華中科技大學

课程设计报告

题目： 基于堆的优先级队列 ADT 实现及应用

课程名称： 数据结构课程设计

专业班级： 计卓 1401 班

学　　号： U201410081

姓　　名： 李冠宇

指导教师： 许贵平

报告日期： 2016 年 3 月 5 日

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

目　录

1 引言

1.1 课题背景与意义

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）。

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。

1.2 课程设计的主要研究工作

（1）基于堆的概念设计优先级队列(Priority Queue)抽象数据类型

（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

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

（5）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

2 系统需求分析与总体设计

2.1 系统需求分析

（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。

2.2 系统总体设计

从整体上分析，我们的系统要设计成一个界面整洁、美观，操作方便的医院门诊医师与病人看诊服务事件仿真系统。其中，对于病人数据应采用基于堆的概念设计的优先级队列，并且提供一系列的操作函数，同时，还要有人机交互的图形界面，因此整个系统可以分为三大模块：数据模块，图形界面模块，主程序模块。接下来，将详细介绍各模块的功能和作用。

数据模块。在该模块中，我们将会实现整个系统用到的核心的数据结构——基于堆的概念设计的优先级队列，底层的物理存储结构将会采用顺序表，并且实现针对优先级队列的相关操作函数。该模块主要负责为主程序中的数据类型和函数提供模板，用于存储、查询、操作对应的病人数据。

图形界面模块。在该模块中，我们将会实现整个系统的图形化界面，包括主程序窗口以及各种弹出窗口等。该模块将会利用Qt图形库进行实现且主程序窗口的查询和操作均通过菜单栏实现，主要负责系统的整洁、美观、操作方便的图形界面的实现。

主程序模块。在该模块中，我们将会实现仿真事件（如病人到达，病情复杂度/就诊时间，病人离开等）根据某种概率分布或随机模型生成，并且利用数据模块中实现的数据类型与相关函数以存储、查询、操作对应的病人数据，同时利用图形界面模块实现的图形化界面来最终实现整个系统。该模块主要负责综合数据模块与图形界面模块，以实现一个界面整洁、美观，操作方便的医院门诊医师与病人看诊服务事件仿真系统。

综上所述，我们医院门诊医师与病人看诊服务事件仿真系统的系统模块结构图，如图2-1所示

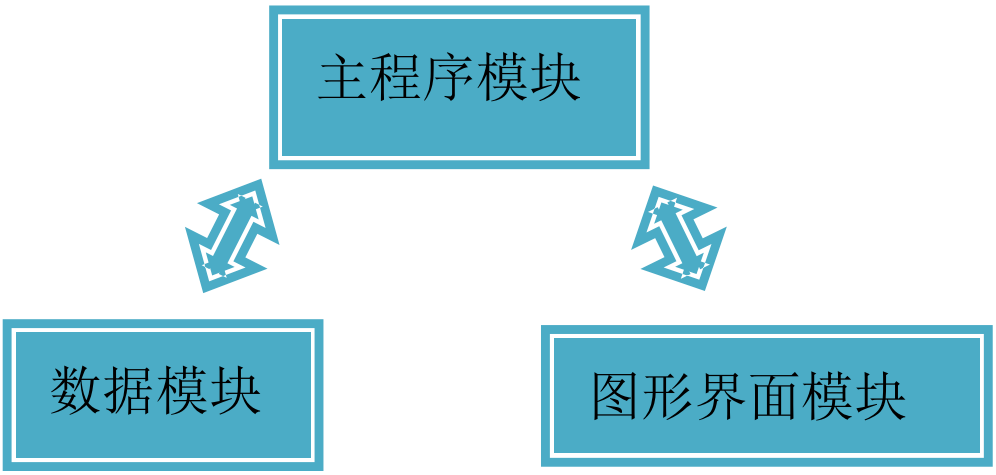


图 2-1 系统模块结构图

3 系统详细设计

3.1 有关数据结构的定义

整个系统中最主要也是最核心的便是基于堆的概念设计的优先级队列。为了实现这一数据结构，我们利用C++的类的思想把相关的数据结构及操作函数封装成一个类并利用继承功能一级级地实现。主要用到的数据结构类型有SqList、Heap、PriorityQueue、Patient等，接下来将逐一介绍(此部分只展示类的成员变量，成员函数将在3.2中介绍)。

（1） SqList模板

SqList实现的主要是优先级队列的物理存储结构，显然采用了顺序表作为物理存储结构，并包括顺序表的操作函数，其具体的组成如表3-1所示。

名称	类型	备注
elem	T(模板)*	顺序表数组元素指针
len	int	当前顺序表长度
list_size	int	顺序表最大长度

表3-1 SqList成员变量

（2） Heap模板

Heap实现的主要是优先级队列的基本结构，采用了堆的概念进行实现，Heap继承于SqList类，因此其含有SqList的所有成员变量与成员函数，由于Heap并未再定义其他的成员变量，因此这里不再给出Heap的成员变量。需要注意的是，实现Heap结构的关键并不在于其成员变量，而是其成员函数。

（3） PriorityQueue模板

PriorityQueue实现的主要是优先级队列，PriorityQueue继承于Heap类，因此其含有Heap的所有成员变量与成员函数，由于PriorityQueue并未再定义其他的成员变量，因此这里不再给出PriorityQueue的成员变量。

（4） Patient

Patient实现的是存储病人信息的数据结构，其具体组成，如表3-2所示。

名称	类型	备注
id	int	病人的序号
degree	int	病人的病急程度
arrive	QTime	病人的到达时间
expense	int	病人需要的就诊时间

表 3-2 Patient 成员变量

4种数据在系统中的关联可通过图3-1直观的看出。

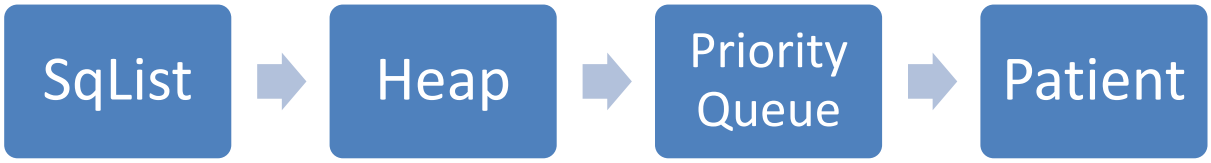


图3-1 数据结构关系图

3.2 主要算法设计

（1）数据模块

数据模块正如前面介绍的，最主要的便是要实现系统的各种数据结构，以及针对各数据结构的操作函数。根据3.1的介绍，我们知道系统中共涉及到4大类数据结构以及对应的成员变量，SqList类为顺序表，其相关操作函数在前面的数据结构实验中详细介绍过了，这里便不再累述；Heap类为堆，其相关操作函数大部分继承于SqList并稍微改动，但改写程度最大的主要有3个函数包括插入元素、删除元素和获取堆顶元素，其中获取堆顶元素实现比较简单，删除元素算法与插入元素算法相类似，因此将主要介绍Heap的插入元素的算法：

首先，将新增元素插入顺序表，也就是堆底，然后对堆进行调整，即比较插入元素的优先级与其双亲节点的优先级，如果新插入元素的优先级大于双亲结点，则需将插入元素上移直到其小于双亲结点的优先级。

该算法的流程图，如图3-2所示。

（2）图形界面模块

图形界面的实现主要采用了Qt，利用的都是Qt封装好的函数API，因此实现起来相对简单。其总体流程如下：

首先，载入程序主窗口的设计UI，然后，对主窗口以及主窗口中的个别组件进行相关设置，接着，为主窗口中的部分组件(如菜单栏、按钮)设置回调函数，

至此，图形化界面的启动过程完成。接下来，程序进入循环来捕获鼠标或键盘事件调用相关组件的回调函数直到退出程序。

该算法的流程图，如图3-3所示。

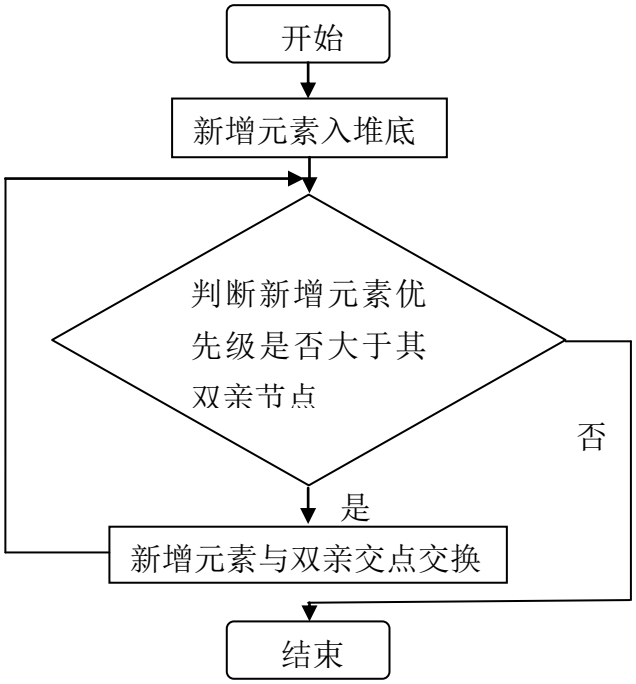


图 3-2

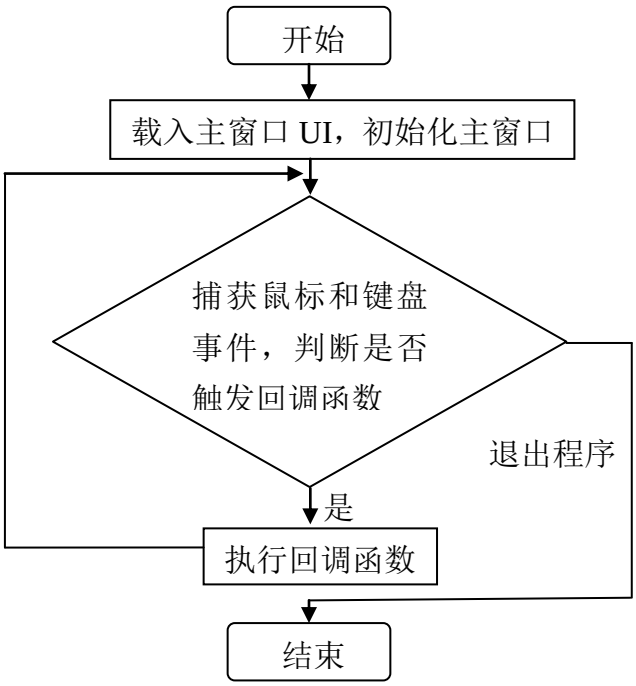


图 3-3

（3）主程序模块

该模块主要是调度和结合两个模块，结合方式主要体现在回调函数中。

4 系统实现与测试

4.1 系统实现

（1）实现目标

应用优先级队列 ADT 设计与实现一个界面整洁、美观，操作方便的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。可以仿真模拟病人的挂号与就诊，通过设置优先级（根据病人的到达时间、病急程度与需要的就诊时间按权重综合得到），合理安排病人的就诊顺序。

（2）系统环境

系统：x86_64 GNU/Linux 内核：4.2.5-1 发行版本：Arch
编程 IDE：Qt Creater3.6.0(利用 Qt 实现图形化界面)

Qt 版本：Qt5.5.1

GCC：GCC4.9.1

（3）程序代码见附录，程序数据结构声明实现如下

SqList：

```
template<class T>
class Sqlist
{
public:
    T* elem;              //顺序表数组元素指针
    int len;              //当前顺序表长度
    int list_size;        //顺序表最大长度

    void list_init();                   //初始化顺序表
    void list_destroy();                //销毁顺序表
    void list_clear();                  //清空顺序表
    bool list_is_empty();               //判断顺序表是否为空
    bool list_is_full();                //判断顺序表是否为满
    int list_get_len();                 //获取顺序表长度
    void list_insert(int i, T t);       //在顺序表第 i 处插入元素
    void list_delete_elem(int i);       //删除顺序表第 i 处的元素
```

```
    T list_get_item(int i);              //返回顺序表第 i 处的元素
};
```

Heap：

```
template<class T>
class Heap: public Sqlist<T> {
public:
    bool (*greater)(T t1, T t2);              //优先级比较函数

    Heap(bool (*func)(T t1, T t2));

    void heap_insert_elem(T t);               //向堆中插入元素
    T heap_get_top();                         //获取堆顶元素
    void heap_delete_top();                   //删除堆顶的元素

};
```

PriorityQueue：

```
template<class T>
class PriorityQueue: public Heap<T> {
public:
    PriorityQueue(bool (*greater)(T, T));
    PriorityQueue(const PriorityQueue& pd);
    PriorityQueue(bool (*greater)(T, T), const PriorityQueue& pd);

    int size();                  //获取优先级队列长度
    void insert(T t);           //插入新元素
    void delete_elem();         //删除元素
    void clear_all();           //清空优先级队列
    void destroy();              //销毁优先级队列
    T top();                      //获取优先级队首元素
    bool empty();                 //判断优先级队列是否为空
    bool full();                 //判断优先级队列是否已满
};
```

Patient:

```
class Patient
{
public:
    Patient();
    Patient(int id, int degree, QTime arrive, int expense);

    void generate_data();

    static bool greater(Patient const p1, Patient const p2)
    {
        if(p1.arrive.secsTo(p2.arrive) > p1.expense * 60)
            return true;
        else
            return((p1.expense-p2.expense)*60*0.7>p2.arrive.secsTo(p1.arrive)*
0.3);
    }

    static bool later(Patient const p1, Patient const p2)
    {
        return p1.arrive.secsTo(p2.arrive) > 0;
    }

    int id;              //病人的序号
    int degree;          //病人的病急程度
    QTime arrive;        //病人的到达时间
    int expense;         //病人需要的就诊时间
};
```

4.2 系统测试

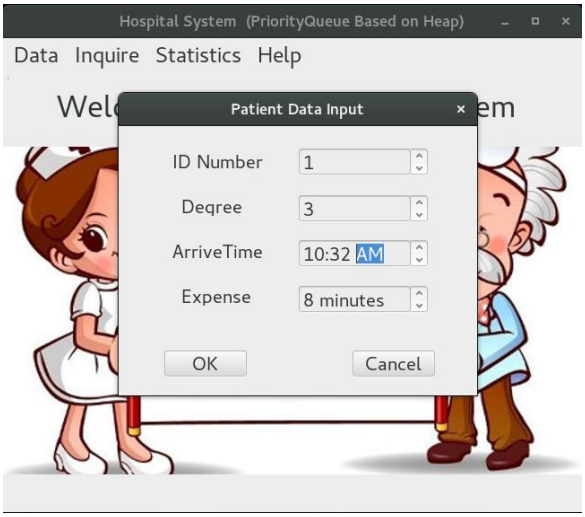
我们将从四个方面对系统进行测试，包括数据操作、数据查询、数据统计、相关信息。

图4-1 程序欢迎界面(主窗口)

（1） 数据操作(Data)

该测试主要针对的菜单栏Data下的各项数据操作，包括“Random Generate”(自动生成患者数据,即数据仿真)、“Input”(输入一个患者数据)、“Delete”(删除一个患者数据，即表示优先级最高的患者已经就诊完)、“Clear”(清空当前所有患者数据)四个数据操作项。接下来将按顺序依次测试Input、Random Generate、Delete、Clear操作。

Input：



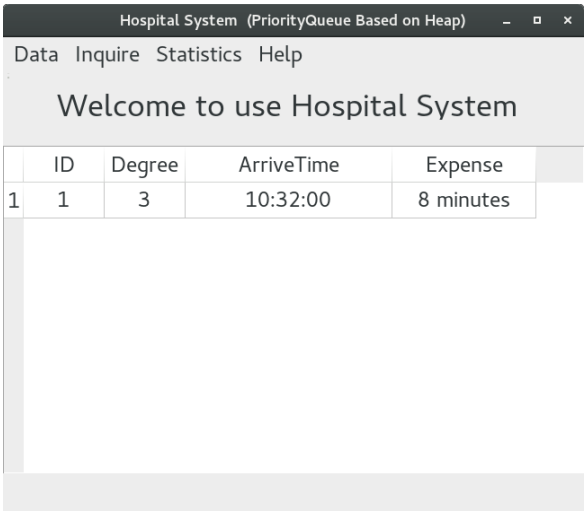
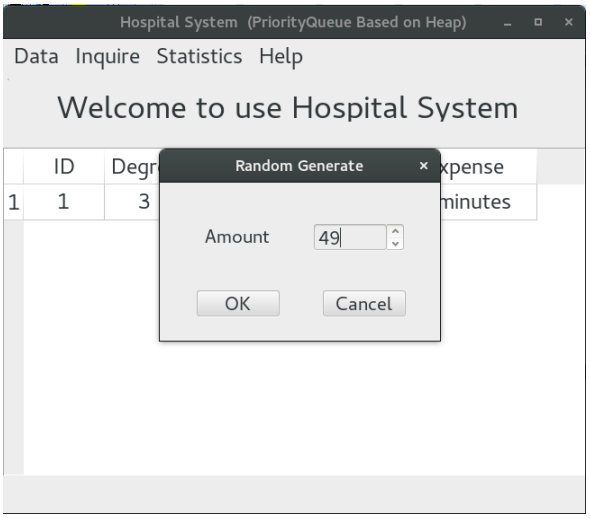


图4-2 Input窗口

图4-3 Input后主窗口

Random Generate:



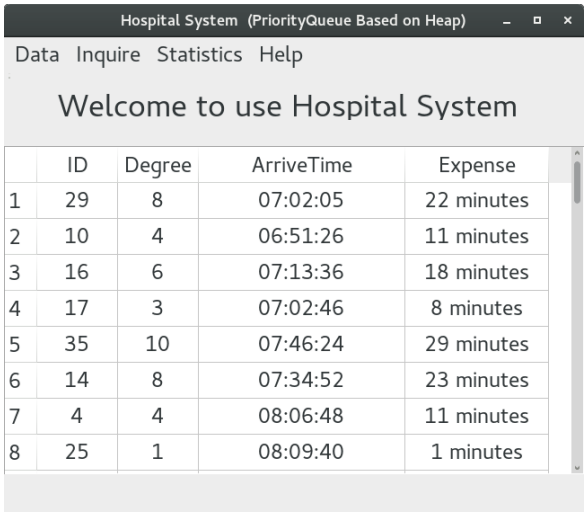


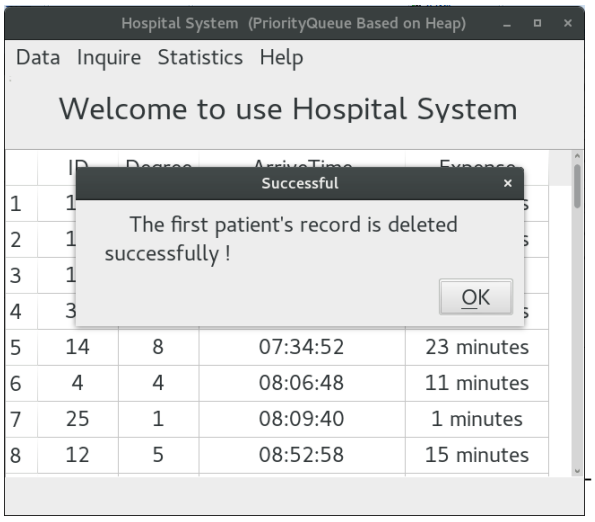
图4-4 Randm Generate窗口

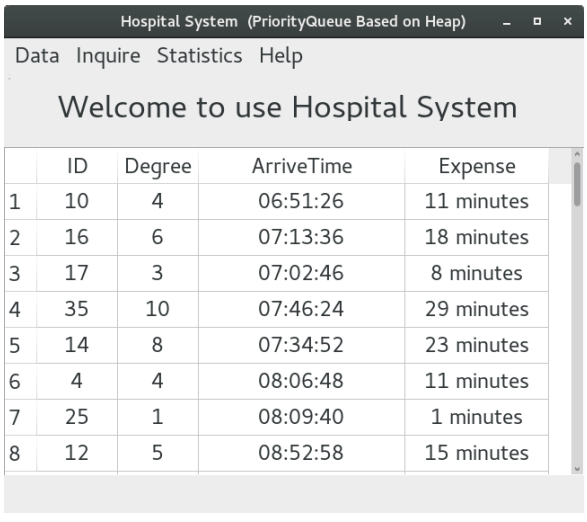
图4-5 Random Generate后主窗口

Delete:

图4-6 Delete窗口

图4-5 Delete后主窗口





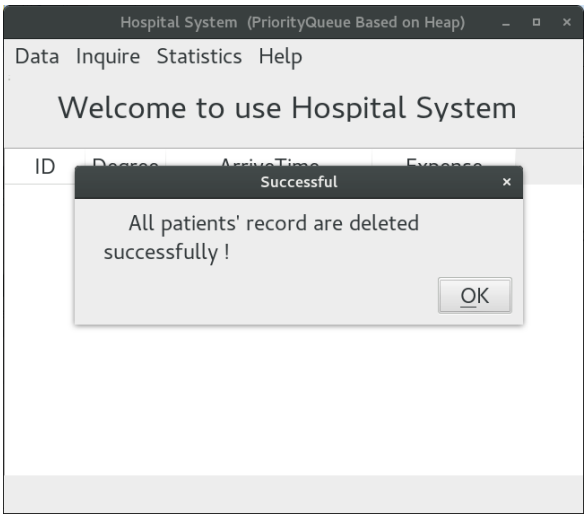


图4-8 Clear窗口

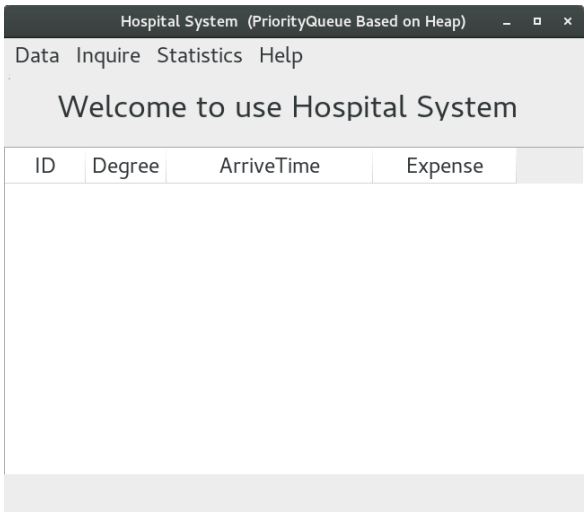


图4-9 Clear后主窗口

（2） 数据查询(Inquire)

该测试主要针对的菜单栏Inquire下的各项数据统计，包括“Is Full”(当前队列是否已存满)和“Is Empty”(当前队列是否为空)两个数据查询项。接下来将按顺序依次测试Is Empty、Is Full操作。(进行完Is Empty后再用Random Generate生成20个测试数据，再进行Is Full)

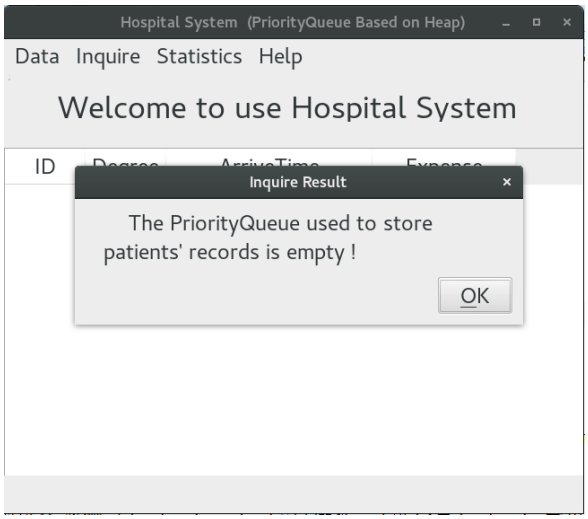


图4-10 Is Empty窗口

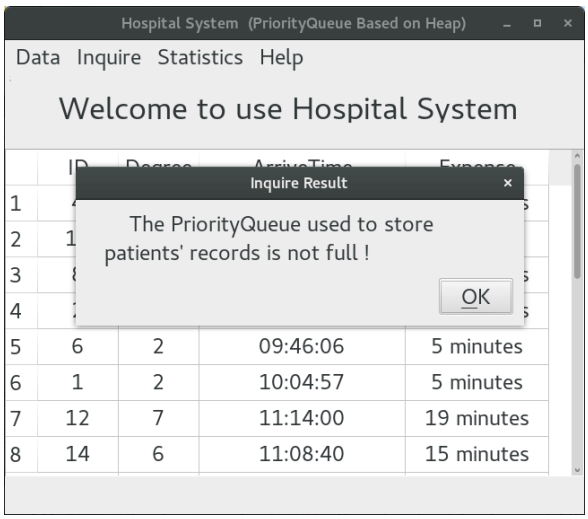


图4-11 Is Full窗口

（3） 数据统计(Statistics)

该测试主要针对的菜单栏Statistics下的各项数据统计，包括“One Wait Time”(查询某病人的等待时间)、“Average Wait Time”(队列中所有病人的平均等待时间)和“Longest Wait Time”(队列中所有病人中的最长等待时间)三个数据统计项。接下来将按顺序依次测试One Wait Time、Average Wait Time和Longest Wait Time操作。

One Wait Time：

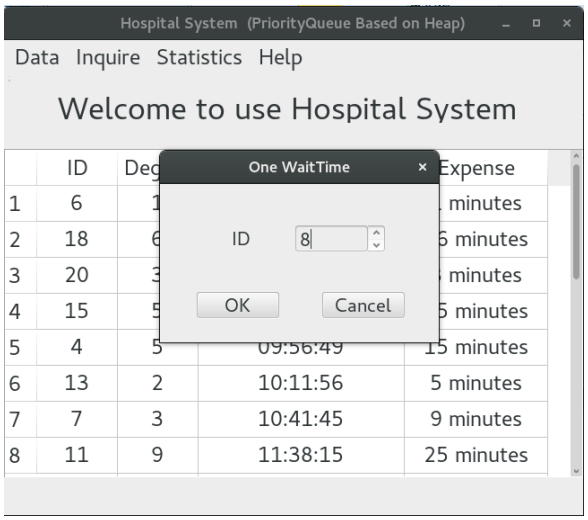


图4-12 One Wait Time窗口

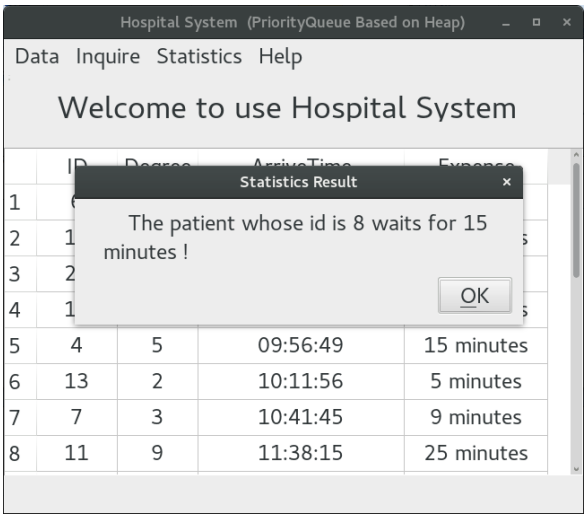


图4-13 One Wait Time结果窗口

Average Wait Time、Longest Wait Time：

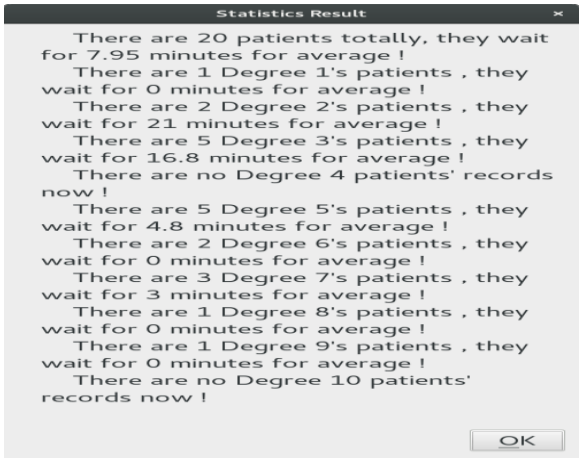


图4-14 Average Wait Time窗口

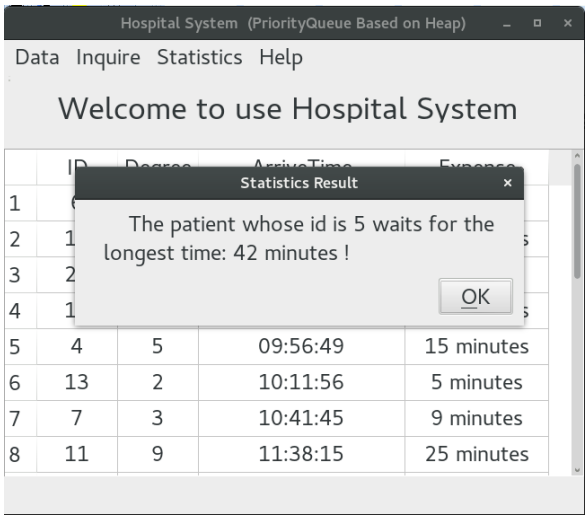


图4-15 Longest Wait Time窗口

在这里我们也可以进行仿真效果测试，即对程序通过Random Generate生成的数据与真实情况的相符性进行测试，为此，考虑到医生一天的工作时间和能够就诊的病人数，我们选取20个作为测试样本的容量，即每次利用Random Generate生成20个病人数据，再利用Average Wait Time统计平均等待时间。我们总共选取了10个样本，这10个样本的统计结果如表4-1所示。

次数	lv.1	lv.2	lv.3	lv.4	lv.5	lv.6	lv.7	lv.8	lv.9	lv10	Tol
1	15	30	13	12	10	9	7	8	0	0	8.9
2	11	27	10	32	7	0	0	5	0	1	8.1

3	29	13	27	18	7	12	9	7	2	0	7.6
4	21	22	19	19	0	9	0	0	0	1	9.2
5	19	16	21	21	8	7	0	2	3	0	8.5
6	30	31	29	24	5	10	6	3	0	0	9.1
7	0	20	15	25	11	8	7	0	3	0	9.4
8	12	16	12	17	4	6	4	0	2	2	7.8
9	7	19	16	14	7	0	2	0	0	0	8.2
10	19	25	24	21	9	3	5	3	1	1	7.2
Avg	18.7	22.3	20.1	21.2	7.8	6.9	5.6	4.1	2.2	1.1	9.0

表4-1 仿真效果测试

（4） 相关信息 (Statistics)

该测试主要针对的菜单栏Help下的各项相关信息，包括“About Program”(关于程序)和“About Me”(关于我)两个相关信息。

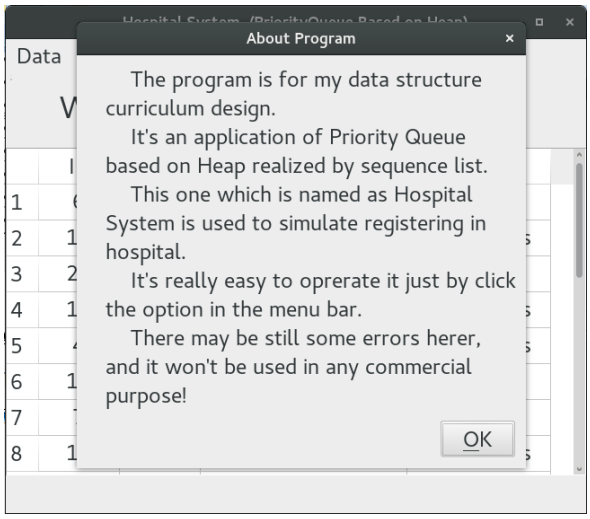


图4-16 About Program窗口

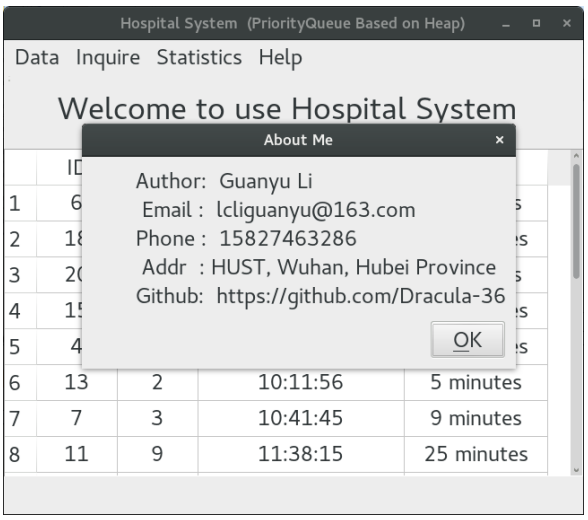


图4-17 About Me窗口

5 总结与展望

5.1 全文总结

通过这次数据结构的课程设计，我最终有所收获、有所改进、有所提升的方面有：

（1)掌握了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，同时加深了对数据结构应用性的认识。这次采用的基于堆实现的优先级队列让我对顺序表、堆和优先级队列的理解都有所深化，而且自己动手实现得到的学习效果远远大于仅仅观摩学习书本上的代码,只有亲自动手编写与课堂学习相结合才能掌握的更牢固。

（2）了解并能运用 C++的模板。在之前的数据结构实验课上，我的代码均是用 C 写的，但是这次为了能够使用优秀的 Qt 图形化库(提供了 C++的 API 而没有 C 的 API)，我决定改用 C++，在了解了 C++基本语法后，便开始琢磨如何借用 C++模板的特性使程序编写起来更系统、更可读、更方便。最后，在请教了相关同学后终于有了自己的思路。可以说，这次的数据结构课程设计，让我又学会了一门语言 C++，希望以后能进一步提升 C++代码的编写能力。

（3）了解并学习了 Qt 编程。之前 C 语言的课程设计我用的是 Gtk 来实现的图形化界面，感觉不甚方便，于是这次决定改用更加完善的 Qt，因此这次的程序开发也采用了 Qt 提供的 IDE——Qt Creater，一边学习文档，一边完成课设，最后的效果还是不错的。

5.2 学习展望

在今后的学习中，我将从以下两个方面进一步深化对数据结构的学习、运用和创新：

（1）不能学过就忘，要能真的把数据结构活学活用。在以后的开发过程中，编写前,要先深入思考程序中的各种数据应采用什么样的数据结构才能使得效率最高，但要能结合具体情况具体分析，不能一味地死套硬套。例如，基于链表和基于顺序表实现的线性表各有优势,当修改操作的使用频率更大时应该采用基于链表的顺序表，当查询操作的使用频率更大时则应该采用基于顺序表的线性表。

（2）应该注意数据结构与其他专业课的关联性，最好能结合其他专业课形

成自己的知识体系。在以后几个学期的学习中，专业课将会成为主要部分，应时刻考虑着数据结构是如何与其他专业课相关联的，要能够发现它们之间的共性，这样学习才能事半功倍。

6 体　会

此次数据结构设计让我感受颇深，不仅让我巩固了上个学期在数据结构课上的所学知识并且学习运用了新的语言和图形化库，还大大磨练了我的耐力和耐心，提高了我的编程能力。

纵观这次的数据结构课程设计，可谓收获颇丰。一方面，掌握了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，同时加深了对数据结构应用性的认识。这次采用的基于堆实现的优先级队列让我对顺序表、堆和优先级队列的理解都有所深化，而且自己动手实现得到的学习效果远远大于仅仅观摩学习书本上的代码，只有亲自动手编写与课堂学习相结合才能掌握的更牢固。另一方面，了解并能运用C++的模板。在之前的数据结构实验课上，我的代码均是用C写的，但是这次为了能够使用优秀的Qt图形化库(提供了C++的API而没有C的API)，我决定改用C++，在了解了C++基本语法后，便开始琢磨如何借用C++模板的特性使程序编写起来更系统、更可读、更方便。最后，在请教了相关同学后终于有了自己的思路。可以说，这次的数据结构课程设计，让我又学会了一门语言C++，希望以后能进一步提升C++代码的编写能力。除此之外，我还了解并学习了Qt编程。之前C语言的课程设计我用的是Gtk来实现的图形化界面，感觉不甚方便，于是这次决定改用更加完善的Qt，因此这次的程序开发也采用了Qt提供的IDE即Qt Creater，一边学习文档，一边完成课设。

在完成课程设计的过程中，也遇到了不少困难。比如，在对病人的挂号和就诊进行仿真时，选择什么作为每个病人就诊的优先级便让我思考了好长时间。为了解决这一问题，我进行了很多尝试，例如，改变优先级中病人的到达时间、病急程度以及就诊时间分别所占有的权重，然后随机生成一些数据统计最后的等待时间，考虑最合理的权重设置情况。整个过程及其耗费时间，极大地磨练了我的耐心和毅力。

这次课设让我意识到了数据结构的重大意义，因此在以后的开发和学习过程中应该注意，一方面，不能学过就忘，要能真的把数据结构活学活用。在以后的开发过程中，编写前，要先深入思考程序中的各种数据应采用什么样的数据结构才能使得效率最高，但要能结合具体情况具体分析，不能一味地死套硬套。例如，基于链表和基于顺序表实现的线性表各有优势，当修改操作的使用频率更大时应该采用基于链表的顺序表，当查询操作的使用频率更大时则应该采用基于顺序表

的线性表。另一方面，则应该注意数据结构与其他专业课的关联性，最好能结合其他专业课形成自己的知识体系。在以后几个学期的学习中，专业课将会成为主要部分，应时刻考虑着数据结构是如何与其他专业课相关联的，要能够发现它们之间的共性，这样学习才能效率更高。

综合来说，对数据结构的学习不能就此停止！只有通过不断的练习和实践，我们对数据结构的理解才能不断加深，对各种结构的运用才能更加灵活和得心应手！

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

代码结构：

Hospital

Hospital.pro

头文件：

- sqlist.h
- heap.h
- priority_queue.h
- patient.h
- mainwindow.h
- generatedialog.h
- inputdialog.h
- wtdialog.h

源文件：

- main.cpp
- sqlist.cpp
- heap.cpp
- priority_queue.cpp
- patient.cpp
- mainwindow.cpp
- generatedialog.cpp
- inputdialog.cpp
- wtdialog.cpp

ui 文件：

- mainwindow.ui
- generatedialog.ui
- inputdialog.ui
- wtdialog.ui

程序代码

Hospital.pro:

```
#-------------------------------------------------
#
# Project created by QtCreator 2016-02-22T20:59:03
#
#-------------------------------------------------

QT       += core gui

greaterThan(QT_MAJOR_VERSION, 4): QT += widgets

TARGET = Hospital
TEMPLATE = app

CONFIG += c++11
SOURCES += main.cpp\
        mainwindow.cpp \
    sqlist.cpp \
    patient.cpp \
    heap.cpp \
    priority_queue.cpp \
    inputdialog.cpp \
    generatedialog.cpp \
    wtdialog.cpp

HEADERS  += mainwindow.h \
    sqlist.h \
    patient.h \
    heap.h \
```

```
    priority_queue.h \
    inputdialog.h \
    generatedialog.h \
    wtdialog.h

FORMS       += mainwindow.ui \
    inputdialog.ui \
    generatedialog.ui \
    wtdialog.ui
```

sqlist.h:

```
#ifndef SQLIST_H
#define SQLIST_H

#include "patient.h"

#define INITSIZE 1000
#define INCREMENT 10
#define MAXSIZE 999

template<class T>
class Sqlist
{
public:
    T* elem;            //顺序表数组元素指针
    int len;            //当前顺序表长度
    int list_size;   //顺序表最大长度

    void list_init();                   //初始化顺序表
    void list_destroy();                //销毁顺序表
    void list_clear();                  //清空顺序表
```

```
    bool list_is_empty();                  //判断顺序表是否为空
    bool list_is_full();                   //判断顺序表是否为满
    int list_get_len();                    //获取顺序表长度
    void list_insert(int i, T t);      //在顺序表第 i 处插入元素
    void list_delete_elem(int i);      //删除顺序表第 i 处的元素
    T list_get_item(int i);                //返回顺序表第 i 处的元素

};

#endif // SQLIST_H
```

heap.h:

```
#ifndef HEAP_H
#define HEAP_H

#include "sqlist.h"

template<class T>
class Heap: public Sqlist<T> {
public:
    bool (*greater)(T t1, T t2);             //优先级比较函数

    Heap(bool (*func)(T t1, T t2));

    void heap_insert_elem(T t);              //向堆中插入元素
    T heap_get_top();                          //获取堆顶元素
    void heap_delete_top();                  //删除堆顶的元素

};

#endif // HEAP_H
```

priority_queue.h:

```
#ifndef _PRIORITY_QUEE_H
#define _PRIORITY_QUEE_H

#include "heap.h"

template<class T>
class PriorityQueue: public Heap<T> {
public:
    PriorityQueue(bool (*greater)(T, T));
    PriorityQueue(const PriorityQueue& pd);
    PriorityQueue(bool (*greater)(T, T), const PriorityQueue& pd);

    int size();                 //获取优先级队列长度
    void insert(T t);           //插入新元素
    void delete_elem();         //删除元素
    void clear_all();           //清空优先级队列
    void destroy();             //销毁优先级队列
    T top();                    //获取优先级队首元素
    bool empty();               //判断优先级队列是否为空
    bool full();                //判断优先级队列是否已满
};

#endif
```

patient.h:

```
#ifndef PATIENT_H
#define PATIENT_H

#include <queue>
#include <QTime>
#include <random>
#include <stdlib.h>
```

```
class Patient
{
public:
    Patient();
    Patient(int id, int degree, QTime arrive, int expense);

    void generate_data();

    static bool greater(Patient const p1, Patient const p2)
    {
        if(p1.arrive.secsTo(p2.arrive) > p1.expense * 60)
            return true;
        else
            return (p1.expense - p2.expense) * 60 * 0.7 > p2.arrive.secsTo(p1.arrive) * 0.3;
    }

    static bool later(Patient const p1, Patient const p2)
    {
        return p1.arrive.secsTo(p2.arrive) > 0;
    }

    int id;             //病人的序号
    int degree;         //病人的病急程度
    QTime arrive;       //病人的到达时间
    int expense;        //病人需要的就诊时间
};

#endif // PATIENT_H
```

mainwindow.h:

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QMainWindow>
#include <priority_queue.h>
#include <patient.h>
#include <QMessageBox>
#include <QImage>
#include <QStandardItemModel>

#include "inputdialog.h"
#include "generatedialog.h"
#include "wtdialog.h"

namespace Ui {
class MainWindow;
}

class MainWindow : public QMainWindow
{
    Q_OBJECT

public:
    explicit MainWindow(QWidget *parent = 0);
    ~MainWindow();

private slots:
    void generate();
    void random();
    void input();
    void delete_min();
    void clear();
    void full();
    void empty();
```

```
    void one_wt();
    void average_wt();
    void longest_wt();
    void about_program();
    void about_me();
    void display_table();

private:
    Ui::MainWindow *ui;
    QMessageBox* msgBox;
    QStandardItemModel* table;
    PriorityQueue<Patient> patients = PriorityQueue<Patient>(Patient::greater);
};

#endif // MAINWINDOW_H
```

generatedialog.h:

```
#ifndef GENERATEDIALOG_H
#define GENERATEDIALOG_H

#include <QDialog>

namespace Ui {
class GenerateDialog;
}

class GenerateDialog : public QDialog
{
    Q_OBJECT

public:
    explicit GenerateDialog(QWidget *parent = 0);
    ~GenerateDialog();
```

```
    int get_amount();

private:
    Ui::GenerateDialog *ui;
};

#endif // GENERATEDIALOG_H
```

inputdialog.h:

```
#ifndef INPUTDIALOG_H
#define INPUTDIALOG_H

#include <QDialog>
#include <patient.h>

namespace Ui {
class InputDialog;
}

class InputDialog : public QDialog
{
    Q_OBJECT

public:
    explicit InputDialog(QWidget *parent = 0);
    ~InputDialog();
    Patient get_input();

private:
    Ui::InputDialog *ui;
};

#endif // INPUTDIALOG_H
```

wtdialog.h:

```
#ifndef WTDIALOG_H
#define WTDIALOG_H

#include <QDialog>

namespace Ui {
class WTDialog;
}

class WTDialog : public QDialog
{
    Q_OBJECT

public:
    explicit WTDialog(QWidget *parent = 0);
    ~WTDialog();
    int get_id();

private:
    Ui::WTDialog *ui;
};

#endif // WTDIALOG_H
```

main.cpp:

```
#include "mainwindow.h"
#include <QApplication>

int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    MainWindow w;
```

```
    w.show();

    return a.exec();
}
```

sqlist.cpp:

```
#include <iostream>

#include "sqlist.h"
using namespace std;

template<class T>
void Sqlist<T>::list_init()
{
    elem = new T[INITSIZE];
    len = 0;
    list_size = INITSIZE;
    if(elem == NULL)
        return;
}

template<class T>
void Sqlist<T>::list_destroy()
{
    delete[] elem;
    elem = NULL;
}

template<class T>
void Sqlist<T>::list_clear()
{
    len = 0;
}
```

```
template<class T>
bool Sqlist<T>::list_is_empty()
{
	if (len == 0)
		return true;
	else
		return false;
}

template<class T>
bool Sqlist<T>::list_is_full()
{
	if (len == MAXSIZE)
		return true;
	else
		return false;
}

template<class T>
int Sqlist<T>::list_get_len()
{
	return len;
}

template<class T>
void Sqlist<T>::list_insert(int i, T t)
{
	if(i < 0 || i > len)
		return;

	if(len >= INITSIZE)	//原有数需表存储空间不够
	{
```

```
            T* temp = new T[list_size + INCREMENT];
            if(!temp)
                return;
            for(int j=0,k=0; k<len; ++j, ++k)
            { // 将 elem 中元素复制至 temp 中，同时插入新元素
                if (i == j)
                    temp[j] = t;
                else
                    temp[j] = elem[k];
            }
            delete[] elem;
            elem = temp;
            list_size += INCREMENT;
            len++;
        }
        else
        {
            for(int j=len-1; j>=i; ++j)
            {
                elem[j+1] = elem[j];
            }
            elem[i] = t;
            len++;
        }
}

template<class T>
void Sqlist<T>::list_delete_elem(int i)
{
        for(int j=i; j<len-1; ++j)
            elem[j] = elem[j+1];
        --len;
}
```

```
template<class T>
T Sqlist<T>::list_get_item(int i)
{
    return elem[i];
}
template class Sqlist<Patient>;
```

heap.cpp:

```
#include <iostream>

#include "heap.h"

using namespace std;

template<class T>
Heap<T>::Heap(bool (*func)(T, T)){
    greater = func;
    this->list_init();
}

template<class T>
void Heap<T>::heap_insert_elem(T t){
    this->list_insert(this->len, t);          //将新增元素插入顺序表，也就是堆底
    int child_index = this->len-1;
    int parent_index = (int)(child_index-1)/2;
    T temp = this->elem[child_index];
    while(child_index > 0){                   //通过循环向上调整堆
        if(greater(this->elem[parent_index], this->elem[child_index]))
            break;
        else{
            this->elem[child_index] = this->elem[parent_index];
            this->elem[parent_index] = temp;
```

```
                child_index = parent_index;
                parent_index = (int)(child_index-1)/2;
            }
        }
        this->elem[child_index] = temp;
}

template <class T>
T Heap<T>::heap_get_top() {
        return this->elem[0];
}

template <class T>
void Heap<T>::heap_delete_top() {
        this->elem[0] = this->elem[--this->len];        //删除堆顶元素并将最后一个元素存入堆顶


        int parent_index = 0;
        int child_index = 1;
        while(child_index  <  this->len){                    //通过循环为堆顶元素寻找合适位置

            if(child_index  <  this->len-1&&  greater(this->elem[child_index], this->elem[child_index+1]) == false)
                ++child_index;                                  //找出左右子节点中较大的子节点
            if(greater(this->elem[parent_index], this->elem[child_index]))
                break;                                          //双亲节点已经不小于子节点，调整完成
            else{                                               //交换双亲节点与较大子节点
```

```
                T temp = this->elem[parent_index];
                this->elem[parent_index] = this->elem[child_index];
                this->elem[child_index] = temp;
                parent_index = child_index;
                child_index = 2 * parent_index + 1;
            }
        }
}

template class Heap<Patient>;
```

priority_queue.cpp:

```
#include <iostream>

#include "priority_queue.h"

using namespace std;

template<class T>
PriorityQueue<T>::PriorityQueue(bool (*greater)(T, T)) :    Heap<T>(greater){
}

template<class T>
PriorityQueue<T>::PriorityQueue(const PriorityQueue &pd):
Heap<T>(pd.greater){

    this->len = pd.len;
    this->list_size = pd.list_size;
    this->elem = new T[this->list_size];
    for(int i=0; i < this->list_size; ++i)
        this->elem[i] = pd.elem[i];
}

template<class T>
PriorityQueue<T>::PriorityQueue(bool (*greater)(T, T), const PriorityQueue &pd):
Heap<T>(greater){

    this->len = pd.len;
```

```
    this->list_size = pd.list_size;
    this->elem = new T[this->list_size];
    for(int i=0; i < this->list_size; ++i)
        this->elem[i] = pd.elem[i];
}

template<class T>
void PriorityQueue<T>::insert(T t){
    //调用堆的插入
    this->heap_insert_elem(t);
}

template<class T>
void PriorityQueue<T>::delete_elem(){
    //调用堆的删除，删除堆顶元素
    this->heap_delete_top();
}

template<class T>
void PriorityQueue<T>::clear_all(){
    //调用线性表的清空
    this->list_clear();
}

template<class T>
void PriorityQueue<T>::destroy(){
    //调用线性表的删除
    this->destroy();
}

template<class T>
int PriorityQueue<T>::size(){
    return this->list_get_len();
}

template<class T>
T PriorityQueue<T>::top(){
    return this->heap_get_top();
}
```

```
template<class T>
bool PriorityQueue<T>::empty(){
    return this->list_is_empty();
}

 template<class T>
bool PriorityQueue<T>::full(){
    return this->list_is_full();
}

template class PriorityQueue<Patient>;
```

patient.cpp:

```
#include "patient.h"

#include <QDebug>
Patient::Patient()
{

}

Patient::Patient(int id, int degree, QTime arrive, int expense)
    :id(id), degree(degree), arrive(arrive), expense(expense)
{

}

void Patient::generate_data()
{
    std::default_random_engine e(rand() * time(NULL));
    // 按照均值为 12，方差为 4 的正太分布模拟小时（用数字模拟时间）
    std::normal_distribution<double> hour_distribution(12, 4);
    // 按照均值为 30，方差为 100 的正太分布模拟分和秒（用数字模拟时间）
    std::normal_distribution<double> min_sec_distribution(30, 100);
    // 按照均值为 1，方差为 1 的正太分布模拟严重程度
    std::normal_distribution<double> degree_distribution(4, 4);
    std::normal_distribution<double> expense_distribution(2, 1);

    int hour, min, sec, degree, expense;
    do{
```

```
        hour = std::lround(hour_distribution(e));
    } while(hour > 18 || hour < 6);
    do{
        min = std::lround(min_sec_distribution(e));
    } while(min >= 60 || min < 0);
    do{
        sec = std::lround(min_sec_distribution(e));
    } while(sec >= 60 || sec < 0);
    do{
        degree = std::lround(degree_distribution(e));
    } while(degree > 10 || degree < 1);
    do{
        expense = std::lround(expense_distribution(e));
    } while(expense > 3 || expense < 0);


    // 将时、分、秒拼接成字符串
    std::string time = std::to_string(hour) + std::string(":")
                                + std::to_string(min) + std::string(":")
                                + std::to_string(sec);
    this->arrive          =          QTime::fromString(QString::fromStdString(time),
QString("h:m:s"));
    this->degree = degree;
    this->expense = (degree - 1) * 3 + expense;
}
```

mainwindow.cpp:

```
#include "mainwindow.h"
#include "ui_mainwindow.h"

MainWindow::MainWindow(QWidget *parent) :
    QMainWindow(parent),
    ui(new Ui::MainWindow)
{
    ui->setupUi(this);
    setWindowTitle("Hospital System    (PriorityQueue Based on Heap)");

    QImage *img = new QImage;
    img->load("/home/dracula/Documents/IT_Study/data\
structure/final/Hospital/hospital.jpg");
```

```
    ui->picture->setPixmap(QPixmap::fromImage(* img));

    table = new QStandardItemModel(100, 4);
    ui->tableView->setVisible(0);

    connect(ui->actionRandom_Generate, SIGNAL(triggered()), this,
SLOT(random()));
    connect(ui->actionInput, SIGNAL(triggered()), this,  SLOT(input()));
    connect(ui->actionDelete, SIGNAL(triggered()), this,  SLOT(delete_min()));
    connect(ui->actionClear, SIGNAL(triggered()), this,  SLOT(clear()));
    connect(ui->actionIs_Full, SIGNAL(triggered()), this,  SLOT(full()));
    connect(ui->actionIs_Empty, SIGNAL(triggered()), this,  SLOT(empty()));
    connect(ui->actionOne_Wait_Time, SIGNAL(triggered()), this,
SLOT(one_wt()));
    connect(ui->actionAverage_Wait_Time, SIGNAL(triggered()), this,
SLOT(average_wt()));
    connect(ui->actionLongest_Wait_Time, SIGNAL(triggered()), this,
SLOT(longest_wt()));
    connect(ui->actionAbout_Program, SIGNAL(triggered()), this,
SLOT(about_program()));
    connect(ui->actionAbout_me, SIGNAL(triggered()), this,  SLOT(about_me()));
}

MainWindow::~MainWindow()
{
    delete ui;
}

void MainWindow::generate()
{
    Patient data;
    data.generate_data();
    data.id = patients.size() + 1;
    patients.insert(data);
}

void MainWindow::random()
{
    GenerateDialog dialog;
    if (dialog.exec() == GenerateDialog::Accepted) {
```

```
            int amount;
            amount = dialog.get_amount();
            for (int i = 0; i < amount; i++)
            {
                generate();
            }
            display_table();
        }
}

void MainWindow::input()
{
        InputDialog dialog;
        if (dialog.exec() == InputDialog::Accepted) {
            Patient data = dialog.get_input();
            patients.insert(data);
            display_table();
        }
}

void MainWindow::delete_min()
{
        msgBox = new QMessageBox();
        if (patients.size() > 0)
        {
            patients.delete_elem();
            display_table();
            msgBox->setWindowTitle("Successful");
            msgBox->setText("       The first patient's record is deleted successfully ! ");
        }
        else
        {
            msgBox->setWindowTitle("Failed");
            msgBox->setText("       There are no records for patients ! ");
        }
        QFont font;
        font.setPointSize(18);
        msgBox->setFont(font);
        msgBox->exec();
```

```
}

void MainWindow::clear()
{
    msgBox = new QMessageBox();
    if (patients.size() > 0)
    {
        patients.list_clear();
        display_table();
        msgBox->setWindowTitle("Successful");
        msgBox->setText("       All patients' record are deleted successfully ! ");
    }
    else
    {
        msgBox->setWindowTitle("Failed");
        msgBox->setText("       There are no records for patients ! ");
    }
    QFont font;
    font.setPointSize(18);
    msgBox->setFont(font);
    msgBox->exec();
}

void MainWindow::full()
{
    msgBox = new QMessageBox();
    msgBox->setWindowTitle("Inquire Result");
    if (patients.full())
        msgBox->setText("       The PriorityQueue used to store patients' records is
full ! ");
    else
        msgBox->setText("       The PriorityQueue used to store patients' records is
not full ! ");
    QFont font;
    font.setPointSize(18);
    msgBox->setFont(font);
    msgBox->exec();
}

void MainWindow::empty()
```

```
{
    msgBox = new QMessageBox();
    msgBox->setWindowTitle("Inquire Result");
    if (patients.empty())
        msgBox->setText("    The PriorityQueue used to store patients' records is
empty ! ");
    else
        msgBox->setText("    The PriorityQueue used to store patients' records is
not empty ! ");
    QFont font;
    font.setPointSize(18);
    msgBox->setFont(font);
    msgBox->exec();
}

void MainWindow::one_wt()
{
    int id;
    WTDialog dialog;
    if (dialog.exec() == WTDialog::Accepted)
        id = dialog.get_id();
    msgBox = new QMessageBox();
    msgBox->setWindowTitle("Statistics Result");

    Patient data;
    PriorityQueue<Patient> temp = PriorityQueue<Patient>(patients);
    QTime cur_time = temp.top().arrive;
    msgBox->setText(QString("    No patients whose id is %0 ! ").arg(id));
    for (int i = 0; i < patients.size(); i++)
    {
        data = temp.top();
        if (data.id == id)
            msgBox->setText(QString("    The patient whose id is %0 ").arg(id)
                    + QString("waits for %0 minutes !
").arg(data.arrive.secsTo(cur_time) / 60));
        temp.delete_elem();
        if (cur_time.addSecs(data.expense * 60) > temp.top().arrive)
            cur_time = cur_time.addSecs(data.expense * 60);
        else
            cur_time = temp.top().arrive;
```

```
    }
    QFont font;
    font.setPointSize(18);
    msgBox->setFont(font);
    msgBox->exec();
}

void MainWindow::average_wt()
{
    msgBox = new QMessageBox();
    msgBox->setWindowTitle("Statistics Result");

    Patient data;
    PriorityQueue<Patient> temp = PriorityQueue<Patient>(patients);
    QTime cur_time = temp.top().arrive;
    int wait_time = 0;
    int degree_wt[10] = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0};
    int degree_size[10] = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0};
    QString output;
    for (int i = 0; i < patients.size(); i++)
    {
        data = temp.top();
        wait_time += data.arrive.secsTo(cur_time) / 60;
        degree_wt[data.degree - 1] += data.arrive.secsTo(cur_time) / 60;
        degree_size[data.degree - 1] += 1;
        temp.delete_elem();
        if (cur_time.addSecs(data.expense * 60) > temp.top().arrive)
            cur_time = cur_time.addSecs(data.expense * 60);
        else
            cur_time = temp.top().arrive;
    }
    if (patients.size() == 0)
        output = QString("      There are no patients' records now !\n");
    else
        output = QString("      There are %0 patients totally, they wait for %1
minutes for average !\n")
                    .arg(patients.size()).arg((float)wait_time / patients.size());
    for (int i = 0; i < 10; i++)
    {
        if (degree_size[i] == 0)
```

```
                output += QString("      There are no Degree %0 patients' records now !\n").arg(i + 1);
            else
                output += QString("      There are %0 Degree %1's patients , they wait for %2 minutes for average !\n")
                        .arg(degree_size[i]).arg(i + 1).arg((float)degree_wt[i] / degree_size[i]);
        }
        msgBox->setText(output);
        QFont font;
        font.setPointSize(18);
        msgBox->setFont(font);
        msgBox->exec();
}

void MainWindow::longest_wt()
{
        msgBox = new QMessageBox();
        msgBox->setWindowTitle("Statistics Result");

        Patient data;
        PriorityQueue<Patient> temp = PriorityQueue<Patient>(patients);
        QTime cur_time = temp.top().arrive;
        int wait_time = 0;
        int id = 1;
        for (int i = 0; i < patients.size(); i++)
        {
            data = temp.top();
            if (wait_time < data.arrive.secsTo(cur_time) / 60)
            {
                wait_time = data.arrive.secsTo(cur_time) / 60;
                id = data.id;
            }
            temp.delete_elem();
            if (cur_time.addSecs(data.expense * 60) > temp.top().arrive)
                cur_time = cur_time.addSecs(data.expense * 60);
            else
                cur_time = temp.top().arrive;
        }
        if (patients.size() != 0)
```

```
            msgBox->setText(QString("       The patient whose id is %0 ").arg(id) +
                            QString("waits for the longest time: %0 minutes ! ").arg(wait_time));
        else
            msgBox->setText("       There are no records for patients ! ");
        QFont font;
        font.setPointSize(18);
        msgBox->setFont(font);
        msgBox->exec();
}

void MainWindow::about_program()
{
        msgBox = new QMessageBox();
        msgBox->setWindowTitle("About Program");
        msgBox->setText(QString("       The program is for my data structure curriculum design.\n"
                                "       It's an application of Priority Queue based on Heap realized by sequence list.\n"
                                "       This one which is named as Hospital System is used to simulate registering in hospital.\n"
                                "       It's really easy to oprerate it just by click the option in the menu bar.\n"
                                "       There may be still some errors herer, and it won't be used in any commercial purpose!"));
        QFont font;
        font.setPointSize(18);
        msgBox->setFont(font);
        msgBox->exec();
}

void MainWindow::about_me()
{
        msgBox = new QMessageBox();
        msgBox->setWindowTitle("About Me");
        msgBox->setText(QString("       Author:    Guanyu Li\n"
                                "       Email :    lcliguanyu@163.com\n"
                                "       Phone :    15827463286\n"
                                "       Addr    : HUST, Wuhan, Hubei Province\n"
                                "       Github:    https://github.com/Dracula-36"));
```

```
    QFont font;
    font.setPointSize(18);
    msgBox->setFont(font);
    msgBox->exec();
}

void MainWindow::display_table()
{
    table->clear();

    table->setRowCount(patients.size());
    table->setColumnCount(4);
    ui->tableView->setVisible(1);

    table->setHeaderData(0, Qt::Horizontal, "ID");
    table->setHeaderData(1, Qt::Horizontal, "Degree");
    table->setHeaderData(2, Qt::Horizontal, "ArriveTime");
    table->setHeaderData(3, Qt::Horizontal, "Expense");
    ui->tableView->setModel(table);
    ui->tableView->setEditTriggers(QAbstractItemView::NoEditTriggers);
    ui->tableView->setColumnWidth(0, 90);
    ui->tableView->setColumnWidth(1, 90);
    ui->tableView->setColumnWidth(2, 230);
    ui->tableView->setColumnWidth(3, 160);

    PriorityQueue<Patient> displays = PriorityQueue<Patient>(patients);
    Patient data;
    QStandardItem *item;
    for (int i = 0; i < patients.size(); i++)
    {
        data = displays.top();
        displays.delete_elem();
        item = new QStandardItem(QString("%0").arg(data.id));
        item->setTextAlignment(Qt::AlignCenter);
        table->setItem(i, 0, item);
        item = new QStandardItem(QString("%0").arg(data.degree));
        item->setTextAlignment(Qt::AlignCenter);
        table->setItem(i, 1, item);
        item = new QStandardItem(QString("%0").arg(data.arrive.toString()));
        item->setTextAlignment(Qt::AlignCenter);
```

```
            table->setItem(i, 2, item);
            item = new QStandardItem(QString("%0 minutes").arg(data.expense));
            item->setTextAlignment(Qt::AlignCenter);
            table->setItem(i, 3, item);
        }

}
```

generatedialog.cpp:

```
#include "generatedialog.h"
#include "ui_generatedialog.h"

GenerateDialog::GenerateDialog(QWidget *parent) :
    QDialog(parent),
    ui(new Ui::GenerateDialog)
{
    ui->setupUi(this);
    this->setWindowTitle("Random Generate");

    ui->spinBox->setMinimum(1);
    ui->spinBox->setMaximum(999);

    connect(ui->okButton, SIGNAL(released()), this, SLOT(accept()));
    connect(ui->cancelButton, SIGNAL(released()), this, SLOT(reject()));
}

GenerateDialog::~GenerateDialog()
{
    delete ui;
}

int GenerateDialog::get_amount()
{
    return ui->spinBox->value();
}
```

inputdialog.cpp:

```
#include "inputdialog.h"
#include "ui_inputdialog.h"
```

```
InputDialog::InputDialog(QWidget *parent) :
    QDialog(parent),
    ui(new Ui::InputDialog)
{
    ui->setupUi(this);
    this->setWindowTitle("Patient Data Input");

    ui->spinBox_ID->setMinimum(1);
    ui->spinBox_ID->setMaximum(999);

    ui->spinBox_Degree->setMinimum(1);
    ui->spinBox_Degree->setMaximum(10);

    ui->spinBox_Expense->setMinimum(0);
    ui->spinBox_Expense->setMaximum(300);
    ui->spinBox_Expense->setSuffix(" minutes");

    connect(ui->okButton, SIGNAL(released()), this, SLOT(accept()));
    connect(ui->cancelButton, SIGNAL(released()), this, SLOT(reject()));

}

InputDialog::~InputDialog()
{
    delete ui;
}

Patient InputDialog::get_input()
{
    Patient data(ui->spinBox_ID->value(), ui->spinBox_Degree->value(),
                 ui->timeEdit->time(), ui->spinBox_Expense->value());
    return data;
}
```

wtdialog.cpp:

```
#include "wtdialog.h"
#include "ui_wtdialog.h"

WTDialog::WTDialog(QWidget *parent) :
    QDialog(parent),
```

```
    ui(new Ui::WTDialog)
{
    ui->setupUi(this);
    this->setWindowTitle("One WaitTime");

    ui->spinBox->setMinimum(1);
    ui->spinBox->setMaximum(999);

    connect(ui->okButton, SIGNAL(released()), this, SLOT(accept()));
    connect(ui->cancelButton, SIGNAL(released()), this, SLOT(reject()));
}

WTDialog::~WTDialog()
{
    delete ui;
}

int WTDialog::get_id()
{
    return ui->spinBox->value();
}
```

注意：ui 文件需用 QtCreater 打开，这里就不贴出源代码了，具体文件见随文档的文件夹内