

华中科技大学

# 课程实验报告

课程名称: 数据结构实验

专业班级: 物联网工程 201601

学 号: U201614897

姓 名: 潘越

指导教师: 李国辉

报告日期: 2017 年 1 月 4 日

# 目 录

<b>1 基于顺序存储结构的线性表实现.....</b>	<b>5</b>
1. 1 问题描述.....	5
1.1.1 线性表抽象数据类型.....	5
1.1.2 多线性表的管理.....	7
1.1.3 演示系统和文件存储的设计.....	7
1. 2 系统设计.....	7
1.2.1 数据物理结构.....	7
1.2.2 演示系统.....	8
1.2.3 线性表运算实现算法.....	9
1.2.4 多线性表管理实现算法.....	12
1.2.5 文件存储实现算法.....	12
1. 3 系统实现.....	13
1.3.1 实验环境.....	13
1.3.2 代码亮点.....	13
1.3.3 操作演示.....	13
1.3.4 系统健壮性/错误提示演示.....	24
1. 4 实验小结.....	30
<b>2 基于链式存储结构的线性表实现.....</b>	<b>32</b>
2. 1 问题描述.....	32
2.1.1 线性表抽象数据类型.....	32
2.1.2 多线性表的管理.....	34
2.1.3 演示系统和文件存储的设计.....	34
2. 2 系统设计.....	34
2.2.1 数据物理结构.....	34
2.2.2 演示系统.....	35
2.2.3 线性表运算实现算法.....	36
2.2.4 多线性表管理实现算法.....	39
2.2.5 文件存储实现算法.....	39

2.3 系统实现.....	40
2.3.1 实验环境.....	40
2.3.2 代码亮点.....	40
2.3.3 操作演示.....	40
2.3.4 系统健壮性/错误提示演示.....	49
2.4 实验小结.....	55
<b>3 基于二叉链表的二叉树实现.....</b>	<b>58</b>
3.1 问题描述.....	58
3.1.1 二叉树抽象数据类型.....	58
3.1.2 多二叉树的管理.....	61
3.1.3 演示系统和文件存储的设计.....	61
3.2 系统设计.....	61
3.2.1 数据物理结构.....	61
3.2.2 演示系统.....	63
3.2.3 二叉树运算实现算法.....	64
3.2.4 多二叉树管理实现算法.....	69
3.2.5 文件存储实现算法.....	69
3.3 系统实现.....	70
3.3.1 实验环境.....	70
3.3.2 代码亮点.....	70
3.3.3 操作演示.....	70
3.3.4 系统健壮性/错误提示演示.....	85
3.4 实验小结.....	93
<b>4 基于邻接表的图实现.....</b>	<b>95</b>
4.1 问题描述.....	95
4.1.1 图抽象数据类型.....	95
4.1.2 多图的管理.....	97
4.1.3 演示系统和文件存储的设计.....	97
4.2 系统设计.....	97
4.2.1 数据物理结构.....	97

4.2.2 演示系统.....	99
4.2.3 四种不同的图的区别.....	100
4.2.4 图运算实现算法.....	101
4.2.5 多图管理实现算法.....	105
4.2.6 文件存储实现算法.....	105
4.3 系统实现.....	106
4.3.1 实验环境.....	106
4.3.2 代码亮点.....	106
4.3.3 操作演示.....	106
4.3.4 系统健壮性/错误提示演示.....	116
4.4 实验小结.....	122
<b>参考文献.....</b>	<b>124</b>
<b>附录 A 实验中用到的 mylibqueue 库说明和源代码.....</b>	<b>126</b>
A.1 库接口函数说明.....	126
A.2 源代码.....	126
<b>附录 B 基于顺序存储结构线性表实现的源代码.....</b>	<b>129</b>
B.1 CMakeLists 编译文件.....	129
B.2 my_linearlist.h.....	129
B.3 my_linearlist.c.....	133
B.4 main.h.....	138
B.5 main.c.....	139
<b>附录 C 基于链式存储结构线性表实现的源代码.....</b>	<b>154</b>
C.1 CMakeLists.....	154
C.2 my_linklist.h.....	154
C.3 my_linklist.c.....	158
C.4 main.h.....	164
C.5 main.c.....	165
<b>附录 D 基于二叉链表二叉树实现的源代码.....</b>	<b>180</b>
D.1 CMakeLists.....	180
D.2 my_binarytree.h.....	180

D. 3 my_binarytree.c.....	186
D. 4 main.h.....	199
D. 5 main.c.....	200
<b>附录 E 基于邻接表图实现的源程序.....</b>	<b>224</b>
E. 1 CMakeLists.....	224
E. 2 my_graph.h.....	224
E. 3 my_graph.c.....	229
E. 4 main.h.....	241
E. 5 main.c.....	242

# 1 基于顺序存储结构的线性表实现

## 1.1 问题描述

实现基于顺序存储结构的线性表，并实现线性表的基本运算。

要求构造一个具有菜单的功能演示系统。其中，在主程序中完成函数调用所需实参值的准备和函数执行结果的显示，并给出适当的操作提示显示。演示系统可选择实现线性表的文件形式保存。

演示系统可以实现多线性表管理。

整个系统的设计模式如下：

### 1.1.1 线性表抽象数据类型

以下抽象数据类型源自参考文献[1]P19。

依据最小完备性和常用性相结合的原则，设计了线性表的数据对象和数据关系，并定义了线性表的初始化表、销毁表、清空表、判定空表、求表长和获得元素等 12 种基本运算，具体数据和运算功能定义如下。

ADT SqList {

    数据对象：  $D = \{a_i | a_i \in ElemSet, i = 1, 2, \dots, n, n \geq 0\}$

    数据关系：  $R_l = \{(a_{i-1}, a_i) | a_{i-1}, a_i \in D, i = 1, 2, \dots, n\}$

    基本操作：

        IntialList(&L)

            初始条件：线性表 L 不存在。

            操作结果：构造一个空的线性表。

        DestroyList(&L)

            初始条件：线性表 L 已存在。

            操作结果：销毁线性表 L。

        ClearList(&L)

            初始条件：线性表 L 已存在。

            操作结果：将 L 重置为空表。

        ListEmpty(L)

初始条件：线性表 L 已存在。

操作结果：若 L 为空表则返回 TRUE，否则返回 FALSE。

ListLength(L)

初始条件：线性表 L 已存在。

操作结果：返回 L 中数据元素的个数。

GetElem(L, idx, &ele)

初始条件：线性表 L 已存在且非空， $1 \leq \text{idx} \leq \text{ListLength}(L)$ 。

操作结果：用 ele 返回 L 中第 idx 个数据元素的值。

LocateElem(L, val)

初始条件：线性表 L 已存在且非空。

操作结果：返回 L 中第一个值为 val 的数据元素的位序。若这样的元素不存在则返回 0。

PriorElem(L, cur, &pre)

初始条件：线性表 L 已存在、非空且 cur 不是第一个数据元素。

操作结果：若 cur 是 L 的数据元素，则将它的前驱存入 pre，否则返回 FALSE。

NextElem(L, cur, &next)

初始条件：线性表 L 已存在、非空且 cur 不是最后一个数据元素。

操作结果：若 cur 是 L 的数据元素，则将它的后继存入 next，否则返回 FALSE。

ListInsert(&L, key, val)

初始条件：线性表 L 已存在且非空， $1 \leq \text{key} \leq \text{ListLength}(L)$ 。

操作结果：在 L 的第 key 个位置之前插入新的数据元素 val。

ListDelete(&L, key, &val)

初始条件：线性表 L 已存在且非空， $1 \leq \text{key} \leq \text{ListLength}(L)$ 。

操作结果：删除 L 的第 key 个数据元素，其值由 val 带回。

ListTraverse(L)

初始条件：线性表 L 已存在。

操作结果：依次访问并输出 L 的每个数据元素。

} ADT SqList

### 1.1.2 多线性表的管理

我在整个链表外又设计了一个包装链表，将已存在的所有线性表用链式结构串起，并对每个线性表赋予独特的 ID 来标识，在每次操作中通过 ID 来选择要进行操作的链表。

### 1.1.3 演示系统和文件存储的设计

没有使用参考文件，自行设计演示框架，该框架将完成函数调用所需实参值的准备和函数执行结果的实现，并给出适当的操作提示显示，整体以命令行呈现。

此外设计的数据文件实时存储，本地数据文件为 linearlistdb，存储方式为将所有的线性表结构体和数据区域依次直接将内存区块写入文件中，存为二进制文件。读取时通过这些数据来还原整个系统中的所有线性表。

## 1.2 系统设计

### 1.2.1 数据物理结构

#### 1. 线性表的存储数据结构

线性表结构体定义如下：

```
// the main struct of linear list
typedef struct linear_list {
    int id; // 线性表 ID
    int *data; // 指向数据存储区的指针
    int length; // 线性表长度
    int size; // 线性表大小
    struct linear_list *next; // 指向下一个线性表，多表管理用
} linear_list;
```

#### 2. 多线性表的存储数据结构

附加一个头结点，以链式结构存储所有的线性表

```
// the struct for managing the linear list
typedef struct Linear_list_main {
    int num;
    linear_list *head;
} Linear_list_main;
```

在本程序中，数据原子类型被定义为整形 int。

### 1.2.2 演示系统

演示系统包括用户操作界面和功能调用部分。

演示系统界面语言为英文，所有操作和提示语言均为英文。

用户操作界面输出可选的线性表操作，用户输入数字选择要进行的操作。在用户选择操作后，功能调用部分会显示函数的名称，参数，返回值和作用，系统提示用户输入参数。

功能调用部分将用户输入的有关信息传递给线性数据结构的操作函数进行调用，并对函数的返回值进行处理判断输出相应的提示信息。

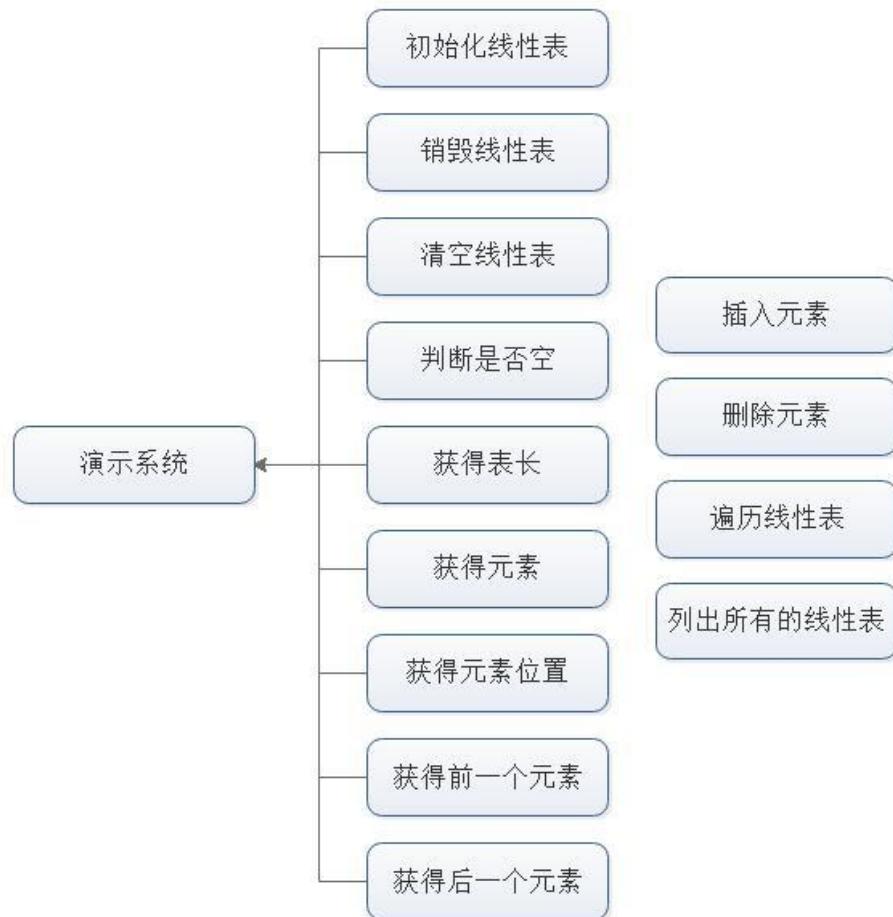


图 1-1 演示系统模块结构图

### 1.2.3 线性表运算实现算法

本部分说明算法时直接用函数原型代替。

1. `int init_list(linear_list *L);`

**功能:** 初始化线性表。

**算法实现:** 为线性表 L 的 data 区域申请空间，如果申请失败则返回 ERROR，否则返回成功。

**时空效率分析:** 算法的时间复杂度为  $O(1)$ ，空间上为 L 的 data 区域申请空间，所以空间复杂度为  $O(1)$ 。

2. `int destroy_list(linear_list *L);`

**功能:** 销毁线性表。

**算法实现:** 如果 L 的 data 区域不为空，则 free 掉 data 区域，然后将整个

L 占有的内存空间用 0 填满。

**时空效率分析：** 算法的时间复杂度为  $O(1)$ ，空间复杂度为  $O(1)$ 。

3. `int clear_list(linear_list *L);`

**功能：** 清空线性表。

**算法实现：** 如果的 data 区域不为空，则将 data 区域占有的内存空间用 0 填满

**时空效率分析：** 算法的时间复杂度为  $O(1)$ ，空间复杂度为  $O(1)$ 。

4. `int is_list_empty(linear_list L);`

**功能：** 判断线性表是否为空。

**算法实现：** 如果 `L->length` 为 0，则返回 TURE，否则返回 FALSE。

**时空效率分析：** 算法的时间复杂度为  $O(1)$ ，空间复杂度为  $O(1)$ 。

5. `int list_length(linear_list L);`

**功能：** 求线性表的长度。

**算法实现：** 直接返回线性表结构体中存储的 `L->length` 即可。

**时空效率分析：** 算法的时间复杂度为  $O(1)$ ，空间复杂度为  $O(1)$ 。

6. `int get_list_item(linear_list L, int order, int *elem);`

**功能：** 获得线性表中指定位置的数据。

**算法实现：** 首先判断指定位置是否在 1 与 `L->length` 之间，若不是则返回 ERROR，否则把 `L->data[order-1]` 的地址赋值给 elem，返回 OK。

**时空效率分析：** 算法的时间复杂度为  $O(1)$ ，空间复杂度为  $O(1)$ 。

7. `int locate_list_item(linear_list L, int ordered_elem);`

**功能：** 寻找指定元素在线性表中的位置。

**算法实现：** 遍历线性表的数据区域，如果发现就返回 `index + 1`，否则返回 0。

**时空效率分析：**由于要找的数据可能存在于 data 区域的任何一个位置，在等概率的情形下，求得算法的时间复杂度为  $O(n)$ ，空间上用 index 来存储要返回的位置，所以空间复杂度为  $O(1)$ 。

8. `int prior_list_item(linear_list L, int elem, int *elem_pre);`

**功能：**获得指定元素之前的一个元素。

**算法实现：**类似寻找元素的位置，我们直接遍历查找，然后把前一个位置的元素的地址赋值给 elem\_pre。如果找到就返回 OK，否则返回 ERROR。

**时空效率分析：**同上面一个函数，算法的时间复杂度为  $O(n)$ ，空间复杂度为  $O(1)$ 。

9. `int next_list_item(linear_list L, int elem, int *elem_next);`

**功能：**获得指定元素之后的一个元素。

**算法实现：**类似寻找元素的位置，我们直接遍历查找，然后把后一个位置的元素的地址赋值给 elem\_pre。如果找到就返回 OK，否则返回 ERROR。

**时空效率分析：**同上面一个函数，算法的时间复杂度为  $O(n)$ ，空间复杂度为  $O(1)$ 。

10. `int list_insert(linear_list *L, int order, int elem);`

**功能：**插入元素。

**算法实现：**首先判断要插入的位置是否在 1 与 `L->length` 之间，若不是则返回 ERROR。然后判断 `L->size` 是否和 `L->length` 相同，如果相同，说明线性表空间已满，就再给线性表分配固定的 `LIST_INCREASEMENT` 的空间。

然后将指定位置之后的元素全部向后移动一个位置，将新的元素插入在指定位置，返回 OK。

**时空效率分析：**算法中需要移动元素，在等概率的情况下计算得算法的时间复杂度为  $O(n)$ ，空间上还是遍历空间，所以空间复杂度为  $O(1)$ 。

11. `int list_delete(linear_list *L, int order, int *elem);`

**功能:** 删除元素。

**算法实现:** 首先判断要删除的位置是否在 1 与  $L \rightarrow length$  之间，若不是则返回 ERROR。

然后将指定位置之后的元素全部向前移动一个位置，返回 OK。

**时空效率分析:** 算法中需要移动元素，在等概率的情况下计算得算法的时间复杂度为  $O(n)$ ，空间上还是遍历空间，所以空间复杂度为  $O(1)$ 。

12. void print\_list(linear\_list L);

**功能:** 遍历并打印线性表。

**算法实现:** 直接遍历并打印线性表中的所有元素即可。

**时空效率分析:** 算法的时间复杂度为  $O(n)$ ，空间复杂度为  $O(1)$ 。

#### 1.2.4 多线性表管理实现算法

由于多线性表的管理采用链式线性表模型，只涉及到查找操作，所以时间复杂度是遍历用的  $O(n)$ ，空间复杂度为  $O(1)$ 。

#### 1.2.5 文件存储实现算法

1. void load\_data(Linear\_list\_main \*Main\_L);

**功能:** 数据读取。

**算法实现:** 打开文件，依次读取一个线性表结构体，然后根据线性表的  $L \rightarrow size$  读取数据空间。之后插入总链式线性表中。直到所有的线性表被读取入系统。

**时空效率分析:** 算法的时间复杂度为  $O(n)$ ，为遍历  $n$  个线性表；空间复杂度为  $O(n)$ ，为申请  $n$  个线性表的空间。

2. void save\_data(Linear\_list\_main \*Main\_L);

**功能:** 数据保存。

**算法实现:** 打开文件，依次存入一个线性表结构体，然后根据线性表的

`L->size` 存入数据空间。直到所有的线性表被存入系统。

**时空效率分析：** 算法的时间复杂度为  $O(n)$ ，为遍历  $n$  个线性表；空间复杂度为  $O(n)$ ，为存入  $n$  个线性表的空间。

## 1.3 系统实现

### 1.3.1 实验环境

实验环境为 Arch linux 4.14.8-1，编译器为 gcc 版本 7.2.1，代码采用开源的编辑器 vscode 编写。由指定的 Makefile 来完成编译。

文件说明：

- \* my\_linearlist.h: 线性表库头文件
- \* my\_linearlist.c: 线性表库实现
- \* main.h: 演示系统头文件
- \* main.c: 演示系统实现
- \* CMakeLists.txt: CMake 列表（用以生成完备的 Makefile）
- \* Makefile: 自动化编译命令

### 1.3.2 代码亮点

所有的代码采用 Google C/C++ 标准代码规范，函数库所有注释采用英文，符合生产规范。错误检查和提示全面，根据不同的错误会有不同的提示信息。由于在 Linux 环境下编程，所有的 API 接口命名改用 unix 环境编程的标准短横线命名模式。

### 1.3.3 操作演示

界面展示：

```
./lab01
文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)
panyue@Saltedfish ~ > cd code/my_github/HUST-DataStructure-Labs
panyue@Saltedfish ~/code/my_github/HUST-DataStructure-Labs > master > cd lab01
panyue@Saltedfish ~/code/my_github/HUST-DataStructure-Labs/lab01 > master > ./lab01
+-----+
| Welcome to panyue's linear list demo system!
| Here are some functions you can call:
|
| 1: init_list          2: destroy_list
| 3: clear_list         4: is_list_empty
| 5: list_length         6: get_list_item
| 7: locate_list_item   8: prior_list_item
| 9: next_list_item     10: list_insert
| 11: list_delete        12: print_list
| 13: ls_list            0: quit
|
| Enter the number of the function and see the usage and call it!
| Enter 0 to quit the demo system.
|
| Copyright (C) 2017 Yue Pan
| Github: zxc479773533
+-----+
Your choose: [
```

图 1-2 演示系统界面

接下来我们以如下的结构开始，来演示本系统的各项操作：

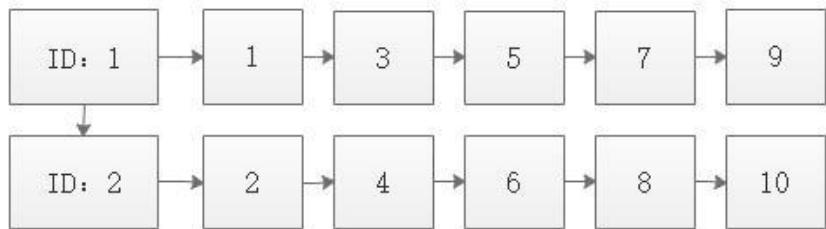


图 1-3 初始的线性表结构

系统中已经存入了两个线性表，结点和 ID 分别如图所示。

### 1. 首先列出存在的线性表

```
+-----+  
| Welcome to panyue's linear list demo system!  
| Here are some functions you can call:  
|  
| 1: init_list          2: destroy_list  
| 3: clear_list         4: is_list_empty  
| 5: list_length        6: get_list_item  
| 7: locate_list_item   8: prior_list_item  
| 9: next_list_item    10: list_insert  
| 11: list_delete       12: print_list  
| 13: ls_list           0: quit  
|  
| Enter the number of the function and see the usage and call it!  
| Enter 0 to quit the demo system.  
|  
| Copyright (C) 2017 Yue Pan  
| Github: zxc479773533  
+-----+  
  
Your choose: 13  
ID: 2  
ID: 1
```

图 1-4 打印所有的线性表

## 2. 获得线性表 1 第 3 位的元素

```
Your choose: 6  
/*  
 * Function Name: get_list_item  
 * Module: Data structures  
 * Parameter: linear_list L, int order, int *elem  
 * Return: int(status)  
 * Use: get the ordered element of link list  
 */  
  
Format: id index  
Then, enter the list id and index: 1 3  
The element is 5.
```

图 1-5 获得线性表 1 第 3 位的元素

## 3. 获得元素 8 的位置

```
Your choose: 7
/*
 * Function Name: locate_list_item
 * Module: Data structures
 * Parameter: linear_list L, int ordered_elem
 * Return: int(index)
 * Use: get the index of ordered item
 */

Format: id element
Then, enter the list id and element: 2 8
The index of the 8, is 4.
```

图 1-6 获得元素 8 的位置

#### 4. 获得元素 7 之前的元素

```
Your choose: 8
/*
 * Function Name: piror_list_item
 * Module: Data structures
 * Parameter: linear_list L, int elem, int *elem_pre
 * Return: int(status)
 * Use: get the ordered element's piror
 */

Format: id element
Then, enter the list id and element: 1 7
The piror element of 7 is 5.
```

图 1-7 获得元素 7 之前的元素

#### 5. 获得元素 2 的后一个元素

```
Your choose: 9
/*
 * Function Name: next_list_item
 * Module: Data structures
 * Parameter: linear_list L, int elem, int *elem_next
 * Return: int(status)
 * Use: get the ordered element's next
 */

Format: id element
Then, enter the list id and element: 2 4
The next element of 4 is 8.
```

图 1-8 获得元素 2 的后一个元素

#### 6. 获得表 1 的表长

```
Your choose: 5
/*
 * Function Name: list_length
 * Module: Data structures
 * Parameter: linear_list L
 * Return: int(the length)
 * Use: get the length of linear list
 */

Then, enter the list id: 1
The length of linear list 1 is 5.
```

图 1-9 获得表 1 的表长

#### 7. 打印表 1

```
Your choose: 12
/*
 * Function Name: print_list
 * Module: Data structures
 * Parameter: linear_list L, char *payload
 * Return: None
 * Use: print the elements of the linklist to the payload
 */

Then, enter the list id: 1
The elements in linear list 1 are:
Index: 1, Data: 1
Index: 2, Data: 3
Index: 3, Data: 5
Index: 4, Data: 7
Index: 5, Data: 9
```

图 1-10 打印表 1

#### 8. 在表 1 第 2 个位置处插入 99

```
Your choose: 10
/*
 * Function Name: list_insert
 * Module: Data structures
 * Parameter: linear_list *L, int order, int elem
 * Return: int(status)
 * Use: insert a element in the link list
 */

Format: id index element
Then, enter the list id, index and element: 1 2 99
Insert 99 succeed!
```

图 1-11 在表 1 第 2 个位置处插入 99

此时的线性表:

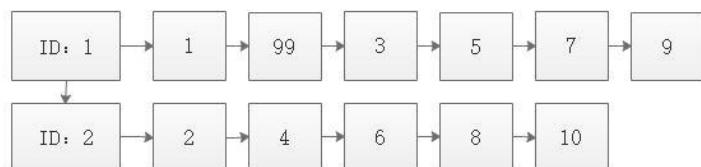


图 1-12 此时的线性表

9. 打印表 1 的表长，此时应该是 6

```
Your choose: 5
/*
 * Function Name: list_length
 * Module: Data structures
 * Parameter: linear_list L
 * Return: int(the length)
 * Use: get the length of linear list
 */

Then, enter the list id: 1
The length of linear list 1 is 6.
```

图 1-13 打印表 1 的表长

10. 打印插入 99 之后的表 1

```
Your choose: 12
/*
 * Function Name: print_list
 * Module: Data structures
 * Parameter: linear_list L, char *payload
 * Return: None
 * Use: print the elements of the linklist to the payload
 */

Then, enter the list id: 1
The elements in linear list 1 are:
Index: 1, Data: 1
Index: 2, Data: 99
Index: 3, Data: 3
Index: 4, Data: 5
Index: 5, Data: 7
Index: 6, Data: 9
```

图 1-14 打印插入 99 之后的表 1

11. 删除表 1 中第 5 位的元素

```
Your choose: 11
/*
 * Function Name: list_delete
 * Module: Data structures
 * Parameter: linear_list *L, int order, int *elem
 * Return: int(status)
 * Use: delete a element in the link list
 */

Format: id index
Then, enter the list id and index: 1 5
Delete 7 succeed!
```

图 1-15 删除表 1 中第五位的元素

12. 获得表 1 表长，此时可以发现是 5

```
Your choose: 5
/*
 * Function Name: list_length
 * Module: Data structures
 * Parameter: linear_list L
 * Return: int(the length)
 * Use: get the length of linear list
 */

Then, enter the list id: 1
The length of linear list 1 is 5.
```

图 1-16 获得表 1 表长

13. 删除表 1 中第 1 位的元素

```
Your choose: 11
/*
 * Function Name: list_delete
 * Module: Data structures
 * Parameter: linear_list *L, int order, int *elem
 * Return: int(status)
 * Use: delete a element in the link list
 */

Format: id index
Then, enter the list id and index: 1 1
Delete 1 succeed!
```

图 1-17 删除表 1 中第一位的元素

此时的线性表：

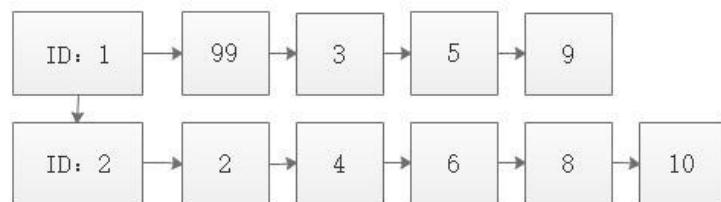


图 1-18 此时的线性表

14. 查看表 1 表长，此时可以发现是 4

```
Your choose: 5
/*
 * Function Name: list_length
 * Module: Data structures
 * Parameter: linear_list L
 * Return: int(the length)
 * Use: get the length of linear list
 */

Then, enter the list id: 1
The length of linear list 1 is 4.
```

图 1-19 查看表 1 表长

15. 打印表 1

```
Then, enter the list id: 1
The elements in linear list 1 are:
Index: 1, Data: 99
Index: 2, Data: 3
Index: 3, Data: 5
Index: 4, Data: 9
```

图 1-20 打印表 1

16. 清空之前，判断表 2 是否为空

```
Your choose: 4
/*
 * Function Name: is_list_empty
 * Module: Data structures
 * Parameter: linear_list L
 * Return: int(true or false)
 * Use: judge if the link list is empty
 */

Then, enter the list id: 2
Linear list 2 is not empty!
```

图 1-21 判断表 2 是否为空

17. 清空表 2

```
Your choose: 3
/*
 * Function Name: clear_list
 * Module: Data structures
 * Parameter: linear_list *L
 * Return: int(status)
 * Use: let the link list empty
 */

Then, enter the list id: 2
Linear list 2 has been cleared!
```

图 1-22 清空表 2

18. 清空后，判断表 2 是否为空

```
Your choose: 4
/*
 * Function Name: is_list_empty
 * Module: Data structures
 * Parameter: linear_list L
 * Return: int(true or false)
 * Use: judge if the link list is empty
 */

Then, enter the list id: 2
Linear list 2 is empty!
```

图 1-23 判断表 2 是否为空

19. 删除表 1 前，列出所有的线性表

```
Your choose: 13
ID: 2
ID: 1
```

图 1-24 列出所有的线性表

20. 删除表 1

```
Your choose: 2
/*
 * Function Name: destroy_list
 * Module: Data structures
 * Parameter: linear_list *L
 * Return: int(status)
 * Use: destroy the link list
 */

Then, enter the list id: 1
Linear list 1 has been deleted!
```

图 1-25 删除表 1

21. 删除表 1 后，列出所有的线性表

```
Your choose: 13  
ID: 2
```

图 1-26 列出所有的线性表

22. 正常退出系统

```
+-----+  
| Welcome to panyue's linear list demo system!  
| Here are some functions you can call:  
|  
| 1: init_list          2: destroy_list  
| 3: clear_list         4: is_list_empty  
| 5: list_length        6: get_list_item  
| 7: locate_list_item   8: prior_list_item  
| 9: next_list_item    10: list_insert  
| 11: list_delete       12: print_list  
| 13: ls_list           0: quit  
|  
| Enter the number of the function and see the usage and call it!  
| Enter 0 to quit the demo system.  
|  
| Copyright (C) 2017 Yue Pan  
| Github: zxc479773533  
+-----+  
  
Your choose: 0  
Thanks for using my demo system!  
panyue@Saltedfish ~ /code/my github/HUST-DataStructure-Labs/lab01 master
```

图 1-27 正常退出系统

通过以上的演示证明，我们的线性表的功能是完整实现了的，可以演示要求的十二种线性表上的运算，并且做到了多线性表管理，实现了文件存储功能，不存在内存泄漏，完整的达到了实验的要求。

#### 1. 3. 4 系统健壮性/错误提示演示

我们仍然以如下的结构开始，来演示错误提示：

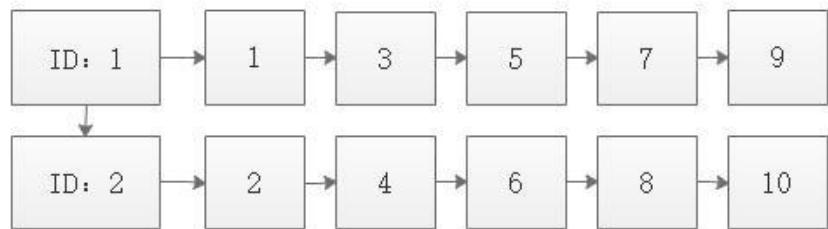


图 1-28 初始的线性表结构

系统中已经存入了两个线性表，结点和 ID 分别如图所示。

### 1. 初始化已存在的线性表

```

Your choose: 1
/*
 * Function Name: init_list
 * Module: Data structures
 * Parameter: linear_list *L
 * Return: int(status)
 * Use: initial the linear list
 */

Then, enter the list id: 1
Initial error, this linear list is already exists!

```

图 1-29 初始化已存在的线性表

### 2. 删除不存在的线性表

```

Your choose: 2
/*
 * Function Name: destroy_list
 * Module: Data structures
 * Parameter: linear_list *L
 * Return: int(status)
 * Use: destroy the link list
 */

Then, enter the list id: 3
Destroy error, this linear list is not exits!

```

图 1-30 删除不存在的线性表

3. 清空不存在的线性表

```
Your choose: 3
/*
 * Function Name: clear_list
 * Module: Data structures
 * Parameter: linear_list *L
 * Return: int(status)
 * Use: let the link list empty
 */

Then, enter the list id: 5
Clear error, this linear list is not exists!
```

图 1-31 清空不存在的线性表

4. 判断不存在的线性表是否为空

```
Your choose: 4
/*
 * Function Name: is_list_empty
 * Module: Data structures
 * Parameter: linear_list L
 * Return: int(true or false)
 * Use: judge if the link list is empty
 */

Then, enter the list id: 5
Error, this linear list is not exists!
```

图 1-32 判断不存在的线性表是否为空

5. 获得不存在的线性表的表长

```
Your choose: 5
/*
 * Function Name: list_length
 * Module: Data structures
 * Parameter: linear_list L
 * Return: int(the length)
 * Use: get the length of linear list
 */

Then, enter the list id: 5
Error, this linear list is not exists!
```

图 1-33 获得不存在的线性表的表长

#### 6. 获得表中不存在的元素

```
Your choose: 6
/*
 * Function Name: get_list_item
 * Module: Data structures
 * Parameter: linear_list L, int order, int *elem
 * Return: int(status)
 * Use: get the ordered element of link list
 */

Format: id index
Then, enter the list id and index: 1 7
Get element error! Please check your input.
```

图 1-34 获得表中不存在的元素

#### 7. 确定表中不存在的元素的位置

```
Your choose: 7
/*
 * Function Name: locate_list_item
 * Module: Data structures
 * Parameter: linear_list L, int ordered_elem
 * Return: int(index)
 * Use: get the index of ordered item
 */

Format: id element
Then, enter the list id and element: 1 10
The index of the 10, is 0.
```

图 1-35 确定表中不存在的元素的位置

#### 8. 获得表中不存在的元素的前元素

```
Your choose: 8
/*
 * Function Name: piror_list_item
 * Module: Data structures
 * Parameter: linear_list L, int elem, int *elem_pre
 * Return: int(status)
 * Use: get the ordered element's piror
 */

Format: id element
Then, enter the list id and element: 1 2
Error, the piror of 2 is not exists!
```

图 1-36 获得表中不存在的元素的前元素

#### 9. 获得表中不存在的元素的后元素

```
Your choose: 9
/*
 * Function Name: next_list_item
 * Module: Data structures
 * Parameter: linear_list L, int elem, int *elem_next
 * Return: int(status)
 * Use: get the ordered element's next
 */

Format: id element
Then, enter the list id and element: 1 9
Error, the next of 9 is not exists!
```

图 1-37 获得表中不存在的元素的后元素

#### 10 插入元素超过线性表的最大大小

```
Your choose: 10
/*
 * Function Name: list_insert
 * Module: Data structures
 * Parameter: linear_list *L, int order, int elem
 * Return: int(status)
 * Use: insert a element in the link list
 */

Format: id index element
Then, enter the list id, index and element: 1 101 1
Insert failed, please check your input!
```

图 1-38 插入元素超过线性表的最大大小

#### 11. 删除表中不存在的元素

```
Your choose: 11
/*
 * Function Name: list_delete
 * Module: Data structures
 * Parameter: linear_list *L, int order, int *elem
 * Return: int(status)
 * Use: delete a element in the link list
 */

Format: id index
Then, enter the list id and index: 1 99
Delete failed, please check your input!
```

图 1-39 删除表中不存在的元素

## 12. 打印不存在的线性表

```
Your choose: 12
/*
 * Function Name: print_list
 * Module: Data structures
 * Parameter: linear_list L, char *payload
 * Return: None
 * Use: print the elements of the linklist to the payload
 */

Then, enter the list id: 5
Error, this linear list is not exists!
```

图 1-40 打印不存在的线性表

通过以上的演示证明，我们的线性表演示系统的健壮性是足够的，可以对各种错误输入进行处理，输出相应的错误提示。

## 1.4 实验小结

这次实验作为数据结构实验课程的第一次实验，内容还是非常简单的，通过整个编程过程，使我又熟悉了一边顺序结构的线性表，也算是一个很好的复习了。毕竟，在日常开发中很少用到这种结构，我一般都用的是链式结构，主要还是因为插入删除操作比较多吧。

整个实验在 linux 环境下编程，所有的代码采用 Google C/C++ 标准代码规范，函数库所有注释采用英文，符合生产规范。错误检查和提示全面，根据不同的错误会有不同的提示信息。由于在 Linux 环境下编程，所有的 API 接口命名改用 unix 环境编程的标准短横线命名模式。这样规范自己有助于以后走向工作岗位之前的面试环节。

本系统完整的实现了实现的课程要求的全部功能，实现了多线性表管理和文件存储功能，系统健壮性良好，可以输出各种情况下的错误提示，并且在整个实验过程中避免了处处的内存泄漏，完整的达到了预期的效果。

## 2 基于链式存储结构的线性表实现

### 2.1 问题描述

实现基于链式存储结构的线性表，并实现线性表的基本运算。

要求构造一个具有菜单的功能演示系统。其中，在主程序中完成函数调用所需实参值的准备和函数执行结果的显示，并给出适当的操作提示显示。演示系统可选择实现线性表的文件形式保存。

演示系统可以实现多线性表管理。

整个系统的设计模式如下：

#### 2.1.1 线性表抽象数据类型

以下抽象数据类型源自参考文献[1]P19。

依据最小完备性和常用性相结合的原则，设计了线性表的数据对象和数据关系，并定义了线性表的初始化表、销毁表、清空表、判定空表、求表长和获得元素等 12 种基本运算，具体数据和运算功能定义如下。

ADT SqList {

    数据对象：  $D = \{a_i | a_i \in ElemSet, i = 1, 2, \dots, n, n \geq 0\}$

    数据关系：  $R_l = \{(a_{i-1}, a_i) | a_{i-1}, a_i \in D, i = 1, 2, \dots, n\}$

    基本操作：

        IntialList(&L)

            初始条件：线性表 L 不存在。

            操作结果：构造一个空的线性表。

        DestroyList(&L)

            初始条件：线性表 L 已存在。

            操作结果：销毁线性表 L。

        ClearList(&L)

            初始条件：线性表 L 已存在。

            操作结果：将 L 重置为空表。

        ListEmpty(L)

初始条件：线性表 L 已存在。

操作结果：若 L 为空表则返回 TRUE，否则返回 FALSE。

ListLength(L)

初始条件：线性表 L 已存在。

操作结果：返回 L 中数据元素的个数。

GetElem(L, idx, &ele)

初始条件：线性表 L 已存在且非空， $1 \leq \text{idx} \leq \text{ListLength}(L)$ 。

操作结果：用 ele 返回 L 中第 idx 个数据元素的值。

LocateElem(L, val)

初始条件：线性表 L 已存在且非空。

操作结果：返回 L 中第一个值为 val 的数据元素的位序。若这样的元素不存在则返回 0。

PriorElem(L, cur, &pre)

初始条件：线性表 L 已存在、非空且 cur 不是第一个数据元素。

操作结果：若 cur 是 L 的数据元素，则将它的前驱存入 pre，否则返回 FALSE。

NextElem(L, cur, &next)

初始条件：线性表 L 已存在、非空且 cur 不是最后一个数据元素。

操作结果：若 cur 是 L 的数据元素，则将它的后继存入 next，否则返回 FALSE。

ListInsert(&L, key, val)

初始条件：线性表 L 已存在且非空， $1 \leq \text{key} \leq \text{ListLength}(L)$ 。

操作结果：在 L 的第 key 个位置之前插入新的数据元素 val。

ListDelete(&L, key, &val)

初始条件：线性表 L 已存在且非空， $1 \leq \text{key} \leq \text{ListLength}(L)$ 。

操作结果：删除 L 的第 key 个数据元素，其值由 val 带回。

ListTraverse(L)

初始条件：线性表 L 已存在。

操作结果：依次访问并输出 L 的每个数据元素。

} ADT SqList

## 2.1.2 多线性表的管理

同实验一一样，我在整个链表外又设计了一个包装链表，将已存在的所有线性表用链式结构串起，并对每个线性表赋予独特的 ID 来标识，在每次操作中通过 ID 来选择要进行操作的链表。

## 2.1.3 演示系统和文件存储的设计

没有使用参考文件，自行设计演示框架，该框架将完成函数调用所需实参值的准备和函数执行结果的实现，并给出适当的操作提示显示，整体以命令行呈现。

此外设计的数据文件实时存储，本地数据文件为 linklistdb，存储方式为将所有的线性表结构体和数据区域依次直接将内存区块写入文件中，存为二进制文件。读取时通过这些数据来还原整个系统中的所有线性表。

## 2.2 系统设计

### 2.2.1 数据物理结构

#### 1. 线性表的存储数据结构

线性表结构体定义如下：

```
// the main struct of link list
typedef struct link_list {
    int id; //线性表 ID
    int length; //线性表长度
    struct link_node *head; // 指向线性表头结点的指针
    struct link_list *next; // 指向下一个线性表，多表管理用
} link_list;
```

#### 2. 链表结点的存储数据结构

```
// the struct of link node  
  
typedef struct link_node {  
    int data; // 结点数据区域  
    struct link_node *next; // 指向下一个结点的指针  
} link_node;
```

### 3. 多线性表的存储数据结构

附加一个头结点，以链式结构存储所有的线性表

```
// the struct for managing the link list  
  
typedef struct Link_list_main {  
    int num;  
    link_list *head;  
} Link_list_main;
```

在本程序中，数据原子类型被定义为整形 int。

## 2. 2. 2 演示系统

演示系统包括用户操作界面和功能调用部分。

演示系统界面语言为英文，所有操作和提示语言均为英文。

用户操作界面输出可选的线性表操作，用户输入数字选择要进行的操作。在用户选择操作后，功能调用部分会显示函数的名称，参数，返回值和作用，系统提示用户输入参数。

功能调用部分将用户输入的有关信息传递给线性数据结构的操作函数进行调用，并对函数的返回值进行处理判断输出相应的提示信息。

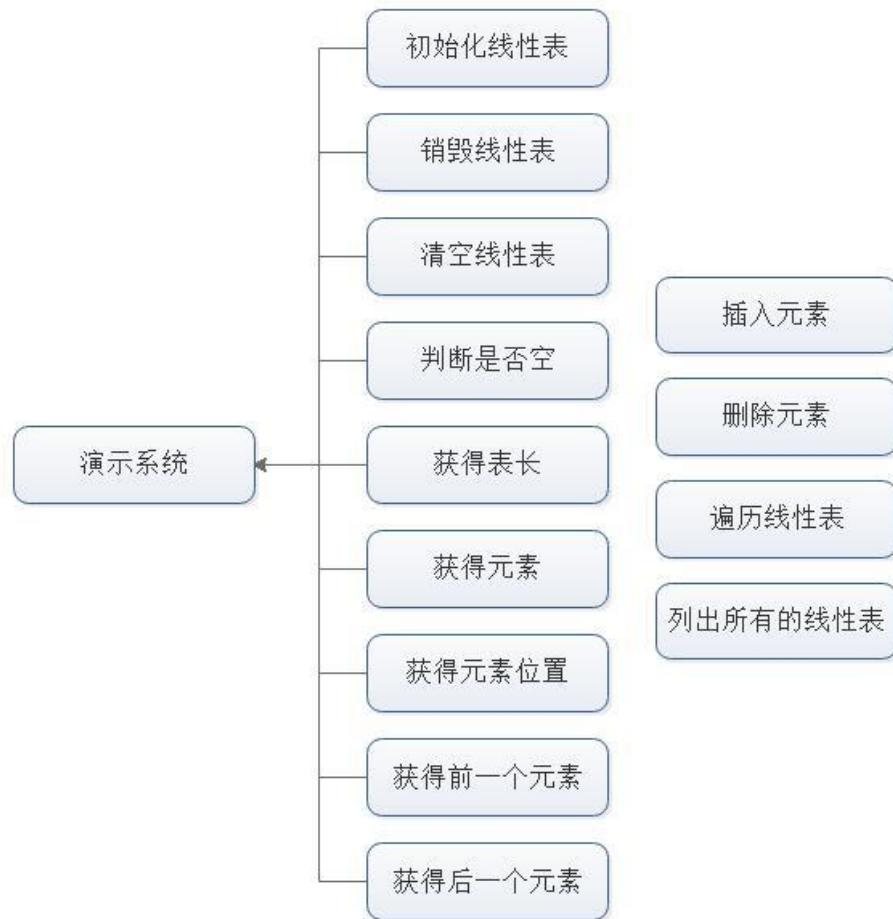


图 2-1 演示系统模块结构图

整个演示系统的设计基本和实验一一致，接口使用相同接口，只是库函数的实现改变了。

### 2.2.3 线性表运算实现算法

本部分说明算法时直接用函数原型代替。

1. `int init_list(link_list *L);`

**功能：** 初始化线性表。

**算法实现：** 将线性表的长度设置为 0，将其指向头结点的指针设置为 NULL，返回 OK。

**时空效率分析：** 算法的时间复杂度为  $O(1)$ ，空间复杂度为  $O(1)$ 。

2. `int destroy_list(link_list *L);`

**功能：** 销毁线性表。

**算法实现:** 遍历 free 掉所有的链表结点, 最后 free 掉链表结构体自身, 返回 OK。

**时空效率分析:** 算法的时间复杂度为  $O(n)$ , 空间复杂度为  $O(1)$ 。

3. `int clear_list(link_list *L);`

**功能:** 清空线性表。

**算法实现:** 如果 `L->head` 为空, 则返回 ERROR, 否则遍历并 free 掉所有的链表结点, 返回 OK。

**时空效率分析:** 算法的时间复杂度为  $O(n)$ , 空间复杂度为  $O(1)$ 。

4. `int is_list_empty(link_list L);`

**功能:** 判断线性表是否为空。

**算法实现:** 如果 `L->length` 为 0, 则返回 TURE, 否则返回 FALSE。

**时空效率分析:** 算法的时间复杂度为  $O(1)$ , 空间复杂度为  $O(1)$ 。

5. `int list_length(linear_list L);`

**功能:** 求线性表的长度。

**算法实现:** 直接返回线性表结构体中存储的 `L->length` 即可。

**时空效率分析:** 算法的时间复杂度为  $O(1)$ , 空间复杂度为  $O(1)$ 。

6. `int get_list_item(link_list L, int order, int *elem);`

**功能:** 获得线性表中指定位置的数据。

**算法实现:** 遍历链表, 如果找到指定位置的数据, 就将其地址赋值给 `elem`, 返回 OK, 否则返回 ERROR。

**时空效率分析:** 由于要找的数据可能存在于 data 区域的任何一个位置, 在等概率的情形下, 求得算法的时间复杂度为  $O(n)$ , 空间上用 index 来存储要返回的位置, 所以空间复杂度为  $O(1)$ 。

7. `int locate_list_item(link_list L, int ordered_elem);`

**功能:** 寻找指定元素在线性表中的位置。

**算法实现:** 遍历线性表的数据区域, 如果发现就返回 `index + 1`, 否则返回 0。

**时空效率分析:** 由于要找的数据可能存在于 `data` 区域的任何一个位置, 在等概率的情形下, 求得算法的时间复杂度为  $O(n)$ , 空间上用 `index` 来存储要返回的位置, 所以空间复杂度为  $O(1)$ 。

8. `int prior_list_item(link_list L, int elem, int *elem_pir);`

**功能:** 获得指定元素之前的一个元素。

**算法实现:** 遍历查找相应的结点, 并用一个 `pre` 指针指向前一个结点, 如果找到了结点, 就把前一个结点的地址赋值给 `elem_pre`。如果找到就返回 OK, 否则返回 ERROR。

**时空效率分析:** 同上面一个函数, 算法的时间复杂度为  $O(n)$ , 空间复杂度为  $O(1)$ 。

9. `int next_list_item(link_list L, int elem, int *elem_next);`

**功能:** 获得指定元素之后的一个元素。

**算法实现:** 遍历查找相应的结点, 如果找到了结点, 就把后一个结点的地址赋值给 `elem_pre`。如果找到就返回 OK, 否则返回 ERROR。

**时空效率分析:** 同上面一个函数, 算法的时间复杂度为  $O(n)$ , 空间复杂度为  $O(1)$ 。

10. `int list_insert(link_list *L, int order, int elem);`

**功能:** 插入元素。

**算法实现:** 遍历查找相应的结点, 并用一个 `pre` 指针指向前一个结点, 如果找到了结点, 就申请一个新结点的空间, 并把 `pre->next` 赋值为新结点的地址, 把新结点的下一个赋值为当前结点的地址。如果找到就返回 OK, 否则返回 ERROR。

**时空效率分析:** 同上面一个函数, 算法的时间复杂度为  $O(n)$ , 空间复杂度为  $O(1)$ 。

11. int list\_delete(link\_list \*L, int order, int \*elem);

**功能:** 删除元素。

**算法实现:** 遍历查找相应的结点，并用一个 pre 指针指向当前一个结点，如果找到了结点，就把 `pre->next` 赋值当前结点的下一个结点的地址，并 free 掉当前结点。如果找到就返回 OK，否则返回 ERROR。

**时空效率分析:** 同上面一个函数，算法的时间复杂度为  $O(n)$ ，空间复杂度为  $O(1)$ 。

12. void print\_list(link\_list L);

**功能:** 遍历并打印线性表。

**算法实现:** 直接遍历并打印线性表中的所有元素即可。

**时空效率分析:** 算法的时间复杂度为  $O(n)$ ，空间复杂度为  $O(1)$ 。

## 2.2.4 多线性表管理实现算法

由于多线性表的管理采用链式线性表模型，只涉及到查找操作，所以时间复杂度是遍历用的  $O(n)$ ，空间复杂度为  $O(1)$ 。

## 2.2.5 文件存储实现算法

1. void load\_data(Link\_list\_main \*Main\_L);

**功能:** 数据读取。

**算法实现:** 打开文件，依次读取一个线性表结构体，然后根据线性表的 `L->length` 读取相应个数的链表结点，插入线性表中。之后将新的线性表插入总链式线性表中。直到所有的线性表被读取入系统。

**时空效率分析:** 算法的时间复杂度为  $O(n * m)$ ，为遍历  $n$  个线性表，平均每个  $m$  个结点；空间复杂度为  $O(n * n)$ ，为申请  $n * m$  个链表结点的空间。

2. void save\_data(Link\_list\_main \*Main\_L);

**功能:** 数据保存。

**算法实现:** 打开文件，依次存入一个线性表结构体，然后根据线性表的

`L->length` 存入相应个数的链表结点。直到所有的线性表被存入系统。

**时空效率分析：** 算法的时间复杂度为  $O(n * m)$ ，为遍历  $n$  个线性表，平均每个  $m$  个结点；空间复杂度为  $O(n * n)$ ，为存入  $n * m$  个链表结点的空间。

## 2.3 系统实现

### 2.3.1 实验环境

实验环境为 Arch Linux 4.14.8-1，编译器为 gcc 版本 7.2.1，代码采用开源的编辑器 vscode 编写。由指定的 Makefile 来完成编译。

文件说明：

- \* my\_linklist.h: 线性表库头文件
- \* my\_linklist.c: 线性表库实现
- \* main.h: 演示系统头文件
- \* main.c: 演示系统实现
- \* CMakeLists.txt: CMake 列表（用以生成完备的 Makefile）
- \* Makefile: 自动化编译命令

### 2.3.2 代码亮点

所有的代码采用 Google C/C++ 标准代码规范，函数库所有注释采用英文，符合生产规范。错误检查和提示全面，根据不同的错误会有不同的提示信息。由于在 Linux 环境下编程，所有的 API 接口命名改用 unix 环境编程的标准短横线命名模式。

### 2.3.3 操作演示

界面展示：

```
panyue@Saltedfish:~/code/my_github/HUST-DataStructure-Labs/lab02$ master . ./lab02
+-----+
| Welcome to panyue's link list demo system!
| Here are some functions you can call:
|
| 1: init_list           2: destroy_list
| 3: clear_list          4: is_list_empty
| 5: list_length          6: get_list_item
| 7: locate_list_item     8: prior_list_item
| 9: next_list_item       10: list_insert
| 11: list_delete         12: print_list
| 13: ls_list              0: quit
|
| Enter the number of the function and see the usage and call it!
| Enter 0 to quit the demo system.
|
| Copyright (C) 2017 Yue Pan
| Github: zxc479773533
+-----+
```

图 2-2 演示系统界面

接下来我们以如下的结构开始，来演示本系统的各项操作：

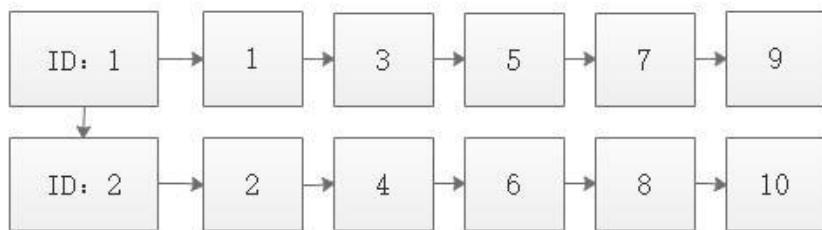


图 2-3 初始的线性表结构

系统中已经存入了两个线性表，结点和 ID 分别如图所示。

### 1. 首先列出存在的线性表

```
+-----+  
| Welcome to panyue's link list demo system!  
| Here are some functions you can call:  
|  
| 1: init_list           2: destroy_list  
| 3: clear_list          4: is_list_empty  
| 5: list_length          6: get_list_item  
| 7: locate_list_item     8: prior_list_item  
| 9: next_list_item       10: list_insert  
| 11: list_delete         12: print_list  
| 13: ls_list              0: quit  
|  
| Enter the number of the function and see the usage and call it!  
| Enter 0 to quit the demo system.  
|  
| Copyright (C) 2017 Yue Pan  
| Github: zxc479773533  
+-----+  
  
Your choose: 13  
ID: 2  
ID: 1
```

图 2-4 初始的线性表结构

## 2. 获得线性表 1 第 2 位的数据

```
Your choose: 6  
/*  
 * Function Name: get_list_item  
 * Module: Data structures  
 * Parameter: link_list L, int order, int *elem  
 * Return: int(status)  
 * Use: get the ordered element of link list  
 */  
  
Format: id index  
Then, enter the list id and index: 1 2  
The element is 3.
```

图 2-5 获得线性表 1 第 2 位的数据

## 3. 获得线性表 2 中 8 的位置

```
Your choose: 7
/*
 * Function Name: locate_list_item
 * Module: Data structures
 * Parameter: link_list L, int ordered_ele
 * Return: int(index)
 * Use: get the index of ordered item
 */

Format: id element
Then, enter the list id and element: 2 8
The index of the 8, is 4.
```

图 2-6 获得线性表 2 中 8 的位置

#### 4. 获得线性表 1 中 5 前一个的数据

```
Your choose: 8
/*
 * Function Name: piror_list_item
 * Module: Data structures
 * Parameter: link_list L, int elem, int *elem_pre
 * Return: int(status)
 * Use: get the ordered element's piror
 */

Format: id element
Then, enter the list id and element: 1 5
The piror element of 5 is 3.
```

图 2-7 获得线性表 1 中 5 前一个的数据

#### 5. 获得线性表 2 中 4 后一个的数据

```
Your choose: 9
/*
 * Function Name: next_list_item
 * Module: Data structures
 * Parameter: link_list L, int elem, int *elem_next
 * Return: int(status)
 * Use: get the ordered element's next
 */

Format: id element
Then, enter the list id and element: 2 4
The next element of 4 is 6.
```

图 2-8 获得线性表 2 中 4 后一个的数据

6. 插入之前，获得线性表 1 的长度

```
Your choose: 5
/*
 * Function Name: list_length
 * Module: Data structures
 * Parameter: link_list L
 * Return: int(the length)
 * Use: get the length of link list
 */

Then, enter the list id: 1
The length of link list 1 is 5.
```

图 2-9 获得线性表 1 的长度

7. 插入之后，获得线性表 1 的长度

```
Your choose: 5
/*
 * Function Name: list_length
 * Module: Data structures
 * Parameter: link_list L
 * Return: int(the length)
 * Use: get the length of link list
 */

Then, enter the list id: 1
The length of link list 1 is 6.
```

图 2-10 获得线性表 1 的长度

8. 删除线性表 2 第 2 个位置的数据

```
Your choose: 11
/*
 * Function Name: list_delete
 * Module: Data structures
 * Parameter: link_list *L, int order, int *elem
 * Return: int(status)
 * Use: delete a element in the link list
 */

Format: id index
Then, enter the list id and index: 2 2
Delete 4 succeed!
```

图 2-11 删除线性表 2 第 2 个位置的数据

9. 删除后，获得线性表 2 的表长

```
Your choose: 5
/*
 * Function Name: list_length
 * Module: Data structures
 * Parameter: link_list L
 * Return: int(the length)
 * Use: get the length of link list
 */

Then, enter the list id: 2
The length of link list 2 is 4.
```

图 2-12 获得线性表 2 的表长

#### 10. 打印线性表 2

```
Your choose: 12
/*
 * Function Name: print_list
 * Module: Data structures
 * Parameter: link_list L, char *payload
 * Return: None
 * Use: print the elements of the linklist to the payload
 */

Then, enter the list id: 2
The elements in link list 2 are:
Index: 1, Data: 2
Index: 2, Data: 6
Index: 3, Data: 8
Index: 4, Data: 10
```

图 2-13 打印线性表 2

#### 11. 清空线性表 2 前，判断是否为空

```
Your choose: 4
/*
 * Function Name: is_list_empty
 * Module: Data structures
 * Parameter: link_list L
 * Return: int(true or false)
 * Use: judge if the link list is empty
 */

Then, enter the list id: 2
Link list 2 is not empty!
```

图 2-14 判断线性表 2 是否为空

12. 清空线性表 2

```
Your choose: 3
/*
 * Function Name: clear_list
 * Module: Data structures
 * Parameter: link_list *L
 * Return: int(status)
 * Use: let the link list empty
 */

Then, enter the list id: 2
Link list 2 has been cleared!
```

图 2-15 清空线性表 2

13. 清空线性表 2 后，判断是否为空

```
Your choose: 4
/*
 * Function Name: is_list_empty
 * Module: Data structures
 * Parameter: link_list L
 * Return: int(true or false)
 * Use: judge if the link list is empty
 */

Then, enter the list id: 2
Link list 2 is empty!
```

图 2-15 判断线性表 2 是否为空

14. 删 除 线 性 表 1 前，列 出 所 有 线 性 表

```
Your choose: 13
ID: 2
ID: 1
```

图 2-16 列出所有线性表

15. 删 除 线 性 表 1

```
Your choose: 2
/*
 * Function Name: destroy_list
 * Module: Data structures
 * Parameter: link_list *L
 * Return: int(status)
 * Use: destroy the link list
 */

Then, enter the list id: 1
Link list 1 has been deleted!
```

图 2-17 删除线性表 1

### 16. 删除线性表 1 后，列出所有线性表

```
Your choose: 13
ID: 2
```

图 2-18 删除线性表 1

### 17. 正常退出系统

```
+-----+
| Welcome to panyue's link list demo system!
| Here are some functions you can call:
|
| 1: init_list          2: destroy_list
| 3: clear_list         4: is_list_empty
| 5: list_length        6: get_list_item
| 7: locate_list_item   8: prior_list_item
| 9: next_list_item     10: list_insert
| 11: list_delete       12: print_list
| 13: ls_list           0: quit
|
| Enter the number of the function and see the usage and call it!
| Enter 0 to quit the demo system.
|
| Copyright (C) 2017 Yue Pan
| Github: zxc479773533
+-----+
Your choose: 0
Thanks for using my demo system!
panyue@Saltedfish ~ /code/my_github/HUST-DataStructure-Labs/lab02 ↵ master ●
```

图 2-19 正常退出系统

通过以上的演示证明，我们的线性表的功能是完整实现了的，可以演示要求的十二种线性表上的运算，并且做到了多线性表管理，实现了文件存储功能，不存在内存泄漏，完整的达到了实验的要求。

### 2.3.4 系统健壮性/错误提示演示

我们仍然以如下的结构开始，来演示错误提示：

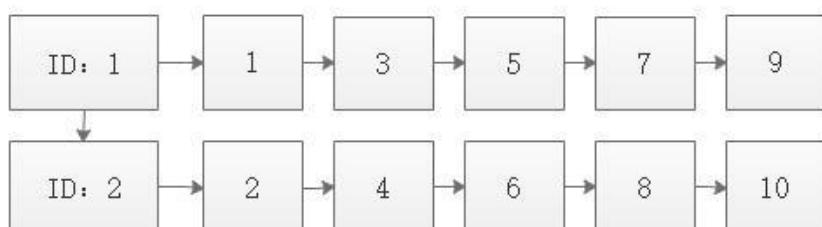


图 2-20 初始的线性表结构

系统中已经存入了两个线性表，结点和 ID 分别如图所示。

1. 初始化已存在的线性表

```
Your choose: 1
/*
 * Function Name: init_list
 * Module: Data structures
 * Parameter: link_list *L
 * Return: int(status)
 * Use: initial the link list
 */

Then, enter the list id: 1
Initial error, this link list is already exists!
```

图 2-21 初始化已存在的线性表

2. 删除不存在的线性表

```
Your choose: 2
/*
 * Function Name: destroy_list
 * Module: Data structures
 * Parameter: link_list *L
 * Return: int(status)
 * Use: destroy the link list
 */

Then, enter the list id: 5
Destroy error, this link list is not exits!
```

图 2-22 删除不存在的线性表

3. 清空不存在的线性表

```
Your choose: 3
/*
 * Function Name: clear_list
 * Module: Data structures
 * Parameter: link_list *L
 * Return: int(status)
 * Use: let the link list empty
 */

Then, enter the list id: 5
Clear error, this link list is not exists!
```

图 2-23 清空不存在的线性表

#### 4. 判断不存在的线性表是否为空

```
Your choose: 4
/*
 * Function Name: is_list_empty
 * Module: Data structures
 * Parameter: link_list L
 * Return: int(true or false)
 * Use: judge if the link list is empty
 */

Then, enter the list id: 5
Error, this link list is not exists!
```

图 2-24 判断不存在的线性表是否为空

#### 5. 求不存在的线性表的长度

```
Your choose: 5
/*
 * Function Name: list_length
 * Module: Data structures
 * Parameter: link_list L
 * Return: int(the length)
 * Use: get the length of link list
 */

Then, enter the list id: 5
Error, this link list is not exists!
```

图 2-25 求不存在的线性表长度

#### 6. 获得线性表中不存在的元素

```
Your choose: 6
/*
 * Function Name: get_list_item
 * Module: Data structures
 * Parameter: link_list L, int order, int *elem
 * Return: int(status)
 * Use: get the ordered element of link list
 */

Format: id index
Then, enter the list id and index: 1 99
Get element error! Please check your input.
```

图 2-26 获得线性表中不存在的元素

#### 7. 寻找线性表中不存在的元素

```
Your choose: 7
/*
 * Function Name: locate_list_item
 * Module: Data structures
 * Parameter: link_list L, int ordered_elem
 * Return: int(index)
 * Use: get the index of ordered item
 */

Format: id element
Then, enter the list id and element: 1 99
The index of the 99, is 0.
```

图 2-27 获得线性表中不存在的元素

#### 8. 获得不存在的元素的前一个元素

```
Your choose: 8
/*
 * Function Name: piror_list_item
 * Module: Data structures
 * Parameter: link_list L, int elem, int *elem_pre
 * Return: int(status)
 * Use: get the ordered element's piror
 */

Format: id element
Then, enter the list id and element: 1 99
Error, the piror of 99 is not exists!
```

图 2-28 获得不存在的元素的前一个元素

#### 9. 获得不存在的元素的后一个元素

```
Your choose: 9
/*
 * Function Name: next_list_item
 * Module: Data structures
 * Parameter: link_list L, int elem, int *elem_next
 * Return: int(status)
 * Use: get the ordered element's next
 */

Format: id element
Then, enter the list id and element: 1 99
Error, the next of 99 is not exists!
```

图 2-29 获得不存在的元素的后一个元素

#### 10. 在不正确的位置插入元素

```
Your choose: 10
/*
 * Function Name: list_insert
 * Module: Data structures
 * Parameter: link_list *L, int order, int elem
 * Return: int(status)
 * Use: insert a element in the link list
 */

Format: id index element
Then, enter the list id, index and element: 1 101 99
Insert failed, please check your input!
```

图 2-30 在不正确的位置插入元素

#### 11. 删 除不存在的元素

```
Your choose: 11
/*
 * Function Name: list_delete
 * Module: Data structures
 * Parameter: link_list *L, int order, int *elem
 * Return: int(status)
 * Use: delete a element in the link list
 */

Format: id index
Then, enter the list id and index: 1 99
Delete failed, please check your input!
```

图 2-31 删除不存在的线性表

## 12. 打印不存在的线性表

```
Your choose: 12
/*
 * Function Name: print_list
 * Module: Data structures
 * Parameter: link_list L, char *payload
 * Return: None
 * Use: print the elements of the linklist to the payload
 */

Then, enter the list id: 5
Error, this link list is not exists!
```

图 2-32 打印不存在的线性表

通过以上的演示证明，我们的线性表演示系统的健壮性是足够的，可以对各种错误输入进行处理，输出相应的错误提示。

## 2.4 实验小结

这次实验内容就很熟悉了，因为链式结构的线性表在之前已经接触过很多了，包括在日常开发用到的一些栈，队列我大都用链式结构实现的，甚至在 C 语

言课程设计中已经使用了三级链表。

整个实验在 linux 环境下编程，所有的代码采用 Google C/C++标准代码规范，函数库所有注释采用英文，符合生产规范。错误检查和提示全面，根据不同的错误会有不同的提示信息。由于在 Linux 环境下编程，所有的 API 接口命名改用 unix 环境编程的标准短横线命名模式。这样规范自己有助于以后走向工作岗位之前的面试环节。

本系统完整的实现了实现的课程要求的全部功能，实现了多线性表管理和文件存储功能，系统健壮性良好，可以输出各种情况下的错误提示，并且在整个实验过程中避免了处处的内存泄漏，完整的达到了预期的效果。



### 3 基于二叉链表的二叉树实现

#### 3.1 问题描述

实现基于链式存储结构的二叉树，并实现二叉树的基本运算。

要求构造一个具有菜单的功能演示系统。其中，在主程序中完成函数调用所需实参值的准备和函数执行结果的显示，并给出适当的操作提示显示。演示系统可选择实现二叉树的文件形式保存。

演示系统可以实现多二叉树管理。

整个系统的设计模式如下：

##### 3.1.1 二叉树抽象数据类型

以下抽象数据类型源自参考文献[1]P121。

依据最小完备性和常用性相结合的原则，设计了二叉树的数据对象和数据关系，并定义了二叉树的初始化二叉树、销毁二叉树、置空二叉树、判定二叉树是否为空树、求二叉树深度和获得指定元素等 20 种基本运算，具体数据和运算功能定义如下。

ADT BinaryTree {

    数据对象 D：D 是具有相同特性的数据元素的集合。

    数据关系 R：

        若  $D=\emptyset$ ，则  $R=\emptyset$ ，称 BinaryTree 为空二叉树；

        若  $D \neq \emptyset$ ，则  $R=\{H\}$ ，H 是如下二元关系：

        在 D 中存在唯一的称为根的数据元素 root，它在关系 H 下无前驱；

        若  $D-\{root\} \neq \emptyset$ ，则存在  $D-\{root\}=\{D_l, D_r\}$ ，且  $D_l \cap D_r = \emptyset$ ；

        若  $D_l \neq \emptyset$ ，则  $D_l$  中存在唯一的元素  $x_l$ ， $\langle root, x_l \rangle \in H$ ，且存在  $D_r$

        上的关系  $H_r \subset H$ ； $H=\{\langle root, x_l \rangle, \langle root, x_r \rangle, H_l, H_r\}$ ；

$(D_l, \{H_l\})$  是一棵符合本定义的二叉树，成为根的左子树， $(D_r, \{H_r\})$

是一颗符合本定义的二叉树，成为根的右子树。

基本操作 P:

InitBiTree(&T)

操作结果：构造空二叉树 T。

DestroyBiTree(&T)

初始条件：二叉树 T 存在。

操作结果：销毁二叉树 T。

CreateBiTree(&T, definition)

初始条件：definition 给出二叉树 T 的定义。

操作结果：按 definition 构造二叉树 T。

ClearBiTree(&T)

初始条件：二叉树 T 存在。

操作结果：将二叉树 T 清为空树。

BiTreeEmpty(T)

初始条件：二叉树 T 存在。

操作结果：若 T 为空二叉树，则返回 TRUE，否则返回 FALSE。

BiTreeDepth(T)

初始条件：二叉树 T 存在。

操作结果：返回 T 的深度。

Root(T)

初始条件：二叉树 T 存在。

操作结果：返回 T 的根。

Value(T, key)

初始条件：二叉树 T 存在，key 是 T 中某个节点的唯一编号。

操作结果：返回由 key 指明的节点的值。

Assign(T, key, value)

初始条件：二叉树 T 存在，key 是 T 中某个节点的唯一编号。

操作结果：由 key 知名的节点复制为 value。

Parent(T, key)

初始条件：二叉树 T 存在，key 是 T 中某个节点的唯一编号。

操作结果：若由 key 指明的节点是 T 的非根节点，则返回它的双亲，否则返回 NULL。

#### LeftChild(T, key)

初始条件：二叉树 T 存在，key 是 T 中某个节点的唯一编号。

操作结果：返回由 key 指明的节点的左孩子。若无左孩子，则返回 NULL。

#### RightChild(T, key)

初始条件：二叉树 T 存在，key 是 T 中某个节点的唯一编号。

操作结果：返回由 key 指明的节点的右孩子。若无右孩子，则返回 NULL。

#### LeftSibling(T, key)

初始条件：二叉树 T 存在，key 是 T 中某个节点的唯一编号。

操作结果：返回由 key 指明的节点的左兄弟。若该节点为其双亲的左孩子或无左兄弟，则返回 NULL。

#### RightSibling(T, key)

初始条件：二叉树 T 存在，key 是 T 中某个节点的唯一编号。

操作结果：返回由 key 指明的节点的右兄弟。若该节点为其双亲的右孩子或无右兄弟，则返回 NULL。

#### InsertChild(T, key, LR, C)

初始条件：二叉树 T 存在，key 是 T 中某个节点的唯一编号，LR 为 0 或 1，非空二叉树 C 与 T 不相交且右子树为空。

操作结果：根据 LR 为 0 或 1，插入 C 为 T 中 key 指明的节点的左或右子树。由 key 指明节点的原有左或右子树则成为 C 的右子树。

#### DeleteChild(T, key, LR)

初始条件：二叉树 T 存在，key 是 T 中某个节点的唯一编号，LR 为 0 或 1。

操作结果：根据 LR 为 0 或 1，删除 T 中 key 指明的节点的左或右子树。

#### PreOrderTraverse(T)

初始条件：二叉树 T 存在。

操作结果：先序遍历 T，并输出每一个节点的值。

InOrderTraverse(T)

初始条件：二叉树 T 存在。

操作结果：中序遍历 T，并输出每一个节点的值。

PostOrderTraverse(T)

初始条件：二叉树 T 存在。

操作结果：后序遍历 T，并输出每一个节点的值。

LevelOrderTraverse(T)

初始条件：二叉树 T 存在。

操作结果：层次遍历遍历 T，并输出每一个节点的值。

} ADT BinaryTree

### 3.1.2 多二叉树的管理

我设计了一个包装链表，将已存在的所有二叉树用链式结构串起，并对每个二叉树赋予独特的 ID 来标识，在每次操作中通过 ID 来选择要进行操作的链表。

### 3.1.3 演示系统和文件存储的设计

没有使用参考文件，自行设计演示框架，该框架将完成函数调用所需实参值的准备和函数执行结果的实现，并给出适当的操作提示显示，整体以命令行呈现。

此外设计的数据文件实时存储，本地数据文件为 binarytreedb，为便于存储又为每个二叉树设计了一个结构体，存放先序遍历和中序遍历的结果，将结构体占有的内存区块直接写入二进制文件中储存。

## 3.2 系统设计

### 3.2.1 数据物理结构

#### 1. 二叉树的存储数据结构

二叉树结构体定义如下：

```
// the main struct of binary tree
typedef struct binary_tree {
```

```
int id; // 二叉树 ID  
int size; // 二叉树结点个数  
struct binary_tree_node *root; // 二叉树的根结点  
struct binary_tree *next; // 指向下一个二叉树，多二叉树管理用  
} binary_tree;
```

## 2. 二叉树结点的存储数据结构

```
// the struct of binary tree node  
typedef struct binary_tree_node {  
    int index; // 键  
    int value; // 值  
    struct binary_tree_node *left_child; // 指向左儿子的指针  
    struct binary_tree_node *right_child; // 指向右儿子的指针  
} binary_tree_node;
```

## 3. 多线性表的存储数据结构

附加一个头结点，以链式结构存储所有的二叉树

```
// the struct for managin the binary tree  
typedef struct Binary_tree_main {  
    binary_tree *head;  
} Binary_tree_main;
```

## 4. 二叉树的存储结构体

```
// the struct to save the binary tree  
typedef struct Binary_tree_store {  
    int id; // 二叉树的 ID  
    int size; // 二叉树的结点个数  
    int *pre_index; // 二叉树先序遍历的 index 数组  
    int *pre_defination; // 二叉树先序遍历的 value 数组
```

```

int *in_index; // 二叉树中序遍历的 index 数组
int *in_definition; // 二叉树中序遍历的 value 数组
} Binary_tree_store;

```

在本程序中，数据原子类型被定义为键值对 int-int 的形式。

### 3.2.2 演示系统

演示系统包括用户操作界面和功能调用部分。

演示系统界面语言为英文，所有操作和提示语言均为英文。

用户操作界面输出可选的线性表操作，用户输入数字选择要进行的操作。在用户选择操作后，功能调用部分会显示函数的名称，参数，返回值和作用，系统提示用户输入参数。

功能调用部分将用户输入的有关信息传递给线性数据结构的操作函数进行调用，并对函数的返回值进行处理判断输出相应的提示信息。

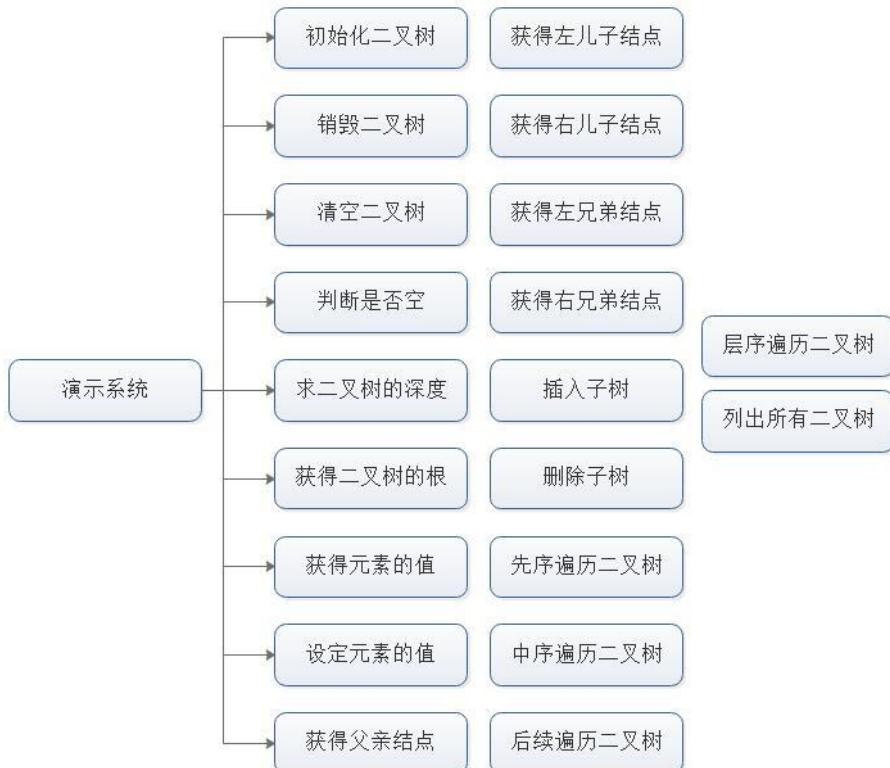


图 3-1 演示系统模块结构图

整个演示系统的设计基本和实验一一致，接口使用相同接口，只是库函数的

实现改变了。

### 3.2.3 二叉树运算实现算法

本部分说明算法时直接用函数原型代替。

1. int init\_binary\_tree(binary\_tree \*T);

**功能:** 初始化二叉树。

**算法实现:** 将二叉树的 `T->size` 赋值为 0, 然后将 `T->root` 赋值为 NULL, 返回 OK。

**时空效率分析:** 算法的时间复杂度为  $O(1)$ , 空间复杂度为  $O(1)$ 。

2. int destroy\_bitree(binary\_tree \*T);

**功能:** 删除二叉树。

**算法实现:** 递归 free 掉二叉树所有的结点, 再 free 掉二叉树结构体自身, 返回 OK。

**时空效率分析:** 由于每个结点都遍历了一次, 所以算法的时间复杂度为  $O(n)$ , 同理空间复杂度为  $O(n)$ 。

3. int create\_bitree(binary\_tree \*T, int \*pre\_index, int \*pre\_definition, int \*in\_index, int \*in\_definition, int definition\_len);

**功能:** 创建二叉树。

**算法实现:** 利用前序遍历与中序遍历构建二叉树, 算法如下:

(1) 寻找前序遍历的第一个结点在中序遍历的位置, 其左侧为该结点的左子树, 右侧为右子树。

(2) 在左子树部分递归此操作

(3) 在右子树部分递归此操作

**时空效率分析:** 由于每个结点都遍历了一次, 所以算法的时间复杂度为  $O(n)$ , 同理空间复杂度为  $O(n)$ 。

4. int clear\_bitree(binary\_tree \*T);

**功能:** 清空二叉树。

**算法实现:** 递归 free 掉二叉树所有的结点，返回 OK。

**时空效率分析:** 同删除二叉树，算法的时间复杂度为  $O(n)$ ，空间复杂度为  $O(n)$ 。

5. int is\_bitree\_empty(binary\_tree T);

**功能:** 判断二叉树是否为空。

**算法实现:** 如果  $T->size$  为 0，就返回 TRUE，否则返回 FALSE。

**时空效率分析:** 算法的时间复杂度为  $O(1)$ ，空间复杂度为  $O(1)$ 。

6. int bitree\_depth(binary\_tree T);

**功能:** 求二叉树的深度。

**算法实现:** 对任意一个结点，返回其左子树的深度和右子树的深度中大的那一个加一。递归执行这个操作，最后返回深度。

**时空效率分析:** 由于每个结点都遍历了一次，所以算法的时间复杂度为  $O(n)$ ，同理空间复杂度为  $O(n)$ 。

7. binary\_tree\_node\* bitree\_root(binary\_tree T);

**功能:** 获得二叉树的根结点。

**算法实现:** 直接返回  $T->root$  即可。

**时空效率分析:** 算法的时间复杂度为  $O(1)$ ，空间复杂度为  $O(1)$ 。

8. int bitree\_get\_value(binary\_tree T, int index);

**功能:** 在二叉树中查找一个结点。

**算法实现:** 从根结点开始, 遍历查找即可, 若找到, 返回其值, 否则返回 0。

**时空效率分析:** 由于结点的位置不确定, 所以算法的时间复杂度为  $O(n)$ , 空间复杂度为  $O(n)$ 。

9. `int bitree_set_value(binary_tree *T, int index, int value);`

**功能:** 设置指定结点的值。

**算法实现:** 同 8 一样, 先找到这个结点, 再给其 `value` 部分赋值, 若找到 返回 OK, 否则返回 ERROR。

**时空效率分析:** 同上一个函数一样, 算法的时间复杂度为  $O(n)$ , 空间复杂度为  $O(n)$ 。

10. `binary_tree_node* bitree_parent(binary_tree T, int index);`

**功能:** 获得指定结点的父结点。

**算法实现:** 递归查找, 若找到某结点的左儿子或者右儿子是指定的结点, 那么返回该节点, 若未找到返回 NULL。

**时空效率分析:** 同查找一致, 算法的时间复杂度为  $O(n)$ , 空间复杂度为  $O(n)$ 。

11. `binary_tree_node* bitree_left_child(binary_tree T, int index);`

**功能:** 获得指定结点的左儿子。

**算法实现:** 递归查找, 若找到指定结点, 返回其左儿子, 若未找到或者其左儿子不存在, 返回 NULL。

**时空效率分析:** 同查找一致, 算法的时间复杂度为  $O(n)$ , 空间复杂度为  $O(n)$ 。

12. `binary_tree_node* bitree_right_child(binary_tree T, int index);`

**功能:** 获得指定结点的右儿子。

**算法实现:** 递归查找, 若找到指定结点, 返回其右儿子, 若未找到或者其右儿子不存在, 返回 NULL。

**时空效率分析:** 同查找一致, 算法的时间复杂度为  $O(n)$ , 空间复杂度为  $O(n)$ 。

```
13. binary_tree_node* bitree_left_sibling(binary_tree T, int index);
```

**功能:** 获得指定结点的左兄弟。

**算法实现:** 递归查找, 若找到指定结点, 返回其左兄弟, 若未找到或者其本身是父结点的左儿子, 返回 NULL。

**时空效率分析:** 同查找一致, 算法的时间复杂度为  $O(n)$ , 空间复杂度为  $O(n)$ 。

```
14. binary_tree_node* bitree_right_sibling(binary_tree T, int index);
```

**功能:** 获得指定结点的右兄弟。

**算法实现:** 递归查找, 若找到指定结点, 返回其右兄弟, 若未找到或者其本身是父结点的右儿子, 返回 NULL。

**时空效率分析:** 同查找一致, 算法的时间复杂度为  $O(n)$ , 空间复杂度为  $O(n)$ 。

```
15. int bitree_insert_child(binary_tree *T, int index, int L or R, binary_tree *C);
```

**功能:** 插入子树。

**算法实现:** 递归查找, 若找到指定结点, 根据 L or R 等于 0 或 1 将二叉树 C 的根结点设置为指定结点的左儿子或者右儿子, 指定结点原来的左儿子或者右儿子被设置为二叉树 C 的根结点的右儿子, 若未找到或者二叉树 C 根结点的右儿子不为空, 则返回 ERROR, 否则返回 OK。

**时空效率分析:** 同查找一致, 算法的时间复杂度为  $O(n)$ , 空间复杂度则为  $O(n+m)$ ,  $m$  为二叉树 C 的结点个数。

16. int bitree\_delete\_child(binary\_tree \*T, int index, int LorR);

**功能:** 删除子树。

**算法实现:** 递归查找, 若找到指定结点, 根据 LorR 为 0 或 1 删除指定结点的左或者右子树, 删除算法等同 clear 函数。

**时空效率分析:** 由于查找和遍历的原因, 算法的时间复杂度为  $O(n)$ , 空间复杂度为  $O(n)$ 。

17. int bitree\_preorder\_traverse(binary\_tree T);

**功能:** 前序遍历并打印二叉树中的各个结点。

**算法实现:** 采用递归来进行前序遍历。

- (1) 打印该结点
- (2) 在该结点左子树上执行前序遍历
- (3) 在该结点右子树上执行前序遍历

**时空效率分析:** 算法的时间复杂度为  $O(n)$ , 空间复杂度为  $O(n)$ 。

18. int bitree\_inorder\_traverse(binary\_tree T);

**功能:** 中序遍历并打印二叉树中的各个结点。

**算法实现:** 采用递归来进行中序遍历。

- (1) 在该结点左子树上执行中序遍历
- (2) 打印该结点
- (3) 在该结点右子树上执行中序遍历

**时空效率分析:** 算法的时间复杂度为  $O(n)$ , 空间复杂度为  $O(n)$ 。

19. int bitree\_postorder\_traverse(binary\_tree T);

**功能:** 后序遍历并打印二叉树中的各个结点。

**算法实现:** 采用递归来进行后序遍历。

- (1) 在该结点左子树上执行后序遍历
- (2) 在该结点右子树上执行后序遍历
- (3) 打印该结点

**时空效率分析：** 算法的时间复杂度为  $O(n)$ ，空间复杂度为  $O(n)$ 。

20. `int bitree_levelorder_traverse(binary_tree T);`

**功能：** 层序遍历并打印二叉树中的各个结点。

**算法实现：** 采用队列来进行层序遍历。

- (1) 根结点进队列
- (2) 若队列不为空，循环：
  - 1) 队首出列，其左儿子和右儿子分别进队列
  - 2) 打印出列的结点

**时空效率分析：** 算法的时间复杂度为  $O(n)$ ，空间复杂度为  $O(n)$ 。

### 3.2.4 多二叉树管理实现算法

由于多二叉树的管理采用链式线性表模型，只涉及到查找操作，所以时间复杂度是遍历用的  $O(n)$ ，空间复杂度为  $O(1)$ 。

### 3.2.5 文件存储实现算法

1. `void load_data(Binary_tree_main *Main_T);`

**功能：** 数据读取。

**算法实现：** 打开文件，依次读取一个二叉树存储结构体，根据结构体内的数据调用 `creat_bitree` 来创建二叉树，直到所有的二叉树被创建完毕。

**时空效率分析：** 算法的时间复杂度为  $O(n * m)$ ，为创建  $n$  个二叉树，平均每个  $m$  个结点；空间复杂度为  $O(n * m)$ ，为申请  $n * m$  个二叉树结点的空间。

2. `void save_data(Link_list_main *Main_L);`

**功能：** 数据保存。

**算法实现：** 打开文件，依次构造一个二叉树存储结构体，进行一次先序遍历

和中序遍历，将结果存入二叉树存储结构体，然后将该结构体写入文件，直到所有的二叉树被写入文件。

**时空效率分析：** 算法的时间复杂度为  $O(n * m)$ ，为遍历  $n$  个二叉树，平均每个  $m$  个结点；空间复杂度为  $O(n * m)$ ，为  $n * m$  个二叉树结点的空间。

### 3.3 系统实现

#### 3.3.1 实验环境

实验环境为 Arch linux 4.14.8-1，编译器为 gcc 版本 7.2.1，代码采用开源的编辑器 vscode 编写。由指定的 Makefile 来完成编译。

文件说明：

- \* my\_binarytree.h: 二叉树库头文件
- \* my\_binarytree.c: 二叉树库实现
- \* main.h: 演示系统头文件
- \* main.c: 演示系统实现
- \* mylibqueue.c: 自用队列库
- \* CMakeLists.txt: CMake 列表（用以生成完备的 Makefile）
- \* Makefile: 自动化编译命令

#### 3.3.2 代码亮点

所有的代码采用 Google C/C++ 标准代码规范，函数库所有注释采用英文，符合生产规范。错误检查和提示全面，根据不同的错误会有不同的提示信息。由于在 Linux 环境下编程，所有的 API 接口命名改用 unix 环境编程的标准短横线命名模式。

#### 3.3.3 操作演示

界面展示：

```

panyue@Saltedfish:~/code/my_github/HUST-DataStructure-Labs/lab03$ master ./lab03
+-----+
Welcome to panyue's binary tree demo system!
Here are some functions you can call:

1: init_bitree           2: destroy_bitree
3: create_bitree          4: clear_bitree
5: is_bitree_empty        6: bitree_depth
7: bitree_root            8: bitree_get_value
9: bitree_set_value       10: bitree_parent
11: bitree_left_child     12: bitree_right_child
13: bitree_left_sibling   14: bitree_right_sibling
15: bitree_insert_child   16: bitree_delete_child
17: bitree_preorder_traverse 18: bitree_inorder_traverse
19: bitree_postorder_traverse 20: bitree_levelorder_traverse
21: ls_bitree              0: quit

Enter the number of the function and see the usage and call it!
Enter 0 to quit the demo system.

Copyright (C) 2017 Yue Pan
Github: zxc479773533
+-----+
Your choose: 

```

图 3-2 演示系统界面

接下来我们以如下的结构开始，来演示本系统的各项操作：

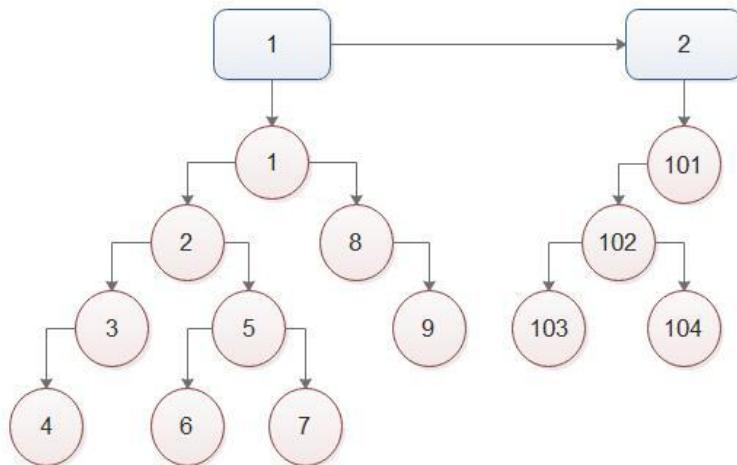


图 3-3 初始的二叉树结构

我们首先向系统中存入两个二叉树，结点和 ID 分别如图所示，为便于测试，每个结点中设置 index 和 value 一样。

### 1. 首先创建二叉树 2

```
Your choose: 3
/*
 * Function Name: create_bitree
 * Module: Data structures
 * Parameter: binary_tree *T, int *pre_index, int *pre_defination,
 * int *in_index, int *in_defination, int difinition_len
 * Return: int(status)
 * Use: create the binary tree with some nodes
 */

Format: id size
Then, enter the binary tree id and tree size: 1 9
Please enter the preorder sequence: (Format: index value)
1 1
2 2
3 3
4 4
5 5
6 6
7 7
8 8
9 9
Please enter the inorder sequence: (Format: index value)
4 4
3 3
2 2
6 6
5 5
7 7
1 1
8 8
9 9
Create binary tree 1 succeed!
```

图 3-4 创建二叉树 1

## 2. 创建二叉树 2

```
Your choose: 3
/*
 * Function Name: create_bitree
 * Module: Data structures
 * Parameter: binary_tree *T, int *pre_index, int *pre_defination,
 * int *in_index, int *in_defination, int difinition_len
 * Return: int(status)
 * Use: create the binary tree with some nodes
 */

Format: id size
Then, enter the binary tree id and tree size: 2 4
Please enter the preorder sequence: (Format: index value)
101 101
102 102
103 103
104 104
Please enter the inorder sequence: (Format: index value)
103 103
102 102
104 104
101 101
Create binary tree 2 succeed!
```

图 3-5 创建二叉树 2

## 3. 获得二叉树 1 的深度，一定是 4

```
Your choose: 6
/*
 * Function Name: bitree_depth
 * Module: Data structures
 * Parameter: binary_tree T
 * Return: int(the depth)
 * Use: get the depth of binary tree
 */

Then, enter the binary tree id: 1
The length of binary tree 1 is 4.
```

图 3-6 获得二叉树 1 的深度

#### 4. 获得二叉树 1 的根结点

```
Your choose: 7
/*
 * Function Name: bitree_root
 * Module: Data structures
 * Parameter: binary_tree T
 * Return: binary_tree_node*
 * Use: get the root of binary tree
 */

Then, enter the binary tree id: 1
Root of binary tree 1:
Index: 1
Value: 1
```

图 3-7 获得二叉树 1 的深度

#### 5. 获得二叉树 1 中结点 5 的值

```
Your choose: 8
/*
 * Function Name: bitree_get_value
 * Module: Data structures
 * Parameter: binary_tree T, int index, int value
 * Return: int(the value)
 * Use: get the value of a node
 */

Format: id index
Then, enter the binary tree id and node index: 1 5
Binary tree 1, Index 5
The value is: 5
```

图 3-8 获得二叉树 1 中结点 5 的值

#### 6. 前序遍历二叉树 1

```
Your choose: 17
/*
 * Function Name: bitree_preorder_traverse
 * Module: Data structures
 * Parameter: binary_tree T
 * Return: int(status)
 * Use: preorder traverse and print the values
 */

Then, enter the binary tree id: 1
Preorder traverse:
Tree size: 9
Index: 1, value: 1
Index: 2, value: 2
Index: 3, value: 3
Index: 4, value: 4
Index: 5, value: 5
Index: 6, value: 6
Index: 7, value: 7
Index: 8, value: 8
Index: 9, value: 9
```

图 3-9 前序遍历二叉树 1

#### 7. 中序遍历二叉树 1

```
Your choose: 18
/*
 * Function Name: bitree_inorder_traverse
 * Module: Data structures
 * Parameter: binary_tree T
 * Return: int(status)
 * Use: inorder traverse and print the values
 */

Then, enter the binary tree id: 1
Inorder traverse:
Tree size: 9
Index: 4, value: 4
Index: 3, value: 3
Index: 2, value: 2
Index: 6, value: 6
Index: 5, value: 5
Index: 7, value: 7
Index: 1, value: 1
Index: 8, value: 8
Index: 9, value: 9
```

图 3-10 中序遍历二叉树 1

### 8. 后序遍历二叉树 1

```
Your choose: 19
/*
 * Function Name: bitree_postorder_traverse
 * Module: Data structures
 * Parameter: binary_tree T
 * Return: int(status)
 * Use: postorder traverse and print the values
 */

Then, enter the binary tree id: 1
Postorder traverse:
Tree size: 9
Index: 4, value: 4
Index: 3, value: 3
Index: 6, value: 6
Index: 7, value: 7
Index: 5, value: 5
Index: 2, value: 2
Index: 9, value: 9
Index: 8, value: 8
Index: 1, value: 1
```

图 3-11 后序遍历二叉树 1

9. 层序遍历二叉树 1

```
Your choose: 20
/*
 * Function Name: bitree_levelorder_traverse
 * Module: Data structures
 * Parameter: binary_tree T
 * Return: int(status)
 * Use: levelorder traverse and print the values
 */

Then, enter the binary tree id: 1
Levelorder traverse:
Tree size: 9
Index: 1, value: 1
Index: 2, value: 2
Index: 8, value: 8
Index: 3, value: 3
Index: 5, value: 5
Index: 9, value: 9
Index: 4, value: 4
Index: 6, value: 6
Index: 7, value: 7
```

图 3-12 层序遍历二叉树 1

10. 设置二叉树 1 中结点 9 的值为 100

```
Your choose: 9
/*
 * Function Name: bitree_set_value
 * Module: Data structures
 * Parameter: binary_tree *T, int index, int value
 * Return: int(status)
 * Use: set the value of a node
 */

Format: id index value
Then, enter the binary tree id, the node index and the value: 1 9 100
Set the node 9's value succeed!
```

图 3-13 设置二叉树 1 中结点 9 的值为 100

11. 设置之后，再层序遍历二叉树 1

```
Your choose: 20
/*
 * Function Name: bitree_levelorder_traverse
 * Module: Data structures
 * Parameter: binary_tree T
 * Return: int(status)
 * Use: levelorder traverse and print the values
 */

Then, enter the binary tree id: 1
Levelorder traverse:
Tree size: 9
Index: 1, value: 1
Index: 2, value: 2
Index: 8, value: 8
Index: 3, value: 3
Index: 5, value: 5
Index: 9, value: 100
Index: 4, value: 4
Index: 6, value: 6
Index: 7, value: 7
```

图 3-14 层序遍历二叉树 1

### 12. 获得二叉树 1 中结点 5 的父结点

```
Your choose: 10
/*
 * Function Name: bitree_parent
 * Module: Data structures
 * Parameter: binary_tree T, int index
 * Return: binary_tree_node*
 * Use: get parent of a node
 */

Format: id index
Then, enter the binary tree id and node index: 1 5
The parent of node 5
Key: 2
Value: 2
```

图 3-15 获得二叉树 1 中结点 5 的父结点

### 13. 获得二叉树 1 中结点 5 的左儿子

```
Your choose: 11
/*
 * Function Name: bitree_left_child
 * Module: Data structures
 * Parameter: binary_tree T, int index
 * Return: binary_tree_node*
 * Use: get left child of a node
 */

Format: id index
Then, enter the binary tree id and node index: 1 5
The left child of node 5
Key: 6
Value: 6
```

图 3-16 获得二叉树 1 中结点 5 的左儿子

#### 14. 获得二叉树 1 中结点 5 的右儿子

```
Your choose: 12
/*
 * Function Name: bitree_right_child
 * Module: Data structures
 * Parameter: binary_tree T, int index
 * Return: binary_tree_node*
 * Use: get right child of a node
 */

Format: id index
Then, enter the binary tree id and node index: 1 5
The right child of node 5
Key: 7
Value: 7
```

图 3-17 获得二叉树 1 中结点 5 的右儿子

#### 15. 获得二叉树 1 中结点 5 的左兄弟

```
Your choose: 13
/*
 * Function Name: bitree_left_sibling
 * Module: Data structures
 * Parameter: binary_tree T, int index
 * Return: binary_tree_node*
 * Use: get left sibling of a node
 */

Format: id index
Then, enter the binary tree id and node index: 1 5
The left sibling of node 5
Key: 3
Value: 3
```

图 3-18 获得二叉树 1 中结点 5 的左兄弟

### 16. 获得二叉树 1 中结点 6 的右兄弟

```
Your choose: 14
/*
 * Function Name: bitree_right_sibling
 * Module: Data structures
 * Parameter: binary_tree T, int index
 * Return: binary_tree_node*
 * Use: get right sibling of a node
 */

Format: id index
Then, enter the binary tree id and node index: 1 6
The right sibling of node 6
Key: 7
Value: 7
```

图 3-19 获得二叉树 1 中结点 6 的右兄弟

### 17. 将二叉树 2 插入为二叉树 1 中结点 5 的左儿子

```
Your choose: 15
/*
 * Function Name: bitree_insert_child
 * Module: Data structures
 * Parameter: binary_tree *T, int index, int LorR, binary_tree *C
 * Return: int(status)
 * Use: insert a child tree in a binary tree
 */

Format: id index LorR child_id
Then, enter the binary tree id, index, LorR, child tree id: 1 5 0 2
Insert child tree 2 succeed!
```

图 3-20 插入子树

插入子树后，二叉树应该是以下这个样子：

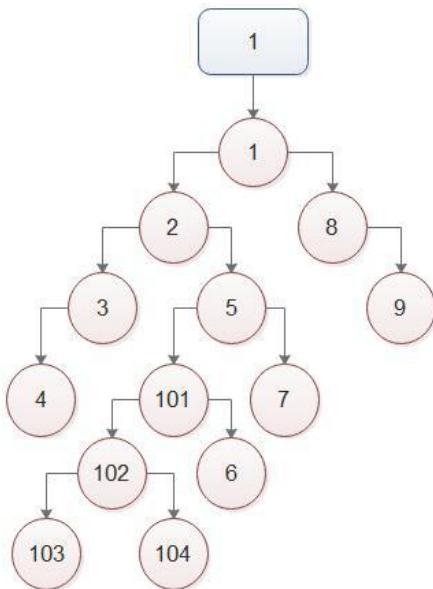


图 3-21 插入子树后二叉树的状态

18. 获得二叉树 1 的深度，此时是 6

```
Your choose: 6
/*
 * Function Name: bitree_depth
 * Module: Data structures
 * Parameter: binary_tree T
 * Return: int(the depth)
 * Use: get the depth of binary tree
 */
```

```
Then, enter the binary tree id: 1
The length of binary tree 1 is 6.
```

图 3-22 获得二叉树 1 的深度

19. 插入子树后，层序遍历二叉树 1

```
Your choose: 20
/*
 * Function Name: bitree_levelorder_traverse
 * Module: Data structures
 * Parameter: binary_tree T
 * Return: int(status)
 * Use: levelorder traverse and print the values
 */

Then, enter the binary tree id: 1
Levelorder traverse:
Tree size: 13
Index: 1, value: 1
Index: 2, value: 2
Index: 8, value: 8
Index: 3, value: 3
Index: 5, value: 5
Index: 9, value: 100
Index: 4, value: 4
Index: 101, value: 101
Index: 7, value: 7
Index: 102, value: 102
Index: 6, value: 6
Index: 103, value: 103
Index: 104, value: 104
```

图 3-23 层序遍历二叉树 1

## 20. 删除二叉树 1 的根结点的左子树

```
Your choose: 16
/*
 * Function Name: bitree_delete_child
 * Module: Data structures
 * Parameter: binary_tree *T, int index, int LorR
 * Return: int(status)
 * Use: delete a child tree in a binary tree
 */

Format: id index LorR
Then, enter the binary tree id, index and LorR: 1 1 0
Delete child tree succeed!
```

图 3-24 删除二叉树 1 的根结点的左子树

删除子树后，二叉树应该是以下这个样子

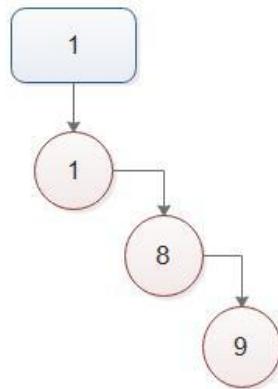


图 3-25 删除子树后二叉树的状态

## 21. 删除子树后，层序遍历二叉树

```
Your choose: 20
/*
 * Function Name: bitree_levelorder_traverse
 * Module: Data structures
 * Parameter: binary_tree T
 * Return: int(status)
 * Use: levelorder traverse and print the values
 */

Then, enter the binary tree id: 1
Levelorder traverse:
Tree size: 3
Index: 1, value: 1
Index: 8, value: 8
Index: 9, value: 100
```

图 3-26 层序遍历二叉树

## 22. 清空之前，判断二叉树 1 是否为空

```
Your choose: 5
/*
 * Function Name: is_bitree_empty
 * Module: Data structures
 * Parameter: binary_tree T
 * Return: int(true or false)
 * Use: judge if the binary tree is empty
 */

Then, enter the binary tree id: 1
Binary tree 1 is not empty!
```

图 3-27 判断二叉树是否为空

23. 清空二叉树 1

```
Your choose: 4
/*
 * Function Name: clear_bitree
 * Module: Data structures
 * Parameter: binary_tree *T
 * Return: int(status)
 * Use: clear the binary tree
 */

Then, enter the binary tree id: 1
Binary tree 1 has been cleared!
```

图 3-28 清空二叉树 1

24. 清空之后，判断二叉树 1 是否为空

```
Your choose: 5
/*
 * Function Name: is_bitree_empty
 * Module: Data structures
 * Parameter: binary_tree T
 * Return: int(true or false)
 * Use: judge if the binary tree is empty
 */

Then, enter the binary tree id: 1
Binary tree 1 is empty!
```

图 3-29 判断二叉树是否为空

25. 删除二叉树 1 之前，列出所有二叉树

```
Your choose: 21
ID: 1
```

图 3-30 列出所有二叉树

26. 删除二叉树 1

```
Your choose: 2
/*
 * Function Name: destroy_bitree
 * Module: Data structures
 * Parameter: binary_tree *T
 * Return: int(status)
 * Use: destroy the binary tree
 */

Then, enter the binary tree id: 1
Binary tree 1 has been deleted!
```

图 3-31 删除二叉树 1

### 27. 删除二叉树 1 之后，列出所有二叉树

```
Your choose: 21
```

图 3-32 列出所有二叉树

### 28. 正常退出系统

```
+-----+
| Welcome to panyue's binary tree demo system!
| Here are some functions you can call:
|
| 1: init_binary_tree          2: destroy_bitree
| 3: create_bitree             4: clear_bitree
| 5: is_bitree_empty           6: bitree_depth
| 7: bitree_root               8: bitree_get_value
| 9: bitree_set_value          10: bitree_parent
| 11: bitree_left_child        12: bitree_right_child
| 13: bitree_left_sibling      14: bitree_right_sibling
| 15: bitree_insert_child       16: bitree_delete_child
| 17: bitree_preorder_traverse 18: bitree_inorder_traverse
| 19: bitree_postorder_traverse 20: bitree_levelorder_traverse
| 21: ls_bitree                 0: quit
|
| Enter the number of the function and see the usage and call it!
| Enter 0 to quit the demo system.
|
| Copyright (C) 2017 Yue Pan
| Github: zxc479773533
+-----+
Your choose: 0
Thanks for using my demo system!
panyue@Saltedfish ~ /code/my_github/HUST-DataStructure-Labs/lab03 > ↵ master ➔
```

图 3-33 正常退出系统

通过以上的演示证明，我们的二叉树的功能是完整实现了的，可以演示要求的二十种二叉树上的运算，并且做到了二叉树表管理，实现了文件存储功能，不存在内存泄漏，完整的达到了实验的要求。

### 3.3.4 系统健壮性/错误提示演示

我们仍然以如下的结构开始，来演示错误提示：

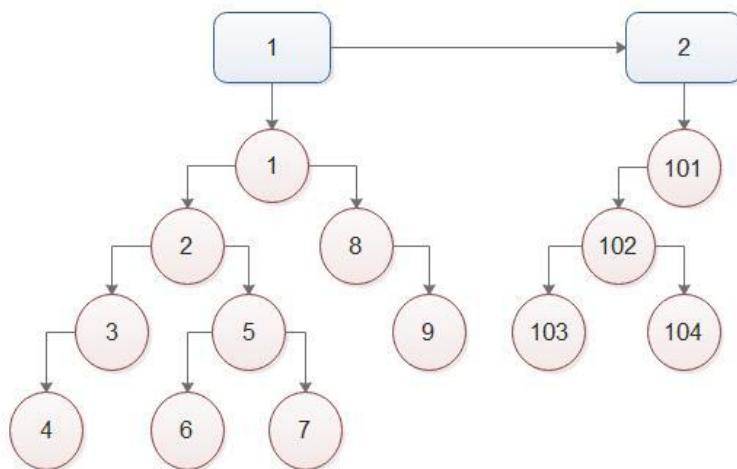


图 3-34 初始的二叉树结构

系统中已经存入了两个线性表，结点和 ID 分别如图所示。

#### 1. 初始化已存在的二叉树

```

Your choose: 1
/*
 * Function Name: init_binary_tree
 * Module: Data structures
 * Parameter: binary_tree *T
 * Return: int(status)
 * Use: initial the binary tree
 */

Then, enter the bintry tree id: 1
Initial error, this bintry tree is already exists!
  
```

图 3-35 初始化已存在的二叉树

#### 2. 删除不存在的二叉树

```
Your choose: 2
/*
 * Function Name: destroy_bitree
 * Module: Data structures
 * Parameter: binary_tree *T
 * Return: int(status)
 * Use: destroy the binary tree
 */

Then, enter the binary tree id: 5
Destroy error, this binary trss is not exists!
```

图 3-36 删除不存在的二叉树

### 3. 创建已存在的二叉树

```
Your choose: 3
/*
 * Function Name: create_bitree
 * Module: Data structures
 * Parameter: binary_tree *T, int *pre_index, int *pre_defination,
 * int *in_index, int *in_defination, int difination_len
 * Return: int(status)
 * Use: create the binary tree with some nodes
 */

Format: id size
Then, enter the binary tree id and tree size: 1 9
Create error, this binary tree is already exists!
```

图 3-37 创建已存在的二叉树

### 4. 清空不存在的二叉树

```
Your choose: 4
/*
 * Function Name: clear_bitree
 * Module: Data structures
 * Parameter: binary_tree *T
 * Return: int(status)
 * Use: clear the binary tree
 */

Then, enter the binary tree id: 5
Clear error, this binary tree is not exists!
```

图 3-38 清空不存在的二叉树

5. 判断不存在的二叉树是否为空

```
Your choose: 5
/*
 * Function Name: is_bitree_empty
 * Module: Data structures
 * Parameter: binary_tree T
 * Return: int(true or false)
 * Use: judge if the binary tree is empty
 */

Then, enter the binary tree id: 5
Error, this binary tree is not exists!
```

图 3-39 判断不存在的二叉树是否为空

6. 求不存在的二叉树的深度

```
Your choose: 6
/*
 * Function Name: bitree_depth
 * Module: Data structures
 * Parameter: binary_tree T
 * Return: int(the depth)
 * Use: get the depth of binary tree
 */

Then, enter the binary tree id: 5
Error, this binary tree is not exists!
```

图 3-40 求不存在的二叉树深度

7. 获得不存在的二叉树的根结点

```
Your choose: 7
/*
 * Function Name: bitree_root
 * Module: Data structures
 * Parameter: binary_tree T
 * Return: binary_tree_node*
 * Use: get the root of binary tree
 */

Then, enter the binary tree id: 5
Error, this binary tree is not exists!
```

图 3-41 获得不存在的二叉树的根结点

#### 8. 获得二叉树中不存在的结点

```
Your choose: 8
/*
 * Function Name: bitree_get_value
 * Module: Data structures
 * Parameter: binary_tree T, int index, int value
 * Return: int(the value)
 * Use: get the value of a node
 */

Format: id index
Then, enter the binary tree id and node index: 1 10
Binary tree 1, Index 10
The value is: 0
```

图 3-42 获得二叉树中不存在的结点

#### 9. 设定二叉树中不存在的结点的值

```
Your choose: 9
/*
 * Function Name: bitree_set_value
 * Module: Data structures
 * Parameter: binary_tree *T, int index, int value
 * Return: int(status)
 * Use: set the value of a node
 */

Format: id index value
Then, enter the binary tree id, the node index and the value: 1 10 10
Error, the node 10 is not found!
```

图 3-43 设定二叉树中不存在的结点的值

10. 父结点不存在

```
Your choose: 10
/*
 * Function Name: bitree_parent
 * Module: Data structures
 * Parameter: binary_tree T, int index
 * Return: binary_tree_node*
 * Use: get parent of a node
 */

Format: id index
Then, enter the binary tree id and node index: 1 1
Error, the parent not found!
```

图 3-44 父结点不存在

11. 左儿子不存在

```
Your choose: 11
/*
 * Function Name: bitree_left_child
 * Module: Data structures
 * Parameter: binary_tree T, int index
 * Return: binary_tree_node*
 * Use: get left child of a node
 */

Format: id index
Then, enter the binary tree id and node index: 1 4
Error, the left child not found!
```

图 3-45 左儿子不存在

12. 右儿子不存在

```
Your choose: 12
/*
 * Function Name: bitree_right_child
 * Module: Data structures
 * Parameter: binary_tree T, int index
 * Return: binary_tree node*
 * Use: get right child of a node
 */

Format: id index
Then, enter the binary tree id and node index: 1 7
Error, the right child not found!
```

图 3-46 右儿子不存在

13. 左兄弟不存在

```
Your choose: 13
/*
 * Function Name: bitree_left_sibling
 * Module: Data structures
 * Parameter: binary_tree T, int index
 * Return: binary_tree node*
 * Use: get left sibling of a node
 */

Format: id index
Then, enter the binary tree id and node index: 1 2
Error, the left sibling not found!
```

图 3-47 左兄弟不存在

14. 右兄弟不存在

```
Your choose: 14
/*
 * Function Name: bitree_right_sibling
 * Module: Data structures
 * Parameter: binary_tree T, int index
 * Return: binary_tree node*
 * Use: get right sibling of a node
 */

Format: id index
Then, enter the binary tree id and node index: 1 8
Error, the right sibling not found!
```

图 3-48 右兄弟不存在

### 15. 插入不存在的子树

```
Your choose: 15
/*
 * Function Name: bitree_insert_child
 * Module: Data structures
 * Parameter: binary_tree *T, int index, int LorR, binary_tree *C
 * Return: int(status)
 * Use: insert a child tree in a binary tree
 */

Format: id index LorR child_id
Then, enter the binary tree id, index, LorR, child tree id: 2 5 0 1
Error, this binary tree is not exists!
```

图 3-49 插入不存在的子树

### 16. 删除不存在的子树

```
Your choose: 16
/*
 * Function Name: bitree_delete_child
 * Module: Data structures
 * Parameter: binary_tree *T, int index, int LorR
 * Return: int(status)
 * Use: delete a child tree in a binary tree
 */

Format: id index LorR
Then, enter the binary tree id, index and LorR: 1 10 0
Delete error, please check your input!
```

图 3-50 删除不存在的子树

### 17. 前序遍历不存在的二叉树

```
Your choose: 17
/*
 * Function Name: bitree_preorder_traverse
 * Module: Data structures
 * Parameter: binary_tree T
 * Return: int(status)
 * Use: preorder traverse and print the values
 */

Then, enter the binary tree id: 2
Error, this binary tree is not exists!
```

图 3-51 前序遍历不存在的二叉树

### 18. 中序遍历不存在的二叉树

```
Your choose: 18
/*
 * Function Name: bitree_inorder_traverse
 * Module: Data structures
 * Parameter: binary_tree T
 * Return: int(status)
 * Use: inorder traverse and print the values
 */

Then, enter the binary tree id: 2
Error, this binary tree is not exists!
```

图 3-52 中序遍历不存在的二叉树

### 19. 后序遍历不存在的二叉树

```
Your choose: 19
/*
 * Function Name: bitree_postorder_traverse
 * Module: Data structures
 * Parameter: binary_tree T
 * Return: int(status)
 * Use: postorder traverse and print the values
 */

Then, enter the binary tree id: 2
Error, this binary tree is not exists!
```

图 3-53 后序遍历不存在的二叉树

## 20. 层序遍历不存在的二叉树

```
Your choose: 20
/*
 * Function Name: bitree_levelorder_traverse
 * Module: Data structures
 * Parameter: binary_tree T
 * Return: int(status)
 * Use: levelorder traverse and print the values
 */

Then, enter the binary tree id: 2
Error, this binary tree is not exists!
```

图 3-54 层序遍历不存在的二叉树

通过以上的演示证明，我们的二叉树演示系统的健壮性是足够的，可以对各种错误输入进行处理，输出相应的错误提示。

### 3.4 实验小结

这次实验和之前相比就略有复杂了，主要锻炼我们使用递归的能力。我的二叉树创建算法使用了前序遍历序列加上中序遍历序列的方式来唯一确定二叉树，其实也可以使用前序带空的序列来创建二叉树。

同时实验中用到了一个队列库 mylibqueue，是一个链式结构实现的队列，详情在后面的附录中介绍。

自己在实验过程中遇到的问题主要出现在文件存储上，存储的内存区块多了以后，就没有分清哪个二叉树是哪个的。于是在后期我又重新设计了二叉树的存储结构，这样大大减小了存储的内存大小，同时也使得处理变得方便起来，同时存储的结构还可以直接调用接口创建二叉树，可以说是非常舒服了。

事实上本实验作为最基本的二叉树，主要是让我们熟练底层结构（倒不如说是练习 C 语言），本人曾经写过 AVL 树和在其基础上再改进的左倾红黑树，关于树的算法联系也算是完全了。

整个实验在 linux 环境下编程，所有的代码采用 Google C/C++ 标准代码规范，函数库所有注释采用英文，符合生产规范。错误检查和提示全面，根据不同的错误会有不同的提示信息。由于在 Linux 环境下编程，所有的 API 接口命名改用 unix 环境编程的标准短横线命名模式。这样规范自己有助于以后走向工作岗位之前的面试环节。

本系统完整的实现了实现的课程要求的全部功能，实现了多二叉树管理和文件存储功能，系统健壮性良好，可以输出各种情况下的错误提示，并且在整个实验过程中避免了处处的内存泄漏，完整的达到了预期的效果。

## 4 基于邻接表的图实现

### 4.1 问题描述

实现基于邻接表存储结构的图，并实现图的基本运算。

实验中要求的图类型有：无向图，有向图，无向网，有向网，其中网是指带权的图。

要求构造一个具有菜单的功能演示系统。其中，在主程序中完成函数调用所需实参值的准备和函数执行结果的显示，并给出适当的操作提示显示。演示系统可选择实现二叉树的文件形式保存。

演示系统可以实现多二叉树管理。

整个系统的设计模式如下：

#### 4.1.1 图抽象数据类型

以下抽象数据类型源自参考文献[1]P156。

依据最小完备性和常用性相结合的原则，设计了图的数据对象和数据关系，并定义了图的初始化图、销毁图、查找顶点、获取邻接顶点等 13 种基本运算，具体数据和运算功能定义如下。

ADT MGraph{

    数据对象：  $n = n$  是具有相同特征的数据元素集合，称为顶点集。

    数据关系：  $DR = \{<v, w> | v, w \in n \text{ 且 } <v, w> \text{ 表示从 } v \text{ 指向 } w \text{ 的弧}\}$

基本操作：

**CreateMGraph**

    初始条件：  $n$  是图的顶点集，  $e$  是图的边集

    操作结果： 按和  $n$  的  $e$  定义构造图  $G$

**DestroyGraph**

    初始条件： 图  $G$  存在

    操作结果： 销毁图  $G$

**GetVex**

初始条件: 图 G 存在, v 是 G 中某个顶点

操作结果: 返回 v 的值

#### LocateVex

初始条件: 图 G 存在, v 和 G 中顶点有相同特征

操作结果: 若 G 中存在顶点 v, 则返回该顶点再图中的位置; 否则返回空

#### PutVex

初始条件: 图 G 存在, v 是 G 中某个顶点

操作结果: 对 v 赋值 u

#### FirstAdjVex

初始条件: 图 G 存在, v 是 G 中某个顶点

操作结果: 返回的第一个邻接顶点。若顶点在 G 中没有邻接顶点, 则返回空

#### NextAdjVex

初始条件: 图 G 存在, v 是 G 中某个顶点, w 是 v 的邻接顶点

操作结果: 返回 v (相对 w) 的下一个邻接顶点。若 w 是 v 的最后一个邻接点, 则返回空

#### InsertVex

初始条件: 图 G 存在, v 和图 G 中顶点有相同特征

操作结果: 在图 G 中增添新顶点 v(不增添与顶点相关的边, 留待 InsertArc() 去做)

#### DeleteVex

初始条件: 图 G 存在, v 是 G 中某个顶点

操作结果: 删除 G 中顶点 v 及其相关的弧

#### InsertArc

初始条件: 图 G 存在, v 和 w 是 G 中两个顶点

操作结果: 在 G 中增添弧<v,w>

#### DeleteArc

初始条件: 图 G 存在, v 和 w 是 G 中两个顶点

操作结果：在 G 中删除弧 $\langle v, w \rangle$

DFSTraverseM

初始条件：图 G 存在

操作结果：对图进行深度优先遍历

BFSTraverseM

初始条件：图 G 存在

操作结果：对图进行广度优先遍历

}ADT MGraph

#### 4.1.2 多图的管理

我设计了一个包装链表，将已存在的所有图用链式结构串起，并对每个图赋予独特的 ID 来标识，在每次操作中通过 ID 来选择要进行操作的链表。

#### 4.1.3 演示系统和文件存储的设计

没有使用参考文件，自行设计演示框架，该框架将完成函数调用所需实参值的准备和函数执行结果的实现，并给出适当的操作提示显示，整体以命令行呈现。

此外设计的数据文件实时存储，本地数据文件为 graphdb，为便于存储又为每个二叉树设计了一个结构体，存放先序遍历和中序遍历的结果，将结构体占有的内存区块直接写入二进制文件中储存。

### 4.2 系统设计

#### 4.2.1 数据物理结构

##### 1. 图的存储数据结构

我们这里将图的类型（无向图/无向网/有向图/有向网）设计成图结构体本身的一种属性，再根据图本身的属性来决定各种操作的进行。

图结构体定义如下：

```
// the struct of graph
typedef struct graph {
    int id; // 图的 ID
    vertex_node *first_vertex; // 指向第一个结点的指针
```

```
int vertex_num; // 结点的个数  
int arc_num; // 边的格式  
int kind; // 图的种类，包含无向图/无向网/有向图/有向网四种  
struct graph *next; // 指向下一个图，多图管理用  
} graph;
```

## 2. 图结点的存储数据结构

```
// the struct of vertex  
typedef struct vertex_node {  
    int index; // 键  
    int value; // 值  
    arc_node *first_arc; // 该点所邻接的第一条边  
    struct vertex_node *next; // 指向下一个结点  
} vertex_node;
```

## 2. 图的边的存储数据结构

```
// the struct of arc  
typedef struct arc_node {  
    int vertex; // 该边指向的结点  
    struct arc_node *next; // 指向下一条边  
    int weight; // 该边的权值  
} arc_node;
```

## 3. 多图的存储数据结构

附加一个头结点，以链式结构存储所有的图

```
// the struct for managin the graph  
typedef struct Graph_main {  
    graph *head;  
} Graph_main;
```

#### 4. 二叉树的存储结构体

```
// the struct to save the graph
typedef struct Graph_store {
    int id; // 二叉树的 ID
    int kind; // 二叉树的种类
    int vertex_num; // 结点的个数
    int arc_num; // 边的个数
    int *v_indexes; // 结点 index 的数组
    int *v_values; // 结点 value 的数组
    int *a_matrix; // 邻接矩阵
} Graph_store;
```

在本程序中，数据原子类型被定义为键值对 int-int 的形式。

#### 4. 2. 2 演示系统

演示系统包括用户操作界面和功能调用部分。

演示系统界面语言为英文，所有操作和提示语言均为英文。

用户操作界面输出可选的线性表操作，用户输入数字选择要进行的操作。在用户选择操作后，功能调用部分会显示函数的名称，参数，返回值和作用，系统提示用户输入参数。

功能调用部分将用户输入的有关信息传递给线性数据结构的操作函数进行调用，并对函数的返回值进行处理判断输出相应的提示信息。

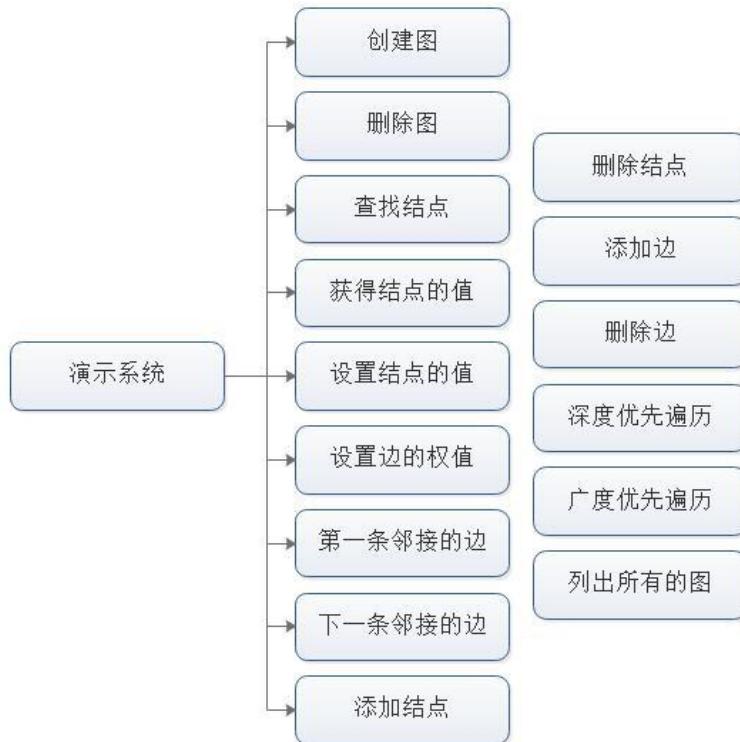


图 3-1 演示系统模块结构图

其中，功能 6 设置边的权值仅在图为有权图时才可生效。

整个演示系统的设计基本和实验一一致，接口使用相同接口，只是库函数的实现改变了。

#### 4.2.3 四种不同的图的区别

本次实验我实现了四种图，分别为无向图，无向网，有向图，有向网。网即为有权图。

在图的结构体中设计了一个 kind 变量储存图的类型，在每次操作前，通过判断结构体中的类型来实现对不同图的操作。

在算法上，有向和无向的区别在于，若图是无向图，则对于边的添加和删除都是成对的，每次执行添加或删除 i 到 j 的边时，同时也要执行添加或删除 j 到 i 的边的操作。而当删除一个结点时，也应该遍历其他所有结点，删除含有指向该结点的边。

而网和图的区别在于，其中的边可以带有权值，这个权值可以通过函数修改，同时也在邻接矩阵中反应出。

这样，当图的种类成为一个属性之后，四种图的各种操作便可以很方便的在一个函数中实现了，节省了很多的代码量。

#### 4.2.4 图运算实现算法

本部分说明算法时直接用函数原型代替。

```
1. int creat_graph(graph *G, int kind, int v_num, int a_num,  
int *v_indexs, int *v_values, int *a_matrix);
```

**功能：**创建一个图。

**算法实现：**根据输入的 `index` 数组和 `value` 数组执行插入函数创建 `v_num` 个结点，再根据输入的邻接矩阵执行插入边的函数来创建相应的边，创建成功最后返回 `OK`，否则返回 `ERROR`，并 `free` 掉所有已经申请的空间。

**时空效率分析：**由于存在遍历  $n^2$  次来创建边，算法的时间复杂度为  $O(n^2)$ ，同理需要的空间也是这么大，因此空间复杂度为  $O(n^2)$ 。

```
2. int destory_graph(graph *G);
```

**功能：**删除图。

**算法实现：**遍历并 `free` 掉所有的边和结点（事实上我们这里存储的结构就是一个二级链表结构），最后 `free` 掉自身，返回 `OK`。

**时空效率分析：**算法的时间复杂度为  $O(m+n)$ ，为  $m$  个结点， $n$  条边，同理空间复杂度为  $O(m+n)$ 。

```
3. int locate_vex(graph *G, int value);
```

**功能：**根据值查找结点。

**算法实现：**由于我们的存储结构就是二级链表，这里相当于链表的遍历查找操作，若查找到相应的结点，返回其值，否则返回 0。

**时空效率分析：**和链表的遍历查找操作一样，算法的时间复杂度为  $O(n)$ ，空间复杂度为  $O(1)$ 。

4. int get\_vex\_value(graph \*G, int index);

**功能:** 获得结点的值。

**算法实现:** 和上一个函数一样, 这里只不过是换成了根据值查找 `index` 而已, 若找到相应的结点, 则返回其 `index`, 否则返回 0。

**时空效率分析:** 同上, 算法的时间复杂度为  $O(n)$ , 空间复杂度为  $O(1)$ 。

5. int set\_vex\_value(graph \*G, int index, int value);

**功能:** 设置结点的值。

**算法实现:** 将同上, 也是查找操作, 若找到相应的结点, 那么其值设置为 `value`, 返回 OK, 否则返回 ERROR。

**时空效率分析:** 同上, 算法的时间复杂度为  $O(n)$ , 空间复杂度为  $O(1)$ 。

6. int set\_arc\_weight(graph \*G, int src\_index, int dst\_index, int weight);

**功能:** 设置边的权值 (仅在有权图中才生效)。

**算法实现:** 我们执行以下的步骤:

1. 将首先搜索源结点是否存在, 若不存在, 则返回 ERROR。

2. 若存在, 则遍历其指向的边, 若未发现相应的边, 则返回 ERROR。

3. 若找到相应的边, 将其权值设置为 `weight`。

4. 若该图是无向的, 还要再对对称边执行该函数。

**时空效率分析:** 算法的时间复杂度为  $O(n)$ , 空间复杂度为  $O(1)$ 。

7. vertex\_node\* first\_adj\_vex(graph \*G, int index);

**功能:** 初获得第一条邻接的边。

**算法实现:** 首先查找 `index` 结点是否存在, 若存在, 返回其 `first_arc` 即可, 否则返回 NULL。

**时空效率分析:** 算法的时间复杂度为查找所需的  $O(n)$ , 空间复杂度为  $O(1)$ 。

```
8. vertex_node* next_adj_vex(graph *G, int index, int now_index);
```

**功能:** 初始化二叉树。

**算法实现:** 同样查找 `index` 结点, 若不存在, 返回 NULL, 否则遍历该结点指向的边链表, 找到 `now_index` 的下一个结点的 `index`, 最后再根据 `index` 执行查找操作。

**时空效率分析:** 算法的时间复杂度为查找操作的  $O(n)$ , 空间复杂度为  $O(1)$ 。

```
9. int insert_vex(graph *G, int index, int value);
```

**功能:** 插入结点。

**算法实现:** 为了保证广度优先搜索中序号的一致性, 这里插入采用表尾插入。

**时空效率分析:** 算法的时间复杂度为遍历表的  $O(n)$ , 空间复杂度为  $O(1)$ 。

```
10. int delete_vex(graph *G, int index);
```

**功能:** 删除结点。

**算法实现:** 类似于二级链表的删除操作, 遍历查找 `index` 结点, 若找到, 删除其所有邻接的边, 然后删除自身结点。同时遍历所有边, 删除指向该结点的边, 最后返回 OK, 否则返回 ERROR。

**时空效率分析:** 算法的时间复杂度为  $O(m+n)$ , 为  $m$  个结点条边, 空间复杂度为  $O(1)$ 。

```
12. int insert_arc(graph *G, int src_vex, int dst_vex, int weight);
```

**功能:** 插入边。

**算法实现:** 首先搜索 `src_vex` 结点和 `dst_vex` 是否存在, 若不存在, 返回 ERROR, 否则在 `src_vex` 结点的边链表中插入一条指向 `dst_vex` 的边, 返回 OK。

注意如果图是无向的, 则还要插入对称的边, 再执行一遍函数。

**时空效率分析：** 算法的时间复杂度为查找的  $O(n)$ ，空间复杂度为  $O(1)$ 。

13. `int delete_arc(graph *G, int src_vex, int dst_vex);`

**功能：** 删除边。

**算法实现：** 首先搜索 `src_vex` 结点是否存在，若不存在，返回 ERROR，否则遍历 `src_vex` 结点的边链表并删除这条边，若这条边不存在则返回 ERROR，否则删除成功返回 OK。

注意如果图是无向的，则还要删除对称的边，再执行一遍函数。

**时空效率分析：** 算法的时间复杂度为查找的  $O(n)$ ，空间复杂度为  $O(1)$ 。

14. `int dfs_traverse(graph *G);`

**功能：** 深度优先遍历图。

**算法实现：** 首先设置一个数组 `visit` 作为标记数组，以标记结点是否被遍历过，数组的初始值均为 0，然后从第一个结点开始，对每个结点执行以下操作：

1. 打印该结点的值，并且把 `visit` 值赋值为 1
2. 若第一个邻接结点没有被访问过，则对其执行操作
3. 若其下一个邻接结点没有被访问过，则对其执行操作
4. .....

Fin. 若其最后一个邻接结点没有被访问过，则对其执行操作

**时空效率分析：** 算法的时间复杂度为  $O(m+n)$ ，为  $m$  个结点和  $n$  条边，空间复杂度为  $O(m)$ 。

15. `int bfs_traverse(graph *G);`

**功能：** 广度优先遍历图。

**算法实现：** 首先设置一个数组 `visit` 作为标记数组，以标记结点是否被遍历过，数组的初始值均为 0，然后使用一个辅助队列，对每个结点执行以下操作：

1. 若该结点没有被访问过，则该结点进入队列

2 若队列不为空，则队列首元素出列，打印该结点的值，并且把 `visit` 值赋值为 1

2. 若该结点的第一个邻接结点没有被访问过，则进入队列尾
3. 若该结点的下一个邻接结点没有被访问过，则进入队列尾
4. .....

Fin. 若该结点的最后一个邻接结点没有被访问过，则进入队列尾，跳转至步骤 2

**时空效率分析：** 算法的时间复杂度为  $O(m+n)$ ，为  $m$  个结点和  $n$  条边，空间复杂度为  $O(m)$ 。

#### 4.2.5 多图管理实现算法

由于多图的管理采用链式线性表模型，只涉及到查找操作，所以时间复杂度是遍历用的  $O(n)$ ，空间复杂度为  $O(1)$ 。

#### 4.2.6 文件存储实现算法

1. `void load_data(Graph_main *Main_G);`

**功能：** 数据读取。

**算法实现：** 打开文件，依次读取一个图存储结构体，根据结构体内的数据调用 `creat_graph` 来创建图，直到所有的图被创建完毕。

**时空效率分析：** 算法的时间复杂度为  $O(n * m^2)$ ，为创建  $n$  个二叉树，平均每个  $m$  个结点；同理空间复杂度也为  $O(n * m^2)$ 。

2. `void save_data(Graph_main *Main_G);`

**功能：** 数据保存。

**算法实现：** 打开文件，依次构造一个图存储结构体，遍历结点和边，将结果存入图存储结构体，然后将该结构体写入文件，直到所有的图被写入文件。

**时空效率分析：** 算法的时间复杂度为  $O(n * m^2)$ ，为遍历  $n$  个图，平均每个  $m$  个结点；同理空间复杂度也为  $O(n * m^2)$ 。

## 4.3 系统实现

### 4.3.1 实验环境

实验环境为 Arch linux 4.14.8-1，编译器为 gcc 版本 7.2.1，代码采用开源的编辑器 vscode 编写。由指定的 Makefile 来完成编译。

文件说明：

- \* my\_graph.h: 图库头文件
- \* my\_graph.c: 图库实现
- \* main.h: 演示系统头文件
- \* main.c: 演示系统实现
- \* mylibqueue.c: 自用队列库
- \* CMakeLists.txt: CMake 列表（用以生成完备的 Makefile）
- \* Makefile: 自动化编译命令

### 4.3.2 代码亮点

所有的代码采用 Google C/C++ 标准代码规范，函数库所有注释采用英文，符合生产规范。错误检查和提示全面，根据不同的错误会有不同的提示信息。由于在 Linux 环境下编程，所有的 API 接口命名改用 unix 环境编程的标准短横线命名模式。

### 4.3.3 操作演示

注：这里只展示四种图中的无向图操作。

界面展示：

```
panyue@Saltedfish:~/code/my_github/HUST-DataStructure-Labs/lab04$ master . ./lab04
+-----+
| Welcome to panyue's graph demo system!
| Here are some functions you can call:
|
| 1: creat_graph           2: destory_graph
| 3: locate_vex            4: get_vex_value
| 5: set_vex_value          6: set_arc_weight
| 7: first_adj_vex          8: next_adj_vex
| 9: insert_vex             10: delete_vex
| 11: insert_arc            12: delete_arc
| 13: dfs_traverse          14: bfs_traverse
| 15: ls_list                0: quit
|
| Enter the number of the function and see the usage and call it!
| Enter 0 to quit the demo system.
|
| Copyright (C) 2017 Yue Pan
| Github: zxc479773533
+-----+
Your choose: [
```

图 4-2 演示系统界面

接下来我们以如下的结构开始，来演示本系统的各项操作：

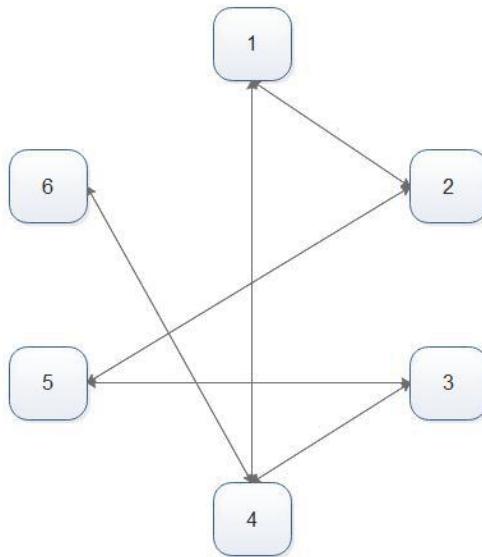


图 4-3 初始的图结构

我们首先向系统中存入一个无向图，结点和 ID 分别如图所示，为便于测试，每个结点中设置 index 和 value 一样。

## 1. 创建图

```
Your choose: 1
/*
 * Function Name: creat_graph
 * Module: Data structures
 * Parameter: graph &G, int v_num, int kind,
 * int a_num, int *v_indexs, int *v_values, int *a_matrix
 * Return: int(status)
 * Use: create a graph
 */

Format: id kind v_num a_num
Then, enter the graph id, kind, vertex num, arc num : 1 0 6 6
Please enter the node info (Format: index value):
1 1
2 2
3 3
4 4
5 5
6 6
please enter the adjacency matrix:
0 1 0 1 0 0
1 0 0 0 1 0
0 0 0 1 1 0
1 0 1 0 0 1
0 1 1 0 0 0
0 0 0 1 0 0
Create graph 1 succeed!
```

图 4-4 创建图

## 2. 查找结点 5

```
Your choose: 3
/*
 * Function Name: locate_vex
 * Module: Data structures
 * Parameter: graph *G, int value
 * Return: int(the index)
 * Use: find a vertex in graph
 */

Format: id value
Then, enter the graph id and vertex value: 1 5
The index is 5.
```

图 4-5 查找结点 5

## 3. 查找值为 6 的结点

```
Your choose: 4
/*
 * Function Name: get_vex_value
 * Module: Data structures
 * Parameter: graph *G, int index
 * Return: int(the index)
 * Use: find a value of a vertex
 */

Format: id index
Then, enter the graph id and vertex index: 1 6
The value of 6 is 6.
```

图 4-6 查找值为 6 的结点

#### 4. 设置结点的 6 的值为 99

```
Your choose: 5
/*
 * Function Name: set_vex_value
 * Module: Data structures
 * Parameter: graph &G, int index, int value
 * Return: int(the index)
 * Use: set a vertex's value
 */

Format: id index value
Then, enter the graph id, vertex index and value: 1 6 99
Set value 99 of index 6 succeed!
```

图 4-7 设置结点的值

#### 5. 再次查找结点 6，此时其值应该是 99

```
Your choose: 4
/*
 * Function Name: get_vex_value
 * Module: Data structures
 * Parameter: graph *G, int index
 * Return: int(the index)
 * Use: find a value of a vertex
 */

Format: id index
Then, enter the graph id and vertex index: 1 6
The value of 6 is 99.
```

图 4-8 查找结点 6

6. 查找结点 1 的第一个邻接结点

```
Your choose: 7
/*
 * Function Name: first_adj_vex
 * Module: Data structures
 * Parameter: graph &G, int index
 * Return: vertex_node*(the first vertex)
 * Use: get the fitst adjacency vertex
 */

Format: id index
Then, enter the graph id and index: 1 1
The first adjacency vertex of 1 is 2.
```

图 4-9 查找结点 1 的第一个邻接结点

7. 查找结点 1 的下一个邻接结点

```
Your choose: 8
/*
 * Function Name: next_adj_vex
 * Module: Data structures
 * Parameter: graph &G, int index, int now_index
 * Return: vertex_node*(the next vertex)
 * Use: get the next adjacency vertex
 */

Format: id index now_index
Then, enter the graph id, index and now index: 1 1 2
The nex adjacency vertex is 4.
```

图 4-10 查找结点 1 的下一个邻接结点

#### 8. 深度优先遍历

```
Your choose: 13
/*
 * Function Name: dfs_traverse
 * Module: Data structures
 * Parameter: graph &G
 * Return: int(statue)
 * Use: Depth_First Search traverse the graph
 */

Then, enter the graph id: 1
Graph: 1
DFS traverse:
Vertex num: 6
Index: 1, value 1
Index: 2, value 2
Index: 5, value 5
Index: 3, value 3
Index: 4, value 4
Index: 6, value 99
```

图 4-11 深度优先遍历

#### 9. 广度优先遍历

```
Your choose: 14
/*
 * Function Name: bfs_traverse
 * Module: Data structures
 * Parameter: graph &G
 * Return: int(statue)
 * Use: Broadht_First Search traverse the graph
 */

Then, enter the graph id: 1
Graph: 1
BFS traverse:
Vertex num: 6
Index: 1, value 1
Index: 2, value 2
Index: 4, value 4
Index: 5, value 5
Index: 3, value 3
Index: 6, value 99
```

图 4-12 广度优先遍历

## 10. 删除结点

```
Your choose: 10
/*
 * Function Name: delete_vex
 * Module: Data structures
 * Parameter: graph &G, int index
 * Return: int(statue)
 * Use: delete a vertex in graph
 */

Format: id index
Then, enter the graph id and index: 1 4
Delete vertex 4 succeed!
```

图 4-13 删除结点 4

删除结点 4 之后，此时的图为：

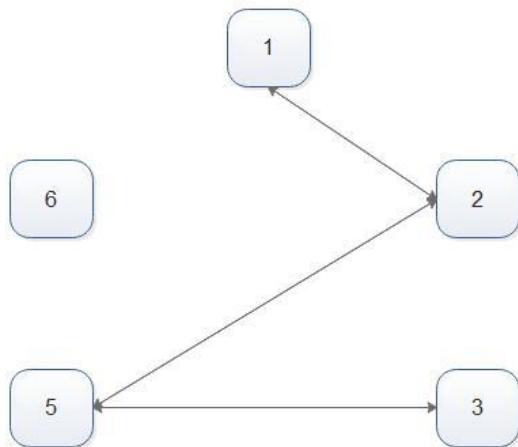


图 4-14 删除结点 4 之后的图

#### 11. 删除结点 4 之后，深度优先遍历图

```
Your choose: 13
/*
 * Function Name: dfs_traverse
 * Module: Data structures
 * Parameter: graph &G
 * Return: int(statue)
 * Use: Depth_First Search traverse the graph
 */

Then, enter the graph id: 1
Graph: 1
DFS traverse:
Vertex num: 5
Index: 1, value 1
Index: 2, value 2
Index: 5, value 5
Index: 3, value 3
Index: 6, value 6
```

图 4-15 深度优先遍历

#### 12. 插入新结点 99

```
Your choose: 9
/*
 * Function Name: insert_vex
 * Module: Data structures
 * Parameter: graph *G, int index, int value
 * Return: int(statue)
 * Use: insert a vertex in graph
 */

Format: id index value
Then, enter the graph id, index and value: 1 99 99
Insert vertex 99 succeed!
```

图 4-16 插入结点 99

### 13. 连接 1 和 99

```
Your choose: 11
/*
 * Function Name: insert_arc
 * Module: Data structures
 * Parameter: graph &G, int src_vex, int dst_vex, int weight
 * Return: int(statue)
 * Use: insert an arc in graph
 */

Format: id src_index dst_index weight
Then, enter the graph id, src and dst index, and weight: 1 1 99 1
Insert arc between 1 and 99 weight 1 succeed!
```

图 4-17 插入边连接 1 和 99

插入结点 99 并且和 1 连接后，此时的图为：

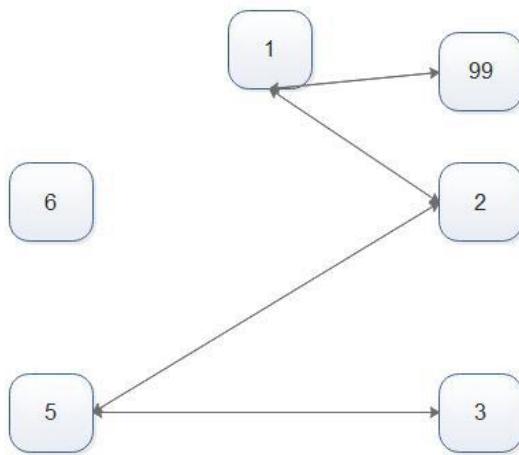


图 4-18 插入结点 99 之后的图

14. 插入结点 99 之后，广度优先遍历

```
Your choose: 14
/*
 * Function Name: bfs_traverse
 * Module: Data structures
 * Parameter: graph &G
 * Return: int(statue)
 * Use: Broadth_First Search traverse the graph
 */

Then, enter the graph id: 1
Graph: 1
BFS traverse:
Vertex num: 6
Index: 1, value 1
Index: 2, value 2
Index: 99, value 99
Index: 5, value 5
Index: 3, value 3
Index: 6, value 6
```

图 4-19 广度优先遍历

15. 正常退出系统

```

panyue@Saltedfish ~ /code/my_github/HUST-DataStructure-Labs/lab04 [master] ./lab04
+-----+
| Welcome to panyue's graph demo system!
| Here are some functions you can call:
|
| 1: creat_graph           2: destory_graph
| 3: locate_vex            4: get_vex_value
| 5: set_vex_value         6: set_arc_weight
| 7: first_adj_vex         8: next_adj_vex
| 9: insert_vex             10: delete_vex
| 11: insert_arc            12: delete_arc
| 13: dfs_traverse          14: bfs_traverse
| 15: ls_graph               0: quit
|
| Enter the number of the function and see the usage and call it!
| Enter 0 to quit the demo system.
|
| Copyright (C) 2017 Yue Pan
| Github: zxc479773533
+-----+
| Your choose: 0
| Thanks for using my demo system!
| panyue@Saltedfish ~ /code/my_github/HUST-DataStructure-Labs/lab04 [master]

```

图 4-20 正常退出系统

通过以上的演示证明，我们的图的功能是完整实现了的，可以演示要求的十五种图上的运算，并且做到了多图管理，实现了文件存储功能，不存在内存泄漏，完整的达到了实验的要求。

#### 4.3.4 系统健壮性/错误提示演示

我们仍然以如下的结构开始，来演示错误提示：

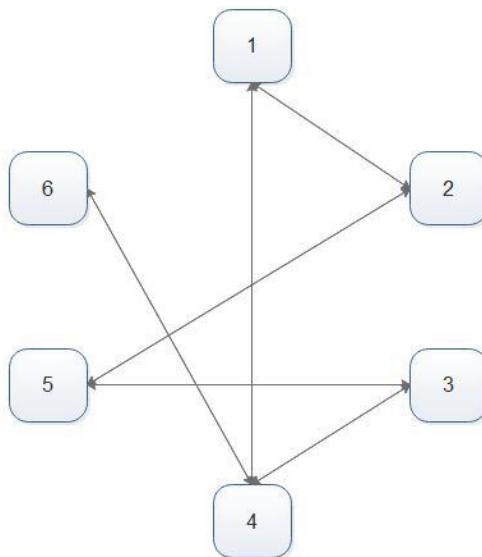


图 4-21 初始的图结构

系统中已经存入了一个，结点和 ID 分别如图所示。

### 1. 创建已存在的图

```
Your choose: 1
/*
 * Function Name: creat_graph
 * Module: Data structures
 * Parameter: graph &G, int v_num, int kind,
 * int a_num, int *v_indexs, int *v_values, int *a_matrix
 * Return: int(status)
 * Use: create a graph
 */

Format: id kind v_num a_num
Then, enter the graph id, kind, vertex num, arc num : 1 0 6 6
Create error, this graph is already exists!
```

图 4-22 创建已存在的图

### 2. 删除不存在的图

```
Your choose: 2
/*
 * Function Name: destory_graph
 * Module: Data structures
 * Parameter: graph &G
 * Return: int(status)
 * Use: destory a graph
 */

Then, enter the graph id: 5
Destory error, this graph is not exists!
```

图 4-23 删除不存在的图

### 3. 查找不存在的结点（根据键）

```
Your choose: 3
/*
 * Function Name: locate_vex
 * Module: Data structures
 * Parameter: graph *G, int value
 * Return: int(the index)
 * Use: find a vertex in graph
 */

Format: id value
Then, enter the graph id and vertex value: 1 99
The index is 0.
```

图 4-24 查找不存在的结点（根据键）

#### 4. 查找不存在的结点（根据值）

```
Your choose: 4
/*
 * Function Name: get_vex_value
 * Module: Data structures
 * Parameter: graph *G, int index
 * Return: int(the index)
 * Use: find a value of a vertex
 */

Format: id index
Then, enter the graph id and vertex index: 1 99
The value of 99 is 0.
```

图 4-25 查找不存在的结点（根据值）

#### 5. 设置不存在的结点的值

```
Your choose: 5
/*
 * Function Name: set_vex_value
 * Module: Data structures
 * Parameter: graph &G, int index, int value
 * Return: int(the index)
 * Use: set a vertex's value
 */

Format: id index value
Then, enter the graph id, vertex index and value: 1 99 99
Error, the index 99 is not exists!
```

图 4-26 设置不存在的结点的值

## 6. 设置不存在的边的权值

```
Your choose: 6
/*
 * Function Name: set_arc_weight
 * Module: Data structures
 * Parameter: graph *G, int src_index, int dst_index, int weight
 * Return: int(the index)
 * Use: set an arc's weight
 */

Format: id src_index dst_index weight
Then, enter the graph id, src and dst index, and arc weight: 1 1 6 1
Set error, this arc is not exists!
```

图 4-27 设置不存在的边的权值

## 7. 获取不存在的结点的第一条邻接边

```
Your choose: 7
/*
 * Function Name: first_adj_vex
 * Module: Data structures
 * Parameter: graph &G, int index
 * Return: vertex_node*(the first vertex)
 * Use: get the fitst adjacency vertex
 */

Format: id index
Then, enter the graph id and index: 1 99
Error, the required vertex is not exists!
```

图 4-28 获取不存在的结点的第一条邻接边

8. 已经是最后一条邻接边时获取下一条邻接边

```
Your choose: 8
/*
 * Function Name: next_adj_vex
 * Module: Data structures
 * Parameter: graph &G, int index, int now_index
 * Return: vertex_node*(the next vertex)
 * Use: get the next adjacency vertex
 */

Format: id index now_index
Then, enter the graph id, index and now index: 1 1 4
Error, the required vertex is not exists!
```

图 4-29 已经是最后一条邻接边时获取下一条邻接边

9. 向不存在的图中插入结点

```
Your choose: 9
/*
 * Function Name: insert_vex
 * Module: Data structures
 * Parameter: graph *G, int index, int value
 * Return: int(statue)
 * Use: insert a vertex in graph
 */

Format: id index value
Then, enter the graph id, index and value: 2 10 10
Error, this graph is not exists!
```

图 4-30 向不存在的图中插入结点

10. 删除不存在的结点

```
Your choose: 10
/*
 * Function Name: delete_vex
 * Module: Data structures
 * Parameter: graph &G, int index
 * Return: int(statue)
 * Use: delete a vertex in graph
 */

Format: id index
Then, enter the graph id and index: 1 99
Error, this vertex is not exists!
```

图 4-31 删除不存在的结点

### 11. 插入已存在的边

```
Your choose: 11
/*
 * Function Name: insert_arc
 * Module: Data structures
 * Parameter: graph &G, int src_vex, int dst_vex, int weight
 * Return: int(statue)
 * Use: insert an arc in graph
 */

Format: id src_index dst_index weight
Then, enter the graph id, src and dst index, and weight: 1 1 2 1
Error, please check your input!
```

图 4-32 插入已存在的边

### 12. 删 除不存在的边

```
Your choose: 12
/*
 * Function Name: delete_arc
 * Module: Data structures
 * Parameter: graph &G, int src_vex, int dst_vex
 * Return: int(statue)
 * Use: delete an arc in graph
 */

Format: id src_index dst_index
Then, enter the graph id, src and dst index: 1 1 6
Error, this arc is not exists!
```

图 4-33 删 除不存在的边

13. 深度优先遍历不存在的图

```
Your choose: 13
/*
 * Function Name: dfs_traverse
 * Module: Data structures
 * Parameter: graph &G
 * Return: int(statue)
 * Use: Depth_First Search traverse the graph
 */

Then, enter the graph id: 2
Error, this graph is not exists!
```

图 4-34 深度优先遍历不存在的图

14. 广度优先遍历不存在的图

```
Your choose: 14
/*
 * Function Name: bfs_traverse
 * Module: Data structures
 * Parameter: graph &G
 * Return: int(statue)
 * Use: Broadth_First Search traverse the graph
 */

Then, enter the graph id: 2
Error, this graph is not exists!
```

图 4-35 广度优先遍历不存在的图

通过以上的演示证明，我们的图演示系统的健壮性是足够的，可以对各种错误输入进行处理，输出相应的错误提示。

#### 4.4 实验小结

图的实验写起来比二叉树其实还要舒服，我大概更改了一下课本上对于图结

构体的设计，用链式结构存储使得结点个数的限制没有了。

吸取上一次实验的经验教训，这次仍然设计的单独的存储结构，大大减小了文件存储量，同时也减少了文件操作的复杂程度和代码量。

整个实验在 linux 环境下编程，所有的代码采用 Google C/C++ 标准代码规范，函数库所有注释采用英文，符合生产规范。错误检查和提示全面，根据不同的错误会有不同的提示信息。由于在 Linux 环境下编程，所有的 API 接口命名改用 unix 环境编程的标准短横线命名模式。这样规范自己有助于以后走向工作岗位之前的面试环节。

本系统完整的实现了实现的课程要求的全部功能，实现了多图管理和文件存储功能，系统健壮性良好，可以输出各种情况下的错误提示，并且在整个实验过程中避免了处处的内存泄漏，完整的达到了预期的效果。

整个四次数据结构实验结束了，对我个人的水平而言，其实还算都是些非常简单的东西，主要考察的是结构实现，对于算法问题到并没有怎么涉及，写起来也很快。

一点遗憾就是其实本来是想做个好看的图形界面拿来演示，倒是最后因为一些其他事情太忙没来得及在这上面花多少时间就鸽掉了。

期待课设，力争做个好项目。

## 参考文献

- [1] 严蔚敏等. 数据结构(C 语言版). 清华大学出版社
- [2] Data Structures and Algorithm Analysis in C. (美)Mark Allen Weiss. 机械工业出版社
- [3] Discrete Mathematics and Its Applications. (美)Kenneth H. Rosen. 机械工业出版社

指导教师评定意见

一、对实验报告的评语

二、对实验报告评分

评分项目 (分值)	程序内容 (36.8 分)	程序规范 (9.2 分)	报告内容 (36.8 分)	报告规范 (9.2 分)	考勤 (8 分)	逾期扣分	合计 (100 分)
得分							

## 附录 A 实验中用到的 mylibqueue 库说明和源代码

### A. 1 库接口函数说明

本队列函数库采用链式结构编写，为二叉树和图的广度优先遍历时使用。

接口函数如下：

`InitQueue`: 初始化队列

`DestroyQueue`: 删除队列

`QueueLength`: 求队列的长度

`EnQueue`: 元素入队列

`DeQueue`: 元素出队列

`IsEmpty`: 判断队列是否为空

### A. 2 源代码

File: `mylibqueue.c`

```
/*
 * My Queue library
 * AUTHOR: Yue Pan zxc479773533@gmail.com
 */

#ifndef __LIBQUEUE_H_
#define __LIBQUEUE_H_

#include <stdio.h>
#include <stdlib.h>
#include "my_graph.h"

// the data type in queue
typedef vertex_node* ElemType;

// queue node struct
typedef struct QNode {
    ElemType data;
    struct QNode *next;
} QNode;

// queue based in linklist
```

```
typedef struct {
    QNode* front;
    QNode* tail;
    int count;
} LinkQueue;

// input:LinkQueue
// output:status
int InitQueue(LinkQueue *Q) {
    Q->count = 0;
    Q->front = Q->tail = (QNode*)malloc(sizeof(QNode));
    if (Q->front == NULL)
        return -1;
    Q->front->next = NULL;
    return 0;
}

// input:LinkQueue
// output:status
int DestroyQueue(LinkQueue *Q) {
    while (Q->front != NULL) {
        Q->tail = Q->front->next;
        free(Q->front);
        Q->front = Q->tail;
    }
    return 0;
}

// input:LinkQueue
// output:the length
int QueueLength(LinkQueue *Q) {
    int count = 0;
    QNode *p = Q->front;
    while (p != NULL) {
        count++;
        p = p->next;
    }
    return count;
}

// input:LinkQueue, the data
// output:status
int EnQueue(LinkQueue *Q, ElemtType data) {
    QNode* p = (QNode*)malloc(sizeof(QNode));
```

```
if (p == NULL)
    return -1;
p->data = data;
p->next = NULL;
Q->tail->next = p;
Q->tail = Q->tail->next;
Q->count++;
return 0;
}

// input:LinkQueue
// output:the first
ElemType DeQueue(LinkQueue *Q) {
    if (Q->front == Q->tail)
        return NULL;
    QNode *p = Q->front->next;
    Q->front->next = p->next;
    if (p == Q->tail)
        Q->front = Q->tail;
    Q->count--;
    return p->data;
}

// input:LinkQueue
// output:status
int IsEmpty(LinkQueue *Q) {
    if (Q->front == Q->tail)
        return 1;
    else
        return 0;
}

#endif // __LIBQUEUE_H_
```

## 附录 B 基于顺序存储结构线性表实现的源代码

注：由于实验的编译文件 Makefile 使用 CMake 工具生成，故在这里附上 CMakeLists 文件

### B. 1 CMakeLists 编译文件

```
# copyright Pan Yue
project(lab01)

add_library(linear_list my_linearlist.c)

add_executable(lab01 main.c)

target_link_libraries(lab01 linear_list)
```

### B. 2 my\_linearlist.h

```
/*
 * AUTHOR: Yue Pan
 * GITHUB: zxc479773533
 * E-MAIL: zxc479773533@gmail.com
 */

#ifndef PY_LINEARLIST_H
#define PY_LINEARLIST_H

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

#define TRUE 1
#define FALSE 0

#define OK 0
#define ERROR 1

#define LIST_INIT_SIZE 100
#define LIST_INCREASEMENT 10
```

```
// the main struct of linear list
typedef struct linear_list {
    int id;
    int *data;
    int length;
    int size;
    struct linear_list *next;
} linear_list;

// the struct for managing the linear list
typedef struct Linear_list_main {
    int num;
    linear_list *head;
} Linear_list_main;

int init_list(linear_list *L);
/*
 * Function Name: init_list
 * Module: Data structures
 * Parameter: linear_list *L
 * Return: int(status)
 * Use: initial the linear list
 */

int destroy_list(linear_list *L);
/*
 * Function Name: destroy_list
 * Module: Data structures
 * Parameter: linear_list *L
 * Return: int(status)
 * Use: destroy the link list
 */

int clear_list(linear_list *L);
/*
 * Function Name: clear_list
 * Module: Data structures
 * Parameter: linear_list *L
 * Return: int(status)
 * Use: let the link list empty
 */
```

```
int is_list_empty(linear_list L);
/*
 * Function Name: is_list_empty
 * Module: Data structures
 * Parameter: linear_list L
 * Return: int(true or false)
 * Use: judge if the link list is empty
 */

int list_length(linear_list L);
/*
 * Function Name: list_length
 * Module: Data structures
 * Parameter: linear_list L
 * Return: int(the length)
 * Use: get the length of linear list
 */

int get_list_item(linear_list L, int order, int *elem);
/*
 * Function Name: get_list_item
 * Module: Data structures
 * Parameter: linear_list L, int order, int *elem
 * Return: int(status)
 * Use: get the ordered element of link list
 */

int locate_list_item(linear_list L, int ordered_elem);
/*
 * Function Name: locate_list_item
 * Module: Data structures
 * Parameter: linear_list L, int ordered_elem
 * Return: int(index)
 * Use: get the index of ordered item
 */

int piror_list_item(linear_list L, int elem, int *elem_pre);
/*
```

```
* Function Name: piror_list_item
* Module: Data structures
* Parameter: linear_list L, int elem, int *elem_pre
* Return: int(status)
* Use: get the ordered element's piror
*/
```

```
int next_list_item(linear_list L, int elem, int *elem_next);
/*
 * Function Name: next_list_item
 * Module: Data structures
 * Parameter: linear_list L, int elem, int *elem_next
 * Return: int(status)
 * Use: get the ordered element's next
*/
```

```
int list_insert(linear_list *L, int order, int elem);
/*
 * Function Name: list_insert
 * Module: Data structures
 * Parameter: linear_list *L, int order, int elem
 * Return: int(status)
 * Use: insert a element in the link list
*/
```

```
int list_delete(linear_list *L, int order, int *elem);
/*
 * Function Name: list_delete
 * Module: Data structures
 * Parameter: linear_list *L, int order, int *elem
 * Return: int(status)
 * Use: delete a element in the link list
*/
```

```
void print_list(linear_list L);
/*
 * Function Name: print_list
 * Module: Data structures
 * Parameter: linear_list L
 * Return: None
```

```
* Use: print the elements of the linklist to the payload
*/
```

```
#endif // !PY_LINEARLIST_H
```

### B.3 my\_linearlist.c

```
/*
 * AUTHOR: Yue Pan
 * GITHUB: zxc479773533
 * E-MAIL: zxc479773533@gmail.com
 */

#include "my_linearlist.h"

int init_list(linear_list *L) {
/*
 * Function Name: init_list
 * Module: Data structures
 * Parameter: linear_list *L
 * Return: int(status)
 * Use: initial the linear list
 */

    memset(L, 0, sizeof(linear_list));
    L->data = (int *)malloc(sizeof(int) * LIST_INIT_SIZE);
    // not enough memory
    if (L->data == NULL)
        return ERROR;
    L->length = 0;
    L->size = LIST_INIT_SIZE;
    return OK;
}

int destroy_list(linear_list *L) {
/*
 * Function Name: destroy_list
 * Module: Data structures
 * Parameter: linear_list *L
 * Return: int(status)
 * Use: destroy the link list
 */
```

```
// directly free the L->data
if (L->data != NULL)
    free(L->data);
memset(L, 0, sizeof(linear_list));
return OK;
}

int clear_list(linear_list *L) {
/*
 * Function Name: clear_list
 * Module: Data structures
 * Parameter: linear_list *L
 * Return: int(status)
 * Use: let the link list empty
*/
if (L->data == NULL)
    return ERROR;
// set the whole memory to 0
memset(L->data, 0, sizeof(int) * L->size);
L->length = 0;
L->size = LIST_INIT_SIZE;
return OK;
}

int is_list_empty(linear_list L) {
/*
 * Function Name: is_list_empty
 * Module: Data structures
 * Parameter: linear_list L
 * Return: int(true or false)
 * Use: judge if the link list is empty
*/
if (L.length == 0)
    return TRUE;
else
    return FALSE;
}

int list_length(linear_list L) {
/*
 * Function Name: list_length
 * Module: Data structures
*/
```

```
* Parameter: linear_list L
* Return: int(the length)
* Use: get the length of link list
*/
return L.length;
}

int get_list_item(linear_list L, int order, int *elem) {
/*
 * Function Name: get_list_item
 * Module: Data structures
 * Parameter: linear_list L, int order, int *elem
 * Return: int(status)
 * Use: get the ordered element of link list
*/
int index = 0;
// condition check
if (order > L.length || order <= 0 || L.length == 0 || L.data == NULL)
    return ERROR;
*elem = *(L.data + order - 1);
return OK;
}

int locate_list_item(linear_list L, int ordered_elem) {
/*
 * Function Name: locate_list_item
 * Module: Data structures
 * Parameter: linear_list L, int ordered_elem
 * Return: int(index)
 * Use: locate the ordered item
*/
int index = 0;
// traverse and find the ordered elem
for (index = 0; index < L.length; index++) {
    if (*(L.data + index) == ordered_elem)
        break;
}
if (index == L.length)
    return 0;
else
```

```
    return index + 1;
}

int piror_list_item(linear_list L, int elem, int *elem_pre) {
/*
 * Function Name: piror_list_item
 * Module: Data structures
 * Parameter: linear_list L, int elem, int *elem_pre
 * Return: int(status)
 * Use: get the ordered element's piror
 */

    int index;
    // traverse and find the ordered elem
    for (index = 1; index < L.length; index++) {
        if (*(L.data + index) == elem) {
            *elem_pre = *(L.data + index - 1);
            break;
        }
    }
    if (index == L.length)
        return ERROR;
    else
        return OK;
}

int next_list_item(linear_list L, int elem, int *elem_next) {
/*
 * Function Name: next_list_item
 * Module: Data structures
 * Parameter: linear_list L, int elem, int *elem_next
 * Return: int(status)
 * Use: get the ordered element's next
 */

    int index;
    // traverse and find the ordered elem
    for (index = 0; index < L.length - 1; index++) {
        if (*(L.data + index) == elem) {
            *elem_next = *(L.data + index + 1);
            break;
        }
    }
}
```

```

if (index == L.length - 1)
    return ERROR;
else
    return OK;
}

int list_insert(linear_list *L, int order, int elem) {
/*
 * Function Name: list_insert
 * Module: Data structures
 * Parameter: linear_list *L, int order, int elem
 * Return: int(status)
 * Use: insert a element in the link list
 */

// condition check
if (order > L->size || order <= 0 || L->size == 0 || L->data == NULL)
    return ERROR;
// not enouth storage, alloc new memory
if (L->length == L->size) {
    int *new_base = (int*)realloc(L->data, (L->size + LIST_INCREASEMENT) * sizeof(int));
    // not enouth memory
    if (new_base == NULL)
        return ERROR;
    L->data = new_base;
    L->size += LIST_INCREASEMENT;
}
int index;
// move the elements after the ordered position
for (index = L->size; index >= order; index--)
    *(L->data + index) = *(L->data + index - 1);
*(L->data + order - 1) = elem;
L->length++;
return OK;
}

int list_delete(linear_list *L, int order, int *elem) {
/*
 * Function Name: list_delete
 * Module: Data structures
 * Parameter: linear_list *L, int order, int *elem
 * Return: int(status)
 */

```

```

* Use: delete a element in the link list
*/
// condition check
if (order > L->length || order <= 0 || L->size == 0 || L->data
== NULL)
    return ERROR;
int index;
*elem = *(L->data + order - 1);
// move the elements after the ordered position
for (index = order - 1; index < L->length - 1; index++)
    *(L->data + index) = *(L->data + index + 1);
L->length--;
return OK;
}

void print_list(linear_list L) {
/*
 * Function Name: print_list
 * Module: Data structures
 * Parameter: linear_list L
 * Return: int(status)
 * Use: print the elements of the linklist to the payload
 */
int index;
for (index = 0; index < L.length; index++) {
    printf("Index: %d, Data: %d\n", index + 1, *(L.data + index));
}
}

```

#### B.4 main.h

```

/*
 * AUTHOR: Yue Pan
 * GITHUB: zxc479773533
 * E-MAIL: zxc479773533@gmail.com
*/
#ifndef PY_LINEARLIST_MAIN_H_
#define PY_LINEARLIST_MAIN_H_

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

```

```
#include <stdlib.h>
#include "my_linearlist.c"

void print_menu(void);
/*
 * Function Name: print_menu
 * Module: Main control
 * Parameter: None
 * Return: None
 * Use: print the menu
 */

void load_data(Linear_list_main *Main_L);
/*
 * Function Name: load_data
 * Module: Main control
 * Parameter: Linear_list_main *Main_L
 * Return: None
 * Use: load linear list from database
 */

void save_data(Linear_list_main *Main_L);
/*
 * Function Name: save_data
 * Module: Main control
 * Parameter: Linear_list_main *Main_L
 * Return: None
 * Use: save linear list from database
 */

#endif // !PY_LINEARLIST_MAIN_H_
```

## B.5 main.c

```
/*
 * AUTHOR: Yue Pan
 * GITHUB: zxc479773533
 * E-MAIL: zxc479773533@gmail.com
 */

#include "main.h"
```

```
void print_menu(void) {
/*
 * Function Name: print_menu
 * Module: Main control
 * Parameter: None
 * Return: None
 * Use: print the menu
*/
printf("+\n-----+\n");
printf("|           Welcome to panyue's linear list demo system!\n|\n");
printf("|           Here are some functions you can call:\n|\n");
printf("|           1: init_list          2: destroy_list\n|\n");
printf("|           3: clear_list        4: is_list_empty\n|\n");
printf("|           5: list_length       6: get_list_item\n|\n");
printf("|           7: locate_list_item  8:\npiror_list_item    |\nprintf("|           9: next_list_item    10: list_insert\n|\n");
printf("|           11: list_delete      12: print_list\n|\n");
printf("|           13: ls_list          0: quit\n|\n");
printf("|           Enter the number of the function and see the usage and\n");
call it!    |\n");
printf("|Enter    0    to    quit    the    demo    system.\n|\n");
printf("|           |\n");
printf("|           |\n");
printf("|Copyrite   (C)    2017    Yue    Pan\n|\n");
printf("|Github:    zxc479773533
```

```
| \n");  
  
printf("-----+-----\n");  
printf("\n");  
printf("Your choose: ");  
}  
  
void load_data(Linear_list_main *Main_L) {  
/*  
 * Function Name: load_data  
 * Module: Main control  
 * Parameter: Linear_list_main *Main_L  
 * Return: None  
 * Use: load linear list from database  
 */  
  
// open file  
FILE *fp = fopen("linearlistdb", "rb");  
if (fp == NULL)  
    return;  
  
int size = 0xff;  
int count = 0;  
linear_list my_list, *L = &my_list;  
L->next = NULL;  
while (1) {  
    linear_list *tmp_L = (linear_list *)malloc(sizeof(linear_list));  
    size = fread(tmp_L, sizeof(linear_list), 1, fp);  
    if (size == 0)  
        break;  
    count++;  
    tmp_L->data = (int *)malloc(sizeof(int) * tmp_L->size);  
    size = fread(tmp_L->data, sizeof(int) * tmp_L->size, 1, fp);  
    L->next = tmp_L;  
    L = L->next;  
}  
  
Main_L->head = my_list.next;  
Main_L->num = count;  
  
// close file  
fclose(fp);
```

```
}

void save_data(Linear_list_main *Main_L) {
/*
 * Function Name: save_data
 * Module: Main control
 * Parameter: Linear_list_main *Main_L
 * Return: None
 * Use: save linear list from database
 */

// open data files
FILE *fp = fopen("linearlistdb", "wb");
if (fp == NULL)
    return;

linear_list *L = Main_L->head;
while (L != NULL) {
    fwrite(L, sizeof(linear_list), 1, fp);
    fwrite(L->data, sizeof(int) * L->size, 1, fp);
    L = L->next;
}

// close file
fclose(fp);
}

int main(void) {

// a variable to save the function choose
int function_choose = 0xff;

while (function_choose != 0) {

    // print the menu
    print_menu();
    scanf("%d", &function_choose);

    // load the data
    Linear_list_main Main_L;
    Main_L.head = NULL;
    Main_L.num = 0;
    load_data(&Main_L);

}
```

```

// some variable to save the parameters
int id, order, elem, index;
linear_list *L = NULL;

switch (function_choose) {

    case 1:
        printf("/*\n");
        printf(" * Function Name: init_list\n");
        printf(" * Module: Data structures\n");
        printf(" * Parameter: linear_list *L\n");
        printf(" * Return: int(status)\n");
        printf(" * Use: initial the linear list\n");
        printf(" */\n");
        printf("\n");
        printf("Then, enter the list id: ");

        scanf("%d", &id);
        L = Main_L.head;
        while (L != NULL) {
            if (L->id == id)
                break;
            L = L->next;
        }
        if (L != NULL) {
            printf("Initial error, this linear list is already
exists!\n");
        }
        else {
            linear_list *new_L = (linear_list
*)malloc(sizeof(linear_list));
            if (init_list(new_L) == OK) {
                printf("Initial list %d succeed!\n", id);
                new_L->id = id;
                new_L->next = Main_L.head;
                Main_L.head = new_L;
            }
            else {
                printf("Initial error, the memory is insufficient!\n");
            }
        }
        printf("\n");
        break;
}

```

```
case 2:  
    printf("/*\n");  
    printf(" * Function Name: destroy_list\n");  
    printf(" * Module: Data structures\n");  
    printf(" * Parameter: linear_list *L\n");  
    printf(" * Return: int(status)\n");  
    printf(" * Use: destroy the link list\n");  
    printf(" */\n");  
    printf("\n");  
    printf("Then, enter the list id: ");  
  
    scanf("%d", &id);  
    L = Main_L.head;  
    if (L->id == id) {  
        Main_L.head = Main_L.head->next;  
        destroy_list(L);  
        printf("Linear list %d has been deleted!\n", id);  
        printf("\n");  
        break;  
    }  
    else {  
        while (L->next != NULL) {  
            if (L->next->id == id)  
                break;  
            L = L->next;  
        }  
    }  
    if (L->next == NULL) {  
        printf("Destroy error, this linear list is not  
exists!\n");  
    }  
    else {  
        linear_list *to_be_del = L->next;  
        L->next = L->next->next;  
        destroy_list(to_be_del);  
        printf("Linear list %d has been deleted!\n", id);  
    }  
  
    printf("\n");  
    break;  
  
case 3:  
    printf("/*\n");
```

```

printf(" * Function Name: clear_list\n");
printf(" * Module: Data structures\n");
printf(" * Parameter: linear_list *L\n");
printf(" * Return: int(status)\n");
printf(" * Use: let the link list empty\n");
printf(" */\n");
printf("\n");
printf("Then, enter the list id: ");

scanf("%d", &id);
L = Main_L.head;
while (L != NULL) {
    if (L->id == id)
        break;
    L = L->next;
}
if (L == NULL) {
    printf("Clear error, this linear list is not exists!\n");
}
else {
    if (clear_list(L) == OK) {
        printf("Linear list %d has been cleared!\n", id);
    }
    else {
        printf("Clear error, this linear is already empty!\n");
    }
}

printf("\n");
break;

case 4:
printf("/*\n");
printf(" * Function Name: is_list_empty\n");
printf(" * Module: Data structures\n");
printf(" * Parameter: linear_list L\n");
printf(" * Return: int(true or false)\n");
printf(" * Use: judge if the link list is empty\n");
printf(" */\n");
printf("\n");
printf("Then, enter the list id: ");

scanf("%d", &id);
L = Main_L.head;

```

```

while (L != NULL) {
    if (L->id == id)
        break;
    L = L->next;
}
if (L == NULL) {
    printf("Error, this linear list is not exists!\n");
}
else {
    if (is_list_empty(*L) == TRUE) {
        printf("Linear list %d is empty!\n", id);
    }
    else {
        printf("Linear list %d is not empty!\n", id);
    }
}

printf("\n");
break;

case 5:
printf("/*\n");
printf(" * Function Name: list_length\n");
printf(" * Module: Data structures\n");
printf(" * Parameter: linear_list L\n");
printf(" * Return: int(the length)\n");
printf(" * Use: get the length of linear list\n");
printf(" */\n");
printf("\n");
printf("Then, enter the list id: ");

scanf("%d", &id);
L = Main_L.head;
while (L != NULL) {
    if (L->id == id)
        break;
    L = L->next;
}
if (L == NULL) {
    printf("Error, this linear list is not exists!\n");
}
else {
    printf("The length of linear list %d is %d.\n", id,
list_length(*L));
}

```

```

    }

    printf("\n");
    break;

case 6:
    printf("/*\n");
    printf(" * Function Name: get_list_item\n");
    printf(" * Module: Data structures\n");
    printf(" * Parameter: linear_list L, int order, int
*elem\n");
    printf(" * Return: int(status)\n");
    printf(" * Use: get the ordered element of link list\n");
    printf(" */\n");
    printf("\n");
    printf("Format: id index\n");
    printf("Then, enter the list id and index: ");

scanf("%d %d", &id, &order);
L = Main_L.head;
while (L != NULL) {
    if (L->id == id)
        break;
    L = L->next;
}
if (L == NULL) {
    printf("Error, this linear list is not exists!\n");
}
else {
    if (get_list_item(*L, order, &elem) == OK) {
        printf("The element is %d.\n", elem);
    }
    else {
        printf("Get element error! Please check your
input.\n");
    }
}

printf("\n");
break;

case 7:
printf("/*\n");
printf(" * Function Name: locate_list_item\n");

```

```

printf(" * Module: Data structures\n");
printf(" * Parameter: linear_list L, int ordered_elem\n");
printf(" * Return: int(index)\n");
printf(" * Use: get the index of ordered item\n");
printf(" */\n");
printf("\n");
printf("                                     Format: id
element\n");
printf("Then, enter the list id and element: ");

scanf("%d %d", &id, &elem);
L = Main_L.head;
while (L != NULL) {
    if (L->id == id)
        break;
    L = L->next;
}
if (L == NULL) {
    printf("Error, this linear list is not exists!\n");
}
else {
    index = locate_list_item(*L, elem);
    printf("The index of the %d, is %d.\n", elem, index);
}

printf("\n");
break;

case 8:
printf("/*\n");
printf(" * Function Name: piror_list_item\n");
printf(" * Module: Data structures\n");
printf(" * Parameter: linear_list L, int elem, int
*elem_pre\n");
printf(" * Return: int(status)\n");
printf(" * Use: get the ordered element's piror\n");
printf(" */\n");
printf("\n");
printf("                                     Format: id
element\n");
printf("Then, enter the list id and element: ");

scanf("%d %d", &id, &elem);
L = Main_L.head;

```

```

while (L != NULL) {
    if (L->id == id)
        break;
    L = L->next;
}
if (L == NULL) {
    printf("Error, this linear list is not exists!\n");
}
else {
    int elem_pre;
    if (piror_list_item(*L, elem, &elem_pre) == OK) {
        printf("The piror element of %d is %d.\n", elem,
elem_pre);
    }
    else {
        printf("Error, the piror of %d is not exists!\n", elem);
    }
}

printf("\n");
break;

case 9:
printf("/*\n");
printf(" * Function Name: next_list_item\n");
printf(" * Module: Data structures\n");
printf(" * Parameter: linear_list L, int elem, int
*elem_next\n");
printf(" * Return: int(status)\n");
printf(" * Use: get the ordered element's next\n");
printf(" */\n");
printf("\n");
printf("                                     Format: id
element\n");
printf("Then, enter the list id and element: ");

scanf("%d %d", &id, &elem);
L = Main_L.head;
while (L != NULL) {
    if (L->id == id)
        break;
    L = L->next;
}
if (L == NULL) {

```

```

        printf("Error, this linear list is not exists!\n");
    }
    else {
        int elem_next;
        if (next_list_item(*L, elem, &elem_next) == OK) {
            printf("The next element of %d is %d.\n", elem,
elem_next);
        }
        else {
            printf("Error, the next of %d is not exists!\n", elem);
        }
    }

    printf("\n");
    break;

case 10:
    printf("/*\n");
    printf(" * Function Name: list_insert\n");
    printf(" * Module: Data structures\n");
    printf(" * Parameter: linear_list *L, int order, int
elem\n");
    printf(" * Return: int(status)\n");
    printf(" * Use: insert a element in the link list\n");
    printf(" */\n");
    printf("\n");
    printf("                                     Format: id index
element\n");
    printf("Then, enter the list id, index and element: ");

    scanf("%d %d %d", &id, &order, &elem);
    L = Main_L.head;
    while (L != NULL) {
        if (L->id == id)
            break;
        L = L->next;
    }
    if (L == NULL) {
        printf("Error, this linear list is not exists!\n");
    }
    else {
        if (list_insert(L, order, elem) == OK) {
            printf("Insert %d succeed!\n", elem);
        }
    }
}

```

```
    else {
        printf("Insert failed, please check your input!\n");
    }
}

printf("\n");
break;

case 11:
printf("/*\n");
printf(" * Function Name: list_delete\n");
printf(" * Module: Data structures\n");
printf(" * Parameter: linear_list *L, int order, int
*elem\n");
printf(" * Return: int(status)\n");
printf(" * Use: delete a element in the link list\n");
printf(" */\n");
printf("\n");
printf("                         Format: id index\n");
printf("Then, enter the list id and index: ");

scanf("%d %d", &id, &order);
L = Main_L.head;
while (L != NULL) {
    if (L->id == id)
        break;
    L = L->next;
}
if (L == NULL) {
    printf("Error, this linear list is not exists!\n");
}
else {
    if (list_delete(L, order, &elem) == OK) {
        printf("Delete %d succeed!\n", elem);
    }
    else {
        printf("Delete failed, please check your input!\n");
    }
}

printf("\n");
break;

case 12:
```

```
printf("/*\n");
printf(" * Function Name: print_list\n");
printf(" * Module: Data structures\n");
printf(" * Parameter: linear_list L, char *payload\n");
printf(" * Return: None\n");
printf(" * Use: print the elements of the linklist to the
payload\n");
printf(" */\n");
printf("\n");
printf("Then, enter the list id: ");

scanf("%d", &id);
L = Main_L.head;
while (L != NULL) {
    if (L->id == id)
        break;
    L = L->next;
}
if (L == NULL) {
    printf("Error, this linear list is not exists!\n");
}
else {
    printf("The elements in linear list %d are:\n", id);
    print_list(*L);
}

printf("\n");
break;

case 13:
L = Main_L.head;
while(L != NULL) {
    printf("ID: %d\n", L->id);
    L = L->next;
}

printf("\n");
break;

case 0:
printf("Thanks for using my demo system!\n");
break;

default:
```

```
    printf("You entered the wrong num, please re-enter the
num.\n");
    printf("\n");
    break;

}

save_data(&Main_L);
}

return 0;
}
```

## 附录 C 基于链式存储结构线性表实现的源代码

注：由于实验的编译文件 Makefile 使用 CMake 工具生成，故在这里附上 CMakeLists 文件

### C. 1 CMakeLists

```
# copyright Pan Yue
project(lab02)

add_library(link_list my_linklist.c)

add_executable(lab02 main.c)

target_link_libraries(lab02 link_list)
```

### C. 2 my\_linklist.h

```
/*
 * AUTHOR: Yue Pan
 * GITHUB: zxc479773533
 * E-MAIL: zxc479773533@gmail.com
 */

#ifndef PY_LINKLIST_H
#define PY_LINKLIST_H

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

#define TRUE 1
#define FALSE 0

#define OK 0
#define ERROR 1

// the struct of link node
typedef struct link_node {
    int data;
    struct link_node *next;
}
```

```
    } link_node;

    // the main struct of link list
    typedef struct link_list {
        int id;
        int length;
        struct link_node *head;
        struct link_list *next;
    } link_list;

    // the struct for managing the link list
    typedef struct Link_list_main {
        int num;
        link_list *head;
    } Link_list_main;

int init_list(link_list *L);
/*
 * Function Name: init_list
 * Module: Data structures
 * Parameter: link_list *L
 * Return: int(status)
 * Use: initial the link list
 */

int destroy_list(link_list *L);
/*
 * Function Name: destroy_list
 * Module: Data structures
 * Parameter: link_list *L
 * Return: int(status)
 * Use: destroy the link list
 */

int clear_list(link_list *L);
/*
 * Function Name: clear_list
 * Module: Data structures
 * Parameter: link_list *L
 * Return: int(status)
 * Use: let the link list empty
```

```
/*
int is_list_empty(link_list L);
/*
 * Function Name: is_list_empty
 * Module: Data structures
 * Parameter: link_list L
 * Return: int(true or false)
 * Use: judge if the link list is empty
 */

int list_length(link_list L);
/*
 * Function Name: list_length
 * Module: Data structures
 * Parameter: link_list L
 * Return: int(the length)
 * Use: get the length of link list
 */

int get_list_item(link_list L, int order, int *elem);
/*
 * Function Name: get_list_item
 * Module: Data structures
 * Parameter: link_list L, int order, int *elem
 * Return: int(status)
 * Use: get the ordered element of link list
 */

int locate_list_item(link_list L, int ordered_elem);
/*
 * Function Name: locate_list_item
 * Module: Data structures
 * Parameter: link_list L, int ordered_elem
 * Return: int(index)
 * Use: get the index of ordered item
 */

int piror_list_item(link_list L, int elem, int *elem_pir);
```

```
/*
 * Function Name: piror_list_item
 * Module: Data structures
 * Parameter: link_list L, int elem, int *elem_pir
 * Return: int(status)
 * Use: get the ordered element's piror
 */

int next_list_item(link_list L, int elem, int *elem_next);
/*
 * Function Name: next_list_item
 * Module: Data structures
 * Parameter: link_list L, int elem, int *elem_next
 * Return: int(status)
 * Use: get the ordered element's next
 */

int list_insert(link_list *L, int order, int elem);
/*
 * Function Name: list_insert
 * Module: Data structures
 * Parameter: link_list *L, int order, int elem
 * Return: int(status)
 * Use: insert a element in the link list
 */

int list_delete(link_list *L, int order, int *elem);
/*
 * Function Name: list_delete
 * Module: Data structures
 * Parameter: link_list *L, int order, int *elem
 * Return: int(status)
 * Use: delete a element in the link list
 */

void print_list(link_list L);
/*
 * Function Name: print_list
 * Module: Data structures
 * Parameter: link_list L
```

```
* Return: None
* Use: print the elements of the linklist to the payload
*/
```

```
#endif // !PY_LINKLIST_H
```

### C.3 my\_linklist.c

```
/*
 * AUTHOR: Yue Pan
 * GITHUB: zxc479773533
 * E-MAIL: zxc479773533@gmail.com
 */

#include "my_linklist.h"

int init_list(link_list *L) {
/*
 * Function Name: init_list
 * Module: Data structures
 * Parameter: link_list *L
 * Return: int(status)
 * Use: initial the link list
 */
    L->length = 0;
    L->head = NULL;
    return OK;
}

int destroy_list(link_list *L) {
/*
 * Function Name: destroy_list
 * Module: Data structures
 * Parameter: link_list *L
 * Return: int(status)
 * Use: destroy the link list
 */
    link_node *a_node = L->head;
    // traverse and free nodes
    while (a_node != NULL) {
        link_node *tmp = a_node;
```

```
a_node = a_node->next;
free(tmp);
}
free(L);
return OK;
}

int clear_list(link_list *L) {
/*
 * Function Name: clear_list
 * Module: Data structures
 * Parameter: link_list *L
 * Return: int(status)
 * Use: let the link list empty
*/
link_node *a_node = L->head;
if (a_node == NULL)
    return ERROR;
// traverse and free node
while (a_node != NULL) {
    link_node *tmp = a_node;
    a_node = a_node->next;
    free(tmp);
}
L->length = 0;
L->head = NULL;
return OK;
}

int is_list_empty(link_list L) {
/*
 * Function Name: is_list_empty
 * Module: Data structures
 * Parameter: link_list L
 * Return: int(true or false)
 * Use: judge if the link list is empty
*/
if (L.length == 0)
    return TRUE;
else
    return FALSE;
}
```

```
int list_length(link_list L) {
/*
 * Function Name: list_length
 * Module: Data structures
 * Parameter: link_list L
 * Return: int(the length)
 * Use: get the length of link list
 */

    return L.length;
}

int get_list_item(link_list L, int order, int *elem) {
/*
 * Function Name: get_list_item
 * Module: Data structures
 * Parameter: link_list L, int order, int *elem
 * Return: int(status)
 * Use: get the ordered element of link list
 */

    int index = 0;
    link_node *a_node = L.head;
    // traverse and find the ordered node
    while (a_node != NULL) {
        index++;
        if (index == order) {
            *elem = a_node->data;
            break;
        }
        a_node = a_node->next;
    }
    if (a_node == NULL)
        return ERROR;
    else
        return OK;
}

int locate_list_item(link_list L, int ordered_elem) {
/*
 * Function Name: locate_list_item
 * Module: Data structures
 * Parameter: link_list L, int ordered_elem
 *
```

```

* Return: int(index)
* Use: locate the ordered item
*/
int index = 0;
link_node *a_node = L.head;
// traverse and find the ordered node
while (a_node != NULL) {
    index++;
    if (a_node->data == ordered_elem)
        break;
    a_node = a_node->next;
}
if (a_node != NULL)
    return index;
else
    return 0;
}

int piror_list_item(link_list L, int elem, int *elem_pir) {
/*
 * Function Name: piror_list_item
 * Module: Data structures
 * Parameter: link_list L, int elem, int *elem_pir
 * Return: int(status)
 * Use: get the ordered element's piror
 */
link_node *piror_node = L.head;
link_node *a_node = piror_node->next;
// traverse and find the order node
while (a_node != NULL) {
    if (a_node->data == elem) {
        *elem_pir = piror_node->data;
        break;
    }
    piror_node = piror_node->next;
    a_node = piror_node->next;
}
if (a_node != NULL)
    return OK;
else
    return ERROR;
}

```

```
int next_list_item(link_list L, int elem, int *elem_next) {
/*
 * Function Name: next_list_item
 * Module: Data structures
 * Parameter: link_list L, int elem, int *elem_next
 * Return: int(status)
 * Use: get the ordered element's next
*/
link_node *a_node = L.head;
link_node *next_node = a_node->next;
// traverse and find the order node
while (next_node != NULL) {
    if (a_node->data == elem) {
        *elem_next = next_node->data;
        break;
    }
    a_node = a_node->next;
    next_node = a_node->next;
}
if (next_node != NULL)
    return OK;
else
    return ERROR;
}

int list_insert(link_list *L, int order, int elem) {
/*
 * Function Name: list_insert
 * Module: Data structures
 * Parameter: link_list *L, int order, int elem
 * Return: int(status)
 * Use: insert a element in the link list
*/
int index = 0;
link_node *a_node = L->head;
// the first node should be discussed separately
if (order == 1) {
    link_node *new_node = (link_node*)malloc(sizeof(link_node));
    new_node->data = elem;
    new_node->next = L->head;
    L->head = new_node;
```

```

    L->length++;
    return OK;
}
// start find the ordered node from the 2nd node
while (a_node != NULL) {
    index++;
    if (index == order - 1) {
        link_node *new_node = (link_node*)malloc(sizeof(link_node));
        new_node->data = elem;
        new_node->next = a_node->next;
        a_node->next = new_node;
        L->length++;
        break;
    }
    a_node = a_node->next;
}
if (a_node != NULL)
    return OK;
else
    return ERROR;
}

int list_delete(link_list *L, int order, int *elem) {
/*
 * Function Name: list_delete
 * Module: Data structures
 * Parameter: link_list *L, int order, int *elem
 * Return: int(status)
 * Use: delete a element in the link list
 */
// condition check
if (L->length == 0 || L->head == NULL)
    return ERROR;
int index = 0;
link_node *a_node = L->head;
// the first node should be discussed separately
if (order == 1) {
    L->head = a_node->next;
    free(a_node);
    L->length--;
    return OK;
}

```

```

// start find the ordered node from the 2nd node
while (a_node != NULL) {
    index++;
    if (index == order - 1) {
        link_node *tmp = a_node->next;
        *(elem) = tmp->data;
        a_node->next = a_node->next->next;
        free(tmp);
        L->length--;
        break;
    }
    a_node = a_node->next;
}
if (a_node != NULL)
    return OK;
else
    return ERROR;
}

void print_list(link_list L) {
/*
 * Function Name: print_list
 * Module: Data structures
 * Parameter: link_list L
 * Return: int(status)
 * Use: print the elements of the linklist to the payload
 */
int index = 0;
link_node *a_node = L.head;
while (a_node != NULL) {
    index++;
    printf("Index: %d, Data: %d\n", index, a_node->data);
    a_node = a_node->next;
}
}

```

#### C.4 main.h

```

/*
 * AUTHOR: Yue Pan
 * GITHUB: zxc479773533
 * E-MAIL: zxc479773533@gmail.com
*/

```

```
#ifndef PY_LINKLIST_MAIN_H_
#define PY_LINKLIST_MAIN_H_

#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include "my_linklist.c"

void print_menu(void);
/*
 * Function Name: print_menu
 * Module: Main control
 * Parameter: None
 * Return: None
 * Use: print the menu
 */

void load_data(Link_list_main *Main_L);
/*
 * Function Name: load_data
 * Module: Main control
 * Parameter: Link_list_main *Main_L
 * Return: None
 * Use: load link list from database
 */

void save_data(Link_list_main *Main_L);
/*
 * Function Name: save_data
 * Module: Main control
 * Parameter: Link_list_main *Main_L
 * Return: None
 * Use: save link list from database
 */

#endif // !PY_LINKLIST_MAIN_H_
```

### C.5 main.c

```
/*
```

```

* AUTHOR: Yue Pan
* GITHUB: zxc479773533
* E-MAIL: zxc479773533@gmail.com
*/
#include "main.h"

void print_menu(void) {
/*
 * Function Name: print_menu
 * Module: Main control
 * Parameter: None
 * Return: None
 * Use: print the menu
*/
printf("+-----+\n");
printf("|           Welcome to panyue's link list demo system!\n|\n");
printf("|           Here are some functions you can call:\n|\n");
printf("|           1: init_list          2: destroy_list\n|\n");
printf("|           3: clear_list         4: is_list_empty\n|\n");
printf("|           5: list_length        6: get_list_item\n|\n");
printf("           7: locate_list_item    8:\n|piror_list_item\n|\n");
printf("           9: next_list_item      10: list_insert\n|\n");
printf("           11: list_delete        12: print_list\n|\n");
printf("           13: ls_list            0: quit\n|\n");
printf("           |\nEnter the number of the function and see the usage and\n");
printf("call it!      |\n");
printf("           |Enter      0      to      quit      the      demo      system.\n");

```

```

|\n");
    printf("|
|\n");
    printf("|Copyrite      (C)      2017      Yue      Pan
|\n");
    printf("|Github:          zxc479773533
|\n");

printf("-----+-----+\n");
    printf("\n");
    printf("Your choose: ");
}

void load_data(Link_list_main *Main_L) {
/*
 * Function Name: load_data
 * Module: Main control
 * Parameter: Link_list_main *Main_L
 * Return: None
 * Use: load link list from database
 */

// open file
FILE *fp = fopen("linklistdb", "rb");
if (fp == NULL)
    return;

int size = 0xff;
int count = 0;
int index;
link_list my_list, *L = &my_list;
L->next = NULL;
while (1) {
    link_list *tmp_L = (link_list*)malloc(sizeof(link_list));
    size = fread(tmp_L, sizeof(link_list), 1, fp);
    if (size == 0)
        break;
    link_node my_node, *node = &my_node;
    node->next = NULL;
    for (index = 0; index < tmp_L->length; index++) {
        link_node           *tmp_node
        (link_node*)malloc(sizeof(link_list));
        fread(tmp_node, sizeof(link_node), 1, fp);
        =
}
}
}

```

```

        node->next = tmp_node;
        node = node->next;
    }
    count++;
    tmp_L->head = my_node.next;
    L->next = tmp_L;
    L = L->next;
}

Main_L->head = my_list.next;
Main_L->num = count;

// close file
fclose(fp);
}

void save_data(Link_list_main *Main_L) {
/*
 * Function Name: save_data
 * Module: Main control
 * Parameter: Link_list_main *Main_L
 * Return: None
 * Use: save link list from database
 */
}

// open data files
FILE *fp = fopen("linklistdb", "wb");
if (fp == NULL)
    return;

link_list *L = Main_L->head;
while (L != NULL) {
    fwrite(L, sizeof(link_list), 1, fp);
    link_node *a_node = L->head;
    int index;
    for (index = 0; index < L->length; index++) {
        fwrite(a_node, sizeof(link_node), 1, fp);
        a_node = a_node->next;
    }
    L = L->next;
}

// close file

```

```
fclose(fp);
}

int main(void) {

    // a variable to save the function choose
    int function_choose = 0xff;

    while (function_choose != 0) {

        // print the menu
        print_menu();
        scanf("%d", &function_choose);

        // load the data
        Link_list_main Main_L;
        Main_L.head = NULL;
        Main_L.num = 0;
        load_data(&Main_L);

        // some variable to save the parameters
        int id = 0, order = 0, elem = 0, index = 0;
        link_list *L = NULL;

        switch (function_choose) {

            case 1:
                printf("/*\n");
                printf(" * Function Name: init_list\n");
                printf(" * Module: Data structures\n");
                printf(" * Parameter: link_list *L\n");
                printf(" * Return: int(status)\n");
                printf(" * Use: initial the link list\n");
                printf(" */\n");
                printf("\n");
                printf("Then, enter the list id: ");

                scanf("%d", &id);
                L = Main_L.head;
                while (L != NULL) {
                    if (L->id == id)
                        break;
                    L = L->next;
                }
        }
    }
}
```

```

if (L != NULL) {
    printf("Initial error, this link list is already
exists!\n");
}
else {
    link_list new_L;
    init_list(&new_L);
    printf("Initial list %d succeed!\n", id);
    new_L.id = id;
    new_L.next = Main_L.head;
    Main_L.head = &new_L;
}

printf("\n");
break;

case 2:
printf("/*\n");
printf(" * Function Name: destroy_list\n");
printf(" * Module: Data structures\n");
printf(" * Parameter: link_list *L\n");
printf(" * Return: int(status)\n");
printf(" * Use: destroy the link list\n");
printf(" */\n");
printf("\n");
printf("Then, enter the list id: ");

scanf("%d", &id);
L = Main_L.head;
if (L->id == id) {
    Main_L.head = Main_L.head->next;
    destroy_list(L);
    printf("Link list %d has been deleted!\n", id);
    printf("\n");
    break;
}
else {
    while (L->next != NULL) {
        if (L->next->id == id)
            break;
        L = L->next;
    }
}
if (L->next == NULL) {

```

```

        printf("Destroy error, this link list is not exists!\n");
    }
    else {
        link_list *to_be_del = L->next;
        L->next = L->next->next;
        destroy_list(to_be_del);
        printf("Link list %d has been deleted!\n", id);
    }

    printf("\n");
    break;

case 3:
    printf("/*\n");
    printf(" * Function Name: clear_list\n");
    printf(" * Module: Data structures\n");
    printf(" * Parameter: link_list *L\n");
    printf(" * Return: int(status)\n");
    printf(" * Use: let the link list empty\n");
    printf(" */\n");
    printf("\n");
    printf("Then, enter the list id: ");

    scanf("%d", &id);
    L = Main_L.head;
    while (L != NULL) {
        if (L->id == id)
            break;
        L = L->next;
    }
    if (L == NULL) {
        printf("Clear error, this link list is not exists!\n");
    }
    else {
        if (clear_list(L) == OK) {
            printf("Link list %d has been cleared!\n", id);
        }
        else {
            printf("Clear error, this link is already empty!\n");
        }
    }

    printf("\n");
    break;
}

```

```
case 4:  
    printf("/*\n");  
    printf(" * Function Name: is_list_empty\n");  
    printf(" * Module: Data structures\n");  
    printf(" * Parameter: link_list L\n");  
    printf(" * Return: int(true or false)\n");  
    printf(" * Use: judge if the link list is empty\n");  
    printf(" */\n");  
    printf("\n");  
    printf("Then, enter the list id: ");  
  
    scanf("%d", &id);  
    L = Main_L.head;  
    while (L != NULL) {  
        if (L->id == id)  
            break;  
        L = L->next;  
    }  
    if (L == NULL) {  
        printf("Error, this link list is not exists!\n");  
    }  
    else {  
        if (is_list_empty(*L) == TRUE) {  
            printf("Link list %d is empty!\n", id);  
        }  
        else {  
            printf("Link list %d is not empty!\n", id);  
        }  
    }  
  
    printf("\n");  
    break;  
  
case 5:  
    printf("/*\n");  
    printf(" * Function Name: list_length\n");  
    printf(" * Module: Data structures\n");  
    printf(" * Parameter: link_list L\n");  
    printf(" * Return: int(the length)\n");  
    printf(" * Use: get the length of link list\n");  
    printf(" */\n");  
    printf("\n");  
    printf("Then, enter the list id: ");
```

```

scanf("%d", &id);
L = Main_L.head;
while (L != NULL) {
    if (L->id == id)
        break;
    L = L->next;
}
if (L == NULL) {
    printf("Error, this link list is not exists!\n");
}
else {
    printf("The length of link list %d is %d.\n", id,
list_length(*L));
}

printf("\n");
break;

case 6:
printf("/*\n");
printf(" * Function Name: get_list_item\n");
printf(" * Module: Data structures\n");
printf(" * Parameter: link_list L, int order, int *elem\n");
printf(" * Return: int(status)\n");
printf(" * Use: get the ordered element of link list\n");
printf(" */\n");
printf("\n");
printf("Format: id index\n");
printf("Then, enter the list id and index: ");

scanf("%d %d", &id, &order);
L = Main_L.head;
while (L != NULL) {
    if (L->id == id)
        break;
    L = L->next;
}
if (L == NULL) {
    printf("Error, this link list is not exists!\n");
}
else {
    if (get_list_item(*L, order, &elem) == OK) {
        printf("The element is %d.\n", elem);
    }
}

```

```

    }
    else {
        printf("Get element error! Please check your
input.\n");
    }
}

printf("\n");
break;

case 7:
printf("/*\n");
printf(" * Function Name: locate_list_item\n");
printf(" * Module: Data structures\n");
printf(" * Parameter: link_list L, int ordered_elem\n");
printf(" * Return: int(index)\n");
printf(" * Use: get the index of ordered item\n");
printf(" */\n");
printf("\n");
printf("                                     Format: id
element\n");
printf("Then, enter the list id and element: ");

scanf("%d %d", &id, &elem);
L = Main_L.head;
while (L != NULL) {
    if (L->id == id)
        break;
    L = L->next;
}
if (L == NULL) {
    printf("Error, this link list is not exists!\n");
}
else {
    index = locate_list_item(*L, elem);
    printf("The index of the %d, is %d.\n", elem, index);
}

printf("\n");
break;

case 8:
printf("/*\n");
printf(" * Function Name: piror_list_item\n");

```

```

        printf(" * Module: Data structures\n");
        printf(" * Parameter: link_list L, int elem, int
*elem_pre\n");
        printf(" * Return: int(status)\n");
        printf(" * Use: get the ordered element's piror\n");
        printf(" */\n");
        printf("\n");
        printf("                                     Format: id
element\n");
        printf("Then, enter the list id and element: ");

        scanf("%d %d", &id, &elem);
        L = Main_L.head;
        while (L != NULL) {
            if (L->id == id)
                break;
            L = L->next;
        }
        if (L == NULL) {
            printf("Error, this link list is not exists!\n");
        }
        else {
            int elem_pre;
            if (piror_list_item(*L, elem, &elem_pre) == OK) {
                printf("The piror element of %d is %d.\n", elem,
elem_pre);
            }
            else {
                printf("Error, the piror of %d is not exists!\n", elem);
            }
        }

        printf("\n");
        break;

case 9:
        printf("/*\n");
        printf(" * Function Name: next_list_item\n");
        printf(" * Module: Data structures\n");
        printf(" * Parameter: link_list L, int elem, int
*elem_next\n");
        printf(" * Return: int(status)\n");
        printf(" * Use: get the ordered element's next\n");
        printf(" */\n");

```

```

printf("\n");
printf("                                     Format: id
element\n");
printf("Then, enter the list id and element: ");

scanf("%d %d", &id, &elem);
L = Main_L.head;
while (L != NULL) {
    if (L->id == id)
        break;
    L = L->next;
}
if (L == NULL) {
    printf("Error, this link list is not exists!\n");
}
else {
    int elem_next;
    if (next_list_item(*L, elem, &elem_next) == OK) {
        printf("The next element of %d is %d.\n", elem,
elem_next);
    }
    else {
        printf("Error, the next of %d is not exists!\n", elem);
    }
}

printf("\n");
break;

case 10:
printf("/*\n");
printf(" * Function Name: list_insert\n");
printf(" * Module: Data structures\n");
printf(" * Parameter: link_list *L, int order, int elem\n");
printf(" * Return: int(status)\n");
printf(" * Use: insert a element in the link list\n");
printf(" */\n");
printf("\n");
printf("                                     Format: id index
element\n");
printf("Then, enter the list id, index and element: ");

scanf("%d %d %d", &id, &order, &elem);
L = Main_L.head;

```

```

while (L != NULL) {
    if (L->id == id)
        break;
    L = L->next;
}
if (L == NULL) {
    printf("Error, this link list is not exists!\n");
}
else {
    if (list_insert(L, order, elem) == OK) {
        printf("Insert %d succeed!\n", elem);
    }
    else {
        printf("Insert failed, please check your input!\n");
    }
}

printf("\n");
break;

case 11:
printf("/*\n");
printf(" * Function Name: list_delete\n");
printf(" * Module: Data structures\n");
printf(" * Parameter: link_list *L, int order, int
*elem\n");
printf(" * Return: int(status)\n");
printf(" * Use: delete a element in the link list\n");
printf(" */\n");
printf("\n");
printf("                                Format: id index\n");
printf("Then, enter the list id and index: ");

scanf("%d %d", &id, &order);
L = Main_L.head;
while (L != NULL) {
    if (L->id == id)
        break;
    L = L->next;
}
if (L == NULL) {
    printf("Error, this link list is not exists!\n");
}
else {

```

```
if (list_delete(L, order, &elem) == OK) {
    printf("Delete %d succeed!\n", elem);
}
else {
    printf("Delete failed, please check your input!\n");
}
}

printf("\n");
break;

case 12:
printf("/*\n");
printf(" * Function Name: print_list\n");
printf(" * Module: Data structures\n");
printf(" * Parameter: link_list L, char *payload\n");
printf(" * Return: None\n");
printf(" * Use: print the elements of the linklist to the
payload\n");
printf(" */\n");
printf("\n");
printf("Then, enter the list id: ");

scanf("%d", &id);
L = Main_L.head;
while (L != NULL) {
    if (L->id == id)
        break;
    L = L->next;
}
if (L == NULL) {
    printf("Error, this link list is not exists!\n");
}
else {
    printf("The elements in link list %d are:\n", id);
    print_list(*L);
}

printf("\n");
break;

case 13:
L = Main_L.head;
while(L != NULL) {
```

```
    printf("ID: %d\n", L->id);
    L = L->next;
}

printf("\n");
break;

case 0:
    printf("Thanks for using my demo system!\n");
    break;

default:
    printf("You entered the wrong num, please re-enter the
num.\n");
    printf("\n");
    break;

}

save_data(&Main_L);
}

return 0;
}
```

## 附录 D 基于二叉链表二叉树实现的源代码

注：由于实验的编译文件 Makefile 使用 CMake 工具生成，故在这里附上 CMakeLists 文件

### D. 1 CMakeLists

```
# copyright Pan Yue
project(lab03)

add_library(binary_tree my_binarytree.c)

add_library(libqueue mylibqueue.c)

add_executable(lab03 main.c)

target_link_libraries(lab03 libqueue binary_tree)
```

### D. 2 my\_binarytree.h

```
/*
 * AUTHOR: Yue Pan
 * GITHUB: zxc479773533
 * E-MAIL: zxc479773533@gmail.com
 */

#ifndef PY_BINARY_TREE_H
#define PY_BINARY_TREE_H

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

#define TRUE 1
#define FALSE 0

#define OK 0
#define ERROR 1

#define LEFT 0
#define RIGHT 1
```

```
#define PRE_ORDER 0
#define IN_ORDER 1
#define POST_ORDER 2

// the struct of binary tree node
typedef struct binary_tree_node {
    int index;
    int value;
    struct binary_tree_node *left_child;
    struct binary_tree_node *right_child;
} binary_tree_node;

// the main struct of binary tree
typedef struct binary_tree {
    int id;
    int size;
    struct binary_tree_node *root;
    struct binary_tree *next;
} binary_tree;

// the struct for managin the binary tree
typedef struct Binary_tree_main {
    binary_tree *head;
} Binary_tree_main;

// the struct to save the binary tree
typedef struct Binary_tree_store {
    int id;
    int size;
    int *pre_index;
    int *pre_defination;
    int *in_index;
    int *in_defination;
} Binary_tree_store;

int init_binary_tree(binary_tree *T);
/*
 * Function Name: init_binary_tree
 * Module: Data structures
 * Parameter: binary_tree *T
 * Return: int(status)
 * Use: initial the binary tree
 */
```

```
int destroy_bitree(binary_tree *T);
/*
 * Function Name: destroy_bitree
 * Module: Data structures
 * Parameter: binary_tree *T
 * Return: int(status)
 * Use: destroy the binary tree
 */

int create_bitree(binary_tree *T, int *pre_index, int
*pre_defination, int *in_index, int *in_defination, int
dification_len);
/*
 * Function Name: create_bitree
 * Module: Data structures
 * Parameter: binary_tree *T, int *pre_index, int *pre_defination,
int *in_index, int *in_defination, int dification_len
 * Return: int(status)
 * Use: create the binary tree with some nodes
 */

int clear_bitree(binary_tree *T);
/*
 * Function Name: clear_bitree
 * Module: Data structures
 * Parameter: binary_tree *T
 * Return: int(status)
 * Use: clear the binary tree
 */

int is_bitree_empty(binary_tree T);
/*
 * Function Name: is_bitree_empty
 * Module: Data structures
 * Parameter: binary_tree T
 * Return: int(true or false)
 * Use: judge if the binary tree is empty
 */
```

```
int bitree_depth(binary_tree T);
/*
 * Function Name: bitree_depth
 * Module: Data structures
 * Parameter: binary_tree T
 * Return: int(the depth)
 * Use: get the depth of binary tree
 */

binary_tree_node* bitree_root(binary_tree T);
/*
 * Function Name: bitree_root
 * Module: Data structures
 * Parameter: binary_tree T
 * Return: binary_tree_node*
 * Use: get the root of binary tree
 */

int bitree_get_value(binary_tree T, int index);
/*
 * Function Name: bitree_get_value
 * Module: Data structures
 * Parameter: binary_tree T, int index
 * Return: int(the value)
 * Use: get the value of a node
 */

int bitree_set_value(binary_tree *T, int index, int value);
/*
 * Function Name: bitree_set_value
 * Module: Data structures
 * Parameter: binary_tree *T, int index, int value
 * Return: int(status)
 * Use: set the value of a node
 */

binary_tree_node* bitree_parent(binary_tree T, int index);
/*
 * Function Name: bitree_parent
 * Module: Data structures
```

```
* Parameter: binary_tree T, int index
* Return: binary_tree_node*
* Use: get parent of a node
*/
```

```
binary_tree_node* bitree_left_child(binary_tree T, int index);
/*
 * Function Name: bitree_left_child
 * Module: Data structures
 * Parameter: binary_tree T, int index
 * Return: binary_tree_node*
 * Use: get left child of a node
*/
```

```
binary_tree_node* bitree_right_child(binary_tree T, int index);
/*
 * Function Name: bitree_right_child
 * Module: Data structures
 * Parameter: binary_tree T, int index
 * Return: binary_tree_node*
 * Use: get right child of a node
*/
```

```
binary_tree_node* bitree_left_sibling(binary_tree T, int index);
/*
 * Function Name: bitree_left_sibling
 * Module: Data structures
 * Parameter: binary_tree T, int index
 * Return: binary_tree_node*
 * Use: get left sibling of a node
*/
```

```
binary_tree_node* bitree_right_sibling(binary_tree T, int index);
/*
 * Function Name: bitree_right_sibling
 * Module: Data structures
 * Parameter: binary_tree T, int index
 * Return: binary_tree_node*
 * Use: get right sibling of a node
*/
```

```
int bitree_insert_child(binary_tree *T, int index, int LorR,
binary_tree *C);
/*
 * Function Name: bitree_insert_child
 * Module: Data structures
 * Parameter: binary_tree *T, int index, int LorR, binary_tree *C
 * Return: int(status)
 * Use: insert a child tree in a binary tree
 */

int bitree_delete_child(binary_tree *T, int index, int LorR);
/*
 * Function Name: bitree_delete_child
 * Module: Data structures
 * Parameter: binary_tree *T, int index, int LorR
 * Return: int(status)
 * Use: delete a child tree in a binary tree
 */

int bitree_preorder_traverse(binary_tree T);
/*
 * Function Name: bitree_preorder_traverse
 * Module: Data structures
 * Parameter: binary_tree T
 * Return: int(status)
 * Use: preorder traverse and print the values
 */

int bitree_inorder_traverse(binary_tree T);
/*
 * Function Name: bitree_inorder_traverse
 * Module: Data structures
 * Parameter: binary_tree T
 * Return: int(status)
 * Use: inorder traverse and print the values
 */

int bitree_postorder_traverse(binary_tree T);
```

```

/*
 * Function Name: bitree_postorder_traverse
 * Module: Data structures
 * Parameter: binary_tree T
 * Return: int(status)
 * Use: postorder traverse and print the values
 */

int bitree_levelorder_traverse(binary_tree T);
/*
 * Function Name: bitree_levelorder_traverse
 * Module: Data structures
 * Parameter: binary_tree T
 * Return: int(status)
 * Use: levelorder traverse and print the values
 */
#endif // ! PY_BINARY_TREE_H

```

### D.3 my\_binarytree.c

```

/*
 * AUTHOR: Yue Pan
 * GITHUB: zxc479773533
 * E-MAIL: zxc479773533@gmail.com
 */

#include "my_binarytree.h"
#include "mylibqueue.c"

// assist fuction to search array
int search_array(int *array, int size, int target){
    int i;
    for(i = 0; i < size; i++) {
        if(array[i] == target)
            return i;
    }
    return -1;
}

int init_binary_tree(binary_tree *T) {
/*

```

```
* Function Name: init_binary_tree
* Module: Data structures
* Parameter: binary_tree *T
* Return: int(status)
* Use: initial the binary tree
*/
T->size = 0;
T->root = NULL;
return OK;
}

// assist function to free all nodes
void free_nodes(binary_tree_node *node) {
    if (node == NULL)
        return ;
    // free left child
    if (node->left_child != NULL)
        free_nodes(node->left_child);
    // free right child
    if (node->right_child != NULL)
        free_nodes(node->right_child);
    // free self
    free(node);
}

int destroy_bitree(binary_tree *T){
/*
 * Function Name: destroy_bitree
 * Module: Data structures
 * Parameter: binary_tree *T
 * Return: int(status)
 * Use: destroy the binary tree
*/
    free_nodes(T->root);
    free(T);
    return OK;
}

// assist function to create nodes
binary_tree_node *create_nodes(int *ind1, int *def1, int *ind2,
int *def2, int max_pos, int min_pos, int *pos) {
    // start
```

```

if (pos == NULL) {
    int new_pos = 0;
    return create_nodes(ind1, def1, ind2, def2, max_pos, min_pos,
&new_pos);
}
// get root pos in inorder definition
int root_pos = search_array(def2 + min_pos, max_pos - min_pos,
def1[*pos]);
if (root_pos == -1)
    return NULL;
else
    root_pos += min_pos;
// create a node
binary_tree_node *root =
(binary_tree_node*)malloc(sizeof(binary_tree_node));
root->index = ind1[*pos];
root->value = def1[*pos];
(*pos)++;
// the left of root in preorder definition is the left child tree
root->left_child = create_nodes(ind1, def1, ind2, def2, root_pos,
min_pos, pos);
// the right of root in preorder definition is the right child
tree
    root->right_child = create_nodes(ind1, def1, ind2, def2, max_pos,
root_pos + 1, pos);
    return root;
}

int create_bitree(binary_tree *T, int *pre_index, int
*pre_definition, int *in_index, int *in_definition, int
dification_len) {
/*
 * Function Name: create_bitree
 * Module: Data structures
 * Parameter: binary_tree *T, int *pre_index, int *pre_definition,
int *in_index, int *in_definition, int dification_len
 * Return: int(status)
 * Use: create the binary tree with some nodes
*/
if (dification_len <= 0)
    return ERROR;
T->size = dification_len;
T->root = create_nodes(pre_index, pre_definition, in_index,

```

```
in_defination, difinition_len, 0, NULL);
    return OK;
}

int clear_bitree(binary_tree *T) {
/*
 * Function Name: clear_bitree
 * Module: Data structures
 * Parameter: binary_tree *T
 * Return: int(status)
 * Use: clear the binary tree
*/
    T->size = 0;
    free_nodes(T->root);
    T->root = NULL;
    return OK;
}

int is_bitree_empty(binary_tree T) {
/*
 * Function Name: is_bitree_empty
 * Module: Data structures
 * Parameter: binary_tree T
 * Return: int(tree or false)
 * Use: judge if the binary tree is empty
*/
    if (T.size == 0)
        return TRUE;
    else
        return FALSE;
}

// assist function to get depth
int get_depth(binary_tree_node *node, int depth) {
    if (node == NULL)
        return depth;
    // count left depth
    int left_depth = get_depth(node->left_child, depth + 1);
    // count right depth
    int right_depth = get_depth(node->right_child, depth + 1);
    // return the bigger of left and right depth
    return left_depth > right_depth ? left_depth : right_depth;
}
```

```
}

int bitree_depth(binary_tree T) {
/*
 * Function Name: bitree_depth
 * Module: Data structures
 * Parameter: binary_tree T
 * Return: int(the depth)
 * Use: get the depth of binary tree
 */

    return get_depth(T.root, 0);
}

binary_tree_node* bitree_root(binary_tree T) {
/*
 * Function Name: bitree_root
 * Module: Data structures
 * Parameter: binary_tree T
 * Return: binary_tree_node*
 * Use: get the root of binary tree
 */
    return T.root;
}

// assist function to get value of a node
int get_value(binary_tree_node* node, int index) {
    if (node == NULL)
        return 0;
    if (node->index == index)
        return node->value;
    else {
        // find in left child tree
        int find_left = get_value(node->left_child, index);
        // find in right child tree
        int find_right = get_value(node->right_child, index);
        return find_left | find_right;
    }
}

int bitree_get_value(binary_tree T, int index) {
/*
 * Function Name: bitree_get_value
 * Module: Data structures
 *
```

```

* Parameter: binary_tree T, int index
* Return: int(the value)
* Use: get the value of a node
*/
if (T.root == NULL)
    return 0;
return get_value(T.root, index);
}

// assist function to set value of a node
void set_value(binary_tree_node *node, int index, int value, int
*safe_tag) {
    if (node == NULL)
        return ;
    if (node->index == index) {
        node->value = value;
        // safe tag == 1 means set succeed
        *safe_tag = 1;
    }
    else {
        // set in left child tree
        set_value(node->left_child, index, value, safe_tag);
        // set in right child tree
        set_value(node->right_child, index, value, safe_tag);
    }
}

int bitree_set_value(binary_tree *T, int index, int value) {
/*
 * Function Name: bitree_set_value
 * Module: Data structures
 * Parameter: binary_tree *T, int index, int value
 * Return: int(status)
 * Use: set the value of a node
*/
    int safe_tag = 0;
    set_value(T->root, index, value, &safe_tag);
    if (safe_tag == 1)
        return OK;
    else
        return ERROR;
}

```

```

// assist function to get parent
binary_tree_node* get_parent(binary_tree_node* node, int index)
{
    if (node == NULL)
        return NULL;
    if ((node->left_child != NULL && node->left_child->index == index) ||
        (node->right_child != NULL && node->right_child->index == index))
        return node;
    // find in left child tree
    binary_tree_node *parent = get_parent(node->left_child, index);
    // find in right child tree
    if (parent == NULL)
        parent = get_parent(node->right_child, index);
    return parent;
}

binary_tree_node* bitree_parent(binary_tree T, int index) {
/*
 * Function Name: bitree_parent
 * Module: Data structures
 * Parameter: binary_tree T, int key
 * Return: binary_tree_node*
 * Use: get parent of a node
 */
    return get_parent(T.root, index);
}

// assist function to get child
binary_tree_node* get_child(binary_tree_node *node, int LorR, int
index) {
    if (node == NULL)
        return NULL;
    if (node->index == index) {
        if (LorR == LEFT)
            return node->left_child;
        else
            return node->right_child;
    }
    // find in left child tree
    binary_tree_node *child = get_child(node->left_child, LorR,

```

```

index);
    // find in right child tree
    if (child == NULL)
        child = get_child(node->right_child, LorR, index);
    return child;
}

binary_tree_node* bitree_left_child(binary_tree T, int index) {
/*
 * Function Name: bitree_left_child
 * Module: Data structures
 * Parameter: binary_tree T, int index
 * Return: binary_tree_node*
 * Use: get left child of a node
 */
    return get_child(T.root, LEFT, index);
}

binary_tree_node* bitree_right_child(binary_tree T, int index) {
/*
 * Function Name: bitree_right_child
 * Module: Data structures
 * Parameter: binary_tree T, int index
 * Return: binary_tree_node*
 * Use: get right child of a node
 */
    return get_child(T.root, RIGHT, index);
}

// assist function to get left sibling
binary_tree_node* get_sibling(binary_tree_node *node, int LorR,
int index) {
    if (node == NULL)
        return NULL;
    // get right child's left sibling
    if (node->left_child != NULL && node->left_child->index == index
&& LorR == RIGHT)
        return node->right_child;
    // get left child's right sibling
    else if (node->right_child != NULL && node->right_child->index
== index && LorR == LEFT)
        return node->left_child;
}

```

```

// find in left child tree
binary_tree_node *sibling = get_sibling(node->left_child, LorR,
index);
// find in right child tree
if (sibling == NULL)
    sibling = get_sibling(node->right_child, LorR, index);
return sibling;
}

binary_tree_node* bitree_left_sibling(binary_tree T, int index)
{
/*
 * Function Name: bitree_left_sibling
 * Module: Data structures
 * Parameter: binary_tree T, int index
 * Return: binary_tree_node*
 * Use: get left sibling of a node
 */
return get_sibling(T.root, LEFT, index);
}

binary_tree_node* bitree_right_sibling(binary_tree T, int index)
{
/*
 * Function Name: bitree_right_sibling
 * Module: Data structures
 * Parameter: binary_tree T, int index
 * Return: binary_tree_node*
 * Use: get right sibling of a node
 */
return get_sibling(T.root, RIGHT, index);
}

// assist function to insert child tree
binary_tree_node* insert_child_tree(binary_tree_node *node, int
index, int LorR, binary_tree_node *sub_tree) {
    if (node == NULL)
        return NULL;
    if (node->index == index) {
        binary_tree_node *tmp_node;
        // insert as left child
        if (LorR == LEFT) {

```

```

    tmp_node = node->left_child;
    node->left_child = sub_tree;
}
// insert as right child
else {
    tmp_node = node->right_child;
    node->right_child = sub_tree;
}
sub_tree->right_child = tmp_node;
}
// find in left child tree
node->left_child = insert_child_tree(node->left_child, index,
LorR, sub_tree);
// find in right child tree
node->right_child = insert_child_tree(node->right_child, index,
LorR, sub_tree);
return node;
}

int bitree_insert_child_tree(binary_tree *T, int index, int LorR,
binary_tree *C) {
/*
 * Function Name: bitree_insert_child
 * Module: Data structures
 * Parameter: binary_tree *T, int index, int LorR, binary_tree_node
 * *sub_tree
 * Return: int(status)
 * Use: insert a child tree in a binary tree
 */
if (T->root == NULL || C->root == NULL || C->root->right_child != NULL)
    return ERROR;
T->size += C->size;
T->root = insert_child_tree(T->root, index, LorR, C->root);
return OK;
}

// assist function to search a node
int search_bitree(binary_tree_node *node, int index) {
    if (node == NULL)
        return 0;
    if (node->index == index)
        return 1;
}

```

```

    return      search_bitree(node->left_child,      index)      |
search_bitree(node->right_child, index);
}

// assist function to count binary tree nodes
int count_bitree(binary_tree_node *node) {
    if (node == NULL)
        return 0;
    return      1      +      count_bitree(node->left_child)      +
count_bitree(node->right_child);
}

binary_tree_node* delete_child_tree(binary_tree_node *node, int
index, int LorR) {
    if (node == NULL)
        return NULL;
    if (node->index == index) {
        // delete left child tree
        if (LorR == LEFT) {
            free_nodes(node->left_child);
            node->left_child = NULL;
        }
        // delete right child tree
        else {
            free_nodes(node->right_child);
            node->right_child = NULL;
        }
    }
    // find in left child tree
    node->left_child = delete_child_tree(node->left_child, index,
LorR);
    // find in right child tree
    node->right_child = delete_child_tree(node->right_child, index,
LorR);
    return node;
}

int bitree_delete_child_tree(binary_tree *T, int index, int LorR)
{
/*
 * Function Name: bitree_delete_child
 * Module: Data structures
 * Parameter: binary_tree *T, int index, int LorR

```

```

* Return: int(status)
* Use: delete a child tree in a binary tree
*/
if (T->root == NULL || !search_bitree(T->root, index))
    return ERROR;
T->root = delete_child_tree(T->root, index, LOrR);
T->size = count_bitree(T->root);
return OK;
}

// assist function to traverse the binary tree
void traverse_bitree(binary_tree_node *node, int order) {
    if (node == NULL)
        return ;
    if (order == PRE_ORDER) {
        printf("Index: %d, value: %d\n", node->index, node->value);
        traverse_bitree(node->left_child, order);
        traverse_bitree(node->right_child, order);
    }
    else if (order == IN_ORDER) {
        traverse_bitree(node->left_child, order);
        printf("Index: %d, value: %d\n", node->index, node->value);
        traverse_bitree(node->right_child, order);
    }
    else if (order == POST_ORDER) {
        traverse_bitree(node->left_child, order);
        traverse_bitree(node->right_child, order);
        printf("Index: %d, value: %d\n", node->index, node->value);
    }
}

int bitree_preorder_traverse(binary_tree T) {
/*
 * Function Name: bitree_preorder_traverse
 * Module: Data structures
 * Parameter: binary_tree T
 * Return: int(status)
 * Use: preorder traverse and print the values
*/
if (T.root == NULL)
    return ERROR;
printf("Preorder traverse:\n");

```

```
printf("Tree size: %d\n", T.size);
traverse_bitree(T.root, PRE_ORDER);
return OK;
}

int bitree_inorder_traverse(binary_tree T) {
/*
 * Function Name: bitree_inorder_traverse
 * Module: Data structures
 * Parameter: binary_tree T
 * Return: int(status)
 * Use: inorder traverse and print the values
 */

if (T.root == NULL)
    return ERROR;
printf("Inorder traverse:\n");
printf("Tree size: %d\n", T.size);
traverse_bitree(T.root, IN_ORDER);
return OK;
}

int bitree_postorder_traverse(binary_tree T) {
/*
 * Function Name: bitree_postorder_traverse
 * Module: Data structures
 * Parameter: binary_tree T
 * Return: int(status)
 * Use: postorder traverse and print the values
 */

if (T.root == NULL)
    return ERROR;
printf("Postorder traverse:\n");
printf("Tree size: %d\n", T.size);
traverse_bitree(T.root, POST_ORDER);
return OK;
}

int bitree_levelorder_traverse(binary_tree T) {
/*
 * Function Name: bitree_levelorder_traverse
 * Module: Data structures
 * Parameter: binary_tree T
```

```

* Return: int(status)
* Use: levelorder traverse and print the values
*/
if (T.root == NULL)
    return ERROR;
LinkQueue my_node_queue;
InitQueue(&my_node_queue);
binary_tree_node *node;
printf("Levelorder traverse:\n");
printf("Tree size: %d\n", T.size);
// use a queue to levelorder traverse the binary tree
EnQueue(&my_node_queue, T.root);
while (!IsEmpty(&my_node_queue)) {
    node = DeQueue(&my_node_queue);
    printf("Index: %d, value: %d\n", node->index, node->value);
    if (node->left_child != NULL)
        EnQueue(&my_node_queue, node->left_child);
    if (node->right_child != NULL)
        EnQueue(&my_node_queue, node->right_child);
}
return OK;
}

```

#### D.4 main.h

```

/*
 * AUTHOR: Yue Pan
 * GITHUB: zxc479773533
 * E-MAIL: zxc479773533@gmail.com
 */

#ifndef PY_BINARYTREE_MAIN_H_
#define PY_BINARYTREE_MAIN_H_

#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include "my_binarytree.c"

void print_menu(void);
/*
 * Function Name: print_menu
 * Module: Main control

```

```
* Parameter: None
* Return: None
* Use: print the menu
*/
void load_data(Binary_tree_main *Main_T);
/*
 * Function Name: load_data
 * Module: Main control
 * Parameter: Binary_tree_main *Main_T
 * Return: None
 * Use: load link list from database
*/
void save_data(Binary_tree_main *Main_T);
/*
 * Function Name: save_data
 * Module: Main control
 * Parameter: Binary_tree_main *Main_T
 * Return: None
 * Use: save link list from database
*/
#endif // !PY_LINKLIST_MAIN_H_
```

## D.5 main.c

```
/*
 * AUTHOR: Yue Pan
 * GITHUB: zxc479773533
 * E-MAIL: zxc479773533@gmail.com
*/
#include "main.h"

void print_menu(void) {
/*
 * Function Name: print_menu
 * Module: Main control
 * Parameter: None
 * Return: None
```

```

* Use: print the menu
*/
printf("+
-----+\n");
printf("|           Welcome to panyue's binary tree demo system!
|\n");
printf("|           Here are some functions you can call:
|\n");
printf("|           1: init_binary_tree
|destroy_bitree      |\n");
printf("|           3: create_bitree
|\n");
printf("|           5: is_bitree_empty
|           6: bitree_depth
|\n");
printf("|           7: bitree_root
bitree_get_value     |\n");
printf("|           9: bitree_set_value
bitree_parent        |\n");
printf("|           11: bitree_left_child
bitree_right_child   |\n");
printf("|           13: bitree_left_sibling
bitree_right_sibling |\n");
printf("|           15: bitree_insert_child
bitree_delete_child  |\n");
printf("|           17: bitree_preorder_traverse
bitree_inorder_traverse  |\n");
printf("|           19: bitree_postorder_traverse
bitree_levelorder_traverse |\n");
printf("|           21: ls_bitree
|\n");
printf("|           0: quit
|\n");
printf("|Enter the number of the function and see the usage and
call it!      |\n");
printf("|Enter      0      to      quit      the      demo      system.
|\n");
printf("|           (C)           2017           Yue           Pan
|\n");
printf("|Copyright
|\n");

```

```
printf("|Github:  
|\n");  
  
printf("-----+-----+\n");  
printf("\n");  
printf("Your choose: ");  
}  
  
void load_data(Binary_tree_main *Main_T) {  
/*  
 * Function Name: load_data  
 * Module: Main control  
 * Parameter: Binary_tree_main *Main_T  
 * Return: None  
 * Use: load link list from database  
 */  
  
// open data file  
FILE *fp = fopen("binarytreedb", "rb");  
if (fp == NULL)  
    return ;  
  
binary_tree trees, *a_tree = &trees;  
a_tree->next = NULL;  
  
int size = 0xff;  
while (1) {  
    Binary_tree_store def;  
    binary_tree *tmp_tree = NULL;  
    size = fread(&def, sizeof(Binary_tree_store), 1, fp);  
    if (size == 0)  
        break;  
  
    tmp_tree = (binary_tree*)malloc(sizeof(binary_tree));  
    init_binary_tree(tmp_tree);  
  
    if (def.size != 0) {  
        def.pre_index = (int*)malloc(sizeof(int) * def.size);  
        def.pre_defination = (int*)malloc(sizeof(int) * def.size);  
        def.in_index = (int*)malloc(sizeof(int) * def.size);  
        def.in_defination = (int*)malloc(sizeof(int) * def.size);  
        fread(def.pre_index, sizeof(int) * def.size, 1, fp);  
        fread(def.pre_defination, sizeof(int) * def.size, 1, fp);  
    }  
}
```

```

        fread(def.in_index, sizeof(int) * def.size, 1, fp);
        fread(def.in_defination, sizeof(int) * def.size, 1, fp);
        create_bitree(tmp_tree, def.pre_index, def.pre_defination,
def.in_index, def.in_defination, def.size);
    }

    tmp_tree->id = def.id;
    tmp_tree->next = NULL;
    a_tree->next = tmp_tree;
    a_tree = a_tree->next;
}

Main_T->head = trees.next;

// close data file
fclose(fp);
}

// assist function to preorder, inorder traverse the tree save nodes
void traverse_and_save(binary_tree_node *node, int order, int
*indexes, int *defination, int *pos) {
    if (node == NULL)
        return ;
    if (pos == NULL) {
        int new_pos = 0;
        traverse_and_save(node, order, indexes, defination, &new_pos);
        return ;
    }
    if (order == PRE_ORDER) {
        indexes[*pos] = node->index;
        defination[*pos] = node->value;
        (*pos)++;
        traverse_and_save(node->left_child,      order,      indexes,
defination, pos);
        traverse_and_save(node->right_child,     order,      indexes,
defination, pos);
    }
    else if (order == IN_ORDER) {
        traverse_and_save(node->left_child,      order,      indexes,
defination, pos);
        indexes[*pos] = node->index;
        defination[*pos] = node->value;
        (*pos)++;
        traverse_and_save(node->right_child,     order,      indexes,
defination, pos);
    }
}

```

```

definition, pos);
}
}

void save_data(Binary_tree_main *Main_T) {
/*
 * Function Name: save_data
 * Module: Main control
 * Parameter: Binary_tree_main *Main_T
 * Return: None
 * Use: save link list from database
 */

// open data files
FILE *fp = fopen("binarytreedb", "wb");
if (fp == NULL)
    return ;

binary_tree *a_tree = Main_T->head;
while (a_tree != NULL) {
    Binary_tree_store def;
    def.id = a_tree->id;
    def.size = a_tree->size;
    def.pre_index = (int*)malloc(sizeof(int) * def.size);
    def.pre_defination = (int*)malloc(sizeof(int) * def.size);
    def.in_index = (int*)malloc(sizeof(int) * def.size);
    def.in_defination = (int*)malloc(sizeof(int) * def.size);
    traverse_and_save(a_tree->root, PRE_ORDER, def.pre_index,
def.pre_defination, NULL);
    traverse_and_save(a_tree->root, IN_ORDER, def.in_index,
def.in_defination, NULL);
    fwrite(&def, sizeof(Binary_tree_store), 1, fp);
    fwrite(def.pre_index, sizeof(int) * def.size, 1, fp);
    fwrite(def.pre_defination, sizeof(int) * def.size, 1, fp);
    fwrite(def.in_index, sizeof(int) * def.size, 1, fp);
    fwrite(def.in_defination, sizeof(int) * def.size, 1, fp);
    a_tree = a_tree->next;
}

// close file
fclose(fp);
}

int main(void) {

```

```
// a variable to save the function choose
int function_choose = 0xff;

while (function_choose != 0) {

    // print the menu
    print_menu();
    scanf("%d", &function_choose);
    Binary_tree_main Main_T;
    Main_T.head = NULL;
    load_data(&Main_T);

    // some variable to save the parameters
    int id, node_num, index, value, LorR;
    binary_tree *T = NULL;

    switch(function_choose) {

        case 1:
            printf("/*\n");
            printf(" * Function Name: init_binary_tree\n");
            printf(" * Module: Data structures\n");
            printf(" * Parameter: binary_tree *T\n");
            printf(" * Return: int(status)\n");
            printf(" * Use: initial the binary tree\n");
            printf(" */\n");
            printf("\n");
            printf("Then, enter the bintry tree id: ");

            scanf("%d", &id);
            T = Main_T.head;
            while (T != NULL) {
                if (T->id == id)
                    break;
                T = T->next;
            }
            if (T != NULL) {
                printf("Initial error, this bintry tree is already
exists!\n");
            }
            else {
                binary_tree new_T;
                init_binary_tree(&new_T);
```

```
printf("Initial binary tree %d succeed!\n", id);
new_T.id = id;
new_T.next = Main_T.head;
Main_T.head = &new_T;
}

printf("\n");
break;

case 2:
printf("/*\n");
printf(" * Function Name: destroy_bitree\n");
printf(" * Module: Data structures\n");
printf(" * Parameter: binary_tree *T\n");
printf(" * Return: int(status)\n");
printf(" * Use: destroy the binary tree\n");
printf(" */\n");
printf("\n");
printf("Then, enter the binary tree id: ");

scanf("%d", &id);
T = Main_T.head;
if (T->id == id) {
    Main_T.head = Main_T.head->next;
    destroy_bitree(T);
    printf("Binary tree %d has been deleted!\n", id);
    printf("\n");
    break;
}
else {
    while (T->next != NULL) {
        if (T->next->id == id)
            break;
        T = T->next;
    }
}
if (T->next == NULL) {
    printf("Destroy error, this binary trss is not
exists!\n");
}
else {
    binary_tree *to_be_del = T->next;
    T->next = T->next->next;
    destroy_bitree(to_be_del);
```

```

        printf("Binary tree %d has been deleted!\n", id);
    }

    printf("\n");
    break;

case 3:
    printf("/*\n");
    printf(" * Function Name: create_bitree\n");
    printf(" * Module: Data structures\n");
    printf(" * Parameter: binary_tree *T, int *pre_index, int
*pre_definition,\n");
    printf(" *      int *in_index, int *in_definition, int
definition_len\n");
    printf(" * Return: int(status)\n");
    printf(" * Use: create the binary tree with some nodes\n");
    printf(" */\n");
    printf("\n");
    printf("                                         Format: id
size\n");
    printf("Then, enter the binary tree id and tree size: ");

    scanf("%d %d", &id, &node_num);
    T = Main_T.head;
    while (T != NULL) {
        if (T->id == id)
            break;
        T = T->next;
    }
    if (T != NULL) {
        printf("Create error, this binary tree is already
exists!\n");
    }
    else {
        int *pre_index = (int*)malloc(sizeof(int) * node_num);
        int *pre_definition = (int*)malloc(sizeof(int) *
node_num);
        int *in_index = (int*)malloc(sizeof(int) * node_num);
        int *in_definition = (int*)malloc(sizeof(int) *
node_num);
        printf("Please enter the preorder sequence: (Format:
index value)\n");
        for (int i = 0; i < node_num; i++) { scanf("%d %d",
&pre_index[i], &pre_definition[i]); }
    }
}

```

```

        printf("Please enter the inorder sequence: (Format: index
value)\n");
        for (int i = 0; i < node_num; i++) { scanf("%d %d",
&in_index[i], &in_defination[i]); }
        binary_tree new_T;
        create_bitree(&new_T,     pre_index,     pre_defination,
in_index, in_defination, node_num);
        printf("Create binary tree %d succeed!\n", id);
        new_T.id = id;
        new_T.next = Main_T.head;
        Main_T.head = &new_T;
    }

    printf("\n");
    break;

case 4:
    printf("/*\n");
    printf(" * Function Name: clear_bitree\n");
    printf(" * Module: Data structures\n");
    printf(" * Parameter: binary_tree *T\n");
    printf(" * Return: int(status)\n");
    printf(" * Use: clear the binary tree\n");
    printf(" */\n");
    printf("\n");
    printf("Then, enter the binary tree id: ");

    scanf("%d", &id);
    T = Main_T.head;
    while (T != NULL) {
        if (T->id == id)
            break;
        T = T->next;
    }
    if (T == NULL) {
        printf("Clear error, this binary tree is not exists!\n");
    }
    else {
        if (clear_bitree(T) == OK) {
            printf("Binary tree %d has been cleared!\n", id);
        }
        else {
            printf("Clear error, this binary tree is already
empty!\n");
        }
    }
}

```

```
        }

    printf("\n");
    break;

case 5:
    printf("/*\n");
    printf(" * Function Name: is_bitree_empty\n");
    printf(" * Module: Data structures\n");
    printf(" * Parameter: binary_tree T\n");
    printf(" * Return: int(true or false)\n");
    printf(" * Use: judge if the binary tree is empty\n");
    printf(" */\n");
    printf("\n");
    printf("Then, enter the binary tree id: ");

    scanf("%d", &id);
    T = Main_T.head;
    while (T != NULL) {
        if (T->id == id)
            break;
        T = T->next;
    }
    if (T == NULL) {
        printf("Error, this binary tree is not exists!\n");
    }
    else {
        if (is_bitree_empty(*T) == TRUE) {
            printf("Binary tree %d is empty!\n", id);
        }
        else {
            printf("Binary tree %d is not empty!\n", id);
        }
    }
}

printf("\n");
break;

case 6:
    printf("/*\n");
    printf(" * Function Name: bitree_depth\n");
    printf(" * Module: Data structures\n");
    printf(" * Parameter: binary_tree T\n");
```

```

printf(" * Return: int(the depth)\n");
printf(" * Use: get the depth of binary tree\n");
printf(" */\n");
printf("\n");
printf("Then, enter the binary tree id: ");

scanf("%d", &id);
T = Main_T.head;
while (T != NULL) {
    if (T->id == id)
        break;
    T = T->next;
}
if (T == NULL) {
    printf("Error, this binary tree is not exists!\n");
}
else {
    printf("The length of binary tree %d is %d.\n", id,
bitree_depth(*T));
}

printf("\n");
break;

case 7:
printf("/*\n");
printf(" * Function Name: bitree_root\n");
printf(" * Module: Data structures\n");
printf(" * Parameter: binary_tree T\n");
printf(" * Return: binary_tree_node*\n");
printf(" * Use: get the root of binary tree\n");
printf(" */\n");
printf("\n");
printf("Then, enter the binary tree id: ");

scanf("%d", &id);
T = Main_T.head;
while (T != NULL) {
    if (T->id == id)
        break;
    T = T->next;
}
if (T == NULL) {
    printf("Error, this binary tree is not exists!\n");
}

```

```

    }
else {
    binary_tree_node *the_root = bitree_root(*T);
    if (the_root == NULL) {
        printf("Error, this binary tree is empty!\n");
    }
    else {
        printf("Root of binary tree %d:\n", id);
        printf("Index: %d\n", the_root->index);
        printf("Value: %d\n", the_root->value);
    }
}

printf("\n");
break;

case 8:
printf("/*\n");
printf(" * Function Name: bitree_get_value\n");
printf(" * Module: Data structures\n");
printf(" * Parameter: binary_tree T, int index, int
value\n");
printf(" * Return: int(the value)\n");
printf(" * Use: get the value of a node\n");
printf(" */\n");
printf("\n");
printf("                                Format: id
index\n");
printf("Then, enter the binary tree id and node index: ");

scanf("%d %d", &id, &index);
T = Main_T.head;
while (T != NULL) {
    if (T->id == id)
        break;
    T = T->next;
}
if (T == NULL) {
    printf("Error, this binary tree is not exists!\n");
}
else {
    printf("Binary tree %d, Index %d\n", id, index);
    printf("The value is: %d\n", bitree_get_value(*T,
index));
}

```

```
}

printf("\n");
break;

case 9:
printf("/*\n");
printf(" * Function Name: bitree_set_value\n");
printf(" * Module: Data structures\n");
printf(" * Parameter: binary_tree *T, int index, int
value\n");
printf(" * Return: int(status)\n");
printf(" * Use: set the value of a node\n");
printf(" */\n");
printf("\n");
printf("Format: id index value\n");
printf("Then, enter the binary tree id, the node index and
the value: ");

scanf("%d %d %d", &id, &index, &value);
T = Main_T.head;
while (T != NULL) {
    if (T->id == id)
        break;
    T = T->next;
}
if (T == NULL) {
    printf("Error, this binary tree is not exists!\n");
}
else {
    if (bitree_set_value(T, index, value) == OK) {
        printf("Set the node %d's value succeed!\n", index);
    }
    else {
        printf("Error, the node %d is not found!\n", index);
    }
}

printf("\n");
break;

case 10:
printf("/*\n");
```

```

printf(" * Function Name: bitree_parent\n");
printf(" * Module: Data structures\n");
printf(" * Parameter: binary_tree T, int index\n");
printf(" * Return: binary_tree_node*\n");
printf(" * Use: get parent of a node\n");
printf(" */\n");
printf("\n");
printf("Format: id
index\n");
printf("Then, enter the binary tree id and node index: ");

scanf("%d %d", &id, &index);
T = Main_T.head;
while (T != NULL) {
    if (T->id == id)
        break;
    T = T->next;
}
if (T == NULL) {
    printf("Error, this binary tree is not exists!\n");
}
else {
    binary_tree_node *the_parent = bitree_parent(*T, index);
    if (the_parent == NULL) {
        printf("Error, the parent not found!\n");
    }
    else {
        printf("The parent of node %d\n", index);
        printf("Key: %d\n", the_parent->index);
        printf("Value: %d\n", the_parent->value);
    }
}

printf("\n");
break;

case 11:
printf("/*\n");
printf(" * Function Name: bitree_left_child\n");
printf(" * Module: Data structures\n");
printf(" * Parameter: binary_tree T, int index\n");
printf(" * Return: binary_tree_node*\n");
printf(" * Use: get left child of a node\n");
printf(" */\n");

```

```

        printf("\n");
        printf("                                         Format: id
index\n");
        printf("Then, enter the binary tree id and node index: ");

        scanf("%d %d", &id, &index);
        T = Main_T.head;
        while (T != NULL) {
            if (T->id == id)
                break;
            T = T->next;
        }
        if (T == NULL) {
            printf("Error, this binary tree is not exists!\n");
        }
        else {
            binary_tree_node *the_left_child = bitree_left_child(*T,
index);
            if (the_left_child == NULL) {
                printf("Error, the left child not found!\n");
            }
            else {
                printf("The left child of node %d\n", index);
                printf("Key: %d\n", the_left_child->index);
                printf("Value: %d\n", the_left_child->value);
            }
        }

        printf("\n");
        break;

case 12:
        printf("/*\n");
        printf(" * Function Name: bitree_right_child\n");
        printf(" * Module: Data structures\n");
        printf(" * Parameter: binary_tree T, int index\n");
        printf(" * Return: binary_tree_node*\n");
        printf(" * Use: get right child of a node\n");
        printf(" */\n");
        printf("\n");
        printf("                                         Format: id
index\n");
        printf("Then, enter the binary tree id and node index: ");

```

```

scanf("%d %d", &id, &index);
T = Main_T.head;
while (T != NULL) {
    if (T->id == id)
        break;
    T = T->next;
}
if (T == NULL) {
    printf("Error, this binary tree is not exists!\n");
}
else {
    binary_tree_node *the_right_child =
bitree_right_child(*T, index);
    if (the_right_child == NULL) {
        printf("Error, the right child not found!\n");
    }
    else {
        printf("The right child of node %d\n", index);
        printf("Key: %d\n", the_right_child->index);
        printf("Value: %d\n", the_right_child->value);
    }
}

printf("\n");
break;

case 13:
printf("/*\n");
printf(" * Function Name: bitree_left_sibling\n");
printf(" * Module: Data structures\n");
printf(" * Parameter: binary_tree T, int index\n");
printf(" * Return: binary_tree_node*\n");
printf(" * Use: get left sibling of a node\n");
printf(" */\n");
printf("\n");
printf("                                         Format: id
index\n");
printf("Then, enter the binary tree id and node index: ");

scanf("%d %d", &id, &index);
T = Main_T.head;
while (T != NULL) {
    if (T->id == id)
        break;

```

```

        T = T->next;
    }
    if (T == NULL) {
        printf("Error, this binary tree is not exists!\n");
    }
    else {
        binary_tree_node *the_left_sibling =
bitree_left_sibling(*T, index);
        if (the_left_sibling == NULL) {
            printf("Error, the left sibling not found!\n");
        }
        else {
            printf("The left sibling of node %d\n", index);
            printf("Key: %d\n", the_left_sibling->index);
            printf("Value: %d\n", the_left_sibling->value);
        }
    }

    printf("\n");
    break;

case 14:
    printf("/*\n");
    printf(" * Function Name: bitree_right_sibling\n");
    printf(" * Module: Data structures\n");
    printf(" * Parameter: binary_tree T, int index\n");
    printf(" * Return: binary_tree_node*\n");
    printf(" * Use: get right sibling of a node\n");
    printf(" */\n");
    printf("\n");
    printf("                                         Format: id
index\n");
    printf("Then, enter the binary tree id and node index: ");

    scanf("%d %d", &id, &index);
    T = Main_T.head;
    while (T != NULL) {
        if (T->id == id)
            break;
        T = T->next;
    }
    if (T == NULL) {
        printf("Error, this binary tree is not exists!\n");
    }
}

```

```

    else {
        binary_tree_node *the_right_sibling =
bitree_right_sibling(*T, index);
        if (the_right_sibling == NULL) {
            printf("Error, the right sibling not found!\n");
        }
        else {
            printf("The right sibling of node %d\n", index);
            printf("Key: %d\n", the_right_sibling->index);
            printf("Value: %d\n", the_right_sibling->value);
        }
    }

    printf("\n");
    break;

case 15:
printf("/*\n");
printf(" * Function Name: bitree_insert_child\n");
printf(" * Module: Data structures\n");
printf(" * Parameter: binary_tree *T, int index, int LorR,
binary_tree *C\n");
printf(" * Return: int(status)\n");
printf(" * Use: insert a child tree in a binary tree\n");
printf(" */\n");
printf("\n");
printf("Format: id index LorR child_id\n");
    printf("Then, enter the binary tree id, index, LorR, child
tree id: ");

    int child_id;
    scanf("%d %d %d %d", &id, &index, &LorR, &child_id);
    T = Main_T.head;
    while (T != NULL) {
        if (T->id == id)
            break;
        T = T->next;
    }
    if (T == NULL) {
        printf("Error, this binary tree is not exists!\n");
        printf("\n");
        break;
    }
}

```

```

binary_tree *C = Main_T.head;
C = Main_T.head;
if (C->id == child_id) {
    if (bitree_insert_child_tree(T, index, LorR, C) == OK)
{
        printf("Insert child tree %d succeed!\n", child_id);
        Main_T.head = Main_T.head->next;
        free(C);
    }
    else {
        printf("Insert error, please check your input!\n");
    }
    printf("\n");
    break;
}
if (bitree_insert_child_tree(T, index, LorR, C->next) ==
OK) {
    printf("Insert child tree %d succeed!\n", child_id);
    binary_tree *tmp = C->next;
    C->next = C->next->next;
    free(tmp);
}
else {
    printf("Insert error, please check your input!\n");
}
printf("\n");
break;

case 16:
printf("/*\n");
printf(" * Function Name: bitree_delete_child\n");
printf(" * Module: Data structures\n");
printf(" * Parameter: binary_tree *T, int index, int
LorR\n");
printf(" * Return: int(status)\n");
printf(" * Use: delete a child tree in a binary tree\n");
printf(" */\n");
printf("\n");
printf("                                     Format: id
index LorR\n");
printf("Then, enter the binary tree id, index and LorR: ");

scanf("%d %d %d", &id, &index, &LorR);
T = Main_T.head;

```

```

while (T != NULL) {
    if (T->id == id)
        break;
    T = T->next;
}
if (T == NULL) {
    printf("Error, this binary tree is not exists!\n");
}
else {
    if (bitree_delete_child_tree(T, index, L or R) == OK) {
        printf("Delete child tree succeed!\n");
    }
    else {
        printf("Delete error, please check your input!\n");
    }
}

printf("\n");
break;

case 17:
printf("/*\n");
printf(" * Function Name: bitree_preorder_traverse\n");
printf(" * Module: Data structures\n");
printf(" * Parameter: binary_tree T\n");
printf(" * Return: int(status)\n");
printf(" * Use: preorder traverse and print the values\n");
printf(" */\n");
printf("\n");
printf("Then, enter the binary tree id: ");

scanf("%d", &id);
T = Main_T.head;
while (T != NULL) {
    if (T->id == id)
        break;
    T = T->next;
}
if (T == NULL) {
    printf("Error, this binary tree is not exists!\n");
}
else {
    if (bitree_preorder_traverse(*T) == ERROR) {
        printf("Error, this binary tree is empty!\n");
    }
}

```

```
        }

    printf("\n");
    break;

case 18:
printf("/*\n");
printf(" * Function Name: bitree_inorder_traverse\n");
printf(" * Module: Data structures\n");
printf(" * Parameter: binary_tree T\n");
printf(" * Return: int(status)\n");
printf(" * Use: inorder traverse and print the values\n");
printf(" */\n");
printf("\n");
printf("Then, enter the binary tree id: ");

scanf("%d", &id);
T = Main_T.head;
while (T != NULL) {
    if (T->id == id)
        break;
    T = T->next;
}
if (T == NULL) {
    printf("Error, this binary tree is not exists!\n");
}
else {
    if (bitree_inorder_traverse(*T) == ERROR) {
        printf("Error, this binary tree is empty!\n");
    }
}

printf("\n");
break;

case 19:
printf("/*\n");
printf(" * Function Name: bitree_postorder_traverse\n");
printf(" * Module: Data structures\n");
printf(" * Parameter: binary_tree T\n");
printf(" * Return: int(status)\n");
printf(" * Use: postorder traverse and print the values\n");
printf(" */\n");
```

```
printf("\n");
printf("Then, enter the binary tree id: ");

scanf("%d", &id);
T = Main_T.head;
while (T != NULL) {
    if (T->id == id)
        break;
    T = T->next;
}
if (T == NULL) {
    printf("Error, this binary tree is not exists!\n");
}
else {
    if (bitree_postorder_traverse(*T) == ERROR) {
        printf("Error, this binary tree is empty!\n");
    }
}

printf("\n");
break;

case 20:
printf("/*\n");
printf(" * Function Name: bitree_levelorder_traverse\n");
printf(" * Module: Data structures\n");
printf(" * Parameter: binary_tree T\n");
printf(" * Return: int(status)\n");
printf(" * Use: levelorder traverse and print the
values\n");
printf(" */\n");
printf("\n");
printf("Then, enter the binary tree id: ");

scanf("%d", &id);
T = Main_T.head;
while (T != NULL) {
    if (T->id == id)
        break;
    T = T->next;
}
if (T == NULL) {
    printf("Error, this binary tree is not exists!\n");
}
```

```
else {
    if (bitree_levelorder_traverse(*T) == ERROR) {
        printf("Error, this binary tree is empty!\n");
    }
}

printf("\n");
break;

case 21:
T = Main_T.head;
while (T != NULL) {
    printf("ID: %d\n", T->id);
    T = T->next;
}

printf("\n");
break;

case 0:
printf("Thanks for using my demo system!\n");
break;

default:
    printf("You entered the wrong num, please re-enter the
num.\n");
    break;
}

save_data(&Main_T);
}

return 0;
}
```



## 附录 E 基于邻接表图实现的源程序

注：由于实验的编译文件 Makefile 使用 CMake 工具生成，故在这里附上 CMakeLists 文件

### E. 1 CMakeLists

```
# copyright Pan Yue
project(lab04)

add_library(graph my_graph.c)

add_library(libqueue mylibqueue.c)

add_executable(lab04 main.c)

target_link_libraries(lab04 libqueue graph)
```

### E. 2 my\_graph.h

```
/*
 * AUTHOR: Yue Pan
 * GITHUB: zxc479773533
 * E-MAIL: zxc479773533@gmail.com
 */

#ifndef PY_GRAPH_H
#define PY_GRAPH_H

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

#define TRUE 1
#define FALSE 0

#define OK 0
#define ERROR 1

#define DIRECTED_GRAPH 0
#define UNDIRECTED_GRAPH 1
#define DIRECTED_NET 2
```

```
#define UNDIRECTED_NET 3

// the struct of arc
typedef struct arc_node {
    int vertex;
    struct arc_node *next;
    int weight;
} arc_node;

// the struct of vertex
typedef struct vertex_node {
    int index;
    int value;
    arc_node *first_arc;
    struct vertex_node *next;
} vertex_node;

// the struct of graph
typedef struct graph {
    int id;
    vertex_node *first_vertex;
    int vertex_num;
    int arc_num;
    int kind;
    struct graph *next;
} graph;

// the struct for managin the graph
typedef struct Graph_main {
    graph *head;
} Graph_main;

// the struct to save the graph
typedef struct Graph_store {
    int id;
    int kind;
    int vertex_num;
    int arc_num;
    int *v_indexes;
    int *v_values;
    int *a_matrix;
} Graph_store;
```

```
// assist function to get a vertax's position
int get_vex_pos(graph *G, int index);

int creat_graph(graph *G, int kind, int v_num, int a_num, int
*v_indexs, int *v_values, int *a_matrix);
/*
 * Function Name: creat_graph
 * Module: Data structures
 * Parameter: graph &G, int v_num, int kind, int a_num, int *v_indexs,
int *v_values, int *a_matrix
 * Return: int(status)
 * Use: create a graph
 */

int destory_graph(graph *G);
/*
 * Function Name: destory_graph
 * Module: Data structures
 * Parameter: graph &G
 * Return: int(status)
 * Use: destory a graph
 */

int locate_vex(graph *G, int value);
/*
 * Function Name: locate_vex
 * Module: Data structures
 * Parameter: graph &G, int value
 * Return: int(the index)
 * Use: find a vertex in graph
 */

int get_vex_value(graph *G, int index);
/*
 * Function Name: get_vex_value
 * Module: Data structures
 * Parameter: graph &G, int index
 * Return: int(the value)
 * Use: find a value of a vertex
```

```
/*
int set_vex_value(graph *G, int index, int value);
/*
 * Function Name: set_vex_value
 * Module: Data structures
 * Parameter: graph &G, int index, int value
 * Return: int(statue)
 * Use: set a vertex's value
 */

int set_arc_weight(graph *G, int src_index, int dst_index, int weight);
/*
 * Function Name: set_arc_weight
 * Module: Data structures
 * Parameter: graph *G, int src_index, int dst_index, int weight
 * Return: int(statue)
 * Use: set an arc's weight
 */

vertex_node* first_adj_vex(graph *G, int index);
/*
 * Function Name: first_adj_vex
 * Module: Data structures
 * Parameter: graph &G, int index
 * Return: vertex_node*(the first vertex)
 * Use: get the fitst adjacency vertex
 */

vertex_node* next_adj_vex(graph *G, int index, int now_index);
/*
 * Function Name: next_adj_vex
 * Module: Data structures
 * Parameter: graph &G, int index, int now_index
 * Return: vertex_node*(the next vertex)
 * Use: get the next adjacency vertex
 */

int insert_vex(graph *G, int index, int value);
```

```
/*
 * Function Name: insert_vex
 * Module: Data structures
 * Parameter: graph &G, int index, int value
 * Return: int(statue)
 * Use: insert a vertex in graph
 */

int delete_vex(graph *G, int index);
/*
 * Function Name: delete_vex
 * Module: Data structures
 * Parameter: graph &G, int index
 * Return: int(statue)
 * Use: delete a vertex in graph
 */

int insert_arc(graph *G, int src_vex, int dst_vex, int weight);
/*
 * Function Name: insert_arc
 * Module: Data structures
 * Parameter: graph &G, int src_vex, int dst_vex, int weight
 * Return: int(statue)
 * Use: insert an arc in graph
 */

int delete_arc(graph *G, int src_vex, int dst_vex);
/*
 * Function Name: delete_arc
 * Module: Data structures
 * Parameter: graph &G, int src_vex, int dst_vex
 * Return: int(statue)
 * Use: delete an arc in graph
 */

int dfs_traverse(graph *G);
/*
 * Function Name: dfs_traverse
 * Module: Data structures
 * Parameter: graph &G
```

```

* Return: int(statue)
* Use: Depth_First Search traverse the graph
*/
int bfs_traverse(graph *G);
/*
 * Function Name: bfs_traverse
 * Module: Data structures
 * Parameter: graph &G
 * Return: int(statue)
 * Use: Broadht_First Search traverse the graph
*/
#endif // !PY_GRAPH_H

```

### E. 3 my\_graph.c

```

/*
 * AUTHOR: Yue Pan
 * GITHUB: zxc479773533
 * E-MAIL: zxc479773533@gmail.com
*/

#include "my_graph.h"
#include "mylibqueue.c"

// assist function to search a vertex in graph
vertex_node* search_graph(graph *G, int index) {
    vertex_node *a_node = G->first_vertex;
    while (a_node != NULL) {
        // find the vertex
        if (a_node->index == index)
            break;
        a_node = a_node->next;
    }
    return a_node;
}

// assist function to search an arc in graph
arc_node* search_arc(graph *G, int src_vex, int dst_vex) {
    vertex_node *src_node = search_graph(G, src_vex);
    if (src_node == NULL)

```

```

    return NULL;
    arc_node *an_arc = src_node->first_arc;
    while (an_arc != NULL) {
        // find the arc
        if (an_arc->vertex == dst_vex)
            break;
        an_arc = an_arc->next;
    }
    return an_arc;
}

int creat_graph(graph *G, int kind, int v_num, int a_num, int
*v_indexs, int *v_values, int *a_matrix) {
/*
 * Function Name: creat_graph
 * Module: Data structures
 * Parameter: graph&G, int kind, int v_num, int a_num, int *v_indexs,
int *v_values, int *a_matrix
 * Return: int(status)
 * Use: create a graph
*/
    int i, j;
    int status = OK;
    G->kind = kind;
    // insert vertexs
    for(i = 0; i < v_num; i++) {
        status = insert_vex(G, v_indexs[i], v_values[i]);
    }
    // insert arcs
    for (i = 0; i < v_num; i++) {
        for (j = 0; j < v_num; j++) {
            if (*(a_matrix + i * v_num + j) != 0) {
                insert_arc(G, v_indexs[i], v_indexs[j], *(a_matrix + i *
v_num + j));
            }
        }
    }
    G->vertex_num = v_num;
    G->arc_num = a_num;
    return status;
}

int destory_graph(graph *G) {

```

```
/*
 * Function Name: destory_graph
 * Module: Data structures
 * Parameter: graph &G
 * Return: int(status)
 * Use: destory a graph
 */

vertex_node *a_node = G->first_vertex;
// free nodes
while (a_node != NULL) {
    vertex_node *tmp_node = a_node;
    a_node = a_node->next;
    free(tmp_node);
}
if (G->first_vertex != NULL)
    return ERROR;
// free graph
free(G);
if (G != NULL)
    return ERROR;
return OK;
}

int locate_vex(graph *G, int value) {
/*
 * Function Name: locate_vex
 * Module: Data structures
 * Parameter: graph &G, int value
 * Return: int(the index)
 * Use: find a vertex in graph
 */
vertex_node *a_node = G->first_vertex;
while (a_node != NULL) {
    // find the vertex
    if (a_node->value == value)
        return a_node->index;
    a_node = a_node->next;
}
return 0;
}

int get_vex_value(graph *G, int index) {
```

```
/*
 * Function Name: get_vex_value
 * Module: Data structures
 * Parameter: graph &G, int index
 * Return: int(the value)
 * Use: find a value of a vertex
 */

vertex_node *a_node = G->first_vertex;
while (a_node != NULL) {
    // find the vertex
    if (a_node->index == index)
        return a_node->value;
    a_node = a_node->next;
}
return 0;
}

int set_vex_value(graph *G, int index, int value) {
/*
 * Function Name: set_vex_value
 * Module: Data structures
 * Parameter: graph &G, int index, int value
 * Return: int(statue)
 * Use: set a vertex's value
 */

vertex_node *a_node = G->first_vertex;
while (a_node != NULL) {
    // find the vertex
    if (a_node->index == index) {
        a_node->value = value;
        break;
    }
    a_node = a_node->next;
}
if (a_node == NULL)
    return ERROR;
else
    return OK;
}

int set_arc_weight(graph *G, int src_index, int dst_index, int
weight) {
```

```

/*
 * Function Name: set_arc_weight
 * Module: Data structures
 * Parameter: graph *G, int src_index, int dst_index, int weight
 * Return: int(statue)
 * Use: set an arc's weight
 */

arc_node *arc = search_arc(G, src_index, dst_index);
if (arc == NULL)
    return ERROR;
arc->weight = weight;
// if the graph is undirected, also set the weight of "dst" to
"src"
if (G->kind == UNDIRECTED_GRAPH || G->kind == UNDIRECTED_NET) {
    arc = search_arc(G, dst_index, src_index);
    arc->weight = weight;
}
return OK;
}

vertex_node* first_adj_vex(graph *G, int index) {
/*
 * Function Name: first_adj_vex
 * Module: Data structures
 * Parameter: graph &G, int index
 * Return: vertex_node*(the first vertex)
 * Use: get the fitst adjacency vertex
*/
vertex_node *a_node = G->first_vertex;
vertex_node *first_adj_vertex = NULL;
while (a_node != NULL) {
    // find the vertex
    if (a_node->index == index) {
        // search in graph
        if (a_node->first_arc != NULL)
            first_adj_vertex = search_graph(G,
a_node->first_arc->vertex);
        break;
    }
    a_node = a_node->next;
}
return first_adj_vertex;
}

```

```

}

vertex_node* next_adj_vex(graph *G, int index, int now_index) {
/*
 * Function Name: next_adj_vex
 * Module: Data structures
 * Parameter: graph &G, int index, int now_index
 * Return: vertex_node*(the next vertex)
 * Use: get the next adjacency vertex
*/
// the same as first_adj_index
if (now_index == index)
    return first_adj_vex(G, index);
// others
vertex_node *a_node = G->first_vertex;
vertex_node *next_adj_vertex = NULL;
while (a_node != NULL) {
    // find the vertex
    if (a_node->index == index) {
        arc_node *an_arc = a_node->first_arc;
        while (an_arc != NULL) {
            // find the now_index
            if (an_arc->vertex == now_index && an_arc->next != NULL)
{
                next_adj_vertex = search_graph(G,
an_arc->next->vertex);
                break;
            }
            an_arc = an_arc->next;
        }
        break;
    }
    a_node = a_node->next;
}
return next_adj_vertex;
}

int insert_vex(graph *G, int index, int value) {
/*
 * Function Name: insert_vex
 * Module: Data structures
 * Parameter: graph &G, int index, int value
 * Return: int(statue)

```

```

* Use: insert a vertex in graph
*/
if (search_graph(G, index) != NULL)
    return ERROR;
vertex_node *new_vex = (vertex_node*)malloc(sizeof(vertex_node));
if (new_vex == NULL)
    return ERROR;
// set new vertex
G->vertex_num++;
new_vex->index = index;
new_vex->value = value;
new_vex->first_arc = NULL;
new_vex->next = NULL;
// insert in the first
if (G->first_vertex == NULL)
    G->first_vertex = new_vex;
// insert in behind the last
else {
    vertex_node *a_node = G->first_vertex;
    while (a_node->next != NULL)
        a_node = a_node->next;
    a_node->next = new_vex;
}
return OK;
}

int delete_vex(graph *G, int index) {
/*
 * Function Name: delete_vex
 * Module: Data structures
 * Parameter: graph &G, int index
 * Return: int(statue)
 * Use: delete a vertex in graph
*/
if (search_graph(G, index) == NULL)
    return ERROR;
G->vertex_num--;

// delete the arc whose dst is index
vertex_node *a_node = G->first_vertex;
while (a_node != NULL) {

```

```

arc_node *an_arc = a_node->first_arc;
if (an_arc == NULL) {
    a_node = a_node->next;
    continue;
}
// if delete the first arc
if (an_arc->vertex == index) {
    a_node->first_arc = a_node->first_arc->next;
    free(an_arc);
    G->arc_num--;
    a_node = a_node->next;
    continue;
}
// find the arc whose dst is index
while (an_arc->next != NULL) {
    if (an_arc->next->vertex == index) {
        arc_node *tmp_arc = an_arc->next;
        an_arc->next = an_arc->next->next;
        free(tmp_arc);
        G->arc_num--;
        break;
    }
    an_arc = an_arc->next;
}
a_node = a_node->next;
}

// delete the arc whose src is index
a_node = G->first_vertex;
// if delete the first node
if (a_node->index == index) {
    // free the arcs
    arc_node *an_arc = a_node->first_arc;
    while (an_arc != NULL) {
        arc_node *tmp_arc = an_arc;
        an_arc = an_arc->next;
        free(tmp_arc);
        G->arc_num--;
    }
    // free the vertex
    G->first_vertex = G->first_vertex->next;
    free(a_node);
    return OK;
}

```

```

while (a_node->next != NULL) {
    // find the vertex to delete
    if (a_node->next->index == index) {
        vertex_node *tmp_node = a_node->next;
        // free the arcs
        arc_node *an_arc = tmp_node->first_arc;
        while (an_arc != NULL) {
            arc_node *tmp_arc = an_arc;
            an_arc = an_arc->next;
            free(tmp_arc);
            G->arc_num--;
        }
        // free the node
        a_node->next = a_node->next->next;
        free(tmp_node);
    }
    a_node = a_node->next;
}
return OK;
}

// assist function to add an arc in graph
void add_an_arc(vertex_node *src_vex, int dst_vex, int weight) {
    arc_node *new_arc = (arc_node*)malloc(sizeof(arc_node));
    new_arc->vertex = dst_vex;
    new_arc->weight = weight;
    new_arc->next = NULL;
    // insert in first
    if (src_vex->first_arc == NULL)
        src_vex->first_arc = new_arc;
    // insert behind the last
    else {
        arc_node *an_arc = src_vex->first_arc;
        while (an_arc->next != NULL)
            an_arc = an_arc->next;
        an_arc->next = new_arc;
    }
}

int insert_arc(graph *G, int src_vex, int dst_vex, int weight) {
/*
 * Function Name: insert_arc
 * Module: Data structures
 * Parameter: graph &G, int src_vex, int dst_vex, int weight

```

```

* Return: int(statue)
* Use: insert an arc in graph
*/
vertex_node *src_vex_node = search_graph(G, src_vex);
vertex_node *dst_vex_node = search_graph(G, dst_vex);
if (src_vex_node == NULL || dst_vex_node == NULL)
    return ERROR;
if (search_arc(G, src_vex, dst_vex) != NULL)
    return ERROR;
// if the graph is undirected, also add arc "dst to src"
if (G->kind == UNDIRECTED_GRAPH || G->kind == UNDIRECTED_NET) {
    add_an_arc(dst_vex_node, src_vex, weight);
    G->arc_num++;
}
add_an_arc(src_vex_node, dst_vex, weight);
G->arc_num++;
return OK;
}

// assist function to delete an arc
void del_an_arc(vertex_node *src_vex, int dst_vex) {
    // if delete first
    if (src_vex->first_arc->vertex == dst_vex) {
        arc_node *tmp_arc = src_vex->first_arc;
        src_vex->first_arc = src_vex->first_arc->next;
        free(tmp_arc);
    }
    // traverse and find the arc
    else {
        arc_node *an_arc = src_vex->first_arc;
        while (an_arc->next != NULL) {
            if (an_arc->next->vertex == dst_vex) {
                arc_node *tmp_arc = an_arc->next;
                an_arc->next = an_arc->next->next;
                free(tmp_arc);
            }
            an_arc = an_arc->next;
        }
    }
}

int delete_arc(graph *G, int src_vex, int dst_vex) {
/*

```

```

* Function Name: delete_arc
* Module: Data structures
* Parameter: graph &G, int src_vex, int dst_vex
* Return: int(statue)
* Use: delete an arc in graph
*/
vertex_node *src_vex_node = search_graph(G, src_vex);
vertex_node *dst_vex_node = search_graph(G, dst_vex);
if (src_vex_node == NULL || dst_vex_node == NULL)
    return ERROR;
if (search_arc(G, src_vex, dst_vex) == NULL)
    return ERROR;
// if the graph is undirected, also delete the arc "dst to src"
if (G->kind == UNDIRECTED_GRAPH || G->kind == UNDIRECTED_NET) {
    del_an_arc(dst_vex_node, src_vex);
    G->arc_num--;
}
del_an_arc(src_vex_node, dst_vex);
G->arc_num--;
return OK;
}

// assist function to get a vertex's position
int get_vex_pos(graph *G, int index) {
    int pos = 0;
    vertex_node *a_node = G->first_vertex;
    while (a_node->index != index) {
        pos++;
        a_node = a_node->next;
    }
    return pos;
}

// assist function to ephth_First Search traverse
void DFS(graph *G, vertex_node *vertex, int *visit, int pos) {
    visit[pos] = 1;
    printf("Index: %d, value %d\n", vertex->index, vertex->value);
    vertex_node *a_node = NULL;
    for (a_node = first_adj_vex(G, vertex->index); a_node != NULL;
         a_node = next_adj_vex(G, vertex->index, a_node->index)) {
        int pos = get_vex_pos(G, a_node->index);
        // if not visited, just visit it
        if (!visit[pos])

```

```
        DFS(G, a_node, visit, pos);
    }
}

int dfs_traverse(graph *G) {
/*
 * Function Name: dfs_traverse
 * Module: Data structures
 * Parameter: graph &G
 * Return: int(statue)
 * Use: Depth_First Search traverse the graph
 */

vertex_node *a_node = G->first_vertex;
if (a_node == NULL)
    return ERROR;
int pos = 0;
int *visit = (int*)malloc(sizeof(int) * G->vertex_num);
memset(visit, 0, sizeof(int) * G->vertex_num);
printf("Graph: %d\n", G->id);
printf("DFS traverse:\n");
printf("Vertex num: %d\n", G->vertex_num);
while (a_node != NULL) {
    if (!visit[pos])
        DFS(G, a_node, visit, pos);
    a_node = a_node->next;
    pos++;
}
return OK;
}

int bfs_traverse(graph *G) {
/*
 * Function Name: bfs_traverse
 * Module: Data structures
 * Parameter: graph &G
 * Return: int(statue)
 * Use: Broadht_First Search traverse the graph
 */

vertex_node *traverse_node = G->first_vertex;
vertex_node *node = NULL;
if (traverse_node == NULL)
    return ERROR;
```

```

int pos = 0;
int *visit = (int*)malloc(sizeof(int) * G->vertex_num);
memset(visit, 0, sizeof(int) * G->vertex_num);
LinkQueue my_node_queue;
InitQueue(&my_node_queue);
printf("Graph: %d\n", G->id);
printf("BFS traverse:\n");
printf("Vertex num: %d\n", G->vertex_num);
// use a queue to bfs traverse the graph
while (traverse_node != NULL) {
    if (!visit[pos]) {
        visit[pos] = 1;
        EnQueue(&my_node_queue, traverse_node);
        while (!IsEmpty(&my_node_queue)) {
            node = DeQueue(&my_node_queue);
            printf("Index: %d, value %d\n", node->index, node->value);
            vertex_node *a_node = NULL;
            for (a_node = first_adj_vex(G, node->index); a_node != NULL;
                 a_node = next_adj_vex(G, node->index, a_node->index)) {
                int next_pos = get_vex_pos(G, a_node->index);
                // if not visited, just visit it
                if (!visit[next_pos]) {
                    visit[next_pos] = 1;
                    EnQueue(&my_node_queue, a_node);
                }
            }
        }
        traverse_node = traverse_node->next;
        pos++;
    }
    return OK;
}

```

#### E. 4 main.h

```

/*
 * AUTHOR: Yue Pan
 * GITHUB: zxc479773533
 * E-MAIL: zxc479773533@gmail.com
 */

#ifndef PY_GRAPH_MAIN_H_

```

```
#define PY_GRAPH_MAIN_H_

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

void print_menu(void);
/*
 * Function Name: print_menu
 * Module: Main control
 * Parameter: None
 * Return: None
 * Use: print the menu
 */

void load_data(Graph_main *Main_G);
/*
 * Function Name: load_data
 * Module: Main control
 * Parameter: Graph_main *Main_T
 * Return: None
 * Use: load link list from database
 */

void save_data(Graph_main *Main_G);
/*
 * Function Name: save_data
 * Module: Main control
 * Parameter: Graph_main *Main_T
 * Return: None
 * Use: save link list from database
 */

#endif // !PY_GRAPH_MAIN_H_
```

## E.5 main.c

```
/*
 * AUTHOR: Yue Pan
 * GITHUB: zxc479773533
 * E-MAIL: zxc479773533@gmail.com
```

```

*/
#include "main.h"

void print_menu(void) {
/*
 * Function Name: print_menu
 * Module: Main control
 * Parameter: None
 * Return: None
 * Use: print the menu
 */

printf("+-----+-----+\n");
    printf("|           Welcome to panyue's graph demo system!\n"
|\n");
    printf("|           Here are some functions you can call:\n"
|\n");
    printf("|           1: creat_graph               2:\n"
|\n");
    printf("destory_graph           |           3: locate_vex               4:\n"
|\n");
    printf("get_vex_value           |           5: set_vex_value            6:\n"
|\n");
    printf("set_arc_weight          |           7: first_adj_vex          8: next_adj_vex\n"
|\n");
    printf("                   9: insert_vex           10: delete_vex\n"
|\n");
    printf("                   11: insert_arc          12: delete_arc\n"
|\n");
    printf("                   13: dfs_traverse        14: bfs_traverse\n"
|\n");
    printf("                   15: ls_graph           0: quit\n"
|\n");
    printf("Enter the number of the function and see the usage and\n"
call it!      |\n");
    printf("Enter      0      to      quit      the      demo      system.\n"
|\n");
}

```

```
printf("|  
|\n");  
printf("|Copyrite (C) 2017 Yue Pan  
|\n");  
printf("|Github:  
|\n");  
  
printf("-----+-----+\n");  
printf("\n");  
printf("Your choose: ");  
}  
  
void load_data(Graph_main *Main_G) {  
/*  
 * Function Name: load_data  
 * Module: Main control  
 * Parameter: Graph_main *Main_T  
 * Return: None  
 * Use: load link list from database  
 */  
  
// open data file  
FILE *fp = fopen("graphdb", "rb");  
if (fp == NULL)  
    return ;  
  
graph graphs, *a_graph = &graphs;  
a_graph->next = NULL;  
  
int size = 0xff;  
while (1) {  
    Graph_store def;  
    graph *tmp_graph = NULL;  
    size = fread(&def, sizeof(Graph_store), 1, fp);  
    if (size == 0)  
        break;  
  
    tmp_graph = (graph*)malloc(sizeof(graph));  
    memset(tmp_graph, 0, sizeof(graph));  
  
    def.v_indexes = (int*)malloc(sizeof(int) * def.vertex_num);  
    def.v_values = (int*)malloc(sizeof(int) * def.vertex_num);  
    def.a_matrix = (int*)malloc(sizeof(int) * def.vertex_num *  
        def.vertex_num);
```

```

def.vertex_num);
    fread(def.v_indexs, sizeof(int) * def.vertex_num, 1, fp);
    fread(def.v_values, sizeof(int) * def.vertex_num, 1, fp);
    fread(def.a_matrix, sizeof(int) * def.vertex_num * def.vertex_num, 1, fp);
creat_graph(tmp_graph, def.kind, def.vertex_num, def.arc_num,
def.v_indexs, def.v_values, def.a_matrix);

tmp_graph->id = def.id;
tmp_graph->next = NULL;
a_graph->next = tmp_graph;
a_graph = a_graph->next;
}

Main_G->head = graphs.next;

// close file
fclose(fp);
}

// assist function to save graph
void traverse_and_save(graph *G, int *indexs, int* values, int
*matrix, int size) {
    vertex_node *a_node = G->first_vertex;
    int pos = 0;
    while (a_node != NULL) {
        indexs[pos] = a_node->index;
        values[pos] = a_node->value;
        arc_node *an_arc = a_node->first_arc;
        while (an_arc != NULL) {
            *(matrix + pos * size + get_vex_pos(G, an_arc->vertex)) =
an_arc->weight;
            an_arc = an_arc->next;
        }
        pos++;
        a_node = a_node->next;
    }
}

void save_data(Graph_main *Main_G) {
/*
 * Function Name: save_data
 * Module: Main control
 * Parameter: Graph_main *Main_T

```

```

* Return: None
* Use: save link list from database
*/
// open data files
FILE *fp = fopen("graphdb", "wb");
if (fp == NULL)
    return ;

graph *a_graph = Main_G->head;
while (a_graph != NULL) {
    Graph_store def;
    def.id = a_graph->id;
    def.kind = a_graph->kind;
    def.vertex_num = a_graph->vertex_num;
    def.arc_num = a_graph->arc_num;
    def.v_indexs = (int*)malloc(sizeof(int) * def.vertex_num);
    def.v_values = (int*)malloc(sizeof(int) * def.vertex_num);
    def.a_matrix = (int*)malloc(sizeof(int) * def.vertex_num * def.vertex_num);
    memset(def.a_matrix, 0, sizeof(int) * def.vertex_num * def.vertex_num);
    traverse_and_save(a_graph, def.v_indexs, def.v_values,
def.a_matrix, def.vertex_num);
    fwrite(&def, sizeof(Graph_store), 1, fp);
    fwrite(def.v_indexs, sizeof(int) * def.vertex_num, 1, fp);
    fwrite(def.v_values, sizeof(int) * def.vertex_num, 1, fp);
    fwrite(def.a_matrix, sizeof(int) * def.vertex_num * def.vertex_num, 1, fp);
    a_graph = a_graph->next;
}
// close file
fclose(fp);
}

int main(void) {
// a variable to save the function choose
int function_choose = 0xff;

while (function_choose != 0) {

// print the menu
print_menu();

```

```

scanf("%d", &function_choose);
Graph_main Main_G;
Main_G.head = NULL;
load_data(&Main_G);

// some variable to save the parameters
int id, kind, v_num, a_num, index, value, src, dst, weight;
graph *G = NULL;

switch(function_choose) {
    case 1:
        printf("/*\n");
        printf(" * Function Name: creat_graph\n");
        printf(" * Module: Data structures\n");
        printf(" * Parameter: graph &G, int v_num, int kind,\n");
        printf(" * int a_num, int *v_indexes, int *v_values, int
*a_matrix\n");
        printf(" * Return: int(status)\n");
        printf(" * Use: create a graph\n");
        printf(" */\n");
        printf("\n");
        printf("Format: id kind v_num a_num\n");
        printf("Then, enter the graph id, kind, vertex num, arc
num : ");

        scanf("%d %d %d %d", &id, &kind, &v_num, &a_num);
        G = Main_G.head;
        while (G != NULL) {
            if (G->id == id)
                break;
            G = G->next;
        }
        if (G != NULL) {
            printf("Creat error, this graph is already exists!\n");
        }
        else {
            int *v_indexes = (int*)malloc(sizeof(int) * v_num);
            int *v_values = (int*)malloc(sizeof(int) * v_num);
            int *a_matrix = (int*)malloc(sizeof(int) * v_num *
v_num);
            printf("Please enter the node info (Format: index
value):\n");
            for (int i = 0; i < v_num; i++) { scanf("%d %d",

```

```

&v_indexes[i], &v_values[i]); }
    printf("please enter the adjacency matrix:\n");
    for (int i = 0; i < v_num; i++) {
        for (int j = 0; j < v_num; j++) { scanf("%d", a_matrix
+ i * v_num + j); }
    }
    graph new_G;
    creat_graph(&new_G, kind, v_num, a_num, v_indexes,
v_values, a_matrix);
    printf("Create graph %d succeed!\n", id);
    new_G.id = id;
    new_G.next = Main_G.head;
    Main_G.head = &new_G;
}

printf("\n");
break;

case 2:
printf("/*\n");
printf(" * Function Name: destory_graph\n");
printf(" * Module: Data structures\n");
printf(" * Parameter: graph &G\n");
printf(" * Return: int(status)\n");
printf(" * Use: destory a graph\n");
printf(" */\n");
printf("\n");
printf("Then, enter the graph id: ");

scanf("%d", &id);
G = Main_G.head;
if (G->id == id) {
    Main_G.head = Main_G.head->next;
    destory_graph(G);
    printf("Graph %d has been deleted!", id);
    printf("\n");
    break;
}
else {
    while (G->next != NULL) {
        if (G->next->id == id)
            break;
        G = G->next;
    }
}

```

```

    }

    if (G->next == NULL) {
        printf("Destory error, this graph is not exists!\n");
    }
    else {
        graph *to_be_del = G->next;
        G->next = G->next->next;
        destory_graph(to_be_del);
        printf("Graph %d has been deleted!\n", id);
    }

    printf("\n");
    break;
}

case 3:
printf("/*\n");
printf(" * Function Name: locate_vex\n");
printf(" * Module: Data structures\n");
printf(" * Parameter: graph *G, int value\n");
printf(" * Return: int(the index)\n");
printf(" * Use: find a vertex in graph\n");
printf(" */\n");
printf("\n");
printf("                                         Format: id\n");
value\n");
printf("Then, enter the graph id and vertex value: ");

scanf("%d %d", &id, &value);
G = Main_G.head;
while (G != NULL) {
    if (G->id == id)
        break;
    G = G->next;
}
if (G == NULL) {
    printf("Error, this graph is not exists!\n");
}
else {
    printf("The index is %d.\n", locate_vex(G, value));
}

printf("\n");
break;

```

```

case 4:
    printf("/*\n");
    printf(" * Function Name: get_vex_value\n");
    printf(" * Module: Data structures\n");
    printf(" * Parameter: graph *G, int index\n");
    printf(" * Return: int(the index)\n");
    printf(" * Use: find a value of a vertex\n");
    printf(" */\n");
    printf("\n");
    printf("                                         Format: id
index\n");
    printf("Then, enter the graph id and vertex index: ");

    scanf("%d %d", &id, &index);
    G = Main_G.head;
    while (G != NULL) {
        if (G->id == id)
            break;
        G = G->next;
    }
    if (G == NULL) {
        printf("Error, this graph is not exists!\n");
    }
    else {
        printf("The value of %d is %d.\n", index, get_vex_value(G,
index));
    }

    printf("\n");
    break;

case 5:
    printf("/*\n");
    printf(" * Function Name: set_vex_value\n");
    printf(" * Module: Data structures\n");
    printf(" * Parameter: graph &G, int index, int value\n");
    printf(" * Return: int(the index)\n");
    printf(" * Use: set a vertex's value\n");
    printf(" */\n");
    printf("\n");
    printf("                                         Format: id
index value\n");
    printf("Then, enter the graph id, vertex index and value:
");

```

```

scanf("%d %d %d", &id, &index, &value);
G = Main_G.head;
while (G != NULL) {
    if (G->id == id)
        break;
    G = G->next;
}
if (G == NULL) {
    printf("Error, this graph is not exists!\n");
}
else {
    if (set_vex_value(G, index, value) == OK) {
        printf("Set value %d of index %d succeed!\n", value,
index);
    }
    else {
        printf("Error, the index %d is not exists!\n", index);
    }
}

printf("\n");
break;

case 6:
printf("/*\n");
printf(" * Function Name: set_arc_weight\n");
printf(" * Module: Data structures\n");
printf(" * Parameter: graph *G, int src_index, int dst_index,
int weight\n");
printf(" * Return: int(the index)\n");
printf(" * Use: set an arc's weight\n");
printf(" */\n");
printf("\n");
printf("Format: id src_index dst_index weight\n");
printf("Then, enter the graph id, src and dst index, and
arc weight: ");

scanf("%d %d %d", &id, &src, &dst, &weight);
G = Main_G.head;
while (G != NULL) {
    if (G->id == id)
        break;

```

```

        G = G->next;
    }
    if (G == NULL) {
        printf("Error, this graph is not exists!\n");
    }
    else {
        if (set_arc_weight(G, src, dst, weight) == OK) {
            printf("Set the arc between %d and %d weight %d
succeed!\n", src, dst, weight);
        }
        else {
            printf("Set error, this arc is not exists!\n");
        }
    }

    printf("\n");
    break;

case 7:
    printf("/*\n");
    printf(" * Function Name: first_adj_vex\n");
    printf(" * Module: Data structures\n");
    printf(" * Parameter: graph &G, int index\n");
    printf(" * Return: vertex_node*(the first vertex)\n");
    printf(" * Use: get the fitst adjacency vertex\n");
    printf(" */\n");
    printf("\n");
    printf("Format: id index\n");
    printf("Then, enter the graph id and index: ");

    scanf("%d %d", &id, &index);
    G = Main_G.head;
    while (G != NULL) {
        if (G->id == id)
            break;
        G = G->next;
    }
    if (G == NULL) {
        printf("Error, this graph is not exists!\n");
    }
    else {
        vertex_node *first_adj_vertex = first_adj_vex(G, index);
        if (first_adj_vertex == NULL) {
            printf("Error, the required vertex is not exists!\n");
        }
    }
}

```

```

    }
    else {
        printf("The first adjacency vertex of %d is %d.\n",
index, first_adj_vertex->index);
    }
}

printf("\n");
break;

case 8:
printf("/*\n");
printf(" * Function Name: next_adj_vex\n");
printf(" * Module: Data structures\n");
printf(" * Parameter: graph &G, int index, int
now_index\n");
printf(" * Return: vertex_node*(the next vertex)\n");
printf(" * Use: get the next adjacency vertex\n");
printf(" */\n");
printf("\n");
printf("                                         Format: id
index now_index\n");
printf("Then, enter the graph id, index and now index: ");

int now_index;
scanf("%d %d %d", &id, &index, &now_index);
G = Main_G.head;
while (G != NULL) {
    if (G->id == id)
        break;
    G = G->next;
}
if (G == NULL) {
    printf("Error, this graph is not exists!\n");
}
else {
    vertex_node *next_adj_vertex = next_adj_vex(G, index,
now_index);
    if (next_adj_vertex == NULL) {
        printf("Error, the required vertex is not exists!\n");
    }
    else {
        printf("The nex adjacency vertex is %d.\n",
next_adj_vertex->index);
    }
}
}

```

```
        }

    }

    printf("\n");
    break;

case 9:
    printf("/*\n");
    printf(" * Function Name: insert_vex\n");
    printf(" * Module: Data structures\n");
    printf(" * Parameter: graph *G, int index, int value\n");
    printf(" * Return: int(statue)\n");
    printf(" * Use: insert a vertex in graph\n");
    printf(" */\n");
    printf("\n");
    printf("Format: id index\n");
    printf("value\n");
    printf("Then, enter the graph id, index and value: ");

    scanf("%d %d %d", &id, &index, &value);
    G = Main_G.head;
    while (G != NULL) {
        if (G->id == id)
            break;
        G = G->next;
    }
    if (G == NULL) {
        printf("Error, this graph is not exists!\n");
    }
    else {
        if (insert_vex(G, index, value) == OK) {
            printf("Insert vertex %d succeed!\n", index);
        }
        else {
            printf("Error, please check your input!\n");
        }
    }
}

printf("\n");
break;

case 10:
printf("/*\n");
printf(" * Function Name: delete_vex\n");
```

```

printf(" * Module: Data structures\n");
printf(" * Parameter: graph &G, int index\n");
printf(" * Return: int(statue)\n");
printf(" * Use: delete a vertex in graph\n");
printf(" */\n");
printf("\n");
printf("Format: id index\n");
printf("Then, enter the graph id and index: ");

scanf("%d %d", &id, &index);
G = Main_G.head;
while (G != NULL) {
    if (G->id == id)
        break;
    G = G->next;
}
if (G == NULL) {
    printf("Error, this graph is not exists!\n");
}
else {
    if (delete_vex(G, index) == OK) {
        printf("Delete vertex %d succeed!\n", index);
    }
    else {
        printf("Error, this vertex is not exists!\n");
    }
}

printf("\n");
break;

case 11:
printf("/*\n");
printf(" * Function Name: insert_arc\n");
printf(" * Module: Data structures\n");
printf(" * Parameter: graph &G, int src_vex, int dst_vex,
int weight\n");
printf(" * Return: int(statue)\n");
printf(" * Use: insert an arc in graph\n");
printf(" */\n");
printf("\n");
printf("Format: id src_index dst_index weight\n");
printf("Then, enter the graph id, src and dst index, and

```

```

weight: ");

    scanf("%d %d %d %d", &id, &src, &dst, &weight);
    G = Main_G.head;
    while (G != NULL) {
        if (G->id == id)
            break;
        G = G->next;
    }
    if (G == NULL) {
        printf("Error, this graph is not exists!\n");
    }
    else {
        if (insert_arc(G, src, dst, weight) == OK) {
            printf("Insert arc between %d and %d weight %d
succeed!\n", src, dst, weight);
        }
        else {
            printf("Error, please check your input!\n");
        }
    }
}

printf("\n");
break;

case 12:
printf("/*\n");
printf(" * Function Name: delete_arc\n");
printf(" * Module: Data structures\n");
printf(" * Parameter: graph &G, int src_vex, int
dst_vex\n");
printf(" * Return: int(statue)\n");
printf(" * Use: delete an arc in graph\n");
printf(" */\n");
printf("\n");
printf("                                             Format: id
src_index dst_index\n");
printf("Then, enter the graph id, src and dst index: ");

scanf("%d %d %d", &id, &src, &dst);
G = Main_G.head;
while (G != NULL) {
    if (G->id == id)
        break;

```

```

        G = G->next;
    }
    if (G == NULL) {
        printf("Error, this graph is not exists!\n");
    }
    else {
        if (delete_arc(G, src, dst) == OK) {
            printf("Delete the arc between %d and %d succeed!\n",
src, dst);
        }
        else {
            printf("Error, this arc is not exists!\n");
        }
    }

    printf("\n");
    break;

case 13:
    printf("/*\n");
    printf(" * Function Name: dfs_traverse\n");
    printf(" * Module: Data structures\n");
    printf(" * Parameter: graph &G\n");
    printf(" * Return: int(statue)\n");
    printf(" * Use: Depth_First Search traverse the graph\n");
    printf(" */\n");
    printf("\n");
    printf("Then, enter the graph id: ");

    scanf("%d", &id);
    G = Main_G.head;
    while (G != NULL) {
        if (G->id == id)
            break;
        G = G->next;
    }
    if (G == NULL) {
        printf("Error, this graph is not exists!\n");
    }
    else {
        if (dfs_traverse(G) == ERROR)
            printf("Error, this graph is empty!\n");
    }
}

```

```
printf("\n");
break;

case 14:
printf("/*\n");
printf(" * Function Name: bfs_traverse\n");
printf(" * Module: Data structures\n");
printf(" * Parameter: graph &G\n");
printf(" * Return: int(statue)\n");
printf(" * Use: Breadth_First Search traverse the
graph\n");
printf("*/\n");
printf("\n");
printf("Then, enter the graph id: ");

scanf("%d", &id);
G = Main_G.head;
while (G != NULL) {
    if (G->id == id)
        break;
    G = G->next;
}
if (G == NULL) {
    printf("Error, this graph is not exists!\n");
}
else {
    if (bfs_traverse(G) == ERROR)
        printf("Error, this graph is empty!\n");
}

printf("\n");
break;

case 15:
G = Main_G.head;
while (G != NULL) {
    printf("ID: %d\n", G->id);
    G = G->next;
}

printf("\n");
break;

case 0:
```

```
    printf("Thanks for using my demo system!\n");
    break;

default:
    printf("You entered the wrong num, please re-enter the
num.\n");
    break;
}

save_data(&Main_G);
}

return 0;
}
```

