

华中科技大学

# 课程设计报告

题目： 楼盘查询系统

---

课程名称： C 语言程序设计

专业班级： 信息安全 1101 班

学 号： u201114051

姓 名： 周乐

指导教师： 周时阳

报告日期： 2012 年 10 月 16 日

计算机科学与技术学院

# 《C 语言程序设计》课程设计

## 任务书

### 题目(一) 楼盘查询系统

#### (1) 主要内容

建立楼盘楼盘信息系统，提供创建、编辑和综合查询等基本业务管理和服务。

#### (2) 任务要求

收集与阅读相关文献资料，确定系统目标与范围，分析系统需求，确定系统功能；

设计系统方案，完成系统实现；提交《课程设计报告》。

#### (3) 参考文献

[1]曹计昌,卢萍,李开. C 语言程序设计,北京：科学出版社,2008

[2]张引. C 程序设计基础课程设计, 杭州：浙江大学出版社,2007

[3]黄明,梁旭,万洪莉. C 语言课程设计,北京：电子工业出版社,2006

### 题目和难度选择的规定

#### 1、题目选择的规定

学号尾数（最后一位）为：1、4、7、0 者自动选择题目(一)楼盘查询系统。

学号尾数（最后一位）为：2、5、8 者自动选择题目(二)招生查询系统。

学号尾数（最后一位）为：3、6、9 者自动选择题目(三)景点查询系统。

#### 2、难度选择的规定

（1）选三个方向的十字交叉链表数据结构的难度系数为 1；选二个方向十字交叉链表数据结构的难度系数为 0.85。

（2）采用文本菜单界面的难度系数为 1；采用教材 p215-p216 类似的简易菜单的难度系数为 0.85。

请每个同学根据实际掌握程度和能力选择相应难度系数的题，要求能够独立完成。

## 基本要求

- 1、只能使用 C 语言，源程序要有适当的注释，使程序容易阅读。
- 2、要有用户界面。要求至少采用教材 p215-p216 类似的简易菜单；鼓励采用文本菜单界面甚至采用图形菜单界面。
- 3、必须使用结构和十字交叉链表等数据结构。
- 4、使用文件保存数据。
- 5、至少输出一份报表（屏幕输出即可），鼓励自行增加新功能。
- 6、设计测试的模拟数据，完成系统测试。
- 7、写实验报告（要求正规打印，A4 幅面），内容包括：

题目

系统功能模块结构图

数据结构设计及用法说明

程序结构（画流程图）

各模块的功能

试验结果（包括输入数据和输出结果）

体会

参考文献

附录：程序清单及源程序软盘

- 7、凡发现抄袭，抄袭者与被抄袭者皆以零分计入本课程设计成绩并向学院报告。凡发现实验报告或源程序雷同，涉及的全部人员皆以零分计入本课程设计成绩并向学院报告。
- 8、课程设计报告封面统一格式，标准格式见附件。

# 目录

- 任务书..... 1
- 一 选题背景..... 5
  - 1.1 课程设计课题的目的与意义 ..... 5
  - 1.2 本题目的研究现状 ..... 5
  - 1.3 核心问题..... 5
  - 1.4 技术难点..... 5
- 二 方案论证..... 6
  - 2.1 方案综述..... 6
  - 2.2 设计方案..... 6
  - 2.3 方案分析..... 6
  - 2.4 方案特点..... 7
- 三 过程论述..... 7
  - 3.1 界面设计..... 7
  - 3.2 数据结构与算法设计 ..... 8
  - 3.3 功能实现..... 13
  - 3.4 功能测试..... 13
- 四 结果分析..... 14
  - 4.1 实际界面..... 14

4.2 实现的功能与分析 .....	16
五 总结.....	16
六 参考文献.....	17
七 附录.....	17
7.1 技术文档.....	17
7.2 源程序代码.....	19
\main.c.....	19
\core.h .....	21
\core_io.h.....	24
\core_EST.h .....	29
\core_BDN.h .....	36
\core_CEL.h .....	43
\core_query.h .....	52

# 一 选题背景

## 1.1 课程设计课题的目的与意义

在我国建设中国特色社会主义市场经济的大时代背景下,我国各大产业发展迅猛,尤其是房地产行业。不管是一线、二线还是三线城市,房地产都成为一个炙手可热的“香饽饽”。因为各地的各种房地产信息鳞次栉比,如汗牛充栋,消费者在购房的时候经常会面临找不到地段、户型、价位都适合自己的商品房的困境。这也就是本选题的目的,建立一套楼盘信息管理系统,使购房者能够方便、快捷地找到适合自己的商品房。

## 1.2 本题目的研究现状

目前,由于充斥于社会中额信息越来越多,无论是什么公司或组织,一般都会有一套类似于信息管理系统的平台。可以说,此类研究还是较为成熟的。

## 1.3 核心问题

本楼盘信息管理系统的核心问题包括数据的存储结构,数据显示与编辑模式,以及数据的查询方法。由于本系统的功能很大程度依赖于对于数据的读写,数据的存储结构直接关系到整个系统的运行效率。因为本系统是为普通的,没有太多计算机应用经验的用户设计的,本系统的数据显示与编辑模式也是非常重要的内容。因此,本系统应当采用一种图形化的,与普通 windows 应用程序类似的界面,而不是控制台界面。另外,由于此类信息的量都非常庞大,查询与检索就显得非常重要了。

## 1.4 技术难点

本系统的技术难点主要在以下几个方面:图形化界面,数据存储结构,以及如何进行高效的查询。由于 C 本身对于图形化界面的支持不佳,需要通过许多复杂的函数与功能才能做到在控制台下模拟图形界面,我选择 GTK+ 来作为本系统的图形化解决方案。为了能更好地体现数据的层次结构与每条数据之间的关系,我使用双向十字交叉链表将磁盘上读取的数据存储到内存中。对于查询功能,我给了用户多种方案可供选择以得出用户希望的查询结果。

## 二 方案论证

### 2.1 方案综述

本系统的主要实现方案是：使用 GTK+构建图形化界面，编写相应的事件响应函数，同时使用标准的 C 语言进行底层数据结构的读取与存储。

### 2.2 设计方案

本系统的设计方案主要分为以下几个方面：

- 2.2.1 从文件读取数据并以双向十字交叉链表的形式存储与内存中；
- 2.2.2 从链表中获取数据并显示（选用 GTK+中 TreeView 控件与 Tree 模型）；
- 2.2.3 编辑数据并同时改变链表中的相应值；
- 2.2.4 通过关键词等条件对链表数据进行检索，并显示检索结果；
- 2.2.5 将链表中的数据保存在磁盘中。

### 2.3 方案分析

选用该方案的原因主要有以下几点：

- 2.3.1 GTK+是一个应用非常广泛的图形化工具包，它提供了一套完整的图形化控件，适用于各类应用程序的开发。GTK+能够加速项目的开发速度，同时也能得到非常美观的 windows 界面。GTK+被广泛地应用于 Windows, GNU/Linux and Unix, OSX,甚至是移动终端上的应用程序开发。此外，GTK+中提供的事件、信号处理机制也非常的成熟与强大。
- 2.3.2 受目前在互联网技术中广泛使用的 XML 技术的启发，本系统使用双向十字交叉链表在内存中存储数据。选用这种方法的优点是内存中的数据结构层次分明有条理。其特点有：每个节点都有指向前一个节点与后一个节点的指针，使对于数据的增加与删除操作更加便捷，同时也方便查询；较深层次的链表节点中均有指向上一级链表节点的指针，这样能够很方便地查询子节点的父亲节点与祖父节点的信息。
- 2.3.3 GTK+中提供了 TreeView 控件和与之搭配的树模型。TreeView 能够直观地以列表的形式显示数据，同时通过在它的单元格改变事件上绑定编辑函数，用户

修改 TreeView 控件中的数据时相应的链表节点能够同时得到修改。这样不管是查看还是增加、修改、删除数据，用户都能非常直观地进行操作。

2.3.4 为了避免错误，存储时把内存中的所有数据都进行了存储。在数据量大的情况下应改为仅存储被修改的数据。

2.3.5 由于控制台下的图形化界面不美观，操作复杂，用户友好度差，实现方法复杂且容易加重 CPU 负担，故未选择《C 语言程序设计》上介绍的控制台图形化界面。

## 2.4 方案特点

2.4.1 界面美观，易于使用；

2.4.2 可在查看数据的同时进行编辑，方便快捷；

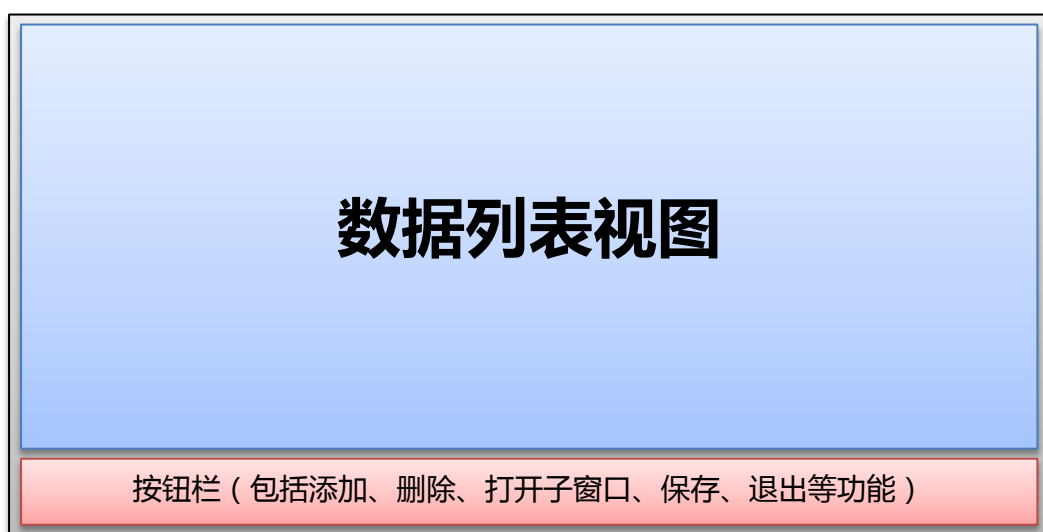
2.4.3 查看每个子节点的内容时可以轻易了解它的父节点的相关信息，简洁直观；

2.4.4 查询功能有多种模式可供选择，灵活易用。

# 三 过程论述

## 3.1 界面设计

3.1.1 在正式开始代码的编写过程之前，对程序界面原型设计如下：

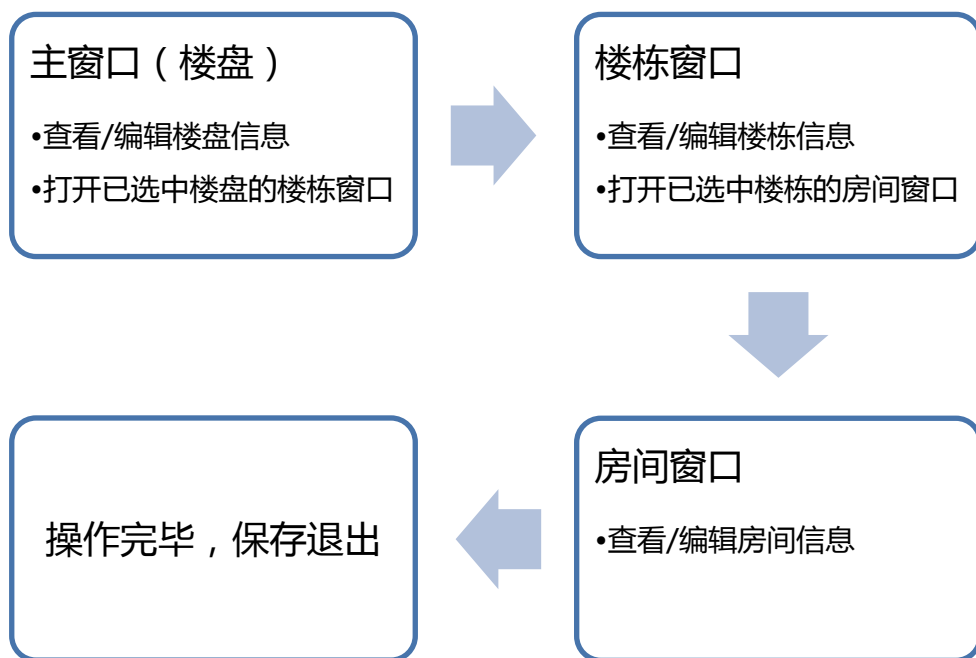


3.1.2 设计了多个窗口，每个窗口显示一种类型的数据（如楼盘、楼栋、房间）。



3.1.3 每个窗口由表格视图与功能按钮组成。

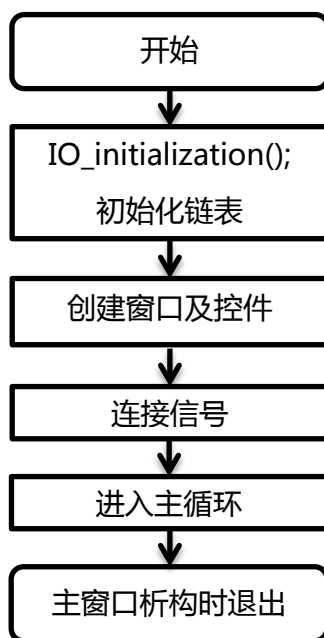
3.1.4 界面的操作流程设计如下：



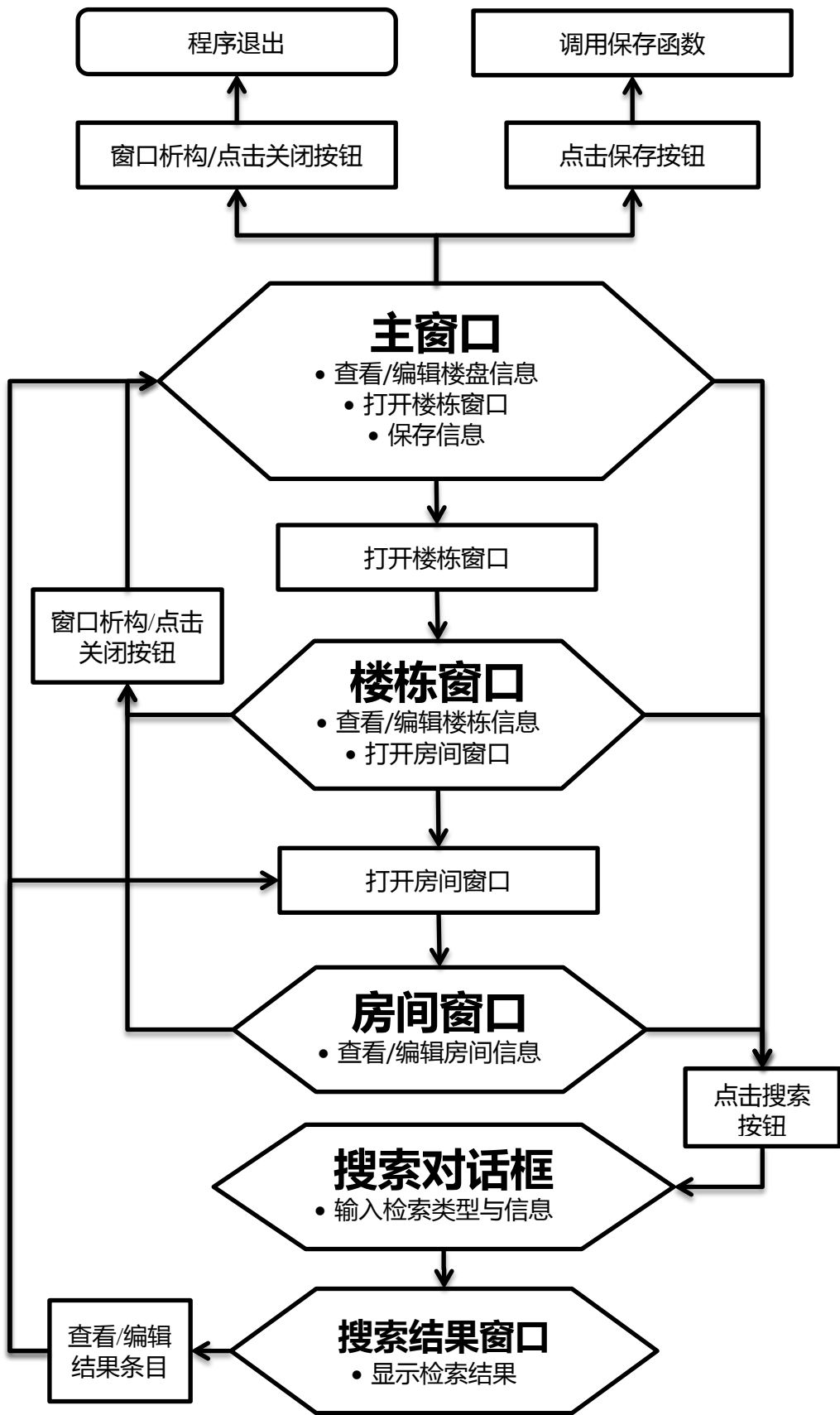
## 3.2 数据结构与算法设计

3.2.1 除主程序部分外，程序模块主要分为以下几个部分，分别使用下列头文件编写：core.h，包括基础结构类型与枚举类型还有全局变量的声明，及数个通用表格操作函数；core\_io.h，包括对磁盘上的数据的读取与写入功能；core\_EST.h，包括楼盘视图的相关显示和操作函数；core\_BDN.h，包括楼栋视图的相关显示和操作函数；core\_CEL.h，包括房间视图的相关显示和操作函数；core\_query.h，包括查询相关函数和查询视图相关显示和操作函数。

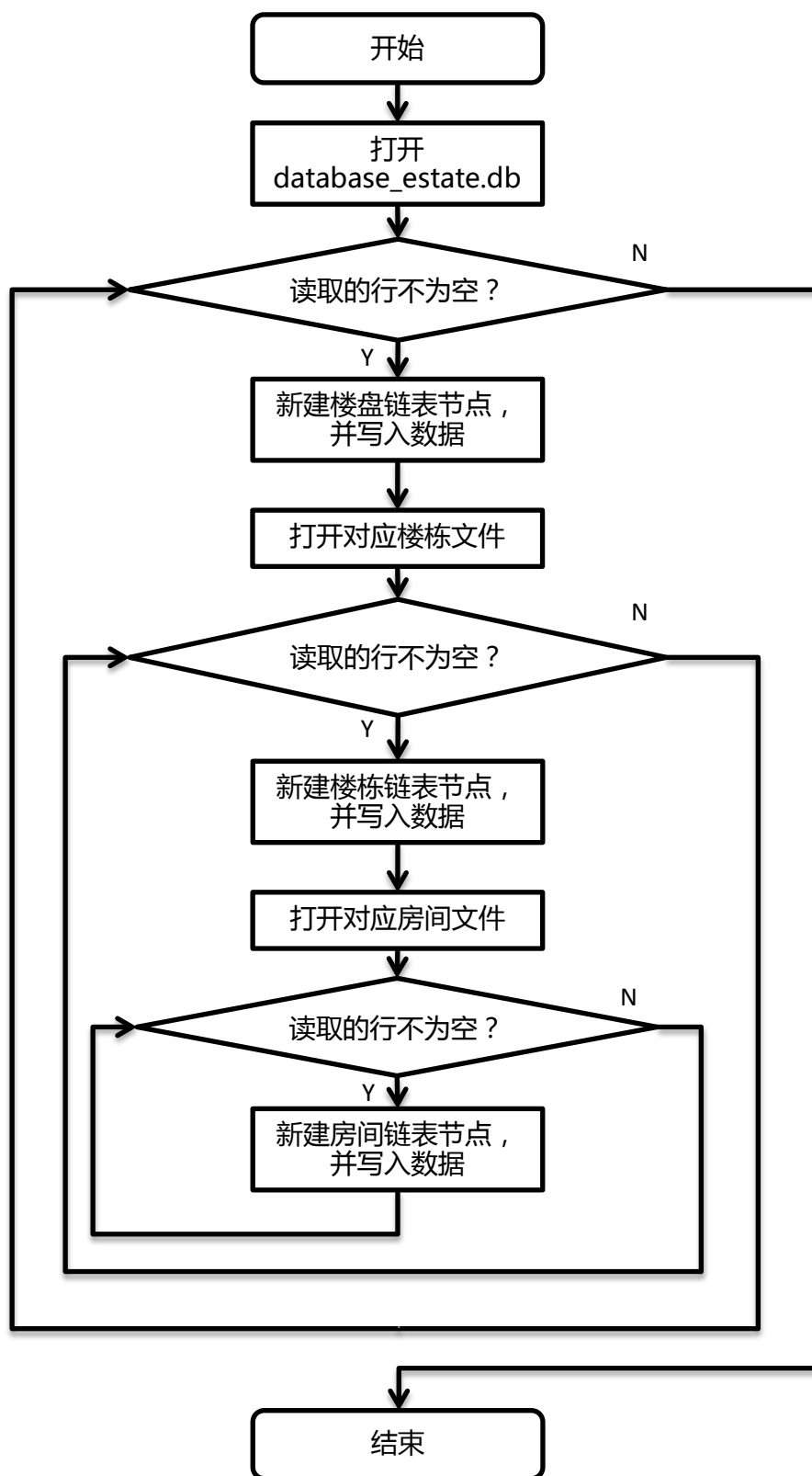
3.2.2 main 函数执行流程图如下：



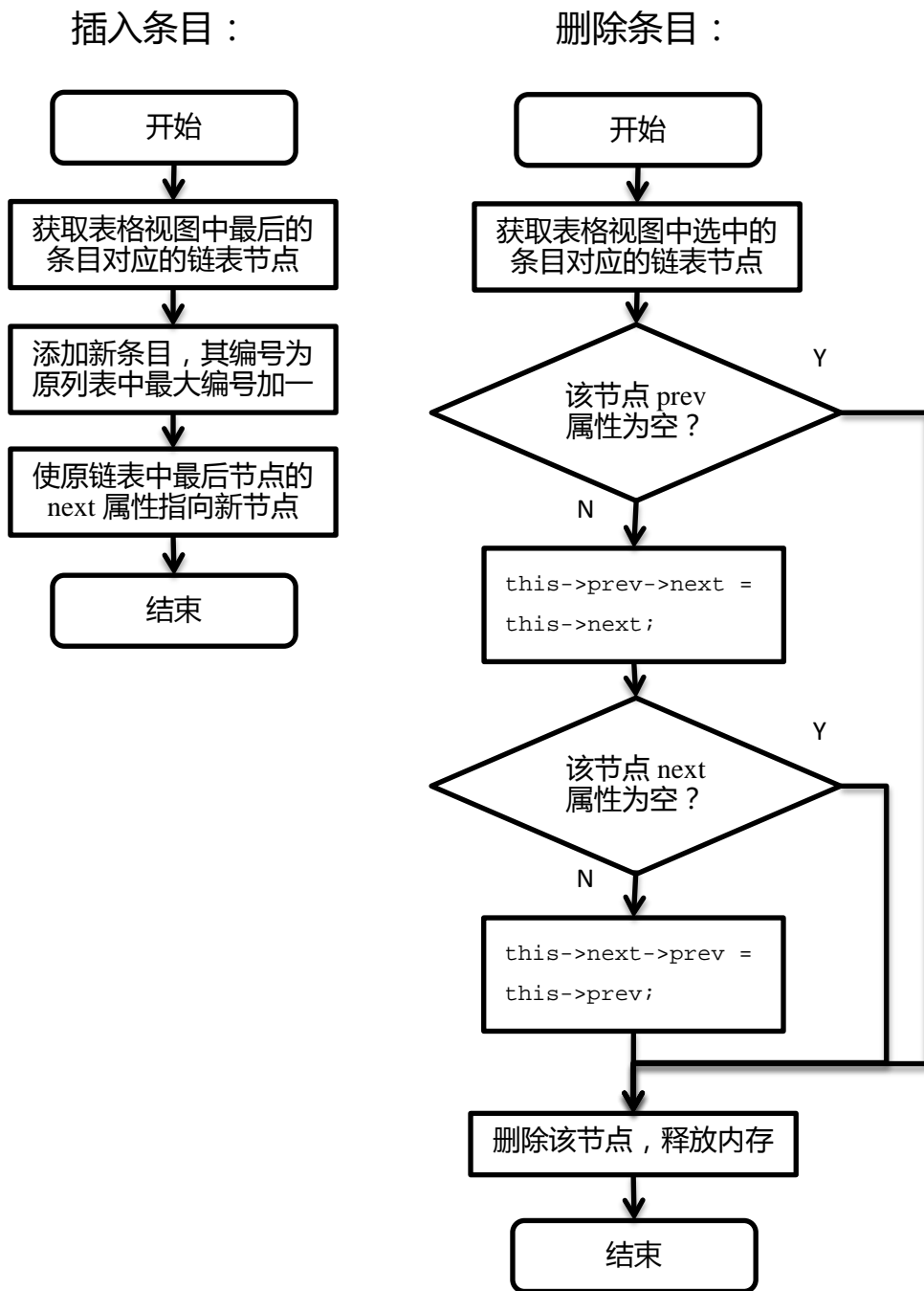
3.2.3 系统功能模块结构示意图如下：



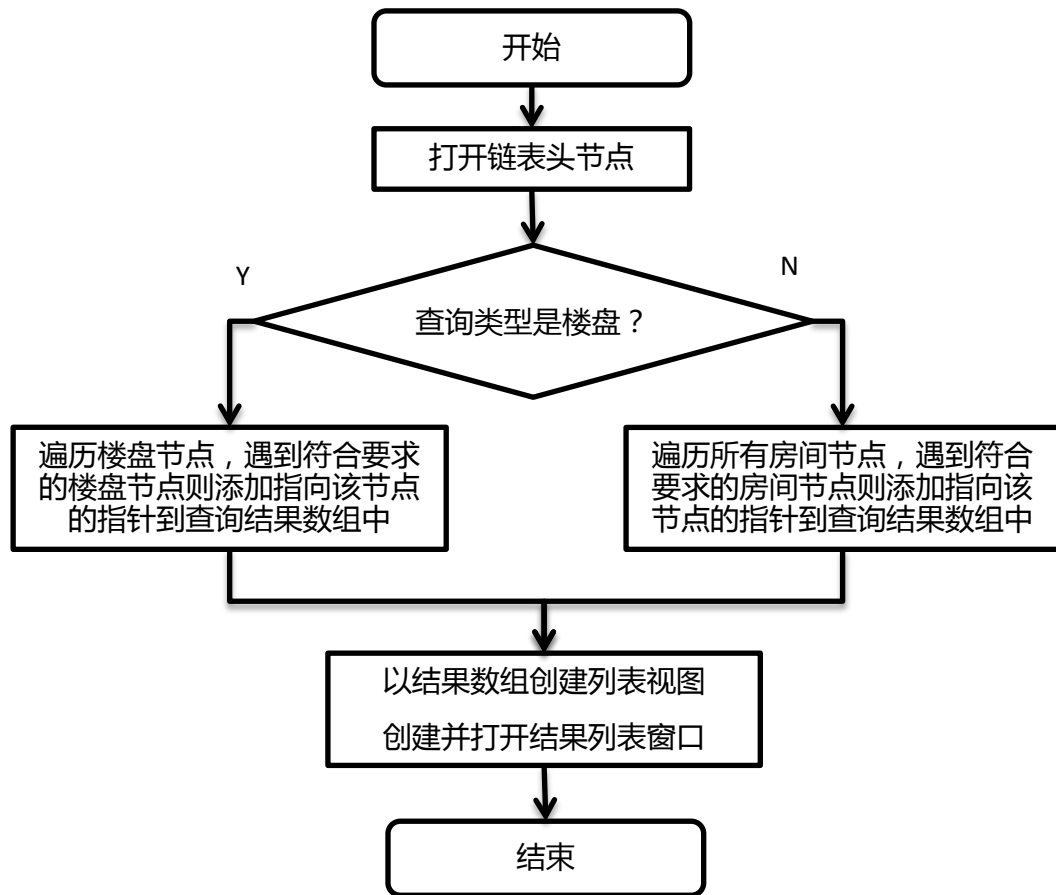
3.2.4 链表初始化算法流程图如下：



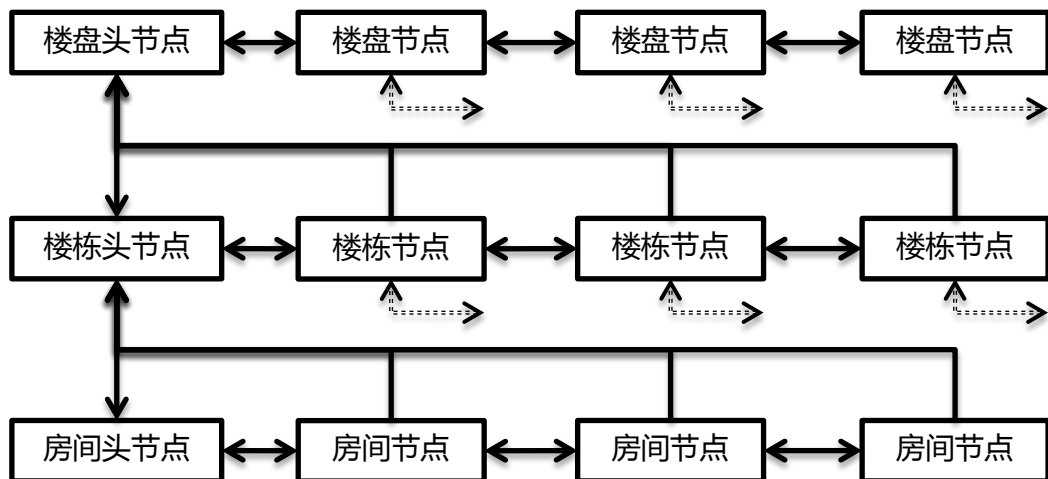
3.2.5 插入与删除条目算法流程图如下：



3.2.6 查询算法流程图如下：



3.2.7 链表结构示意图如下：



### 3.3 功能实现

- 3.3.1 GTK+工具包与 Codeblocks IDE 的配置。安装 Codeblocks IDE ,并创建 GTK+项目。  
在向导中选择 GTK+工具包的路径。测试运行 GTK+范例程序。
- 3.3.2 核心数据结构的声明。包括楼盘、楼栋、房间结构体的声明及全局变量的声明。
- 3.3.3 主窗口控件部分的代码的书写（ main ）包括新建窗口，向窗口中加入容器控件，将列表视图、按钮等控件加入容器，按钮、列表视图信号与回调函数的连接，及全部控件准备完备之后显示窗口及其子控件。
- 3.3.4 楼盘、楼栋、房间逻辑功能部分的代码的书写；楼栋、房间窗口控件部分的代码的书写。包括单元格编辑结束之后修改链表需要的回调函数（ EST\_cell\_edited, BDN\_cell\_edited, CEL\_cell\_edited ），添加/删除条目相关函数（ EST\_add\_item, BDN\_add\_item, CEL\_add\_item, EST\_remove\_item, BDN\_remove\_item, CEL\_remove\_item ），创建并添加列表模型函数（ EST\_create\_and\_fill\_model, BDN\_create\_and\_fill\_model, CEL\_create\_and\_fill\_model ），创建列表视图函数（ EST\_create\_view\_and\_model, BDN\_create\_view\_and\_model, CEL\_create\_view\_and\_model ），返回窗口控件指针的新建窗口函数（ BDN\_new\_window, CEL\_new\_window ），及初始化数据结构并打开新窗口的函数（ BDN\_open\_window, CEL\_open\_window ）。
- 3.3.5 IO 功能代码的书写。包括读取文件并创建链表的初始化函数（ IO\_initialization ），及将链表保存为文件的函数（ IO\_save ）。
- 3.3.6 查询功能代码的书写。包括主查询函数（ QRY\_main ），查询结果列表模型的创建函数（ QRY\_create\_and\_fill\_model ），查询结果列表视图的创建函数（ QRY\_create\_view\_and\_model ），查询结果窗口的创建函数（ QRY\_new\_result\_window ），初始化查询结果并打开窗口的函数（ QRY\_open\_result\_window ），及根据列表中选中的条目打开新窗口查看或编辑其详细的函数（ QRY\_view\_entry ）。

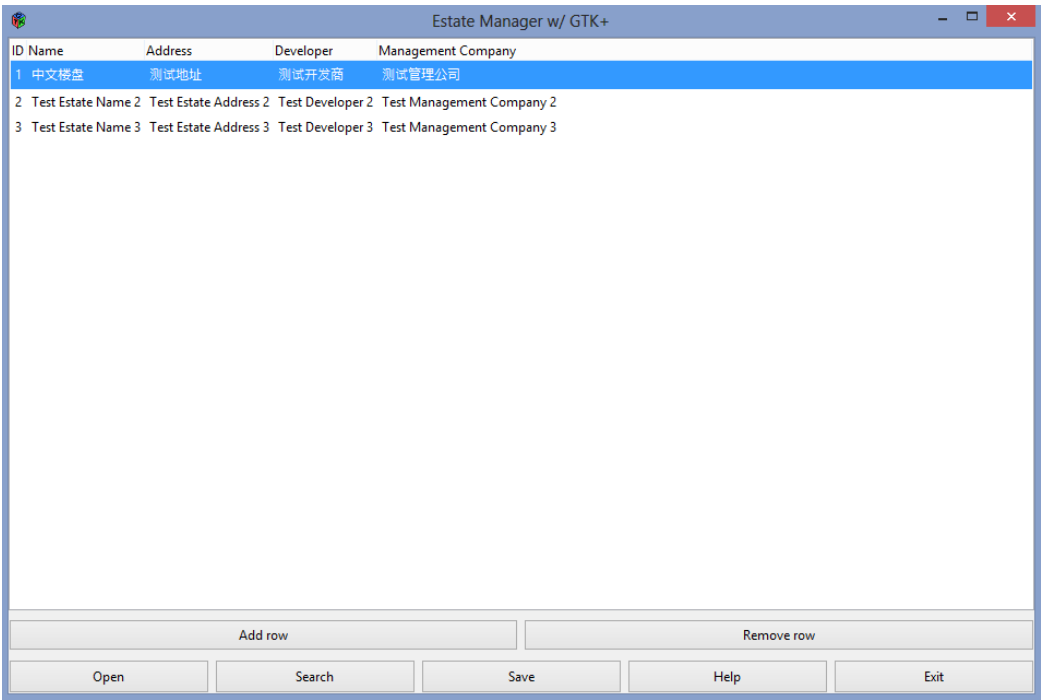
### 3.4 功能测试

- 3.4.1 为程序加入数组测试数据，测试基础列表显示功能。
- 3.4.2 测试程序的单元格编辑功能。
- 3.4.3 测试程序的条目添加、删除与保存功能。
- 3.4.4 测试程序的查询功能。

# 四 结果分析

## 4.1 实际界面

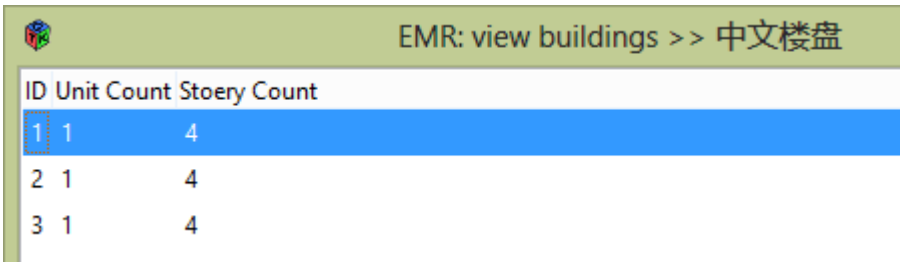
4.1.1 主界面（楼盘查看与编辑界面）如图所示：



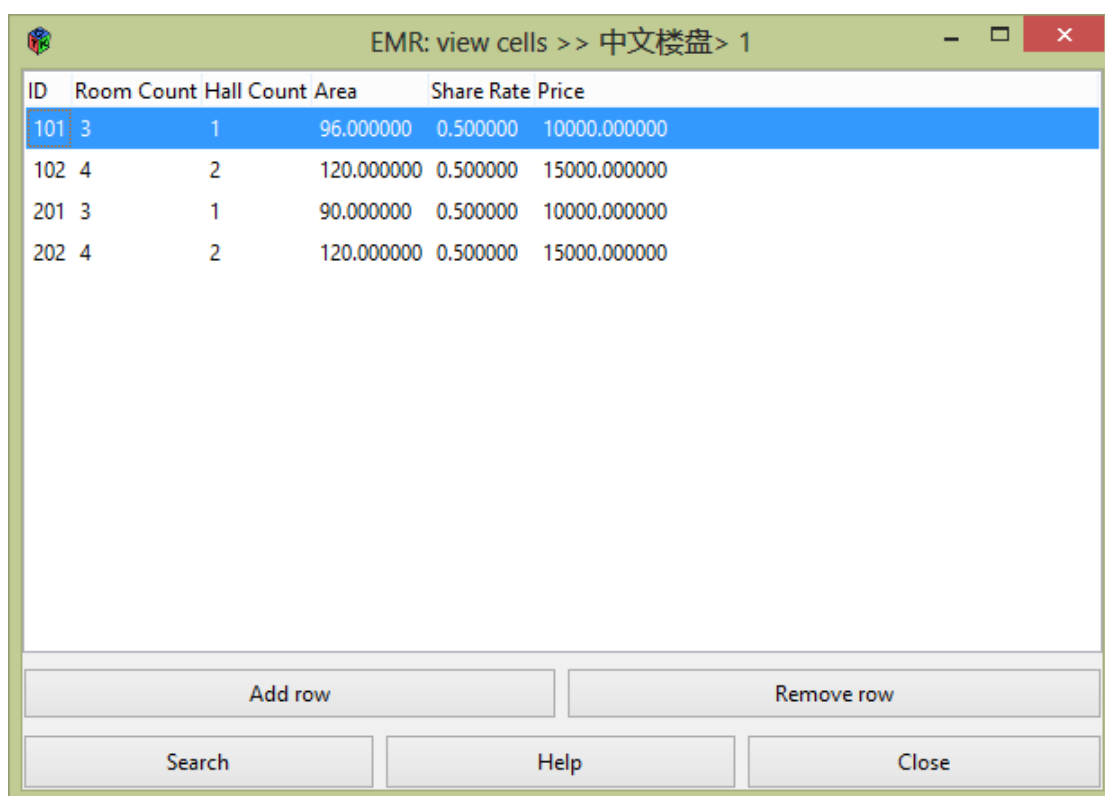
### 4.1.2 编辑功能示例



### 4.1.3 楼栋信息查看/编辑界面（标题上标明了楼盘名称）

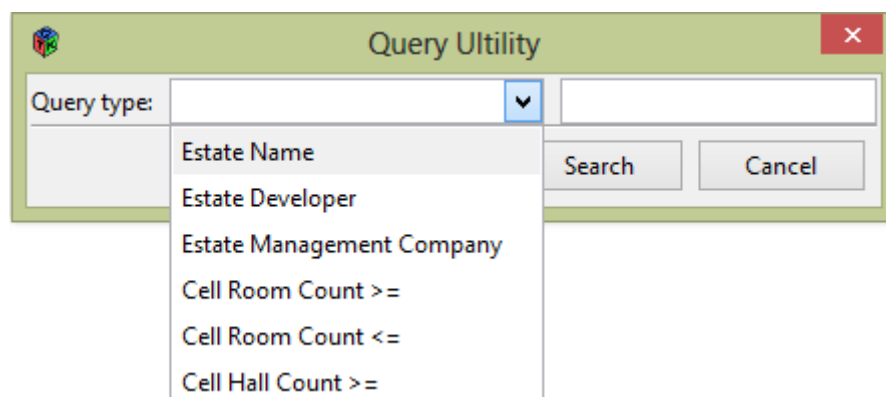


#### 4.1.4 房间信息查看/编辑界面



ID	Room Count	Hall Count	Area	Share Rate	Price
101	3	1	96.000000	0.500000	10000.000000
102	4	2	120.000000	0.500000	15000.000000
201	3	1	90.000000	0.500000	10000.000000
202	4	2	120.000000	0.500000	15000.000000

#### 4.1.5 搜索对话框

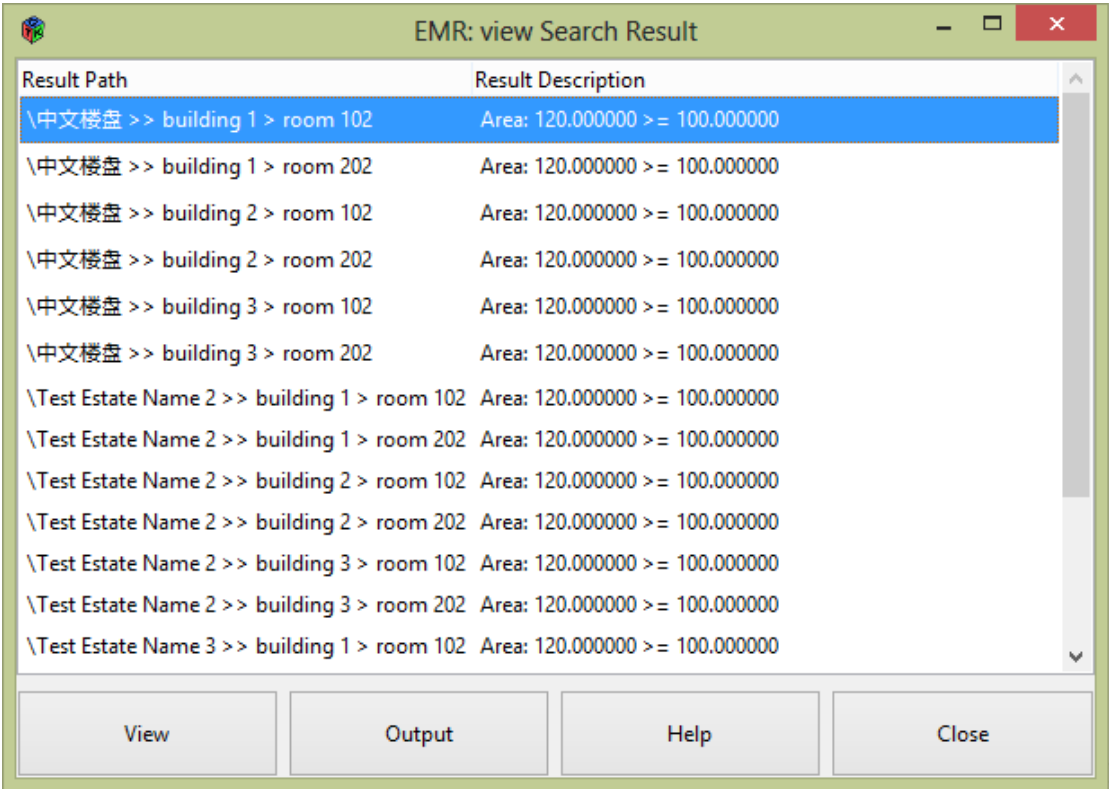


Query type: ▼

- Estate Name
- Estate Developer
- Estate Management Company
- Cell Room Count >=
- Cell Room Count <=
- Cell Hall Count >=



4.1.6 检索结果列表窗口



4.2 实现的功能与分析

- 4.2.1 本程序完成了 2.2 中方案所提到的全部功能。包括从文件读取数据；中获取数据并以列表的形式显示；编辑数据；检索数据，并显示检索结果；保存编辑过的数据。
- 4.2.2 本系统还有很多功能可供扩展，如输出到打印机、输出表格并发送、分享到社交网络等。由于时间仓促，不能一一实现。

五 总结

通过本次课程设计，我对 C 语言的各种特性，以及 GTK+图形界面工具包有了更深的了解，同时也体验了项目开发的基本过程。经过本次课程设计，我的 C 语言技能有了显著提高，同时也加深了我对各类信息处理系统的算法的理解。可以说，本次课程设计是非常有益的，我受益匪浅。

不过根据我对 C 语言之外的其他计算机技术的了解，我有几点想法。首先，我觉得 C 语言并非此类信息管理系统的最佳选择。C 作为最接近系统底层的语言，其速度在编程语言界中具有非常明显的优势。不过，对于信息系统而言，程序运行的效率更大程度地依赖于对于存储介质的读写。不管是磁盘还是内存，其读写速度与 CPU 的时钟都不是一个数量级的。其次，C 语言主要用于编写在本机上运行的应用程序，而在每台使

用信息管理系统的终端上都存储一份数据不仅是一种对于存储空间的浪费,还可能导致两处数据不同的尴尬局面。如果把数据发到网络上共享,则存在操作权限等安全性问题。综上所述,我认为比较适宜开发此类信息管理系统的语言是一项网页后台语言与一项数据库语言,如 PHP+MySQL。原因有四:一,这种方案能够很容易地实现数据的共享,即用户只需要使用网页浏览器就能够得到自己需要的信息;二,通过控制每个用户的修改权限,可以保证数据的完整性、可靠性与安全性;三,这种系统更易于建立,部署与维护;四,这种系统可以比较容易地实现商业化,对于开发者有经济利益。

总而言之,虽然在项目的选择上不是特别有针对性,我认为这次课程设计还是很成功,有收获且有建设性的。

## 六 参考文献

[1] 曹计昌, 卢萍, 李开. C 语言程序设计. 北京: 科学出版社, 2008

[2] <http://developer.gnome.org/gtk/stable/>

[3] <http://www.cplusplus.com/reference/clibrary/>

## 七 附录

### 7.1 技术文档

#### 7.1.1 关于本技术文档

- A 本文档的目的是从日后参考及解释基本原理概念两个角度阐述本楼盘管理系统(以下简称“系统”)的结构、功能、操作方式。下文每一小节将讲述本系统中的一个模块或一个功能。
- B 本系统仍在逐步完善的过程中,在这一阶段,部分组件或模块比其他组件或模块更加可靠。由于很多模块现在已经非常依赖于某一特定模块,或者某一特定模块已经经过大量和系统的测试并被认为是稳定可靠的,这些特定模块不太可能发生变动。也有部分模块有产生问题的记录,因此使用这些模块时需要多加注意。
- C 稳定性分为以下几个等级: 1- 实验性:本功能于最近加入,属于实验性质,且可能有问题并将有改动; 2- 不稳定 本功能尚未经过足够的测试 不能认为它是稳定的; 3- 稳定:本功能已经通过了测试,将来改动的可能性不大; 4- 已冻结:本部分已经经过了很多测试,且很多功能依赖于它,因此将来不太可能变动; 5- 已锁定:除非非常严重的错误出现在本部分,它不会再进行改动。

### 7.1.2 依赖组件 [稳定性：5 - 已锁定]

- A 由于本系统是以 GTK+图形工具包为技术设计的，因此程序运行时需要依赖于部分 GTK+动态链接库。这些文件应当被保存在程序可执行文件的目录下，或者将其注册为环境变量。
- B 这些可能需要的组件包括：freetype6.dll，intl.dll，libasprintf-0.dll，libatk-1.0-0.dll，libcairo-2.dll，libcairo-gobject-2.dll，libcairo-script-interpreter-2.dll，libexpat-1.dll，libfontconfig-1.dll，libgailutil-18.dll，libgcc\_s\_dw2-1.dll，libgdk\_pixbuf-2.0-0.dll，libgdk-win32-2.0-0.dll，libgio-2.0-0.dll，libglib-2.0-0.dll，libgmodule-2.0-0.dll，libgobject-2.0-0.dll，libgthread-2.0-0.dll，libgtk-win32-2.0-0.dll，libpango-1.0-0.dll，libpangocairo-1.0-0.dll，libpangoft2-1.0-0.dll，libpangowin32-1.0-0.dll，libpng14-14.dll，zlib1.dll。
- C 这些组件可以从 GTK+官方网站上下载：<http://www.gtk.org/download/index.php>。请根据您系统的版本选用合适的组件包。

### 7.1.3 系统数据文件 [稳定性：4 - 已冻结]

- A 本系统使用.db 扩展名的文件存储数据。这些文件将被保存在程序可执行文件所在目录下的 db 文件夹里。首次使用本系统时请确保本系统提供的范例数据文件夹与程序可执行文件在同一目录，且该文件夹及其中的文件未被设为只读。
- B 本系统的楼盘、楼栋、房间信息文件名命名方式不同。楼盘数据文件命名为 database\_estate.db，包括全部楼盘的信息；楼栋数据文件命名为 database\_bldn\_x.db，其中 x 为这些楼栋所属楼盘的编号；房间数据文件命名为 database\_cell\_x\_y.db，其中 x 为这些房间所属楼盘的编号，y 为这些房间所属楼栋的编号。
- C 每个数据文件以 UTF-8 的字符编码保存。数据文件的第一行应当以 <meta>[DB\_type]</meta> 的形式标识本数据文件的类型（如 <meta>EstateDB</meta>）。从第二行起，每一行存储一个条目。每个条目的属性应当按顺序排列，并用尖括号（< 和 >）标识属性的开始与末尾。文件中不应有空行。

### 7.1.4 查看信息 [稳定性：4 - 已冻结]

- A 本系统启动是会自动读取数据文件，并在主窗口上以列表的形式显示楼盘数据。
- B 需要查看某楼盘的楼栋数据时，首先在列表上左键点击以选中该楼盘，然后点击第二排按钮中的“打开”（open）按钮。这些楼栋信息将在一个新窗口中显示。
- C 查看房间数据的方式与查看某楼盘的楼栋数据的方式相同。

### 7.1.5 修改信息 [稳定性：3 - 稳定]

需要修改信息条目时，点击需要修改的数据的单元格，单元格中即出现光标，然后可以直接进行编辑。

#### 7.1.6 删除信息 [稳定性：3 - 稳定]

需要删除信息条目时,选中需要删除的条目,点击第一排按钮中的“删除行”( remove row ), 该条目即被删除。

#### 7.1.7 添加信息 [稳定性：3 - 稳定]

A 需要添加新信息条目时,直接点击第一排按钮中的“添加行”( Add Row ), 表格视图底部将会出现新添加的条目。

B 新添加的条目的各项属性均为默认值, 您可以参考 5.1.5 对其进行修改。

#### 7.1.8 保存信息 [稳定性：3 - 稳定]

A 考虑到误操作的可能, 本系统没有提供自动保存功能。

B 需要保存时, 点击主窗口( 楼盘视图 ) 中的“保存”( save ) 按钮, 数据文件即按照 5.1.3 中所述的方式进行保存。

#### 7.1.9 查询信息 [稳定性：3 - 稳定]

A 需要查询时, 点击“搜索”按钮( search ), 系统会弹出一个对话框要求用户选择查询种类和输入查询信息。

B 点击对话框的“搜索”( search ) 按钮时系统将根据用户输入的条件进行检索, 检索完毕后将打开一个新窗口显示检索结果。该窗口将列出所有符合条件的检索结果, 并指示这些结果的路径以及系统认为该条目符合检索条件的原因。

C 选中一个检索结果之后, 点击“查看”( view ) 按钮将打开该结果在的楼盘、楼栋或房间视图, 用户可以在此根据 5.1.5、5.1.6、5.1.7 中所述之方法操作该条目。

#### 7.1.10 退出系统 [稳定性：5 - 已锁定]

本系统使用完毕时, 可以通过点击主窗口上的“退出”( exit ) 按钮或窗口右上角的红叉退出本系统。

## 7.2 源程序代码

\main.c

```
#include <gtk/gtk.h>
#include <stdlib.h>
#include "core.h"
#include "core_io.h"
#include "core_EST.h"
#include "core_CEL.h"
```

```

#include "core_BDN.h"
#include "core_query.h"

int
main (int argc, char **argv)
{
    GtkWidget *view;
    GtkWidget *vbox, *hbox1, *hbox2;
    GtkWidget *button_add, *button_delete, *button_open, *button_save,
*button_search_dialog, *button_help, *button_exit;
    GtkWidget *scrolled_window;

    /* System Initialization */
    gtk_init (&argc, &argv);
    IO_initialization();
    EST_array_link ();

    window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
    gtk_window_set_default_size (GTK_WINDOW (window), 940, 600);
    gtk_window_set_title (GTK_WINDOW (window), "Estate Manager w/ GTK+");
    g_signal_connect (window, "delete_event", gtk_main_quit, NULL); /* exit main
thread when clicking red 'x' */

    /* create containers */
    vbox = gtk_vbox_new(FALSE, 8);
    hbox1 = gtk_hbox_new(TRUE, 5);
    hbox2 = gtk_hbox_new(TRUE, 5);

    /* Initialize buttons */
    button_add = gtk_button_new_with_label ("Add row");
    button_delete = gtk_button_new_with_label ("Remove row");
    button_open = gtk_button_new_with_label ("Open");
    button_search_dialog = gtk_button_new_with_label ("Search");
    button_save = gtk_button_new_with_label ("Save");
    button_help = gtk_button_new_with_label ("Help");
    button_exit = gtk_button_new_with_label ("Exit");

    g_signal_connect (button_exit, "clicked", gtk_main_quit, NULL);

    /* Create main view */
    scrolled_window = gtk_scrolled_window_new (NULL, NULL);
    gtk_scrolled_window_set_shadow_type (GTK_SCROLLED_WINDOW (scrolled_window),
GTK_SHADOW_ETCHED_IN);
    gtk_scrolled_window_set_policy (GTK_SCROLLED_WINDOW (scrolled_window),

```

```

                                GTK_POLICY_AUTOMATIC,
                                GTK_POLICY_AUTOMATIC);
gtk_widget_set_usize (scrolled_window, 920, 520);
view = EST_create_view_and_model ();

/* Add widgets into their containers */
gtk_container_add (GTK_CONTAINER (window), vbox);
gtk_box_pack_start (GTK_BOX (vbox), scrolled_window, TRUE, TRUE, 0);
gtk_container_add (GTK_CONTAINER (scrolled_window), view);
gtk_container_add (GTK_CONTAINER (vbox), hbox1);
gtk_container_add (GTK_CONTAINER (vbox), hbox2);
gtk_container_add (GTK_CONTAINER (hbox1), button_add);
gtk_container_add (GTK_CONTAINER (hbox1), button_delete);
gtk_container_add (GTK_CONTAINER (hbox2), button_open);
gtk_container_add (GTK_CONTAINER (hbox2), button_search_dialog);
gtk_container_add (GTK_CONTAINER (hbox2), button_save);
gtk_container_add (GTK_CONTAINER (hbox2), button_help);
gtk_container_add (GTK_CONTAINER (hbox2), button_exit);

/* Connect signals */
g_signal_connect (button_add, "clicked",
                  G_CALLBACK (EST_add_item), GTK_TREE_MODEL
(gtk_tree_view_get_model(GTK_TREE_VIEW (view))));
g_signal_connect (button_delete, "clicked",
                  G_CALLBACK (EST_remove_item), view);
g_signal_connect (button_open, "clicked",
                  G_CALLBACK (BDN_open_window), view);
g_signal_connect (button_save, "clicked",
                  G_CALLBACK (IO_save), NULL);
g_signal_connect (button_search_dialog, "clicked",
                  G_CALLBACK (QRY_main), window);

gtk_widget_show_all (window);
gtk_main ();
return 0;
}

```

## \\core.h

```

enum /* columns of estate sheet */
{
    EST_ID = 0,
    EST_NAME,
    EST_ADDR,

```

```

    EST_DEVR,
    EST_MC,
    NUM_EST_COLUMNS
};

enum /* columns of building sheet */
{
    BDN_ID = 0,
    BDN_UCNT,
    BDN_SCNT,
    NUM_BDN_COLUMNS
};

enum /* columns of cell sheet */
{
    CEL_ID = 0,
    CEL_RCT,
    CEL_HCT,
    CEL_ARA,
    CEL_SRT,
    CEL_PRC,
    NUM_CEL_COLUMNS
};

enum /* columns of query result sheet */
{
    QRY_RST_PATH = 0,
    QRY_RST_DESC,
    NUM_QRY_RST_COLUMNS
};

static GArray *estates = NULL;
static GArray *buildings = NULL;
static GArray *cells = NULL;
static GArray *results = NULL;

typedef struct GEstate
{
    gint id;
    gchar *name;
    gchar *addr;
    gchar *propertyDevr; //Developer
    gchar *propertyMC;   //Management Company
    struct GEstate *next;

```

```

    struct GEstate *prev;
    struct GBldn *child; //its children buildings
}GEstate;

typedef struct GBldn
{
    gint id;
    gint unitCount;
    gint storeyCount;
    struct GEstate *parent;
    struct GBldn *next;
    struct GBldn *prev;
    struct GCell *child;
}GBldn;

typedef struct GCell
{
    gint id;
    gint roomCount;
    gint hallCount;
    gfloat area;
    gfloat shareRate;
    gfloat price;
    struct GCell *next;
    struct GCell *prev;
    struct GBldn *parent;
}GCell;

typedef struct GSResult
{
    gint type;
    gchar *path;
    gchar *desc;
    gpointer entity;
    //struct GSResult *next;
}GSResult;

static GEstate *EST_head = NULL;
static GBldn *BDN_head = NULL;
static GCell *CEL_head = NULL;

static GtkWidget *window = NULL;
static GtkWidget *window_bldn = NULL;
static GtkWidget *window_cell = NULL;

```



```

static gboolean
separator_row (GtkTreeModel *model,
               GtkTreeIter  *iter,
               gpointer      data)
{
    GtkTreePath *path;
    gint idx;

    path = gtk_tree_model_get_path (model, iter);
    idx = gtk_tree_path_get_indices (path)[0];

    gtk_tree_path_free (path);

    return idx == 5;
}

static void
editing_started (GtkCellRenderer *cell,
                GtkCellEditable *editable,
                const gchar      *path,
                gpointer          data)
{
    gtk_combo_box_set_row_separator_func (GTK_COMBO_BOX (editable),
                                          separator_row, NULL, NULL);
}

```

## \core\_io.h

```

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

static GEstate *
IO_initialization() /* Read file and initialize all link sheets */
{
    FILE *estateDB = fopen("db\\database_estate.db","r");
    FILE *bldnDB = NULL;
    FILE *cellDB = NULL;
    GEstate *estateHead, *estateThis = NULL, *estatePrev = NULL;
    GBldn *bldnHead, *bldnThis, *bldnPrev = NULL;
    GCell *cellHead, *cellThis, *cellPrev = NULL;
    char fileBuffer[255], propertyTemp[100], bldnFileName[40], cellFileName[40];
    gchar *estate_properties[5], *bldn_properties[3], *cell_properties[6];
}

```

```

int i, j, k;
estateHead = (GEstate *)malloc(sizeof(GEstate));
estateThis = estateHead;
fgets(fileBuffer, 255, estateDB); // ignore line 1
do {
    fgets(fileBuffer, 255, estateDB); // estate level
    if (estateThis == NULL)
    {
        estateThis = (GEstate *)malloc(sizeof(GEstate));
        estatePrev->next = estateThis;
    }

    for (i=0, k=0; fileBuffer[i] != '\n' && k < NUM_EST_COLUMNS; i++)
    {
        if (fileBuffer[i]=='<')
        {
            for (j=0,i++;fileBuffer[i]!='>';i++,j++)
            {
                propertyTemp[j] = fileBuffer[i];
            }
            propertyTemp[j] = '\0';
            estate_properties[k] = g_strdup (propertyTemp);
            k++;
        }
    }

    estateThis->id = atoi(estate_properties[0]);
    estateThis->name = g_strdup (estate_properties[1]);
    estateThis->addr = g_strdup (estate_properties[2]);
    estateThis->propertyDevr = g_strdup (estate_properties[3]);
    estateThis->propertyMC = g_strdup (estate_properties[4]);
    strcpy(bldnFileName, "db\\database_bldn_");
    strcat(bldnFileName, estate_properties[0]);
    strcat(bldnFileName, ".db");
    bldnDB = fopen(bldnFileName,"r"); //open building database
    if (bldnDB != NULL)
    {
        bldnHead = (GBldn *)malloc(sizeof(GBldn));
        bldnThis = bldnHead;
        bldnPrev = NULL;
        fgets(fileBuffer, 255, bldnDB); // ignore line 1
        do {
            fgets(fileBuffer, 255, bldnDB); // building level
            if (bldnThis == NULL)

```

```

{
    bldnThis = (GBldn *)malloc(sizeof(GBldn));
    bldnPrev->next = bldnThis;
}

for (i=0, k=0; fileBuffer[i] != '\n' && k < NUM_BDN_COLUMNS; i++)
{
    if (fileBuffer[i] == '<')
    {
        for (j=0, i++; fileBuffer[i] != '>'; i++, j++)
        {
            propertyTemp[j] = fileBuffer[i];
        }
        propertyTemp[j] = '\0';
        bldn_properties[k] = g_strdup (propertyTemp);
        k++;
    }
}

bldnThis->id = atoi(bldn_properties[0]);
bldnThis->unitCount = atoi(bldn_properties[1]);
bldnThis->storeyCount = atoi(bldn_properties[2]);
bldnThis->parent = estateThis;
strcpy(cellFileName, "db\\database_cell_");
strcat(cellFileName, estate_properties[0]);
strcat(cellFileName, "_");
strcat(cellFileName, bldn_properties[0]);
strcat(cellFileName, ".db");
cellDB = fopen(cellFileName, "r"); //open cell database
if (cellDB != NULL)
{
    cellHead = (GCell *)malloc(sizeof(GCell));
    cellThis = cellHead;
    cellPrev = NULL;
    fgets(fileBuffer, 200, cellDB);
    do {
        fgets(fileBuffer, 255, cellDB); // cell level
        if (cellThis == NULL)
        {
            cellThis = (GCell*)malloc(sizeof(GCell));
            cellPrev->next = cellThis;
        }

        for (i=0, k=0; fileBuffer[i] != '\n' && k < NUM_CEL_COLUMNS; i++)

```

```

    {
        if (fileBuffer[i] == '<')
        {
            for (j=0, i++; fileBuffer[i] != '>'; i++, j++)
            {
                propertyTemp[j] = fileBuffer[i];
            }
            propertyTemp[j] = '\0';
            cell_properties[k] = g_strdup (propertyTemp);
            k++;
        }
    }

    cellThis->id = atoi(cell_properties[0]);
    cellThis->roomCount = atoi(cell_properties[1]);
    cellThis->hallCount = atoi(cell_properties[2]);
    cellThis->area = atof(cell_properties[3]);
    cellThis->shareRate = atof(cell_properties[4]);
    cellThis->price = atof(cell_properties[5]);
    cellThis->parent = bldnThis;
    cellThis->next = NULL;
    cellThis->prev = cellPrev;
    cellPrev = cellThis;
    cellThis = NULL;
} while (!feof(cellDB));
fclose(cellDB);
}

bldnThis->child = cellHead;
bldnThis->next = NULL;
bldnThis->prev = bldnPrev;
bldnPrev = bldnThis;
bldnThis = NULL;
} while (!feof(bldnDB));
fclose(bldnDB);
}

estateThis->child = bldnHead;
estateThis->next = NULL;
estateThis->prev = estatePrev;
estatePrev = estateThis;
estateThis = NULL;
} while (!feof(estateDB));
fclose(estateDB);
EST_head = estateHead;
return estateHead;

```

```

}

static void
IO_save()
{
    FILE *estateDB = fopen("db\\database_estate.db","w");
    FILE *bldnDB = NULL;
    FILE *cellDB = NULL;
    GEstate *estateThis = EST_head;
    GBldn *bldnHead, *bldnThis;
    GCell *cellHead, *cellThis;
    char bldnFileName[40], cellFileName[40], idString[8];
    fprintf(estateDB,"<meta>EstateDB</meta>"); // print line 1
    do {
        fprintf(estateDB,
            "\n<%d><%s><%s><%s><%s>",
            estateThis->id,
            estateThis->name,
            estateThis->addr,
            estateThis->propertyDevr,
            estateThis->propertyMC);
        strcpy(bldnFileName , "db\\database_bldn_");
        sprintf(idString, "%d", estateThis->id);
        strcat(bldnFileName,idString);
        strcat(bldnFileName, ".db");
        bldnDB = fopen(bldnFileName,"w"); //write building database
        bldnHead = estateThis->child;
        if (bldnDB != NULL)
        {
            bldnThis = bldnHead;
            fprintf(bldnDB,"<meta>BldnDB</meta>"); // print line 1
            while (bldnThis != NULL)
            {
                fprintf(bldnDB,
                    "\n<%d><%d><%d>",
                    bldnThis->id,
                    bldnThis->unitCount,
                    bldnThis->storeyCount);
                cellHead = bldnThis->child;
                strcpy(cellFileName , "db\\database_cell_");
                sprintf(idString, "%d", estateThis->id);
                strcat(cellFileName, idString);
                strcat(cellFileName, "_");
                sprintf(idString, "%d", bldnThis->id);
            }
        }
    } while (estateThis != NULL);
}

```

```

        strcat(cellFileName, idString);
        strcat(cellFileName, ".db");
        cellDB = fopen(cellFileName, "w"); //write cell database
        if (cellDB != NULL && cellHead != NULL)
        {
            cellThis = cellHead;
            fprintf(cellDB, "<meta>CellDB</meta>");
            while (cellThis != NULL)
            {
                fprintf(cellDB,
                    "\n<d><d><d><%0.1f><%0.2f><%0.0f>",
                    cellThis->id, cellThis->roomCount,
                    cellThis->hallCount,
                    cellThis->area,
                    cellThis->shareRate,
                    cellThis->price);
                cellThis = cellThis->next;
            }
            fclose(cellDB);
        }
        bldnThis = bldnThis->next;
    }
    fclose(bldnDB);
}

estateThis = estateThis->next;
}while (estateThis!=NULL);
fclose(estateDB);
}

```

## \core\_EST.h

```

static void
EST_array_link () /* Link struct GEstate to a GArray for viewing and editing */
{
    GEstate *estate_temp = EST_head;
    estates = g_array_sized_new (FALSE, FALSE, sizeof (GEstate *), 1);
    g_return_if_fail (estates != NULL);

    for (;estate_temp != NULL; estate_temp = estate_temp -> next)
    {
        g_array_append_vals (estates, &estate_temp, 1);
    }
}

```

```

static void
EST_cell_edited (GtkCellRendererText *cell,
                 const gchar          *path_string,
                 const gchar          *new_text,
                 gpointer              data) /* response function for an edited cell
*/
{
    GtkTreeModel *model = (GtkTreeModel *)data;
    GtkTreePath *path = gtk_tree_path_new_from_string (path_string);
    GtkTreeIter iter;

    gint column = GPOINTER_TO_INT (g_object_get_data (G_OBJECT (cell), "column"));

    gtk_tree_model_get_iter (model, &iter, path);

    switch (column) /* different actions depending on the column which was edited
*/
    {
        case EST_ID:
            break;
        case EST_NAME:
            {
                gint i;
                gchar *old_text;

                gtk_tree_model_get (model, &iter, column, &old_text, -1);
                g_free (old_text);

                i = gtk_tree_path_get_indices (path)[0];
                g_free (g_array_index (estates, GEstate *, i)->name);
                g_array_index (estates, GEstate *, i)->name = g_strdup (new_text);

                gtk_list_store_set (GTK_LIST_STORE (model), &iter, column,
                                   g_array_index (estates, GEstate *, i)->name, -1);
            }
            break;
        case EST_ADDR:
            {
                gint i;
                gchar *old_text;

                gtk_tree_model_get (model, &iter, column, &old_text, -1);
                g_free (old_text);
            }
    }
}

```

```

        i = gtk_tree_path_get_indices (path)[0];
        g_free (g_array_index (estates, GEstate *, i)->addr);
        g_array_index (estates, GEstate *, i)->addr = g_strdup (new_text);

        gtk_list_store_set (GTK_LIST_STORE (model), &iter, column,
                            g_array_index (estates, GEstate *, i)->addr, -1);
    }
    break;
case EST_DEVR:
    {
        gint i;
        gchar *old_text;

        gtk_tree_model_get (model, &iter, column, &old_text, -1);
        g_free (old_text);

        i = gtk_tree_path_get_indices (path)[0];
        g_free (g_array_index (estates, GEstate *, i)->propertyDevr);
        g_array_index (estates, GEstate *, i)->propertyDevr = g_strdup (new_text);

        gtk_list_store_set (GTK_LIST_STORE (model), &iter, column,
                            g_array_index (estates, GEstate *, i)->propertyDevr,
-1);
    }
    break;
case EST_MC:
    {
        gint i;
        gchar *old_text;

        gtk_tree_model_get (model, &iter, column, &old_text, -1);
        g_free (old_text);

        i = gtk_tree_path_get_indices (path)[0];
        g_free (g_array_index (estates, GEstate *, i)->propertyMC);
        g_array_index (estates, GEstate *, i)->propertyMC = g_strdup (new_text);

        gtk_list_store_set (GTK_LIST_STORE (model), &iter, column,
                            g_array_index (estates, GEstate *, i)->propertyMC, -1);
    }
    break;
}
gtk_tree_path_free (path);
}

```



```

static void
EST_add_item (GtkWidget *button,
              gpointer data) /* Append a new estate */
{
    GEstate *estate_temp = (GEstate *)malloc(sizeof(GEstate));
    GtkTreeIter iter;
    GtkTreeModel *model = (GtkTreeModel *)data;

    g_return_if_fail (estates != NULL);

    estate_temp->id = g_array_index (estates, GEstate *, estates->len - 1)->id +
1;
    estate_temp->name = g_strdup ("Enter name here");
    estate_temp->addr = g_strdup ("Enter address here");
    estate_temp->propertyDevr = g_strdup ("Enter Developer here");
    estate_temp->propertyMC = g_strdup ("Enter Management Company here");
    estate_temp->prev = g_array_index (estates, GEstate *, estates->len - 1);
    estate_temp->next = NULL;
    estate_temp->child = NULL;
    g_array_index (estates, GEstate *, estates->len - 1)->next = estate_temp;
    g_array_append_vals (estates, &estate_temp, 1);

    gtk_list_store_append (GTK_LIST_STORE (model), &iter);
    gtk_list_store_set (GTK_LIST_STORE (model), &iter,
                        EST_ID, estate_temp->id,
                        EST_NAME, estate_temp->name,
                        EST_ADDR, estate_temp->addr,
                        EST_DEVR, estate_temp->propertyDevr,
                        EST_MC, estate_temp->propertyMC,
                        -1);
}

static void
EST_remove_item (GtkWidget *widget,
                 gpointer data) /* remove selected estate */
{
    GtkTreeIter iter;
    GtkTreeView *treeview = (GtkTreeView *)data;
    GtkTreeModel *model = gtk_tree_view_get_model (treeview);
    GtkTreeSelection *selection = gtk_tree_view_get_selection (treeview);

    if (gtk_tree_selection_get_selected (selection, NULL, &iter))
    {

```

```

    gint i;
    GtkTreePath *path;

    path = gtk_tree_model_get_path (model, &iter);
    i = gtk_tree_path_get_indices (path)[0];
    gtk_list_store_remove (GTK_LIST_STORE (model), &iter);

    g_array_index (estates, GEstate *, i)->prev = g_array_index (estates,
GEstate *, i)->next;
    free (g_array_index (estates, GEstate *, i));
    g_array_remove_index (estates, i);

    gtk_tree_path_free (path);
}
}

static GtkTreeModel *
EST_create_and_fill_model () /* create a tree model based on GArray estates */
{
    gint i = 0;
    GtkListStore *store;
    GtkTreeIter iter;

    store = gtk_list_store_new (NUM_EST_COLUMNS, G_TYPE_INT, G_TYPE_STRING,
G_TYPE_STRING, G_TYPE_STRING, G_TYPE_STRING);

    for (i = 0; i < estates->len; i++)
    {
        gtk_list_store_append (store, &iter);

        gtk_list_store_set (store, &iter,
                            EST_ID, g_array_index (estates, GEstate *, i)->id,
                            EST_NAME, g_array_index (estates, GEstate *, i)->name,
                            EST_ADDR, g_array_index (estates, GEstate *, i)->addr,
                            EST_DEVR, g_array_index (estates, GEstate *,
i)->propertyDevr,
                            EST_MC, g_array_index (estates, GEstate *, i)->propertyMC,
                            -1);
    }
    /* Append a row and fill in some data */
    return GTK_TREE_MODEL (store);
}

```

```

static GtkWidget *
EST_create_view_and_model () /* Create tree view; create tree model by calling
'EST_create_and_fill_model' */
{
    GtkCellRenderer *renderer;
    GtkTreeModel *model;
    GtkWidget *view;
    view = gtk_tree_view_new ();
    model = EST_create_and_fill_model();
    /* --- Column #1 --- */
    renderer = gtk_cell_renderer_text_new ();
    g_object_set (renderer,
                  "editable", FALSE,
                  NULL);
    g_signal_connect (renderer, "edited",
                      G_CALLBACK (EST_cell_edited), model);
    g_signal_connect (renderer, "editing-started",
                      G_CALLBACK (editing_started), NULL);
    g_object_set_data (G_OBJECT (renderer), "column", GINT_TO_POINTER (EST_ID));
    gtk_tree_view_insert_column_with_attributes (GTK_TREE_VIEW (view),
                                                -1,
                                                "ID",
                                                renderer,
                                                "text", EST_ID,
                                                NULL);

    /* --- Column #2 --- */
    renderer = gtk_cell_renderer_text_new ();
    g_object_set (renderer,
                  "editable", TRUE,
                  NULL);
    g_signal_connect (renderer, "edited",
                      G_CALLBACK (EST_cell_edited), model);
    g_object_set_data (G_OBJECT (renderer), "column", GINT_TO_POINTER (EST_NAME));
    gtk_tree_view_insert_column_with_attributes (GTK_TREE_VIEW (view),
                                                -1,
                                                "Name",
                                                renderer,
                                                "text", EST_NAME,
                                                NULL);

    /* --- Column #3 --- */
    renderer = gtk_cell_renderer_text_new ();
    g_object_set (renderer,
                  "editable", TRUE,
                  NULL);

```

```

g_signal_connect (renderer, "edited",
                  G_CALLBACK (EST_cell_edited), model);
g_object_set_data (G_OBJECT (renderer), "column", GINT_TO_POINTER (EST_ADDR));
gtk_tree_view_insert_column_with_attributes (GTK_TREE_VIEW (view),
                                             -1,
                                             "Address",
                                             renderer,
                                             "text", EST_ADDR,
                                             NULL);

/* --- Column #4 --- */
renderer = gtk_cell_renderer_text_new ();
g_object_set (renderer,
              "editable", TRUE,
              NULL);
g_signal_connect (renderer, "edited",
                  G_CALLBACK (EST_cell_edited), model);
g_object_set_data (G_OBJECT (renderer), "column", GINT_TO_POINTER (EST_DEVR));
gtk_tree_view_insert_column_with_attributes (GTK_TREE_VIEW (view),
                                             -1,
                                             "Developer",
                                             renderer,
                                             "text", EST_DEVR,
                                             NULL);

/* --- Column #5 --- */
renderer = gtk_cell_renderer_text_new ();
g_object_set (renderer,
              "editable", TRUE,
              NULL);
g_signal_connect (renderer, "edited",
                  G_CALLBACK (EST_cell_edited), model);
g_object_set_data (G_OBJECT (renderer), "column", GINT_TO_POINTER (EST_MC));
gtk_tree_view_insert_column_with_attributes (GTK_TREE_VIEW (view),
                                             -1,
                                             "Management Company",
                                             renderer,
                                             "text", EST_MC,
                                             NULL);

gtk_tree_view_set_model (GTK_TREE_VIEW (view), model);
/* The tree view has acquired its own reference to the
 * model, so we can drop ours. That way the model will
 * be freed automatically when the tree view is destroyed */

g_object_unref (model);

```

```

    return view;
}

```

## \core\_BDN.h

```

static void
BDN_cell_edited (GtkCellRendererText *cell,
                 const gchar          *path_string,
                 const gchar          *new_text,
                 gpointer              data) /* response function for an edited cell
*/
{
    GtkTreeModel *model = (GtkTreeModel *)data;
    GtkTreePath *path = gtk_tree_path_new_from_string (path_string);
    GtkTreeIter iter;

    gint column = GPOINTER_TO_INT (g_object_get_data (G_OBJECT (cell), "column"));

    gtk_tree_model_get_iter (model, &iter, path);

    switch (column)
    {
    {
    case BDN_ID:
        {
            gint i;

            i = gtk_tree_path_get_indices (path)[0];
            g_array_index (buildings, GBldn *, i)->id = atoi (new_text);

            gtk_list_store_set (GTK_LIST_STORE (model), &iter, column,
                               g_array_index (buildings, GBldn *, i)->id, -1);
        }
        break;
    case BDN_UCNT:
        {
            gint i;

            i = gtk_tree_path_get_indices (path)[0];
            g_array_index (buildings, GBldn *, i)->unitCount = atoi (new_text);

            gtk_list_store_set (GTK_LIST_STORE (model), &iter, column,
                               g_array_index (buildings, GBldn *, i)->unitCount, -1);
        }
        break;
    }
}

```

```

case BDN_SCNT:
{
    gint i;

    i = gtk_tree_path_get_indices (path)[0];
    g_array_index (buildings, GBldn *, i)->storeyCount = atoi (new_text);

    gtk_list_store_set (GTK_LIST_STORE (model), &iter, column,
                        g_array_index (buildings, GBldn *, i)->storeyCount, -1);
}
break;
}
//BDN_array_sync ();
gtk_tree_path_free (path);
}

static void
BDN_add_item (GtkWidget *button,
              gpointer data) /* Append a new building */
{
    GBldn *bldn_temp = (GBldn *)malloc (sizeof(GBldn));
    GtkTreeIter iter;
    GtkTreeModel *model = (GtkTreeModel *)data;

    g_return_if_fail (buildings != NULL);

    bldn_temp->id = g_array_index (buildings, GBldn *, buildings->len - 1)->id +
1;
    bldn_temp->unitCount = 0;
    bldn_temp->storeyCount = 0;
    bldn_temp->prev = g_array_index (buildings, GBldn *, buildings->len - 1);
    bldn_temp->next = NULL;
    bldn_temp->parent = g_array_index (buildings, GBldn *, 0)->parent;
    bldn_temp->child = NULL;
    g_array_index (buildings, GBldn *, buildings->len - 1)->next = bldn_temp;
    g_array_append_vals (buildings, &bldn_temp, 1);

    gtk_list_store_append (GTK_LIST_STORE (model), &iter);
    gtk_list_store_set (GTK_LIST_STORE (model), &iter,
                        BDN_ID, bldn_temp->id,
                        BDN_UCNT, bldn_temp->unitCount,
                        BDN_SCNT, bldn_temp->storeyCount,
                        -1);
}

```

```

static void
BDN_remove_item (GtkWidget *widget,
                 gpointer data)
{
    GtkTreeIter iter;
    GtkTreeView *treeview = (GtkTreeView *)data;
    GtkTreeModel *model = gtk_tree_view_get_model (treeview);
    GtkTreeSelection *selection = gtk_tree_view_get_selection (treeview);

    if (gtk_tree_selection_get_selected (selection, NULL, &iter))
    {
        gint i;
        GtkTreePath *path;

        path = gtk_tree_model_get_path (model, &iter);
        i = gtk_tree_path_get_indices (path)[0];
        gtk_list_store_remove (GTK_LIST_STORE (model), &iter);

        g_array_index (buildings, GBldn *, i)->prev = g_array_index (buildings,
GBldn *, i)->next;
        free (g_array_index (buildings, GBldn *, i));
        g_array_remove_index (buildings, i);
        //BDN_array_sync ();

        gtk_tree_path_free (path);
    }
}

static GtkTreeModel *
BDN_create_and_fill_model () /* create a tree model based on GArray buildings
*/
{
    gint i = 0;
    GtkListStore *store;
    GtkTreeIter iter;

    store = gtk_list_store_new (NUM_BDN_COLUMNS, G_TYPE_INT, G_TYPE_INT,
G_TYPE_INT);

    for (i = 0; i < buildings->len; i++)
    {
        gtk_list_store_append (store, &iter);
    }
}

```

```

        gtk_list_store_set (store, &iter,
                            BDN_ID, g_array_index (buildings, GBldn *, i)->id,
                            BDN_UCNT, g_array_index (buildings, GBldn *, i)->unitCount,
                            BDN_SCNT, g_array_index (buildings, GBldn *,
i)->storeyCount,
                            -1);
    }
    /* Append a row and fill in some data */
    return GTK_TREE_MODEL (store);
}

static GtkWidget *
BDN_create_view_and_model (/*GtkWidget *button_add, GtkWidget *button_remove*/)
{
    GtkCellRenderer *renderer;
    GtkTreeModel *model;
    GtkWidget *view;
    view = gtk_tree_view_new ();
    model = BDN_create_and_fill_model();
    /* --- Column #1 --- */
    renderer = gtk_cell_renderer_text_new ();
    g_object_set (renderer,
                  "editable", TRUE,
                  NULL);
    g_signal_connect (renderer, "edited",
                      G_CALLBACK (BDN_cell_edited), model);
    g_signal_connect (renderer, "editing-started",
                      G_CALLBACK (editing_started), NULL);
    g_object_set_data (G_OBJECT (renderer), "column", GINT_TO_POINTER (BDN_ID));
    gtk_tree_view_insert_column_with_attributes (GTK_TREE_VIEW (view),
                                                -1,
                                                "ID",
                                                renderer,
                                                "text", BDN_ID,
                                                NULL);

    /* --- Column #2 --- */
    renderer = gtk_cell_renderer_text_new ();
    g_object_set (renderer,
                  "editable", TRUE,
                  NULL);
    g_signal_connect (renderer, "edited",
                      G_CALLBACK (BDN_cell_edited), model);
    g_object_set_data (G_OBJECT (renderer), "column", GINT_TO_POINTER (BDN_UCNT));
    gtk_tree_view_insert_column_with_attributes (GTK_TREE_VIEW (view),

```



```

-1,
"Unit Count",
renderer,
"text", BDN_UCNT,
NULL);

/* --- Column #3 --- */
renderer = gtk_cell_renderer_text_new ();
g_object_set (renderer,
              "editable", TRUE,
              NULL);
g_signal_connect (renderer, "edited",
                  G_CALLBACK (BDN_cell_edited), model);
g_object_set_data (G_OBJECT (renderer), "column", GINT_TO_POINTER (BDN_SCNT));
gtk_tree_view_insert_column_with_attributes (GTK_TREE_VIEW (view),
-1,
"Stoery Count",
renderer,
"text", BDN_SCNT,
NULL);

gtk_tree_view_set_model (GTK_TREE_VIEW (view), model);
/* The tree view has acquired its own reference to the
 * model, so we can drop ours. That way the model will
 * be freed automatically when the tree view is destroyed */

g_object_unref (model);
return view;
}

static void
BDN_destroy_window(GtkWidget *widget,
                  gpointer data)
{
    GtkWidget *window_this = (GtkWidget *)data;
    gtk_widget_destroy(window_this);
    window_bldn = NULL;
    g_return_if_fail (buildings != NULL);
    BDN_head->parent->child = g_array_index (buildings, GBldn *, 0);
    BDN_head = NULL;
    g_array_free(buildings, FALSE);
}

static GtkWidget *
BDN_new_window ()

```

```

{
    GtkWidget *window_bldn;
    GtkWidget *view;
    GtkWidget *vbox, *hbox1, *hbox2;
    GtkWidget *button_add, *button_delete, *button_open, *button_search_dialog,
*button_help, *button_close;
    GtkWidget *scrolled_window;
    gchar *window_title;

    window_bldn = gtk_window_new (GTK_WINDOW_TOPLEVEL);
    //window_title = g_strdup_printf("EMR: view buildings",);
    window_title = g_strdup_printf("EMR: view buildings >> %s",
BDN_head->parent->name);
    gtk_window_set_default_size (GTK_WINDOW (window_bldn), 600, 400);
    gtk_window_set_title (GTK_WINDOW (window_bldn), window_title);
    g_free(window_title);

    g_signal_connect (window_bldn, "delete_event", G_CALLBACK
(gtk_widget_destroy), NULL);

    vbox = gtk_vbox_new(FALSE, 8);
    hbox1 = gtk_hbox_new(TRUE, 5);
    hbox2 = gtk_hbox_new(TRUE, 5);

    /* Initialize buttons */
    button_add = gtk_button_new_with_label ("Add row");
    button_delete = gtk_button_new_with_label ("Remove row");
    button_open = gtk_button_new_with_label ("Open");
    button_search_dialog = gtk_button_new_with_label ("Search");
    button_help = gtk_button_new_with_label ("Help");
    button_close = gtk_button_new_with_label ("Close");

    scrolled_window = gtk_scrolled_window_new (NULL, NULL);
    gtk_scrolled_window_set_shadow_type (GTK_SCROLLED_WINDOW (scrolled_window),
GTK_SHADOW_ETCHED_IN);
    gtk_scrolled_window_set_policy (GTK_SCROLLED_WINDOW (scrolled_window),
GTK_POLICY_AUTOMATIC,
GTK_POLICY_AUTOMATIC);

    g_signal_connect (button_close, "clicked", G_CALLBACK (BDN_destroy_window),
window_bldn);

    view = BDN_create_view_and_model ();
    gtk_widget_set_usize (scrolled_window, 580, 320);

```

```

gtk_container_add (GTK_CONTAINER (window_bldn), vbox);
gtk_box_pack_start (GTK_BOX (vbox), scrolled_window, TRUE, TRUE, 0);
gtk_container_add (GTK_CONTAINER (scrolled_window), view);
gtk_container_add (GTK_CONTAINER (vbox), hbox1);
gtk_container_add (GTK_CONTAINER (vbox), hbox2);
gtk_container_add (GTK_CONTAINER (hbox1), button_add);
gtk_container_add (GTK_CONTAINER (hbox1), button_delete);
gtk_container_add (GTK_CONTAINER (hbox2), button_open);
gtk_container_add (GTK_CONTAINER (hbox2), button_search_dialog);
gtk_container_add (GTK_CONTAINER (hbox2), button_help);
gtk_container_add (GTK_CONTAINER (hbox2), button_close);

g_signal_connect (button_add, "clicked",
                  G_CALLBACK (BDN_add_item), GTK_TREE_MODEL
(gtk_tree_view_get_model(GTK_TREE_VIEW (view))));
g_signal_connect (button_delete, "clicked",
                  G_CALLBACK (BDN_remove_item), view);
g_signal_connect (button_open, "clicked",
                  G_CALLBACK (CEL_open_window), view);
return window_bldn;
}

static void
BDN_open_window (GtkWidget *widget,
                 gpointer data) /* Get estate list selection and open building window
*/
{
    GtkTreeIter iter;
    GtkTreeView *treeview = (GtkTreeView *)data;
    GtkTreeModel *model = gtk_tree_view_get_model (treeview);
    GtkTreeSelection *selection = gtk_tree_view_get_selection (treeview);

    if (gtk_tree_selection_get_selected (selection, NULL, &iter))
    {
        gint i;
        GtkTreePath *path;
        GBldn *bldn_temp = NULL;

        path = gtk_tree_model_get_path (model, &iter);
        i = gtk_tree_path_get_indices (path)[0];
        bldn_temp = BDN_head = g_array_index (estates, GEstate *, i)->child;

        buildings = g_array_sized_new (FALSE, FALSE, sizeof (GBldn *), 1);

```

```

    for (; bldn_temp != NULL; bldn_temp = bldn_temp -> next)
    {
        g_array_append_vals (buildings, &bldn_temp, 1);
    }
    if (window_bldn != NULL)
    {
        BDN_destroy_window(NULL, window_bldn);
    }
    window_bldn = BDN_new_window();
    gtk_widget_show_all (window_bldn);

    gtk_tree_path_free (path);
}
}

```

## \\core\_CEL.h

```

static void
CEL_cell_edited (GtkCellRendererText *cell,
                 const gchar          *path_string,
                 const gchar          *new_text,
                 gpointer              data)
{
    GtkTreeModel *model = (GtkTreeModel *)data;
    GtkTreePath *path = gtk_tree_path_new_from_string (path_string);
    GtkTreeIter iter;

    gint column = GPOINTER_TO_INT (g_object_get_data (G_OBJECT (cell), "column"));

    gtk_tree_model_get_iter (model, &iter, path);

    switch (column)
    {
        {
            case CEL_ID:
            {
                gint i;

                i = gtk_tree_path_get_indices (path)[0];
                g_array_index (cells, GCell *, i)->id = atoi (new_text);

                gtk_list_store_set (GTK_LIST_STORE (model), &iter, column,
                                    g_array_index (cells, GCell *, i)->id, -1);
            }
        }
    }
}

```

```

        break;
case CEL_RCT:
{
    gint i;

    i = gtk_tree_path_get_indices (path)[0];
    g_array_index (cells, GCell *, i)->roomCount = atoi (new_text);

    gtk_list_store_set (GTK_LIST_STORE (model), &iter, column,
                        g_array_index (cells, GCell *, i)->roomCount, -1);
}
    break;
case CEL_HCT:
{
    gint i;

    i = gtk_tree_path_get_indices (path)[0];
    g_array_index (cells, GCell *, i)->hallCount = atoi (new_text);

    gtk_list_store_set (GTK_LIST_STORE (model), &iter, column,
                        g_array_index (cells, GCell *, i)->hallCount, -1);
}
    break;
case CEL_ARA:
{
    gint i;

    i = gtk_tree_path_get_indices (path)[0];
    g_array_index (cells, GCell *, i)->area = atof (new_text);

    gtk_list_store_set (GTK_LIST_STORE (model), &iter, column,
                        g_array_index (cells, GCell *, i)->area, -1);
}
    break;
case CEL_SRT:
{
    gint i;

    i = gtk_tree_path_get_indices (path)[0];
    g_array_index (cells, GCell *, i)->shareRate = atof (new_text);

    gtk_list_store_set (GTK_LIST_STORE (model), &iter, column,
                        g_array_index (cells, GCell *, i)->shareRate, -1);
}

```

```

        break;
    case CEL_PRC:
    {
        gint i;

        i = gtk_tree_path_get_indices (path)[0];
        g_array_index (cells, GCell *, i)->price = atof (new_text);

        gtk_list_store_set (GTK_LIST_STORE (model), &iter, column,
                            g_array_index (cells, GCell *, i)->price, -1);
    }
    break;
}
gtk_tree_path_free (path);
}

static void
CEL_add_item (GtkWidget *button,
              gpointer data)
{
    GCell *cell_temp = (GCell *)malloc(sizeof(GCell));
    GtkTreeIter iter;
    GtkTreeModel *model = (GtkTreeModel *)data;

    g_return_if_fail (cells != NULL);

    cell_temp->id = g_array_index (cells, GCell *, cells->len - 1)->id + 1;
    cell_temp->roomCount = 1;
    cell_temp->hallCount = 1;
    cell_temp->area = 100.00;
    cell_temp->shareRate = 0.50;
    cell_temp->price = 10000.0;
    cell_temp->next = NULL;
    cell_temp->parent = g_array_index (cells, GCell *, 0)->parent;
    g_array_index (cells, GCell *, cells->len - 1)->next = cell_temp;
    g_array_append_vals (cells, &cell_temp, 1);

    gtk_list_store_append (GTK_LIST_STORE (model), &iter);
    gtk_list_store_set (GTK_LIST_STORE (model), &iter,
                        CEL_ID, cell_temp->id,
                        CEL_RCT, cell_temp->roomCount,
                        CEL_HCT, cell_temp->hallCount,
                        CEL_ARA, cell_temp->area,
                        CEL_SRT, cell_temp->shareRate,

```

```

        CEL_PRC, cell_temp->price,
        -1);
}

static void
CEL_remove_item (GtkWidget *widget,
                 gpointer data)
{
    GtkTreeIter iter;
    GtkTreeView *treeview = (GtkTreeView *)data;
    GtkTreeModel *model = gtk_tree_view_get_model (treeview);
    GtkTreeSelection *selection = gtk_tree_view_get_selection (treeview);

    if (gtk_tree_selection_get_selected (selection, NULL, &iter))
    {
        gint i;
        GtkTreePath *path;

        path = gtk_tree_model_get_path (model, &iter);
        i = gtk_tree_path_get_indices (path)[0];
        gtk_list_store_remove (GTK_LIST_STORE (model), &iter);

        g_array_index (cells, GCell *, i)->prev = g_array_index (cells, GCell *,
i)->next;
        free (g_array_index (cells, GCell *, i));
        g_array_remove_index (cells, i);
        //CEL_array_sync ();

        gtk_tree_path_free (path);
    }
}

static GtkTreeModel *
CEL_create_and_fill_model () /* create a tree model based on GArray cells */
{
    gint i = 0;
    GtkListStore *store;
    GtkTreeIter iter;

    store = gtk_list_store_new (NUM_CEL_COLUMNS, G_TYPE_INT, G_TYPE_INT,
G_TYPE_INT, G_TYPE_FLOAT, G_TYPE_FLOAT, G_TYPE_FLOAT);

    for (i = 0; i < cells->len; i++)
    {

```

```

gtk_list_store_append (store, &iter);

gtk_list_store_set (store, &iter,
                    CEL_ID, g_array_index (cells, GCell *, i)->id,
                    CEL_RCT, g_array_index (cells, GCell *, i)->roomCount,
                    CEL_HCT, g_array_index (cells, GCell *, i)->hallCount,
                    CEL_ARA, g_array_index (cells, GCell *, i)->area,
                    CEL_SRT, g_array_index (cells, GCell *, i)->shareRate,
                    CEL_PRC, g_array_index (cells, GCell *, i)->price,
                    -1);
}
/* Append a row and fill in some data */
return GTK_TREE_MODEL (store);
}

static GtkWidget *
CEL_create_view_and_model ()
{
    GtkCellRenderer *renderer;
    GtkTreeModel *model;
    GtkWidget *view;
    view = gtk_tree_view_new ();
    model = CEL_create_and_fill_model();
    /* --- Column #1 --- */
    renderer = gtk_cell_renderer_text_new ();
    g_object_set (renderer,
                  "editable", TRUE,
                  NULL);
    g_signal_connect (renderer, "edited",
                      G_CALLBACK (CEL_cell_edited), model);
    g_object_set_data (G_OBJECT (renderer), "column", GINT_TO_POINTER (CEL_ID));
    gtk_tree_view_insert_column_with_attributes (GTK_TREE_VIEW (view),
                                                -1,
                                                "ID",
                                                renderer,
                                                "text", CEL_ID,
                                                NULL);

    /* --- Column #2 --- */
    renderer = gtk_cell_renderer_text_new ();
    g_object_set (renderer,
                  "editable", TRUE,
                  NULL);
    g_signal_connect (renderer, "edited",
                      G_CALLBACK (CEL_cell_edited), model);

```



```

g_object_set_data (G_OBJECT (renderer), "column", GINT_TO_POINTER (CEL_RCT));
gtk_tree_view_insert_column_with_attributes (GTK_TREE_VIEW (view),
                                             -1,
                                             "Room Count",
                                             renderer,
                                             "text", CEL_RCT,
                                             NULL);

/* --- Column #3 --- */
renderer = gtk_cell_renderer_text_new ();
g_object_set (renderer,
              "editable", TRUE,
              NULL);
g_signal_connect (renderer, "edited",
                  G_CALLBACK (CEL_cell_edited), model);
g_object_set_data (G_OBJECT (renderer), "column", GINT_TO_POINTER (CEL_HCT));
gtk_tree_view_insert_column_with_attributes (GTK_TREE_VIEW (view),
                                             -1,
                                             "Hall Count",
                                             renderer,
                                             "text", CEL_HCT,
                                             NULL);

/* --- Column #4 --- */
renderer = gtk_cell_renderer_text_new ();
g_object_set (renderer,
              "editable", TRUE,
              NULL);
g_signal_connect (renderer, "edited",
                  G_CALLBACK (CEL_cell_edited), model);
g_object_set_data (G_OBJECT (renderer), "column", GINT_TO_POINTER (CEL_ARA));
gtk_tree_view_insert_column_with_attributes (GTK_TREE_VIEW (view),
                                             -1,
                                             "Area",
                                             renderer,
                                             "text", CEL_ARA,
                                             NULL);

/* --- Column #5 --- */
renderer = gtk_cell_renderer_text_new ();
g_object_set (renderer,
              "editable", TRUE,
              NULL);
g_signal_connect (renderer, "edited",
                  G_CALLBACK (CEL_cell_edited), model);
g_object_set_data (G_OBJECT (renderer), "column", GINT_TO_POINTER (CEL_SRT));
gtk_tree_view_insert_column_with_attributes (GTK_TREE_VIEW (view),

```

```

-1,
"Share Rate",
renderer,
"text", CEL_SRT,
NULL);

/* --- Column #6 --- */
renderer = gtk_cell_renderer_text_new ();
g_object_set (renderer,
              "editable", TRUE,
              NULL);
g_signal_connect (renderer, "edited",
                  G_CALLBACK (CEL_cell_edited), model);
g_object_set_data (G_OBJECT (renderer), "column", GINT_TO_POINTER (CEL_PRC));
gtk_tree_view_insert_column_with_attributes (GTK_TREE_VIEW (view),
-1,
"Price",
renderer,
"text", CEL_PRC,
NULL);

gtk_tree_view_set_model (GTK_TREE_VIEW (view), model);
/* The tree view has acquired its own reference to the
 * model, so we can drop ours. That way the model will
 * be freed automatically when the tree view is destroyed */
g_object_unref (model);
return view;
}

static void
CEL_destroy_window(GtkWidget *widget,
                  gpointer data)
{
    GtkWidget *window_this = (GtkWidget *)data;
    gtk_widget_destroy(window_this);
    window_cell = NULL;
    g_return_if_fail(cells != NULL);
    CEL_head->parent->child = g_array_index (cells, GCell *, 0);
    CEL_head = NULL;
    g_array_free(cells, FALSE);
}

static GtkWidget *
CEL_new_window (gint focus) /* I hope the tree view can let me set a selection,
unfortunately it does not support this method */

```

```

{
    GtkWidget *window_cell;
    GtkWidget *view;
    GtkWidget *vbox, *hbox1, *hbox2;
    GtkWidget *button_add, *button_delete, *button_search_dialog, *button_help,
*button_close;
    GtkWidget *scrolled_window;
    gchar *window_title;

    window_cell = gtk_window_new (GTK_WINDOW_TOPLEVEL);
    //window_title = g_strdup_printf("EMR: view cells");
    window_title = g_strdup_printf("EMR: view cells >> %s> %d",
CEL_head->parent->parent->name, CEL_head->parent->id);
    gtk_window_set_default_size (GTK_WINDOW (window_cell), 600, 400);
    gtk_window_set_title (GTK_WINDOW (window_cell), window_title);
    g_free(window_title);

    g_signal_connect (window_cell, "delete_event", G_CALLBACK
(gtk_widget_destroy), NULL);

    vbox = gtk_vbox_new(FALSE, 8);
    hbox1 = gtk_hbox_new(TRUE, 5);
    hbox2 = gtk_hbox_new(TRUE, 5);

    button_add = gtk_button_new_with_label ("Add row");
    button_delete = gtk_button_new_with_label ("Remove row");
    button_search_dialog = gtk_button_new_with_label ("Search");
    button_help = gtk_button_new_with_label ("Help");
    button_close = gtk_button_new_with_label ("Close");

    scrolled_window = gtk_scrolled_window_new (NULL, NULL);
    gtk_scrolled_window_set_shadow_type (GTK_SCROLLED_WINDOW (scrolled_window),
GTK_SHADOW_ETCHED_IN);
    gtk_scrolled_window_set_policy (GTK_SCROLLED_WINDOW (scrolled_window),
GTK_POLICY_AUTOMATIC,
GTK_POLICY_AUTOMATIC);

    g_signal_connect (button_close, "clicked", G_CALLBACK (CEL_destroy_window),
window_cell);

    view = CEL_create_view_and_model ();
    gtk_widget_set_usize (scrolled_window, 580, 320);

    gtk_container_add (GTK_CONTAINER (window_cell), vbox);

```

```

gtk_box_pack_start (GTK_BOX (vbox), scrolled_window, TRUE, TRUE, 0);
gtk_container_add (GTK_CONTAINER (scrolled_window), view);
gtk_container_add (GTK_CONTAINER (vbox), hbox1);
gtk_container_add (GTK_CONTAINER (vbox), hbox2);
gtk_container_add (GTK_CONTAINER (hbox1), button_add);
gtk_container_add (GTK_CONTAINER (hbox1), button_delete);
gtk_container_add (GTK_CONTAINER (hbox2), button_search_dialog);
gtk_container_add (GTK_CONTAINER (hbox2), button_help);
gtk_container_add (GTK_CONTAINER (hbox2), button_close);

g_signal_connect (button_add, "clicked",
                  G_CALLBACK (CEL_add_item), GTK_TREE_MODEL
(gtk_tree_view_get_model(GTK_TREE_VIEW (view))));
g_signal_connect (button_delete, "clicked",
                  G_CALLBACK (CEL_remove_item), view);
return window_cell;
}

static void
CEL_open_window (GtkWidget *widget,
                 gpointer data) /* Get estate list selection and open building window
*/
{
    GtkTreeIter iter;
    GtkTreeView *treeview = (GtkTreeView *)data;
    GtkTreeModel *model = gtk_tree_view_get_model (treeview);
    GtkTreeSelection *selection = gtk_tree_view_get_selection (treeview);

    if (gtk_tree_selection_get_selected (selection, NULL, &iter))
    {
        gint i;
        GtkTreePath *path;
        GCell *cell_temp = NULL;

        path = gtk_tree_model_get_path (model, &iter);
        i = gtk_tree_path_get_indices (path)[0];
        cell_temp = CEL_head = g_array_index (buildings, GBldn *, i)->child;

        cells = g_array_sized_new (FALSE, FALSE, sizeof (GCell *), 1);

        for (;cell_temp != NULL; cell_temp = cell_temp -> next)
        {
            g_array_append_vals (cells, &cell_temp, 1);
        }
    }
}

```

```

        if (window_cell != NULL)
        {
            CEL_destroy_window(NULL, window_cell);
        }
        window_cell = CEL_new_window(0);
        gtk_widget_show_all (window_cell);

        gtk_tree_path_free (path);
    }
}

```

## core\_query.h

```

enum /* Query combo box selections */
{
    QRY_EST_NAME = 0,
    QRY_EST_DEVR,
    QRY_EST_MC,
    QRY_CEL_RCT_L,
    QRY_CEL_RCT_S,
    QRY_CEL_HCT_L,
    QRY_CEL_HCT_S,
    QRY_CEL_ARA_L,
    QRY_CEL_ARA_S,
    QRY_CEL_SRT_L,
    QRY_CEL_SRT_S,
    QRY_CEL_PRC_L,
    QRY_CEL_PRC_S,
    NUM_QRY_TYPES
};

static GtkTreeModel *
QRY_create_and_fill_model () /* Create and fill query tree model */
{
    gint i = 0;
    GtkListStore *store;
    GtkTreeIter iter;
    //struct estate *estateThis = estateHead;
    if (results == NULL)
    {
        return NULL;
    }
    store = gtk_list_store_new (2, G_TYPE_STRING, G_TYPE_STRING);

```

```

for (i = 0; i < results->len; i++)
{
    gtk_list_store_append (store, &iter);

    gtk_list_store_set (store, &iter,
        QRY_RST_PATH, g_array_index (results, GSResult, i).path,
        QRY_RST_DESC, g_array_index (results, GSResult, i).desc,
        -1);
}
/* Append a row and fill in some data */
return GTK_TREE_MODEL (store);
}

static GtkWidget *
QRY_create_view_and_model () /* Create tree view; create tree model by calling
QRY_create_and_fill_model () */
{
    GtkCellRenderer *renderer;
    GtkTreeModel *model;
    GtkWidget *view;
    view = gtk_tree_view_new ();
    model = QRY_create_and_fill_model();
    /* --- Column #1 --- */
    renderer = gtk_cell_renderer_text_new ();
    g_object_set_data (G_OBJECT (renderer), "column", GINT_TO_POINTER
(QRY_RST_PATH));
    gtk_tree_view_insert_column_with_attributes (GTK_TREE_VIEW (view),
                                                -1,
                                                "Result Path",
                                                renderer,
                                                "text", QRY_RST_PATH,
                                                NULL);

    /* --- Column #2 --- */
    renderer = gtk_cell_renderer_text_new ();
    g_object_set_data (G_OBJECT (renderer), "column", GINT_TO_POINTER
(QRY_RST_DESC));
    gtk_tree_view_insert_column_with_attributes (GTK_TREE_VIEW (view),
                                                -1,
                                                "Result Description",
                                                renderer,
                                                "text", QRY_RST_DESC,
                                                NULL);
}

```

```

gtk_tree_view_set_model (GTK_TREE_VIEW (view), model);
/* The tree view has acquired its own reference to the
 * model, so we can drop ours. That way the model will
 * be freed automatically when the tree view is destroyed */
g_object_unref (model);
return view;
}

static void
QRY_view_entry (GtkWidget *widget,
                gpointer data) /* open a window that contains the selected query
entry */
{
    GtkTreeIter iter;
    GtkTreeView *treeview = (GtkTreeView *)data;
    GtkTreeModel *model = gtk_tree_view_get_model (treeview);
    GtkTreeSelection *selection = gtk_tree_view_get_selection (treeview);

    if (gtk_tree_selection_get_selected (selection, NULL, &iter))
    {
        gint i;
        GtkTreePath *path;
        GCell *cell_temp;

        path = gtk_tree_model_get_path (model, &iter);
        i = gtk_tree_path_get_indices (path)[0];
        if (g_array_index (results, GSResult, i).type)
        {
            if (window_cell != NULL) /* Destroy previous window if already opened */
                CEL_destroy_window (NULL, window_cell);

            /* initialize a new cell view window */
            cell_temp = CEL_head = ((GCell *) g_array_index (results, GSResult,
i).entity)->parent->child;
            cells = g_array_sized_new (FALSE, FALSE, sizeof (GCell *), 1);
            for (; cell_temp != NULL; cell_temp = cell_temp -> next)
            {
                g_array_append_vals (cells, &cell_temp, 1);
            }
            window_cell = CEL_new_window(i);
            gtk_widget_show_all (window_cell);
        }
        gtk_tree_path_free (path);
    }
}

```

```

}

static void
QRY_destroy_result_window (GtkWidget *widget,
                           gpointer data)
{
    GtkWidget *window = (GtkWidget *)data;
    gtk_widget_destroy(window);
    g_return_if_fail(results != NULL);
    g_array_free(results, FALSE);
}

static GtkWidget *
QRY_new_result_window () /* create a new result window */
{
    GtkWidget *window_result;
    GtkWidget *view;
    GtkWidget *vbox, *hbox1;
    GtkWidget *button_view, *button_output, *button_help, *button_close;
    GtkWidget *scrolled_window;
    gchar *window_title;

    window_result = gtk_window_new (GTK_WINDOW_TOPLEVEL);
    window_title = g_strdup_printf("EMR: view Search Result");
    gtk_window_set_default_size (GTK_WINDOW (window_result), 600, 400);
    gtk_window_set_title (GTK_WINDOW (window_result), window_title);
    g_free(window_title);

    g_signal_connect (window_result, "delete_event", G_CALLBACK
(gtk_widget_destroy), NULL);

    vbox = gtk_vbox_new(FALSE, 8);
    hbox1 = gtk_hbox_new(TRUE, 5);

    button_view = gtk_button_new_with_label ("View");
    button_output = gtk_button_new_with_label ("Output");
    button_help = gtk_button_new_with_label ("Help");
    button_close = gtk_button_new_with_label ("Close");

    scrolled_window = gtk_scrolled_window_new (NULL, NULL);
    gtk_scrolled_window_set_shadow_type (GTK_SCROLLED_WINDOW (scrolled_window),
                                         GTK_SHADOW_ETCHED_IN);
    gtk_scrolled_window_set_policy (GTK_SCROLLED_WINDOW (scrolled_window),
                                    GTK_POLICY_AUTOMATIC,

```



```

        GTK_POLICY_AUTOMATIC);

    g_signal_connect (button_close, "clicked", G_CALLBACK
(QRY_destroy_result_window), window_result);

    view = QRY_create_view_and_model (results);
    gtk_widget_set_usize (scrolled_window, 580, 320);

    g_signal_connect (button_view, "clicked", G_CALLBACK (QRY_view_entry), view);

    gtk_container_add (GTK_CONTAINER (window_result), vbox);
    gtk_box_pack_start (GTK_BOX (vbox), scrolled_window, TRUE, TRUE, 0);
    gtk_container_add (GTK_CONTAINER (scrolled_window), view);
    gtk_container_add (GTK_CONTAINER (vbox), hbox1);
    gtk_container_add (GTK_CONTAINER (hbox1), button_view);
    gtk_container_add (GTK_CONTAINER (hbox1), button_output);
    gtk_container_add (GTK_CONTAINER (hbox1), button_help);
    gtk_container_add (GTK_CONTAINER (hbox1), button_close);

    return window_result;
}

static void
QRY_open_result_window(gint          search_type,
                      const gchar *search_content)
{
    GtkWidget *window_result;
    GEstate *estateThis = EST_head;
    GBldn *bldnThis = NULL;
    GCell *cellThis = NULL;
    GSResult result_temp;
    gint found;
    gint search_content_int, search_result_int;
    gfloat search_content_float, search_result_float;
    gchar *property_name;
    results = g_array_sized_new (FALSE, FALSE, sizeof (GSResult), 1);
    for (;estateThis != NULL; estateThis = estateThis->next)
    {
        if (search_type <= QRY_EST_MC)
        {
            switch (search_type)
            {
                case QRY_EST_NAME:
                    found = !g_strcmp0(search_content, estateThis->name);

```

```

        property_name = g_strdup ("Name");
        break;
    case QRY_EST_DEVR:
        found = !g_strcmp0(search_content, estateThis->propertyDevr);
        property_name = g_strdup ("Property Developer");
        break;
    case QRY_EST_MC:
        found = !g_strcmp0(search_content, estateThis->propertyMC);
        property_name = g_strdup ("Property Management Company");
        break;
    }
    if (found)
    {
        result_temp.type = 0;
        result_temp.entity = (gpointer) estateThis;
        result_temp.path = g_strdup_printf("\\%s", estateThis->name);
        result_temp.desc = g_strdup_printf("%s: %s", property_name,
search_content);
        g_array_append_vals (results, &result_temp, 1);
    }
}
else
{
    for (bldnThis = estateThis->child; bldnThis != NULL; bldnThis =
bldnThis->next)
    {
        for (cellThis = bldnThis->child; cellThis != NULL; cellThis =
cellThis->next)
        {
            if (search_type <= QRY_CEL_HCT_S)
                search_content_int = atoi(search_content);
            else
                search_content_float = atof(search_content);
            switch (search_type)
            {
                case QRY_CEL_RCT_L:
                    found = cellThis->roomCount >= search_content_int;
                    search_result_int = cellThis->roomCount;
                    property_name = g_strdup ("Room count");
                    break;
                case QRY_CEL_RCT_S:
                    found = cellThis->roomCount <= search_content_int;
                    search_result_int = cellThis->roomCount;
                    property_name = g_strdup ("Room count");

```

```

        break;
case QRY_CEL_HCT_L:
    found = cellThis->hallCount >= search_content_int;
    search_result_int = cellThis->hallCount;
    property_name = g_strdup ("Hall count");
    break;
case QRY_CEL_HCT_S:
    found = cellThis->hallCount <= search_content_int;
    search_result_int = cellThis->hallCount;
    property_name = g_strdup ("Hall count");
    break;
case QRY_CEL_ARA_L:
    found = cellThis->area >= search_content_float;
    search_result_float = cellThis->area;
    property_name = g_strdup ("Area");
    break;
case QRY_CEL_ARA_S:
    found = cellThis->area <= search_content_float;
    search_result_float = cellThis->area;
    property_name = g_strdup ("Area");
    break;
case QRY_CEL_SRT_L:
    found = cellThis->shareRate >= search_content_float;
    search_result_float = cellThis->shareRate;
    property_name = g_strdup ("Share rate");
    break;
case QRY_CEL_SRT_S:
    found = cellThis->shareRate <= search_content_float;
    search_result_float = cellThis->shareRate;
    property_name = g_strdup ("Share rate");
    break;
case QRY_CEL_PRC_L:
    found = cellThis->price >= search_content_float;
    search_result_float = cellThis->price;
    property_name = g_strdup ("Price");
    break;
case QRY_CEL_PRC_S:
    found = cellThis->price >= search_content_float;
    search_result_float = cellThis->price;
    property_name = g_strdup ("Price");
    break;
}
if (found)
{

```

```

        result_temp.type = 2;
        result_temp.entity = (gpointer) cellThis;
        result_temp.path = g_strdup_printf("\\\\%s >> building %d > room %d",
estateThis->name, bldnThis->id, cellThis->id);
        if (search_type <= QRY_CEL_HCT_S)
            result_temp.desc = g_strdup_printf("%s: %d %s %d",
                                                property_name,
                                                search_result_int,
                                                search_result_int >=
search_content_int?">=":"<=",
                                                search_content_int);
        else
            result_temp.desc = g_strdup_printf("%s: %f %s %f",
                                                property_name,
                                                search_result_float,
                                                search_result_float >=
search_content_float?">=":"<=",
                                                search_content_float);
        g_array_append_vals (results, &result_temp, 1);
    }
}
}
}
}
}
g_free(property_name);
window_result = QRY_new_result_window();
gtk_widget_show_all (window_result);
}

static void
QRY_main(GtkWidget *widget,
        gpointer data) /* Main function of query feature */
{
    GtkWidget *window_parent = (GtkWidget *)data;
    GtkWidget *dialog, *dialog_error;
    GtkWidget *combobox;
    GtkListStore *store;
    GtkTreeIter iter;
    GtkCellRenderer *renderer;
    GtkWidget *label_search;
    GtkWidget *entry_search;
    GtkWidget *hbox;
    gint dialog_response;

```

```

label_search = gtk_label_new("Query type:");

dialog = gtk_dialog_new_with_buttons("Query Utility",
                                     GTK_WINDOW (window_parent),
                                     GTK_DIALOG_MODAL |
GTK_DIALOG_DESTROY_WITH_PARENT,
                                     "Search",
                                     GTK_RESPONSE_YES,
                                     "Cancel",
                                     GTK_RESPONSE_CLOSE,
                                     NULL);

dialog_error = gtk_message_dialog_new (NULL,
                                       GTK_DIALOG_MODAL,
                                       GTK_MESSAGE_ERROR,
                                       GTK_BUTTONS_OK,
                                       "Invalid search content!");

/* A tree model for query type selections */
store = gtk_list_store_new (1, G_TYPE_STRING);
gtk_list_store_append (store, &iter);
gtk_list_store_set (store, &iter, 0, "Estate Name", -1);

gtk_list_store_append (store, &iter);
gtk_list_store_set (store, &iter, 0, "Estate Developer", -1);

gtk_list_store_append (store, &iter);
gtk_list_store_set (store, &iter, 0, "Estate Management Company", -1);

gtk_list_store_append (store, &iter);
gtk_list_store_set (store, &iter, 0, "Cell Room Count >=", -1);

gtk_list_store_append (store, &iter);
gtk_list_store_set (store, &iter, 0, "Cell Room Count <=", -1);

gtk_list_store_append (store, &iter);
gtk_list_store_set (store, &iter, 0, "Cell Hall Count >=", -1);

gtk_list_store_append (store, &iter);
gtk_list_store_set (store, &iter, 0, "Cell Hall Count <=", -1);

gtk_list_store_append (store, &iter);
gtk_list_store_set (store, &iter, 0, "Cell Size >=", -1);

```

```

gtk_list_store_append (store, &iter);
gtk_list_store_set (store, &iter, 0, "Cell Size <=", -1);

gtk_list_store_append (store, &iter);
gtk_list_store_set (store, &iter, 0, "Cell Share Rate >=", -1);

gtk_list_store_append (store, &iter);
gtk_list_store_set (store, &iter, 0, "Cell Share Rate <=", -1);

gtk_list_store_append (store, &iter);
gtk_list_store_set (store, &iter, 0, "Cell Price >=", -1);

gtk_list_store_append (store, &iter);
gtk_list_store_set (store, &iter, 0, "Cell Price <=", -1);

gtk_list_store_append (store, &iter);
combobox = gtk_combo_box_new_with_model(GTK_TREE_MODEL (store));
g_object_unref (store);
renderer = gtk_cell_renderer_text_new ();
gtk_cell_layout_pack_start (GTK_CELL_LAYOUT (combobox), renderer, TRUE);
gtk_cell_layout_set_attributes (GTK_CELL_LAYOUT (combobox), renderer,
                                "text", 0,
                                NULL);

entry_search = gtk_entry_new ();

hbox = gtk_hbox_new (FALSE, 8);
gtk_container_add (GTK_CONTAINER (hbox), label_search);
gtk_container_add (GTK_CONTAINER (hbox), combobox);
gtk_container_add (GTK_CONTAINER (hbox), entry_search);
gtk_box_pack_start (GTK_BOX (GTK_DIALOG (dialog)->vbox), hbox, FALSE, FALSE,
0);
gtk_widget_show_all (dialog);

/* prompt the user to input query information */
dialog_response = gtk_dialog_run(GTK_DIALOG (dialog));
if (dialog_response == GTK_RESPONSE_YES)
{
    if (gtk_entry_get_text(GTK_ENTRY (entry_search)) != NULL)
    {
        QRY_open_result_window(gtk_combo_box_get_active(GTK_COMBO_BOX
(combobox)), gtk_entry_get_text(GTK_ENTRY (entry_search)));
    }
    else

```

```
{
    gtk_dialog_run (GTK_DIALOG (dialog_error));
    gtk_widget_destroy (dialog_error);
}
}
gtk_widget_destroy (dialog);
}
```