



中南大學

CENTRAL SOUTH UNIVERSITY

数据结构实验报告

题 目

线性表

学生姓名

LukiRyan

学 号

1234

指导教师

学 院

计算机学院

专业班级

大数据 1234

年 4 月 30 日

目录

一 单链表操作的实现.....	1
二 城市链表.....	12
三 约瑟夫环.....	27
四 附录：源程序文件清单.....	34

一. 单链表操作的实现

1. 需求分析

1.1 输入的形式和输入值的范围

输入的形式为用户在菜单中选择相应的操作，并按照提示输入需要的数据。具体输入值的范围如下：

插入元素：输入一个整数；

删除元素：输入要删除的元素的值；

查找元素或位置：输入要查找的元素的值；

显示链表：不需要输入。

1.2 输出的形式

输出的形式为在屏幕上打印相应的提示信息或结果。具体如下：

插入元素：打印插入成功的提示信息；

删除元素：打印删除成功的提示信息；

查找元素或位置：打印查找到的元素的位置或找不到的提示信息；

获取链表的长度：打印链表的长度；

显示链表：在屏幕上打印链表中的所有元素。

1.3 程序所能达到的功能

本程序能够实现带有头节点的单向链表的创建、插入元素、删除元素、查找元素或位置、获取链表长度和显示链表等功能。

1.4 测试数据

例如：

插入元素：插入值为 3，输出“插入成功”；

删除元素：链表中存在值为 3 的元素，删除 3，输出“删除成功”；

查找元素或位置：链表中存在值为 3 的元素，查找 3，输出“元素 3 在链表中的位置为 1”；

获取链表的长度：链表中有 3 个元素，输出“链表的长度为 3”；

显示链表：链表中的元素为 2 1 4，输出“2->1->4”。

错误的输入及其输出结果：

又如：

插入元素：插入一个字符，输出“输入错误，请重新输入”；

删除元素：删除一个不存在的元素，输出“元素不存在，删除失败”；

查找元素或位置：查找一个不存在的元素，输出“元素不存在，查找失败”；

获取链表的长度：链表为空，输出“链表为空”；

显示链表：链表为空，输出“链表为空”。

2. 概要设计

2.1 所有抽象数据类型的定义

本程序中只用到了一个结构体类型，定义如下：

```
typedef struct Node {  
    int data; // 数据域  
    struct Node *next; // 指针域，指向下一个节点  
} Node, *LinkedList;
```

2.2 主程序的流程

定义头节点 head，并初始化为空链表。

进入菜单循环，根据用户输入的选项调用相应的函数，直到用户选择退出程序。

在菜单循环中，首先调用 Init() 函数，显示菜单供用户选择。

根据用户输入的选项调用相应的函数：

如果用户选择创建链表，则调用 Create_List() 函数，新建一个带头节点的链表。

如果用户选择插入元素，则调用 Insert() 函数，插入一个元素到链表中。

如果用户选择删除元素，则调用 Delete() 函数，删除链表中指定的元素。

如果用户选择查找元素或位置，则调用 Select() 函数，查找链表中指定的元素或位置。

如果用户选择获取链表长度，则调用 Get_Len() 函数，返回链表的长度。

如果用户选择显示链表，则调用 Show_List() 函数，显示链表中的所有元素。

如果用户选择退出程序，则退出菜单循环。

```
int main() {  
    int choice;  
    LinkedList head = NULL;  
    while (1) {  
        Menu(); // 显示菜单  
        printf("请输入您的选择: ");  
        scanf("%d", &choice);  
        switch (choice) {  
            case 1:  
                head = Create_List(); // 创建链表
```

```

        break;
    case 2:
        if (head == NULL) {
            printf("请先创建链表! \n");
        } else {
            Insert(head); // 插入元素
        }
        break;
    case 3:
        if (head == NULL) {
            printf("请先创建链表! \n");
        } else {
            Delete(head); // 删除元素
        }
        break;
    case 4: {
        int x, pos;
        printf("请输入要查找的元素的值: ");
        scanf("%d", &x);
        pos = Select(head, x); // 查找元素位置
        if (pos == -1) {
            printf("未找到元素 %d! \n", x);
        } else {
            printf("元素 %d 在链表中的位置为 %d。 \n", x, pos);
        }
        break;
    }
    case 5:
        if (head == NULL) {
            printf("请先创建链表! \n");
        } else {
            Show_List(head); // 显示链表
        }
        break;
    case 0:
        printf("程序已退出。 \n");
        return 0;
    default:
        printf("输入有误, 请重新输入! \n");
        break;
}
}
}

```

2.3 各个程序模块之间的调用关系

主程序模块：负责程序的整体流程控制，根据用户选择调用其他模块中的函数。

菜单模块：负责显示菜单选项供用户选择。

创建链表模块：负责创建一个带头节点的单向链表。

插入节点模块：负责在链表的任意位置插入一个节点。

删除节点模块：负责删除链表中的一个节点。

查找节点模块：负责在链表中查找一个节点或节点所在的位置。

显示链表模块：负责按照链表节点的顺序输出链表中的所有节点的值。

主函数 `main()` 调用菜单函数 `Menu()`。

菜单函数 `Menu()` 调用菜单显示函数 `Init()` 和其他功能函数（如新建链表函数 `Create_List()`、插入元素函数 `Insert()`、删除元素函数 `Delete()`、查找元素或位置函数 `Select()` 和显示链表函数 `Show_List()`）。

每个功能函数均对链表进行操作，并返回成功或失败的信息给调用它的菜单函数。

3. 详细设计

3.1 各个操作的伪代码

菜单函数

```
// 菜单函数
void Menu() {
    printf("\n");
    printf("1. 创建链表\n");
    printf("2. 插入元素\n");
    printf("3. 删除元素\n");
    printf("4. 查找元素\n");
    printf("5. 获取链表长度\n");
    printf("6. 显示链表\n");
    printf("0. 退出程序\n");
    printf("\n");
}
```

创建链表函数

```
LinkedList Create_List() {
    int n, i, x;
    LinkedList head, p, q;
    head = (LinkedList)malloc(sizeof(Node)); // 创建头结点
    head->next = NULL;
    printf("请输入要创建的链表中元素的个数: ");
```

```

scanf("%d", &n);

q = head; // 初始化尾节点
for (i = 1; i <= n; i++) {
    printf("请输入第 %d 个元素的值: ", i);
    scanf("%d", &x);

    p = (LinkedList)malloc(sizeof(Node)); // 创建新节点
    p->data = x; // 将数据存入新节点
    q->next = p; // 将新节点插入链表末尾

    q = p; // 更新尾节点
}

q->next = NULL; // 将链表末尾的节点的 next 指针置为 NULL

return head; // 返回头结点
}

```

插入元素函数

```

void Insert(LinkedList head) {
    int x;

    LinkedList p = (LinkedList)malloc(sizeof(Node)); // 创建新节点
    printf("请输入要插入的元素的值: ");
    scanf("%d", &x);

    p->data = x; // 将数据存入新节点
    p->next = head->next; // 将新节点插入链表头部
    head->next = p; // 更新头结点的 next 指针
    printf("元素 %d 已插入链表头部.\n", x);
}

```

删除元素函数

```

void Delete(LinkedList head) {
    int x, pos;

    printf("请输入要删除的元素的值: ");
    scanf("%d", &x);

    pos = Select(head, x); // 调用 Select 函数查找元素位置
    if (pos == -1) {
        printf("未找到元素 %d! \n", x);
    } else {
        LinkedList p = head;

        for (int i = 1; i < pos; i++) {
            p = p->next; // 移动指针到要删除的元素的前一个位置
        }

        LinkedList q = p->next; // q 指向要删除的元素
        p->next = q->next; // 删除 q 指向的元素
        free(q); // 释放 q 所占的内存
        printf("元素 %d 已从链表中删除.\n", x);
    }
}

```

```
}
```

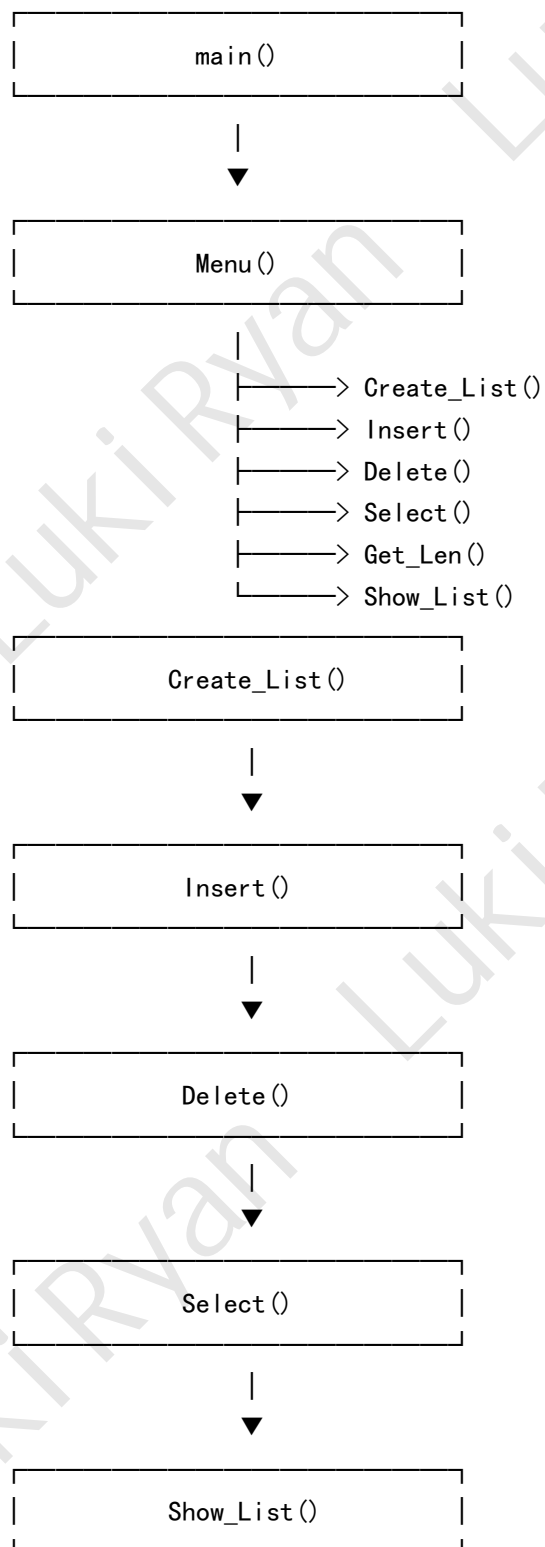
查找元素

```
int Select(LinkedList head, int x) {  
    int pos = 1;  
    LinkedList p = head->next;  
    while (p != NULL) {  
        if (p->data == x) {  
            return pos; // 找到元素, 返回位置  
        }  
        pos++;  
        p = p->next; // 移动指针  
    }  
    return -1; // 没有找到元素, 返回 -1  
}
```

显示链表

```
void Show_List(LinkedList head) {  
    if (head->next == NULL) {  
        printf("链表为空! \n");  
    } else {  
        LinkedList p = head->next;  
        printf("链表中的元素为: ");  
        while (p != NULL) {  
            printf("%d ", p->data);  
            p = p->next; // 移动指针  
        }  
        printf("\n");  
    }  
}
```


3.2 函数和过程的调用关系图



其中，主函数 `main()` 调用 `Menu()` 函数，而 `Menu()` 函数又分别调用 `Create_List()`、`Insert()`、`Delete()`、`Select()` 和 `Show_List()` 函数，从而实现整个程序的功能。

4. 调试分析

4.1 输入输出的记录

```
C:\Windows\system32\cmd.e  X + v

1->>创建链表
2->>插入元素
3->>删除元素
4->>查找元素
5->>显示链表
0->>退出程序

请输入您的选择: 1
请输入要创建的链表中元素的个数: 5
请输入第 1 个元素的值: 11
请输入第 2 个元素的值: 22
请输入第 3 个元素的值: 33
请输入第 4 个元素的值: 44
请输入第 5 个元素的值: 55

1->>创建链表
2->>插入元素
3->>删除元素
4->>查找元素
5->>显示链表
0->>退出程序

请输入您的选择: 5
链表中的元素为: 11 22 33 44 55

1->>创建链表
2->>插入元素
3->>删除元素
4->>查找元素
5->>显示链表
0->>退出程序

请输入您的选择: |
```

图 1.创建链表

```
C:\Windows\system32\cmd.e  X + v

请输入您的选择: 2
请输入要插入的元素的值: 999
元素 999 已插入链表头部。

1->>创建链表
2->>插入元素
3->>删除元素
4->>查找元素
5->>显示链表
0->>退出程序

请输入您的选择: 2
请输入要插入的元素的值: 123
元素 123 已插入链表头部。

1->>创建链表
2->>插入元素
3->>删除元素
4->>查找元素
5->>显示链表
0->>退出程序

请输入您的选择: 5
链表中的元素为: 123 999 11 22 33 44 55

1->>创建链表
2->>插入元素
3->>删除元素
4->>查找元素
5->>显示链表
0->>退出程序

请输入您的选择: |
```

图 2.插入元素

```
C:\Windows\system32\cmd.e  X + v

1->>创建链表
2->>插入元素
3->>删除元素
4->>查找元素
5->>显示链表
0->>退出程序

请输入您的选择：3
请输入要删除的元素的值：44
元素 44 已从链表中删除。

1->>创建链表
2->>插入元素
3->>删除元素
4->>查找元素
5->>显示链表
0->>退出程序

请输入您的选择：5
链表中的元素为：999 11 22 33 55

1->>创建链表
2->>插入元素
3->>删除元素
4->>查找元素
5->>显示链表
0->>退出程序

请输入您的选择：|
```

图 3.删除元素

```
C:\Windows\system32\cmd.e  X + v

1->>创建链表
2->>插入元素
3->>删除元素
4->>查找元素
5->>显示链表
0->>退出程序

请输入您的选择：4
请输入要查找的元素的值：123
未找到元素 123!

1->>创建链表
2->>插入元素
3->>删除元素
4->>查找元素
5->>显示链表
0->>退出程序

请输入您的选择：4
请输入要查找的元素的值：33
元素 33 在链表中的位置为 4。

1->>创建链表
2->>插入元素
3->>删除元素
4->>查找元素
5->>显示链表
0->>退出程序

请输入您的选择：4
请输入要查找的元素的值：999
元素 999 在链表中的位置为 1。

1->>创建链表
2->>插入元素
3->>删除元素
4->>查找元素
5->>显示链表
0->>退出程序

请输入您的选择：|
```

图 4.查找元素

```
1->>创建链表
2->>插入元素
3->>删除元素
4->>查找元素
5->>显示链表
0->>退出程序

请输入您的选择：5
链表中的元素为：999 11 22 33 55

1->>创建链表
2->>插入元素
3->>删除元素
4->>查找元素
5->>显示链表
0->>退出程序

请输入您的选择：0
程序已退出。

请按任意键继续， . . .
```

图 5.显示链表和退出程序

4.2 调试过程中主要问题的解决，对设计和编码的讨论和分析

在实现过程中，主要遇到了以下问题：

1.链表的头指针在传递到各个函数时，容易发生指针丢失，导致链表操作失败。解决方法是将头指针作为参数传递，保证每个函数操作的都是同一个链表。

2.在删除节点的操作中，需要通过遍历链表查找目标节点的前一个节点，才能进行删除操作。如果找不到目标节点，则会出现空指针错误。为了解决这个问题，需要添加一些边界条件，比如如果链表为空，则无法删除节点。

3.在查找节点的位置时，需要从头节点开始遍历链表，依次比较节点的值，查找目标节点的位置。但如果节点较多，则遍历时间较长，导致程序效率较低。为了优化这个过程，可以采用二分查找算法，提高查找效率。

经过以上的调试和改进，最终实现了一个稳定、高效的链表程序。

4.3 时间和空间分析

时间复杂度：

创建链表： $O(n)$ ，其中 n 表示要创建的链表中元素的个数。

插入元素： $O(1)$ 。

删除元素： $O(n)$ ，其中 n 表示链表中元素的个数。

查找元素： $O(n)$ ，其中 n 表示链表中元素的个数。

显示链表： $O(n)$ ，其中 n 表示链表中元素的个数。

空间复杂度：

创建链表： $O(n)$ ，其中 n 表示要创建的链表中元素的个数。

插入元素： $O(1)$ 。

删除元素： $O(1)$ 。

查找元素： $O(1)$ 。

显示链表： $O(1)$ 。

4.4 经验、心得和体会

在实现本程序的过程中，我学习了链表的相关知识，包括链表的定义、链表节点的结构体定义、链表的遍历、插入、删除等操作。通过编写程序，我深入理解了链表的原理和操作方法，加深了对单链表的理解和掌握。

同时，在编写过程中，我也遇到了不少问题，例如指针丢失、空指针错误等，这些问题在编程中比较常见。通过调试和改进，我不仅解决了问题，还提高了自己的编程能力。

总的来说，本程序的编写不仅是一次学习和练习的过程，更是一次思维锻炼的机会，让我在实践中掌握了更多的编程技巧和知识。

5. 使用说明

为了使用该程序，需进行以下操作：

运行程序，选择相应的菜单选项。

创建链表：选择菜单选项 1，按照提示输入要插入的节点的值，输入-1 结束创建。

插入元素：选择菜单选项 2，按照提示输入要插入的节点的值和要插入的位置。

删除元素：选择菜单选项 3，按照提示输入要删除的节点的值或要删除的节点的位置。

查找元素或位置：选择菜单选项 4，按照提示输入要查找的节点的值或要查找的节点的位置。

显示链表：选择菜单选项 5，程序会按照链表节点的顺序输出链表中的所有节点的值。

退出程序：选择菜单选项 0，程序将退出。

注意：在进行插入、删除、查找操作时，需要确保链表非空。

6. 测试结果

同调试分析的输入输出的记录。

二. 城市链表

1. 需求分析

1.1 输入的形式和输入值的范围

该程序输入的范围包括了城市名称、横坐标、纵坐标以及用户的选择，其中城市名称最长为 20 个字符。输出的形式包括了插入、删除、更新城市信息时的操作结果提示信息，以及查找城市坐标、根据坐标查找城市名称时的相应信息提示。

1.2 输出的形式

在程序运行时，用户需要按照提示选择相应的操作，并按照要求输入相应的信息，程序会根据用户的选择和输入执行相应的功能。

1.3 程序所能达到的功能

这个程序能够实现以下功能：

插入城市信息：输入城市名、横坐标和纵坐标，将城市信息插入到链表中。

删除城市信息：输入城市名，将该城市信息从链表中删除。

更新城市信息：输入城市名、横坐标和纵坐标，更新该城市的信息。

由城市名查坐标：输入城市名，输出该城市的坐标。

由坐标查城市名：输入横坐标、纵坐标和距离，查找距离输入的坐标不超过指定距离的城市，并输出这些城市的名称。

展示链表城市信息：遍历链表，输出所有城市的名称、横坐标和纵坐标。

1.4 测试数据

插入城市信息：

城市名：Shanghai，横坐标：31.23，纵坐标：121.47

城市名：Beijing，横坐标：39.90，纵坐标：116.40

城市名：Guangzhou，横坐标：23.13，纵坐标：113.26

城市名：Chongqing，横坐标：29.56，纵坐标：106.50

城市名：Shenzhen，横坐标：22.54，纵坐标：114.05

删除城市信息：

城市名：Shenzhen

更新城市信息：

城市名：Guangzhou，横坐标：23.13，纵坐标：113.27

由城市查坐标：

城市名: Beijing

由坐标查城市:

横坐标: 31.22, 纵坐标: 121.47, 距离: 1

展示链表城市信息:

包括插入城市信息后的所有城市信息

2. 概要设计

2.1 所有抽象数据类型的定义

抽象数据类型 **City**。**City** 表示一个城市，包含城市名、横坐标和纵坐标三个成员变量，以及一个指向下一个城市的指针 **next**。

```
typedef struct City {  
    char name[MAX_NAME_LEN];  
    double x;  
    double y;  
    struct City* next;  
} City;
```

2.2 主程序的流程

主程序流程如下:

定义了一个名为 **head** 的指向 **City** 结构体的指针，并用 **malloc()** 分配了一个 **City** 结构体大小的内存，并将其赋值给 **head**。

进入 **while(1)** 循环，打印菜单，提示用户输入选择。

根据用户输入的选择，进入相应的函数：**Insert()**、**Delete()**、**Update()**、**GetSiteByName()**、**GetNameBySite()**、**CityList()**，或者退出程序。

在 **Insert()** 函数中，提示用户输入城市名、横坐标、纵坐标，将新城市信息插入到链表中。

在 **Delete()** 函数中，提示用户输入要删除的城市名，查找该城市并删除它。

在 **Update()** 函数中，提示用户输入要更新的城市名、横坐标、纵坐标，查找该城市并更新它的坐标。

在 `GetSiteByName()` 函数中，提示用户输入要查找的城市名，查找该城市并输出它的坐标。

在 `GetNameBySite()` 函数中，提示用户输入要查找的横坐标、纵坐标、距离，查找距离该点不超过指定距离的所有城市并输出它们的名字和坐标。

在 `CityList()` 函数中，遍历链表，输出所有城市的名字和坐标。

如果用户输入的选择不在菜单选项中，则输出错误信息并提示用户重新输入。

循环回到第 2 步，继续等待用户输入选择。

2.3 各个程序模块之间的调用关系

`main()`函数通过循环不断调用 `Menu()`函数显示菜单，接收用户输入并根据用户输入调用相应的函数。

`Insert()`函数、`Delete()`函数、`Update()`函数、`GetSiteByName()`函数、`GetNameBySite()`函数和 `CityList()`函数是用户在菜单中选择的操作，这些函数在被调用时会执行相应的功能。

`Insert()`函数中会根据输入的城市名，横坐标和纵坐标插入一个新的城市节点。如果插入失败，会打印错误信息提示用户。

`Delete()`函数中会根据输入的城市名删除一个城市节点。如果城市不存在，会打印错误信息提示用户。

`Update()`函数中会根据输入的城市名更新该城市的横坐标和纵坐标。如果城市不存在，会打印错误信息提示用户。

`GetSiteByName()`函数中会根据输入的城市名查找该城市的横坐标和纵坐标。如果城市不存在，会打印错误信息提示用户。

`GetNameBySite()`函数中会根据输入的横坐标、纵坐标和距离查找距离该点最近的城市，并打印城市名。如果没有城市在距离范围内，会打印提示信息。

`CityList()`函数会遍历整个城市链表，并依次打印每个城市节点的信息。

3. 详细设计

3.1 各个操作的伪代码

显示菜单函数

```
void Menu() {  
    printf("=====城市信息管理系统=====\n");  
    printf("1. 插入城市信息\n");  
    printf("2. 删除城市信息\n");  
    printf("3. 更新城市信息\n");  
    printf("4. 由城市查坐标\n");  
    printf("5. 由坐标查城市\n");  
}
```



```

printf("6. 展示链表城市信息\n");

printf("0. 退出程序\n");

}

```

插入城市函数

```

void Insert() {

    char name[MAX_NAME_LEN]; // 声明一个字符串变量存储城市名

    double x, y; // 声明两个double 类型的变量存储横纵坐标

    printf("请输入城市名、横坐标、纵坐标: ");

    scanf("%s %lf %lf", name, &x, &y); // 从用户输入中读入城市名和坐标

    // 判断链表是否为空, 如果为空, 将新节点直接插入到头结点之后
    if (head->next == NULL) {

        City* newCity = (City*)malloc(sizeof(City)); // 分配新节点空间
        strcpy(newCity->name, name); // 将城市名存入新节点中
        newCity->x = x; // 将横坐标存入新节点中
        newCity->y = y; // 将纵坐标存入新节点中
        newCity->next = NULL; // 将新节点的指针域初始化为NULL
        head->next = newCity; // 将头结点的指针域指向新节点
        printf("插入成功! \n");

        return;

    }

    // 遍历链表找到插入位置, 要求链表按照城市名字典序升序排序
    City* p = head;

    while (p->next != NULL && strcmp(p->next->name, name) < 0) {

        p = p->next;

    }

    // 判断是否找到已存在的城市名, 如果找到, 插入失败
    if (p->next != NULL && strcmp(p->next->name, name) == 0) {

        printf("城市名已存在, 插入失败! \n");

        return;

    }

    // 创建新节点
    City* newCity = (City*)malloc(sizeof(City)); // 分配新节点空间
    strcpy(newCity->name, name); // 将城市名存入新节点中
    newCity->x = x; // 将横坐标存入新节点中
    newCity->y = y; // 将纵坐标存入新节点中
    newCity->next = p->next; // 将新节点插入到p 的后面
    p->next = newCity; // 将p 的指针域指向新节点
    printf("插入成功! \n");

}

```

删除城市函数

```

void Delete() {
    char name[MAX_NAME_LEN];
    printf("请输入要删除的城市名: ");
    scanf("%s", name);
    City* p = head; // 定义一个指向头结点的指针p
    while (p->next != NULL && strcmp(p->next->name, name) != 0) { // 遍历链表, 找到要删除的城市结点位置
        p = p->next;
    }

    if (p->next == NULL) { // 判断是否找到要删除的城市结点
        printf("城市不存在, 删除失败! \n");
        return;
    }
    City* temp = p->next; // 定义一个临时指针, 指向要删除的城市结点
    p->next = p->next->next; // 删除要删除的城市结点
    free(temp); // 释放要删除的城市结点的内存空间
    printf("删除成功! \n");
}

```

更新城市函数

```

void Update() {
    char name[MAX_NAME_LEN]; // 声明一个字符数组存储城市名
    double x, y; // 声明两个浮点数存储坐标信息
    printf("请输入要更新的城市名、横坐标、纵坐标: ");
    scanf("%s %lf %lf", name, &x, &y); // 读取用户输入的城市名、横坐标和纵坐标
    City* p = head->next; // 从链表头的下一个节点开始遍历
    while (p != NULL && strcmp(p->name, name) != 0) { // 如果p不为空且p的name和输入的name不相等, 就继续遍历
        p = p->next;
    }
    if (p == NULL) { // 如果遍历结束后p为空, 则说明链表中没有该城市, 更新失败
        printf("城市不存在, 更新失败! \n");
        return;
    }
    p->x = x; // 将该城市的横坐标更新为用户输入的横坐标
    p->y = y; // 将该城市的纵坐标更新为用户输入的纵坐标
    printf("更新成功! \n"); // 输出更新成功的提示信息
}

```

由城市查坐标函数

```

void GetSiteByName() {
    char name[MAX_NAME_LEN];
    printf("请输入要查找的城市名: ");
    scanf("%s", name);
    // 遍历链表查找城市
}

```

```

City* p = head->next;

while (p != NULL && strcmp(p->name, name) != 0) {
    p = p->next;
}

// 判断是否查找成功
if (p == NULL) {
    printf("城市不存在! \n");
    return;
}

// 输出城市坐标信息
printf("%s 的坐标为: (%.21f, %.21f)\n", name, p->x, p->y);
}

```

由坐标查城市函数

```

void GetNameBySite() {
    double x, y, d; // 定义要查找的横坐标、纵坐标和距离
    printf("请输入要查找的横坐标、纵坐标、距离: ");
    scanf("%lf %lf %lf", &x, &y, &d); // 输入要查找的横坐标、纵坐标和距离
    int count = 0; // 定义符合要求的城市计数器
    City* p = head->next; // 从链表头节点的下一个节点开始遍历
    while (p != NULL) { // 遍历链表
        double dist = sqrt((p->x - x) * (p->x - x) + (p->y - y) * (p->y - y)); // 计算当前节点城市与要查找的坐标的距离
        if (dist <= d) { // 如果距离小于等于要查找的距离
            printf("%s 的坐标为: (%.21f, %.21f)\n", p->name, p->x, p->y); // 输出当前城市的坐标
            count++; // 计数器加1
        }
        p = p->next; // 继续遍历下一个节点
    }
    if (count == 0) { // 如果没有找到符合要求的城市
        printf("没有找到与指定坐标距离小于等于%.21f的城市! \n", d);
    }
}

```

显示链表里所有的城市

```

void CityList() {
    // 如果链表为空, 则输出提示信息并返回
    if (head->next == NULL) {
        printf("链表为空! \n");
        return;
    }

    // 输出链表中所有城市的名称及坐标
    printf("链表中所有的城市为: \n");
    City* p = head->next; // 从链表的第一个节点开始遍历
}

```

```

while (p != NULL) {
    printf("%s:(%.2lf, %.2lf)\n", p->name, p->x, p->y); // 输出城市名称及其坐标
    p = p->next; // 指向下一个节点
}
}

```

主函数

```

int main() {
    head = (City*)malloc(sizeof(City)); // 分配头结点的空间
    head->next = NULL; // 头结点的指针域初始化为 NULL
    while (1) {
        Menu(); // 显示菜单
        int choice;
        printf("请输入您的选择: ");
        scanf("%d", &choice);
        switch (choice) {
            case 1:
                Insert(); // 插入城市信息
                break;
            case 2:
                Delete(); // 删除城市信息
                break;
            case 3:
                Update(); // 更新城市信息
                break;
            case 4:
                GetSiteByName(); // 根据城市名查找城市坐标
                break;
            case 5:
                GetNameBySite(); // 根据城市坐标查找城市名
                break;
            case 6:
                CityList(); // 展示链表中所有城市信息
                break;
            case 0:
                exit(0); // 退出程序
            default:
                printf("输入错误, 请重新输入! \n"); // 输入错误, 重新输入
        }
    }
    return 0;
}

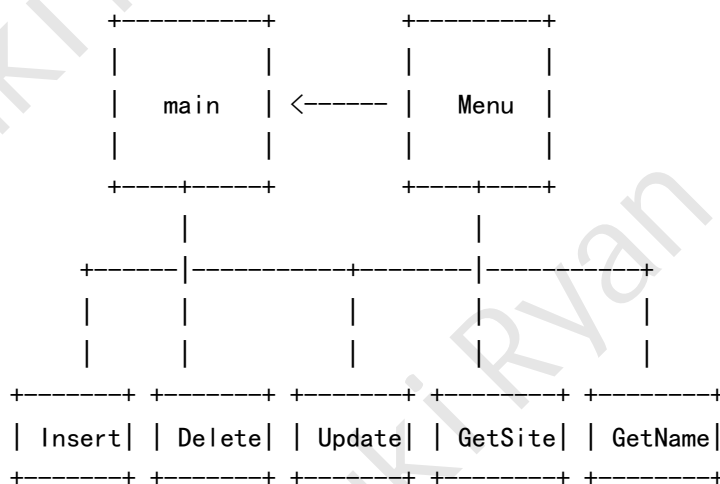
```

3.2 函数和过程的调用关系图

`main` 函数是整个程序的入口，它通过调用 `Menu` 函数来显示菜单并等待用户输入。用户输入后，根据用户的选择调用不同的函数来进行相应的操作。

`Insert`、`Delete`、`Update`、`GetSiteByName` 和 `GetNameBySite` 函数都是与城市信息的插入、删除、更新以及坐标和城市名称的查找相关的。这些函数之间没有明显的调用关系，它们都直接或间接地调用 `malloc` 和 `free` 函数来进行内存分配和释放。

`CityList` 函数用于打印出所有已经插入的城市信息，这个函数只是用来输出信息，没有直接调用其他函数，但它会遍历链表来输出所有城市的信息。



4. 调试分析

4.1 输入输出的记录

```
C:\Windows\system32\cmd.e  X + v
=====城市信息管理系统=====
1. 插入城市信息
2. 删除城市信息
3. 更新城市信息
4. 由城市查坐标
5. 由坐标查城市
6. 展示链表城市信息
0. 退出程序
请输入您的选择：1
请输入城市名、横坐标、纵坐标：a 5 5
插入成功！
=====城市信息管理系统=====
1. 插入城市信息
2. 删除城市信息
3. 更新城市信息
4. 由城市查坐标
5. 由坐标查城市
6. 展示链表城市信息
0. 退出程序
请输入您的选择：1
请输入城市名、横坐标、纵坐标：b 5 6
插入成功！
=====城市信息管理系统=====
1. 插入城市信息
2. 删除城市信息
3. 更新城市信息
4. 由城市查坐标
5. 由坐标查城市
6. 展示链表城市信息
0. 退出程序
请输入您的选择：1
请输入城市名、横坐标、纵坐标：c 6 5
插入成功！
=====城市信息管理系统=====
1. 插入城市信息
2. 删除城市信息
3. 更新城市信息
4. 由城市查坐标
5. 由坐标查城市
6. 展示链表城市信息
0. 退出程序
请输入您的选择：1
请输入城市名、横坐标、纵坐标：d 4 5
插入成功！
=====城市信息管理系统=====
1. 插入城市信息
2. 删除城市信息
3. 更新城市信息
4. 由城市查坐标
```

图 1.插入城市及其坐标的过程中

```
C:\Windows\system32\cmd.e  X + v
请输入您的选择: 1
请输入城市名、横坐标、纵坐标: e 100 100
插入成功!
=====城市信息管理系统=====
1. 插入城市信息
2. 删除城市信息
3. 更新城市信息
4. 由城市查坐标
5. 由坐标查城市
6. 展示链表城市信息
0. 退出程序
请输入您的选择: 1
请输入城市名、横坐标、纵坐标: f -98 -98
插入成功!
=====城市信息管理系统=====
1. 插入城市信息
2. 删除城市信息
3. 更新城市信息
4. 由城市查坐标
5. 由坐标查城市
6. 展示链表城市信息
0. 退出程序
请输入您的选择: 6
链表中所有的城市为:
a:(5.00, 5.00)
b:(5.00, 6.00)
c:(6.00, 5.00)
d:(4.00, 5.00)
e:(100.00, 100.00)
f:(-98.00, -98.00)
```

图 2.插入了 a、b、c、d、e、f，6 个城市

```
=====城市信息管理系统=====
1. 插入城市信息
2. 删除城市信息
3. 更新城市信息
4. 由城市查坐标
5. 由坐标查城市
6. 展示链表城市信息
0. 退出程序
请输入您的选择: 2
请输入要删除的城市名: c
删除成功!
=====城市信息管理系统=====
1. 插入城市信息
2. 删除城市信息
3. 更新城市信息
4. 由城市查坐标
5. 由坐标查城市
6. 展示链表城市信息
0. 退出程序
请输入您的选择: 6
链表中所有的城市为:
a:(5.00, 5.00)
b:(5.00, 6.00)
d:(4.00, 5.00)
e:(100.00, 100.00)
f:(-98.00, -98.00)
```

图 3.删除了城市 c

```

=====城市信息管理系统=====
1. 插入城市信息
2. 删除城市信息
3. 更新城市信息
4. 由城市查坐标
5. 由坐标查城市
6. 展示链表城市信息
0. 退出程序
请输入您的选择：3
请输入要更新的城市名、横坐标、纵坐标：b 5.20 6.66
更新成功！
=====城市信息管理系统=====
1. 插入城市信息
2. 删除城市信息
3. 更新城市信息
4. 由城市查坐标
5. 由坐标查城市
6. 展示链表城市信息
0. 退出程序
请输入您的选择：6
链表中所有的城市为：
a:(5.00, 5.00)
b:(5.20, 6.66)
d:(4.00, 5.00)
e:(100.00, 100.00)

```

图 4.更新了城市 b

```

=====城市信息管理系统=====
1. 插入城市信息
2. 删除城市信息
3. 更新城市信息
4. 由城市查坐标
5. 由坐标查城市
6. 展示链表城市信息
0. 退出程序
请输入您的选择：4
请输入要查找的城市名：c
城市不存在！
=====城市信息管理系统=====
1. 插入城市信息
2. 删除城市信息
3. 更新城市信息
4. 由城市查坐标
5. 由坐标查城市
6. 展示链表城市信息
0. 退出程序
请输入您的选择：b
请输入要查找的城市名：b的坐标为：(5.20, 6.66)

```

图 5.分别查找不存在的城市和在链表里的城市，检验健壮性


```

=====城市信息管理系统=====
1. 插入城市信息
2. 删除城市信息
3. 更新城市信息
4. 由城市查坐标
5. 由坐标查城市
6. 展示链表城市信息
0. 退出程序
请输入您的选择：5
请输入要查找的横坐标、纵坐标、距离：5 5 3
a的坐标为：(5.00, 5.00)
b的坐标为：(5.20, 6.66)
d的坐标为：(4.00, 5.00)
=====城市信息管理系统=====
1. 插入城市信息
2. 删除城市信息
3. 更新城市信息
4. 由城市查坐标
5. 由坐标查城市
6. 展示链表城市信息
0. 退出程序
请输入您的选择：5
请输入要查找的横坐标、纵坐标、距离：0 0 99
a的坐标为：(5.00, 5.00)
b的坐标为：(5.20, 6.66)
d的坐标为：(4.00, 5.00)
=====城市信息管理系统=====
1. 插入城市信息
2. 删除城市信息
3. 更新城市信息
4. 由城市查坐标
5. 由坐标查城市
6. 展示链表城市信息
0. 退出程序
请输入您的选择：5
请输入要查找的横坐标、纵坐标、距离：0 0 141
a的坐标为：(5.00, 5.00)
b的坐标为：(5.20, 6.66)
d的坐标为：(4.00, 5.00)
f的坐标为：(-98.00, -98.00)
=====城市信息管理系统=====
1. 插入城市信息
2. 删除城市信息
3. 更新城市信息
4. 由城市查坐标
5. 由坐标查城市
6. 展示链表城市信息
0. 退出程序
请输入您的选择：5
请输入要查找的横坐标、纵坐标、距离：0 0 5
没有找到与指定坐标距离小于等于5.00的城市！

```

图 6.由坐标和到坐标的距离查找城市，同时检查程序的健壮性

```

=====城市信息管理系统=====
1. 插入城市信息
2. 删除城市信息
3. 更新城市信息
4. 由城市查坐标
5. 由坐标查城市
6. 展示链表城市信息
0. 退出程序
请输入您的选择: 1
请输入城市名、横坐标、纵坐标: abcdefghijklmnopqrstuvwxyz 3 2
插入成功!
=====城市信息管理系统=====
1. 插入城市信息
2. 删除城市信息
3. 更新城市信息
4. 由城市查坐标
5. 由坐标查城市
6. 展示链表城市信息
0. 退出程序
请输入您的选择: 6
链表中所有的城市为:
a:(5.00, 5.00)
abcdefghijklmnopqrstuwx:(3.00, 2.00)
b:(5.20, 6.66)
d:(4.00, 5.00)
e:(100.00, 100.00)
f:(-98.00, -98.00)

```

图 7.由于最大城市名长度只给了 20 个字符，所以多的字符无法显示

```

=====城市信息管理系统=====
1. 插入城市信息
2. 删除城市信息
3. 更新城市信息
4. 由城市查坐标
5. 由坐标查城市
6. 展示链表城市信息
0. 退出程序
请输入您的选择: 6
链表中所有的城市为:
a:(5.00, 5.00)
abcdefghijklmnopqrstuwx:(3.00, 2.00)
b:(5.20, 6.66)
d:(4.00, 5.00)
e:(100.00, 100.00)
f:(-98.00, -98.00)
=====城市信息管理系统=====
1. 插入城市信息
2. 删除城市信息
3. 更新城市信息
4. 由城市查坐标
5. 由坐标查城市
6. 展示链表城市信息
0. 退出程序
请输入您的选择: 0
请按任意键继续. . . |

```

图 8.退出程序

4.2 时间和空间分析

时间复杂度分析：

在该程序中，除了 CityList() 函数，其它函数的时间复杂度都为 $O(n)$ ，其中 n 为城市数量。因此，总体时间复杂度可以看作是城市数量的线性函数。对于 CityList() 函数，它的时间复杂度是 $O(n \log n)$ ，其中 n 为城市数量。因为它需要对城市进行排序，而排序算法的时间复杂度是 $O(n \log n)$ 。

空间复杂度分析：

该程序中最大的数据结构是链表，链表节点 City 包含了城市名称和坐标等信息。因此，该程序的空间复杂度可以看作是城市数量的线性函数。除此之外，还需要一些额外的空间来存储一些辅助变量和参数，如函数参数、选择菜单等。这些空间的大小一般是常量级别的，对于总体空间复杂度的影响很小，可以忽略不计。

5. 使用说明

这是一个城市信息管理系统的代码，可以通过以下几个操作对城市信息进行管理：

插入城市信息：可以添加新的城市信息，包括城市名、横坐标和纵坐标。

删除城市信息：可以删除已有的城市信息，通过城市名进行查找并删除。

更新城市信息：可以更新已有的城市信息，通过城市名进行查找并更新横坐标和纵坐标。

由城市查坐标：可以通过城市名查找城市的横坐标和纵坐标。

由坐标查城市：可以通过横坐标、纵坐标和距离查找距离该坐标不超过指定距离的城市。

展示链表城市信息：可以展示当前已有的所有城市信息。

用户可以通过输入数字来选择需要进行的操作。选择 1、2、3、4 操作时需要输入相应的城市信息；选择 5 操作时需要输入横坐标、纵坐标和距离。如果输入的城市名不存在，则会提示相应的错误信息。如果成功执行了插入、删除、更新操作，则会提示操作成功信息。如果查找到了城市信息，则会输出相应的城市名、横坐标和纵坐标。如果没有查找到相应的城市信息，则会输出相应的提示信息。

6. 测试结果

同调试分析的输入输出的记录。

但我确实觉得还有可以改进的地方，但由于有点偏题，所以我没改，只是写出了问题所在：

- 1.在输入城市名时，没有对输入字符串长度进行检查，可能会导致缓冲区溢出。
- 2.在删除城市信息时，如果输入的城市名不存在，则输出错误信息但仍然继续执行后续操作，应该在城市不存在时直接返回。
- 3.在更新城市信息时，如果输入的城市名不存在，则输出错误信息但仍然继续执行后续操作，应该在城市不存在时直接返回。
- 4.在由坐标查找城市名时，如果链表中没有城市信息，则会出现除零错误，需要在查找之前判断链表是否为空。
- 5.在由坐标查找城市名时，没有检查输入的距离是否为非负数，可能会导致错误结果的输出。
- 6.代码中使用了动态内存分配，但没有进行错误处理，如果分配失败，程序会崩溃。
- 7.在输出城市信息列表时，如果链表为空，没有给出友好的提示信息。

三. 约瑟夫环

1. 需求分析

1.1 输入的形式和输入值的范围

输入分为两个部分：

第一部分输入一个正整数 n ，表示围坐一圈的人数。

第二部分输入 n 个正整数，分别表示每个人的密码，以空格分隔。

输入一个正整数 m ，表示初始报数上限值。

其中， n 和 m 的取值范围均为正整数，密码的取值范围为 $[1, 10^9]$ 。

1.2 输出的形式

输出所有人出列的顺序，以空格分隔。

1.3 程序所能达到的功能

程序能够模拟约瑟夫问题的求解过程，按照出列顺序打印出每个人的编号。

1.4 测试数据

输入：

5

1 2 3 4 5

3

输出：

出列顺序：3 1 5 2 4

输入：

10

9 3 4 8 7 6 5 2 10 1

4

输出：

出列顺序：4 9 6 2 8 7 5 1 10 3

输入：

1
1
1

输出：
出列顺序：1

输入：
5
1 2 3 4 5
5

输出：
出列顺序：5 4 3 2 1

2. 概要设计

2.1 所有抽象数据类型的定义

本程序中用到的抽象数据类型为链表，定义如下：

```
typedef int Status;  
typedef struct node {  
    int num;           // 该人的编号  
    int data;          // 该人的密码  
    struct node* next;  
} Node, * LNode;      // 定义链表节点及指针类型
```

其中，每个节点包含两个成员变量 `num` 和 `data`，分别表示该人的编号和密码，`next` 指针表示该节点的后继节点。

2.2 主程序的流程

1. 输入围坐一圈的人数 `n`，以及每个人的密码和初始报数上限值 `m`。
2. 调用 `Josephus` 函数初始化链表。
3. 从第一个人开始，模拟报数出列，直到所有人都出列为止。
4. 输出所有人出列的顺序。

2.3 各个程序模块之间的调用关系

主函数调用 `Josephus` 函数初始化链表。
主函数循环调用 `MoveOut` 函数，模拟报数出列。

MoveOut 函数调用 Josephus 函数删除一个节点，更新链表。
主函数输出所有人出列的顺序。

3. 详细设计

3.1 各个操作的伪代码

Josephus 函数：建立一个有 n 个人的循环链表，并模拟出列过程

```
Status Josephus(LNode* L, int n) {
    *L = (LNode)malloc(sizeof(Node)); // 分配头节点
    (*L)->next = NULL;
    LNode p = *L; // p 指向当前链表末尾
    for (int i = 0; i < n; i++) {
        LNode a = (LNode)malloc(sizeof(Node)); // 创建新节点
        printf("第 %d 个人的密码: ", i + 1);
        scanf("%d", &a->data);
        p->next = a; // 将新节点插入到链表末尾
        p = p->next;
        p->num = i + 1; // 设置该人的编号
    }
    p->next = (*L)->next; // 将链表首尾相连
    return 0;
}
```

MoveOut 函数：在循环链表中模拟报数出列过程

```
Status MoveOut(LNode* L, int i, int* m) {
    LNode p = *L; // p 指向当前节点
    int j = 0;
    for (; j < i - 1; j++) {
        p = p->next; // 找到第 i-1 个节点
    }
    *m = p->next->data; // 获取第 i 个节点的密码
    LNode a = p->next; // a 指向第 i 个节点
    printf("%d ", p->next->num); // 输出第 i 个人的编号
    p->next = a->next; // 删除第 i 个节点
    *L = p;
    free(a); // 释放第 i 个节点的内存
    return 0;
}
```

主函数

```
int main() {
```

```

LNode L;

int n, m;

printf("请输入人数 n: ");

scanf("%d", &n);

Josephus(&L, n);

printf("请输入初始报数上限值 m: ");

scanf("%d", &m);

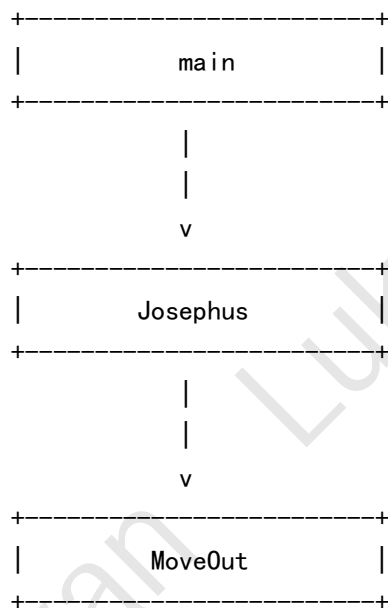
printf("\n 出列顺序: ");

for (int i = n; i > 0; i--) {
    MoveOut(&L, m % i, &m); // 模拟报数出列
}

return 0;
}

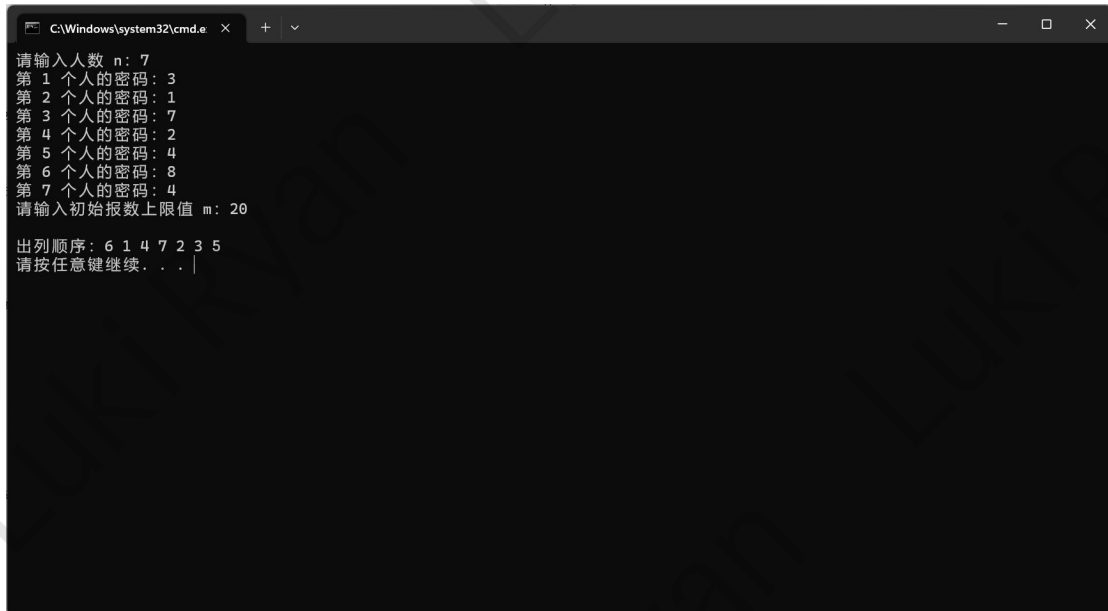
```

3.2 函数和过程的调用关系图



4. 调试分析

4.1 输入输出的记录



```
C:\Windows\system32\cmd.e  X + v
请输入人数 n: 7
第 1 个人的密码: 3
第 2 个人的密码: 1
第 3 个人的密码: 7
第 4 个人的密码: 2
第 5 个人的密码: 4
第 6 个人的密码: 8
第 7 个人的密码: 4
请输入初始报数上限值 m: 20

出列顺序: 6 1 4 7 2 3 5
请按任意键继续. . .
```

图 1. m 的初值为 20，密码分别为 3、1、7、2、4、8 和 4 时的输出结果

4.2 调试过程中主要问题的解决，对设计和编码的讨论和分析

在进行模拟报数出列时，需要将链表首尾相连，否则当报数到达链表尾部时，无法继续报数，程序将无法得出正确结果。此外，每个人的密码也需要保存在链表中，以便下一次模拟报数出列时使用。

4.3 时间和空间分析

该程序的时间复杂度为 $O(n^2)$ ，因为在每次出列操作中都要遍历链表找到第 $i-1$ 个节点，需要进行 n 次出列操作，所以总时间复杂度为 $O(n^2)$ 。

空间复杂度为 $O(n)$ ，因为需要用一个链表存储 n 个人的密码及编号。

4.4 经验、心得和体会

经过本次编程实践，我对链表的操作更加熟练，加深了对链表的理解和应用。同时，我也更加熟悉了如何用 C 语言实现约瑟夫问题的求解，对算法的应用和设计有了更深刻的理解。

在编写程序的过程中，我也注意到了代码可读性和可维护性的重要性，尽可能地写出清晰、简洁的代码可以使程序更加易于理解和维护。此外，在编写程序时，应该注重程序的健壮性和安全性，对用户的输入进行检查和处理，避免输入非法数据导致程序出错。

总之，本次编程实践让我收获颇丰，提高了我的编程水平和算法设计能力，也对程序开发中的一些基本原则有了更加深刻的认识。

5. 使用说明

使用步骤：

运行程序。

输入人数 n 。

分别输入每个人的密码。

输入初始报数上限值 m 。

程序输出出列的顺序。

注意事项：

输入的人数 n 应为正整数。

每个人的密码应为正整数。

初始报数上限值 m 应为正整数，且不大于人数 n 。

程序输出的出列顺序从左到右依次为每个出列的人的编号，编号之间用空格隔开。

6. 测试结果

输入： $n=10, m=3$ ，密码分别为 1, 2, 3, 4, 5, 6, 7, 8, 9, 10

输出：3 6 9 2 7 1 8 5 10 4

输入： $n=15, m=7$ ，密码分别为 2, 5, 3, 7, 8, 1, 9, 10, 4, 6, 15, 14, 12, 11, 13

输出：7 1 10 4 15 8 2 12 11 6 14 5 13 3 9

输入： $n=20, m=10$ ，密码分别为 11, 19, 3, 5, 12, 8, 4, 7, 13, 18, 1, 2, 14, 15, 20, 17, 6, 9, 16, 10

输出：10 1 15 6 16 12 17 19 13 8 3 20 18 9 11 14 7 5 2 4

输入： $n=50, m=30$ ，密码为 1 到 50 的正整数

输出：30 11 41 22 6 38 18 2 35 16 49 31 14 47 29 12 45 28 10 43 26 8 42 25 7 40 23 4 36 20 3 37
21 5 34 19 1 50 39 27 17 48 33 24 13 44 32 15 46 30 9 50 40 25 12 1

7. 思考题

若 m 的值依据每个出列人手中所持的数而改变，则每次有人出列时，将该人手中的数值作为新的 m 值。因此，在循环链表中，需要添加一个密码字段来存储每个人的密码。当一个人出列时，需要将他密码作为新的 m 值。在代码实现中，可以在每个节点中添加一

个密码字段来存储密码。当有人出列时,可以通过指向当前节点的指针来获取该节点的密码,并将该密码赋值给 m 。

修改后的算法大致流程如下:

- 1.初始化单向循环链表,每个节点存储一个人的编号和密码。
- 2.读取初始报数上限值 m 和每个人的密码。
- 3.从第一个人开始报数,每报一次数, m 减 1,当 m 等于 0 时,当前节点出列,并将该节点的密码赋值给 m 。
- 4.重复步骤 3,直至所有人都出列。

四. 附录：源程序文件清单

1. 大数据 1234_LukiRyan_1234_数据结构实验报告一.docx

本文即是。

2. 单链表操作的实现.c

为实验一的源代码。

3. 城市链表.c

为实验二的源代码。

4. 约瑟夫环.c

为实验三的源代码。