

**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： 数据结构课程设计**

**专业班级：计算机科学与技术1409班**

**学 号： U201414797**

**姓 名： 张丹朱**

**指导教师： 魏巍**

**报告日期： 2016年2月27日**

**计算机科学与技术学院**

# 任　务　书

**设计目的：**优先级队列在计算机系统与技术中应用广泛而深入。本设计使学生掌握堆作为优先级队列的数据表示方法，物理存储结构与相关算法，从而为后续学习，研究与实践奠定良好的数据结构基础，提高数据结构方法与技术的应用实践能力。

**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] Mark Allen Weiss.Data Structures and Algorithm Analysis in C, 机械工业出版社，2010, 177-192

**目　　录**

[**任务书** I](#_Toc444867444)

[**1　　　引言** 1](#_Toc444867445)

[1.1　　课题背景与意义 1](#_Toc444867446)

[1.2　　问题描述与技术现状分析 1](#_Toc444867447)

[**2　　　系统需求分析与总体设计** 2](#_Toc444867448)

[2.1　　系统需求分析 2](#_Toc444867449)

[2.2　　系统总体设计 2](#_Toc444867450)

[**3　　　系统详细设计** 3](#_Toc444867451)

[3.1　　有关数据结构的定义 3](#_Toc444867452)

[3.2　　主要算法设计 4](#_Toc444867453)

[3.2.1　　ADT部分 4](#_Toc444867454)

[3.2.2　　模拟测试部分 7](#_Toc444867455)

[**4　　　系统实现与测试** 11](#_Toc444867456)

[4.1　　系统实现 11](#_Toc444867457)

[4.2　　系统测试 12](#_Toc444867458)

[**5　　　总结与展望** 16](#_Toc444867459)

[5.1　　全文总结 16](#_Toc444867460)

[5.2　　工作展望 16](#_Toc444867461)

[**6　　　体会** 17](#_Toc444867462)

[**参考文献** 18](#_Toc444867463)

[**附录　1** 19](#_Toc444867464)

[**附录　2** 25](#_Toc444867465)

# 1　引言

## 1.1　课题背景与意义

数据结构是计算机科学技术、信息安全与物联网等专业的一门重要专业基础课，牢固掌握数据结构的基础知识，熟练地运用数据结构的思想与技术方法解决实际应用问题是是本课程学习的基本任务与目标。而课程设计是实现这一学习目标的重要环节和组成部分。通过课程设计的训练，使学生加深对数据结构知识的理解，牢固掌握其应用方法，并合理灵活地解决一定实际问题，增强和提高综合分析问题与解决问题的能力。

## 1.2　问题描述与技术现状分析

问题描述：选择适当的物理存储结构实现优先级队列ADT并应用优先级队列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。

技术现状：能够使用优先级队列实现。

# 2　系统需求分析与总体设计

## 2.1　系统需求分析

本系统应当要能够模拟不同数量窗口下不同情况的N个病人前来医院进行门诊的情况，根据其到达及诊治的时间利用小顶堆来进行排序，达到模拟排队，按序就诊或离开的目的，最后统计平均每个病人的等待时间，根据总用时及等待时间简单分析是否需要增减窗口以达到更高的效率。

需要处理的事务有：用户设置、模拟测试、记录分析，事务处理流程如下图2.1.1所示：（此同时为系统模块结构）

用户设置

模拟测试

记录分析

结束

图2.1.1　事务处理流程及系统模块结构

## 2.2　系统总体设计

系统采用简易菜单，先提示用户输入初始参数然后直接进行模拟得出统计结果并记入文件。过程如下：（其中Setting、SimulTest等函数也调用其他小的函数）

1

0

调用Setting函数进行初始参数设置

调用SimulTest函数进行模拟测试

屏幕打印统计结果，文件记录具体情况

再次或结束

结束

开始

图2.2.1　系统工作过程示意图

使用两个队列，wait\_que与treat\_que.

病人根据预先设置好的参数（数量，就诊时间范围，到达间隔等）使用rand函数随机生成，按到达时间先后顺序存入wait\_que队列中。当treat\_que队列未满时，从等待队列中将当前第一个病人加入treat\_que，当treat\_que满时，以时间为优先级排成小顶堆，取第一个病人，根据其是否已经就诊，展开相应就诊或离开事件。

# 3　系统详细设计

## 3.1　有关数据结构的定义

系统中主要要处理的数据有：设置的模拟参数、病人队列及病人本身的信息。

设置参数有门诊窗口数、模拟病人数、病人诊治时间、病人到达间隔等，具体如下表3.1.1所示：

表3.1.1　模拟参数类型

|  |  |  |
| --- | --- | --- |
| 数据项 | 数据名 | 数据类型 |
| 最长诊治时间 | LTREATTIME | int |
| 最短诊治时间 | STREATTIME | int |
| 门诊窗口数 | DOCNUM | Int |
| 测试病人数 | PATNUM | Int |
| 两病人最长到达时间间隔 | LINTERVAL | Int |
| 两病人最短到达时间间隔 | SINTERVAL | int |
| 模拟次数 | SIMULTIMES | int |

病人队列与病人信息如表3.1.2与3.1.3所示：

表3.1.2　病人队列结构

|  |  |  |
| --- | --- | --- |
| 数据项 | 数据名 | 数据类型 |
| 全部病人数组 | patients | Patient\* |
| 当前病人数 | cur\_num | int |
| 最大病人数 | max\_num | Int |

表3.1.3　病人信息结构

|  |  |  |
| --- | --- | --- |
| 数据项 | 数据名 | 数据类型 |
| ID | ID | int |
| 病人的到达或离开时间 | altime | int |
| 病人的诊治时间 | treattime | int |
| 标记病人是否已经诊治过 | istreated | int |

等待病人队列与治疗病人队列有如图3.1.1关系：

ID altime treattime istreated

0

1

…

…

PAT\_NUM

ID altime treattime istreated

0

1

…

…

DOC\_NUM

wait\_que

treat\_que

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
| 1 | 0 | 随机 | 0 |
|  |  |  |  |
| PAT\_NUM | 0 | 随机 | 0 |

图3.1.1　病人数组存储结构及关系示意图

## 3.2　主要算法设计

### 3.2.1　ADT部分

（1）Status Init\_PriorityQue(PatientQue\* pat\_que, int num);

\*函数名称：Init\_PriorityQue

\*输入参数：队列地址 &pat\_que 最大人数int num

\*返回值： Status 成功返回OK，分配失败退出

\*函数功能：创建空队列

\*设计：1.为数组分配存储空间；

2. pat\_que->cur\_num = 0;pat\_que->maxnum = num；

3.return OK；

\*复杂度：时间复杂度T(n) = O(1)

空间复杂度S(n) = O(1)

（2）Status Destroy\_PriorityQue(PatientQue\* pat\_que);

\*函数名称：Destroy\_PriorityQue

\*输入参数：队列地址 &pat\_que

\*返回值： Status 成功返回OK

\*函数功能：销毁队列

\*设计：1.释放存储空间；

2. pat\_que->patients = NULL;pat\_que->cur\_num = 0；

3.return OK；

\*复杂度：时间复杂度T(n) = O(1)

空间复杂度S(n) = O(1)

（3）Status Clear\_PriorityQue(PatientQue\* pat\_que);

\*函数名称：Clear\_PriorityQue

\*输入参数：队列地址 &pat\_que

\*返回值： Status 成功返回OK

\*函数功能：置空队列

\*设计：1. pat\_que->cur\_num = 0；

2.return OK；

\*复杂度：时间复杂度T(n) = O(1)

空间复杂度S(n) = O(1)

（4）Status PriorityQue\_Empty(PatientQue pat\_que);

\*函数名称：PriorityQue\_Empty

\*输入参数：队列 pat\_que

\*返回值： Status

\*函数功能：判断队列是否为空，空返回TRUE，否则FALSE

\*设计：return (pat\_que.cur\_num == 0)?TRUE:FALSE;

\*复杂度：时间复杂度T(n) = O(1)

空间复杂度S(n) = O(1)

（5）Status PriorityQue\_Full(PatientQue pat\_que);

\*函数名称：PriorityQue\_Full

\*输入参数：队列 pat\_que

\*返回值： Status

\*函数功能：判断队列是否满，满返回TRUE，否则FALSE

\*设计：return (pat\_que.cur\_num == pat\_que.maxnum)?TRUE:FALSE;

\*复杂度：时间复杂度T(n) = O(1)

空间复杂度S(n) = O(1)

（6）Status PriorityQue\_Insert(PatientQue\* pat\_que, Patient pat)

\*函数名称：PriorityQue\_Insert

\*输入参数：队列地址 &pat\_que, 插入的病人pat

\*返回值： Status

\*函数功能：插入新的病人，成功返回OK，失败返回ERROR

\*设计：1.判断队列是否满，满则返回ERROR，否则继续第2步

2. pat\_que->patients[++(pat\_que->cur\_num)] = pat;插入病人

3. 将patients[1...cur\_num]重新调整成小顶堆：

for (i = pat\_que->cur\_num/2; i > 0; --i)

{

HeapAdjust(pat\_que->patients, i, pat\_que->cur\_num);

}

4.return OK；

\*复杂度：时间复杂度T(n) = O(n)

空间复杂度S(n) = O(1)

（7）Patient PriorityQue\_DeletMin(PatientQue\* pat\_que)

\*函数名称：PriorityQue\_DeletMin

\*输入参数：队列地址 &pat\_que

\*返回值： 删除的病人 Patient

\*函数功能：删除优先级最小的病人，并返回其信息

\*设计：（如图2.3.1）

开始

将patients[1]与patients[cur\_num]互换位置，cur\_num减1

将patients[1…cur\_num]重新调整为小顶堆，return OK；

结束

图3.2.1　DeletMin函数设计流程图

\*复杂度：时间复杂度T(n) = O(n)

空间复杂度S(n) = O(1)

### 3.2.2　模拟测试部分

（1）Status Setting(FILE\*\* fp);

\*函数名称：Setting

\*输入参数：文件指针地址 fp

\*返回值： Status

\*函数功能：进行模拟参数设定

\*设计：展开设置界面，提示用户一步步进行模拟参数设置

\*复杂度：时间复杂度T(n) = O(1)

空间复杂度S(n) = O(1)

（2）void LogSettingInfo(FILE\* fp)；

\*函数名称：LogSettingInfo

\*输入参数：文件指针fp

\*返回值： 无

\*函数功能：记录设置信息

\*设计：使用fprintf函数将初始设置按格式录入文件中

\*复杂度：时间复杂度T(n) = O(1)

空间复杂度S(n) = O(1)

（3）void CreatePatients(PatientQue\* pat\_que);

\*函数名称：CreatePatients

\*输入参数：病人队列地址&pat\_que

\*返回值： 无

\*函数功能：产生随机病人序列

\*设计：1. srand((unsigned)time(NULL));

2.创建第一个病人，其到达时间设为0；其余信息按照初始设置随机生成。

3.循环，将剩下的病人创建，各信息按照初始设置随机生成。

\*复杂度：时间复杂度T(n) = O(n)

空间复杂度S(n) = O(1)

（4）void ShowInterface(void);

\*函数名称：ShowInterface

\*输入参数：无

\*返回值： 无

\*函数功能：显示主界面

\*设计：按照预先设置的，printf出标题、设置内容等信息

\*复杂度：时间复杂度T(n) = O(1)

空间复杂度S(n) = O(1)

（5）void LogTreat(FILE\* fp, Patient pat, int time);

\*函数名称：LogTreat

\*输入参数：Patient pat接受诊治的病人,int time当前时间,fp存入的文件

\*返回值： 无

\*函数功能：记录诊治病人的信息

\*设计：将病人的等待时间、开始治疗时间等数据录入fp指向的文件

\*复杂度：时间复杂度T(n) = O(1)

空间复杂度S(n) = O(1)

（6）void Analyse(int totaltime, int totalwaittime, FILE\* fp);

\*函数名称：Analyse

\*输入参数：int totaltime总时,int totalwaittime总等待时间 fp存入的文件

\*返回值： 无

\*函数功能：对最后结果进行分析

\*设计：计算多次模拟的平均总用时和平均每个病人的等待时间，根据这两个值的大小，用if …else if….else输出预先设定好的分析

\*复杂度：时间复杂度T(n) = O(1)

空间复杂度S(n) = O(1)

（7）void SimulTest(FILE\* fp);

\*函数名称：SimulTest

\*输入参数：文件指针fp

\*返回值： 无

\*函数功能：模拟测试

\*设计： 单次模拟如图3.2.2所示：

Y

Y

Y

Y

开始

等待队列未空，即有人等待或未到

治疗队列未满，即有空闲窗口

将等待队列中第一人加进治疗队列，等待队列人数减1，治疗队列调整成小顶堆

治疗队列中优先级最小的人未治疗

1.该病人altime变为离开时间（altime+treattime），istreted = 1，计算等待时间，并使当前时间到达该病人治疗时的时间。

2.将其信息录入文件。 3.治疗队列重新调整成小顶堆

该病人离开，当前时间进行到离开时间。

治疗队列重新调整成小顶堆

治疗队列未空

优先级第一的病人诊治或离开并记录信息（如上）

治疗队列重新调整成小顶堆

结束

N

N

N

N

图3.2.2　SimulTest函数设计流程图

\*复杂度：时间复杂度T(n) = O(n)

空间复杂度S(n) = O(1)

# 4　系统实现与测试

## 4.1　系统实现

编程环境： 本系统在win8.1上使用Dev—C++进行编译调试，使用语言C语言。

数据类型定义：根据3.1，用C语言定义如下：

//函数结果状态代码

#define TRUE 1

#define FALSE 0

#define OK 1

#define ERROR 0

#define OVERFLOW -2

typedef int Status;

int LTREATTIME; //最长诊治时间

int STREATTIME; //最短诊治时间

int DOCNUM; //门诊窗口数

int PATNUM; //测试病人数

int LINTERVAL; //两病人最长到达时间间隔

int SINTERVAL; //两病人最短到达时间间隔

int SIMULTIMES; //模拟次数

typedef struct {

int ID; //病人编号

int altime; //病人的到达或离开时间

int treattime; //病人的诊治时间

int istreated; //标记病人是否已经诊治过

}Patient;

typedef struct {

Patient\* patients; //全部病人队列

int cur\_num; //当前病人数

int maxnum; //最大病人数

}PatientQue;

程序代码：详见附录1 PriorityQue.h，附录2 main.c。

程序使用函数及各函数功能和说明详见**3.2**；

各函数调用关系如下：

Setting

LogSettingInfo

（用户完成数据设置并记录）

ShowInterface

SimulTest

Init\_PriorityQue

CreatePatients

PriorityQue\_Insert

PriorityQue\_DeletMin

LogTreat

Analyse

Destroy\_PriorityQue

（模拟并记录分析）

图4.1.1　函数调用及实现模块关系图

## 4.2　系统测试

本系统由用户输入测试数据并自动进行模拟与存储，故可以直接进行测试来查看各模块是否符合预期。

测试过程截图如下：

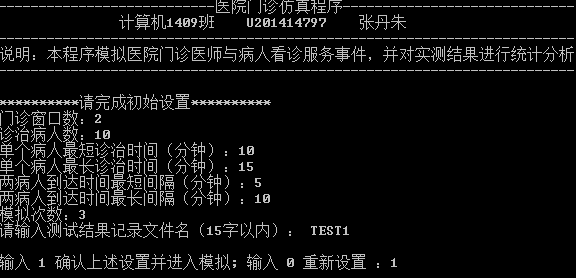


图4.2.1　设置模拟参数



图4.2.2　模拟过程

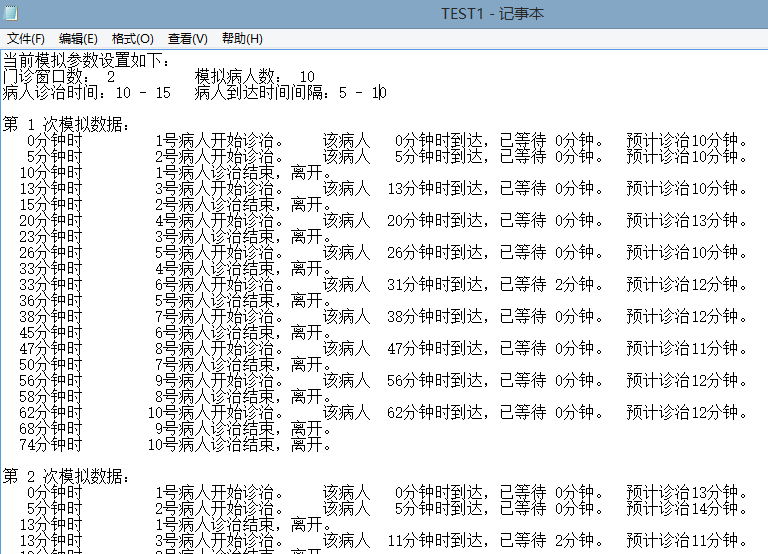


图4.2.3　模拟结果文件记录

由此可见，本系统能够按照预期进行病人诊治的模拟并进行简单的统计分析，逻辑顺序等无误，该有的设置、模拟、记录模块都完成了。界面简洁易懂易操作，用户只需按照提示即可进行模拟获得想要的数据。

课题要求测试数据有一定的规模，故接着测试数据较大的情况：

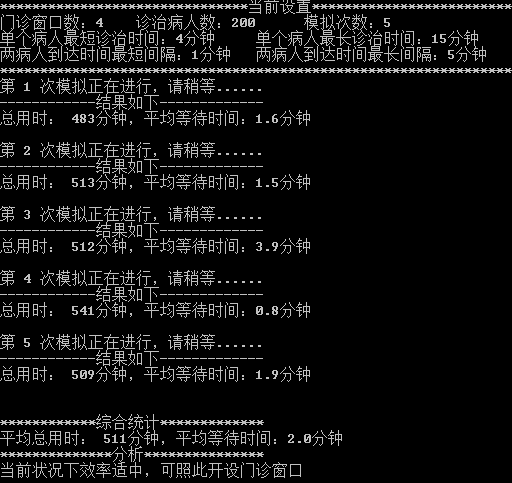


图4.2.4　较大数据模拟过程

能够顺利进行模拟，接着查看文件中的详细记录，发现逻辑顺序无误，成功的完成了模拟（见图4.2.5）

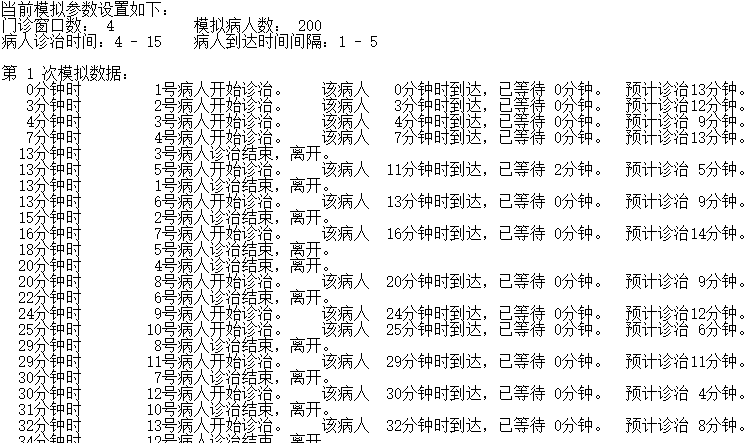


图4.2.5　模拟结果文件记录2

# 5　总结与展望

## 5.1　全文总结

本次设计主要工作如下：

（1）对课题进行分析，明确了课题任务，确定了所要处理的数据，进行了相应的常量、结构等的定义。

（2）使用小顶堆并以病人的到达或离开时间作为优先级进行了优先级队列的设计。使用rand函数随机生成相应数据来模拟现实情况。

（4）进行了优先级队列ADT的设计与测试。

（4）分析系统该完成的事务，进行了设置、模拟、记录的相关函数的设计。

（5）设计系统界面。

（6）进行测试并改进。

## 5.2　工作展望

在今后的研究中，围绕着如下几个方面开展工作。。。。。。。。

（1）算法的优化，从时间与空间复杂度的角度尽可能地简化代码达到提高运行效率的目的。

（2）对于模拟类的程序，要尽量全面的考虑到实际生活情况，尽可能真实地模拟现实情况。对于生成的模拟数据，可以不用随机，而是使其服从某一分布规律从而更贴近实际。

（3）界面设计与操作需要提高，尽可能方便用户。

# 6　体　　会

本次课程设计，使我充分地理解了数据结构在实际的编程中所起的重要作用。数据结构本身便是对现实所要处理的数据的一种抽象化处理，反过来对于一个实际的问题，在编程时应当先考虑清楚所要处理的数据有哪些，并明确数据之间的关系从而选择相应的数据结构来存储数据，由此可以使程序的设计与运行更加高效合理。

在模拟现实中的事物时，应尽可能考虑全面并选用合适的分布规律等来表现实际情况。在模拟时要抓住主要的内容尽可能详尽地模拟出来，而对于一些相对不太重要的数据则可以进行大致模拟。

同时，在本次设计中最主要的是学会了优先级队列的使用，明白了优先级队列的使用场合与优点。

# 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]　Mark Allen Weiss.Data Structures and Algorithm Analysis in C, 机械工业出版社，2010, 177-192

# 附录　1

PriorityQue.h

#include <stdio.h>

#include <stdlib.h>

#include <windows.h>

//函数结果状态代码

#define TRUE 1

#define FALSE 0

#define OK 1

#define ERROR 0

#define OVERFLOW -2

typedef int Status;

int LTREATTIME; //最长诊治时间

int STREATTIME; //最短诊治时间

int DOCNUM; //门诊窗口数

int PATNUM; //测试病人数

int LINTERVAL; //两病人最长到达时间间隔

int SINTERVAL; //两病人最短到达时间间隔

int SIMULTIMES; //模拟次数

typedef struct {

int ID; //病人编号

int altime; //病人的到达或离开时间

int treattime; //病人的诊治时间

int istreated; //标记病人是否已经诊治过

}Patient;

typedef struct {

Patient\* patients; //全部病人队列

int cur\_num; //当前病人数

int maxnum; //最大病人数

}PatientQue;

Status Init\_PriorityQue(PatientQue\* pat\_que, int num);

Status Destroy\_PriorityQue(PatientQue\* pat\_que);

Status Clear\_PriorityQue(PatientQue\* pat\_que);

Status PriorityQue\_Insert(PatientQue\* pat\_que, Patient pat);

Patient PriorityQue\_DeletMin(PatientQue\* pat\_que);

Status PriorityQue\_Empty(PatientQue pat\_que);

Status PriorityQue\_Full(PatientQue pat\_que);

void HeapAdjust(Patient\* patients, int s, int m);

void HeapSort(Patient\* patients, int num);

/\*\*

\*函数名称：Init\_PriorityQue

\*输入参数：队列地址 &pat\_que 最大人数int num

\*返回值： Status 成功返回OK，分配失败退出

\*函数功能：创建空队列

\*/

Status Init\_PriorityQue(PatientQue\* pat\_que, int num)

{

pat\_que->patients = (Patient\*)malloc((num + 1) \* sizeof(Patient));

if (!pat\_que->patients)

{

exit(OVERFLOW);

}

pat\_que->cur\_num = 0;

pat\_que->maxnum = num;

return OK;

}

/\*\*

\*函数名称：Destroy\_PriorityQue

\*输入参数：队列地址 &pat\_que

\*返回值： Status 成功返回OK

\*函数功能：销毁队列

\*/

Status Destroy\_PriorityQue(PatientQue\* pat\_que)

{

free(pat\_que->patients);

pat\_que->patients = NULL;

pat\_que->cur\_num = 0;

return OK;

}

/\*\*

\*函数名称：Clear\_PriorityQue

\*输入参数：队列地址 &pat\_que

\*返回值： Status 成功返回OK

\*函数功能：置空队列

\*/

Status Clear\_PriorityQue(PatientQue\* pat\_que)

{

pat\_que->cur\_num = 0;

return OK;

}

/\*\*

\*函数名称：PriorityQue\_Empty

\*输入参数：队列 pat\_que

\*返回值： Status

\*函数功能：判断队列是否为空，空返回TRUE，否则FALSE

\*/

Status PriorityQue\_Empty(PatientQue pat\_que)

{

return (pat\_que.cur\_num == 0)?TRUE:FALSE;

}

/\*\*

\*函数名称：PriorityQue\_Full

\*输入参数：队列 pat\_que

\*返回值： Status

\*函数功能：判断队列是否满，满返回TRUE，否则FALSE

\*/

Status PriorityQue\_Full(PatientQue pat\_que)

{

return (pat\_que.cur\_num == pat\_que.maxnum)?TRUE:FALSE;

}

/\*\*

\*函数名称：PriorityQue\_Insert

\*输入参数：队列地址 &pat\_que, 插入的病人pat

\*返回值： Status

\*函数功能：插入新的病人，成功返回OK，失败返回ERROR

\*/

Status PriorityQue\_Insert(PatientQue\* pat\_que, Patient pat)

{

int i;

if (PriorityQue\_Full(\*pat\_que))

{

return ERROR;

}

pat\_que->patients[++(pat\_que->cur\_num)] = pat;

//将patients[1...cur\_num]重新调整成小顶堆

for (i = pat\_que->cur\_num/2; i > 0; --i)

{

HeapAdjust(pat\_que->patients, i, pat\_que->cur\_num);

}

return OK;

}

/\*\*

\*函数名称：PriorityQue\_DeletMin

\*输入参数：队列地址 &pat\_que

\*返回值： 删除的病人 Patient

\*函数功能：删除优先级最小的病人，并返回其信息

\*/

Patient PriorityQue\_DeletMin(PatientQue\* pat\_que)

{

Patient temp;

temp = pat\_que->patients[1];

pat\_que->patients[1] = pat\_que->patients[pat\_que->cur\_num];

pat\_que->patients[(pat\_que->cur\_num)--] = temp;

//将patients[1...cur\_num]重新调整成小顶堆

HeapAdjust(pat\_que->patients, 1, pat\_que->cur\_num);

return temp;

}

/\*\*

\*函数名称：HeapAdjust

\*输入参数：病人数组 patient\* patients,起始与结束位置序号s、m

\*返回值： 无

\*函数功能：将patients[s...m]调整成小顶堆

\*/

void HeapAdjust(Patient\* patients, int s, int m)

{

int j;

Patient sb = patients[s];

for (j = 2 \* s; j <= m; j \*= 2)//沿着关键字较小的孩子结点向下筛选

{

if (j < m && patients[j].altime > patients[j+1].altime)

{

j++;

}

if (!(sb.altime > patients[j].altime))

{

break; //sb应插入在位置s上

}

patients[s] = patients[j];

s = j;

}

patients[s] = sb; //插入

}

# 附录　2

main.c

#include "PriorityQue.h"

Status Setting(FILE\*\* fp);

void ShowInterface(void);

void CreatePatients(PatientQue\* pat\_que);

void SimulTest(FILE\* fp);

void LogTreat(FILE\* fp, Patient pat, int time);

void LogSettingInfo(FILE\* fp);

void Analyse(int totaltime, int totalwaittime, FILE\* fp);

/\*\*

\*函数名称：Setting

\*输入参数：文件指针地址 fp

\*返回值： Status

\*函数功能：进行模拟参数设定

\*/

Status Setting(FILE\*\* fp)

{

int confirm = 0;

char filename[30];

while(!confirm) {

system("cls");

printf("---------------------------医院门诊仿真程序-----------------------------\n");

printf(" 计算机1409班 U201414797 张丹朱\n");

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

printf("说明：本程序模拟医院门诊医师与病人看诊服务事件，并对实测结果进行统计分析\n");

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

printf("\n\*\*\*\*\*\*\*\*\*\*请完成初始设置\*\*\*\*\*\*\*\*\*\*\n");

printf("门诊窗口数：");

scanf("%d", &DOCNUM);

printf("诊治病人数：");

scanf("%d", &PATNUM);

printf("单个病人最短诊治时间（分钟）：");

scanf("%d", &STREATTIME);

printf("单个病人最长诊治时间（分钟）：");

scanf("%d", &LTREATTIME);

while(LTREATTIME <= STREATTIME)

{

printf(" 设置有误！最长时间应大于最短时间！请重输：");

scanf("%d", &LTREATTIME);

}

printf("两病人到达时间最短间隔（分钟）：");

scanf("%d", &SINTERVAL);

printf("两病人到达时间最长间隔（分钟）：");

scanf("%d", &LINTERVAL);

while(LINTERVAL <= SINTERVAL)

{

printf(" 设置有误！最长间隔应大于最短间隔！请重输：");

scanf("%d", &LINTERVAL);

}

printf("模拟次数：");

scanf("%d", &SIMULTIMES);

printf("请输入测试结果记录文件名（15字以内）： ");

scanf("%s",&filename);

if ((\*fp = fopen(filename, "wt+")) == NULL)

{

printf("文件打开失败！数据无法保存！\n ");

return ERROR;

}

LogSettingInfo(\*fp); //记录设置

printf("\n输入 1 确认上述设置并进入模拟; 输入 0 重新设置 ：");

scanf("%d", &confirm);

}

return OK;

}

/\*\*

\*函数名称：LogSettingInfo

\*输入参数：文件指针fp

\*返回值： 无

\*函数功能：记录设置信息

\*/

void LogSettingInfo(FILE\* fp)

{

fprintf(fp, "%s\n", "当前模拟参数设置如下：");

fprintf(fp, "%s%d\t\t%s%d\n", "门诊窗口数： ", DOCNUM, "模拟病人数： ", PATNUM);

fprintf(fp, "%s%d%s%d\t", "病人诊治时间：", STREATTIME, " - ", LTREATTIME);

fprintf(fp, "%s%d%s%d\n", "病人到达时间间隔：", SINTERVAL, " - ", LINTERVAL);

}

/\*\*

\*函数名称：ShowInterface

\*输入参数：无

\*返回值： 无

\*函数功能：显示主界面

\*/

void ShowInterface(void)

{

system("cls");

printf("---------------------------医院门诊仿真程序-----------------------------\n");

printf(" 计算机1409班 U201414797 张丹朱\n");

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

printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*当前设置\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

printf("门诊窗口数：%d 诊治病人数：%d 模拟次数：%d\n", DOCNUM, PATNUM, SIMULTIMES);

printf("单个病人最短诊治时间：%d分钟\t单个病人最长诊治时间：%d分钟\n", STREATTIME, LTREATTIME);

printf("两病人到达时间最短间隔：%d分钟\t两病人到达时间最长间隔：%d分钟\n", SINTERVAL, LINTERVAL);

printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

}

/\*\*

\*函数名称：CreatePatients

\*输入参数：病人队列地址&pat\_que

\*返回值： 无

\*函数功能：产生随机病人序列

\*/

void CreatePatients(PatientQue\* pat\_que)

{

int i;

srand((unsigned)time(NULL));

//产生第一个病人并以其到达时间为时间零点

++(pat\_que->cur\_num);

pat\_que->patients[1].ID = 1;

pat\_que->patients[1].altime = 0;

pat\_que->patients[1].treattime = rand() % (LTREATTIME - STREATTIME) + STREATTIME;

pat\_que->patients[1].istreated = 0;

//产生剩下病人

for (i = 2; i <= PATNUM; i++)

{

++(pat\_que->cur\_num);

pat\_que->patients[i].ID = i;

pat\_que->patients[i].altime = pat\_que->patients[i-1].altime + rand() % (LINTERVAL - SINTERVAL) + SINTERVAL;

pat\_que->patients[i].treattime = rand() % (LTREATTIME - STREATTIME) + STREATTIME;

pat\_que->patients[i].istreated = 0;

}

}

/\*\*

\*函数名称：SimulTest

\*输入参数：文件指针fp

\*返回值： 无

\*函数功能：模拟测试

\*/

void SimulTest(FILE\* fp)

{

int i;

int waittime, time; //记录单次模拟总用时+总等待时间

int totalwaittime = 0, totaltime = 0;//记录全部模拟总用时+总等待时间

PatientQue wait\_que; //等待病人队列

PatientQue treat\_que; //诊治病人队列

Patient temp;

Init\_PriorityQue(&wait\_que, PATNUM); //创建空队列

Init\_PriorityQue(&treat\_que, DOCNUM); //创建空队列

for (i = 1; i <= SIMULTIMES; i++)

{

printf("第 %d 次模拟正在进行，请稍等......\n", i);

Sleep(1000);

fprintf(fp, "\n%s%d%s\n", "第 ", i, " 次模拟数据：");

CreatePatients(&wait\_que);

waittime = 0;

time = 0;

//模拟 统计 记录

while (wait\_que.cur\_num) //有病人在等待

{

if (!PriorityQue\_Full(treat\_que))//门诊窗口有空闲

{

temp = wait\_que.patients[PATNUM - (--(wait\_que.cur\_num))];

PriorityQue\_Insert(&treat\_que, temp);//病人进入治疗队列

}

else //门诊窗口满

{

temp = PriorityQue\_DeletMin(&treat\_que);

if (!temp.istreated) //最先发生的是开始诊治事件

{

if (time < temp.altime)//病人到达前无事件发生

{

time = temp.altime; //时间进行到病人到达

}

LogTreat(fp, temp, time); //记录诊治病人的信息

waittime += time - temp.altime;//总等待时间增加

temp.altime = time + temp.treattime;//获得病人离开时间

temp.istreated = 1;

PriorityQue\_Insert(&treat\_que, temp);//病人进入治疗队列

}

else ////最先发生的是离开事件

{

time = temp.altime; //时间进行到病人离开

fprintf(fp, "%4d%s\t%4d%s\n", time, "分钟时", temp.ID, "号病人诊治结束，离开。");

}

}

}

while (treat\_que.cur\_num) //有病人在治疗

{

temp = PriorityQue\_DeletMin(&treat\_que);

if (!temp.istreated) //最先发生的是开始诊治事件

{

if (time < temp.altime)//病人到达前无事件发生

{

time = temp.altime; //时间进行到病人到达

}

LogTreat(fp, temp, time); //记录诊治病人的信息

waittime += time - temp.altime;//总等待时间增加

temp.altime = time + temp.treattime;//获得病人离开时间

temp.istreated = 1;

PriorityQue\_Insert(&treat\_que, temp);//病人进入治疗队列

}

else ////最先发生的是离开事件

{

time = temp.altime; //时间进行到病人离开

fprintf(fp, "%4d%s\t%4d%s\n", time, "分钟时", temp.ID, "号病人诊治结束，离开。");

}

}

//输出统计结果：总用时+平均等待时间等并记入日志

printf("------------结果如下-------------\n");

printf("总用时：%4d分钟，平均等待时间：%.1f分钟\n\n", time, (float)waittime/PATNUM);

//将本次模拟得到的数据存入总数据

totalwaittime += waittime;

totaltime += time;

}

//输出多次模拟平均统计结果，并分析记录

Analyse(totaltime, totalwaittime, fp);

fclose(fp);

Destroy\_PriorityQue(&wait\_que);

Destroy\_PriorityQue(&treat\_que);

}

/\*\*

\*函数名称：LogTreat

\*输入参数：Patient pat接受诊治的病人,int time当前时间,fp存入的文件

\*返回值： 无

\*函数功能：记录诊治病人的信息

\*/

void LogTreat(FILE\* fp, Patient pat, int time)

{

fprintf(fp, "%4d%s\t%4d%s", time, "分钟时", pat.ID, "号病人开始诊治。");

fprintf(fp, "\t%s%4d%s", "该病人", pat.altime, "分钟时到达，");

fprintf(fp, "%s%2d%s", "已等待", time-pat.altime, "分钟。");

fprintf(fp, "%s%2d%s\n", " 预计诊治", pat.treattime, "分钟。");

}

/\*\*

\*函数名称：Analyse

\*输入参数：int totaltime总时,int totalwaittime总等待时间 fp存入的文件

\*返回值： 无

\*函数功能：对最后结果进行分析

\*/

void Analyse(int totaltime, int totalwaittime, FILE\* fp)

{

int ave\_time = totaltime/SIMULTIMES;

float ave\_wait = (float)totalwaittime/(SIMULTIMES \* PATNUM);

printf("\n\*\*\*\*\*\*\*\*\*\*\*\*综合统计\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

printf("平均总用时：%4d分钟，平均等待时间：%.1f分钟\n", ave\_time, ave\_wait);

fprintf(fp, "\n%s%4d", "平均总用时：", ave\_time);

fprintf(fp, "%s%.2f%s\n", "分钟，平均等待时间：", ave\_wait, "分钟");

printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*分析\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

if (ave\_time > 600 || ave\_wait > 15)

{

printf("当前状况下繁忙，效率低下，建议增开门诊窗口\n\n");

}

else if (ave\_time < 300 && ave\_wait < 8)

{

printf("当前状况下清闲，可视情况减少门诊窗口\n\n");

}

else

{

printf("当前状况下效率适中，可照此开设门诊窗口\n\n");

}

}

int main(int argc, char \*argv[])

{

int choice = 1;

FILE\* fp;

while (choice)

{

if (Setting(&fp) == ERROR)

{

break;

}

ShowInterface();

SimulTest(fp);

printf("\n输入 1 重新设置参数进行模拟; 输入 0 退出 ：");

scanf("%d", &choice);

}

printf("欢迎下次再使用本系统！");

return 0;

}