**“程序设计基础（C语言）课程设计”报告**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 题目： | 租房管理系统 | | | |
| 班级： | 计算机类Ⅰ2305班 | | | |
| 组长： | 学号： | 20235795 | 姓名： | 张佳栋 |
| 组员： | 学号： | 20236094 | 姓名： | 方 维 |
| 组员： | 学号： | 20235892 | 姓名： | 姜舜航 |
| 组员： | 学号： | 20235948 | 姓名： | 叶瑞卿 |

2024年4月

**目 录**

[1. 概述 1](#_Toc165566779)

[1.1 问题描述 1](#_Toc165566780)

[1.2 基本要求 1](#_Toc165566781)

[1.3 人员及工作量占比 1](#_Toc165566782)

[2. 需求分析 2](#_Toc165566783)

[2.1 目标与功能 2](#_Toc165566784)

[2.2 数据来源 2](#_Toc165566785)

[2.3 输出 2](#_Toc165566786)

[2.4测试数据 2](#_Toc165566787)

[3. 总体设计 3](#_Toc165566788)

[3.1 总体功能模块 3](#_Toc165566789)

[3.2 总体数据结构 4](#_Toc165566790)

[3.3 成员分工 4](#_Toc165566791)

[4. 详细设计 5](#_Toc165566792)

[4.1 成员1详细设计 5](#_Toc165566793)

[4.1.1 算法流程设计 5](#_Toc165566794)

[4.1.2 代表性数据结构 5](#_Toc165566795)

[4.2 成员2详细设计 5](#_Toc165566796)

[4.2.1 算法流程设计 5](#_Toc165566797)

[4.2.2 代表性数据结构 5](#_Toc165566798)

[4.3 成员3详细设计 5](#_Toc165566799)

[4.3.1 算法流程设计 6](#_Toc165566800)

[4.3.2 代表性数据结构 6](#_Toc165566801)

[4.4 成员4详细设计 6](#_Toc165566802)

[4.4.1 算法流程设计 6](#_Toc165566803)

[4.4.2 代表性数据结构 6](#_Toc165566804)

[5. 调试分析 7](#_Toc165566805)

[5.1 成员1调试分析 7](#_Toc165566806)

[5.1.1 问题及解决方法 7](#_Toc165566807)

[5.1.2 讨论和分析 7](#_Toc165566808)

[5.1.3 经验与体会 7](#_Toc165566809)

[5.2 成员2调试分析 7](#_Toc165566810)

[5.2.1 问题及解决方法 7](#_Toc165566811)

[5.2.2 讨论和分析 7](#_Toc165566812)

[5.2.3 经验与体会 8](#_Toc165566813)

[5.3 成员3调试分析 8](#_Toc165566814)

[5.3.1 问题及解决方法 8](#_Toc165566815)

[5.3.2 讨论和分析 8](#_Toc165566816)

[5.3.3 经验与体会 9](#_Toc165566817)

[5.4 成员4调试分析 9](#_Toc165566818)

[5.4.1 问题及解决方法 9](#_Toc165566819)

[5.4.2 讨论和分析 9](#_Toc165566820)

[5.4.3 经验与体会 9](#_Toc165566821)

[6. 结果展示 10](#_Toc165566822)

[7. 总结与体会 11](#_Toc165566823)

[7.1 成员1总结与体会 11](#_Toc165566824)

[7.2 成员2总结与体会 11](#_Toc165566825)

[7.3 成员3总结与体会 11](#_Toc165566826)

[7.4 成员4总结与体会 12](#_Toc165566827)

# 1. 概述

## 1.1 问题描述

租房管理系统针对中介公司的租房业务进行模拟管理。中介公司拥有多个房源，包括房源信息、房主信息等。中介公司需要管理房源信息，分配中介与租客进行沟通、看房和租房手续等流程。租客可以通过中介预约看房、管理个人信息和租房信息。

## 1.2 基本要求

1. 管理员功能：
2. 能够进行中介人员、房源信息和分类信息的增删改查操作。
3. 能够分配中介处理租客的预约看房申请。
4. 能够对各类信息进行查询、排序、统计，并能进行数据维护和备份。
5. 中介功能：
6. 能够查看、管理预约看房和租房信息，包括查看、修改、删除等操作。
7. 能够对各类信息进行查询、排序、统计。
8. 能够对个人密码进行维护。
9. 租客功能：
10. 能够注册个人信息并进行管理。
11. 能够预约看房、管理看房预约信息。
12. 能够查询、排序、统计个人预约看房和租房信息。
13. 能够对个人密码进行维护。

## 1.3 人员及工作量占比

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 组长： | 学号： | 20235795 | 姓名： | 张佳栋 | 35% |
| 组员： | 学号： | 20236094 | 姓名： | 方 维 | 30% |
| 组员： | 学号： | 20235892 | 姓名： | 姜舜航 | 20% |
| 组员： | 学号： | 20235948 | 姓名： | 叶瑞卿 | 15% |

# 2. 需求分析

## 2.1 目标与功能

目标：开发一个租房管理系统，模拟中介公司的租房业务流程，包括管理员、中介和租客三个角色的功能。

功能概述：

管理员功能：管理中介人员、房源信息和分类信息，分配中介处理预约看房申请，查询、排序、统计各类信息。

中介功能：管理预约看房和租房信息，查询、排序、统计各类信息，对个人密码进行维护。

租客功能：注册个人信息，预约看房、管理预约信息，查询、排序、统计个人租房信息，对个人密码进行维护。

## 2.2 数据来源

键盘输入：用于用户输入信息，如注册信息、查询条件等。

文件读取：用于读取存储在文件中的数据，如房源信息、用户信息等。

## 2.3 输出

屏幕显示：用于展示查询结果、操作提示等信息。

文件存储：用于保存系统数据，如房源信息、用户信息、日志信息等。

## 2.4测试数据

合法数据：符合系统规定格式和要求的数据，用于正常业务流程的测试。

非法数据：不符合规定格式或要求的数据，包括格式错误、越权访问等，用于测试系统对异常情况的处理和安全性的校验。程序能够识别非法数据并给出提示。

# 3. 总体设计

## 3.1 总体功能模块

## 3.2 总体数据结构

Admin：管理员相关的结构体和功能。

Middium：中介相关的结构体和功能。

User：用户相关的结构体和功能。

Tag：用于分类标记的结构体。

Date：处理日期类型的数据。

IntList：存储整型数据的链表结构。

VoidList：通用链表的相关结构体和功能。

VoidTreap：通用Treap树的相关结构体和功能。

Hash：哈希表相关的结构体和功能。

HashTreap：哈希Treap树相关的结构体和功能。

Place：地点相关的结构体和功能。

AddressTree：管理房源的地理位置信息。

RentHouseMsg：记录租房信息。

ViewHouseMsg：记录看房信息。

## 3.3 成员分工

|  |  |
| --- | --- |
| 张佳栋 | 编写treap平衡树用于数据检索，core.h中各种操作的实现。main.c中main函数编写。 |
| 方 维 | 编写hash.c 生成哈希值、比较哈希值大小；date.c 日期类型的修改、日期的比较、不同日期间隔天数的计算；ui.c 读取键盘输入，通用界面模板。 |
| 姜舜航 | 编写void\*类型链表的增加，删除，插入，筛选满足条件的节点等函数，以适用其他数据类型的使用。 |
| 叶瑞卿 | 编写int类型链表的增加，删除，插入，筛选满足条件的节点等函数。 |

# 4. 详细设计

## 4.1 成员1详细设计

编写Treap平衡树。包括节点创建、树的初始化、节点更新、合并操作、分裂操作、节点添加和节点查询等。

### 4.1.1 算法流程设计

添加节点流程：

1. 输入数据和节点ID

2. 创建新节点

3. 分裂Treap树得到两部分 x 和 y

4. 将新节点与 x 合并得到新的 x

5. 将新的 x 和 y 合并得到最终的Treap树

查询节点数据流程：

1. 输入要查询的节点ID

2. 分裂Treap树得到三部分 x, y 和 z

3. 获取节点数据（如果存在）

4. 将 x, y 和 z 合并得到最终的Treap树

### 4.1.2 代表性数据结构

struct VoidTreapNode：Treap节点结构体，包含数据指针、节点ID、随机数、左右子节点指针和节点大小等信息。

struct VoidTreap：Treap树结构体，包含根节点指针。

## 4.2 成员2详细设计

概述性文字。

### 4.2.1 算法流程设计

### 4.2.2 代表性数据结构

## 4.3 成员3详细设计

实现各种对链表的操作，包括初始化、获取元素个数、查找特定值、转换为数组、数组转换为链表、在特定位置插入节点、删除头部/尾部/特定位置的节点以及筛选满足条件的节点。

### 程序设计流程图程序设计流程图4.3.1 算法流程设计

### 4.3.2 代表性数据结构

VoidListNode 结构，用于表示链表中的每个节点，以及一个 VoidList 结构，用于表示整个链表，其中 head 是指向链表第一个节点的指针，cnt 是链表中节点的数量。

## 4.4 成员4详细设计

关于整数链表的操作程序，其中包含了初始化整数链表、获取链表元素个数、查找特定值、链表与数组的相互转换、在链表头部或尾部添加节点、从链表头部或尾部删除节点、筛选整数链表以及清空整数链表等功能。整个程序中包含了大量的内存动态分配和释放的操作，用于管理整数链表的节点。

### 算法流程设计

初始化整数链表->获取整数链表的元素个数->在整数链表中查找特定值->整数链表转换为整数数组->整数数组转换为整数链表->在整数链表头部添加节点->在整数链表尾部添加节点->从整数链表头部删除节点->从整数链表尾部删除节点->筛选整数链表->清空整数链表

### 4.4.2 代表性数据结构

List 结构体：代表整数链表。

成员变量：

head：指向链表的头节点的指针。

cnt：记录链表中元素的个数。

IntListNode 结构体：代表整数链表的节点。

成员变量：

value：节点的值。

nxt：指向下一个节点的指针。

这两个数据结构共同构成了整数链表，其中 IntList 结构体包含了链表的头指针和元素个数，用于管理整数链表的操作。而 IntListNode 结构体则用于表示整数链表中的每个节点，包含节点的值和指向下一个节点的指针。

通过这两个数据结构的组合，程序实现了对整数链表的初始化、添加、删除、查找等操作。

# 5. 调试分析

每个成员列出调试分析过程，内容包括：调试过程中遇到的问题及解决方法，对设计与实现的回顾讨论和分析；经验与体会。

## 5.1 成员1调试分析

代码逻辑出现问题，通过调试和打印输出内容解决。

### 5.1.1 问题及解决方法

问题：程序运行结果与预期不符，有算法逻辑上的错误。

解决方法：逐步检查代码逻辑，使用调试器（如gdb）逐行调试并观察变量值，定位错误并修复。

### 5.1.2 讨论和分析

逐步调试是我的常用策略之一。通过逐行或逐步执行代码，并观察变量的值和程序的执行流程，有助于发现问题所在。利用调试工具如gdb或IDE集成的调试器，能够更快速地定位问题，查看变量、调用栈等信息，对于复杂的问题尤其有帮助。

我也会在关键代码位置添加打印语句输出变量的值或程序执行状态，这样可以更直观地跟踪代码执行流程和数据变化。同时，注意检查边界条件下的代码行为，特别是数组越界、指针操作等可能导致问题的地方。

### 5.1.3 经验与体会

调试是软件开发中不可或缺的重要环节。在调试过程中，我发现耐心和细致是非常重要的品质。解决问题需要花费时间和精力，不能急躁或心急，要仔细观察和分析代码。深入理解代码的逻辑和结构也是至关重要的，只有理解清楚代码的运行方式，才能更准确地定位和解决问题。

## 5.2 成员2调试分析

概述性文字。

### 5.2.1 问题及解决方法

### 5.2.2 讨论和分析

### 5.2.3 经验与体会

## 5.3 成员3调试分析

在编写链表操作函数时，出现了内存泄漏、删除尾部节点错误以及插入节点位置不正确的问题。解决方法包括在涉及动态内存分配的函数中加入内存释放语句、正确处理删除尾部节点时的头指针问题，以及在插入节点前进行位置合法性判断。这些改进措施有助于避免内存泄漏和链表操作错误，提高程序的稳定性和健壮性。

### 5.3.1 问题及解决方法

问题：内存泄漏在编写链表初始化函数 initVoidList 时，我没有正确释放链表的内存。这导致在一些长时间运行的程序中，内存泄漏的问题会逐渐显现出来。

解决方法： 我在每个涉及动态内存分配的函数中，包括初始化、插入节点和删除节点等函数中，都加入了相应的内存释放语句。特别是在销毁链表的函数中，确保释放了所有节点的内存。

问题：删除尾部节点时出现错误当链表只有一个节点时，调用删除尾部节点的函数会出现错误，因为没有对头指针进行正确的处理。

解决方法： 我在删除尾部节点的函数中增加了对链表是否为空的判断。如果链表只有一个节点，直接释放头指针指向的节点，并将头指针设为 NULL。

问题：插入节点位置不正确在插入节点函数 addVoidListNode 中，如果节点位置不正确，会导致插入的节点顺序错乱。

解决方法： 我在插入节点前，增加了对要插入位置是否合法的判断。如果要插入的位置是链表的尾部或者是合法的位置，才执行插入操作，否则给出错误提示。

### 5.3.2 讨论和分析

在设计和实现这些链表操作函数时，我尽量使其接口简单易用，但在实际使用过程中，仍然发现了一些问题。主要是在边界条件处理上还不够严谨。例如，在处理链表为空或链表只有一个节点时，我原本认为这些情况不会造成问题，但在实际测试和使用中发现，这些边界情况需要额外的处理才能保证程序的稳定性和正确性。

另外，我发现了一些可以优化的地方，比如在链表转换为数组的函数中，我每次都需要遍历整个链表，这在链表较长时会影响效率。我考虑可以在链表中增加一个指向尾节点的指针，这样在尾部插入节点时会更快速。

### 5.3.3 经验与体会

边界情况的处理十分重要。 在处理链表等数据结构时，要充分考虑空链表、只有一个节点、插入和删除节点等边界情况，确保程序在任何情况下都能正常运行。

代码复用和模块化设计。 将链表操作封装成函数，有助于提高代码的可读性和可维护性。同时，尽量保持函数的独立性，便于复用和调试。

内存管理是关键。 在涉及到动态内存分配的函数中，一定要小心管理内存，确保内存分配与释放的配对，并且避免内存泄漏和悬挂指针等问题。

调试和测试是不可或缺的。 尽管尽可能地设计健壮的程序，但仍然难免出现 bug。及时进行调试和测试是发现和解决问题的关键

## 5.4 成员4调试分析

• 检查是否正确引入了头文件 intlist.h，确保文件路径和文件名正确，确认函数内部逻辑是否正确，包括将链表的头指针和元素个数初始化为正确的值。确认返回链表元素个数的逻辑是否正确，确认在遍历链表查找特定值的过程中是否正确判断值相等，并返回正确的结果，确认在添加节点时，是否正确创建新节点、连接节点的指针，并更新链表的头指针和元素个数，确认在删除节点时，是否正确释放内存、更新链表的头指针和元素个数。

### 5.4.1 问题及解决方法

问题：链表返回元素个数异常

解决方法：通过添加调试输出语句或使用调试工具（如断点调试）来观察程序的执行过程和变量的值，以定位问题所在。

### 5.4.2 讨论和分析

调试分析的目标是确保程序的每个功能都能正常运行，并且没有内存泄漏或错误的指针操作。通过仔细观察和逐步调试，可以找到并解决程序中的错误和问题。

### 5.4.3 经验与体会

在遇到错误时，首先需要理解错误信息，这通常能帮助你定位问题所在。如果错误信息不明确，尝试简化代码，分段调试。在关键位置添加输出语句，查看变量的值，以确定代码的运行是否符合预期。有时，问题可能在于逻辑错误或对某个库函数的理解不准确。仔细检查代码，确保逻辑正确，对使用的函数和库有深入理解。调试可能需要时间，保持耐心，不要急于求成。有时候，暂时离开问题，稍后再回来看，可能会有新的发现。

# 6. 结果展示

# 7. 总结与体会

## 7.1 成员1总结与体会

领导一个团队完成项目需要良好的沟通和协调能力。我花了很多时间与团队成员交流、讨论和分配任务，确保每个人都明确自己的责任和任务目标。这样有助于整个团队的高效运作和项目的顺利完成。

在项目进行过程中，我发现及时解决团队成员遇到的问题和困难非常重要。我经常与团队成员保持密切联系，了解项目进展情况并及时提供帮助和支持。这种及时沟通和支持可以有效避免问题的积累和影响项目进度。

另外，作为组长我还需要具备良好的组织和规划能力。我制定了详细的项目计划和进度安排，并监督团队成员按计划执行任务。这样可以确保项目按时完成，并且在项目执行过程中可以灵活应对变化和调整任务优先级。

团队合作也是项目成功的关键。我鼓励团队成员之间相互合作、交流经验和技巧，共同解决问题，这有助于提高团队整体的协作能力和项目质量。

## 7.2 成员2总结与体会

## 7.3 成员3总结与体会

在设计这个程序时，我负责实现了链表的基本操作，包括链表的初始化、获取元素个数、查询元素是否存在、将链表转换为数组、将数组转换为链表、以及插入、删除节点等功能。以下是我的总结与体会：

达到设计题目要求与功能完善：

我设计的程序基本满足了设计题目的要求，实现了所需的链表功能。

链表操作函数的设计考虑到了常见需求，例如插入、删除、查询等，功能相对完善。

特点与不足之处：

特点： 程序具有较好的模块化和可扩展性，操作接口简单易用。

不足之处： 在处理边界情况时不够严谨，例如链表为空或只有一个节点时的处理，需要额外注意。

困难与解决方法：

困难： 在设计过程中，遇到了内存泄漏、删除尾部节点和插入节点位置不正确等问题。

解决方法： 通过仔细分析问题，增加相应的边界条件处理，并加入了正确的内存管理和释放操作，解决了这些问题。

收获与自学：

通过这次计算机技能训练，我加深了对链表数据结构的理解，提高了代码设计和实现的能力。

我学会了如何设计和实现链表结构，以及如何处理链表操作中的常见问题。

在解决问题的过程中，我学会了如何调试程序并找到 bug 的原因，提高了自己的解决问题的能力。

学习新知识和技术：

在设计链表操作函数时，我学习了如何正确地使用动态内存分配和释放，避免内存泄漏和悬挂指针等问题。

我学会了如何设计和实现函数接口，使其易用且功能清晰。

团队协作与作用发挥：

在团队协作中，我负责了链表部分的设计和实现，与其他成员协作良好，及时解决了遇到的问题。

我们相互取长补短，共同努力，最终完成了任务。在团队中，我贡献了自己的代码设计和调试经验，帮助团队更快地解决问题。

## 7.4 成员4总结与体会

1、程序逻辑：

在阅读和调试程序之前，先要对程序的逻辑有一个清晰的理解，包括数据结构的设计和函数的功能。只有理解了程序的逻辑，才能更好地定位问题和进行调试。

2、注重细节：

在调试过程中，需要注重细节和每个函数内部的逻辑。要仔细检查每个变量的赋值、条件判断和返回结果，确保每一步操作都符合预期，并及时发现和修复错误。

3、合理使用调试工具：

调试工具如断点调试可以帮助观察程序的执行过程、变量的值和内存的变化，对于定位问题非常有帮助。合理使用调试工具可以提高调试效率。

4、编写测试用例：

编写测试用例是验证函数的正确性的重要手段。通过编写各种边界条件的测试用例，可以更全面地验证函数的逻辑和处理能力，帮助发现潜在的问题。

5、持续优化：

调试分析不仅是为了修复错误，还是为了优化和提升程序的性能和质量。在调试过程中，可以思考并实现一些优化措施，如减少内存分配次数、优化遍历算法等。

通过对程序的调试分析，我深刻体会到了调试的重要性和技巧。在实际开发中，遇到问题时不要慌张，要有耐心和细心地进行排查和分析，逐步解决问题。同时，编写清晰的代码和合理的注释也能提高调试的效率和准确性。不断学习和积累调试经验，能够为程序的开发和维护提供更好的支持。