计算机应用基础与程序设计 A

（三级项目报告）

手机销售信息管理系统

项目小组	23 级计算机科学与技术 5 班		第 9 组
小组成员	学号	姓名	所做工作
	202211090252	李济岑	代码和报告二次编写，资料查找
	202311040120	刘小娴	PPT 制作及报告大纲和一次编写
开发日期	2023.11.4 - 2023.11.15		

指导老师：郜山权

摘要

[摘　要]

本文主要分为摘要、引言、目录、正文、心得及体会、附录六个部分。本报告主要记录了本小组在设计该系统的过程中的设计方案以及心得体会等内容。重点阐述了本组对手机销售管理信息系统的整体设计思路。

手机销售管理信息系统是生活中常见的一种管理系统设计，具有广泛的应用。本小组尝试以 C 语言实现一个小型的手机销售管理信息系统，具有添加信息、更改信息、删除信息、查询用户信息、保存和读取手机销售信息文件等功能。在实现过程中，采用了二叉树的数据结构，指针和函数等 C 语言编程方法，并且使用了动态内存分配和字符串库等功能进行实际开发。

[关键词]　C 语言；文件读写；链表与二叉树；动态内存分配；模块化。

引言

计算机科学与技术的迅速发展使得信息管理系统在各个领域中得到广泛应用。随着移动设备的普及，手机信息管理系统成为一个备受关注的话题，它不仅能够提高信息的整合和管理效率，还能够为用户提供更便捷、快速的服务。本论文旨在介绍并详细阐述一款由本文作者及其团队开发的基于 C 语言的手机信息管理系统。

本系统的设计灵感来源于对现代社会信息化需求的深刻理解，以及对 C 语言这一经典编程语言的深入研究。该系统不仅具备基本的信息管理功能，如增、删、改、查等，还包含了文件读写、获取当前时间以及随机生成 11 位订单号等实用功能。为了提升用户体验，我们还采用一定程度的用户界面设计，使系统更加直观、友好。

通过这个项目，我们旨在展示在 C 语言编程环境下，如何构建一个功能丰富、高效稳定的手机信息管理系统。同时，我们将深入讨论系统设计的各个方面，包括界面设计、数据结构选择、算法优化等。在项目的实现过程中，我们充分发挥团队协作的优势，通过有效的沟通和协同工作，开展这一挑战性的任务。

本项目报告将分为多个部分，首先将介绍系统的背景和目的，然后详细阐述系统的设计与实现过程，包括功能模块划分、数据结构选择、算法设计等。接着，我们将展示系统的具体功能和效果，并通过实例验证系统的可行性和稳定性。最后，我们将总结整个项目的经验和教训，在此基础上进行展望。

目录

正文

一、研究内容的基本原理

本项目的主要工作是实现一个手机销售信息管理系统，主要考察对 C 语言的运用。项目指导书还要求采用链表（树）、函数等方式实现该函数。

二、所采用的研究方法和工具

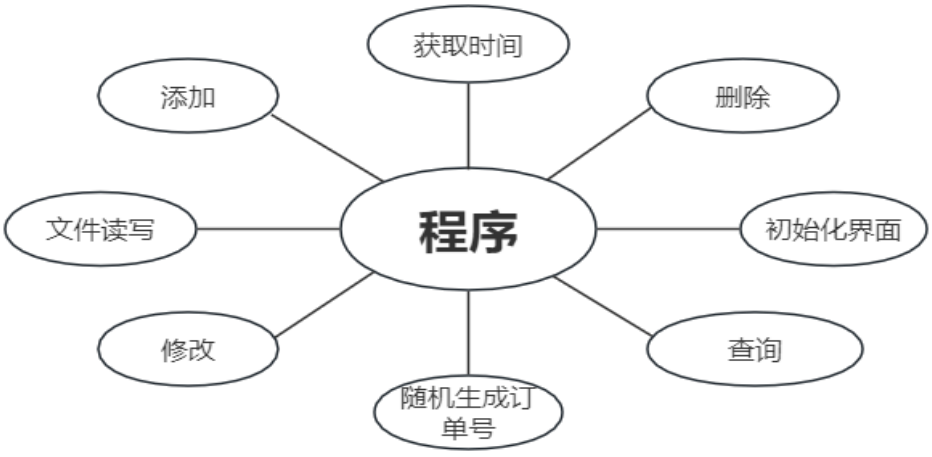
本次三级项目采用小组成员合作的方式进行。小组成员在课外时间进行面对面讨论和线上交流合作，合力完成本项目的编写和制作。

小组成员通过查阅 C 语言教科书和网上的资料，以 Visual Studio Code 为文本编辑器，CLion 为主要代码编写工具完成该项目的开发（以 Clang-tidy 作为编译核心，遵循 C11 国际语言标准）。并以 Word 和 PowerPoint 软件完成项目报告撰写和答辩 PPT 制作等工作。

三、项目的方案设计

（一）项目功能设计

在功能方面，根据项目指导书和实际的信息管理系统应用，经过小组讨论得出如下的信息管理系统的项目功能设计：



其中，信息的添加、删除、修改、查询四大模块为项目指导书所明确要求的核心功能，也是本信息管理系统的四大主要功能。本系统通过该四大功能实现对数据的处理。

同时创新扩展了文件读写、初始化用户图形界面、获取当前时间、随机生成订单号等辅助功能，使该项目更贴近于实际开发，让该信息管理系统更智能化、现代化，提升用户实际使用感受。

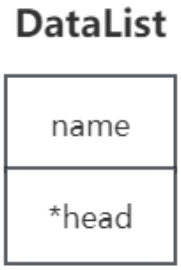
（二）结构体设计

为方便对数据进行处理，采用结构体对不同数据进行不同的封装抽象，将每个结构体抽象为一个节点，通过访问结构体进行访问数据和数据处理。经小组讨论，对数据封装成如下结构体：

1. DataList 节点封装如下:

```
typedef struct DataList  {
    UserNode *head;                    // 指向主链表的头节点
    char listName[LIST_NAME_LENGTH]; // 存储链表名
} DataList;
```

该结构体 DataList 包含 head（结构体指针）和 listName（字符串）两个属性。其中，listName 存储该手机信息管理系统的名称，head 指针指向存储数据的链式结构。DataList 作为存储结构的开头，并将其抽象为如下节点：



2. UserNode 节点封装如下：

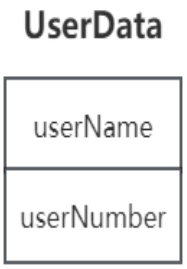
```
typedef struct UserNode{
    UserData data;          // 存储用户信息
    SaleNode *sale;         // 该 UserNode 下的销售信息子链表
    struct UserNode *next; // 指向下一个 UserNode 节点
} UserNode;
```

该结构体 UserNode 包含 data（结构体）、sale（结构体指针）和 next（同类型结构体指针）三个属性。其中，data 存储手机销售的用户信息的节点，sale 指向该用户的所有手机销售信息，next 指向下一级 UserNode 节点形成链式结构。UserNode 作为用户节点，并将其抽象为如下节点：

3. UserData 节点封装如下：

```
typedef struct UserData  {
    char userName[USER_NAME_LENGTH];      // 存储用户名
    char userNumber[USER_NUMBER_LENGTH]; // 存储用户手机号
} UserData;
```

该结构体 UserData 包含 userName（字符串）、userNumber（字符串）两个属性。其中，userName 存储用户名，userNumber 存储用户手机号。UserData 作为用户信息节点，将其抽象为如下节点。



4. SaleNode 节点封装如下：

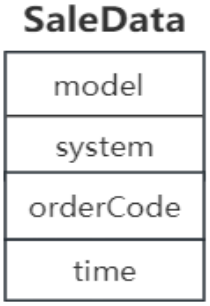
```
typedef struct SaleNode  {
    SaleData data;
    struct SaleNode *bro;
} SaleNode;
```

该结构体 SaleNode 包含 data（结构体）、bro（同类型结构体指针）两个属性。其中，data 存储该用户下的销售信息的节点，bro 指向下一级 SaleNode 节点形成链式结构。SaleNode 作为销售节点，将其抽象为如下节点。

5. SaleData 节点封装如下：

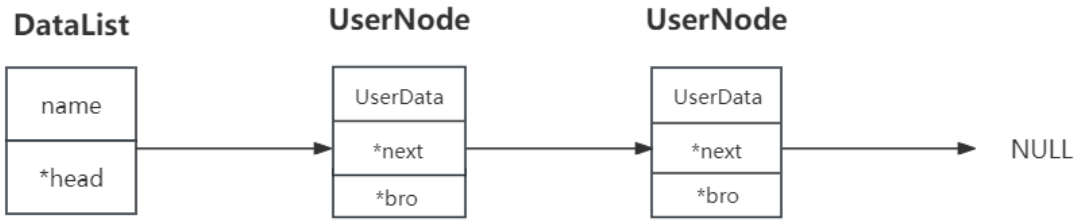
```
typedef struct SaleData {
    char model[PHONE_MODEL_LENGTH];           // 存储手机型号
    char system[PHONE_SYSTEM_LENGTH];         // 存储手机系统
    char orderCode[PHONE_ORDER_CODE_LENGTH]; // 存储订单号
    char time[TIME_NOW_LENGTH];               // 存储销售时间
} SaleData;
```

该结构体 SaleData 包含 model（字符串）、system（字符串）、orderCode（字符串）、time（字符串）四个属性。其中，model 存储手机型号，system 存储手机操作系统，orderCode 存储订单号，time 存储销售时间。SaleData 作为该用户销售节点，将其抽象为如下节点。



（三）数据结构设计

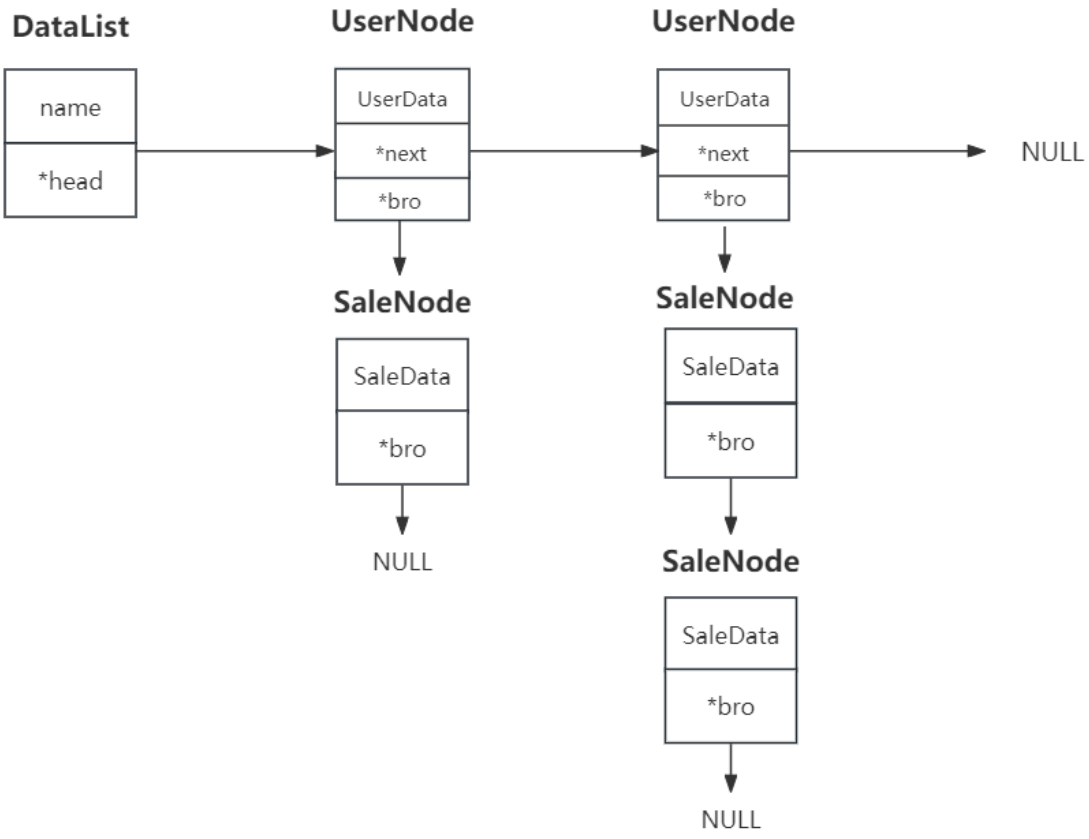
在数据结构方面，为了满足手机销售信息管理系统整体设计要求，信息应呈链式结构进行存储。根据查询资料和项目指导书的要求，经小组成员讨论，最初合作讨论出如下单链表的数据结构：



最初采用了单链表的数据结构。其中，以结构体作为链表的节点。DataList 结构代表该数据链表的起始节点，UserNode 作为存储信息的节点，直到 UserNode 指向 NULL 代表链表截止。

此时假设 UserData 存储了全部的信息，现实中的手机销售信息管理系统常用手机号或用户名进行搜索销售信息，但如果存在一个人有多条销售信息，则无法查询到后一个数据。

为解决这一问题，我们在原有单链表的数据结构上进行了如下改进：



我们将原有单链表结构优化为多条单链表结构组成的二叉树，该二叉树以 UserNode 为节点的链表成为主链表，以 SaleNode 为节点的链表作为子链表。

UserData 中仅存储用户信息，该 UserNoda 下的 SaleData 存储该用户的销售信息。

在实际对数据进行增删改查进行操作时，利用深度优先搜索（DFS）遍历整个二叉树，即通过一次查询定位到用户节点，再通过二次查询定位到该用户下的销售节点，避免了一个用户多条销售信息无法完全查询的问题。

四、项目函数及算法设计

（一）项目大纲设计

经小组讨论，该项目使用模块化编程，即将程序不同的功能进行封装，使整个项目更容易被维护和管理，在调试过程中更容易定位到 BUG。通过函数间的相互调用实现项目功能集成。

为使代码耦合性更低，经讨论我们将程序分成如下 19 个函数（加上 main 函数共 20 个），使每个函数尽可能仅完成单一功能：

```
// const用于构造函数内部方式误操作修改固定的数据
UserNode *createUser(const char *userName, const char *userPhone);
SaleNode *createSale(const char *model, const char *system, int type, co
           nst char *Date,  const char *Code);
UserNode *searchUser(const DataList *list, const char *userPhone);
SaleNode *searchSale(const UserNode *userNode, const char *orderCode);
DataList *readToFile(char *listName, const char *dest);
void printNode(const UserNode *userNode, const SaleNode *saleNode);
void printList(const DataList *list);
void deleteUser(UserNode *before);
void deleteSale(SaleNode *before);
int askJudge(int type);
void initInter(DataList *list);
void addModule(DataList *list);
void searchModule(const DataList *list);
void deleteModule(DataList *list);
void changeModule(DataList *list);
void randCode(char *code);
void getTimeNow(char *time);
void saveToFile(const DataList *list, const char *dest);
void freeList(DataList *list);
```

［注］：下述中仅展示主要函数的流程和效果，具体函数的完整代码见附录 B

除此之外，为了提高代码复用性，使代码更加易于维护，我们小组经过讨论使用了大量的定义宏。大量的宏使代码简化，且在编译时进行计算，能够提升程序的性能，经讨论设计定义宏如下：

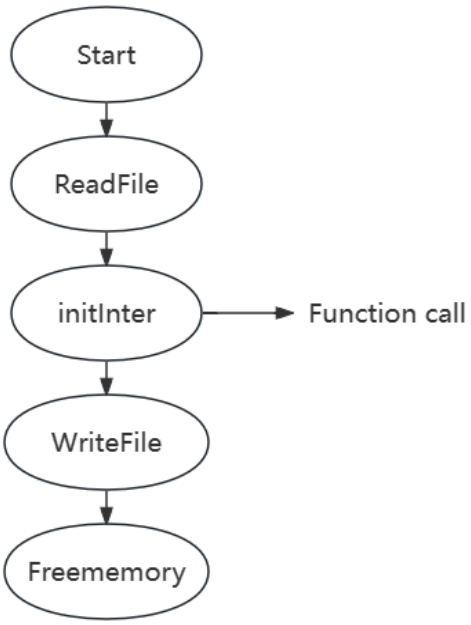
```
#define PHONE_MODEL_LENGTH 40      // 存储 手机型号 字符串长度
#define PHONE_SYSTEM_LENGTH 40     // 存储 手机系统 字符串长度
#define PHONE_ORDER_CODE_LENGTH 20 // 存储 订单号    字符串长度
#define TIME_NOW_LENGTH 40         // 存储 销售时间 字符串长度
#define USER_NAME_LENGTH 40        // 存储 用户名    字符串长度
#define USER_NUMBER_LENGTH 20      // 存储 用户号码 字符串长度
#define LIST_NAME_LENGTH 40        // 存储 链表名    字符串长度
#define STAMP_HEAD printf("======================%s=======================
===\n\n", list->listName);
#define STAMP_FORMAT printf("UserName\tUserPhone\tModel\tSystem\tTime\t\
tOrderCode\n");
#define STAMP_LINE_ONE printf("----------------------------------------
---------------------------------\n");
#define STAMP_LINE_TWO printf("========================================
=================================\n");
#define STAMP_LINE_THREE printf("++++++++++++++++++++++++++++++++++++++
++++++++++++++++++++++++++++++++\n");
#define STAMP_VERSION printf("
                                                version: 1.0.0\n");
#define STAMP_AUTHOR printf("
                                                Author: LeeJc02 from YS
U\n");
```

（二）主要函数及算法设计

经小组讨论，我们分为三个部分完成主要函数及算法设计：1.主函数设计，2.界面设计，3.命令设计。其中，主函数设计主要设计主函数对各个函数的整体调；界面设计主要设计用户交互界面，处理程序的功能调用；命令设计主要实现具体的功能。下面我们将分类详细介绍各个部分的具体设计：

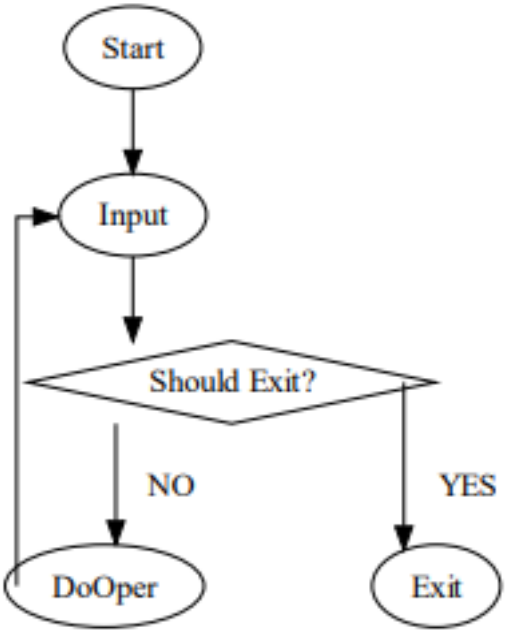
1. 主函数设计

在模块化编程中我们通常通过主函数对多个模块的内容进行调用，实现多个模块的功能集成。main 主函数设计的流程图如下：

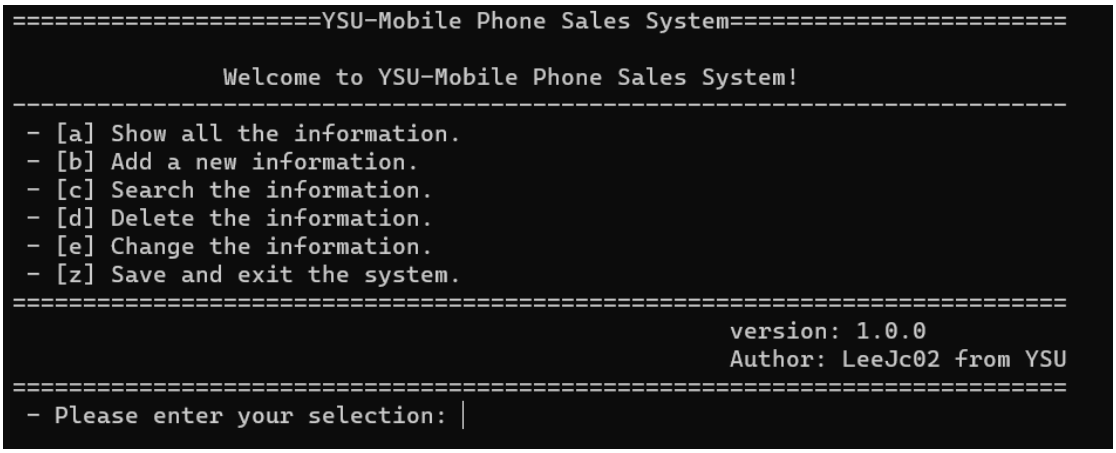


2. 界面设计

程序与人交互的地方就是程序的界面，本项目所要设计的手机销售信息管理系统需要多种操作，输出很多数据。因此，我们设计出一种不断循环读取命令并处理的循环式交互界面。initInter 界面设计流程图如下：



initInter 函数初始化用户界面后，进入菜单，并通过不同的输入进行不同的操作。该功能运行效果如下：



3. 命令设计

在命令设计中，小组成员综合考虑了项目要求和实际代码能力，共商议出以下几个命令并通过交互界面进行循环调用和处理：

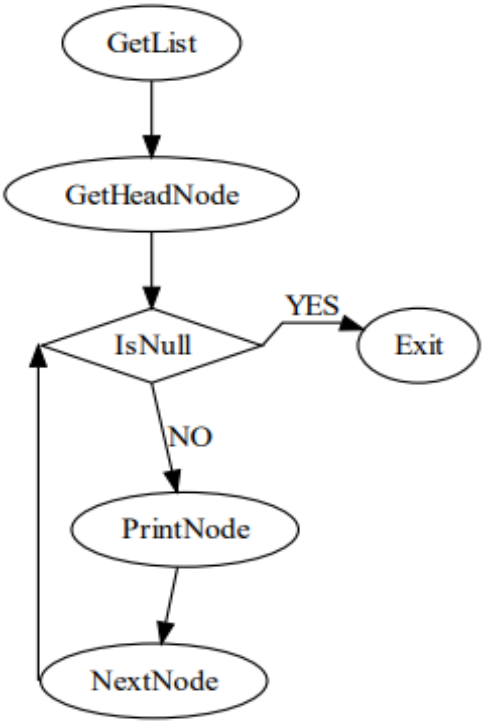
- 打印当前全部销售数据
- 添加新的销售数据
- 通过手机号查询并输出全部数据
- 删除查询到的销售数据
- 修改查询到的销售数据

在实际程序设计中，我们通过对 main 函数中的 DataList 的指针在对多个封装的函数调用时进行传参操作并进行数据处理下面依次介绍每个命令函数的设计思路和具体实现：

打印当前全部销售数据

该功能本质上就是应用深度优先搜索 DFS（Depth-First Search）算法遍历整个二叉树，并结合 UserData 中的用户信息和 SaleNode 中的销售信息，输出每条销售数据的过程。

printList 函数流程图如下：



printList 函数调用后，打印全部的销售数据。该功能运行效果如下：

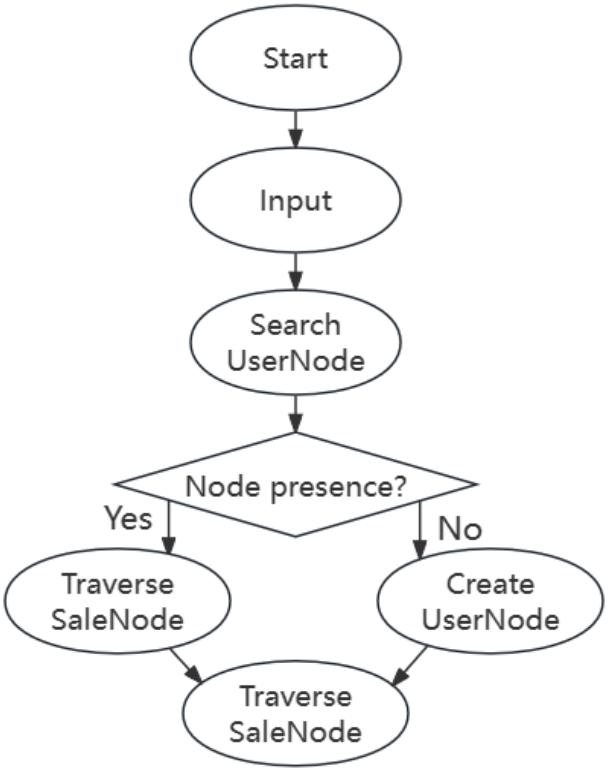
```
======================YSU-Mobile Phone Sales System=========================

                          All sale information!
--------------------------------------------------------------------------------
UserName        UserPhone       Model   System  Time            OrderCode
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
Lee             18081278716     Honor   MagicOS 2023-11-19      166986
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
 - [Notice] Print successfully!
 - Please enter to return the main menu!
```

添加新的销售数据

该功能输入添加数据后首先查询是否存在该用户是否存在，若存在则仅添加到该用户节点 UserNode 下的 SaleNode；若不存在则添加到主链表尾创建一个新的 UserNode 和该 UserNode 下的第一个 SaleNode。

addModule 函数流程图如下：



addModule 函数调用后，先输入添加的数据，在确认后进行添加操作。该功能运行效果如下：

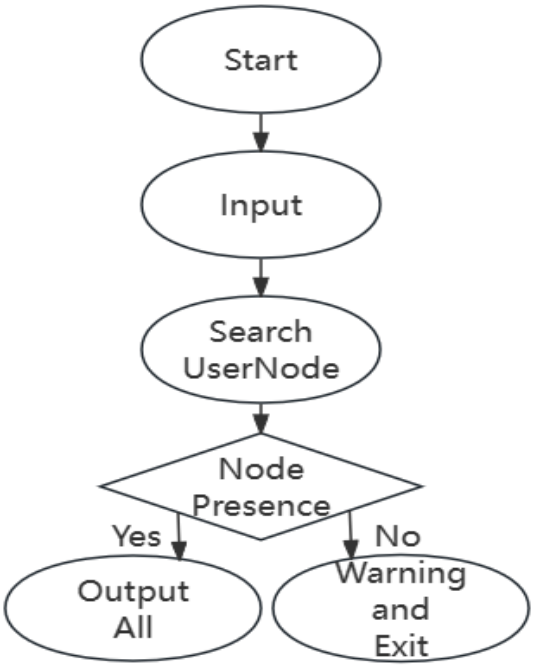
```
======================YSU-Mobile Phone Sales System=========================

                       Add a new sale information!
--------------------------------------------------------------------------
 - Please enter the user name: Lee
 - Please enter the user phone: 18081278716
 - Please enter the phone model: Huawei
 - Please enter the phone system: Harmony
 - [Notice] Are you sure to add the information? [y/n]: y
 - [Notice] Add successfully!
 - Please enter to return the main menu!
```

通过手机号查询并输出全部数据

该功能仅通过手机号查询到相应的用户节点 UserNode，并打印该 UserNode 下的所有 SaleNode 销售节点。

searchModule 函数流程图如下：



searchModule 函数调用后，打印全部手机号查询到的数据。该功能运行效果如下：

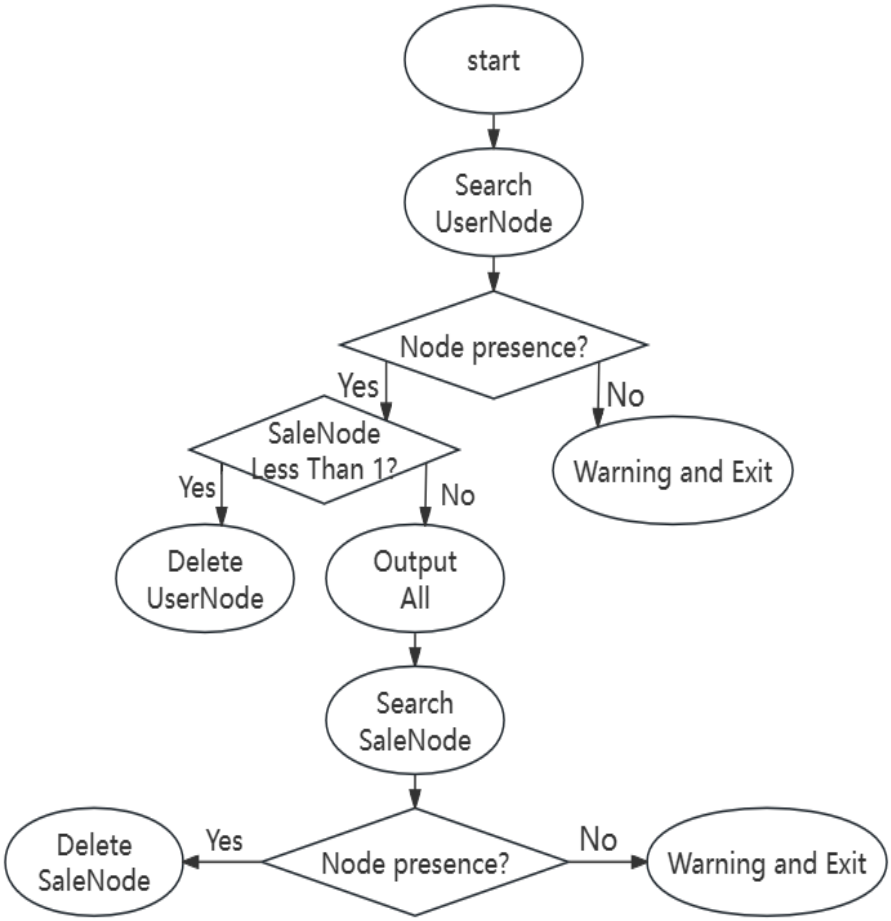
```
======================YSU-Mobile Phone Sales System=========================

                     Search the sale information!
--------------------------------------------------------------------------------
 - Please enter the user phone: 18081278716
--------------------------------------------------------------------------------
UserName        UserPhone       Model   System  Time            OrderCode
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
Lee             18081278716     Honor   MagicOS 2023-11-19      166986
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
Lee             18081278716     Huawei  Harmony 2023-12-10      75702908317
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
 - [Notice] Search successfully!
 - Please enter to return the main menu!
```

删除查询到的销售数据

该功能先通过手机号查询用户节点 UserNode，若该用户节点仅下仅有一个 SaleNode 销售节点，则直接删除该用户节点；反之则进行订单号二次查询定位到精确的 SaleNode，并仅删除该销售节点。上述查询若查询失败则弹出提示信息并返回。

deleteModule 函数流程图如下：



deleteModule 函数调用后，删除查询到的数据。该功能运行效果如下：

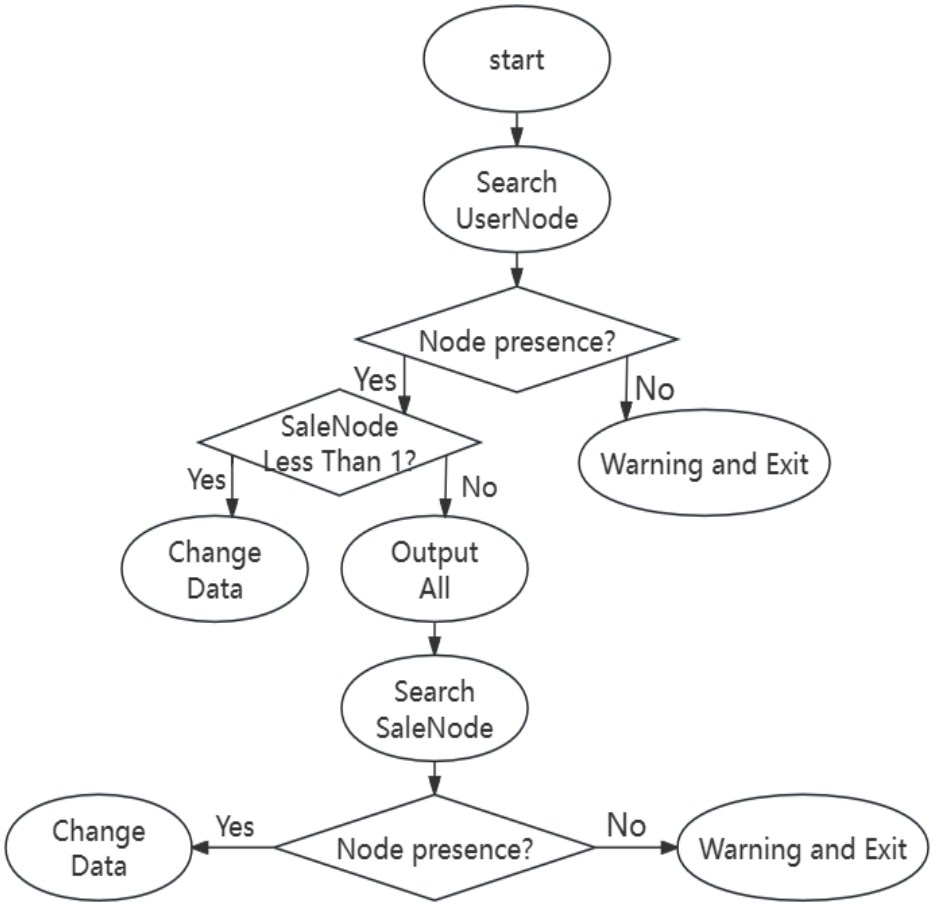
```
======================YSU-Mobile Phone Sales System=========================

                Delete the sale information!
--------------------------------------------------------------------------
 - Please enter the user phone: 18081278716
UserName        UserPhone       Model   System  Time            OrderCode
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
Lee             18081278716     Honor   MagicOS 2023-11-13      14639205518
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
Lee             18081278716     Huawei  Harmony 2023-11-19      16698652604
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
 - Please enter the order code: 14639205518
--------------------------------------------------------------------------
UserName        UserPhone       Model   System  Time            OrderCode
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
Lee             18081278716     Honor   MagicOS 2023-11-13      14639205518
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
 - [Notice] Are you sure to delete the information? [y/n]: y
 - [Notice] Delete successfully!
 - Please enter to return the main menu!
```

修改查询到的销售数据

该功能和删除功能类似，一次查询若仅有一个 SaleNode 销售节点则直接进行修改操作，反之则二次查询定位到精确的 SaleNode 再进行修改操作。

changeModule 函数流程图如下：



changeModule 函数调用后，修改查询到的数据。该功能运行效果如下：

```
======================YSU-Mobile Phone Sales System=======================

                     Change the sale information!
--------------------------------------------------------------------------
 - Please enter the user phone: 18081278716
UserName        UserPhone       Model   System  Time            OrderCode
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
Lee             18081278716     Huawei  Harmony 2023-11-19      16698652604
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
 - Please enter the user name: Lee
 - Please enter the phone model: Honor
 - Please enter the phone system: MagicOS
 - [Notice] Are you sure to change the information? [y/n]: y
 - [Notice] Change successfully!
 - Please enter to return the main menu!
```

（三）辅助函数及算法设计

除主要功能函数外，经小组讨论，我们在项目原有的基础上进行新增了如下的辅助功能函数，以便项目实际功能更接近真实和自动化处理：

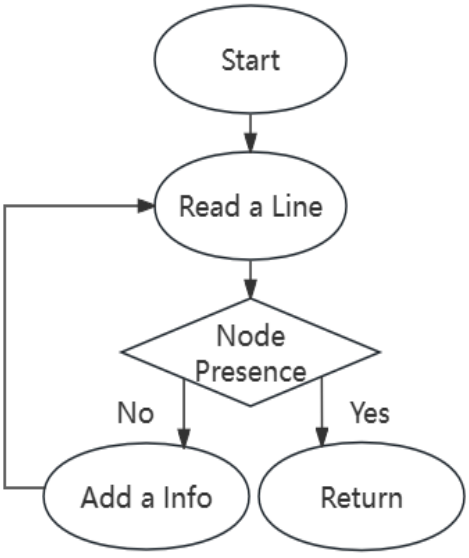
- 文件读取
- 文件写入
- 随机生成订单号
- 获取当前时间
- 释放动态内存

这些函数并不直接实现项目的主要功能，而是支持、补充或组织主要功能的实现。下面依次介绍每个辅助函数的算法设计和具体实现：

文件读取

经小组讨论，我们将读取数据文件实现读档，并通过读取算法将 csv 文件中读取到的数据初始化位二叉树上的节点。若 csv 文件不存在则自动创建空白文件。

readToFile 函数流程图如下：

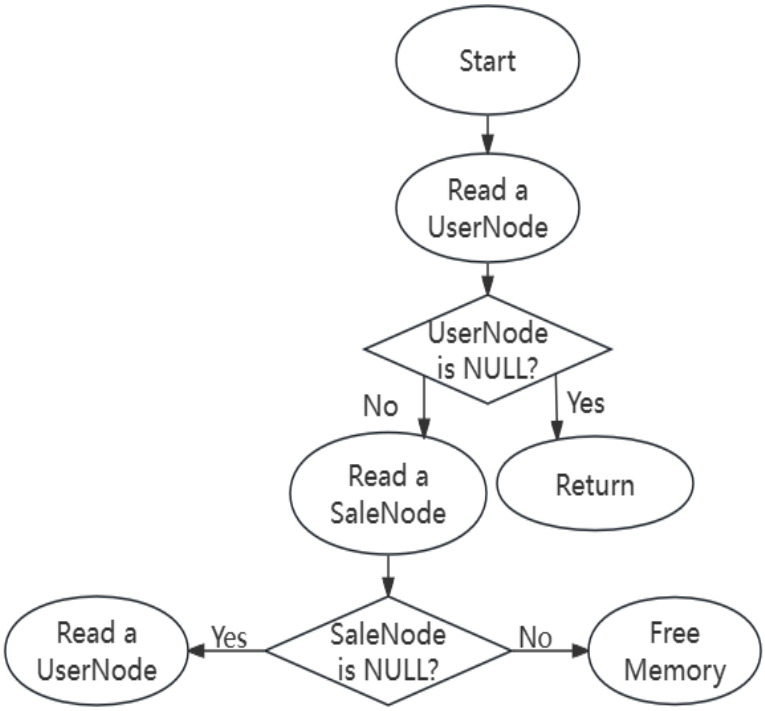


文件写入

手机销售的数据持久化存储是本项目的亮点之一，经过小组讨论，我们将 UserData 中的用户信息和 SaleData 中的销售信息结合起来，通过保存算法进行

文件写入。该功能用于将二叉中的数据写入 main 文件同级目录下的 csv 文件。

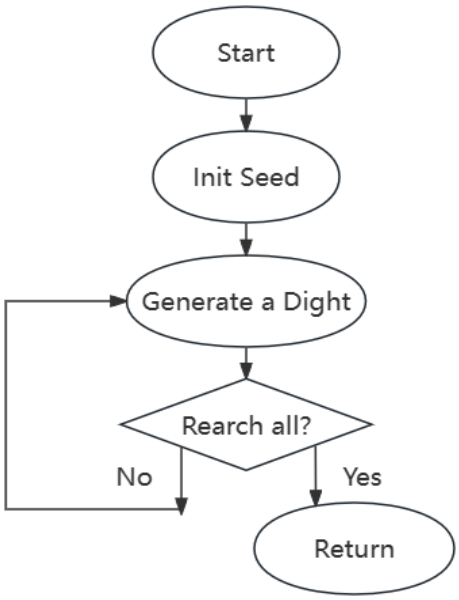
saveToFile 函数流程图如下：



随机生成订单号

实际的手机销售信息管理系统中，订单号应该是自动录入而非手动录入。该功能用于自动生成 11 位订单号并进行写入和存储。

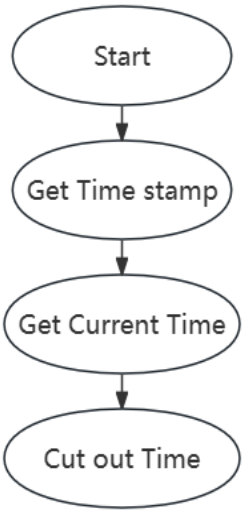
randCode 函数流程图如下：



获取当前时间

实际的手机销售信息管理系统中，时间也应该是自动录入而非手动录入。该功能用于自动获取当前时间并进行写入和存储。

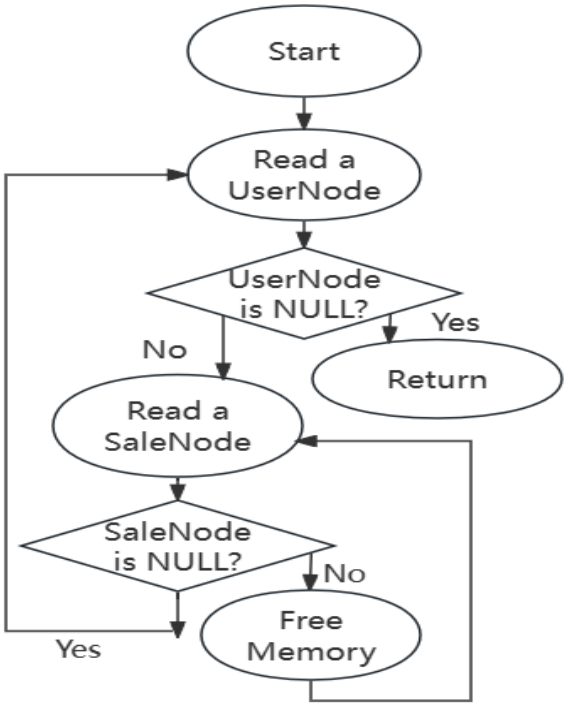
getTimeNow 函数流程图如下：



释放动态内存

该功能用于动态释放二叉树的内存。手动开辟在内存堆上的内存用户手动释放是个好习惯。

freeList 函数流程图如下：



五、项目总结及展望

在该项目中，我们小组精诚合作，积极探讨项目设计和查阅资料，付出了极大的心血。经小组讨论，我们将以创新方面、遇到的困难、不足与展望三个方面进行如下展开论述：

（一）创新方面

1. **结构体方面：**

采用结构体的层层嵌套，并将其进行抽象为二叉树的节点，避免一个结构体内含多项数据所导致的层次混乱，和单个结构体内存过大问题。

2. **数据结构方面：**

采用了二叉树的数据结构，在链式数据存储的基础上方便对数据进行增删改查操作，并且避免了多个符合要求数据仅能查找到一个的问题。

3. **耦合性方面：**

为提升代码内聚性，降低代码耦合性，将代码分割成多个函数，使函数尽量只实现单一功能，并减少全局变量的使用，尽量避免涟漪效应。

4. **存储方面：**

引用了文件读写模块，让程序对数据执行操作之前读取已存档数据，并操作结束后进行保存，长时间存储，提升了程序复用性。

5. **程序开发方面：**

在具体项目开发中，采用了动态内存管理，在内存堆上对数据的内存进行手动开辟和释放，使程序开发过程中更加自由和灵活。

6. **辅助功能方面：**

引用了获取当前时间 getTimeNow 和随机生成订单号 randCode 等辅助函数，使项目更自动化智能化，更贴近实际项目应用。

（二）遇到的困难

1. **如何降低耦合性，避免涟漪效应**[1]

解决方法：尽量使每个函数实现单一功能，减少函数间关联和全局变量。

2.**程序异常退出**[2]

解决方法：访问某些值为 NULL 的指针导致内存出错，通过调试避免使用时指针值为 NULL，在访问前有限检测是否值为 NULL。

3.**VScode/CLion/VS 等内置终端运行出错**[3]

解决方法：内置终端不支持 system 系统级指令，将内置终端的 IDE 或文本编辑器更改设置为外置终端进行运行。

4.**fscanf/fgets 读取文件时部分内容被覆盖**[4]

解决方法：数组长度过小，导致数组溢出对后续数据进行覆盖，更改数组长度；局部变量实际上为内存栈上的残余数据值，在使用前用 memset 函数对其进行初始化清空操作。

（三）不足与展望

1.**结构方面：**

增删改查操作都是基于数据结构的遍历查询，未能将二叉树进行结构优化，导致二叉树的高度过高，查询效率低。

2.**功能方面：**

该项目作为一个信息处理的系统，缺乏登陆、注册、加密等功能，开发功能不完善，同时用户图形化界面简陋（主要是 C 语言图形化仅有 EasyX 图形库）。

3.**应用方面：**

作为一个销售信息管理系统，仅能单用户使用，未设置多用户信息交互功能或部署在服务器上供多人同时共享信息和进行同步操作。

4.**存储方面：**

存储数据仅为单个文件的文件读写，未能使用数据库对数据进行存储，数据存储不安全且读写效率低。

本项目在小组成员的共同努力下，虽然依然存在诸多不足，但依然让我们受益匪浅。在查阅相关资料和积极讨论交流之下，我们也将在后续的学习之中，利用所学的知识在该项目基础上继续开发完善。

心得体会

本次三级项目为编写一个小型管理系统，我们小组的选题为“手机销售信息管理系统”。做一个三级项目对于刚刚进入大学学习的我们大一新生来说极具挑战性。但是我们小组每个人都迎难而上，永不言弃。每天我们都在图书馆里一起讨论项目的进展和实现细节，不时提出一些全新的思路和方法，找出项目中的亮点和不足。

在本次项目中，我们提出了使用文件 IO 来进行保存和读取手机销售信息的功能，然而，使用文件 IO 涉及到了内存和文件系统的细节，难度比较大。我们在遇到问题之后，通过广泛的查找资料，查看网络上的代码和教程，最终攻克了文件 IO 和动态内存分配的问题，实现了该功能。

在实现代码的过程中，难免会遇到出人意料的 Bug，在解决这些 Bug 的过程中，我们小组成员学习了如何使用 GDB 进行动态调试，如何在代码中设置断点，如何使用步过和步进等调试功能……我们相信，这些技能都将为我们将来的学习提供帮助。

在处理文件读写中，我们遇到了 fscanf 和 fgets 函数部分数据内存区覆盖问题。如果不能妥善处理好这个问题，将导致字符串写入时内容遭到覆盖，以至于其他内存出现问题。为了解决这一问题，我们研究了未初始化的局部变量并非是随机值，而是内存栈中残余的数据。同时我们还学习到了相关内存栈和内存堆的有关知识，这帮助了我们更加深入了解 C 语言的内存和底层原理。

在这个项目进行过程中，每个人都展现出了对项目的责任感和设计项目的积极性。最好完成了这个项目之后，每个人都感觉到了实打实的进步，通过大家精诚合作努力完成一件事，最终证明了自己的能力。

附录

A 参考文献

[1] 知乎：耦合度以及降低耦合性的方法
https://zhuanlan.zhihu.com/p/579691985

[2] 21xrx.com：C++程序异常退出：如何排查和解决问题？
https://21xrx.com/Articles/read_article/180574

[3] CSDN：解决CLion中system()函数调试时终端乱码问题
https://blog.csdn.net/weixin_73186358/article/details/133944130

[4] CSDN：在C语言中,定义局部变量时如果未初始化,则值是随机的。为什么?
https://blog.csdn.net/yuechifanfan/article/details/111315390

[5] C Reference：Malloc函数所属头文件包含问题
https://en.cppreference.com/w/cn

B 完整程序源代码 GitHub：https://github.com/LeeJc02/YSU_Project

```
//YSU-计算机程序设计 A 三级项目:手机销售信息管理系统
#include <stdio.h>
#include <stdlib.h>
#include <sys/time.h>
#include <string.h>

// 用宏提高代码复用性，更易于维护
#define PHONE_MODEL_LENGTH 40      // 存储  手机型号  字符串长度
#define PHONE_SYSTEM_LENGTH 40     // 存储  手机系统  字符串长度
#define PHONE_ORDER_CODE_LENGTH 20 // 存储  订单号    字符串长度
#define TIME_NOW_LENGTH 40         // 存储  销售时间  字符串长度
#define USER_NAME_LENGTH 40        // 存储  用户名    字符串长度
#define USER_NUMBER_LENGTH 20      // 存储  用户号码  字符串长度
#define LIST_NAME_LENGTH 40        // 存储  链表名    字符串长度
#define STAMP_HEAD
printf("======================%s=======================\n\n",
list->listName);
#define STAMP_FORMAT
printf("UserName\tUserPhone\tModel\tSystem\tTime\t\tOrderCode\n");
#define STAMP_LINE_ONE
printf("-----------------------------------------------------------------
------------\n");
#define STAMP_LINE_TWO
printf("=================================================================
============\n");
#define STAMP_LINE_THREE
printf("+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++++++++\n");
#define STAMP_VERSION
printf("                                                  version:
1.0.0\n");
#define STAMP_AUTHOR
printf("                                                  Author: LeeJc02
from YSU\n");

// 存储销售的手机参数和销售信息
typedef struct SaleData
{
    char model[PHONE_MODEL_LENGTH];          // 存储手机型号
    char system[PHONE_SYSTEM_LENGTH];        // 存储手机系统
```

```
    char orderCode[PHONE_ORDER_CODE_LENGTH]; // 存储订单号
    char time[TIME_NOW_LENGTH];              // 存储销售时间
} SaleData;

// 将 SaleData 和下一级指针抽象为新的一个节点并存储在 UserNode 之下
typedef struct SaleNode
{
    SaleData data;
    struct SaleNode *bro;
} SaleNode;

// 存储销售的用户信息
typedef struct UserData
{
    char userName[USER_NAME_LENGTH];     // 存储用户名
    char userNumber[USER_NUMBER_LENGTH]; // 存储用户手机号
} UserData;

// 将 UserData 和下一级指针抽象为新的一个节点并存储在 DataList 之下作为主链表
typedef struct UserNode
{
    UserData data;         // 存储用户信息
    SaleNode *sale;        // 该 UserNode 下的销售信息子链表
    struct UserNode *next; // 指向下一个 UserNode 节点
} UserNode;

// 存储链表名链表头作为链表的开始
typedef struct DataList
{
    UserNode *head;                  // 指向主链表的头节点
    char listName[LIST_NAME_LENGTH]; // 存储链表名
} DataList;

// const 用于构造函数内部方式误操作修改固定的数据
UserNode *createUser(const char *userName, const char
*userPhone);                                          // 新建一个 UserNode 节
点并初始化
SaleNode *createSale(const char *model, const char *system, int type, const
char *Date, const char *Code); // 新建一个 SaleNode 节点并根据传入的 type 参
数进行不同初始化
```

```
UserNode *searchUser(const DataList *list, const char
*userPhone);                                          // 查找所查询的
UserNode 节点，并返回该节点的前一个节点
SaleNode *searchSale(const UserNode *userNode, const char
*orderCode);                                        // 查找所查询的 SaleNode 节
点，并返回该节点的前一个节点
DataList *readToFile(char *listName, const char
*dest);                                                  // 从文件中读取
数据并初始化为链表
void printNode(const UserNode *userNode, const SaleNode
*saleNode);                                          // 打印当前节点数据
void printList(const DataList
*list);
  // 打印整条链表数据
void deleteUser(UserNode
*before);
       // 删除传入节点的下一个 UserNode 并释放内存
void deleteSale(SaleNode
*before);
       // 删除传入节点的下一个 SaleNode 并释放内存
int askJudge(int
type);
              // 各种操作前询问是否执行，确认返回为 1，取消返回为 0
void initInter(DataList
*list);
       // 初始化系统主菜单
void addModule(DataList
*list);
       // 实现添加一个销售信息的模块
void searchModule(const DataList
*list);                                                                  /
/ 实现查询一个用户信息的模块
void deleteModule(DataList
*list);
     // 实现删除一个销售信息的模块
void changeModule(DataList
*list);
     // 实现修改一个销售信息的模块
void randCode(char
*code);
            // 随机生成 11 位的订单号
```

```
void getTimeNow(char
*time);
          // 获取当前时间
void saveToFile(const DataList *list, const char
*dest);                                                    // 保存数据并写入
csv 文件中
void freeList(DataList
*list);
         // 手动释放链表内存

int main()
{
    DataList *list = NULL;
    char listName[LIST_NAME_LENGTH] = "YSU-Mobile Phone Sales System"; //
链表名
    char fileName[LIST_NAME_LENGTH] = "saleData.csv";                  //
存储数据文件名
    list = readToFile(listName, fileName);                             //
从文件中读取存储的数据
    initInter(list);                                                  //
初始化用户界面
    saveToFile(list, fileName);                                        //
保存数据并写入文件中
    freeList(list);                                                   //
手动释放链表内存
    return 0;
}

/*
 * UserNode *createUser(char *, char *)
 * - 新建一个 UserNode 节点并初始化
 */
UserNode *createUser(const char *userName, const char *userPhone)
{
    UserNode *userNode = (UserNode *)malloc(sizeof(UserNode));
    memset(userNode, 0, sizeof(UserNode)); // 初始化该 UserNode 节点，防止
内存栈中残留数据造成数据影响，下述所有 memset 函数同理
    strcpy(userNode->data.userName, userName);
    strcpy(userNode->data.userNumber, userPhone);
    userNode->sale = NULL;
    userNode->next = NULL;
    return userNode; // 返回所创造的该 UserNode 节点
```

```
}

/*
 * SaleNode *createSale(char *, char *, int, char *, char *)
 * - 新建一个 SaleNode 节点并根据传入的 type 参数进行不同初始化
 */
SaleNode *createSale(const char *model, const char *system, int type, const
char *Date, const char *Code)
{
    char time[TIME_NOW_LENGTH];
    char code[PHONE_ORDER_CODE_LENGTH];
    memset(time, 0, sizeof(time));
    memset(code, 0, sizeof(code));

    if (!type) // 通过传入状态参数 type 的值，判定是否需要生成时间和订单号
    {
        getTimeNow(time); // 表示新建销售订单，获取当前时间
        randCode(code);   // 随机生成订单号
    }
    else
    {
        strcpy(time, Date); // 仅是修改该销售订单，无需生成时间和订单号
        strcpy(code, Code);
    }

    SaleNode *saleNode = (SaleNode *)malloc(sizeof(SaleNode));
    memset(saleNode, 0, sizeof(SaleNode));
    strcpy(saleNode->data.orderCode, code);
    strcpy(saleNode->data.model, model);
    strcpy(saleNode->data.system, system);
    strcpy(saleNode->data.time, time);
    saleNode->bro = NULL;
    return saleNode; // 返回所创造的该 SaleNode 节点
}

/*
 * UserNode *searchUser(DataList *, char *)
 * - 查找所查询的 UserNode 节点，并返回该节点的前一个节点
 * - 确保 UserNode 的节点数大于 1
 */
UserNode *searchUser(const DataList *list, const char *userPhone)
```

```
{
    UserNode *userNode = list->head;
    while (userNode->next != NULL) // 遍历整个链表进行数据匹配
    {
        if (strcmp(userNode->next->data.userNumber, userPhone) == 0)
            return userNode; // 匹配成功则返回查询节点的上一个节点
        userNode = userNode->next;
    }
    return NULL; // 匹配失败则返回 NULL 表示查询为空
}

/*
 * SaleNode *searchSale(UserNode *, char *)
 * - 查找所查询的 SaleNode 节点，并返回该节点的前一个节点
 * - 确保该 UserNode 下 SaleNode 的节点数大于 1
 */
SaleNode *searchSale(const UserNode *userNode, const char *orderCode)
{
    SaleNode *saleNode = userNode->sale;
    while (saleNode->bro != NULL) // 遍历该 UserNode 下的所有 SaleNode 节点进
行数据匹配
    {
        if (strcmp(saleNode->bro->data.orderCode, orderCode) == 0)
            return saleNode; // 匹配成功则返回查询节点的上一个节点
        saleNode = saleNode->bro;
    }
    return NULL; // 匹配失败则返回 NULL 表示查询为空
}

/*
 * DataList *readToFile(char *, char *)
 * - 从文件中读取数据并初始化为链表
 */
DataList *readToFile(char *listName, const char *dest)
{
    DataList *list = (DataList *)malloc(sizeof(DataList)); // 初始化链表
    memset(list, 0, sizeof(DataList));
    strcpy(list->listName, listName);
    list->head = NULL;
    UserNode *userNode = NULL;
    SaleNode *saleNode = NULL;
```

```
    FILE *file = fopen(dest, "r"); // 先用 r 只读模式，若文件存在则读取
    if (file == NULL)              // 若文件打开失败则不存在该文件
    {
        file = fopen(dest, "w+"); // 用 w+读写模式生成一个新的文件用于存储信息
        if (file == NULL)         // 若还是打开失败则报错
        {
            fprintf(stderr, " - File open failed!\n");
            return NULL;
        }
        return list;
    }

    char userName[USER_NAME_LENGTH];
    char userPhone[USER_NUMBER_LENGTH];
    char model[PHONE_MODEL_LENGTH];
    char system[PHONE_SYSTEM_LENGTH];
    char orderCode[PHONE_ORDER_CODE_LENGTH];
    char time[TIME_NOW_LENGTH];
    memset(userName, 0, USER_NAME_LENGTH); // 全是为了防止内存栈中残留数据造成数据影响
    memset(userPhone, 0, USER_NUMBER_LENGTH);
    memset(model, 0, PHONE_MODEL_LENGTH);
    memset(system, 0, PHONE_SYSTEM_LENGTH);
    memset(orderCode, 0, PHONE_ORDER_CODE_LENGTH);
    memset(time, 0, TIME_NOW_LENGTH);

    // 用 fscanf 函数进行文件读取操作，读取到文件末尾时返回 EOF
    while (fscanf(file, "%s %s %s %s %s %s", userName, userPhone, model, system, time, orderCode) != EOF)
    {
        UserNode *flag = NULL;
        if (list->head == NULL) // 通过链表是否为空判断是否为第一个节点进行新建节点操作
            flag = NULL;
        else if (list->head->next == NULL && strcmp(list->head->data.userNumber, userPhone) == 0) // 若链表只有一个节点且为一个用户的多个销售数据
            flag = list->head;
        else // 若链表有多个节点则进行查找是否是一个用户的多个销售信息
```

```
        {
            flag = searchUser(list, userPhone); // 此时链表节点数一定大于 1，查询是否存在该用户
            if (flag != NULL)                   // 如果返回为 NULL 则表示未查询到用户，否则返回查询到的节点的上一个节点
                flag = flag->next;
        }

        if (flag == NULL) // 若不存在该用户则新建一个节点
        {
            if (list->head == NULL) // 判定是否为第一个节点
            {
                list->head = createUser(userName, userPhone);
                userNode = list->head;
            }
            else
            {
                userNode->next = createUser(userName, userPhone);
                userNode = userNode->next;
            }
            userNode->sale = createSale(model, system, 1, time, orderCode);
        }
        else // 若用户存在则在该用户下新建一个销售信息节点
        {
            saleNode = flag->sale;
            while (saleNode->bro != NULL)
                saleNode = saleNode->bro;
            saleNode->bro = createSale(model, system, 1, time, orderCode);
        }
    }

    if (feof(file)) // 判断是否读取到文件末尾，若是则关闭文件
        fclose(file);
    return list;
}

/*
 * void printNode(UserNode *, SaleNode *)
 * - 打印当前节点数据
 */
void printNode(const UserNode *userNode, const SaleNode *saleNode)
```

```
{
    printf("%s\t\t%s\t%s\t%s\t%s\t%s\n", userNode->data.userName,
userNode->data.userNumber, saleNode->data.model,
           saleNode->data.system, saleNode->data.time,
saleNode->data.orderCode);
    STAMP_LINE_THREE
}

/*
 * void printList(DataList *)
 * - 打印整条链表数据
 */
void printList(const DataList *list)
{
    system("cls"); // 清屏操作
    STAMP_HEAD     // 界面初始化
        printf("                          All sale information!\n");
    STAMP_LINE_ONE

    if (list->head == NULL) // 空链表则警告并返回
        printf(" - [Warning] There is no information!\n");
    else
    {
        STAMP_FORMAT
        STAMP_LINE_THREE
        UserNode *userNode = list->head;
        SaleNode *saleNode = NULL;
        while (userNode != NULL) // 遍历所有主链表和子链表以打印所有数据
        {
            saleNode = userNode->sale;
            while (saleNode != NULL)
            {
                printNode(userNode, saleNode);
                saleNode = saleNode->bro;
            }
            userNode = userNode->next;
        }
        printf(" - [Notice] Print successfully!\n"); // 提示信息
    }
```

```
    printf(" - Please enter to return the main menu!\n"); // 返回主菜单提
示信息
    getchar();
}

/*
 * void deleteUser(UserNode *)
 * - 删除传入节点的下一个 UserNode 并释放内存
 * - 确保 UserNode 节点数大于 1
 */
void deleteUser(UserNode *before)
{
    UserNode *userNode = before->next;
    before->next = userNode->next;
    free(userNode);
}

/*
 * void saveToFile(DataList *, char *)
 * - 删除传入节点的下一个 SaleNode 并释放内存
 * - 确保该 UserNode 下 SaleNode 节点数大于 1
 */
void deleteSale(SaleNode *before)
{
    SaleNode *saleNode = before->bro;
    before->bro = saleNode->bro;
    free(saleNode);
}

/*
 * int askJudge(int)
 * - 各种操作前询问是否执行，确认返回为 1，取消返回为 0
 */
int askJudge(int type)
{
    char choice; // select the type you are operating on
    switch (type)
    {
    case 0:
        printf(" - [Notice] Are you sure to add the information? [y/n]: ");
        break;
```

```
    case 1:
        printf(" - [Notice] Are you sure to delete the information? [y/n]:
");
        break;
    case 2:
        printf(" - [Notice] Are you sure to change the information? [y/n]:
");
        break;
    default:
        printf(" - [Warning] Please enter the correct number!\n");
        return -1; // 错误传参，异常返回
    }

    scanf("%c", &choice);
    getchar(); // 清除 scanf 函数输入后参与在输入缓冲区的一个换行符，下述所有
scanf 函数后的 getchar 函数同理
    if (choice == 'y')
        return 1;
    return 0;
}

/*
 * void initInter(DataList *)
 * - 初始化系统主菜单
 */
void initInter(DataList *list)
{
    while (1) // 循环使用主菜单，直到用户退出并保存
    {
        system("cls"); // 清屏操作
        STAMP_HEAD     // 界面初始化
            printf("               Welcome to %s!\n", list->listName);
        STAMP_LINE_ONE

        char choice; // 操作选项
        printf(" - [a] Show all the information.\n");
        printf(" - [b] Add a new information.\n");
        printf(" - [c] Search the information.\n");
        printf(" - [d] Delete the information.\n");
        printf(" - [e] Change the information.\n");
        printf(" - [z] Save and exit the system.\n");
```

```
        STAMP_LINE_TWO
        STAMP_VERSION
        STAMP_AUTHOR
        STAMP_LINE_TWO

        printf(" - Please enter your selection: ");
        scanf("%c", &choice);
        getchar();
        switch (choice)
        {
        case 'a':
            printList(list); // 打印操作
            break;
        case 'b':
            addModule(list); // 添加操作
            break;
        case 'c':
            searchModule(list); // 查询操作
            break;
        case 'd':
            deleteModule(list); // 删除操作
            break;
        case 'e':
            changeModule(list); // 修改操作
            break;
        case 'z':
            printf(" - Thank you for using the %s!\n", list->listName);
            return; // 退出系统
        default:
            printf(" - [Warning] Enter error, please re-enter!\n"); // 错
误输入提醒
            getchar();                                                  // 停顿
等待用户回车刷新界面重新输入，下述所有非 scanf 函数后的 getchar 函数同理
            break;
        }
    }
}

/*
 * void addModule(DataList *)
```

```
 * - 实现添加一个销售信息的模块
 */
void addModule(DataList *list)
{
    system("cls"); // 清屏操作
    STAMP_HEAD     // 界面初始化
        printf("                        Add a new sale information!\n");
    STAMP_LINE_ONE

    char userName[USER_NAME_LENGTH];
    char userPhone[USER_NUMBER_LENGTH];
    char model[PHONE_MODEL_LENGTH];
    char system[PHONE_SYSTEM_LENGTH];
    memset(userName, 0, USER_NAME_LENGTH);
    memset(userPhone, 0, USER_NUMBER_LENGTH);
    memset(model, 0, PHONE_MODEL_LENGTH);
    memset(system, 0, PHONE_SYSTEM_LENGTH);

    printf(" - Please enter the user name: ");
    scanf("%s", userName);
    getchar();
    printf(" - Please enter the user phone: ");
    scanf("%s", userPhone);
    getchar();
    printf(" - Please enter the phone model: ");
    scanf("%s", model);
    getchar();
    printf(" - Please enter the phone system: ");
    scanf("%s", system);
    getchar();

    if (!askJudge(0)) // 确认操作
    {
        printf(" - [Warning] Operation canceled!\n");
        printf(" - Please enter to return the main menu!\n");
        getchar();
        return;
    }

    UserNode *userNode = NULL;
    if (list->head == NULL) // 若链表为空则新建一个节点
```

```
    {
        list->head = createUser(userName, userPhone);
        list->head->sale = createSale(model, system, 0, NULL, NULL);
    }
    else if (list->head->next == NULL &&
strcmp(list->head->data.userNumber, userPhone) == 0) // 若链表只有一个节点
且为一个用户的多个销售数据
    {
        SaleNode *tempSale = list->head->sale;
        while (tempSale->bro != NULL)
            tempSale = tempSale->bro;
        tempSale->bro = createSale(model, system, 0, NULL, NULL);
    }
    else // 若链表有多个节点则进行查找是否是一个用户的多个销售信息
    {
        userNode = searchUser(list, userPhone);
        if (userNode == NULL) // 若不存在该用户则新建一个节点
        {
            UserNode *tempUser = list->head;
            while (tempUser->next != NULL) // 遍历 DataList 下的主链表
                tempUser = tempUser->next;
            tempUser->next = createUser(userName, userPhone);
            tempUser->next->sale = createSale(model, system, 0, NULL,
NULL);
        }
        else // 若用户存在则在该用户下新建一个销售信息节点
        {
            userNode = userNode->next;
            SaleNode *tempSale = userNode->sale;
            while (tempSale->bro != NULL) // 遍历 UserNode 下的子链表
                tempSale = tempSale->bro;
            tempSale->bro = createSale(model, system, 0, NULL, NULL);
        }
    }

    printf(" - [Notice] Add successfully!\n");
    printf(" - Please enter to return the main menu!\n");
    getchar();
}

/*
```

```
 * void searchModule(DataList *)
 * - 实现查询一个用户信息的模块
 */
void searchModule(const DataList *list)
{
    system("cls"); // 清屏操作
    STAMP_HEAD     // 界面初始化
        printf("                      Search the sale information!\n");
    STAMP_LINE_ONE

    char userPhone[USER_NUMBER_LENGTH];
    printf(" - Please enter the user phone: ");
    scanf("%s", userPhone); // 输入用户手机号进行遍历查询
    getchar();

    UserNode *userNode = NULL;
    if (list->head == NULL) // 空链表则未查询到
        userNode = NULL;
    else if (list->head->next == NULL &&
strcmp(list->head->data.userNumber, userPhone) == 0) // 若链表只有一个节点且为一个用户的多个销售数据
        userNode = list->head;
    else // 若链表有多个节点则进行查找是否是一个用户的多个销售信息
    {
        userNode = searchUser(list, userPhone);
        if (userNode != NULL)
            userNode = userNode->next;
    }

    if (userNode == NULL) // 若不存在该用户则警告并返回
    {
        printf(" - [Warning] The information does not exist!\n");
        printf(" - Please enter to return the main menu!\n");
        getchar();
        return;
    }

    SaleNode *saleNode = userNode->sale;
    STAMP_LINE_ONE
    STAMP_FORMAT
    STAMP_LINE_THREE
```

```
    while (saleNode != NULL) // 遍历该用户下的所有销售信息并打印
    {
        printNode(userNode, saleNode);
        saleNode = saleNode->bro;
    }

    printf(" - [Notice] Search successfully!\n");
    printf(" - Please enter to return the main menu!\n");
    getchar();
}

/*
 * void deleteModule(DataList *)
 * - 实现删除一个销售信息的模块
 */
void deleteModule(DataList *list)
{
    system("cls"); // 清屏操作
    STAMP_HEAD     // 界面初始化
        printf("                Delete the sale information!\n");
    STAMP_LINE_ONE

    char userPhone[USER_NUMBER_LENGTH];
    char orderCode[PHONE_ORDER_CODE_LENGTH];
    memset(userPhone, 0, USER_NUMBER_LENGTH);
    memset(orderCode, 0, PHONE_ORDER_CODE_LENGTH);
    printf(" - Please enter the user phone: ");
    scanf("%s", userPhone); // 输入用户手机号进行遍历查询
    getchar();

    UserNode *userNode = NULL;
    UserNode *userBefore = NULL; // 用于记录查询节点的上一个节点
    if (list->head == NULL)      // 空链表则未查询到
        userNode = NULL;
    else if (list->head->next == NULL &&
strcmp(list->head->data.userNumber, userPhone) == 0) // 若链表只有一个节点且为一个用户的多个销售数据
        userNode = list->head;
    else // 若链表有多个节点则进行查找是否是一个用户的多个销售信息
    {
```

```
        userBefore = searchUser(list, userPhone);
        if (userBefore != NULL)
            userNode = userBefore->next;
    }

    if (userNode == NULL) // 若不存在该用户则警告并返回
    {
        printf(" - [Warning] The information does not exist!\n");
        printf(" - Please enter to return the main menu!\n");
        getchar();
        return;
    }

    SaleNode *saleNode = userNode->sale;
    STAMP_FORMAT
    STAMP_LINE_THREE
    while (saleNode != NULL) // 遍历该用户下的所有销售信息并打印
    {
        printNode(userNode, saleNode);
        saleNode = saleNode->bro;
    }

    if (userNode->sale->bro == NULL) // 若该用户只有一个销售信息则直接删除该
用户节点 (UserNode 下的 SaleNode 一定大于 0)
    {
        if (!askJudge(1)) // 确认操作
        {
            printf(" - [Warning] Operation canceled!\n");
            printf(" - Please enter to return the main menu!\n");
            getchar();
            return;
        }

        if (userBefore == NULL) // 若该用户为第一个节点则直接删除
        {
            list->head = NULL;
            free(userNode);
        }
        else // 若该用户不是第一个节点则传入前一个节点删除该节点
            deleteUser(userBefore);
    }
```

```
    else // 若该用户有多个销售信息则输入订单号进行二次查询
    {
        printf(" - Please enter the order code: ");
        scanf("%s", orderCode);
        getchar();

        if (strcmp(orderCode, userNode->sale->data.orderCode) == 0) // 若
订单号与第一个 SaleNode 匹配
            saleNode = userNode->sale;
        else // 若订单号与第一个 SaleNode 不匹配则进行二次查询
            saleNode = searchSale(userNode, orderCode);

        if (saleNode == NULL) // 若不存在该销售信息则警告并返回
        {
            printf(" - [Warning] The information does not exist!\n");
            printf(" - Please enter to return the main menu!\n");
            getchar();
            return;
        }

        STAMP_LINE_ONE
        STAMP_FORMAT
        STAMP_LINE_THREE
        printNode(userNode, saleNode); // 打印所查询到的销售信息

        if (!askJudge(1)) // 确认操作
        {
            printf(" - [Warning] Operation canceled!\n");
            printf(" - Please enter to return the main menu!\n");
            getchar();
            return;
        }

        if (strcmp(orderCode, saleNode->data.orderCode) == 0) // 若订单号
与第一个 SaleNode 匹配
        {
            userNode->sale = saleNode->bro;
            free(saleNode);
        }
        else // 若订单号与第一个 SaleNode 不匹配则传入前一个节点删除该节点
            deleteSale(saleNode);
```

```
    }

    printf(" - [Notice] Delete successfully!\n");
    printf(" - Please enter to return the main menu!\n");
    getchar();
}

/*
 * void changeModule(DataList *)
 * - 实现修改一个销售信息的模块
 */
void changeModule(DataList *list)
{
    system("cls"); // 清屏操作
    STAMP_HEAD     // 界面初始化
        printf("                        Change the sale information!\n");
    STAMP_LINE_ONE

    char orderCode[PHONE_ORDER_CODE_LENGTH];
    char userName[USER_NAME_LENGTH];
    char userPhone[USER_NUMBER_LENGTH];
    char model[PHONE_MODEL_LENGTH];
    char system[PHONE_SYSTEM_LENGTH];
    memset(userName, 0, USER_NAME_LENGTH);
    memset(userPhone, 0, USER_NUMBER_LENGTH);
    memset(model, 0, PHONE_MODEL_LENGTH);
    memset(system, 0, PHONE_SYSTEM_LENGTH);
    memset(orderCode, 0, PHONE_ORDER_CODE_LENGTH);

    printf(" - Please enter the user phone: ");
    scanf("%s", userPhone); // 输入用户手机号进行遍历查询
    getchar();
    UserNode *userNode = NULL;

    if (list->head == NULL) // 空链表则未查询到
        userNode = NULL;
    else if (list->head->next == NULL &&
strcmp(list->head->data.userNumber, userPhone) == 0) // 若链表只有一个节点且为一个用户的多个销售数据
        userNode = list->head;
    else // 若链表有多个节点则进行查找是否是一个用户的多个销售信息
```

```
    {
        userNode = searchUser(list, userPhone);
        if (userNode != NULL)
            userNode = userNode->next;
    }

    if (userNode == NULL) // 若不存在该用户则警告并返回
    {
        printf(" - [Warning] The information does not exist!\n");
        printf(" - Please enter to return the main menu!\n");
        getchar();
        return;
    }

    SaleNode *saleNode = userNode->sale;
    STAMP_FORMAT
    STAMP_LINE_THREE
    while (saleNode != NULL) // 遍历该用户下的所有销售信息并打印
    {
        printNode(userNode, saleNode);
        saleNode = saleNode->bro;
    } // as with the deleteModule

    if (userNode->sale->bro == NULL) // 若该用户只有一个销售信息则直接修改该
用户节点 (UserNode 下的 SaleNode 一定大于 0)
    {
        printf(" - Please enter the user name: "); // 重新输入节点信息
        scanf("%s", userName);
        getchar();
        printf(" - Please enter the phone model: ");
        scanf("%s", model);
        getchar();
        printf(" - Please enter the phone system: ");
        scanf("%s", system);
        getchar();

        if (!askJudge(2)) // 确认操作
        {
            printf(" - [Warning] Operation canceled!\n");
            printf(" - Please enter to return the main menu!\n");
            getchar();
```

```
            return;
        }

        char *readDate = (char *)malloc(sizeof(char) * TIME_NOW_LENGTH);
        getTimeNow(readDate);                       // 获取当前时间
        strcpy(userNode->data.userName, userName); // 覆盖性修改节点信息
        strcpy(userNode->sale->data.model, model);
        strcpy(userNode->sale->data.system, system);
        strcpy(userNode->sale->data.time, readDate);
    }
    else // 若该用户有多个销售信息则输入订单号进行二次查询
    {
        printf(" - Please enter the order code: ");
        scanf("%s", orderCode);
        getchar();

        if (strcmp(orderCode, userNode->sale->data.orderCode) == 0) // 若
订单号与第一个 SaleNode 匹配
            saleNode = userNode->sale;
        else // 若订单号与第一个 SaleNode 不匹配则进行二次查询
            saleNode = searchSale(userNode, orderCode)->bro;

        if (saleNode == NULL) // 若不存在该销售信息则警告并返回
        {
            printf(" - [Warning] The information does not exist!\n");
            printf(" - Please enter to return the main menu!\n");
            getchar();
            return;
        }

        STAMP_LINE_ONE
        STAMP_FORMAT
        STAMP_LINE_THREE
        printNode(userNode, saleNode); // 打印所查询到的销售信息

        printf(" - Please enter the new user name: "); // 重新输入节点信息
        scanf("%s", userName);
        getchar();
        printf(" - Please enter the new phone model: ");
        scanf("%s", model);
        getchar();
```

```
        printf(" - Please enter the new phone system: ");
        scanf("%s", system);
        getchar();

        if (!askJudge(2)) // 确认操作
        {
            printf(" - [Warning] Operation canceled!\n");
            printf(" - Please enter to return the main menu!\n");
            getchar();
            return;
        }

        char *readDate = (char *)malloc(sizeof(char) * TIME_NOW_LENGTH);
        memset(readDate, 0, TIME_NOW_LENGTH);
        getTimeNow(readDate);                       // 获取当前时间
        strcpy(userNode->data.userName, userName); // 覆盖性修改节点信息
        strcpy(userNode->data.userNumber, userPhone);
        strcpy(saleNode->data.model, model);
        strcpy(saleNode->data.system, system);
        strcpy(userNode->sale->data.time, readDate);
    }

    printf(" - [Notice] Change successfully!\n");
    printf(" - Please enter to return the main menu!\n");
    getchar();
}

/*
 * char *randCode()
 * - 随机生成 11 位的订单号
 */
void randCode(char *code)
{
    struct timeval tv;                              // sys/time.h 中自带的结构体，存储毫秒级的时间间隔
    gettimeofday(&tv, NULL);                        // 系统调用初始化 tv 结构体
    srand(tv.tv_sec * 1000 + tv.tv_usec / 1000); // 初始化随机种子，比srand((unsigned)time(NULL))更加精确
    for (int i = 0; i < 11; ++i)
        code[i] = rand() % 10 + '0'; // 生成精度不高的伪随机数
}
```

```
/*
 * void getTimeNow(char *)
 * - 获取当前时间
 */
void getTimeNow(char *Time)
{
    time_t current_time;
    struct tm *time_info;                  // sys/time.h 中自带的结构体，用于
存储当前所有单位的时间
    time(&current_time);                    // 获取 time_t 类型的时间戳
    time_info = localtime(&current_time); // 将时间戳转换为当前时间
    snprintf(Time, TIME_NOW_LENGTH, "%d-%02d-%02d", time_info->tm_year +
1900, time_info->tm_mon + 1,
             time_info->tm_mday); // 将结构体中的当前时间转换为年月日的字符
串
}

/*
 * void saveToFile(DataList *, char *)
 * - 保存数据并写入 csv 文件中
 */
void saveToFile(const DataList *list, const char *dest)
{
    FILE *file = fopen(dest, "w+"); // 用 w+读写模式进行文件写入
    if (file == NULL)               // 若文件打开失败则报错
    {
        printf(" - File open failed!\n");
        return;
    }

    UserNode *userNode = list->head;
    SaleNode *saleNode = NULL;
    while (userNode != NULL) // 遍历所有主链表和子链表以写入所有数据
    {
        saleNode = userNode->sale;
        while (saleNode != NULL)
        {
            // 用 fprintf 函数进行文件写入操作，写入到文件末尾时返回 EOF
            fprintf(file, "%s %s %s %s %s %s\n", userNode->data.userName,
userNode->data.userNumber,
```

```
                    saleNode->data.model, saleNode->data.system,
saleNode->data.time, saleNode->data.orderCode);
            saleNode = saleNode->bro; // write data to a file
        }
        userNode = userNode->next;
    }

    if (feof(file)) // 判断是否写入到文件末尾，若是则关闭文件
        fclose(file);
}

/*
 * void freeList(DataList *)
 * - 手动释放所有链表节点的内存
 */
void freeList(DataList *list) {
    UserNode *tempUser = list->head;
    if (tempUser == NULL) // 空链表则直接释放链表节点内存
    {
        free(list);
        return;
    }

    while (tempUser != NULL) // 遍历所有主链表的 UserNode 节点并释放内存
    {
        SaleNode *tempSale = tempUser->sale;
        while (tempSale != NULL) // 遍历所有子链表的 SaleNode 节点并释放内存
        {
            SaleNode *tempNextSale = tempSale->bro; // 先获取下一级节点地址
            free(tempSale);
            tempSale = tempNextSale;
        }

        UserNode *tempNextUser = tempUser->next; // 先获取下一级节点地址
        free(tempUser);
        tempUser = tempNextUser;
    }
    free(list); // 最后释放链表节点内存
}
```