**河海大学物联网工程学院**

**数据结构与算法课程设计**

**银行模拟系统**

**组长 刘佳辉 2161310221 数据结构算法编写**

**组员 杨婧蕙 2162810203 可视化+算法改进**

**组员 孔 雯 2162810201 可视化+图像处理**

目录

[一、程序介绍 3](#_Toc137220228)

[1、课题介绍 3](#_Toc137220229)

[2、系统模块划分、各模块的功能介绍及模块之间的调用关系 3](#_Toc137220230)

[2.1数据模块 3](#_Toc137220231)

[2.2视图模块 4](#_Toc137220232)

[2.3控制模块 4](#_Toc137220233)

[2.4各模块之间的调用关系 4](#_Toc137220234)

[3、数据结构的设计与定义 5](#_Toc137220235)

[3.1 事件结点与事件链表 5](#_Toc137220236)

[3.2 顾客结点与窗口队列 6](#_Toc137220237)

[3.3 时间结构体与时间转换 6](#_Toc137220238)

[3.4 点类、路径类、贴图类、背景类 6](#_Toc137220239)

[3.5 全局变量 7](#_Toc137220240)

[4、核心算法流程图 8](#_Toc137220241)

[5、运行界面/结果截图 9](#_Toc137220242)

[5.1 开始界面 9](#_Toc137220243)

[5.2 银行模拟可视化页面 9](#_Toc137220244)

[5.3 营业结束页面 10](#_Toc137220245)

[6、程序实现（附程序代码和相应注释说明） 10](#_Toc137220246)

[6.1 重要变量定义 10](#_Toc137220247)

[6.2 可视化有关 11](#_Toc137220248)

[6.3 数据结构有关 17](#_Toc137220249)

[7、其他说明 25](#_Toc137220250)

[7.1 代码改进 25](#_Toc137220251)

[二、答辩记录 28](#_Toc137220252)

[三、总结 29](#_Toc137220253)

# 一、程序介绍

## 1、课题介绍

本项目为银行模拟系统。银行中设置了四个业务办理窗口。系统通过设计驱动事件和可视化界面，动态演示了各队列的变化情况，实时统计银行的存款与贷款总额和客户在银行的逗留时间。同时计算出到达银行的客户在每个队列排队时长，并为客户智能选择当前办理业务耗时最短的队伍。

本项目模拟了银行一天的业务情况，并且可为客户选择当前排队耗时最短的队伍，有效节约了客户办理业务的时间，具有较强的实用价值与现实意义。

## 2、系统模块划分、各模块的功能介绍及模块之间的调用关系

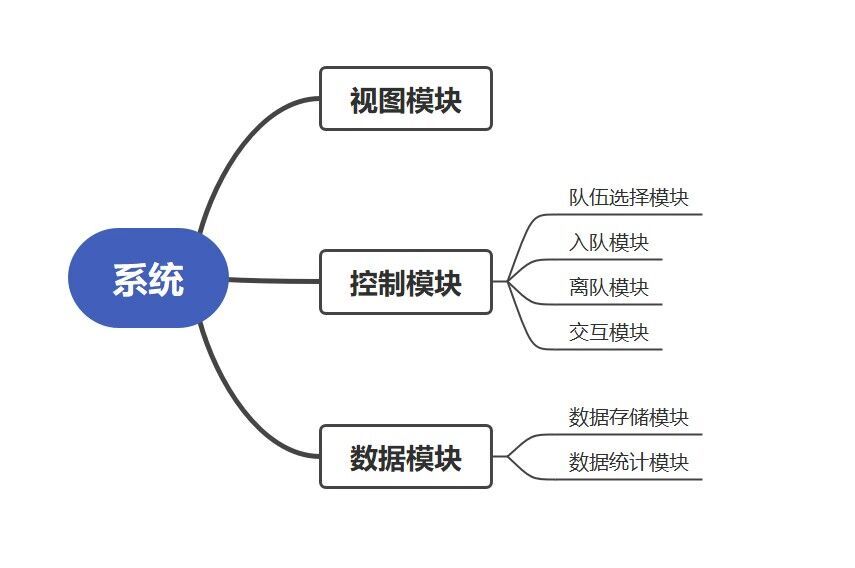


图2.1 模块结构划分

我们所做的银行排队模拟系统的模块结构划分如图2.1所示。对于系统的设计，我们将其划分为数据模块、视图模块和控制模块三大部分。其中，数据模块主要负责数据的存储与统计，视图模块主要负责可视化界面的搭建与美化，控制模块又细分成多个子模块，负责具体功能的实现，以及与另外两大模块的交互。

### 2.1数据模块

本模块的主要功能是存储各类数据以及完成部分数据的统计工作。

首先，我们将事件的信息封装成结构体，其内部存储了事件的发生时间、事件发生的类型（到达与离开）、顾客办理业务的类型（存款与取款）。

然后，我们将顾客的个人信息封装成结构体，其内部存储了顾客的到达时间、办理业务的时间、存取款的金额。

作为两个重要的操作对象，我们用链表存储事件，用队列存储顾客，此后的操作都是在这两个载体上进行。

此外，我们还定义了多个全局变量，用来存储顾客逗留时间、顾客所属队列、业务存取款金额等重要的数据，以及可视化需要的点和图片变量。

### 2.2视图模块

本模块的主要功能是可视化界面的搭建与美化。

本项目定义了一个点类、路径类、贴图类和背景类，来协助搭建可视化界面，它们的功能如下。

点类功能：其中存有视图中点的位置信息等。

路径类功能：其中有两个动态数组，一个用来存放路径的两个端点，用于确定路径的坐标位置；另一个用来存放路径的全部端点，用于确定贴图的坐标位置。此外，还有获取路径信息和绘制路径的功能。

贴图类功能：其中存有贴图的大小与坐标位置信息，还有图像的绘制与移动功能。

背景类功能：其中存有背景变量，还有绘制背景的功能。

### 2.3控制模块

本模块的主要功能是顾客到达和离开等具体操作的实现，以及与数据模块、视图模块的交互。

本模块中定义了创建新事件和新顾客、初始化事件链表和顾客队列、链表与队列的插入删除、最短队列选取（结合每队的总逗留时间）、时间转换、顾客到达函数、顾客离开函数等操作。

同时，还设计了相关函数，在一些特定的条件下，可以调用视图模块的方法绘制图像以及修改更新数据模块存储的数据。

### 2.4各模块之间的调用关系

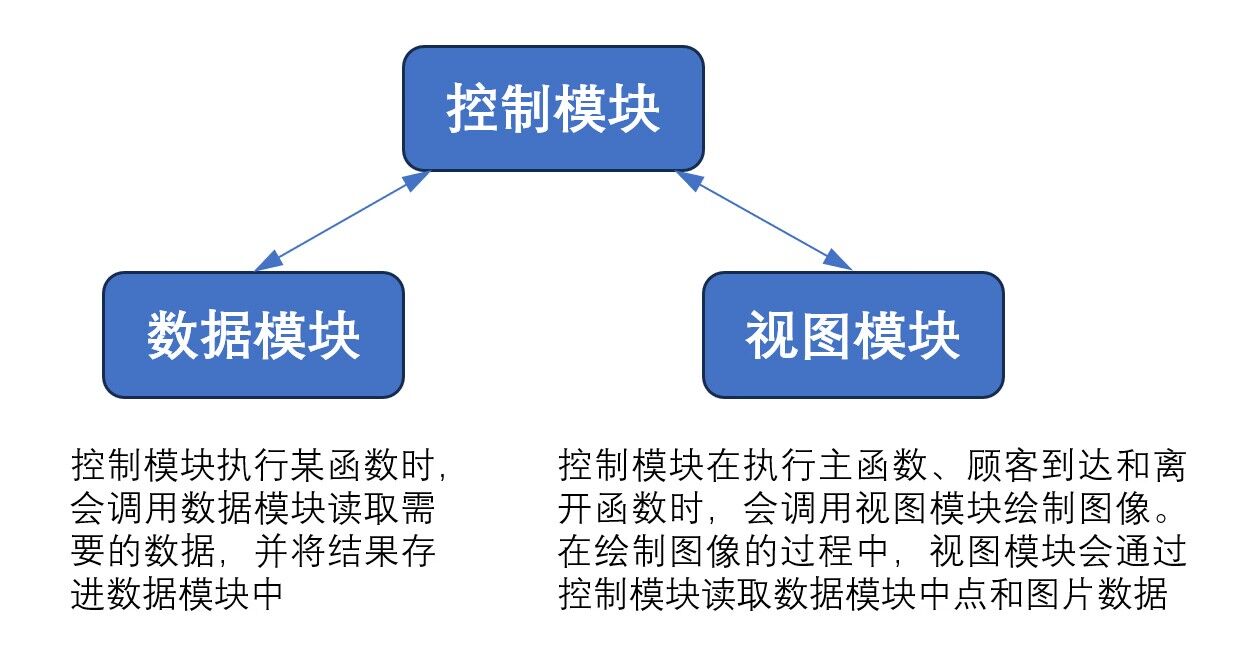


图2.2 模块调用关系

如图2.2所示，控制模块执行某个函数时，会调用数据模块读取需要的数据，并将结果存进数据模块中。

控制模块在执行主函数、顾客到达和离开函数时，会调用视图模块绘制图像。在绘制图像的过程中，视图模块会通过控制模块读取数据模块中点和图片数据。

## 3、数据结构的设计与定义

数据结构由逻辑结构、存储结构和数据运算结构三部分组成。对于逻辑结构，我们选择的是线性结构；对于存储结构，我们选择的是链式存储与顺序存储相结合；对于数据结构算法，涉及到了生成、销毁、查找、插入、删除、遍历等操作。具体的数据结构如下。

### 3.1 事件结点与事件链表

对于驱动事件的设计，我们采用的是定义一个事件结构体，其存储发生事件的具体信息。同时，构造一个事件链表，将事件结点按时间顺序插入链表中。这样，每次获取并执行链表的首元结点，就能按时间顺序实现银行一天的业务模拟。

1. **Event（事件结构体）**

int OccurTime：事件发生时刻。

int NType：事件类型，0表示到达事件，1至4表示四个窗口的离开事件。

int DealAmount：办理业务类型，正数为存款，负数为取款。

struct Event\* next：指向下一个事件结点的指针。

1. **EventList/LinkList（事件链表）**

Event\* head：链表头指针。

Event\* tail：链表尾指针。

int len：链表长度。

1. **相关操作**

NewEvent：创建新事件，根据传送的参数初始化一个新结点。

InitList：创建并初始化事件链表，让链表的头指针和尾指针都指向头节点。

ListEmpty：判断链表是否为空，如果链表为空，返回TRUE，否则返回FALSE。

ListTraverse：遍历链表并输出事件内容。

OrderInsert：将事件结点按发生时间先后顺序插入有序链表中

DelFirst：若链表不为空，删除其首结点，用e返回删除的结点。

### 3.2 顾客结点与窗口队列

我们定义了一个结构体，用来存储顾客的个人信息。此外，创建了四个队列，用来存储顾客结点，模拟银行的四个业务窗口。

1. **CustNode（顾客结点）**

int BeginTime：顾客到达时间。

int Duration：办理业务时间。

int Amount：正数为存款额，负数为贷款额。

struct CustNode\* next：指向下一个事件结点指针域。

1. **LinkQueue（窗口队列）**

CustNode\* front：队列头指针。

CustNode\* rear：队列尾指针。

int length：队列长度。

1. **相关操作**

InitQueue：初始化队列，让队列的头指针和尾指针都指向头节点。

EmptyQueue：判断队列是否为空，如果队列为空，返回TRUE，否则返回FALSE。

QueueTraverse：遍历队列。

EnQueue：入队操作，接收一个参数，在队列尾部插入一个新结点。

DelQueue：出队操作，若队列不为空，删除队列的首结点，用e返回删除的结点。

QueueLength：获取队列的长度。

GetHead：若队列不为空，用e返回队列的首结点。

### 3.3 时间结构体与时间转换

本项目的设计过程中所有时间都是以分钟计时的，而最后输出的是具体的时刻。因此我们定义了时间结构体和相关操作，完成时间的转换。

1. **Time（时间结构体）**

int Hour：小时。

int Minute：分钟。

1. **相关操作**

TransTime：接收一个分钟数，并将其转换为时刻。

### 3.4 点类、路径类、贴图类、背景类

为了可视化的有序展开，我们设计了点类、路径类、贴图类、背景类来分别封装视图的点、线、面信息与相关操作。它们具有以下的成员变量与函数。

1. **Point（点类）**

float x, y：点的横、纵坐标。

构造函数：接收两个float类型的数据，并将其设置为点的坐标。

1. **Path（路径类）**

vector<Point> keyPoints：存储路径的两个端点。

float sampleInterval：路径上两个点的距离。

vector<Point> allPoints：存储路径上所有的点。

getAllPoints：获得路径上所有的点，并存入数组中。

draw：根据数组中存储的点绘制路径。

1. **People（贴图类）**

IMAGE im\_egg1,…, im\_egg12：表示不同的贴图。

Point center：贴图的端点坐标。

float radius：图片宽度。

int indexInPath：贴图位置在Path的allPoints中的对应序号。

int type：贴图类型。

movetoIndexInPath：确定贴图坐标。

Initiate：初始化贴图的形状和大小。

draw：根据type的值绘制不同图片。

1. **Background（背景类）**

IMAGE im\_background：表示背景。

draw：绘制背景图片。

### 3.5 全局变量

int Open：银行开门时间（离0点的分钟数）。

int BankAmount：初始时银行现存资金总额。

int CloseTime：营业结束时间-开门时间。

int TotalTime：客户逗留总时间。

int CustomerNum：客户总数。

int InAmount：存款总额。

int OutAmount：贷款总额。

int m[numHeight]：插入的客户位置（indexInPath）。

int Currentamount[numHeight]：正在办理的业务金额。

int Queuetime[numHeight]：每个队列当前最后一个顾客离开的时间。

EventList ev：事件链表。

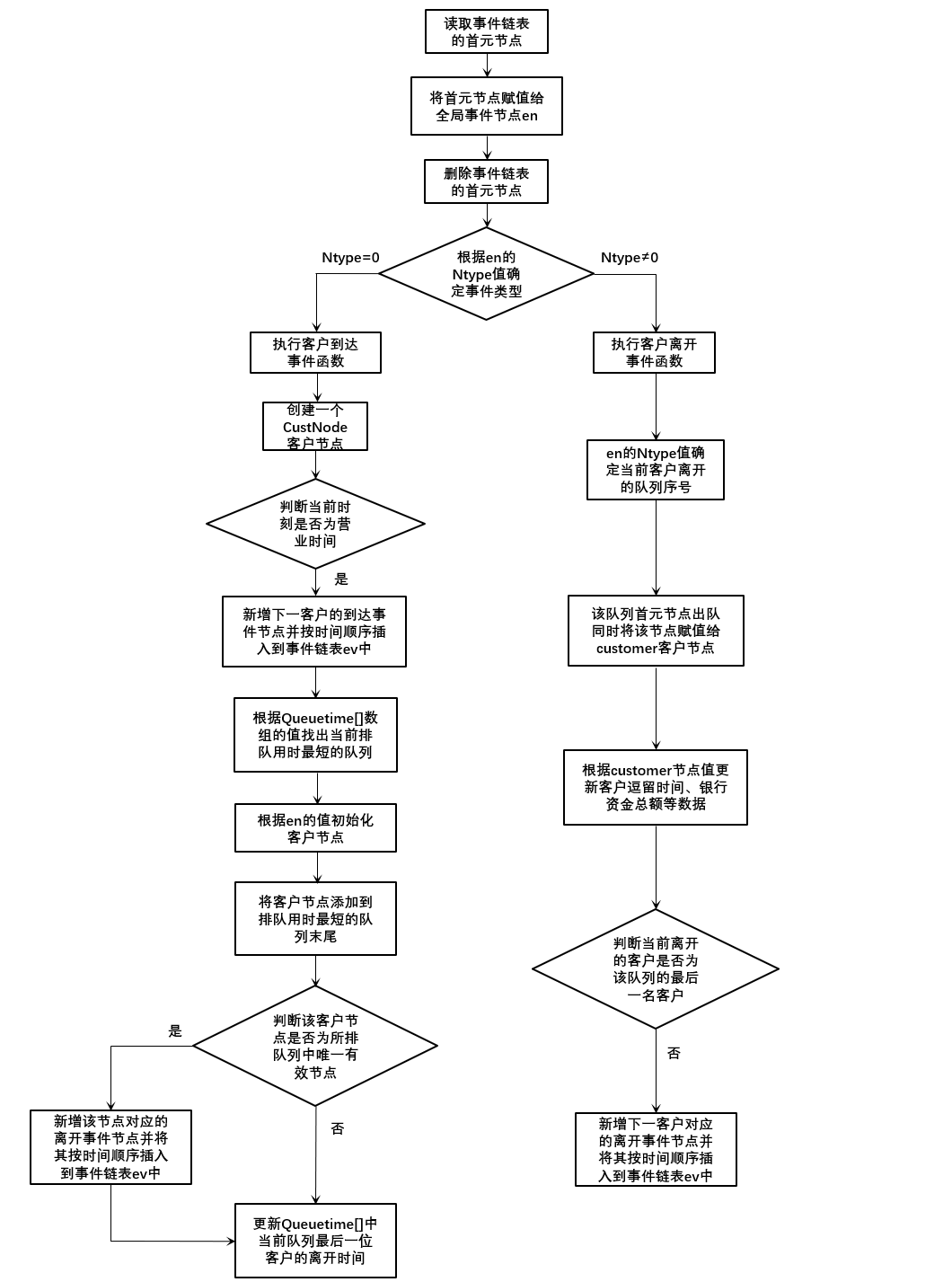
Event en：事件。

LinkQueue q[5]：队列数组存放队列。

CustNode customer：客户节点。

int status：设置状态标记，等于0是初始界面，1是银行内部界面，2是结束界面。

## 4、核心算法流程图

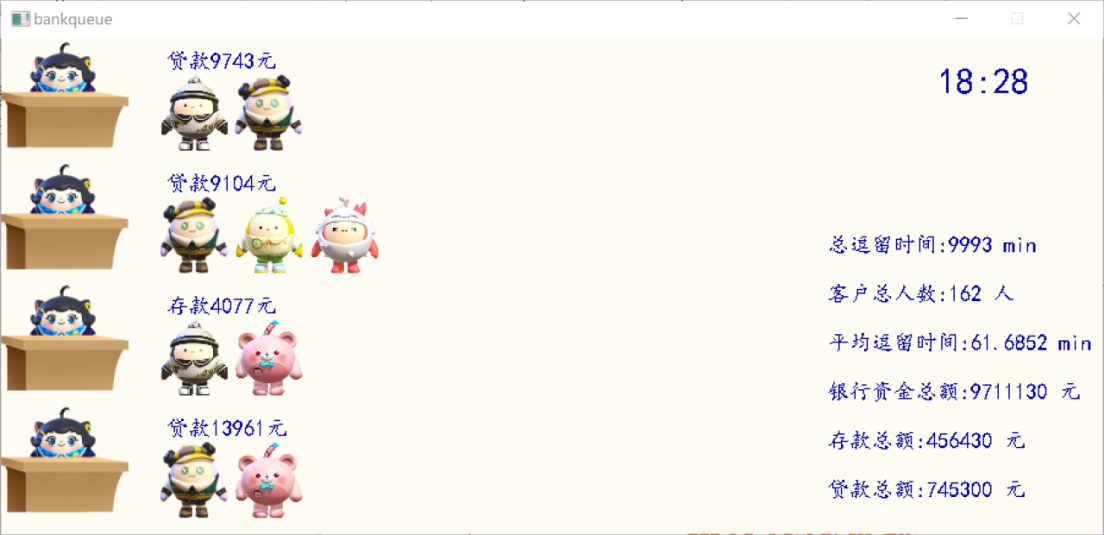


## 5、运行界面/结果截图

### 5.1 开始界面



### 5.2 银行模拟可视化页面



### 5.3 营业结束页面



## 6、程序实现（附程序代码和相应注释说明）

### 6.1 重要变量定义

typedef struct Time { //时间类型

int Hour;

int Minute;

}Time;

typedef struct Event { //事件类型

int OccurTime; //事件发生时刻

int NType; //事件类型，0表示到达事件，1至4表示四个窗口的离开事件

int DealAmount; //办理业务类型,正数为存款，负数为取款

struct Event\* next;

}Event;

typedef struct { //单向链表结构

Event\* head; //头指针

Event\* tail; //尾指针

int len; //长度

}LinkList;

typedef LinkList EventList; //事件链表

typedef struct CustNode {//顾客结构

int BeginTime; //到达时间

int Duration; //办理业务时间

int Amount; //正数为存款额，负数为贷款额

struct CustNode\* next; //指针域

}CustNode;

typedef struct LinkQueue {//队列结构

CustNode\* front;//头指针

CustNode\* rear;//尾指针

}LinkQueue;

### 6.2 可视化有关

class Point //定义点

{

public:

float x, y;

Point()

{

}

Point(float ix, float iy)

{

x = ix;

y = iy;

}

};

class Path //定义路径类

{

public:

vector<Point> keyPoints;

float sampleInterval;

vector<Point> allPoints;

void getAllPoints() //求得该路径上间隔相同的所有点坐标

{

int i=0;

float xd = keyPoints[i + 1].x - keyPoints[i].x;

float length = xd;

int num = length / sampleInterval;

for (int j = 0; j < num; j++)

{

float x\_sample = keyPoints[i].x + j \* xd / num;

allPoints.push\_back(Point(x\_sample, keyPoints[i].y));

}

allPoints.push\_back(Point(keyPoints[i].x, keyPoints[i].y));

}

void draw() //画出路径及路径上所有点，测试用

{

setlinecolor(RGB(0, 0, 0));

setfillcolor(RGB(0, 0, 0));

for (int i = 0; i < keyPoints.size() - 1; i++)

line(keyPoints[i].x, keyPoints[i].y, keyPoints[i + 1].x, keyPoints[i + 1].y);

for (int i = 0; i < allPoints.size(); i++)

fillcircle(allPoints[i].x, allPoints[i].y, 3);

}

};

class People // 定义蛋仔类

{

public:

IMAGE im\_egg1, im\_egg2, im\_egg3, im\_egg4, im\_egg5, im\_egg6, im\_egg7, im\_egg8, im\_egg9, im\_egg10, im\_egg11, im\_egg12;

Point center; // 端点坐标

float radius; // 图片宽度

int indexInPath; // 蛋仔位置在Path的allPoints中的对应序号

int type = rand() % 8;

void draw() // 随机函数，画出不同的蛋仔

{

if (type == c1 )

putimagePng(center.x, center.y, &im\_egg1);

else if (type == c2)

putimagePng(center.x, center.y, &im\_egg2);

else if (type == c3)

putimagePng(center.x, center.y, &im\_egg3);

else if (type == c4)

putimagePng(center.x, center.y, &im\_egg4);

else if (type == c5)

putimagePng(center.x, center.y, &im\_egg5);

else if (type == c6)

putimagePng(center.x, center.y, &im\_egg6);

else if (type == c7)

putimagePng(center.x, center.y, &im\_egg7);

else if (type == c8)

putimagePng(center.x, center.y, &im\_egg8);

else if (type == c9)

putimagePng(center.x, center.y, &im\_egg9);

else if (type == c10)

putimagePng(center.x, center.y, &im\_egg10);

else if (type == c11)

putimagePng(center.x, center.y, &im\_egg11);

else if (type == c12)

putimagePng(center.x, center.y, &im\_egg12);

}

void movetoIndexInPath(Path path)

{

center = path.allPoints[indexInPath];

}

void initiate(Path path) // 初始化蛋仔

{

radius = Radius; // 半径

indexInPath = 0;

movetoIndexInPath(path);

}

};

class Background

{

public:

IMAGE im\_background;

void draw()

{

putimagePng(0, 0, &im\_background);

}

};

void startupBank() {//初始化银行模拟页面

loadimage(&im\_egg1, \_T("egg1.png"), 223 \* 0.25, 251 \* 0.25);//导入蛋仔图片

loadimage(&im\_egg2, \_T("egg2.png"), 223 \* 0.25, 251 \* 0.25);

loadimage(&im\_egg3, \_T("egg3.png"), 223 \* 0.25, 251 \* 0.25);

loadimage(&im\_egg4, \_T("egg4.png"), 223 \* 0.25, 251 \* 0.25);

loadimage(&im\_egg5, \_T("egg5.png"), 223 \* 0.25, 251 \* 0.25);

loadimage(&im\_egg6, \_T("egg6.png"), 223 \* 0.25, 251 \* 0.25);

loadimage(&im\_egg7, \_T("egg7.png"), 223 \* 0.25, 251 \* 0.25);

loadimage(&im\_egg8, \_T("egg8.png"), 223 \* 0.25, 251 \* 0.25);

loadimage(&im\_egg9, \_T("egg9.png"), 223 \* 0.25, 251 \* 0.25);

loadimage(&im\_egg10, \_T("egg10.png"), 223 \* 0.25, 251 \* 0.25);

loadimage(&im\_egg11, \_T("egg11.png"), 223 \* 0.25, 251 \* 0.25);

loadimage(&im\_egg12, \_T("egg12.png"), 223 \* 0.25, 251 \* 0.25);

loadimage(&im\_background, \_T("background1.png"), 1800 \* 0.5, 810 \* 0.5);//导入背景图片

cleardevice();

for (int i = 0; i < numHeight; i++) {//得到各队列路径上的蛋仔索引点

path[i].keyPoints.push\_back(Point(130, 30+100\*i));

path[i].keyPoints.push\_back(Point(930, 30+100\*i));

path[i].sampleInterval = Radius;

path[i].getAllPoints();

}

background.im\_background = im\_background;//背景图片赋值

BeginBatchDraw();// 开始批量绘制

}

void showBank() {//银行模拟页面显示

for (int i = 0; i < numHeight; i++) {//绘制正在排队的蛋仔

for (int j = 0; j < people[i].size(); j++) {

people[i][j].draw();

}

}

Time T;//页面右上角显示当前时间

T = TransTime(en.OccurTime);

TCHAR s[20];

setbkmode(TRANSPARENT);

if (T.Hour < 10 && T.Minute<10) {//时间格式化

\_stprintf\_s(s, \_T("0%d:0%d"), T.Hour, T.Minute);

}

else if (T.Hour < 10 && T.Minute >= 10) {

\_stprintf\_s(s, \_T("0%d:%d"), T.Hour, T.Minute);

}

else if (T.Hour >= 10 && T.Minute < 10) {

\_stprintf\_s(s, \_T("%d:0%d"), T.Hour, T.Minute);

}

else

\_stprintf\_s(s, \_T("%d:%d"),T.Hour,T.Minute);

settextcolor(BLUE);

settextstyle(30, 0, \_T("黑体"));

outtextxy(screenWidth \* 0.85, 20, s);

for (int i = 0; i < numHeight; i++) {//遍历所有队列，显示当前正在办理的业务和金额

TCHAR d[20];

if (Currentamount[i] < 0) {

\_stprintf\_s(d, \_T("贷款%d元"), -1\*Currentamount[i]);

}

else {

\_stprintf\_s(d, \_T("存款%d元"), Currentamount[i]);

}

settextcolor(BLUE);

settextstyle(18, 0, \_T("楷体"));

outtextxy(screenWidth \* 0.15, 10+100\*i, d);

}

TCHAR m[20],n[20], o[20], p[20], q[20], r[20];//页面右侧实时显示当前营业额

\_stprintf\_s(m, \_T("总逗留时间:%d min"), TotalTime);

\_stprintf\_s(n, \_T("客户总人数:%d 人"), CustomerNum);

\_stprintf\_s(o, \_T("平均逗留时间:%g min"), (float)TotalTime / CustomerNum);

\_stprintf\_s(p, \_T("银行资金总额:%d 元"), BankAmount);

\_stprintf\_s(q, \_T("存款总额:%d 元"), InAmount);

\_stprintf\_s(r, \_T("贷款总额:%d 元"), OutAmount);

settextcolor(BLUE);

settextstyle(18, 0, \_T("楷体"));

outtextxy(screenWidth \* 0.75, 160, m);

outtextxy(screenWidth \* 0.75, 200, n);

outtextxy(screenWidth \* 0.75, 240, o);

outtextxy(screenWidth \* 0.75, 280, p);

outtextxy(screenWidth \* 0.75, 320, q);

outtextxy(screenWidth \* 0.75, 360, r);

}

void startupPage() {//初始化页面

mciSendString(\_T("open BGM.mp3 alias bkmusic"), NULL, 0, NULL);//打开背景音乐

mciSendString(\_T("play bkmusic repeat"), NULL, 0, NULL);//循环播放

loadimage(&im\_welcome, \_T("welcome.png"));//导入图片

loadimage(&im\_startButton, \_T("startButton.png"));

loadimage(&im\_closed, \_T("closed.png"));

initgraph(screenWidth, screenHeight);

cleardevice();

BeginBatchDraw();// 开始批量绘制

}

void PlayMusicOnce(TCHAR fileName[80])//播放一次音乐函数

{

TCHAR cmdString1[50];

\_stprintf\_s(cmdString1, \_T("open %s alias tmpmusic"), fileName);//生成命令字符串

mciSendString(\_T("close tmpmusic"), NULL, 0, NULL);//先把前面一次的音乐关闭

mciSendString(cmdString1, NULL, 0, NULL);//打开音乐

mciSendString(\_T("play tmpmusic"), NULL, 0, NULL);//仅播放一次

}

void welcomePageShow() {//开始页面

putimagePng(0, 0, &im\_welcome);//显示开始界面

putimagePng(400, 340, &im\_startButton);

FlushBatchDraw();

}

void closedPageShow() {//结束页面

putimagePng(0, 0, &im\_closed);//显示结束界面

TCHAR m[20], n[20], o[20], p[20], q[20], r[20];//显示一天的营业额

\_stprintf\_s(m, \_T("总逗留时间:%d min"), TotalTime);

\_stprintf\_s(n, \_T("客户总人数:%d 人"), CustomerNum);

\_stprintf\_s(o, \_T("平均逗留时间:%g min"), (float)TotalTime / CustomerNum);

\_stprintf\_s(p, \_T("银行资金总额:%d 元"), BankAmount);

\_stprintf\_s(q, \_T("存款总额:%d 元"), InAmount);

\_stprintf\_s(r, \_T("贷款总额:%d 元"), OutAmount);

settextcolor(BLUE);

settextstyle(18, 0, \_T("楷体"));

outtextxy(screenWidth \* 0.2, 260, m);

outtextxy(screenWidth \* 0.2, 300, n);

outtextxy(screenWidth \* 0.2, 340, o);

outtextxy(screenWidth \* 0.55, 260, p);

outtextxy(screenWidth \* 0.55, 300, q);

outtextxy(screenWidth \* 0.55, 340, r);

FlushBatchDraw();

}

bool bankCloseJudge() {//判断银行是否关门

if (TransTime(en.OccurTime).Hour >= 17 && people[0].size() == 0 && people[1].size() == 0 && people[2].size() == 0 && people[3].size() == 0)

return true;

else

return false;

}

void updateWithInput() {//与输入有关的更新

MOUSEMSG m;//定义鼠标消息

if (MouseHit())//如果鼠标有消息

{

m = GetMouseMsg();//获得鼠标消息

if (m.x >= 400 && m.x <= 500 && m.y >= 340 && m.y <= 380 && m.uMsg == WM\_LBUTTONDOWN)//如果鼠标经过特定区域

{

status = 1;

}

}

}

### 6.3 数据结构有关

//初始化操作

void OpenForDay() {

InitList(&ev);//初始化事件队列

en.OccurTime = 0;

en.NType = 0;

en.DealAmount = rand() % 30000 - 15000;

OrderInsert(&ev, en);

for (int i = 1; i <= 4; i++)

InitQueue(&q[i]);//初始化四个窗口队列

}

Time TransTime(int minute) {//转换时间格式

int timeh, timem;

timeh = (Open + minute) / 60;

timem = (Open + minute) % 60;

Time time;

time.Hour = timeh;

time.Minute = timem;

return time;

}

//根据OccurTime和NType值，创建新事件

Event NewEvent(int occurT, int nType, int money) {

Event e;

e.OccurTime = occurT;

e.NType = nType;

e.DealAmount = money;

return e;

}

//初始化事件链表

int InitList(LinkList\* L) {

L->head = L->tail = (Event\*)malloc(sizeof(Event));

if (!L->head) {

printf("申请空间失败！\n");

exit(0);

}

L->head->next = NULL;

return OK;

}

//判断链表L是否为空

int ListEmpty(LinkList\* L) {

if ((L->head == L->tail) && (L->head != NULL))

return TRUE;

else

return FALSE;

}

//将事件e按发生时间顺序插入有序链表L中

int OrderInsert(LinkList\* L, Event e) {

Event\* p, \* q, \* newptr;

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

if (!newptr) {

printf("申请空间失败！\n");

exit(0);

}

\*newptr = e;

if (TRUE == ListEmpty(L)) {//链表为空

L->head->next = newptr;

L->tail = newptr;

L->tail->next = NULL;

return OK;

}

q = L->head;

p = L->head->next;

while (p) {//遍历整个链表

if (p->OccurTime >= newptr->OccurTime)

break;

q = p;

p = p->next;

}

q->next = newptr;

newptr->next = p;

if (!p)//插入位置为链表尾部

L->tail = newptr;

return OK;

}

//链表L不为空，删除其首结点，用e返回

int DelFirst(LinkList\* L, Event\* e) {

Event\* p = L->head->next;

if (!p)

return ERROR;

L->head->next = p->next;

\*e = \*p;

free(p);

if (L->head->next == NULL)

L->tail = L->head;

return OK;

}

//遍历链表

int ListTraverse(LinkList\* L) {

Event\* p = L->head->next;

string type, deal;

Time T;

if (!p) {

printf("链表为空！\n");

return ERROR;

}

while (p != NULL) {

if (p->NType == 0) type = "到达";

else type = "离开" + to\_string(p->NType) + "号窗口";

if (p->DealAmount >= 0) deal = "存款" + to\_string(p->DealAmount) + "元";

else deal = "贷款" + to\_string(-1 \* p->DealAmount) + "元";

T = TransTime(p->OccurTime);

// printf("事件发生时间：%d点%d分，事件类型：%s，交易类型：%s\n", T.Hour, T.Minute, type.c\_str(), deal.c\_str());

p = p->next;

}

printf("\n");

return OK;

}

//输出事件链表

void PrintEventList() {

// printf("事件链表:\n");

ListTraverse(&ev);

}

//初始化队列Q

int InitQueue(LinkQueue\* Q) {

Q->front = Q->rear = (CustNode\*)malloc(sizeof(CustNode));

if (!Q->front) {

printf("申请空间失败！\n");

exit(0);

}

Q->front->next = NULL;

return OK;

}

//若队列Q为空，返回TRUE，否则返回FALSE

int EmptyQueue(LinkQueue\* Q) {

if (Q->front == Q->rear && Q->front != NULL)

return TRUE;

else

return FALSE;

}

//结点e入队Q

int EnQueue(LinkQueue\* Q, CustNode e) {

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

if (!p) {

printf("申请空间失败！\n");

exit(0);

}

\*p = e;

p->next = NULL;

Q->rear->next = p;//插入队尾

Q->rear = p;//修改队尾指针

return OK;

}

//若队列Q不为空，首结点出队，用e返回

int DelQueue(LinkQueue\* Q, CustNode\* e) {

CustNode\* p = Q->front->next;

if (!p)

return ERROR;

\*e = \*p;

Q->front->next = p->next;//修正队首指针

free(p);

if (!Q->front->next)//队空

Q->rear = Q->front;

return OK;

}

//返回队列Q的长度，即元素个数

int QueueLength(LinkQueue Q) {

int count = 0;

CustNode\* p = Q.front->next;

while (p) {

p = p->next;

count++;

}

return count;

}

//返回长度为n的数组a第一个最小值的下标，从0开始

int Min(int a[], int n) {

int i, tmp, index = 0;

tmp = a[0];

for (i = 1; i < n; i++) {

if (a[i] < tmp) {

tmp = a[i];

index = i;

}

}

return index;

}

//获取最短队列的编号

int ShortestQueue() {

int i, a[4];

for (i = 1; i <= 4; i++) {

a[i - 1] = QueueLength(q[i]);

//printf("队%d的长度为%d\n",i,QueueLength(q[i]));

}

return Min(a, 4) + 1;//队列从1开始编号

}

//若队列Q不为空，用e返回其首结点

int GetHead(LinkQueue\* Q, CustNode\* e) {

if (EmptyQueue(Q))

return ERROR;

\*e = \*(Q->front->next);

return OK;

}

//遍历队列Q

int QueueTraverse(LinkQueue\* Q,int i) {

CustNode\* p = Q->front->next;

string deal;

Time T;

if (!p) {

printf("队列为空！\n");

return ERROR;

}

Currentamount[i - 1] = p->Amount;//求得当前办理业务金额，并将其赋值给全局变量Currentamount

while (p) {

if (p->Amount >= 0) deal = "存款" + to\_string(p->Amount) + "元";

else deal = "贷款" + to\_string(-1 \* p->Amount) + "元";

T = TransTime(p->BeginTime);

p = p->next;

}

// printf("\n");

return OK;

}

//打印当前队列

void PrintQueue() {

int i;

for (i = 1; i <= 4; i++) {

// printf("队列 %d 有 %d 个客户:", i, QueueLength(q[i]));

QueueTraverse(&q[i],i);

}

// printf("\n");

}

//顾客达到事件

void CustomerArrived() {

int durtime, intertime, money, i, j, t, minqueue = 0;

CustNode e;

Time T1, T2;

CustomerNum++;

intertime = rand() % 5 + 1;//间隔时间

durtime = rand() % 10 + 10;//办理业务时间

money = rand() % 30000 - 15000;//客户要办理的金额

t = en.OccurTime + intertime;//下一个顾客到达时间

if (en.OccurTime < CloseTime) {//银行尚未关门

T1 = TransTime(en.OccurTime);

T2 = TransTime(t);

OrderInsert(&ev, NewEvent(t, 0, money));//插入到达事件

for (j = 1; j < numHeight; j++) {//遍历所有队列，选择结束时间最早的队列

if (Queuetime[j] < Queuetime[minqueue]) {

minqueue = j;

}

}

i = minqueue + 1;

People ball;

ball.im\_egg1 = im\_egg1;

ball.im\_egg2 = im\_egg2;

ball.im\_egg3 = im\_egg3;

ball.im\_egg4 = im\_egg4;

ball.im\_egg5 = im\_egg5;

ball.im\_egg6 = im\_egg6;

ball.im\_egg7 = im\_egg7;

ball.im\_egg8 = im\_egg8;

ball.im\_egg9 = im\_egg9;

ball.im\_egg10 = im\_egg10;

ball.im\_egg11 = im\_egg11;

ball.im\_egg12 = im\_egg12;

ball.initiate(path[i - 1]);//初始化蛋仔路径

ball.indexInPath = m[i - 1];//初始化队列末端蛋仔索引位置

ball.movetoIndexInPath(path[i - 1]);//确定蛋仔索引所在位置

people[i - 1].push\_back(ball);//将蛋仔增添到相应队列末端

m[i - 1]++;//队列末端蛋仔索引后移

e.BeginTime = en.OccurTime;//顾客到达时间赋值

e.Duration = durtime;//顾客办理业务时间赋值

e.Amount = en.DealAmount;//顾客办理业务金额赋值

EnQueue(&q[i], e);//顾客e入队

if (QueueLength(q[i]) == 1) {//判断到达前队列是否为空

OrderInsert(&ev, NewEvent(en.OccurTime + durtime, i, en.DealAmount));//如果到达时该窗口没人，插入该顾客的离开事件

Queuetime[i - 1] = en.OccurTime + durtime;//如果到达时该窗口没人，则该队列办理结束时间等于当前时间+该顾客的办理时间

}

else {

Queuetime[i - 1] += durtime;//如果到达时该窗口有人，则该队列办理结束时间等于原队列结束时间+该顾客的办理时间

}

}

else {

// printf("银行关门了！");

}

}

//顾客离开事件

void CustomerDepature() {

int i = en.NType;//确定该顾客离开时的窗口号

Time T;

T = TransTime(en.OccurTime);

DelQueue(&q[i], &customer);// 删除队列头节点并将其赋值给全局变量customer

// printf("客户离开时间:%d点%d分\n", T.Hour, T.Minute);//输出顾客离开时间

TotalTime += en.OccurTime - customer.BeginTime;//更新顾客总逗留时间

BankAmount += en.DealAmount;//更新业务总金额

if (en.DealAmount >= 0) InAmount += en.DealAmount;//更新存款总金额

else OutAmount += -1 \* en.DealAmount;//更新贷款总金额

if (!EmptyQueue(&q[i])) {//判断离开后队列是否为空

GetHead(&q[i], &customer);

OrderInsert(&ev, NewEvent(en.OccurTime + customer.Duration, i, customer.Amount));//求得下一个顾客的离开时间，并插入其离开事件

}

people[i - 1].erase(people[i - 1].begin());//删除蛋仔队列的头节点

m[i-1]--;//队列末端蛋仔索引-1

for (int j = 0; j < people[i - 1].size(); j++) {//将剩余所有蛋仔位置往左移一格

people[i - 1][j].indexInPath --;

people[i - 1][j].movetoIndexInPath(path[i - 1]);

}

}

//银行排队模拟

void Bank\_Simulation() {

srand((unsigned)time(NULL));//随机数发生器的初始化函数

OpenForDay();//初始化操作

while (!ListEmpty(&ev)) {//若事件链表不为空，进行银行模拟

background.draw();//绘制背景

DelFirst(&ev, &en);//删除事件链表首结点并将其赋值给全局变量en

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

if (en.NType == 0)//判断是否为顾客到达事件

CustomerArrived();

else

CustomerDepature();

showBank();

PrintQueue();

timer.Sleep(1000);//调整循环间隔时间

FlushBatchDraw();// 批量绘制

}

}

## 7、其他说明

### 7.1 代码改进

7.1.1 原代码

void CustomerArrived() {

int durtime, intertime, money, i, t;

CustNode e;

Time T1, T2;

CustomerNum++;

intertime = rand() % 5 + 1;//间隔时间

durtime = rand() % 10 + 10;//办理业务时间

money = rand() % 30000 - 15000;//客户要办理的金额

t = en.OccurTime + intertime;

if (en.OccurTime < CloseTime) {//银行尚未关门

T1 = TransTime(en.OccurTime);

T2 = TransTime(t);

OrderInsert(&ev, NewEvent(t, 0, money));

i = ShortestQueue();//最短队列（此处确定了人数最短的队列，而非业务办理最快队列，需要改进）

People ball;

ball.im\_egg1 = im\_egg1;

ball.im\_egg2 = im\_egg2;

ball.im\_egg3 = im\_egg3;

ball.im\_egg4 = im\_egg4;

ball.im\_egg5 = im\_egg5;

ball.im\_egg6 = im\_egg6;

ball.im\_egg7 = im\_egg7;

ball.im\_egg8 = im\_egg8;

ball.im\_egg9 = im\_egg9;

ball.im\_egg10 = im\_egg10;

ball.im\_egg11= im\_egg11;

ball.im\_egg12= im\_egg12;

ball.initiate(path[i-1]);

ball.indexInPath = m[i-1];

ball.movetoIndexInPath(path[i-1]);

people[i-1].push\_back(ball);

m[i-1]++;

e.BeginTime = en.OccurTime;

e.Duration = durtime;

e.Amount = en.DealAmount;

EnQueue(&q[i], e);

if (QueueLength(q[i]) == 1)

OrderInsert(&ev, NewEvent(en.OccurTime + durtime, i, en.DealAmount));

}

else {

// printf("银行关门了！");

}

}

7.1.2 改进代码

void CustomerArrived() {

int durtime, intertime, money, i, j, t, minqueue = 0;

CustNode e;

Time T1, T2;

CustomerNum++;

intertime = rand() % 5 + 1;//间隔时间

durtime = rand() % 10 + 10;//办理业务时间

money = rand() % 30000 - 15000;//客户要办理的金额

t = en.OccurTime + intertime;

if (en.OccurTime < CloseTime) {//银行尚未关门

T1 = TransTime(en.OccurTime);

T2 = TransTime(t);

OrderInsert(&ev, NewEvent(t, 0, money));

for (j = 1; j < numHeight; j++) {//最短队列（遍历所有队列，选择结束时间最早的队列）

if (Queuetime[j] < Queuetime[minqueue]) {

minqueue = j;

}

}

i = minqueue + 1;//窗口号等于下标号+1

People ball;

ball.im\_egg1 = im\_egg1;

ball.im\_egg2 = im\_egg2;

ball.im\_egg3 = im\_egg3;

ball.im\_egg4 = im\_egg4;

ball.im\_egg5 = im\_egg5;

ball.im\_egg6 = im\_egg6;

ball.im\_egg7 = im\_egg7;

ball.im\_egg8 = im\_egg8;

ball.im\_egg9 = im\_egg9;

ball.im\_egg10 = im\_egg10;

ball.im\_egg11 = im\_egg11;

ball.im\_egg12 = im\_egg12;

ball.initiate(path[i - 1]);

ball.indexInPath = m[i - 1];

ball.movetoIndexInPath(path[i - 1]);

people[i - 1].push\_back(ball);

m[i - 1]++;

e.BeginTime = en.OccurTime;

e.Duration = durtime;

e.Amount = en.DealAmount;

EnQueue(&q[i], e);

if (QueueLength(q[i]) == 1) {//插入离开事件，更新队列办理结束时间

OrderInsert(&ev, NewEvent(en.OccurTime + durtime, i, en.DealAmount));

Queuetime[i - 1] = en.OccurTime + durtime;//如果到达时该窗口没人，则该队列办理结束时间等于当前时间+该顾客的办理时间

}

else {

Queuetime[i - 1] += durtime;//如果到达时该窗口有人，则该队列办理结束时间等于原队列结束时间+该顾客的办理时间

}

}

else {

// printf("银行关门了！");

}

}

# 二、答辩记录

1. 答辩者1 刘佳辉:

**问题:** 你们是如何设计驱动事件的?

**回答:** 我们定义了一个事件类型的结构体,作为事件结点,构造了一个事件链表,将事件结点按发生时间的先后顺序插入链表中,每次获取并执行链表的首元结点，就能按时间顺序实现银行一天的业务模拟.

1. 答辩者2 杨婧蕙:

**问题：**你们这个模拟速度有点快，能调吗？

**回答：**能调。操作：Ctrl+F找到timer.Sleep，将括号里的间隔时间调大。

**问题：**顾客来的时候是怎么选择队列的？

**回答：**我们算出了每个队列的长度，然后选择了长度最短的那条队列进行入队。

**建议：**这个排队算法不太智能，应该在顾客到来的时候就知道前面队伍的结束时间，选择结束最早的入队。（我们做出了改进，在其他说明中已标明）

1. 答辩者3孔雯：

**问题：**你们项目的事件类型有哪些？

**回答：**事件类型分别有存钱和取钱、到达和离开两类事件类型。存钱和取钱事件类型用事件节点的DealAmount标记，正数为存款，负数为取款，DealAmount的值为系统随机分配而得。到达和离开事件类型用事件节点的NType标记，NType值为0表示顾客到达事件，值为1到4分别表示顾客从1到4号窗口的离开事件。

# 三、总结

通过本次课程设计的实践，我们更加深入地学习和应用了链表及队列的相关知识，更加直观地认识和掌握了数据结构的定义和运用。在巩固和加深了对数据结构理解的同时，提高了认真思考、深入研究、钻研问题、解决问题的能力和经验。我们三人分工合作，在合作时探索课题，在合作中共同进步，质量与效率兼备，很好地完成了实践。

在实践中，我们也有考虑不周到的地方，比如顾客入队选择队列的问题，我们没有意识到选择最短队列的排队算法存在不合理之处，而在验收时，经过老师的指导才发现这个问题。这是我们在设计和测试过程中没有思考完善，没有全方位考虑所导致的结果。本项目在验收后已经做出相应改进。在今后的课题实践中，我们会吸取这次的教训，反复研讨，反复测试，避免此类问题的发生。

本次课程设计也是一次理论与实践的结合。我们所做的题目很好的融入了课堂上所讲的例子，正可谓学以致用，用以巩固，实践的过程让我们的认识更透彻，记忆更深刻，也让我们发现平时学习的不足，从而加以弥补，更为我们未来对计算机领域其他前沿知识的学习奠定基础。