**课 程 设 计 报 告**

**题目： 基于堆的优先级队列ADT实现及其应用**

**课程名称： 数据结构**

**专业班级： CS1409**

**学 号： U201414808**

**姓 名： 王林**

**指导教师： 李剑军**

**报告日期： 2016.2.14**

**计算机科学与技术学院**

任 务 书

设计内容

传统队列是一种符合先插入的元素必须先删除（FIFO）的处理逻辑，这不总是满足应用要求；很多时候需要优先级高的任务先处理（即后插入的可能先删除）。

1. 基于堆的概念设计优先级队列(Priority Queue)抽象数据类型，至少包含Init\_PriorityQue, Destroy\_PriorityQue, Clear\_PriorityQue，PriorityQue\_Insert, PriorityQue\_DeletMin, PriorityQue\_Empty, PriorityQue\_Full等操作；
2. 选择适当的物理存储结构实现优先级队列ADT;
3. 应用优先级队列ADT设计与实现一个医院门诊医师与病人看诊服务事件仿真程序，使医师服务效率尽量高。

设计要求

（1）仿真事件（如病人到达，病情复杂度/就诊时间，病人离开等）可根据某种概率分布或随机模型生成。

（2）要求对各种算法进行理论分析，同时也对实测结果进行统计分析。测试数据要求有一定规模。

（3）要求界面整洁、美观，操作方便。

参考文献

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

[2] 严蔚敏, 吴伟民, 米宁. 数据结构题集（C语言版）. 北京: 清华大学出版社,1999

[3] Christopher D. Manning等著，王斌译. 信息检索导论，人民邮电出版社，2010

[4]搜索引擎技术揭密：中文分词技术, <http://www.williamlong.info/archives/333.html>

[5] 任丽芸. 搜索引擎中文分词技术研究. 重庆理工大学硕士学位论文，2011

1 引言

* 1. 课题背景与意义

《优先级队列事件仿真模拟》这个课题就目前社会的发展来说具有重要意义，而且对我们的C语言学习来说具有重要意义。

1.1.1 对于当前社会的意义

对于当前社会来说，优先级队列的应用有很多。在很多情况下，并不能按照队列先进先出的规则来完成事件的处理。必须考虑事件的优先级并且按照事件的优先级来排序，来进行处理。比如医院，可以依照病人的病情来设置优先级，从而满足现实情况。

1.1.2 对于我们学习C语言有重要意义

对于学习了一些C语言基础知识的我们来说，通过实践来掌握和巩固所学知识是非常重要的，通过设计这个优先级队列仿真事件模拟，可以更好地加深我们所学的C语言知识。

* 1. 国内外研究状况

目前，由于计算机与网络的发展，优先级队列的研究已经很深入，并且取得了很多研究成果，在各个方面有很多应用。

* 1. 课程设计的主要研究工作

本次课设要求学生独立设计完成一个优先级队列仿真模拟事件的程序，其主要涉及：

1. 队列的相关操作函数：至少包含Init\_PriorityQue, Destroy\_PriorityQue, Clear\_PriorityQue，PriorityQue\_Insert, PriorityQue\_DeletMin, PriorityQue\_Empty, PriorityQue\_Full等操作。
2. 数据在磁盘上以文件形式保存。
3. 有比较良好的人机交互界面，易于操作。

2 系统需求分析与总体设计

2.1 系统需求分析

本系统是基于堆的优先级队列来进行医院就诊事件的仿真模拟，通过随机事件来模拟病人到达医院，在医院就诊，以及就诊结束离开医院。

病人到达就诊时间以及治疗时间可通过随机数来产生，主函数中为一个大的循环，循环直到设置的结束时间标志。期间，要不断的模拟病人到达事件。如果当前医师为工作状态，即正在为病人治疗，则病人进入队列，然后通过堆排序，依据病人的优先级，对病人看病顺序进行适当调整，使满足优先级高的排在队前。

模拟三个事件，病人到达，病人就诊，病人离开。病人到达和离开都要对病人进行重新排序，如果当前处于就诊状态，就要进行排队。系统要求自行模拟处理这三类事件，并且在屏幕上有相应的信息输出提示。

同时，对病人优先级的设定需要通过公式计算得来，以priority=100-（到达时间小时+分钟）+病人病情复杂度+病人可容忍等待时间计算。

而对于病人，一旦超过他的容忍等待时间，病人会选择离开。

并且，可以对已经当前病人以及队列情况进行统计，并在屏幕进行输出。统计当前就诊率。

2.2 系统总体设计

本系统只有一个主要模块，即模拟病人就诊事件。可以分为三部分完成，即仿真模拟病人事件，堆排序，以及队列操作部分。最后有一个文件保存部分。

基于堆的优先级队列仿真模拟事件------医院就诊

队列操作部分

仿真模拟

堆排序

文件保存

**图2-2-1 系统总体设计**

3 系统详细设计

3.1 有关数据结构的定义

1、常量定义：#define TRUE 1、#define FALSE 0、#define OK 1、#define ERROR 0、#define INFEASIBLE -1、#define OVERFLOW -2。#define NUM\_PATIENT 100、#define EXTRA\_PATIENT 50。2、变量类型定义：typedef int Status 整型，typedef int Boolean 布尔型，typedef int Time时间、typedef int Priority优先级。

3、控制台相关变量： #define SCR\_ROW 25、#define SCR\_COL 80。CHAR\_INFO \*gp\_buff\_menubar\_info=NULL、CHAR\_INFO \*gp\_buff\_staeBar\_info=NULL、HANDLE gh\_std\_out、HANDLE gh\_std\_in。

4、变量：i作为计数器，屏幕坐标：pos、pos\_1、pos\_Queue、size；时间变量：intervaltime,nexttime,labletime;

5、储存结构采用队列，队列结构为：typedef struct LinkQueue；病人信息结构：typedef struct patient；

队列中包括队列指针，队列长度，队列容量，以及判断是否处于诊断状态的变量。病人结构包括到达、开始、治疗、离开时间和优先级。

3.2 主要算法设计

1、主函数算法设计



**图3-2-1 主函数算法设计**

2．创建队列

算法名称：InitQueue

算法输入：队列结构地址&Q

算法输出： 创建成功返回OK，否则返回ERROR

算法思想：定义队列结构Q,对成员queue分配存储空间，给予线性存储结构

算法描述：

用malloc函数给Q.queue分配空间

如果分配失败，返回ERROR

q->queuelength = 0;返回OK；

时间复杂度与空间复杂度分析：S(n)=O(n)，线性阶，T(n)=O(1)，常量阶

3．销毁队列

算法名称：DestroyQueue

算法输入：队列结构地址&Q

算法输出：销毁成功返回OK，否则返回ERROR

算法思想：释放Q.queue的存储空间

算法描述：

若队列存在，调用free函数释放Q.queue的存储空间

Q.queue=NULL，返回OK

时间复杂度与空间复杂度分析：S(n)=O(1)常量阶，T(n)=O(1)，常量阶

4. 清空队列

算法名称：ClearQueue

算法输入：队列结构地址&Q

算法输出：清空成功返回OK，否则返回ERROR

算法思想：让Q.queuelength置为0

算法描述：若队列存在，Q.queuelength=0,否则返回ERROR

时间复杂度与空间复杂度分析：S(n)=O(1)，T(n)=O(1)

5. 判断队列是否为空

算法名称：QueueEmpty

算法输入：队列结构Q

算法输出：为空返回TRUE，否则返回FALSE

算法思想：通过Q.queuelength是为0来判断队列是否为空

算法描述：

若该队列存在，直接返回Q.queuelength==0的结果；

否则返回ERROR

时间复杂度与空间复杂度分析：S(n)=O(1)，T(n)=O(1)

6. 输出队列长度

算法名称：QueueLength

算法输入：队列结构Q

算法输出： 队列存在返回队列长度，否则返回ERROR

算法思想：直接返回Q.queuelength的值

算法描述：

若队列存在，直接返回Q。queuelength

否则，返回FALSE

时间复杂度与空间复杂度分析：S(n)=O(1)，T(n)=O(1)

7. 删除队首元素

算法名称：QueueDeletMaxPriority

算法输入：队列结构地址&Q，变量地址&patient

算法输出： 删除成功返回OK，否则返回ERROR

算法思想：直接赋值给patient，指针返回

算法描述：

若队列存在且不为空，将Q.queue[0]的数据域赋给patient;

qit元素前移

返回OK；否则返回ERROR

时间复杂度与空间复杂度分析：S(n)=O(1)，T(n)=O(n).

4 系统实现与测试

4.1 系统实现

4.1.1 实现目标

启动仿真事件模拟后，系统自动进入模拟状态。通过屏幕提示当前的状态——等待病人到来，就诊。同时屏幕上输出信息提示当前事件——病人到达事件，病人离开事件，正在治疗事件。屏幕上有系统当前时间提示，以及下一位病人到达时间提示。屏幕下方显示当前队列情况。

4.1.2 系统环境

本系统操作环境是Windows操作系统，是在CodeBlocks下用C语言编写源程序，再进行编译、链接和运行的。

4.1.3 数据定义

【头文件变量定义】

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#include<windows.h>

#include<io.h>

//常量定义

#define TRUE 1

#define FALSE 0

#define OK 1

#define ERROR 0

#define INFEASIBLE -1

#define OVERFLOW -2

//初始队列长度

#define NUM\_PATIENT 100

//每次扩容的病人数目

#define EXTRA\_PATIENT 50

//变量类型定义

typedef int Status; //整型

typedef int Boolean; // 布尔型

typedef int Time; //储存时间

typedef int Priority; //储存优先级

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*屏幕控制台设定\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#define SCR\_ROW 25 //屏幕行数

#define SCR\_COL 80 //屏幕列数

CHAR\_INFO \*gp\_buff\_menubar\_info=NULL;//存放菜单条屏幕区字符信息的缓冲区

CHAR\_INFO \*gp\_buff\_staeBar\_info=NULL;//存放状态条屏幕区字符信息的缓冲区

HANDLE gh\_std\_out; //标准输出设备句柄

HANDLE gh\_std\_in; //标准输入设备句柄

char \*gp\_sys\_name="优先级队列仿真模拟事件";

BOOL HelloSys();//欢迎界面

//病人信息结构类型

typedef struct patient

{

Time arrivedTime; //储存病人到达时间

Time starttime; //储存病人开始诊断时间

Time tratetime; //储存病人治疗时间

Time leavingTime; //储存病人离开时间

Priority priority; //储存病人优先级

} Patient;

//队列结构类型

typedef struct LinkQueue

{

Patient \*queue;

int queuelength; //队列长度

int Content; //队列容量

Boolean isTreating; //是否正在就诊

} LinkQueue;

4.1.4 函数设计

//-------------------------优先级队列函数------------------------

Status InitQueue(LinkQueue \*q); //创建队列

Status DestroyQueue(LinkQueue \*q);//销毁队列

int QueueLength(LinkQueue q); //计算队列长度

Status QueueInsert(LinkQueue \*q, Patient p);//入队列

Status QueueDeletMaxPriority(LinkQueue \*q, Patient \*p);//删除优先级最高元素

Status QueueFull(LinkQueue q); //判断是否满队列

Status QueueEmpty(LinkQueue q);//判断队列是否为空

//--------------------------堆排序函数------------------------

Status HeapAdjust(LinkQueue \*q, int top, int queuesize);//调整为大顶堆

Status HeapSort(LinkQueue \*q); //堆排序实现

//--------------------------仿真模拟函数-------------------------

Time NowTime();//获取系统当前函数

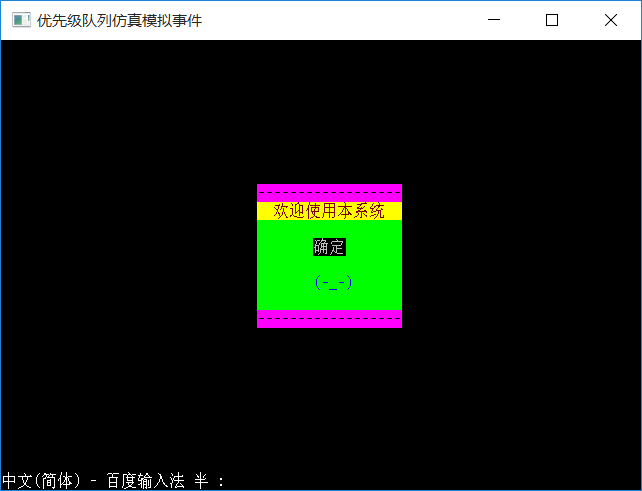
Patient ComePatient();//新的病人来到

Time GetIntervaltime();//设置间隔时间

Status ShowQueue(LinkQueue q);//显示当前队列

4.2 系统测试

1、欢迎界面测试



**图4-2-1 欢迎界面测试**

如图，欢迎界面正常。

2、等待病人部分模拟



**图4-2-2 等待病人部分测试**

如图，等待病人部分正常。

3、病人到达事件



**图4-2-3 病人到达事件**

如图，病人到达事件部分测试正常。

1. 病人离开事件



**图4-2-4 病人离开事件模拟部分**

如图，病人离开事件测试成功。

1. 总体检测



**图4-2-5 总体队列检测**

总体正常，且可以图像显示当前队列。

1. 信息记录测试

本部分主要测试对就诊病人的信息的正确性测试

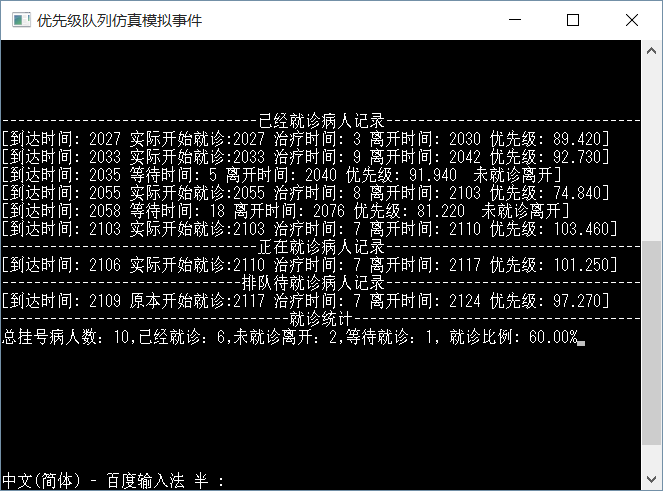




**图 4-2-6 信息记录检测**

如图，系统正常运转且输出就诊病人信息。

1. 统计部分测试



如图，系统正确统计出当前队列以及就诊的信息。

经过检测，系统可以正常运行，且较好的对医院就诊事件进行仿真模拟。

5 总结与展望

5.1 全文总结

本次课程设计实验，先学习了C语言，重点在与指针和队列的操作，队列操作包括 ，销毁，清空，插入，删除等。然后是对队列元素进行堆排序，基于堆实现优先级队列，并应用优先级队列来对医院就诊事件进行仿真模拟。

在具体工作中：

设计本操纵系统。包括界面设设计，队列操作设计，数据结构的设计，系统功能的设计，功能函数的设计，各函数之间的调用设计。

代码编写。首先是屏幕输出界面的编写，通过控制台函数的调用，画出界面。然后编写主体代码，通过查阅资料，参考书本，完善所有程序。

程序的运行与调试。调试系统，检测是否有BUG。

系统界面的美化。通过控制台函数和‘\*’来美化系统界面。

5.2 工作展望

在今后的工作和研究中，要通过实例学习，将优先级队列应用到具体问题中去，从而加深对这一类问题的学习和理解。

6 体会

本次《基于堆的优先级队列仿真模拟事件》的课程设计试验中，代码编写量不大，共有500多行，但是难度在于有关队列的操作函数实现以及对医院就诊事件进行仿真模拟。

1. 一定要先做整体规划设计，对整个系统的各个方面做出明确设计。
2. 在读文件时，读后一定要判断是否到达文件尾部，避免最后一条信息被读入两次。
3. 总之，通过这次课程设计，通过大量代码的编写，增加了自己的经验，提高了自己寻找BUG的能力，减少了出错的几率。

附录

以下是源代码文件整理

------------------------------------------头文件----------------------------------------------

【头文件】

#ifndef PRIORITY\_QUEUE\_H\_INCLUDED

#define PRIORITY\_QUEUE\_H\_INCLUDED

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#include<windows.h>

#include<io.h>

//常量定义

#define TRUE 1

#define FALSE 0

#define OK 1

#define ERROR 0

#define INFEASIBLE -1

#define OVERFLOW -2

//初始队列长度

#define NUM\_PATIENT 100

//每次扩容的病人数目

#define EXTRA\_PATIENT 50

//变量类型定义

typedef int Status; //整型

typedef int Boolean; // 布尔型

typedef int Time; //储存时间

typedef float Priority; //储存优先级

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*屏幕控制台设定\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#define SCR\_ROW 50 //屏幕行数

#define SCR\_COL 80 //屏幕列数

CHAR\_INFO \*gp\_buff\_menubar\_info=NULL;//存放菜单条屏幕区字符信息的缓冲区

CHAR\_INFO \*gp\_buff\_staeBar\_info=NULL;//存放状态条屏幕区字符信息的缓冲区

HANDLE gh\_std\_out; //标准输出设备句柄

HANDLE gh\_std\_in; //标准输入设备句柄

char \*gp\_sys\_name="优先级队列仿真模拟事件";

BOOL HelloSys();//欢迎界面

//病人信息结构类型

typedef struct patient

{

Time arrivedTime; //储存病人到达时间

Time starttime; //储存病人开始诊断时间

Time tratetime; //储存病人治疗时间

Time leavingTime; //储存病人离开时间

Priority priority; //储存病人优先级

Time waittime; //病人能够等待时间

Time toleranttime; //病人容忍的时间

} Patient;

//队列结构类型

typedef struct LinkQueue

{

Patient \*queue;

int queuelength; //队列长度

int Content; //队列容量

Boolean isTreating; //是否正在就诊

} LinkQueue;

//-------------------------优先级队列函数------------------------

Status InitQueue(LinkQueue \*q); //创建队列

Status DestroyQueue(LinkQueue \*q);//销毁队列

int QueueLength(LinkQueue q); //计算队列长度

Status QueueInsert(LinkQueue \*q, Patient p);//入队列

Status QueueDeletMaxPriority(LinkQueue \*q, Patient \*p);//删除优先级最高元素

Status QueueFull(LinkQueue q); //判断是否满队列

Status QueueEmpty(LinkQueue q);//判断队列是否为空

//--------------------------堆排序函数------------------------

Status HeapAdjust(LinkQueue \*q, int top, int queuesize);//调整为大顶堆

Status HeapSort(LinkQueue \*q); //堆排序实现

//--------------------------仿真模拟函数-------------------------

Time NowTime();//获取系统当前函数

Patient ComePatient();//新的病人来到

Time GetIntervaltime();//设置间隔时间

Status ShowQueue(LinkQueue q);//显示当前队列

Status ShowPatientInfo(LinkQueue q);//显示当前病人信息

Priority GetPriority(Time arrivetime,Status lablenum,Time waittime); //设置病人优先级

#endif

// PRIORITY\_QUEUE\_H\_INCLUDED

-----------------------------------------main.c文件-----------------------------------------

【main.c文件】

#include "Priority\_Queue.h"

int main(void)

{

COORD size= {SCR\_COL,SCR\_ROW}; //窗口缓冲区大小

COORD pos={25,8};

gh\_std\_in=GetStdHandle(STD\_INPUT\_HANDLE); //获取标准输入设备句柄

gh\_std\_out=GetStdHandle(STD\_OUTPUT\_HANDLE); //获取标准输出设备句柄

SetConsoleTitle(gp\_sys\_name); //设置窗口标题

SetConsoleScreenBufferSize(gh\_std\_out,size); //设置窗口缓冲区大小

Patient patient;

Patient \*pPatient = (Patient \*) malloc(sizeof(Patient));

LinkQueue Q; //声明就诊队列Q

Q.queue = NULL;

Status label=1;//标记第一次运行

int i=0; //计数器

FILE \*fp;

Time intervaltime,nexttime,labletime; //时间定义

HelloSys();//产生欢迎界面

system("CLS");//清屏

//构造优先级队列

InitQueue(&Q);

ShowQueue(Q);

while(NowTime()<=2359)

{

if(Q.queuelength)

{

if(Q.queue[0].leavingTime>NowTime())

{

Q.isTreating=TRUE;

}

}

else Q.isTreating=FALSE;

SetConsoleCursorPosition(gh\_std\_out,pos);

if(Q.isTreating) printf("----就诊中...\n");

else printf("-----等待病人到来----\n");

Sleep(2000);

if(label==1)//产生第一个病人

{

intervaltime=GetIntervaltime();

nexttime=NowTime()+intervaltime;

if((nexttime%100)>=60)

{

nexttime=(nexttime%100-60)+(nexttime/100+1)\*100;

}

SetConsoleCursorPosition(gh\_std\_out,pos);

printf("-----下一个病人%d分钟后到来-----\n",intervaltime);

while(NowTime()<nexttime);

patient=ComePatient();

if (QueueInsert(&Q,patient)==OK)

{

ShowQueue(Q);

SetConsoleCursorPosition(gh\_std\_out,pos);

printf("[病人达到事件]----排队成功\n");

Sleep(2000);

}

}

else

{

if(Q.queuelength)

{

while(NowTime()<Q.queue[0].arrivedTime);

}

}

while(NowTime()<Q.queue[0].leavingTime)

{

Q.isTreating=TRUE;

intervaltime=GetIntervaltime();

nexttime=NowTime()+intervaltime;

if((nexttime%100)>=60)

{

nexttime=(nexttime%100-60)+(nexttime/100+1)\*100;

}

SetConsoleCursorPosition(gh\_std\_out,pos);

printf("----下一个病人%d分钟后到来-----\n",intervaltime);

if(nexttime<Q.queue[0].leavingTime)

{

while(NowTime()<nexttime);

patient=ComePatient();

if(QueueInsert(&Q,patient))

{

ShowQueue(Q);

SetConsoleCursorPosition(gh\_std\_out,pos);

printf("[病人达到事件]----取号排队成功\n");

Sleep(2000);

}

}

else break;

}

while(NowTime()<Q.queue[0].leavingTime);//到达当前病人离开时间点

if(QueueDeletMaxPriority(&Q, pPatient)==OK)

{

SetConsoleCursorPosition(gh\_std\_out,pos);

labletime=pPatient->leavingTime;

printf("\n\n病人离开事件--[开始时间：[%d] 优先级:[%f] 离去时间：[%d] 就诊时间：[%d] ]\n",

pPatient->starttime,pPatient->priority,pPatient->leavingTime,

pPatient->tratetime);

Sleep(2000);

//文件读写模块

if((fp=fopen("test.dat", "ab+")) == NULL)

{

exit(-1);

}

fwrite(pPatient,sizeof(Patient),1,fp);

fclose(fp);

HeapSort(&Q);//按照优先级对顺序调节

}

//病人时间更新

for(i=0; i<Q.queuelength; i++)

{

Q.queue[i].starttime=(labletime<Q.queue[i].starttime?Q.queue[i].starttime:labletime);

Q.queue[i].leavingTime=Q.queue[i].starttime+Q.queue[i].tratetime;

if((Q.queue[i].leavingTime%100)>=60)

{

Q.queue[i].leavingTime=(Q.queue[i].leavingTime%100-60)+

(Q.queue[i].leavingTime/100+1)\*100;

}

labletime=Q.queue[i].leavingTime;

}

//显示队列

ShowQueue(Q);

//检查当前病人是否还在队列中等待

while(Q.queue[0].toleranttime<NowTime())

{

if(QueueDeletMaxPriority(&Q, pPatient)==OK)

{

SetConsoleCursorPosition(gh\_std\_out,pos);

pPatient->leavingTime=pPatient->toleranttime;

pPatient->starttime=0;

labletime=NowTime();

printf("\n\n病人未就诊离开事件--[达到时间：[%d] 优先级:[%f] 离去时间[%d] 等待时间：[%d] ]\n",

pPatient->arrivedTime,pPatient->priority,pPatient->leavingTime,

pPatient->waittime);

Sleep(2000);

//文件读写模块

if((fp=fopen("test.dat", "ab+")) == NULL)

{

exit(-1);

}

fwrite(pPatient,sizeof(Patient),1,fp);

fclose(fp);

HeapSort(&Q);//按照优先级对顺序调节

//更新时间

for(i=0; i<Q.queuelength; i++)

{

Q.queue[i].starttime=labletime;

Q.queue[i].leavingTime=Q.queue[i].starttime+Q.queue[i].tratetime;

if((Q.queue[i].leavingTime%100)>=60)

{

Q.queue[i].leavingTime=(Q.queue[i].leavingTime%100-60)+

(Q.queue[i].leavingTime/100+1)\*100;

}

labletime=Q.queue[i].leavingTime;

}

//显示队列

ShowQueue(Q);

}

else break;

}

//继续产生上一个病人

while(NowTime()<nexttime);

patient=ComePatient();

if(QueueInsert(&Q,patient))

{

ShowQueue(Q);

SetConsoleCursorPosition(gh\_std\_out,pos);

printf("[病人达到事件]----取号排队成功\n");

}

label++;

}

return 0;

}

/\*----------------------队列操作函数实现-------------------------\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

函数名称：InitQueue

函数功能：创建空队列

函数参数：队列指针q

函数返回值：创建成功，返回OK

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

Status InitQueue(LinkQueue \*q)

{

q->queue = (Patient \*) malloc(sizeof(Patient) \* NUM\_PATIENT);//分配空间

q->Content = NUM\_PATIENT;

q->queuelength = 0;

q->isTreating = FALSE;

return OK;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

函数名称：DestroyQueue

函数功能：销毁队列

函数参数：队列指针q

函数返回值：销毁成功，返回OK

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

Status DestroyQueue(LinkQueue \*q)

{

free(q);

q->queue = NULL;

q->Content = 0;

return OK;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

函数名称：ClearQueue

函数功能：清空队列

函数参数：队列指针q

函数返回值：清空成功，返回OK

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

Status ClearQueue(LinkQueue \*q)

{

q->queuelength = 0;

return OK;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

函数名称：QueueLength

函数功能：计算队列长度

函数参数：队列q

函数返回值：队列长度

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int QueueLength(LinkQueue q)

{

return q.queuelength;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

函数名称：QueueInsert

函数功能：新元素入队

函数参数：队列q，新元素patient

函数返回值：插入成功，返回OK

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

Status QueueInsert(LinkQueue \*q, Patient patient)

{

//是否满队列

if (QueueFull(\*q) == TRUE)

{

q->queue=(Patient \*)realloc(q->queue,sizeof(Patient)\*(q->Content+ EXTRA\_PATIENT));

if (q->queue == NULL) return ERROR;

q->Content += EXTRA\_PATIENT;

}

//插入队列

q->queue[q->queuelength] = patient;

//队列长度增加

q->queuelength++;

return OK;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

函数名称：QueueDeletMaxPriority

函数功能：删除当前队首

函数参数：队列指针q，指向病人的指针

函数返回值：删除成功，返回OK

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

Status QueueDeletMaxPriority(LinkQueue \*q, Patient \*patient)

{

int i;

//当前队列是否为空

if (QueueEmpty(\*q) == TRUE) return ERROR;

\*patient = q->queue[0];//删除队首

for (i = 1; i < q->queuelength; i++) //其他元素前移

{

q->queue[i - 1] = q->queue[i];

}

//队列长度减一

q->queuelength--;

return OK;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

函数名称：QueueEmpty

函数功能：判断队列是否为空

函数参数：队列q

函数返回值：为空返回1，否则返回0

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

Status QueueEmpty(LinkQueue q)

{

return (q.queuelength==0);

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

函数名称：QueueFull

函数功能：判断队列是否满队列

函数参数：队列q

函数返回值：满队列返回1，否则返回0

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

Status QueueFull(LinkQueue q)

{

return (q.queuelength == q.Content);

}

/\*-----------------------堆排序函数实现--------------------------\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

函数名称：HeapAdjust

函数功能：调整，使满足大顶堆的形式

函数参数：队列指针q，整形变量top,队列长度变量

函数返回值：调整成功，返回OK

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

Status HeapAdjust(LinkQueue \*q, int top, int queuelength)

{

int i;

Patient temp = q->queue[top];

if (q->queue == NULL)

return ERROR;

for (i = 2 \* top + 1; i < queuelength; top = i, i = 2 \* i + 1)//进行筛选

{

if (i+1<queuelength&&q->queue[i].priority>q->queue[i + 1].priority)

{

i++;

}

if (temp.priority <= q->queue[i].priority)

{

break;

}

q->queue[top] = q->queue[i];

}

q->queue[top] = temp;

return OK;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

函数名称：HeapSort

函数功能实现堆排序：

函数参数：队列指针q

函数返回值：排序成功，返回OK

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

Status HeapSort(LinkQueue \*q)

{

int i;

Patient temp;

if (q->queue == NULL ||QueueEmpty(\*q) == TRUE)

return ERROR;

for (i =((q->queuelength - 1) / 2); i >= 0; i--)

{

HeapAdjust(q,i, q->queuelength);

}

for (i = q->queuelength - 1; i > 0; i--)

{

temp = q->queue[0];

q->queue[0] = q->queue[i];

q->queue[i] = temp;

HeapAdjust(q, 0, i);//重新调整为大顶堆

}

return OK;

}

/\*--------------------仿真事件模拟函数实现----------------------\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

函数名称：ComePatient

函数功能：模拟来了一位病人

函数参数：无

函数返回值：病人

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

Patient ComePatient()

{

Patient patient;

Time lablenum;

//系统时间作为随机数的种子

srand((unsigned int) time(NULL));

//取治疗时间为[3,9]分钟

patient.tratetime=(int)(rand()%7+3);

//记录病人到达时间为系统当前时间

patient.arrivedTime = NowTime();

//记录病人离开时间为到达时间+治疗时间

patient.leavingTime = patient.arrivedTime + patient.tratetime;

if((patient.leavingTime%100)>=60)

{

patient.leavingTime=(patient.leavingTime%100-60)+

(patient.leavingTime/100+1)\*100;

}

//随机产生lablenum

lablenum=(int)(rand()%30+20);

//随机产生病人容忍等待时间

patient.waittime=(int)(rand()%20+5);

//计算得到病人优先级

patient.priority=GetPriority(patient.arrivedTime,lablenum,patient.waittime);

if((patient.toleranttime%100)>=60)

　　{

　　 patient.toleranttime=(patient.toleranttime%100-60)+

(patient.toleranttime/100+1)\*100;

　　　}

//计算病人能够等待的时刻

patient.toleranttime=patient.arrivedTime+patient.waittime;

//默认的开始治疗时间为病人到达诊室进入队列的时刻

patient.starttime=patient.arrivedTime;

return patient;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

函数名称：NowTime

函数功能：获取系统当前时间

函数参数：无

函数返回值：当前时间

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

Time NowTime()

{

time\_t nowtime;

struct tm \*timeinfo;

Time currenttime;

time(&nowtime); //获取系统当前时间

timeinfo = localtime(&nowtime);

//取小时和分钟的时间

currenttime =timeinfo->tm\_hour\*100 + timeinfo->tm\_min;

return currenttime;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

函数名称：GetIntervaltime

函数功能：模拟产生一个间隔时间段

函数参数：无

函数返回值：间隔时间

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

Time GetIntervaltime()

{

int \_time;

srand((unsigned int) time(NULL));

//病人来的时间间隔为[2,6]

\_time = (Time) (rand() %5+2);

return \_time;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

函数名称： HelloSys

函数功能： 欢迎界面

函数参数： 无

函数返回值:TRUE

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

BOOL HelloSys()

{

int m;

char mark[2]= {'0'}; //存放选项是否标记

DWORD a;

char\* Menu[] = //提示信息

{

"------------------",

"欢迎使用本系统",

"确定",

" (-\_-)",

"------------------"

};

COORD Head\_Line = {32,9}; //标题行起始位置

COORD U= {32,8},B= {32,15},Ok= {39,11};

COORD pos= {32,10}; //欢迎语

COORD pos\_2;

COORD cursor; //鼠标位置

WORD att\_headline =BACKGROUND\_GREEN|BACKGROUND\_RED|BACKGROUND\_INTENSITY

|FOREGROUND\_RED; //标题行颜色：亮黄背景和蓝色前景

//提示信息背景色:暗紫

WORD att\_background =BACKGROUND\_BLUE|BACKGROUND\_RED|BACKGROUND\_INTENSITY;

WORD att\_end =BACKGROUND\_GREEN|BACKGROUND\_INTENSITY

|FOREGROUND\_BLUE|FOREGROUND\_INTENSITY;

//欢迎语字符属性：亮绿背景景,亮蓝前景

WORD att\_choice =FOREGROUND\_BLUE|FOREGROUND\_GREEN|FOREGROUND\_RED;

//选择区域颜色：亮黄

WORD att\_choice\_on =FOREGROUND\_BLUE|FOREGROUND\_INTENSITY;

//鼠标移至"确定"处颜色

DWORD res;

INPUT\_RECORD inRec;

BOOL bRet =TRUE;

FillConsoleOutputAttribute(gh\_std\_out,att\_headline,18,Head\_Line,&a); //设置标题行

for(m=0; m<5; m++)

{

FillConsoleOutputAttribute(gh\_std\_out,att\_end,18,pos,&a);

if(pos.Y==11)

{

pos\_2.X=pos.X+7;

pos\_2.Y=pos.Y;

WriteConsoleOutputCharacter(gh\_std\_out,Menu[2],strlen(Menu[2]),pos\_2,&a);

}

if(pos.Y==13)

{

pos\_2.X=pos.X+6;

pos\_2.Y=pos.Y;

WriteConsoleOutputCharacter(gh\_std\_out,Menu[3],strlen(Menu[3]),pos\_2,&a);

}

pos.Y++;

}

FillConsoleOutputAttribute(gh\_std\_out,att\_background,18,U,&a);

WriteConsoleOutputCharacter(gh\_std\_out,Menu[0],18,U,&a);

FillConsoleOutputAttribute(gh\_std\_out,att\_background,18,B,&a);

WriteConsoleOutputCharacter(gh\_std\_out,Menu[4],18,B,&a);

Head\_Line.X+=2;

WriteConsoleOutputCharacter(gh\_std\_out,Menu[1],strlen(Menu[1]),Head\_Line,&a);

while(bRet)

{

ReadConsoleInput(gh\_std\_in,&inRec,1,&res); //从控制台输入缓冲区中读一条记录

if(inRec.EventType==MOUSE\_EVENT)

{

/\*如果事件由按键产生\*/

cursor =inRec.Event.MouseEvent.dwMousePosition;

//如果鼠标在“是”处且“是”未被标记

if(cursor.Y==Ok.Y&&cursor.X<=Ok.X+4&&cursor.X>=Ok.X&&mark[0]=='0')

{

/\*改变“确定”属性\*/

FillConsoleOutputAttribute(gh\_std\_out,att\_choice\_on,strlen(Menu[2]),Ok,&a);

WriteConsoleOutputCharacter(gh\_std\_out,Menu[2],strlen(Menu[2]),Ok,&a);

mark[0]='1';

}

//如果鼠标不在“是”处且“是”被标记

if((cursor.Y!=Ok.Y||cursor.X<Ok.X||cursor.X>Ok.X+4)&&mark[0]=='1')

{

/\*改变“确定”属性\*/

mark[0]='0';

FillConsoleOutputAttribute(gh\_std\_out,att\_choice,strlen(Menu[2]),Ok,&a);

WriteConsoleOutputCharacter(gh\_std\_out,Menu[2],strlen(Menu[2]),Ok,&a);

}

//如果鼠标在“是”处且按下鼠标左键

if(cursor.Y==Ok.Y&&cursor.X<=Ok.X+2&&cursor.X>=Ok.X&&

inRec.Event.MouseEvent.dwButtonState==FROM\_LEFT\_1ST\_BUTTON\_PRESSED)

{

return TRUE;

}

}

}

return TRUE;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

函数名称： ShowQueue

函数功能： 显示当前队列

函数参数： 队列q

函数返回值:TRUE

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

Status ShowQueue(LinkQueue q)

{

system("CLS");//清屏

int i=0; //计数器

Time currenttime=NowTime();

WORD att\_headline =BACKGROUND\_GREEN|BACKGROUND\_RED|BACKGROUND\_INTENSITY

|FOREGROUND\_BLUE; //标题行颜色：亮黄背景和蓝色前景

WORD att\_queue =BACKGROUND\_GREEN|BACKGROUND\_INTENSITY

|FOREGROUND\_BLUE|FOREGROUND\_INTENSITY;

WORD att\_first=BACKGROUND\_RED|BACKGROUND\_INTENSITY;

COORD Head\_Line ={32,5};

COORD Time\_Line={55,0};

COORD pos\_Queue={5,13};

COORD pos={5,11};

DWORD a;

char \*headline="Start The Simulation";

char \*str=" ";

//设置标题行

FillConsoleOutputAttribute(gh\_std\_out,att\_headline,strlen(headline),Head\_Line,&a);

WriteConsoleOutputCharacter(gh\_std\_out,headline,strlen(headline),Head\_Line,&a);

SetConsoleCursorPosition(gh\_std\_out,Time\_Line);

printf("系统上次更新时间--[%d]",currenttime);

if(q.queuelength)

{

for(i=0; i<q.queuelength; i++)

{

if(i==0)

{

printf("\a");

FillConsoleOutputAttribute(gh\_std\_out,att\_first,2,pos\_Queue,&a);

WriteConsoleOutputCharacter(gh\_std\_out,str,2,pos\_Queue,&a);

SetConsoleCursorPosition(gh\_std\_out,pos);

printf("[---正在就诊---]");

pos\_Queue.X=pos\_Queue.X+5;

continue;

}

FillConsoleOutputAttribute(gh\_std\_out,att\_queue,2,pos\_Queue,&a);

WriteConsoleOutputCharacter(gh\_std\_out,str,2,pos\_Queue,&a);

pos\_Queue.X=pos\_Queue.X+5;

if(pos\_Queue.X>=79)

{

pos\_Queue.Y++;

pos\_Queue.X=0;

}

}

}

else

{

SetConsoleCursorPosition(gh\_std\_out,pos);

printf("---等待病人到来---");

}

ShowPatientInfo(q);

return OK;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

函数名称： ShowPatientInfo

函数功能： 显示当前排队病人信息

函数参数： 队列q

函数返回值:TRUE

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

Status ShowPatientInfo(LinkQueue q)

{

int i;

FILE \*fpout;

COORD pos={0,26};

Patient \*PatientNode;

Status sumcure=0,sumpatient=0,sumquit=0;

PatientNode=(Patient \*)malloc(sizeof(Patient));

if((fpout=fopen("test.dat","ab+"))==NULL)

{

printf("病人信息文件创建失败！\n");

exit(-1);

}

SetConsoleCursorPosition(gh\_std\_out,pos);

for(i=0;i<32;i++) printf(("-"));

printf("已经就诊病人记录");

for(i=0;i<32;i++) printf(("-"));

while(!feof(fpout))

{

fread(PatientNode,sizeof(Patient),1,fpout);

if(!feof(fpout))

{

sumcure++;//已经就诊病人数目

if(PatientNode->starttime)

{

printf("[到达时间: %d 实际开始就诊:%d 治疗时间: %d 离开时间: %d 优先级: %2.3f]\n",PatientNode->arrivedTime,

PatientNode->starttime,PatientNode->tratetime,PatientNode->leavingTime,PatientNode->priority);

}

else

{

sumquit++;//离开病人数目

printf("[到达时间: %d 等待时间: %d 离开时间: %d 优先级: %2.3f 未就诊离开]\n",PatientNode->arrivedTime,

PatientNode->waittime,PatientNode->leavingTime,PatientNode->priority);

}

}

}

fclose(fpout);

if(q.queuelength)

{

for(i=0; i<32; i++) printf(("-"));

printf("正在就诊病人记录");

for(i=0; i<32; i++) printf(("-"));

sumpatient++;

printf("[到达时间: %d 实际开始就诊:%d 治疗时间: %d 离开时间: %d 优先级: %2.3f]\n",q.queue[0].arrivedTime,

q.queue[0].starttime,q.queue[0].tratetime,q.queue[0].leavingTime,q.queue[0].priority);

for(i=0; i<30; i++) printf(("-"));

printf("排队待就诊病人记录");

for(i=0; i<32; i++) printf(("-"));

for(i=1; i<q.queuelength; i++)

{

sumpatient++;

printf("[到达时间: %d 原本开始就诊:%d 治疗时间: %d 离开时间: %d 优先级: %2.3f]\n",q.queue[i].arrivedTime,

q.queue[i].starttime,q.queue[i].tratetime,q.queue[i].leavingTime,q.queue[i].priority);

}

sumpatient=sumpatient+sumcure+sumquit;//病人总数

for(i=0; i<36; i++) printf(("-"));

printf("就诊统计");

for(i=0; i<36; i++) printf(("-"));

printf("总挂号病人数：%d,已经就诊：%d,未就诊离开：%d,等待就诊：%d, 就诊比例: %1.2f",

sumpatient,sumcure,sumquit,sumpatient-sumcure-sumquit-1,((float)sumcure)/sumpatient);

}

else

{

printf("等待病人就诊\n");

}

return OK;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

函数名称： GetPriority

函数功能： 设置病人优先级

函数参数： 优先级priority

函数返回值:TRUE

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

Priority GetPriority(Time arrivetime,Status lablenum,Time waittime)

{

return (100-(arrivetime/100+arrivetime%100)\*0.49)+lablenum\*0.38+waittime\*0.13;

}----------------------------------------------结束---------------------------------------------