004979

学生姓名 詹泽霖

指导教师 李杨

2017 年 1 月 14 日

1. 需求分析

1.1 输入的形式

输入的元素为整型类型；

输入的银行初始存款必须大于 0；

输入的银行营业时间必须大于 0 且必须小于 1440（一天）；

输入的最大到达时间间隔必须大于 0 且必须小于银行营业时间；

输入的最小到达时间间隔必须大于 0 且必须小于最大到达时间间隔；

输入的最大处理时间必须大于 0 且必须小于银行营业时间；

输入的最小处理时间必须大于 0 且必须小于最大处理时间；

输入的交易额的最大上线必须大于 0 且必须小于银行初始存款且必须小于 50000；

1.2 输出的形式

输出的形式为以列表的形式输出事件处理序列；

并在列表输出完后输出需要存款的客户人数，需要取款的客户人数，成功办理存款的客户人数，成功办理取款的客户人数，存款成功办理率，取款成功办理率，客户逗留平均时间，银行当前余额等信息。

1.3 程序功能

实现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。

1.4 测试

测试数据由程序用户手动输入，此处对于正确输入和错误输入给出样例。

(1)错误的输入

```
请输入银行的营业时间：1450
输入错误！一天的营业时间不能超过1440分钟（24个小时）！请再次输入！
请输入银行的营业时间：
```

```
请输入银行的营业时间：500
请输入最大到达时间间隔：501
输入错误！最大到达时间间隔必须小于营业时间！请再次输入！
请输入最大到达时间间隔：
```

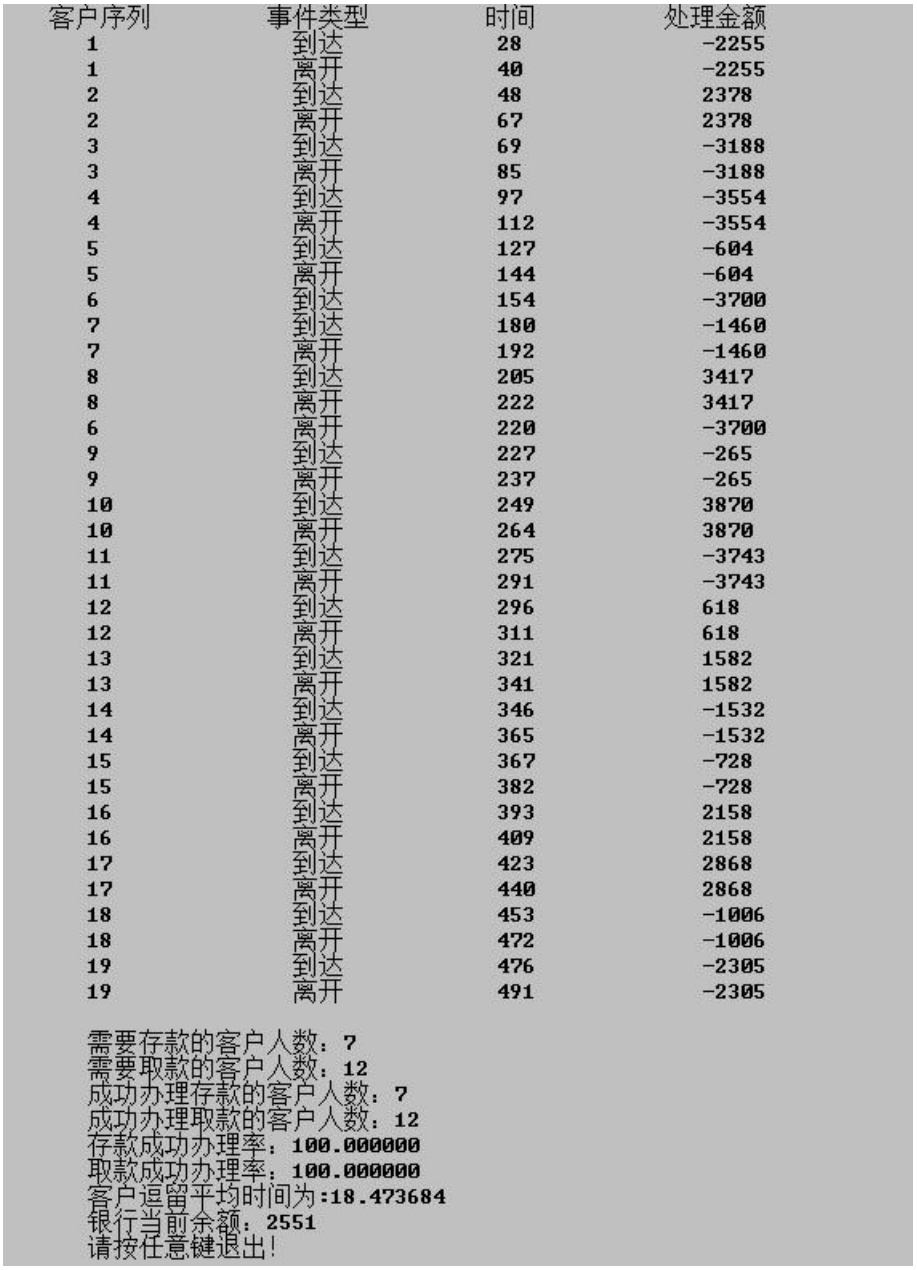
```
请输入最大到达时间间隔：20
请输入最小到达时间间隔：21
输入错误！最小到达时间间隔必须介于零和最大到达时间之间！请再次输入！
请输入最小到达时间间隔：
```

```
请输入交易额的最大上限：100000
输入错误！超出本银行的服务范围！最大交易额应低于银行开始营业时的资金总额且小于50000！请再次输入！
请输入交易额的最大上限：
```

(2)正确的输入

```
请输入选择的操作对应编号：1
请输入银行的初始存款：100000
请输入银行的营业时间：500
请输入最大到达时间间隔：10
请输入最小到达时间间隔：5
请输入最大的处理时间：10
请输入最小的处理时间：5
请输入交易额的最大上限：5000
```

(3)对应的输出结果



2. 概要设计

2.1 数据类型

本设计中用到的数据结构 ADT 定义如下：

ADT Queue{

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

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

基本操作：

void InitQueue(Queue &Q);

操作结果：构造空队列 Q

CustNode *Queuefront(Queue &Q);

初始条件：队列 Q 存在

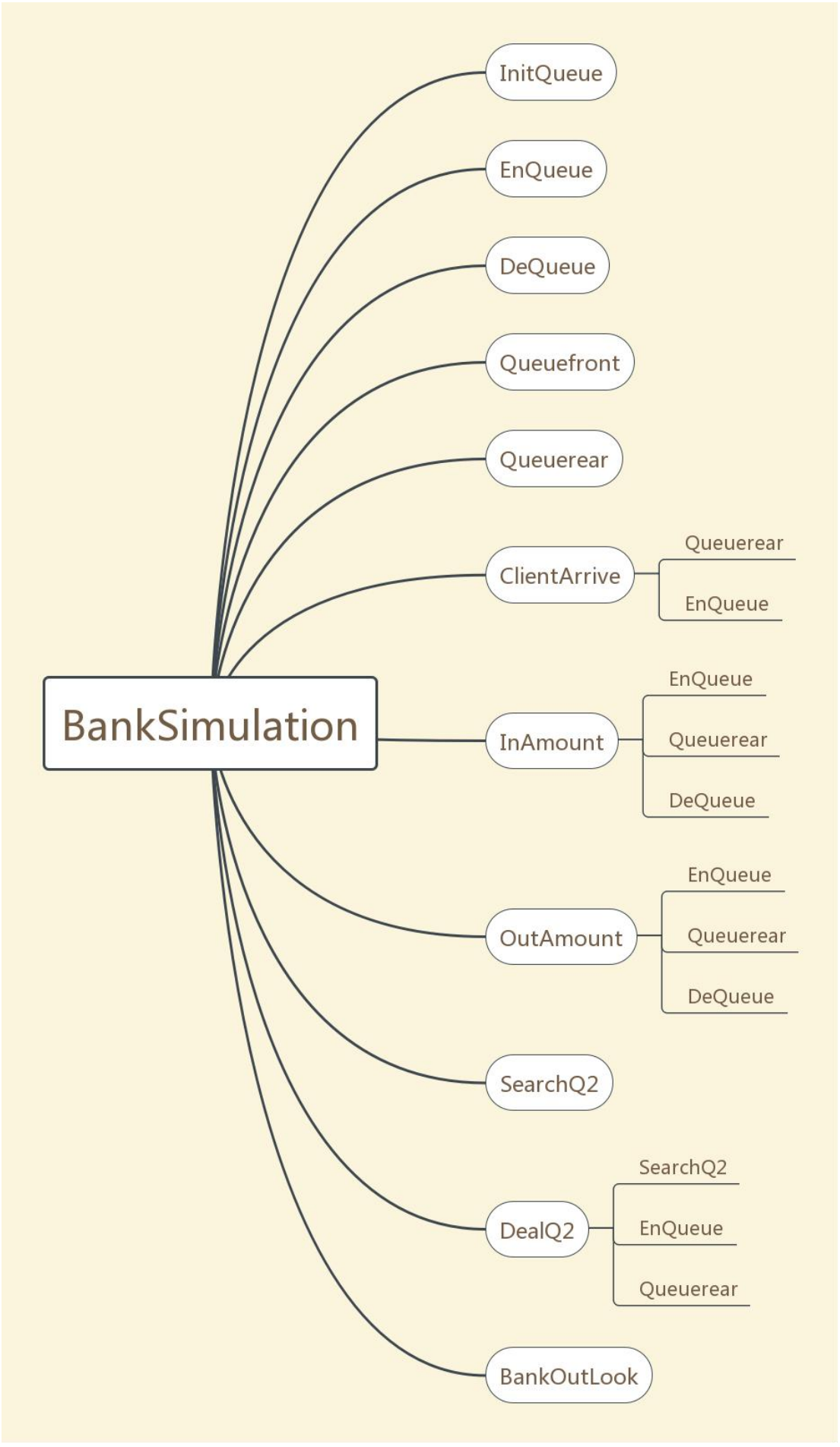
操作结果：返回队首元素

CustNode *Queuerear(Queue &Q);
初始条件：队列 Q 存在
操作结果：返回队尾元素
void EnQueue(Queue &Q,int e);
初始条件：队列 Q 存在
操作结果：插入元素 e 为 Q 的新的队尾元素。
void DeQueue(Queue &Q);
初始条件：队列 Q 存在
操作结果：删除 Q 的队头元素。
}ADT Queue

2.2 主程序的流程

主程序先是让外部进行测试数据输入，待测试数据输入完后，执行银行业务模拟系统，产生需要取款的客户人数，成功办理存款的客户人数，成功办理取款的客户人数，存款成功办理率，取款成功办理率，客户逗留平均时间，银行当前余额等信息。

2.3 程序模块说明


BankSimulation
InitQueue
EnQueue
DeQueue
Queuefront
Queuerear
ClientArrive
Queuerear
EnQueue
InAmount
EnQueue
Queuerear
DeQueue
OutAmount
EnQueue
Queuerear
DeQueue
SearchQ2
DealQ2
SearchQ2
EnQueue
Queuerear
BankOutLook

3. 详细设计

3.1 头文件声明

为了增强代码刻度行，使用头文件来记录各类结构体的声明以及常用变量的定义

```
#ifndef _Bank_H_
#define _Bank_H_
#include <string>
using namespace std;

/*客户结点类型*/
struct CustNode{
    int num;                    //客户号
    string Type;                //到达或离开
    int BeginTime;              //到达时间
    int EndTime;                //离开时间
    int Amount;                 //正数为存款，负数为取款
    CustNode *next;             //指针域
};

Struct Client{
    Int arrivertime;            //到达时间
    Int durtime;                //逗留时间
    Int amount;                 //办理业务金额
    Client *next;               //指针域
};
Client pool[MaxNumber];

/*等待队列类型*/
struct Queue{
    CustNode *front;            //队列头指针
    CustNode *rear;             //队列尾指针
}Queue;

/*常用变量定义*/
int BankAmount;                 //初始时银行现存资金总额
int CloseTime;                  //营业结束时间
int ClientArriveMaxTime;        //两个到达事件之间的间隔上限
int ClientArriveMinTime;        //两个到达事件之间的间隔下限
int DealMaxTime;                //客户之间交易的时间上限
int DealMinTime;                //客户之间交易的时间下限
int MaxAmount;                  //交易额上限
int NeedIn=0;                   //需要存款的人数
int NeedOut=0;                  //需要取款的人数
int SuccessIn=0;                //成功存款的人数
int SuccessOut=0;               //成功取款的人数
```

```
    int CurrentTime=0;                  //当前时间
    int BankAmountTime=0;               //客户逗留总时间
    int counter=0;                      //客户总数
    int number=1;                       //初始客户序列号
    bool state=1;                       //用于判断是否有窗口在处理
    int DealTime=0;                     //交易时间
    int MaxTime=0;                      //最大到达时间
    Queue Event;                        //事件队列
    Queue Q1;                           //队列一
    Queue Q2;                           //队列二
    #endif
```

3.2 选做算法

3.2.1 动态分配函数

```
    /*出栈，将栈顶元素的下标返回*/
```

伪码表示：

```
Begin
    Client *p<-栈顶指针
    e 的 arrivetime<-栈顶元素的 arrivertime
    e 的 durtime<-栈顶元素的 durtime
    e 的 amount<-栈顶元素的 amount
    栈顶指针指向下一元素
End
```

代码表示：

```
void myMalloc(Stack &S,Client &e){
    Client *p=S.top;
    e.arrivetime=(*S.top).arrivetime;
    e.durtime=(*S.top).durtime;
    e.amount=(*S.top).amount;
    p->next=p->next->next;
    S.top++;
    p=p->next；
}
```

3.2.2 归还函数

```
    /*把该分量入栈*/
```

伪码表示：

```
Begin
    Client *p<-栈顶指针
    栈底元素的 arrivertimee<-e 的 arrivetime
    栈底元素的 durtime<-e 的 durtime
    栈底元素的 amount<-e 的 amount
    栈底指针指向下一元素
End
```

代码表示：

```
void myFree(Stack &S,Client e){
```

```
    Client *p=S.rear;
    (*S.rear).arrivertime=e.arrivetime;
    (*S.rear).durtime=e.durtime;
    (*S.rear).amount=e.amount;
    p->next=p->next->next;
    S.rear++;
    p=p->next;
}
```

3.3 函数算法

3.3.1 创建队列

```
    /*初始化操作，建立一个空队列*/
```

伪码表示：

```
Begin
    分配存储空间给头结点和尾结点
    IF 头结点内存分配失败
        EXIT
    头结点的指针域<-空
End
```

代码表示：

```
void InitQueue(Queue &Q){
    Q.front=Q.rear=(CustNode*)malloc(sizeof(CustNode));
    if(!(Q.front))
        exit(1);
    Q.front->next=0;
 }
```

3.3.2 入队列

```
    /*插入元素 e 为队列 Q 的新的队尾元素*/
```

伪码表示：

```
Begin
    分配存储空间给结点 p
    结点 p 的金额置<-e
    结点 p 的指针域<-空
    IF 队列<-空
        Begin
            头指针<-p
            尾指针<-p
        End
    ELSE
        Begin
            头指针 next 域<-p
            尾指针<-尾指针 next 域
        End
End
```

代码表示：

```
void EnQueue(Queue &Q,int e){
    CustNode* p=new CustNode;
    p->Amount=e;
    p->next=NULL;
    if(Q.front==NULL){
    //队列为空，初始化
        Q.front=p;
        Q.rear=p;
    }
    else{
    //队列不为空，插入结点 p
        Q.rear->next=p;
        Q.rear=Q.rear->next;
    }
}
```

3.3.3 出队列

```
    /*使 p 中的第一个元素出队列*/
```

伪码表示：

```
Begin
    p<-头指针
    IF p 的 next 域<-空
        Begin
            头指针<-空
            尾指针<-空
        End
    ELSE
        头指针<-头指针的 next 域
    DELETE p
End
```

代码表示：

```
void DeQueue(Queue &Q){
    CustNode *p;
    p=Q.front;
    if(Q.front->next==NULL)       //队列只有一个元素
        Q.front=Q.rear=NULL;
    else                                  //调整队列头指针
        Q.front=Q.front->next;
    delete p;
}
```

3.3.4 取队首

```
    /*返回队首元素*/
```

伪码表示：

```
Begin
    Return  队首元素
```

End
代码表示：
```
CustNode *Queuefront(Queue &Q){
    return Q.front;
}
```

3.3.5 取队尾

/*返回队尾元素*/
伪码表示：
```
Begin
    Return  队尾元素
End
```
代码表示：
```
CustNode *Queuerear(Queue &Q){
    return Q.rear;
}
```
3.3.6 处理客户到达事件

/*随机产生顾客，进入队列一产生到达事件 进入事件队列*/
伪码表示：
```
Begin
    调用 EnQueue(Q1,随机数）
    Q1 尾结点的 BeginTime<-CurrentTime
    Q1 尾结点的 num<-number
    EnQueue(Event,尾结点的金额)
    Event 尾结点的 BeginTime<-CurrentTime
    Event 尾结点的 Type<-到达
    Event 尾结点的 num<-number
    number<-number+1
End
```
代码表示：
```
void ClientArrive(){
    EnQueue(Q1,(rand()%(2*MaxAmount)-MaxAmount));      //随机产生顾客加入第一队列
    Queuerear(Q1)->BeginTime=CurrentTime;              //当前时间为客户的到达时间
    Queuerear(Q1)->num=number;                         //客户号为客户序列号
    EnQueue(Event,(Queuerear(Q1)->Amount));            //将产生事件加入事件队列
    Queuerear(Event)->BeginTime=CurrentTime;
    Queuerear(Event)->Type="到达";
    Queuerear(Event)->num=number;
    number++;
}
```
3.3.7 存款

/*对客户存款事件进行处理*/
伪码表示：
```
Begin
```

BankAmount<-BankAmount+Q1 头结点的 Amount
调用 EnQueue(Event,Q1 头结点的 Amount)
Event 尾结点的 Type<-离开
Event 尾结点的 num<-Q1 头结点的 num
Event 尾结点的 EndTime<-Q1 头结点的 BeginTime+随机处理时间
counter<-counter+1
BankAmountTime<-BankAmountTime+Event 尾结点的 EndTime-Q1 头结点的 BeginTime
调用 DeQueue(Q1)
DealTime<-Event 尾结点的 EndTime
state<-0
End

代码表示：

```
void InAmount(){
    BankAmount+=Queuefront(Q1)->Amount;              //更新资金总额
    EnQueue(Event,Queuefront(Q1)->Amount);           //加入事件队列
    Queuerear(Event)->Type="离开";
    Queuerear(Event)->num=Queuefront(Q1)->num;
    //离开时间为到达时间加上随机产生的介于最大处理时间和最小处理时间的处理时间
Queuerear(Event)->EndTime=(Queuefront(Q1)->BeginTime+rand()%(DealMaxTime-DealMinTime
+1)+DealMinTime);
    counter++;                                       //更新客户总数
    BankAmountTime+=(Queuerear(Event)->EndTime-Queuefront(Q1)->BeginTime);
    //更新逗留时间
    DeQueue(Q1);                                     //删除第一队列第一个业务
    DealTime=Queuerear(Event)->EndTime;              //交易时间为客户的离开时间
    state=0;                                         //窗口没有交易需要处理
}
```

3.3.8 取款或借款

/*对客户取款或借款事件进行处理*/

伪码表示：

```
Begin
    IF -Q1 头结点的 Amount>BankAmount
        Begin
            调用 EnQueue(Q2,Q1 头结点的 Amount)
            Q2 尾结点的 BeginTime<-Q1 头结点的 BeginTime
            Q2 尾结点的 num<-Q1 头结点的 num
            调用 DeQueue(Q1)
        End
    ELSE
        Begin
            BankAmount<-BankAmount+Q1 尾结点的 Amount
            调用 EnQueue(Event,Q1 头结点的 Amount)
            Event 尾结点的 Type<-离开
            Event 尾结点的 num<-Q1 头结点的 num
```

Event 尾结点的 EndTime<-Q1 头结点的 BeginTime+随机处理时间
DealTime<-Event 尾结点的 EndTime
counter<-counter+1
BankAmountTime<-Event 尾结点的 EndTime-Q1 尾结点的 BeginTime
调用 DeQueue(Q1)
State<-0
End
End

代码表示：

```
void OutAmount(){
    if((-Q1.front->Amount)>BankAmount){
    //资金短缺 加入第二队列
        EnQueue(Q2,Queuefront(Q1)->Amount);
        Queuerear(Q2)->BeginTime=Queuefront(Q1)->BeginTime;
        Queuerear(Q2)->num=Queuefront(Q1)->num;
        DeQueue(Q1);
    }
    else{
        BankAmount+=Queuerear(Q1)->Amount;              //更新资金总额
        EnQueue(Event,Queuefront(Q1)->Amount);          //加入事件队列
        Queuerear(Event)->Type="离开";
        Queuerear(Event)->num=Queuefront(Q1)->num;
        //客户的离开时间为客户的到达时间加上随机产生的介于最大处理时间和最小处
理时间的处理时间
        Queuerear(Event)->EndTime=(Queuefront(Q1)->BeginTime
+rand()%(DealMaxTime-DealMinTime +1)+DealMinTime);
        Queuerear(Event)->BeginTime=0;
        DealTime=Queuerear(Event)->EndTime;             //交易时间为客户的离开时间
        counter++;                                      //更新客户总数
        BankAmountTime+=(Queuerear(Event)->EndTime-Queuerear(Q1)->BeginTime);
//更新逗留时间
        DeQueue(Q1);                                    //删除第一队列第一个业务
        state=0;                                        //窗口没有交易需要处理
    }
}
```

3.3.9 检查队列 Q2

/*顺序检查队列 Q2 中是否有可以处理的事件元素*/

伪码表示：

```
Begin
    sign<-Q 头结点
    CustNode *temp
    While Q 头结点为空
        Begin
            IF -Q 头结点的 Amount<m
```

```
                Begin
                    IF Q 为空
                        Begin
                            temp<-Q 头结点
                            Q 头结点<-空
                            Q 尾结点<-空
                            Return temp
                        End
                    ELSE
                        Begin
                            temp<-Q 头结点
                            Q 头结点<-Q 头结点的 next 域
                            Return temp
                        End
                End
            ELSE
                Begin
                    IF Q 不为空
                        Begin
                            Q 尾结点的 next 域<-Q 头结点
                            Q 尾结点<-Q 尾结点的 next 域
                            Q 头结点<-Q 头结点的 next 域
                            Q 尾结点的 next 域<-空
                        End
                End
            IF Q 头结点等于 sign
                Return  空
        End
        Return  空
End
```

代码表示：

```
CustNode *SearchQ2(Queue &Q,int m){
    CustNode *sign=Q.front;                    //标记头节点
    CustNode *temp;
    while(Q.front!=NULL){
        if((-(Q.front->Amount))<m){            //队首元素可以处理
            if(Q.front==Q.rear){
                temp=Q.front;
                Q.front=Q.rear=NULL;
                return temp;
            }
            else{                              //队首元素出列
                temp=Q.front;
                Q.front=Q.front->next;         //  首节点后移一位,返回原首节点
```

```
                return temp;
            }
        }
        else{                                   //队首元首不能被处理
            if(Q.front==Q.rear){
            }
            else{                               //首节点移到队列尾部

                Q.rear->next=Q.front;
                Q.rear=Q.rear->next;
                Q.front=Q.front->next;
                Q.rear->next=NULL;
            }
        }
        if(Q.front==sign)                       //队列循环一周时停止
            return NULL;
    }
    return NULL;
}
```

3.3.10 处理队列 Q2

/*对于队列 Q2 中可以处理的事件元素进行处理*/

伪码表示：

```
Begin
    CustNode* temped
    int RandomTemp
    While temped<-调用 SearchQ2(Q2,BankAmount)并且 temped 不为空
        Begin
            BankAmount<-temped 的 Amount+BankAmount
            EnQueue(Event,temped 的 Amount
            Event 尾结点的 Type<-离开
            Event 尾结点的 num<-temped 的 num
            RandomTemp<-随机处理时间
            Event 尾结点的 EndTime<-CurrentTime+RandomTemp
            DealTime<-DealTime+RandomTemp
            counter<-counter+1
            BankAmountTime<-Event 尾结点的 EndTime-temped 的 BeginTime+BankAmount
            删除  temped
            temped<-空
        End
        state<-0
End
```

代码表示：

```
void DealQ2(){
```

```
    CustNode* temped;
    int randomTemp;
    while((temped=SearchQ2(Q2,BankAmount))&&temped!=NULL){      //查找可处理取款

        BankAmount+=temped->Amount;                             //更新资金总额
        EnQueue(Event,temped->Amount);                          //加入事件队列
        Queuerear(Event)->Type="离开";
        Queuerear(Event)->num=temped->num;
        RandomTemp=rand()%(DealMaxTime-DealMinTime +1)+DealMinTime;
        //处理时间为随机产生的介于最大处理时间和最小处理时间之间的处理时间
        Queuerear(Event)->EndTime=CurrentTime+randomTemp ;
        //客户离开时间为当前时间加上处理时间
        DealTime+=randomTemp;                                   //更新交易时间
        counter++;                                              //更新客户总数
        BankAmountTime+=(Queuerear(Event)->EndTime-temped->BeginTime);
        //更新逗留时间
        delete temped;                                          //删除节点
        temped = NULL;
    }
    state = 0;
}
```

3.3.11 银行业务模拟系统界面

/*银行业务模拟程序的界面*/

伪码表示：

```
Begin
    输出    ================================================================================
    换行
    输出    ================================================================================
    换行
    输出                              Simulation of The Bank business
    换行
    输出                          --------------------------------------------------------------------------
    换行
    输出                                        ---------------------------------
    换行
    输出                                          Number: 3116004979
    输出                                          CLASS ： 16 网络二班
    输出                                           NAME   ：  詹泽霖
    输出    ================================================================================
    换行
    输出   ********************************************************************************
    换行
    输出    ********************************************************************************
    换行
```

输出 ********** **********

换行

输出 ********** 0.退出 1.进入模拟系统 **********

换行

输出 ********** **********

换行

输出 **********

换行

输出 **********

换行

输出 ********** 请选择服务 **********

换行

输出 ==========

换行

End

代码表示：

```
void BankOutLook(){
    printf("    ================================================================================        \n");
    printf("    ================================================================================        \n");
    printf("                                    Simulation of The Bank business                             \n");
    printf("                    --------------------------------------------------------------------        \n\n");
    printf("                                      ---------------------------------                         \n");
    printf("                                        Number: 3116004979                                      \n");
    printf("                                      CLASS ： 16 网络二班                                      \n");
    printf("                                      NAME  ：  詹泽霖                                          \n");
    printf("                                      ---------------------------------                         \n");
    printf("    ================================================================================        \n");
    printf("    ********************************************************************                    \n");
    printf("    ********************************************************************                    \n");
    printf("    ********** **********                                                                    \n");
    printf("    ********** 0.退出  1.进入模拟系统  **********                                            \n");
    printf("    ********** **********                                                                    \n");
    printf("    ********************************************************************                    \n");
    printf("    ********************************************************************                    \n");
    printf("    ********** 请选择服务 **********                                                        \n");
    printf(" ================================================================================          \n");
    printf("    请输入选择的操作对应编号：");
}
```

3.4 主程序

通过对以上定义过的函数接口的调用，利用主函数来构建一个银行模拟系统，以列表的形式输出事件处理序列；

并在列表输出完后输出需要存款的客户人数，需要取款的客户人数，成功办理存款的客户人数，成功办理取款的客户人数，存款成功办理率，取款成功办理率，客户逗留平均时间，银行当前余额等信息。

3.4.1 伪码表示

```
Begin
    调整界面为黑字白背景
    调用 BankOutLook()
    输入操作编号
    While 操作编号等于 1
        Begin
            初始化随机函数
            输出 请输入银行的初始存款
            输入 银行初始存款
            IF 银行初始存款<0
            Begin
                输出 输入错误，重新输入
                输入 银行初始存款
                IF 银行初始存款<0
                Begin
                    输出 输入错误，重新输入
                    输入 银行初始存款
                    IF 银行初始存款<0
                        输出 三次输入错误，退出
                End
            End

            输出 请输入银行的营业时间
            输入 银行营业时间
            IF 银行营业时间<0 或者>1440
            Begin
                输出 输入错误，请重新输入
                输入 银行营业时间
                IF 银行营业时间<0 或者>1440
                Begin
                    输出 输入错误，请重新输入
                    输入 银行营业时间
                    IF 银行营业时间<0 或者>1440
                        输出 三次输入错误，退出程序
                End
            End

            输出 请输入最大到达时间间隔
            输入 最大到达时间间隔
            IF 最大到达时间间隔>银行营业时间
            Begin
                输出 输入错误，请重新输入
                输入 最大到达时间间隔
```

```
    IF 最大到达时间间隔>银行营业时间
    Begin
        输出 输入错误，请重新输入
        输入 最大到达时间间隔
        IF 最大到达时间间隔>银行营业时间
            输出 三次输入错误，退出程序
    End
End

输出 请输入最小到达时间间隔
输入 最小到达时间间隔
IF 最小到达时间间隔>最大到达时间间隔
Begin
    输出 输入错误，请重新输入
    输入 最小到达时间间隔
    IF 最小到达时间间隔>最大到达时间间隔
    Begin
        输出 输入错误，请重新输入
        输入 最小到达时间间隔
        IF 最小到达时间间隔>最大到达时间间隔
            输出 三次输入错误，退出程序
    End
End

输出 请输入最大交易时间
输入 最大交易时间
IF 最大交易时间>银行营业时间
Begin
    输出 输入错误，请重新输入
    输入 最大交易时间
    IF 最大交易时间>银行营业时间
    Begin
        输出 输入错误，请重新输入
        输入 最大交易时间
        IF 最大交易时间>银行营业时间
            输出 三次输入错误，退出程序
    End
End

输出 请输入最小交易时间
输入 最小交易时间
IF 最小交易时间>最大交易时间
Begin
    输出 输入错误，请重新输入
```

```
    输入 最小交易时间
    IF 最小交易时间>最大交易时间
    Begin
        输出 输入错误，请重新输入
        输入 最小交易时间
        IF 最小交易时间>最大交易时间
            输出 三次输入错误，退出程序
    End
End

输出 请输入最小交易时间
输入 最小交易时间
IF 最小交易时间>最大交易时间
Begin
    输出 输入错误，请重新输入
    输入 最小交易时间
    IF 最小交易时间>最大交易时间
    Begin
        输出 输入错误，请重新输入
        输入 最小交易时间
        IF 最小交易时间>最大交易时间
            输出 三次输入错误，退出程序
    End
End

输出 请输入最大交易额
输入 最大交易额
IF 最大交易额>银行初始金额
Begin
    输出 输入错误，请重新输入
    输入 最小交易时间
    IF 最大交易额>银行初始金额
    Begin
        输出 输入错误，请重新输入
        输入 最大交易额
        IF 最大交易额>银行初始金额
            输出 三次输入错误，退出程序
    End
End

MaxTime<-MaxTime+随机到达间隔
While CurrentTime<CloseTime
Begin
    CurrentTime<-CurrentTime+1
```

```
        If DealTime<CurrentTime
            DealTime<-CurrentTime
        If DealTime 等于 CurrentTime
            State<-1
        IF CurrentTime 等于 MaxTime)
        Begin
            调用 ClientArrive()
            MaxTime<-MaxTime+随机到达间隔+ClientArriveMinTime
        End
        IF state 等于 1&&Q1 头指针不等于 NULL
        Begin
            IF Q1 头结点的 Amount 大于等于  0
            Begin
                调用 InAmount()
                调用 DealQ2()
                NeedIn<-NeedIn+1
            End
            ElSe
            Begin
                调用 InAmount()
                NeedIn<-NeedIn+1
            End
        End
    End

    输出      客户序列    换列      事件类型      换列      处理金额      换行
    While Event 头结点不为空
    Begin
        IF Event 头结点的 Type 等于  "离开"
        Begin
             输出  换列   Event 头结点的 num    换列      离开      Event 头结点
的 EndTime        换列      Event 头结点的 Amount      换行
              IF Event 头结点的 Amount 大于等于 0
                    t1<-t1+1
              Else
                    t3<-t3+1
        End
        Else
        Begin
            输出  换列    Event 头结点的 num    换列      到达      Event 头结点
  的 EndTime      换列      Event 头结点的 Amount      换行
            IF Event 头结点的 Amount 大于等于 0
                t2<-t2+1
            Else
```

```
                    t4<-t4+1
                    SuccessIn<-NeedIn-(t2-t1)
                    SuccessOut<-NeedOut-(t4-t3)
                    调用 DeQueue(Event)
                End
                    While Q1 的头结点等于 NULL
                    Begin
                        BankAmountTime<-BankAmountTime+(CloseTime-Q1 头结点的
                BeginTime)
                        counter<-counter+1;
                        调用 DeQueue(Q1);
                    End
                换行
                输出 需要存款的客户人数
                输出 需要取款的客户人数
                输出 成功办理存款的客户人数
                输出 成功办理取款的客户人数
                输出 存款成功办理率
                输出 取款成功办理率
                输出 客户进入系统平均时间
                输出 银行当前余额
    End
    If 操作数等于 0
        退出模拟系统
    Return 0
End
```

3.4.2 代码表示

```
int main(){
    system("color 70");
    BankOutLook();
    int n,t1=0,t2=0,t3=0,t4=0,m=0;
    scanf("%d",&n);
    while(n==1){
        srand(time(NULL));                                  //初始化随机函数
        printf("    请输入银行的初始存款：");
        scanf("%d",&BankAmount);
        if(BankAmount<0){
            printf("    输入错误！初始存款不能小于 0！请再次输入！\n");
            printf("    请输入银行的初始存款：");
            scanf("%d",&BankAmount);
            if(BankAmount<0){
                printf("    输入错误！初始存款不能小于 0！请最后一次输入！\n");
                printf("    请输入银行的初始存款：");
                scanf("%d",&BankAmount);
```

```
                if(BankAmount<0){
                    printf("    三次输入都错误！请按任意键退出！\n");
                    getch();
                    goto end;
                }
            }
        }
        printf("    请输入银行的营业时间：");
        scanf("%d",&CloseTime);
        if(CloseTime>=1440){
            printf("    输入错误！一天的营业时间不能超过 1440 分钟（24 个小时）！请
再次输入！\n");
            printf("    请输入银行的营业时间：");
            scanf("%d",&CloseTime);
            if(CloseTime>=1440){
                printf("    输入错误！一天的营业时间不能超过 1440 分钟（24 个小时）！
请最后一次输入！\n");
                printf("    请输入银行的营业时间：");
                scanf("%d",&CloseTime);
                if(CloseTime>=1440){
                    printf("    三次输入都错误！请按任意键退出！\n");
                    getch();
                    goto end;
                }
            }
        }
        printf("    请输入最大到达时间间隔：");
        scanf("%d",&ClientArriveMaxTime);
        if(ClientArriveMaxTime>CloseTime){
            printf("    输入错误！最大到达时间间隔必须小于营业时间！请再次输入！\n");
            printf("    请输入最大到达时间间隔：");
            scanf("%d",&ClientArriveMaxTime);
            if(ClientArriveMaxTime>CloseTime){
                printf("    输入错误！最大到达时间间隔必须小于营业时间！请最后一次
输入！\n");
                printf("    请输入最大到达时间间隔：");
                scanf("%d",&ClientArriveMaxTime);
                if(ClientArriveMaxTime>CloseTime){
                    printf("    三次输入都错误！请按任意键退出！\n");
                    getch();
                    goto end;
                }
            }
        }
```

```
        printf("     请输入最小到达时间间隔：");
        scanf("%d",&ClientArriveMinTime);
        if(ClientArriveMinTime<=0||ClientArriveMinTime>=ClientArriveMaxTime){
            printf("     输入错误！最小到达时间间隔必须介于零和最大到达时间之间！请再次输入！\n");
            printf("     请输入最小到达时间间隔：");
            scanf("%d",&ClientArriveMinTime);
            if(ClientArriveMinTime<=0||ClientArriveMinTime>=ClientArriveMaxTime){
                printf("     输入错误!最小到达时间间隔必须介于零和最大到达时间之间!请最后一次输入！\n");
                printf("     请输入最小到达时间间隔：");
                scanf("%d",&ClientArriveMinTime);
                if(ClientArriveMinTime<=0||ClientArriveMinTime>=ClientArriveMaxTime){
                    printf("     三次输入都错误！请按任意键退出！\n");
                    getch();
                    printf("     请按任意键退出!\n");
                    goto end;
                }
            }
        }
        printf("     请输入最大的处理时间：");
        scanf("%d",&DealMaxTime);
        if(DealMaxTime>CloseTime){
            printf("     输入错误！最大处理时间必须小于营业时间！请再次输入！\n");
            printf("     请输入最大的处理时间：");
            scanf("%d",&DealMaxTime);
            if(DealMaxTime>CloseTime){
                printf("     输入错误！最大处理时间必须小于营业时间！请最后一次输入！\n");
                printf("     请输入最大的处理时间：");
                scanf("%d",&DealMaxTime);
                if(DealMaxTime>CloseTime){
                    printf("     三次输入都错误！请按任意键退出！\n");
                    getch();
                    goto end;
                }
            }
        }
        printf("     请输入最小的处理时间：");
        scanf("%d",&DealMinTime);
        if(DealMinTime<=0||DealMinTime>=DealMaxTime){
            printf("     输入错误！最小处理时间必须介于零和最大处理时间之间！请再次输入！\n");
            printf("     请输入最小的处理时间：");
```

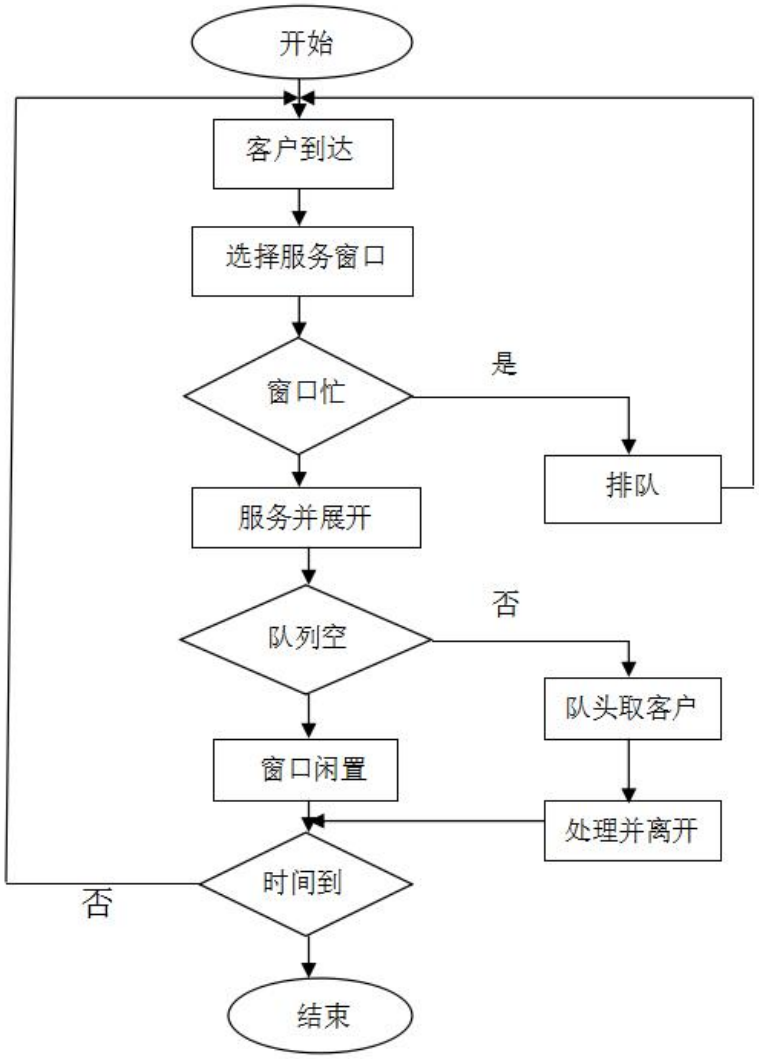
```
            scanf("%d",&DealMinTime);
            if(DealMinTime<=0||DealMinTime>=DealMaxTime){
                printf("    输入错误！最小处理时间必须介于零和最大处理时间之间！请最后一次输入！\n");
                printf("    请输入最小的处理时间：");
                scanf("%d",&DealMinTime);
                if(DealMinTime<=0 || DealMinTime>=DealMaxTime){
                    printf("    三次输入都错误！请按任意键退出！\n");
                    getch();
                    goto end;
                }
            }
        }
        printf("    请输入交易额的最大上限：");
        scanf("%d",&MaxAmount);
        if(MaxAmount>=BankAmount||MaxAmount>50000){
            printf("    输入错误！超出本银行的服务范围！最大交易额应低于银行开始营业时的资金总额且小于 50000！请再次输入！\n");
            printf("    请输入交易额的最大上限：");
            scanf("%d",&MaxAmount);
            if(MaxAmount>=BankAmount||MaxAmount>50000){
                printf("    输入错误！超出本银行的服务范围！最大交易额应低于银行开始营业时的资金总额且小于 50000！请最后一次输入！\n");
                printf("    请输入交易额的最大上限：");
                scanf("%d",&MaxAmount);
                if(MaxAmount>=BankAmount||MaxAmount>50000){
                    printf("    三次输入都错误！请按任意键退出！\n");
                    getch();
                    goto end;
                }
            }
        }
        MaxTime
+=rand()%(ClientArriveMaxTime-ClientArriveMinTime+1)+ClientArriveMinTime;
        //随机生成介于最大到达时间间隔和最小到达时间间隔之间的首次到达时间

        while(CurrentTime<CloseTime){                    //当前时间小于营业时间
            CurrentTime++;
            if(DealTime<CurrentTime)
                DealTime=CurrentTime ;
            if(DealTime==CurrentTime)                    //有窗口在处理交易
                state=1;
            if(CurrentTime==MaxTime){                    //到达事件
                ClientArrive();
```

```
                //随机生成介于最大到达时间间隔和最小到达时间间隔之间的到达时间
MaxTime+=rand()%(ClientArriveMaxTime-ClientArriveMinTime+1)+ClientArriveMinTime;
            }
            if(state==1&&Q1.front!=NULL){
                if(Q1.front->Amount>= 0){
                    InAmount();                         //调用存款函数
                    DealQ2();                           //调用搜索处理函数
                    NeedIn++;
                }
                else{
                    InAmount();                         //调用取款函数
                    NeedOut++;
                }
            }
        }
        printf("      客户序列\t      事件类型          时间          处理金额\n");
        while(Event.front!=NULL){                       //清除事件队列
            if(Event.front->Type=="离开"){
                printf("\t%d\t\t      离      开      \t\t%d\t\t%d\n",Event.front->num,
Event.front->EndTime,Event.front->Amount);
                if(Event.front->Amount>=0)              //成功存款人数
                    t1++;
                else                                    //成功取款人数
                    t3++;
            }
            else{
                printf("\t%d\t\t      到      达      \t\t%d\t\t%d\n",Event.front->num,
Event.front->BeginTime,Event.front->Amount);
                if(Event.front->Amount>=0)              //需要存款人数
                    t2++;
                else                                    //需要取款人数
                    t4++;
            }
            SuccessIn=NeedIn-(t2-t1);
            SuccessOut=NeedOut-(t4-t3);
            DeQueue(Event);
        }
        while(Q1.front!=NULL){
        //更新结束时第一队列中未处理的客户
            BankAmountTime+=(CloseTime-Q1.front->BeginTime);
            counter++;
            DeQueue(Q1);
        }
```

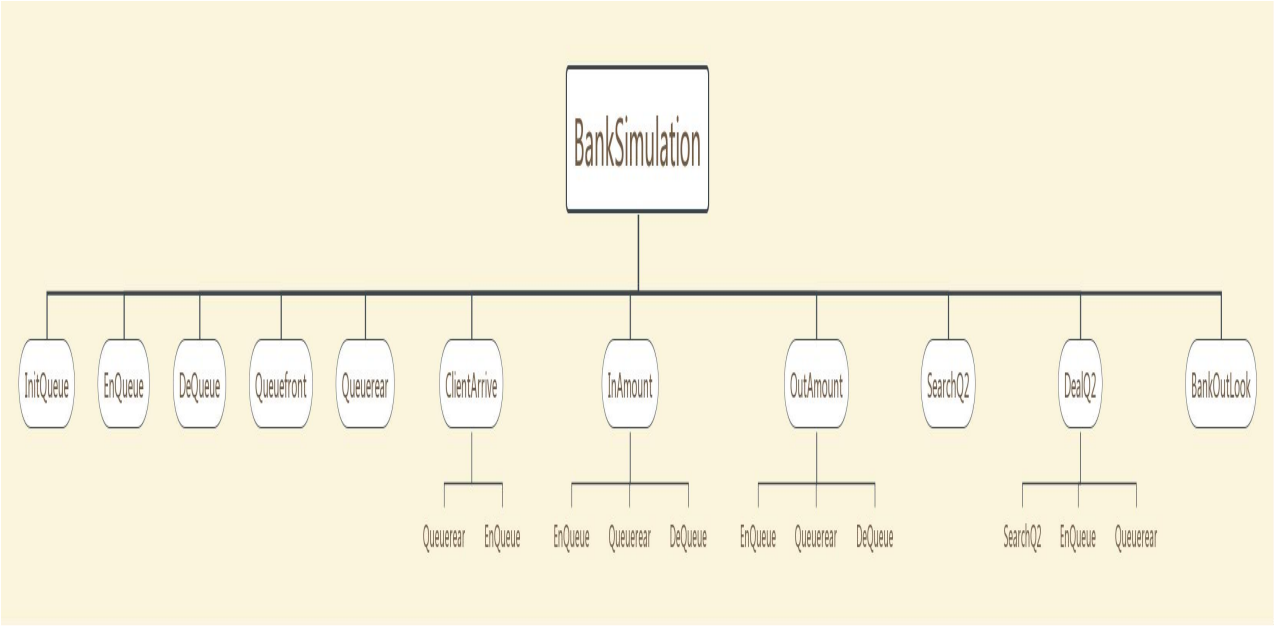
```
            printf("\n");
            printf("    需要存款的客户人数：%d\n",NeedIn);
            printf("    需要取款的客户人数：%d\n",NeedOut);
            printf("    成功办理存款的客户人数：%d\n",SuccessIn);
            printf("    成功办理取款的客户人数：%d\n",SuccessOut);
            printf("    存款成功办理率：%f\n",float(SuccessIn*100)/NeedIn);
            printf("    取款成功办理率：%f\n",float(SuccessOut*100)/NeedOut);
            printf("    客户逗留平均时间为:%f\n",float(BankAmountTime)/counter);
            printf("    银行当前余额：%d\n",BankAmount);
            printf("    请按任意键退出！\n");
            break;
        }
        if(n==0)
            printf("请按任意键退出！\n");
        end:getch();
        return 0;
}
```

3.5 函数调用关系及程序流程

以下为程序的大致流程图：


开始
客户到达
选择服务窗口
窗口忙
是
排队
服务并展开
队列空
否
队头取客户
窗口闲置
处理并离开
时间到
否
结束

函数调用关系图如下：


BankSimulation
InitQueue
EnQueue
DeQueue
Queuefront
Queuerear
ClientArrive
InAmount
OutAmount
SearchQ2
DealQ2
BankOutLook
Queuerear
EnQueue
EnQueue
Queuerear
DeQueue
EnQueue
Queuerear
DeQueue
SearchQ2
EnQueue
Queuerear

4. 调试分析

4.1 调试中遇到的问题

调试中遇到的问题不是很多，但遇到的问题在一定程度上让我更加的了解整个程序的运作机理，对于理解数据结构也有很大的帮助。

主要的问题在于一开始的时候实现检查 Q2 队列的接口时，在进行检查后，未再让经过检查却不满足处理的元素重新进入 Q2 队列，导致最后的元素缺少。经过调试发现了这个问题，对代码进行修正，最终解决了问题。

4.2 算法分析

接口	算法时间复杂度	算法空间复杂度
InitQueue(Queue &Q)	O(1)	O(1)
EnQueue(Queue &Q,int e)	O(n)	O(1)
DeQueue(Queue &Q)	O(1)	O(1)
Queuefront(Queue &Q)	O(1)	O(1)
Queuerear(Queue &Q)	O(1)	O(1)
ClientArrive()	O(n)	O(1)
InAmount()	O(1)	O(1)
OutAmount()	O(n)	O(1)
SearchQ2(Queue &Q,int m)	O(n)	O(1)
DealQ2()	O(n)	O(1)
BankOutLook()	O(1)	O(1)

4.3 经验体会

通过这次的课程设计的编写，学会了在多种数据结构之间进行巧妙的结合运用。同时，对于用到的多种数据结构也有了更多的了解。

在测试功能的时候一定要注意选取的测试数据的正确性和实用性。

5. 用户使用说明

在进入银行业务模拟界面时，选择操作编号，0-退出系统，1-进入模拟系统.

进入模拟系统后，输入的银行初始存款必须大于 0；

输入的银行营业时间必须大于 0 且必须小于 1440（一天）；

输入的最大到达时间间隔必须大于 0 且必须小于银行营业时间；

输入的最小到达时间间隔必须大于 0 且必须小于最大到达时间间隔；

输入的最大处理时间必须大于 0 且必须小于银行营业时间；

输入的最小处理时间必须大于 0 且必须小于最大处理时间；

输入的交易额的最大上线必须大于 0 且必须小于银行初始存款且必须小于 50000；

若输入有误会进行提示，三次错误后退出模拟系统。若输入无误，则开始进行输出，输出事件处理的列表信息，以及需要存款的客户人数，需要取款的客户人数，成功办理存款的客户人数，成功办理取款的客户人数，存款成功办理率，取款成功办理率，客户逗留平均时间，银行当前余额等信息。

6. 测试结果

6.1 输出测试 1

输入较大的银行初始存款，输入较大的到达时间间隔和较大的处理时间，较小的交易额上限

```
请输入选择的操作对应编号：1
请输入银行的初始存款：100000
请输入银行的营业时间：500
请输入最大到达时间间隔：30
请输入最小到达时间间隔：20
请输入最大的处理时间：30
请输入最小的处理时间：20
请输入交易额的最大上限：5000
```

由于到达时间间隔和处理时间输入较大，测试数据会比较少，相对于来说，求得的银行业务模拟的客户平均用时等信息可能就没有那么的精确

```
客户序列        事件类型        时间        处理金额
   1            到达          28          818
   1            离开          49          818
   2            到达          53          3317
   2            离开          77          3317
   3            到达          74          2105
   3            离开          103         2105
   4            到达          101         -3680
   4            离开          131         -3680
   5            到达          124         2390
   5            离开          146         2390
   6            到达          153         3534
   6            离开          175         3534
   7            到达          182         3298
   7            离开          203         3298
   8            到达          209         2966
   8            离开          235         2966
   9            到达          230         1772
   9            离开          256         1772
   10           到达          254         3675
   10           离开          276         3675
   11           到达          276         -4023
   11           离开          303         -4023
   12           到达          299         -1944
   12           离开          323         -1944
   13           到达          322         -2371
   13           离开          350         -2371
   14           到达          344         4806
   14           离开          371         4806
   15           到达          364         -4775
   15           离开          392         -4775
   16           到达          393         -434
   16           离开          421         -434
   17           到达          417         -3685
   17           离开          444         -3685
   18           到达          438         4059
   18           离开          465         4059
   19           到达          465         472
   19           离开          492         472
   20           到达          493         -3822
   20           离开          513         -3822

需要存款的客户人数：12
需要取款的客户人数：8
成功办理存款的客户人数：12
成功办理取款的客户人数：8
存款成功办理率：100.000000
取款成功办理率：100.000000
客户进入系统平均时为:25.299999
银行当前余额：108478
请按任意键退出!
```

6.2 输出测试 2

输入较大的银行初始存款，输入较小的到达时间间隔范围，较小的交易额上限

```
请输入选择的操作对应编号：1
请输入银行的初始存款：100000
请输入银行的营业时间：500
请输入最大到达时间间隔：10
请输入最小到达时间间隔：5
请输入最大的处理时间：10
请输入最小的处理时间：5
请输入交易额的最大上限：5000
```

在较小的到达时间间隔和处理时间的输入下，会得到比前一种情况更多的测试数据输出，在此情况下，样本的容量足够大，对于客户平均用时等信息的统计就可能会更加的精确一些

客户序列	事件类型	时间	处理金额
1	到达	9	4152
1	离开	18	4152
2	到达	14	-922
2	离开	24	-922
3	到达	21	-2231
3	离开	29	-2231
4	到达	29	-3599
4	离开	34	-3599
5	到达	39	-888
5	离开	48	-888
6	到达	48	-697
6	离开	56	-697
7	到达	56	2138
7	离开	63	2138
8	到达	61	1675
8	离开	68	1675
9	到达	67	-1837
9	离开	72	-1837
10	到达	76	3896
10	离开	83	3896
11	到达	85	1374
11	离开	93	1374
12	到达	92	-2989
12	离开	98	-2989
13	到达	100	-3479
13	离开	106	-3479
14	到达	105	-1868
14	离开	110	-1868
15	到达	113	3474
15	离开	123	3474
16	到达	118	4402
16	离开	127	4402
17	到达	127	3559
17	离开	136	3559
18	到达	137	-3215
18	离开	145	-3215
19	到达	143	-4255
19	离开	150	-4255
20	到达	148	4352
20	离开	156	4352
21	到达	158	1013
21	离开	166	1013
22	到达	166	4220
22	离开	174	4220
23	到达	172	1560
23	离开	181	1560
24	到达	179	-2940
24	离开	184	-2940
25	到达	188	-2
25	离开	195	-2
26	到达	197	1713
26	离开	207	1713
27	到达	205	-1828
27	离开	214	-1828
28	到达	210	3323
28	离开	215	3323
29	到达	219	-1889
29	离开	227	-1889
30	到达	228	-2915

30	离开	238	-2915
31	到达	234	2414
31	离开	243	2414
32	到达	240	-2185
32	离开	246	-2185
33	到达	246	4258
33	离开	252	4258
34	到达	254	2727
34	离开	261	2727
35	到达	259	-4931
35	离开	266	-4931
36	到达	267	-2950
36	离开	276	-2950
37	到达	277	1405
37	离开	283	1405
38	到达	285	1292
38	离开	290	1292
39	到达	294	-3268
39	离开	304	-3268
40	到达	302	-2557
40	离开	312	-2557
41	到达	310	-2970
41	离开	319	-2970
42	到达	316	-2437
42	离开	325	-2437
43	到达	322	4244
43	离开	327	4244
44	到达	332	-1624
44	离开	342	-1624
45	到达	338	4889
45	离开	343	4889
46	到达	345	-1743
46	离开	355	-1743
47	到达	355	2397
47	离开	362	2397
48	到达	363	949
48	离开	368	949
49	到达	371	-3219
49	离开	381	-3219
50	到达	377	1493
50	离开	385	1493
51	到达	383	-1895
51	离开	389	-1895
52	到达	392	-91
52	离开	400	-91
53	到达	397	2834
53	离开	407	2834
54	到达	402	-4009
54	离开	409	-4009
55	到达	408	-4847
55	离开	418	-4847
56	到达	415	-561
56	离开	420	-561
57	到达	421	4111
57	离开	429	4111
58	到达	430	1893
58	离开	437	1893
59	到达	440	3490
59	离开	447	3490
60	到达	445	-1213

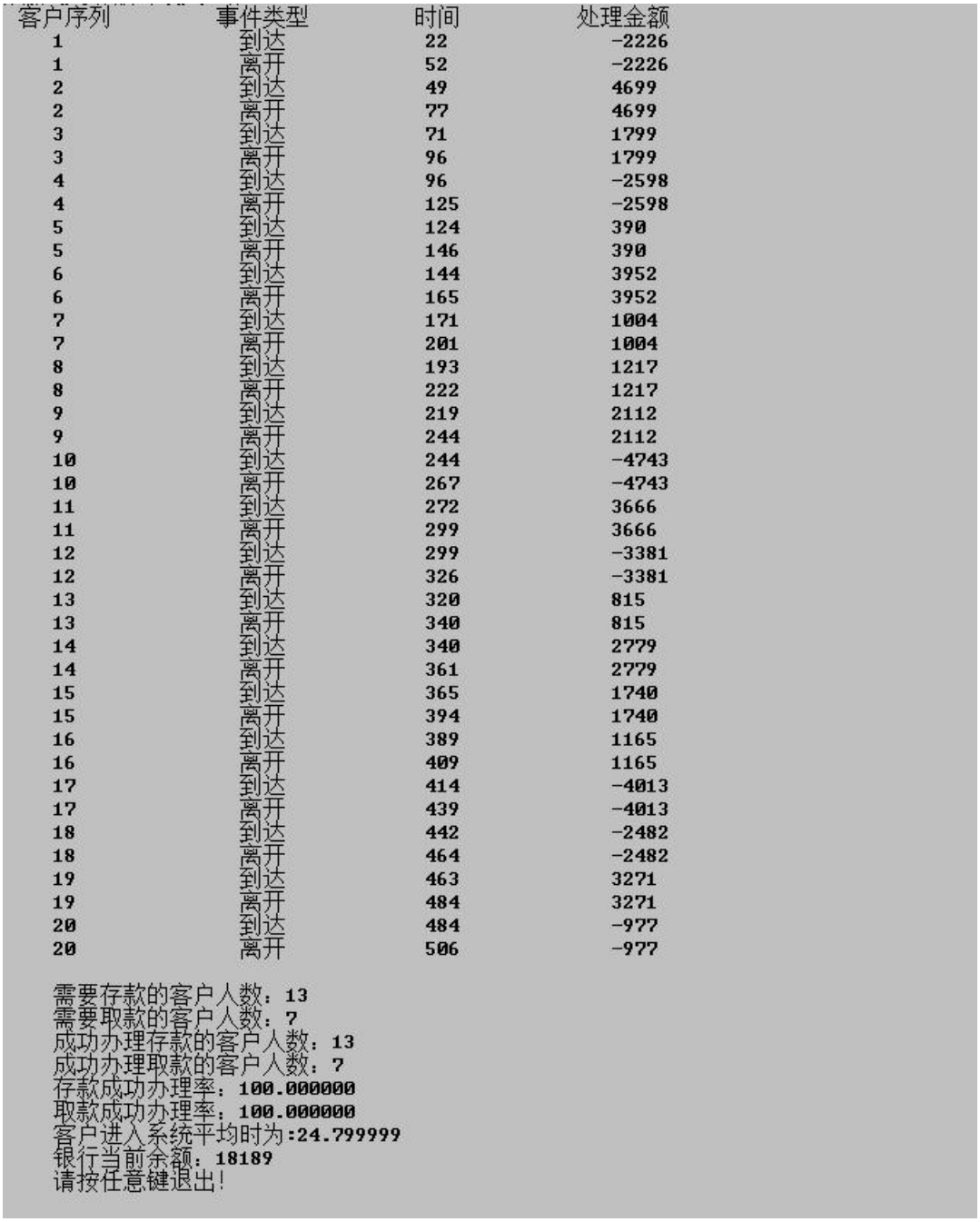
```
60              离开            453             -1213
61              到达            452             -2453
61              离开            458             -2453
62              到达            462             -897
62              离开            471             -897
63              到达            470             378
63              离开            475             378
64              到达            480             -4540
64              离开            488             -4540
65              到达            486             1673
65              离开            494             1673
66              到达            491             1980
66              离开            496             1980

需要存款的客户人数：31
需要取款的客户人数：35
成功办理存款的客户人数：31
成功办理取款的客户人数：35
存款成功办理率：100.000000
取款成功办理率：100.000000
客户进入系统平均时为:7.575758
银行当前余额：99334
请按任意键退出!
```

6.3 输出测试 3

输入较小的银行初始存款，较大的处理时间和较大的时间间隔，较大的交易金额上限。

```
请输入选择的操作对应编号：1
请输入银行的初始存款：10000
请输入银行的营业时间：500
请输入最大到达时间间隔：30
请输入最小到达时间间隔：20
请输入最大的处理时间：30
请输入最小的处理时间：20
请输入交易额的最大上限：5000
```

在这样的输出下，对于客户平均用时的影响比较大，且只能产生少量的数据。



7.参考文献

参考文献

[1] 张小艳，龚尚福编著. 数据结构与算法. 徐州：中国矿业大学出版社，2007

[2] 严蔚敏，吴伟民编著. 数据结构(C 语言版). 北京： 清华大学出版社，1997

[3] 谭浩强编著. C 程序设计(第三版). 北京： 清华大学出版社，2005

8.附录

文件夹
|3116004979 詹泽霖
-----|源代码
----------|Bank.h
----------|银行业务模拟系统.cpp
-----|银行业务模拟系统.exe
-----|课程设计报告.doc/docx
-----|课程设计报告.pdf