北京邮电大学软件学院 2020-2021 学年第 1 学期实验报告

课程名称:	算法与数据	居结构	
实验名称:	线性症	支	
实验完成人:			
姓名: <u>赵冉</u>	_学号:	2020211731	_成绩:
指导教师:	贾红娓	_	

期: 2020 年 10月 10日

日

一、 实验目的

- (1) 本次实验的主要目的在于熟悉线性表的基本运算在两种存储结构上的实现,其中以熟悉各种链表的操作为侧重点。
- (2) 熟悉线性表在软件工程项目中数据存储和处理中的运用。

二、 实验内容

1) 城市链表

[问题描述]

将若干城市的信息,存入一个带头结点的单链表。结点中的城市信息包括:城市名,城市的位置坐标。要求能够利用城市名和位置坐标进行有关查找、插入、删除、更新等操作。

[基本要求]

- (1) 给定一个城市名,返回其位置坐标;
- (2) 给定一个位置坐标 P 和一个距离 D, 返回所有与 P 的距离小于等于 D 的城市。

[测试数据]

由学生依据软件工程的测试技术自己确定。注意测试边界数据。

2) 约瑟夫环

[问题描述]

约瑟夫(Joeph)问题的一种描述是:编号为1,2,...,n的n个人按顺时针方向围坐一圈,每人持有一个密码(正整数)。一开始任选一个正整数作为报数上限值m,从第一个人开始接顺时针方向自1开始顺序报数,报到m时停止报数。报m的人出列,将他的密码作为新的m值,从他在顺时针方向上的下一个人开始重新从1报数,如此下去,直至所有人全部出列为止。试设计一个程序求出出列顺序。

[基本要求]

利用单向循环链表存储结构模拟此过程,按照出列的顺序印出各人的编号。

[测试数据]

m 的初值为 20; 密码: 3, 1, 7, 2, 4, 8, 4 (正确的结果应为 6, 1, 4, 7, 2, 3, 5)。 [实现提示]

程序运行后首先要求用户指定初始报数上限值,然后读取各人的密码。设 n≤30。

选做内容

1) 向上述程序中添加在顺序结构上实现的部分。

三、 实验环境

Dev 或者 vs

四、实验过程和实验结果

1) 城市链表

(1)问题分析

这是一个由带头结点的单链表实现的实验设计。首先,它要求实现存储包括城市名和城市的位置坐标在内的城市信息。另外,要求链表能够实现利用**城市名**和**位置坐标**进行有关查找、插入、删除、更新等操作。除此之外,还要求实现额外功能,给定一个位置坐标 P 和一个距离 D,返回所有与 P 的距离小于等于 D 的城市。另外一个要求给定城市名称返回坐标的要求,在基础操作查找功能中基本实现。

(2) 设计方案

设计时将功能分为基础操作和特殊操作,我们在实现链表的基础功能上组合实现问题分析中提出的功能要求。

- 1) 首先要实现一个创建链表的函数,用以创造城市链表并输入保存所有的城市名称及其位置信息:
- 2) 然后,我们要实现一个打印链表的函数用以显示链表内的所有信息,以此来进行后续的调试和信息展示;
- 3)对于查找城市信息的功能,我们设计两个函数,一个是根据的城市名称查询其城市坐标,另一个是根据输入的城市坐标(所有的坐标数据都以小数点后两位的浮点数为标准)查询城市名称;
- 4)接着,对于删除操作我们同样分为两个功能,一个是根据城市的名称删除城市信息, 另一个是根据输入的城市坐标删除城市信息;
- 5)最后,对于根据中心城市和给定距离输出城市的功能,我们首先利用两点间距离公式求取出每两个城市的距离,然后与给定距离进行比较,在范围内的城市即保留,不在范围内的城市则删除,最后返回链表头结点,以此实现功能。

(3) 算法

//定义一个链表结构

//创建一个链表

```
LinkList CreatList (int e) {
    LinkList 1, tempt, p;
    l = (LinkList)malloc(sizeof(Node));
    l->next = NULL;
    tempt = l;
```

```
int i = 0;
       for(i = 0; i < e; i++) { //向链表中存储数据
          p = (LinkList)malloc(sizeof(Node));
          scanf("%s %f %f", &p->name, &p->x index, &p->y index);
          p->next = tempt->next;
          tempt->next = p;
                      //尾插法
          tempt = p;
       }
       return 1;
   }
//打印出链表的信息
   Status ShowList(LinkList l) {
       LinkList p;
       p = 1;
       p = p->next;
          printf("\t\t\t\ 城市信息为: ");
          printf("\n");
          printf("\t\t\t\t\----\n");
          printf("\t\t\t\ 序\t 城市名称\t 城市坐标\n", p->name, p->x_index, p->y_index);
       int order = 1;
       while(p) {
          p->y_index);
          p = p->next;
          order++;
       }
          printf("\t\t\t\t\----\n");
          return OK;
   }
//利用城市名称查找一个链表元素
   LinkList SearchList_name(LinkList l){
       LinkList q;
```

```
q = (LinkList)malloc(sizeof(Node));
        q->next = NULL;
        scanf("%s", &q->name);
        LinkList p;
        p = 1;
        while(p) {
            p = p->next;
            if(!strcmp(p->name, q->name)){
                 return p;
             }
             if(p->next == NULL){
             printf("没有查找到该城市信息");
             return NULL;
            break;
        }
    }
//利用城市坐标查找一个链表元素
    LinkList SearchList_index(LinkList l){
        LinkList q;
        q = (LinkList)malloc(sizeof(Node));
        q->next = NULL;
        scanf("%f %f", &q->x_index, &q->y_index);
        LinkList p;
        p = 1;
        while(p) {
            p = p->next;
            if(p->x index == q->x index && p->y index == q->y index){
                 return p;
             }
             if(p->next == NULL)
             printf("没有查找到该城市信息");
             return NULL;
            break;
             }
                                          5
        }
```

}

```
//将城市插入链表头部
    LinkList InsertList head (LinkList l, LinkList temp) {
       Node *p, *q;
       p = 1;
        q = temp;
        q = q->next;
        q->next = p->next;
        p->next = q;
        return 1;
    }
//将城市插入任一城市之后
    LinkList InsertList_city (LinkList l, LinkList temp) {
       printf("请输入希望插入其后的城市:");
        Node *index, *q;
        index = SearchList name(l);
       q = temp;
        q = q->next;
        q->next = index->next;
        index->next = q;
        return 1;
    }
//插入一个链表元素
    LinkList InsertList (LinkList 1) {
       // 插入城市信息
       printf("\n");
        printf("请选择您希望将该城市插入的位置: \n");
        printf("A.将该城市插入至城市链表的头部\n");
        printf("B.将该城市插入任一城市之后\n");
        char c;
```

```
scanf("%s", &c);
       LinkList tempt;
        printf("请输入新插入的城市名: \n");
        tempt = CreatList(1);
        switch(c) {
            case 'A':case 'a': l = InsertList head (l, tempt); break;
           case 'B':case 'b': l = InsertList_city (l, tempt) ; break ;
        }
       return 1;
    }
//利用城市名称删除一个链表元素
    LinkList DeleteList_name(LinkList l){
        printf("请输入你想要删除的城市名称: \n");
       //已找到指向想要删除的城市的指针
        Node *index , *p;
        index = SearchList_name(l) ;
       //进行删除操作
       p = 1;
        while(p->next != index) {
           p = p->next;
        p->next = p->next->next;
        return 1;
    }
//利用城市位置删除一个链表元素
    LinkList DeleteList index(LinkList l){
        printf("请输入你想要删除的城市的地理位置: \n");
   //已找到指向想要删除的城市的指针
        Node *index , *p;
```

```
index = SearchList index(l);
       //进行删除操作
       p = 1;
       while(p->next != index) {
           p = p->next;
       }
       p->next = p->next->next;
       return 1;
    }
//更新链表中的一个元素
   LinkList RenewList(LinkList 1) {
       printf("请输入你想要更新的城市名称: \n");
       //已找到指向想要更新的城市的指针
       Node *index , *p;
       index = SearchList_name(l);
       //进行更新操作
       printf("请输入新的城市名称及地理位置:\n");
       scanf("%s %f %f",&index->name, &index->x_index, &index->y_index);
       return 1;
    }
   //进入城市距离计算比较功能
   LinkList DistanceCity(LinkList l, float e){
       printf("请输入一个城市的名称作为定位标志:\n");
       Node *index , *p, *q;
       index = SearchList_name(l);
   //从列表中删除被选中的定位城市的信息
       p = 1;
       while(p->next != index) {
           p = p->next;
       p->next = p->next->next;
```

ShowList(1); //计算列表中其他城市与定位城市的距离 int i = 0; float dx = 0, dy = 0; float distance = 0; LinkList dis; //制作一个链表存储距离小于给定值的城市的所有信息; Node *d; //一直指向 dis 链表末尾的指针; dis = (LinkList)malloc(sizeof(Node)); dis->next = NULL; d = dis; q=1;//利用两个坐标点间的距离公式,开始计算 while(q){ q = q->next; dx = q->x index - index-x index;dy = q->y index - index->y index; distance = sqrt(dx * dx + dy * dy);printf("%-8s 距%-8s 的距离为%.2f\n",q->name, index->name, distance); if(distance <= e) { LinkList temp; temp = (LinkList)malloc(sizeof(Node)); strcpy(temp->name, q->name); temp->x index = q->x index; temp->y index = q->y index; temp->next = d->next; d->next = temp; d = temp;

if(q->next == NULL)

ShowList(dis);

}

}

return dis;

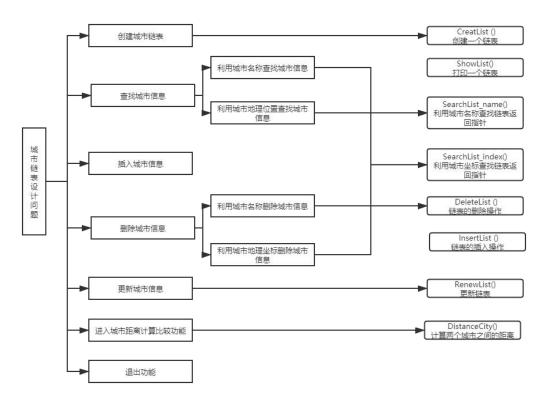
```
Status Entrance(char c, LinkList 1){
    //利用城市名称查找城市信息
     if(c == 'A' \parallel c == 'a')  {
    Node *index;
    printf("请输入希望查询的城市名称:");
    index = SearchList name(1);
    printf("\ (\ \%.2f^\circ\ \ N,\ \%.2f^\circ\ \ E\ )",\quad index->x\_index,\ index->y\_index)\ ;
}
    //利用城市坐标查找城市信息
    if(c == 'B' \parallel c == 'b')  {
    Node *index;
    printf("请输入希望查询的城市坐标:");
    index = SearchList index(l);
    printf("%s", index->name);
}
    //插入城市信息
    if(c == 'C' || c == 'c') {
    l = InsertList(l);
    ShowList(l);
}
    //利用城市名称删除城市信息
    if(c == 'D' || c == 'd') {
    1 = DeleteList name(1);
    ShowList(l);
}
    //利用城市坐标查找城市信息
    if(c == 'E' || c == 'e') {
    l = DeleteList index(l);
    ShowList(l);
}
    //更新城市信息
    if(c == 'F' || c == 'f') {
    l = RenewList(l);
```

```
ShowList(l);
   }
   //进入城市距离计算比较功能
   if(c == 'G' || c == 'g') {
   float n;
   printf("请输入您想查询的距离范围的数值:\n");
   scanf("%f", &n);
   LinkList l_;
   1_ = DistanceCity(1,n);
   ShowList(l_);
   return OK;
}
int main() {
   printf("建立城市链表: ");
   int e = 0;
   printf("请输入您想输入的城市数目:");
   scanf("%d", &e);
   //创建一个城市链表
   LinkList 1;
   printf("请输入城市名及其坐标: \n");
   l = CreatList(e);
   ShowList(l);
   //入口
   printf("进入城市链表操作\n");
   printf("在创建城市链表后,可以进行如下操作\n");
   printf("A.利用城市名称查找城市信息\n");
   printf("B.利用城市地理位置查找城市信息\n");
   printf("C.插入城市信息\n");
   printf("D.利用城市名称删除城市信息\n");
```

```
printf("E.利用城市地理坐标删除城市信息\n");
printf("F.更新城市信息\n");
printf("G.进入城市距离计算比较功能\n");
printf("H.退出操作\n");
while(1){
char c;
printf("\n 请输入操作数:\n");
scanf("%s", &c);
if(c == 'F' || c == 'f'){
printf("退出程序。\n");
break; // 退出程序 }
Entrance(c, l);
}
return 0;
}
```

(4) 设计图

算法设计图如下:



1.1 程序设计图解

(5) 程序

下述为程序截图:

```
1 #include <stdio.h>
 2 #include <stdlib.h>
 3 #include <string.h>
   #include <math.h>
 5
   #define MAXINT 10000
 6
    #define OK 1
 7
    #define ERROR 0
 8
 9
10 typedef int Status ;
11 □ typedef struct Node{
12
                              //存储城市名;
13
        char name[MAXINT] ;
        float x_index, y_index; //存储城市坐标;
14
        struct Node *next;
15
16
17
   Node, *LinkList;
18
19
```

图 2.1 定义一个结构

```
20 //创建一个链表
21 ☐ LinkList CreatList (int e) {
22
23
        LinkList 1, tempt, p;
        1 = (LinkList)malloc(sizeof(Node));
24
25
        1->next = NULL;
26
27
        tempt = 1;
28
        int i = 0;
29
        for(i = 0; i < e; i ++) { //向链表中存储数据
30日
31
           p = (LinkList)malloc(sizeof(Node));
           scanf("%s %f %f", &p->name, &p->x_index, &p->y_index);
32
33
           p->next = tempt->next;
34
           tempt->next = p ;
35
           tempt = p ;
36
37
        return 1;
38 L }
```

图 2.2 链表创建函数

```
40 //打印出链表的信息
41 □ Status ShowList(LinkList 1) {
      LinkList p ;
42
43
      p = 1;
44
      p = p->next;
         printf("\t\t\t\t\t城市信息为: ");
45
         printf("\n");
printf("\t\t\t\t----\n");
46
47
         printf("\t\t\t\tr\\t城市名称\t城市坐标\n", p->name, p->x_index, p->y_index);
48
49
      int order = 1;
50 E
      while(p) {
51
         printf("\t\t\t\t\t\02d\t %-8s\t( %.2f° N, %.2f° E )\n",order, p->name, p->x_index, p->y_index);
52
         p = p->next;
53
         order++ ;
54
55
         printf("\t\t\t\t----\n" );
56
         return OK ;
57 L }
```

图 2.3 链表打印函数

```
59 //利用城市名称查找一个链表元素
60 ☐ LinkList SearchList name(LinkList 1){
        LinkList q;
62
        q = (LinkList)malloc(sizeof(Node));
63
        q->next = NULL;
        scanf("%s", &q->name);
64
65
        LinkList p ;
        p = 1;
66
67 🖨
        while(p) {
68
            p = p->next;
69 ₽
            if(!strcmp(p->name, q->name)){
               return p ;
70
71
72 白
            if(p->next == NULL){
            printf("没有查找到该城市信息");
73
            return NULL;
74
75
           break ;
76
            }
77
78 L }
```

图 2.4 城市名称查找函数

```
80 //利用城市坐标查找一个链表元素
81 ☐ LinkList SearchList_index(LinkList 1){
82
        LinkList q ;
83
        q = (LinkList)malloc(sizeof(Node));
84
        q->next = NULL ;
85
        scanf("%f %f", &q->x_index, &q->y_index);
86
        LinkList p ;
87
        p = 1;
88 🖨
        while(p) {
89
            p = p->next;
            if(p->x_index == q->x_index && p->y_index == q->y_index){
90 🖨
91
                return p ;
92
            if(p->next == NULL){
printf("没有查找到该城市信息");
93 🖨
94
95
            return NULL;
            break ;
96
97
98
99 1
```

图 2.5 城市坐标查找函数

```
101 //将城市插入链表头部
102 □ LinkList InsertList_head (LinkList 1, LinkList temp) {
         Node *p, *q;
103
104
         p = 1;
105
         q = temp;
106
         q = q \rightarrow next;
107
         q->next = p->next;
108
         p\rightarrow next = q;
109
          return 1;
110 |
```

图 2.6 城市插入函数 (插入至头部)

```
112 //将城市插入任一城市之后
113 ☐ LinkList InsertList_city (LinkList l, LinkList temp) {
         printf("请输入希望插入其后的城市:");
115
        Node *index, *q;
116
        index = SearchList_name(1);
117
        q = temp;
118
        q = q \rightarrow next;
119
        q->next = index->next;
120
        index->next = q;
121
        return 1 ;
122 }
```

图 2.7 城市插入函数 (插入至特定函数)

```
124 //插入一个链表元素
125 ☐ LinkList InsertList (LinkList 1) {
126
         // 插入城市信息
127
         printf("\n")
         printf("请选择您希望将该城市插入的位置: \n");
128
         printf("A.将该城市插入至城市链表的头部\n");
printf("B.将该城市插入任一城市之后\n");
129
130
131
         char c ;
         scanf("%s", &c);
132
133
         LinkList tempt ;
printf("请输入新插入的城市名: \n") ;
134
135
         tempt = CreatList(1);
136
137
         switch(c) {
  case 'A':case 'a': 1 = InsertList_head (1, tempt); break;
138 🖨
139
             case 'B':case 'b': 1 = InsertList_city (1, tempt); break;
140
141
142
         return 1 :
143 }
```

图 2.8 插入城市函数入口

```
145 //利用城市名称删除一个链表元素
146 ☐ LinkList DeleteList_name(LinkList 1){
        printf("请输入你想要删除的城市名称: \n");
147
148
149
        //已找到指向想要删除的城市的指针
150
        Node *index , *p;
        index = SearchList_name(1);
151
152
        //进行删除操作
153
        p = 1;
        while(p->next != index) {
154
155
           p = p->next;
156
157
        p->next = p->next->next;
158
        return 1 ;
159 L }
```

图 2.9 城市名称删除函数

```
161 //利用城市位置删除一个链表元素
162 ☐ LinkList DeleteList_index(LinkList 1){
163
        printf("请输入你想要删除的城市的地理位置: \n");
164
        //已找到指向想要删除的城市的指针
165
        Node *index , *p;
index = SearchList_index(1);
166
167
        //进行删除操作
168
169
        p = 1;
170
        while(p->next != index) {
171
           p = p->next;
172
173
        p->next = p->next->next;
174
        return 1 ;
175
```

图 2.10 城市位置删除函数

```
177 //更新链表中的一个元素
178 ☐ LinkList RenewList(LinkList 1) {
        printf("请输入你想要更新的城市名称: \n");
180
181
        //已找到指向想要更新的城市的指针
        Node *index , *p;
index = SearchList_name(1);
182
183
184
        printf("请输入新的城市名称及地理位置:\n");
185
186
        scanf("%s %f %f",&index->name, &index->x_index, &index->y_index);
        return 1;
187
188
```

图 2.11 城市更新函数

```
//进入城市距离计算比较功能
190
194
      index = SearchList_name(1);
195
196
      //从列表中删除被选中的定位城市的信息
197
      p = 1;
198
      while(p->next != index) {
199
        p = p->next;
200
201
      p->next = p->next->next;
202
      ShowList(1);
```

图 2.12 城市距离计算函数

图 2.13 计算两点间距离

16

```
q = 1; //利用两个坐标点间的距离公式, 开始计算
while(q){
    q = q->next;
    dx = q->x_index - index->x_index;
    dy = q->y_index - index->y_index;
    distance = sqrt(dx * dx + dy * dy);
    printf("%-8s指述-8s的距离为统.2f\n",q->name, index->name, distance);
    if(distance, distance);
215 <del>|</del> 216
217
218
219
220
221 =
                                if(distance <= e) {
   LinkList temp;</pre>
223
224
                                         temp = (LinkList)malloc(sizeof(Node));
                                        temp = (LINKISt)mailor(SIZeOr
strcpy(temp->name , q->name) ;
temp->x_index = q->x_index ;
temp->y_index = q->y_index ;
temp->next = d->next ;
225
226
 227
228
                                        d->next = temp;
d = temp;
 229
230
 231
                                if(q->next == NULL)
232
                                ShowList(dis);
 233
                      return dis ;
```

图 2.14 进行链表的删除和保留

```
237 □ Status Entrance(char c, LinkList 1 ){
238
         //利用城市名称查找城市信息
| if(c == 'A' || c == 'a') {
Node *index ;
239
240
241
          printf("请输入希望查询的城市名称:");
242
         index = SearchList_name(1);
printf(" ( %.2f° N, %.2f° E )", index->x_index, index->y_index);
243
244
245
246
          //利用城市坐标查找城市信息
         if(c == 'B' || c == 'b') {
247 白
         Node *index ;
248
249
         printf("请输入希望查询的城市坐标:");
250
          index = SearchList_index(1);
         printf("%s", index->name);
251
252
         //插入城市信息
if(c == 'C' || c =='c') {
253
254
255
          l = InsertList(1);
256
         ShowList(1);
```

图 2.15 入口功能函数

```
258
        //利用城市名称删除城市信息
259 🗦
        if(c == 'D' || c =='d') {
260
        l = DeleteList_name(1);
261
        ShowList(1);
262
    - }
263
264
        //利用城市坐标查找城市信息
        if(c == 'E' || c == 'e') {
265 🖨
        l = DeleteList_index(1);
266
267
        ShowList(1);
268
269
270
        //更新城市信息
        if(c == 'F' || c == 'f') {
271 垣
272
        1 = RenewList(1);
273
        ShowList(1);
274
        }
275
```

图 2.16 入口功能函数 (2)

```
//进入城市距离计算比较功能
if(c == 'G' || c == 'g') {
276
277 🖨
         float n;
printf("请输入您想查询的距离范围的数值: \n");
278
279
         scanf("%f", &n);
280
281
          LinkList 1_ ;
282
          l_ = DistanceCity(l,n);
283
         ShowList(l_);
284
285
286
          return OK ;
287 }
```

图 2.17 入口功能函数 (3)

```
290 □ int main() {
291
292
293
        printf("建立城市链表: ");
294
        int e = 0;
295
        printf("请输入您想输入的城市数目:");
        scanf("%d", &e);
296
297
        //创建一个城市链表
298
        LinkList 1;
printf("请输入城市名及其坐标: \n");
299
300
301
        1 = CreatList(e);
302
        ShowList(1);
303
```

图 2.18 主函数

```
1111
304
305
         printf("进入城市链表操作\n");
         printf("在创建城市链表后,可以进行如下操作\n");
printf("A.利用城市名称查找城市信息\n");
306
307
         printf("B.利用城市地理位置查找城市信息\n");
308
         printf("C.插入城市信息\n");
printf("D.利用城市名称删除城市信息\n");
309
310
311
         printf("E.利用城市地理坐标删除城市信息\n");
         printf("F.更新城市信息\n");
printf("G.进入城市距离计算比较功能\n");
312
313
         printf("H.退出操作\n");
314
315 🖨
         while(1){
         char c;
printf("\n请输入操作数:\n");
316
317
         scanf("%s", &c);
if(c == 'F' || c == 'f'){
318
319 白
         printf("退出程序。\n");
320
321
         break ; // 退出程序
322 - }
323
         Entrance(c, 1);a
324
```

图 2.19 主函数 (2)

(6) 调试过程截图

调试过程中遇到的问题截图:

```
नि 🔼 🗖 । विर
                                                                                                                   城市特表第一版。
                                                                                                                                                                                        printf("B. 利用城市地理位置查找城市信息\n");
printf("C. 插入城市信息\n");
printf("D. 利用城市名称删除城市信息\n");
printf("E. 利用城市地理坐标删除城市信息\n");
printf("F. 更新城市信息\n");
printf("G. 进入城市距离计算比较功能\n");
printf("H. 退出操作\n");
while(1){
                                                                                                                   308
                                                                                                                     309
                                                                                                                     310
                                                                                                                   312
                                                                                                                     313
                                                                                                                   314
                                                                                                                     315
                                                                                                                                                                                         white(),
char c;
printf("\nii\hat{\partial}\hat{\partial}\hat{\partial}\hat{\partial}\hat{\partial}\hat{\partial}\hat{\partial}\hat{\partial}\hat{\partial}\hat{\partial}\hat{\partial}\hat{\partial}\hat{\partial}\hat{\partial}\hat{\partial}\hat{\partial}\hat{\partial}\hat{\partial}\hat{\partial}\hat{\partial}\hat{\partial}\hat{\partial}\hat{\partial}\hat{\partial}\hat{\partial}\hat{\partial}\hat{\partial}\hat{\partial}\hat{\partial}\hat{\partial}\hat{\partial}\hat{\partial}\hat{\partial}\hat{\partial}\hat{\partial}\hat{\partial}\hat{\partial}\hat{\partial}\hat{\partial}\hat{\partial}\hat{\partial}\hat{\partial}\hat{\partial}\hat{\partial}\hat{\partial}\hat{\partial}\hat{\partial}\hat{\partial}\hat{\partial}\hat{\partial}\hat{\partial}\hat{\partial}\hat{\partial}\hat{\partial}\hat{\partial}\hat{\partial}\hat{\partial}\hat{\partial}\hat{\partial}\hat{\partial}\hat{\partial}\hat{\partial}\hat{\partial}\hat{\partial}\hat{\partial}\hat{\partial}\hat{\partial}\hat{\partial}\hat{\partial}\hat{\partial}\hat{\partial}\hat{\partial}\hat{\partial}\hat{\partial}\hat{\partial}\hat{\partial}\hat{\partial}\hat{\partial}\hat{\partial}\hat{\partial}\hat{\partial}\hat{\partial}\hat{\partial}\hat{\partial}\hat{\partial}\hat{\partial}\hat{\partial}\hat{\partial}\hat{\partial}\hat{\partial}\hat{\partial}\hat{\partial}\hat{\partial}\hat{\partial}\hat{\partial}\hat{\partial}\hat{\partial}\hat{\partial}\hat{\partial}\hat{\partial}\hat{\partial}\hat{\partial}\hat{\partial}\hat{\partial}\hat{\partial}\hat{\partial}\hat{\partial}\hat{\partial}\hat{\partial}\hat{\partial}\hat{\partial}\hat{\partial}\hat{\partial}\hat{\partial}\hat{\partial}\hat{\partial}\hat{\partial}\hat{\partial}\hat{\partial}\hat{\partial}\hat{\partial}\hat{\partial}\hat{\partial}\hat{\partial}\hat{\partial}\hat{\partial}\hat{\partial}\hat{\partial}\hat{\partial}\hat{\partial}\hat{\partial}\hat{\partial}\hat{\partial}\hat{\partial}\hat{\partial}\hat{\partial}\hat{\partial}\hat{\partial}\hat{\partial}\hat{\partial}\hat{\partial}\hat{\partial}\hat{\partial}\hat{\partial}
                                                                                                                   316
                                                                                                                     317
                                                                                                                   318
                                                                                                                     320
                                                                                                                     321
                                                                                                                     322
                                                                                                                                                                                 Entrance(c, 1) ;a
                                                                                                                     303
                                                                                                                     324
                                                                                                                     325
                                                                                                                     326
                                                                                                                 327
                                                                                                                                                                                           return 0;
Compiler (4) h Resources Compile Log Debug k Find Results Close
                                                                                                                                                                                                                                                                                          Message
In function 'main':
[Error] 'a' undeclared (first use in this function)
[Note] each undeclared identifier is reported only once for each function it appears in [Error] expected (*) before ')' token
```

图 3.1 代码调试截图 (1)

```
(globals)
                                      printf("B.利用城市地理位置查找城市信息\n");
printf("C.插入城市信息\n");
printf("D.利用城市名称删除城市信息\n");
printf("E.利用城市中型标删除城市信息\n");
printf("F.更新城市信息\n");
printf("G.进入城市距离计算比较功能\n");
printf("H.退出操作\n");
while(1){
char c:
                       城市链表第二版,c
ct Classes Debug
                       308
                       309
                       310
                       311
                       312
313
                       314
315 □
                                       white[],
char c;
printf("\n请输入操作数:\n");
scanf("%s", &c);
if(c == 'f' || c == 'f'){
printf("退出程序。\n");
break; // 退出程序
                       316
317
                       318
319 🛱
                       320
321
                       322
323
                                        Entrance(c, 1);
                       324
325
                       326
327
                                        return 0;
                       328 - }
compiler (2) 🍓 Resources 🛍 Compile Log 🤣 Debug 🔼 Find Results 🥞 Close
 Col File
                                                               Message
        DiLlenovoSoftstore\Install\DevC\MinGW64v86_64-w64-...
cannot open output file C\User\\86188\Desktop\城市链表第二版.exe Permission denied
C\User\\86188\Desktop\collect2.exe
[Error] Id returned 1 exit status
```

图 3.2 代码调试截图 (2)

(7) 运行结果截图

-a. 输入链表功能与打印功能截图

■ C:\Users\86188\Desktop\城市链表第二版.exe

```
建立城市链表: 请输入您想输入的城市数目:6
请输入城市名及其坐标:
沈阳 41.48 123.25
合肥 31.25 117.17
北京 39.55 116.24
晋江 24.49 118.35
深圳 22.33 114.07
桂林 23.22 110.04
```

图 4.1 城市输入

Ė		+++ →= + + +=		
厅	城市名称	城市坐标	100 050	
01	77CPH	(41.48° N,	123. 25°	
02	合肥	(31.25° N,	117. 17°	E)
03	北京	(39.55° N,	116. 24°	E)
04	晋江	(24.49° N,	118.35°	E)
05	深圳	(22.33° N,	114. 07°	E)
06	桂林		110.04°	E)

图 4.2 城市打印

-b. 利用城市名称查找信息截图

```
进入城市链表操作
在创建城市链表后,可以进行如下操作
A. 利用城市名称查找城市信息
B. 利用城市地理位置查找城市信息
C. 插入城市信息
D. 利用城市名称删除城市信息
E. 利用城市地理坐标删除城市信息
F. 更新城市信息
G. 进入城市距离计算比较功能
H. 退出操作
请输入操作数:
a
请输入希望查询的城市名称:北京
(39.55°N, 116.24°E)
```

图 4.3

```
b
请输入希望查询的城市坐标: 39.55 116.24
北京
```

图 4.4

-c. 利用城市坐标查找信息截图

```
清输入操作数:
清选择您希望将该城市插入的位置:
A. 将该城市插入至城市链表的头部
B. 将该城市插入任一城市之后
a.
清输入新插入的城市名:
各阳 34.41 112.27
```

图 4.5

图 4.6

-d. 插入新城市功能截图

```
请输入操作数:
c
请选择您希望将该城市插入的位置:
A. 将该城市插入至城市链表的头部
B. 将该城市插入任一城市之后
b
请输入新插入的城市名:
呼和浩特 40.48 111.41
请输入希望插入其后的城市:晋江
```

图 4.7

```
城市信息为:

| F | 城市名称 | 城市坐标 |
| O1 | 洛阳 | (34.41°N, 112.27°E) |
| O2 | 沈阳 | (41.48°N, 123.25°E) |
| O3 | 合肥 | (31.25°N, 117.17°E) |
| O4 | 北京 | (39.55°N, 116.24°E) |
| O5 | 晋江 | (24.49°N, 118.35°E) |
| O6 | 呼和浩特 | (40.48°N, 111.41°E) |
| O7 | 深圳 | (22.33°N, 114.07°E) |
| O8 | 桂林 | (23.22°N, 110.04°E) |
```

图 4.8

-e. 利用城市名称删除截图

```
请输入操作数:
d
请输入你想要删除的城市名称:
合肥
```

图 4.9

图 4.10

-f. 利用城市坐标删除截图

请输入操作数: e 请输入你想要删除的城市的地理位置: 22.33 114.07

图 4.11

图 4.12

-h. 更新城市信息截图

```
请输入操作数:
f
请输入你想要更新的城市名称:
晋江
请输入新的城市名称及地理位置:
晋江 10 110
```

图 4.13

图 4.14

-i. 城市距离计算比较截图

```
g
请输入您想查询的距离范围的数值:
10.5
请输入一个城市的名称作为定位标志:
北京
```

图 4.15

```
洛阳 距北京 的距离为6.49
沈阳 距北京 的距离为7.27
晋江 距北京 的距离为30.20
呼和浩特距北京 的距离为4.92
桂林 距北京 的距离为17.47
```

图 4.16

图 4.17

```
(8) 顺序结构不同处的算法设计:
    typedef struct Node{
        char name[MAXINT];
                                  //存储城市名;
        float x_index[MAXINT], y_index[MAXINT];//用数组存储城市坐标;
        int order;
        struct Node *next;
    }Node, *LinkList;
//创建一个链表
    LinkList CreatList (int e) {
        LinkList l, tempt, p, h;
        l = (LinkList)malloc(sizeof(Node));
        1->next = NULL;
        tempt = 1;
        int i = 0;
        for(i = 0; i < e; i++) { //向链表中存储数据
            p = (LinkList)malloc(sizeof(Node));
            scanf("%s", &p->name);
            p->next = tempt->next;
            tempt->next = p;
            tempt = p;
```

```
p->order = i;
       }
       for(i = 0; i < e; i ++) { //向链表中存储数据
           \operatorname{scanf}(\text{"}\%f \%f", \&l->x \operatorname{index}[i], \&l->y \operatorname{index}[i]);
       }
       return 1;
//打印出链表的信息
   Status ShowList(LinkList 1) {
       LinkList p;
       p = 1;
       p = p - next;
           printf("\t\t\t\ 城市信息为: ");
           printf("\n");
           printf("\t\t\t\t\t----\n");
           printf("\t\t\t\t 序\t 城市名称\t 城市坐标\n");
       int order = 1, i = 0;
       while(p){
           1->x_index[i], 1->y_index[i]);
           p = p->next;
           order++;
           i++;
       }
           printf("\t\t\t\t\t----\n");
           return OK;
       //进入城市距离计算比较功能
   LinkList DistanceCity(LinkList I, float e, int Number){
       printf("请输入一个城市的名称作为定位标志:\n");
       Node *index , *p, *q;
       index = SearchList(1);
 //从列表中删除被选中的定位城市的信息
       p = 1;
       while(p->next != index) {
           p = p->next;
       p->next = p->next->next;
       ShowList(l);
 //计算列表中其他城市与定位城市的距离
       int i = 0;
```

```
float dx = 0, dy = 0;
        float distance = 0;
        LinkList dis; //制作一个链表存储距离小于给定值的城市的所有信息;
        Node *d; //一直指向 dis 链表末尾的指针;
        dis = (LinkList)malloc(sizeof(Node));
        dis->next = NULL;
        d = dis;
        q=1;//利用两个坐标点间的距离公式,开始计算
        while(q){
             q = q->next;
             dx = q->x \text{ index - index-}x \text{ index};
             dy = q->y index - index->y index;
             distance = sqrt(dx * dx + dy * dy);
             printf("%-8s 距%-8s 的距离为%.2f\n",q->name, index->name, distance);
             if(distance <= e) {
                 LinkList temp;
                 temp = (LinkList)malloc(sizeof(Node));
                 strcpy(temp->name, q->name);
                 temp->x index[MAXINT] = q->x index[MAXINT];
                 temp->y index[MAXINT] = q->y index[MAXINT];
                 temp->next = d->next;
                 d->next = temp;
                 d = temp;
             }
             if(q->next == NULL)
             ShowList(dis);
        }
        return dis;
//获取指定的元素
    int GetListElem(LinkLlist L, int id, LinkList tempt)
        int i;
        for(i=0; i<L->len; i++) {
             if(L\rightarrow? index[i].id == id)
                 break;
        if(i \ge L->len)
             return -1;
        tempt = L - > ? index[i];
        return 0;
    }
```

//删除指定的链表的元素

```
int ListDelete(LinkList index)
{
     int i,k;
     for(i=0; i<L->len; i++) {
          if(L\rightarrow? index[i].id == id)
               break;
     }
     if(i \ge L->len)
          return -1;
     //删除的不是最后位置
     if (i < (L->len-1))
          for(k=i; k<L->len; k++)
               L->? index[k] = L->info[k+1];//将删除位置后继元素前移
     L->len--;
     return 0;
}
//追加数据
int ListAppend(LinkList L, int temp)
L \rightarrow ?-index[L \rightarrow len++] = temp;
return 0;
}
```

2) 约瑟夫环

(1)问题分析

这是一个由循环链表实现的实验设计。约瑟夫问题的一种描述是:编号为1,2,...,n的nn个人按顺时针方向围坐一圈,每人持有一个密码。一开始任选一个正整数作为报数上限值m,从第一个人开始按顺时针方向自1开始顺序报数,报到m时停止报数。报m的人出列,将他的密码作为新的m值,从他在顺时针方向上的下一个人开始重新从1报数,如此下去,直至所有人全部出列为止,需要设计一个程序求出出列顺序。

(2) 设计方案

在求解问题时,首先要完成每个成员序列和所持密码数这两个数据的存储。然后寻找到每次报数的同学,记录它们的序列并及时更替 m 值。

- 1) 首先,实现循环链表输入 n 的操作;
- 2) 实现创建单向循环链表存放的操作;
- 3) 实现打印单向循环链表的操作;
- 4) 找到报 m 的成员,确定其指针,这里采用的思路是利用循环链表,一直遍历,直到

```
得到 m 时停止;
    5) 输出 m 的序号、替换 m 值;
    6)输出得到的 m 序号值;
 (3) 算法
    typedef int Status;
    typedef struct Node{
        int order;
        int code;
        struct Node *next;
    }Node, *LinkList;
//完成数据 n 的输入功能
    int Input(){
        int n;
        scanf("%d", &n);
        if(n \le 30) return n;
            printf("请重新输入: \n");
            Input();
        }
    }
//打印出链表的长度
    int GetLength(LinkList l) {
        int len = 1;
        LinkList p;
        p = 1;
        p = p->next;
        while(p != 1) {
            len ++;
            p = p->next;
```

return len;

}

```
//创建一个循环链表,并键入数据
   LinkList CreatList( int e ) {
       //创建一个链表 LinkList 和一个储存头结点的 node
       LinkList 1;
       Node *1 head;
       l = (LinkList)malloc(sizeof(Node));
       1 \text{ head} = 1;
       //尾插法输入元素
       Node *temp, *p;
       p = 1;
       int i = 0, order = 1;
       for(i = 0; i < e; i +++) {
           //把数据存入临时存储的链表中
           temp = (LinkList)malloc(sizeof(Node)); //每次循环都创造一个新的结点
           scanf("%d", &temp->code);
           temp->order = order;
           //利用尾插法将临时存储的结点插入到1中
           temp->next = p->next;
           p->next = temp;
           p = temp; //尾插
           order ++;
       1 head = 1 head->next;
       p->next = 1_head; //将指向1末尾的指针指回头结点,形成循环链表
       return p;
   }
```

//打印出链表的信息

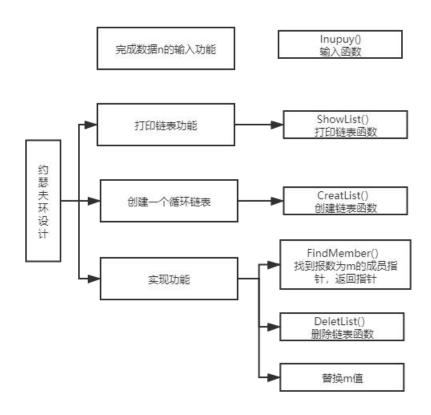
```
Status ShowList(LinkList l) {
    LinkList p;
```

```
p = 1;
        p = 1->next;
        while(p != l) {
             printf("%d-%d ",p->order, p->code);
             p = p->next;
        }
        printf("%d-%d", 1->order, 1->code);
        printf("\n");
        return OK;
    }
//找到报数为 m 的成员对应的指针
    LinkList FindMember(LinkList 1, int m, int n) {
        int counter = 1;
        LinkList p;
        p = 1;
        p = p->next;
        while(counter != m) {
             p = p->next;
             counter ++;
        }
        return p;
    }
//将一个元素插入现有链表的尾部
LinkList InsertList (LinkList l, LinkList temp) {
        Node *p, *q;
        p = 1;
        q = temp;
        q->next = p->next;
        p->next = q;
        return 1;
//删除一个链表元素
```

```
LinkList DeleteList(LinkList l, Node* index, int len){
    Node *p;
    //如果只剩下一个元素,直接将指针1释放
    if(len == 1)
    1 = NULL;
//进行删除操作
    p = 1;
    while(p->next != index) {
        p = p->next;
    }
    p->next = p->next->next;
    return 1;
}
Status Entrance(LinkList l, int m, int Number) {
    LinkList l_code ;
    Node *index, *temp_code;
    index = 1;
    int len = 0;
    while(l) {
    len = GetLength(l);
    //寻找报数为 m 的成员,输出她/他的序号和密码值
    index = FindMember( index , m, Number) ;
    printf("%d ", index->order);
//替换m值
    m = index - > code;
    //将 index 的值备份
    Node *temp_index;
```

```
temp_index = index;
  //将 m 成员从链表中删除
       if(temp\_index == 1) 1 = 1 - next;
       1 = DeleteList(l, temp_index, len) }
       return OK;
   }
   int main() {
  //数据 n 的输入
       printf("请输入小组的成员数据 n: \n");
       int Number = 0;
       Number = Input();
       LinkList 1;
       l = CreatList( Number );
       printf("请输入m的初始值\n");
       int m = 0;
       scanf("%d", &m);
        Entrance(l, m, Number);
       return 0;
   }
(4) 设计图
```

算法设计图如下:



5.1 程序设计图解

(5) 程序

下述为程序截图:

```
1 #include <stdio.h>
 2 #include <stdlib.h>
 3 #define OK 1
4 #define ERROR 0
   typedef int Status ;
6
7 ptypedef struct Node{
8
9
        int order;
        int code ;
10
        struct Node *next;
11
12
13 \Node, *LinkList;
```

图 6.1 定义一个结构

```
15 //完成数据n的输入功能
16 □ int Input(){
17
18
       int n;
        scanf("%d", &n);
19
20
       if(n <= 30) return n ;
21
22 白
        else{
           printf("请重新输入: \n");
23
24
           Input();
25
26 L }
```

图 6.2 数据输入函数

```
28 //打印出链表的长度
29 ☐ int GetLength(LinkList 1) {
        int len = 1;
30
31
        LinkList p ;
32
        p = 1;
33
        p = p->next;
34
35 🖨
        while(p != 1) {
36
            len ++ ;
37
            p = p->next;
38
39
        return len ;
40
41 L }
```

图 6.3 链表打印函数

```
43 //创建一个循环链表,并键入数据
44 ☐ LinkList CreatList( int e ) {
45
       //创建一个链表LinkList和一个储存头结点的node
46
47
       LinkList 1;
       Node *1_head ;
48
       1 = (LinkList)malloc(sizeof(Node));
49
50
       1_head = 1 ;
51
       //尾插法输入元素
52
       Node *temp, *p;
53
54
       p = 1;
55
       int i = 0, order = 1;
```

图 6.4 创建链表函数

图 6.5 创建链表函数 (2)

```
76 //打印出链表的信息
77 □ Status ShowList(LinkList 1) {
         LinkList p ;
79
         p = 1;
p = 1->next;
80
         while(p != 1) {
    printf("%d-%d ",p->order, p->code) ;
81 🖨
82
83
             p = p->next;
84
         printf("%d-%d", 1->order, 1->code);
printf("\n");
85
86
87
         return OK ;
88 1
```

图 6.6

```
90 //找到报数为m的成员对应的指针
91⊟ LinkList FindMember(LinkList l , int m, int n) {
92
93
        int counter = 1;
94
        LinkList p ;
 95
        p = 1;
96
        p = p->next;
97
        while(counter != m ) {
98
            p = p->next;
99
            counter ++ ;
100
101
        return p ;
102 }
```

图 6.7 找到报数成员函数

图 6.8

```
114 //删除一个链表元素
115 ☐ LinkList DeleteList(LinkList l, Node* index, int len){
116
117
         Node *p ;
118
         //如果只剩下一个元素,直接将指针L释放
119
120
         if(len == 1)
         1 = NULL ;
121
122
         //进行删除操作
123
124
         p = 1;
125 🖨
         while(p->next != index) {
126
            p = p \rightarrow next;
127
128
         p->next = p->next->next;
129
         return 1;
130 L }
```

图 6.9 删除函数

```
132 □ Status Entrance(LinkList l, int m, int Number) {
133
           LinkList l_code ;
134
135
           Node *index, *temp_code ;
136
           index = 1;
137
           int len = 0;
138
139 🖨
           while(1) {
140
141
          len = GetLength(1);
142
          // 寻找报数为m的成员,输出她/他的序号和密码值
index = FindMember( index , m, Number) ;
printf("%d ", index->order) ;
143
144
145
146
           //替换m值
147
           m = index->code ;
148
```

图 6.10

```
//将index的值备份
150
151
          Node *temp_index ;
          temp_index = index ;
152
153
           //将m成员从链表中删除
154
          if(temp_index == 1) 1 = 1->next;
1 = DeleteList(1, temp_index, len);
155
156
157
158
159
160 - }
```

图 6.11

```
166 □ int main() {
167
168
          // 数据n的输入
printf("请输入小组的成员数据n: \n");
169
170
171
           int Number = 0;
172
173
           Number = Input();
          LinkList 1;
1 = CreatList( Number );
175
176
177
178
179
           printf("请输入m的初始值\n");
          int m = 0;
scanf("%d", &m);
180
181
            Entrance(1 , m, Number);
182
183
           return 0;
```

图 6.12

(6) 调试过程截图

调试过程中遇到的问题截图:

```
城市链表第二版.c 约瑟夫环.c
             164 L }
             165
             166 p int main() {
             167
                          // 数据n的输入
printf("请输入小组的成员数据n: \n");
             168
             169
             170
                          int Number = 0;
Number = Input();
             171
             172
             173
                          LinkList 1;
1 = CreatList( Number );
             174
             175
             176
177
                           printf("请输入m的初始值\n") ;
int m = 0 ;
scanf("%d", &m) ;
             178
             179
             180
                            Entrance(1 , m, Number)
             181
182
             103
                          return 0 ;
er (2) 👣 Resources 📶 Compile Log 🥒 Debug 🗓 Find Results 💐 Close
File Message
C:\Users\86188\Desktop\线性表实验10.9\约器夫环。
C:\User\86188\Desktop\线性表实验10.9\约器夫环。
[Error] expected \( \) before 'return'
```

图 7.1 代码调试截图 (1)

```
162
                 return OK;
     163
     164 L }
     165
    166 □ int main() {
    167
                 //数据n的输入
     168
                 printf("请输入小组的成员数据n: \n");
     169
    170
                 int Number = 0;
    171
                 Number = Input();
    172
    173
                 LinkList 1;
    174
     175
                 1 = CreatList();
    176
    177
                 printf("请输入m的初始值\n");
     178
                 int m = 0;
Resources 🛍 Compile Log 🤣 Debug 🗓 Find Results 🐉 Close
                                Message
rs\86188\Desktop\线性表实验10.9\约瑟夫环。 In function 'main':
s\86188\Desktop\线性赛实验\0.9\的题关环。 [Error] too few arguments to function 'CreatList' s\86188\Desktop\线性要实验\10.9\的题关环。 [Note] declared here
```

图 7.2 代码调试截图 (2)

(7) 运行结果截图

图 8.1 结果显示

(8) 顺序结构不同处的算法设计:

typedef struct Node{

```
int order;
int code[MAXINT];
struct Node *next;

}Node, *LinkList;

//完成数据 n 的输入功能
int Input(){

int n;
scanf("%d", &n);
```

```
if(n \le 30) return n;
    else{
        printf("请重新输入: \n");
        Input();
    }
}
//计算出出链表的长度
int GetLength(LinkList l) {
    int len = 1;
    LinkList p;
    p = 1;
    p = p->next;
    while(p != 1) {
        len ++;
        p = p->next;
    return len;
}
//打印出链表的信息
Void ShowList(LinkList l) {
    LinkList p;
    p = 1;
    p = 1->next;
    while(p != 1) {
        printf("%d ",p->order,);
        p = p->next;
    }
    For(int i = 0; i < len; i++) {
       Printf("%d", l->code[i]);
  }
}
//创建一个循环链表,并键入数据
LinkList CreatList( int e ) {
    //创建一个链表 LinkList 和一个储存头结点的 node
    LinkList 1;
    Node *1 head;
    l = (LinkList)malloc(sizeof(Node));
    1_{\text{head}} = 1;
    //尾插法输入元素
    Node *temp, *p;
```

```
p = 1;
   int i = 0, order = 1;
    for(i = 0; i < e; i ++) {
        //把数据存入临时存储的链表中
        scanf("%d", &l->code[i]);
        order ++;
    }
    l head = l head->next;
    p->next=1 head; //将指向1末尾的指针指回头结点, 形成循环链表
    return p;
int main() {
               //数据 n 的输入
            printf("请输入小组的成员数据 n: \n");
            int Number = 0;
            Number = Input();
            LinkList 1;
            l = CreatList( Number );
            printf("请输入m的初始值\n");
            int m = 0;
            scanf("%d", &m);
             Entrance(1, m, Number);
            return 0;
void Entrance(LinkList l, int m, int Number) {
    LinkList 1 code;
    Node *index, *temp_code;
    index = 1;
    int len = 0;
    while(l) {
    len = GetLength(l);
   //寻找报数为 m 的成员,输出她/他的序号和密码值
    index = FindMember( index , m, Number) ;
    printf("%d ", index->order);
```

```
//替换 m 值
    m = index->code[index->order];
    //将 m 成员从链表中删除
    if(temp index == 1) 1 = 1 - next;
    l = DeleteList(l, temp index, len);
}
//找到报数为 m 的成员对应的指针
LinkList FindMember(LinkList 1, int m, int n) {
    int counter = 1;
    LinkList p;
    p = 1;
    p = p->next;
    while(counter != m) {
        p = p->next;
        counter ++ ;
    }
    return p;
//删除一个链表元素
LinkList DeleteList(LinkList 1, Node* index, int len){
    Node *p;
//如果只剩下一个元素,直接将指针1释放
    if(len == 1)
    1 = NULL;
    //进行删除操作
    p = 1;
    while(p->next != index) {
        p = p->next;
    p->next = p->next->next;
    return 1;
}
```

五、 实验心得

此次实验重点考察对于线性表的应用。首先是利用单链表实现线性结构。对于链表来说,指针的使用非常重要。在开始做实验的时候,指针的使用很容易让人生出手足无措的感觉,但是在编写实验代码的过程中,我渐渐开始理解指针,并开始明白它的强大和灵活性。例如城市链表中大量信息的存储,一个链表就能将它们的全部信息完全的存储梳理。

通过这次实验,我学习并掌握线性表的顺序存储结构、链式存储结构的设计与操作。对单链表建立、插入、删除等操作有了最基本的理解。在这个过程中,我对于链表的头结点和第一个数据节点、包括循环链表,等等概念,有了更加深入的理解。链表的编写,是理解在先的过程,只有当我们清楚的知道每一个指针现在正位于链表的哪个位置,它返回的指针正处于链表的哪个位置,我们才能让程序正常进行。因为一旦出现逻辑错误,编译器是不会报错的,但程序可能会陷入无限循环从而没有任何输出,让所有调试无从谈起。

同时,我更加明白了数据结构的选择对于解决一个问题的便捷性。例如第一个城市链表的问题,由于单链表的使用,单个城市信息的删除和插入变得异常轻易,不需要挪动大量的数据和改变很多存储结构,只需要修改结点即可达成目的。而之后的约瑟夫环问题,循环链表的使用回避了复杂的数学问题,我们可以通过遍历、循环直接粗暴的解决问题,同时将时间复杂度控制在可以接受的范围之内。而在选做作业的部分,我们可以选择用顺序结构实现线性表。当顺序结构需要插入和删除某个元素时,需要将该元素后面的所有元素都向后移动,十分复杂。

最后,在代码调试阶段,我更加明白了良好的编码习惯对于编写本身的重要意义。不管是括号的规则使用,还是及时的注释和代码分段,都能避免在后续调试阶段遇到太多复杂的问题。我们在调试代码的时候,心态一定要端正,要享受这个不断改正 bug 的过程,感受这个让自己正努力提高的过程。