**中南大学**

数据结构试验报告

题 目 线性表

学生姓名 张子洋

学 号 8208221223

指导老师 郑瑾

学 院 计算机学院

专业班级 计算机与通信类2212班

完成时间 2023.5

指导老师评定 签名

1.单向链表操作的实现（验证性实验）

1. 需求分析
2. 用头插法(或尾插法)建立带头结点的单向链表。
3. 对已建立的单向链表实现插入,删除,查找等基本操作。

这是一个简单的实验，单向链表是链表的一种，其特点是链表的连接方向是单向的，所以对链表的访问要通过顺序读取从头部开始，而从头部开始要从头结点下手。链表结点的组成包含data域和后继结点的指针域。基本操作包括：生成链表，添加节点，返回节点，删除节点，查找结点，遍历链表。

程序可实现下列操作：

尾插建立单链表，插入结点，删除结点，按元素值查找，求指定位置元素值，打印链表

。

1. 概要设计
2. 为了实现程序功能，需要定义单向链表的抽象数据类型。

* 单向链表的节点：
* typedef struct LNode {
* int data;
* struct LNode \*next;
* } LinkNode;
* 基本操作：
* 也为本实验定义的函数

void InitList(LinkNode \*L)

操作结果：初始化链表

void CreateListR(LinkNode \*&L, int a[], int n)

初始条件：单循环链表L存在

操作结果：使用头结点的尾插法

void DispList(LinkNode \*L)

初始条件：单循环链表L存在

操作结果：打印链表及元素

int GetElem(LinkNode \*L, int i)

初始条件：单循环链表L存在

操作结果：求元素的值

int LocateElem(LinkNode \*L, int e)

初始条件：单循环链表L存在

操作结果：按元素值查找

bool ListInsert(LinkNode \*L, int e, int i)

初始条件：单循环链表L存在

操作结果：插入数据元素

bool ListDelete(LinkNode \*L, int &e, int i)

初始条件：单循环链表L存在

操作结果：删除数据元素

本实验较为简单，只需调用上述函数对单向链表进行操作即可。

1. 详细设计

详细源代码如下：

1. # include <stdio.h>
2. # include <stdlib.h>
3. # define MaxSize 50
4. //链表
5. typedef struct LNode {
6. int data;
7. struct LNode \*next;
8. } LinkNode;
9. void InitList(LinkNode \*L) {
10. L = (LinkNode \*)malloc(sizeof(LinkNode));
11. L->next = NULL;
12. } //初始化
13. void CreateListR(LinkNode \*&L, int a[], int n) {
14. LinkNode \*s, \*r;
15. L = (LinkNode \*)malloc(sizeof(LinkNode));
16. r = L;
17. for (int i = 0; i < n; i++) //插入数目
18. {
19. s = (LinkNode \*)malloc(sizeof(LinkNode));//新结点
20. s->data = a[i];//数据域
21. r->next = s;
22. r = s;
23. }
24. r->next = NULL;
25. }//使用头结点的尾插法
26. void DestroyList(LinkNode \*L) {
27. LinkNode \*pre = NULL, \*p = L->next;
28. while (p != NULL) {
29. free(pre);
30. pre = p;
31. p=p->next;
32. }
33. free(pre);
34. }//销毁线性表
35. void DispList(LinkNode \*L) {
36. LinkNode \*p = L->next;
37. while (p != NULL) {
38. printf("%d ", p->data);
39. p = p->next;
40. }
41. printf("\n");
42. } //输出
43. int GetElem(LinkNode \*L, int i) {
44. int e;
45. int j;
46. LinkNode \*p = L;
47. if (i <= 0)
48. return 0;
49. while (j < i && p != NULL) {
50. j++;
51. p = p->next;
52. }
53. if (p == NULL) {
54. return 0;
55. } else {
56. e = p->data;
57. }
58. return e;
59. } //求某个元素值
60. int LocateElem(LinkNode \*L, int e) {
61. int i = 1;
62. LinkNode \*p = L->next;
63. while (p != NULL && p->data != e) {
64. p = p->next;
65. i++;
66. }
67. if (p == NULL) {
68. return 0;
69. } else
70. return i;
71. }//按元素值查找
72. bool ListInsert(LinkNode \*L, int e, int i) {
73. int j = 0;
74. LinkNode \*p = L, \*s;
75. if (i <= 0)
76. return false;
77. while (j < i - 1 && p != NULL) {
78. j++;
79. p = p->next;
80. }
81. if (p == NULL)
82. return false;
83. else {
84. s = (LinkNode \*)malloc(sizeof(LinkNode));
85. s->data = e;
86. s->next = p->next;
87. p->next = s;
88. return true;
89. }
90. }//插入数据元素
91. bool ListDelete(LinkNode \*L, int &e, int i) {
92. int j = 0;
93. LinkNode \*p = L, \*q;
94. if (i <= 0)
95. return false;
96. while (j < i - 1 && p != NULL) {
97. j++;
98. p = p->next;
99. }
100. if (p == NULL)
101. return false;
102. else {
103. q = p->next;
104. if (q == NULL)
105. return false;
106. e = q->data;
107. p->next = q->next;
108. free(q);
109. return true;
110. }
111. }//删除数据元素
112. int main() {
113. LinkNode \*test;
114. InitList(test);
115. printf("索引菜单：\n");
116. printf(" 1.尾插建立单链表\n");
117. printf(" 2.插入结点\n");
118. printf(" 3.删除结点\n");
119. printf(" 4.按元素值查找\n");
120. printf(" 5.求指定位置元素值\n");
121. printf(" 6.打印链表\n");
122. printf(" 7.销毁链表并退出\n");
123. do {
124. printf("请选择功能：\n");
125. int i, j, k, s[MaxSize], t, a, b, c, d, e, f;
126. scanf("%d", &i);
127. switch (i) {
128. case 1:
129. printf("请输入单链表所需要的元素个数：\n");
130. scanf("%d", &t);
131. printf("请输入所有元素：\n");
132. for (int u = 0; u < t; u++) {
133. scanf("%d", &s[u]);
134. }
135. CreateListR(test, s, t);
136. break;
137. case 2:
138. printf("请输入要插入的结点值以及位置：\n");
139. scanf("%d%d", &j, &k);
140. if (ListInsert(test, j, k))
141. printf("完成\n");
142. else
143. printf("输入数据不符合要求\n");
144. break;
145. case 3:
146. printf("请输入要删除的结点值以及位置：\n");
147. scanf("%d%d", &a, &b);
148. if (ListDelete(test, a, b))
149. printf("完成\n");
150. else
151. printf("输入数据不符合要求\n");
152. break;
153. case 4:
154. printf("请输入要查找的元素值:\n");
155. scanf("%d", &c);
156. d = LocateElem(test, c);
157. printf("位置为：%d\n", d);
158. break;
159. case 5:
160. printf("请输入元素所在位置：\n");
161. scanf("%d", &f);
162. e = GetElem(test, f);
163. printf("元素值：%d\n", e);
164. break;
165. case 6:
166. printf("线性表如下：\n");
167. DispList(test);
168. break;
169. case 7:
170. DestroyList(test);
171. printf("销毁成功！");
172. break;
173. }
174. } while (1);
175. return 0;
176. }

4.调试分析

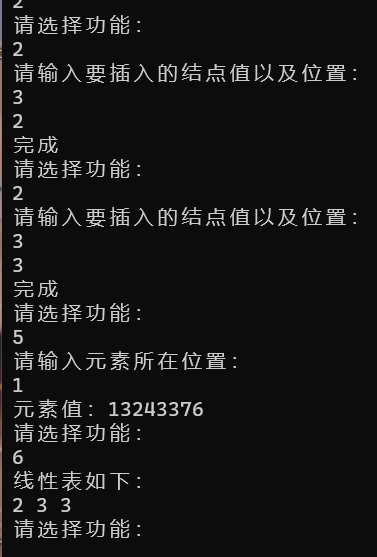
对设计和编码的讨论和分析。该程序实现了顺序栈的操作。分析程序代码的质量，主要从以下几个方面考虑。（见数据结构课本） 

* 正确性。在一定的数据范围内，该程序能实现所需功能，所以正确性是没有问题的。
* 健壮性。在一定的数据输入范围内，该程序能较好的实现链表的操作。但是如果输入数 据非法，该程序还是可能会产生一些预想不到的输出结构，或是不做任何处理。所以， 该程序的健壮性有待进一步的提高。要综合考虑一些情况，当输入有误时，应返回一个 表示错误的值，并中止程序的执行，以便在更高的抽象层次上进行处理。

1. 使用说明

按照屏幕提示，选择想要的功能并输入对应数字，按下ENTER键后，根据屏幕提示进行输入，即可得到想要的结果。

1. 测试程序的运行结果



1. 将数据插入链表
2. 查找
3. 删除
4. 查找元素值
5. 打印链表
6. 心得体会

  本次实验学习包括了单链表的创建，单链表的销毁，判断是否为空，输出线性表，取值，按值查找，插入，删除。其中最为重要的是插入和删除。通过画流程图对算法的了解更加深刻。这次试验使我对链表的使用有了更深的理解，对指针的运用更加熟练，熟悉了对线性表的定义和操作。

附录：源程序文件清单

各程序源代码文件随本实验报告电子版一起打包，存放在文件夹单向链表中。

文件清单如下：

单项链表.c…………………………………………单向链表操作的实现

2.城市链表（设计性实验）

1. 需求分析

需求：

将若干城市的信息存入一个带头结点的单向链表。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操作，做到

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。

分析：

每个节点包含以下信息：

城市名称

经度

纬度

下一个节点指针

添加城市：接受城市名称、经度和纬度作为参数，并将城市添加到链表的末尾。

删除城市：接受城市名称作为参数，在链表中查找并删除该城市节点。

查找城市：接受城市名称作为参数，返回链表中包含该城市名称的节点。

1. 概要设计
2. 为了实现程序功能，需要定义顺序栈的抽象数据类型。

typedef int Status ;

typedef struct Node{

char name[MAXINT] ; //存储城市名；

float x\_index, y\_index ; //存储城市坐标；

struct Node \*next ;

}Node, \*LinkList;

基本操作：

//创建一个链表

LinkList CreatList (int e)

操作结果：构造一个空的线性表L。

//打印出链表的信息

Status ShowList(LinkList l)

初始条件：线性表 L 已存在。

操作结果：打印出链表的信息。

//查找一个链表元素

LinkList SearchList(LinkList l)

初始条件：线性表 L 已存在。

操作结果：查找一个链表元素E。

//将城市插入链表头部

LinkList InsertList\_head (LinkList l, LinkList temp)

初始条件：线性表 L 已存在。

操作结果：将城市插入链表头部

//将城市插入任一城市之后

LinkList InsertList\_city (LinkList l, LinkList temp)

初始条件：线性表 L 已存在。

操作结果：将城市插入任一城市之后。

//插入一个链表元素

LinkList InsertList (LinkList l)

初始条件：线性表 L 已存在。

操作结果：插入一个链表元素 。

//删除一个链表元素

LinkList DeleteList(LinkList l

)

初始条件：线性表 L 已存在

操作结果：删除一个链表元素

//更新链表中的一个元素

LinkList RenewList(LinkList l)

初始条件：线性表 L 已存在。

操作结果：更新链表中的一个元素

线性表置空 ClearList( &L )

初始条件：线性表 L 已存在。

操作结果：将 L 重置为空表。

//进入城市距离计算比较功能

LinkList DistanceCity(LinkList l, float e)

初始条件：线性表 L 已存在

操作结果：进入城市距离计算比较功能

1. 详细设计
2. #include <stdio.h>
3. #include <stdlib.h>
4. #include <string.h>
5. #include <math.h>
6. #define MAXINT 10000
7. #define OK 1
8. #define ERROR 0
9. typedef int Status ;
10. typedef struct Node{
12. char name[MAXINT] ; //存储城市名；
13. float x\_index, y\_index ; //存储城市坐标；
14. struct Node \*next ;
16. }Node, \*LinkList;
17. //创建一个链表
18. LinkList CreatList (int e) {
20. LinkList l, tempt, p ;
21. l = (LinkList)malloc(sizeof(Node)) ;
22. l->next = NULL ;
24. tempt = l ;
26. int i = 0;
27. for(i = 0 ; i < e ; i ++) { //向链表中存储数据
28. p = (LinkList)malloc(sizeof(Node)) ;
29. scanf("%s %f %f", &p->name, &p->x\_index, &p->y\_index) ;
30. p->next = tempt->next ;
31. tempt->next = p ;
32. tempt = p ;
33. }
34. return l ;
35. }
36. //打印出链表的信息
37. Status ShowList(LinkList l) {
38. LinkList p ;
39. p = l ;
40. p = p->next ;
41. printf("\t\t\t\t\t城市信息为：") ;
42. printf("\n") ;
43. printf("\t\t\t\t\t-----------------------------------------------\n") ;
44. printf("\t\t\t\t\t序\t城市名称\t城市坐标\n", p->name, p->x\_index, p->y\_index) ;
45. int order = 1 ;
46. while(p) {
47. printf("\t\t\t\t\t%02d\t %-8s\t( %.2f°N, %.2f°E )\n",order, p->name, p->x\_index, p->y\_index) ;
48. p = p->next ;
49. order++ ;
50. }
51. printf("\t\t\t\t\t-----------------------------------------------\n" ) ;
52. return OK ;
53. }
54. //查找一个链表元素
55. LinkList SearchList(LinkList l){
56. LinkList q ;
57. q = (LinkList)malloc(sizeof(Node)) ;
58. q->next = NULL ;
59. scanf("%s", &q->name) ;
60. LinkList p ;
61. p = l ;
62. while(p) {
63. p = p->next ;
64. if(!strcmp(p->name, q->name)){
65. return p ;
66. }
67. if(p->next == NULL){
68. printf("没有查找到该城市信息") ;
69. return NULL ;
70. break ;
71. }
72. }
73. return 0;
74. }
75. //将城市插入链表头部
76. LinkList InsertList\_head (LinkList l, LinkList temp) {
77. Node \*p, \*q;
78. p = l ;
79. q = temp;
80. q = q->next ;
81. q->next = p->next ;
82. p->next = q ;
83. return l ;
84. }
85. //将城市插入任一城市之后
86. LinkList InsertList\_city (LinkList l, LinkList temp) {
87. printf("请输入希望插入其后的城市：") ;
88. Node \*index, \*q ;
89. index = SearchList(l) ;
90. q = temp ;
91. q = q->next ;
92. q->next = index->next ;
93. index->next = q;
94. return l ;
95. }
96. //插入一个链表元素
97. LinkList InsertList (LinkList l) {
98. // 插入城市信息
99. printf("\n") ;
100. printf("请选择您希望将该城市插入的位置：\n");
101. printf("A.将该城市插入至城市链表的头部\n") ;
102. printf("B.将该城市插入任一城市之后\n") ;
103. char c ;
104. scanf("%s", &c) ;
106. LinkList tempt ;
107. printf("请输入新插入的城市名：\n") ;
108. tempt = CreatList(1) ;
110. switch(c) {
111. case 'A':case 'a': l = InsertList\_head (l, tempt) ; break ;
112. case 'B':case 'b': l = InsertList\_city (l, tempt) ; break ;
113. }
114. return l ;
115. }
116. //删除一个链表元素
117. LinkList DeleteList(LinkList l){
118. printf("请输入你想要删除的城市名称：\n") ;
120. //已找到指向想要删除的城市的指针
121. Node \*index , \*p;
122. index = SearchList(l) ;
123. //进行删除操作
124. p = l ;
125. while(p->next != index) {
126. p = p->next ;
127. }
128. p->next = p->next->next ;
129. return l ;
130. }
131. //更新链表中的一个元素
132. LinkList RenewList(LinkList l) {
133. printf("请输入你想要更新的城市名称：\n") ;
135. //已找到指向想要更新的城市的指针
136. Node \*index , \*p;
137. index = SearchList(l) ;
138. //进行更新操作
139. printf("请输入新的城市名称及地理位置:\n") ;
140. scanf("%s %f %f",&index->name, &index->x\_index, &index->y\_index) ;
141. return l ;
142. }
143. //进入城市距离计算比较功能
144. LinkList DistanceCity(LinkList l, float e){
145. printf("请输入一个城市的名称作为定位标志:\n") ;
146. Node \*index , \*p, \*q;
147. index = SearchList(l) ;
149. //从列表中删除被选中的定位城市的信息
150. p = l ;
151. while(p->next != index) {
152. p = p->next ;
153. }
154. p->next = p->next->next ;
155. ShowList(l) ;
157. //计算列表中其他城市与定位城市的距离
158. int i = 0 ;
159. float dx = 0 , dy = 0 ;
160. float distance = 0 ;
161. 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 ;
162. Node \*d ; //一直指向dis链表末尾的指针;
163. dis = (LinkList)malloc(sizeof(Node)) ;
164. dis->next = NULL ;
165. d = dis ;
167. q = l ; //利用两个坐标点间的距离公式，开始计算
168. while(q){
169. q = q->next ;
170. dx = q->x\_index - index->x\_index ;
171. dy = q->y\_index - index->y\_index ;
172. distance = sqrt(dx \* dx + dy \* dy) ;
173. printf("%-8s距%-8s的距离为%.2f\n",q->name, index->name, distance) ;
174. if(distance <= e) {
175. LinkList temp ;
176. temp = (LinkList)malloc(sizeof(Node)) ;
177. strcpy(temp->name , q->name) ;
178. temp->x\_index = q->x\_index ;
179. temp->y\_index = q->y\_index ;
180. temp->next = d->next ;
181. d->next = temp ;
182. d = temp ;
183. }
184. if(q->next == NULL)
185. ShowList(dis) ;
186. }
187. return dis ;
188. }
189. Status Entrance(char c, LinkList l ){
191. //查找城市信息
192. if(c == 'A' || c == 'a') {
193. Node \*index ;
194. printf("请输入希望查询的城市：") ;
195. index = SearchList(l) ;
196. printf(" ( %.2f°N, %.2f°E )", index->x\_index, index->y\_index) ;
197. }
198. //插入城市信息
199. if(c == 'B' || c =='b') {
200. l = InsertList(l) ;
201. ShowList(l) ;
202. }
203. //删除城市信息
204. if(c == 'C' || c =='c') {
205. l = DeleteList(l) ;
206. ShowList(l) ;
207. }
208. //更新城市信息
209. if(c == 'D' || c =='d') {
210. l = RenewList(l) ;
211. ShowList(l) ;
212. }
214. //进入城市距离计算比较功能
215. if(c == 'E' || c =='e') {
216. float n ;
217. printf("请输入您想查询的距离范围的数值：\n") ;
218. scanf("%f", &n) ;
219. LinkList l\_ ;
220. l\_ = DistanceCity(l,n);
221. ShowList(l\_) ;
222. }
224. return OK ;
225. }
226. int main() {

229. printf("建立城市链表：") ;
230. int e = 0 ;
231. printf("请输入您想输入的城市数目:") ;
232. scanf("%d", &e) ;
234. //创建一个城市链表
235. LinkList l ;
236. printf("请输入城市名及其坐标：\n") ;
237. l = CreatList(e) ;
238. ShowList(l) ;
240. //入口
241. printf("进入城市链表操作\n") ;
242. printf("在创建城市链表后，可以进行如下操作\n") ;
243. printf("A.查找城市信息\n") ;
244. printf("B.插入城市信息\n") ;
245. printf("C.删除城市信息\n") ;
246. printf("D.更新城市信息\n") ;
247. printf("E.进入城市距离计算比较功能\n") ;
248. printf("F.退出操作\n") ;
249. while(1){
250. char c ;
251. printf("\n请输入操作数:\n") ;
252. scanf("%s", &c) ;
253. if(c == 'F' || c == 'f'){
254. printf("退出程序。\n") ;
255. break ; // 退出程序
256. }
257. Entrance(c, l) ;
258. }
260. return 0 ;
261. }

4.调试分析

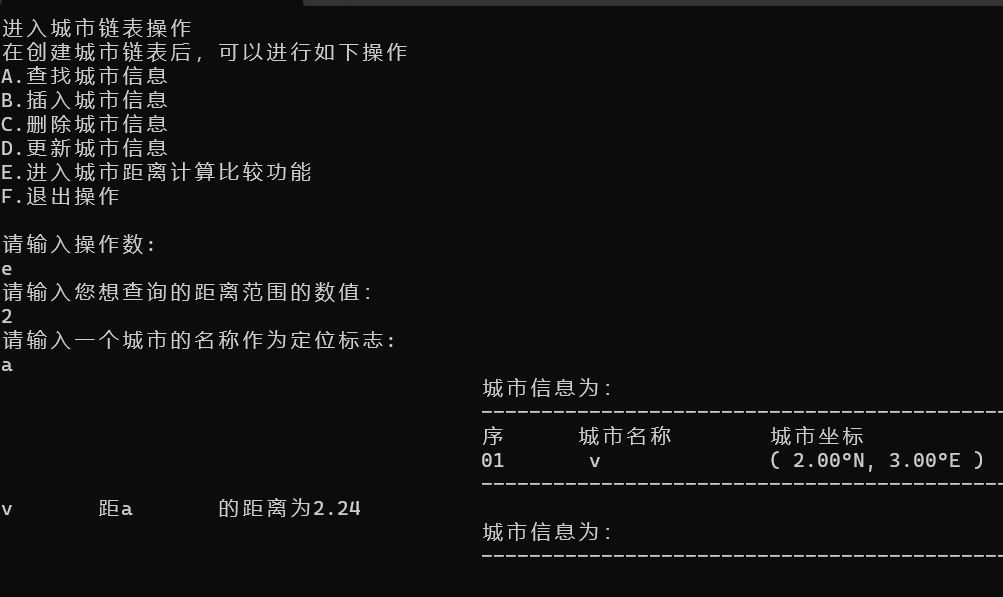
1. 采用IDE中自带的调试功能进行调试，手动添加断点和查看程序。
2. 对设计和编码的讨论和分析。该程序实现了城市链表的操作。分析程序代码的质量，主要从以下几个方面考虑。 

* 正确性。在一定的数据范围内，该程序能实现所需功能，所以正确性是没有问题的。
* 健壮性。在一定的数据输入范围内，该程序能较好的实现链表的操作。但是如果输入数 据非法，该程序还是可能会产生一些预想不到的输出结构，或是不做任何处理。所以， 该程序的健壮性有待进一步的提高。要综合考虑一些情况，当输入有误时，应返回一个 表示错误的值，并中止程序的执行，以便在更高的抽象层次上进行处理。

5.使用说明

按照屏幕提示，将数据放入对应文件中，按下ENTER键，再根据屏幕提示，选择想要的功能并输入对应数字，按下ENTER键后，根据屏幕提示进行输入，即可得到想要的结果。

6.测试程序的运行结果



1. 心得体会

在这个实验过程中，我为您提供了一个简单的城市链表实现方式，但在实际应用中，可能需要更多的功能和优化。例如，您可能需要增加对城市之间距离或交通方式等更多的信息进行存储和计算。为我提供更多深入学习数据结构和算法的思路和启示。

附录：源程序文件清单

各程序源代码文件随本实验报告电子版一起打包，存放在文件夹单向链表中。

文件清单如下：

城市链表.c…………………………………………城市链表操作的实现

3.约瑟夫环（综合性实验）

1. 需求分析

需求：

约瑟夫问题的一种描述是，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，位于他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。试设计一个程序求出出列顺序。要求： 利用单向循环链表存储结构模拟此过程，按照出列的顺序打印出每个人的编号。

基本要求 利用单向循环链表存储结构模拟此过程,按照出列的顺序打印出每个人的编号.

测试数据 m的初值为20,密码分别为3,1,7,2,4,8和4(正确的结果应为6,1,4,7,2,3和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,然后读取每个人的密码(设n≤30).

思考题 若m的值依据每个出列人手中所持的数而改变呢?

分析：

1. 输入的形式和输入值的范围。
2. 输出的形式：输出数个int型数字。
3. 程序所能达到的功能：

利用单向循环链表存储结构模拟约瑟夫环，按照出列的顺序打印出每个人的编号。

1. 测试数据：

两组数据，自行输入。

1. 概要设计
2. 为了实现程序功能，需要定义顺序栈的抽象数据类型。

ADT List {

数据对象： D＝{ ai | ai ∈ElemSet, i=1,2,...,n, n≥0 }

数据关系： R1＝{ <ai-1, ai >|ai-1, ai∈D, i=2,...,n }

typedef struct lnote{ //链表节点

int number; //编号

int data; //密码

struct lnote \*next;

}\*node,NODE;

typedef struct LIST{ //链表结构

node head;

node tail;

int len;

}list;

基本操作：

初始化操作 InitList( &L )

操作结果：构造一个空的线性表L。

结构销毁操作 DestroyList( &L )

初始条件：线性表 L 已存在。

操作结果：销毁线性表 L。

线性表判空操作 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。

求线性表的长度 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。

求数据元素的前驱 PriorElem( L, cur\_e, &pre\_e )

初始条件：线性表 L 已存在。

操作结果：若 cur\_e 是 L 的元素，则用pre\_e 返 回它的前驱，否则操作失败，pre\_e无 定义。

求数据元素的后继 NextElem( L, cur\_e, &next\_e )

初始条件：线性表 L 已存在。

操作结果：若 cur\_e 是 L 的元素，则用next\_e 返 回它的后继，否则操作失败，next\_e 无定义。

求线性表中第i个数据元素 GetElem( L, i, &e )

初始条件：线性表 L 已存在，并且 1≤i≤ListLength(L) 。

操作结果：用 e 返回 L 中第 i 个数据元素的值。

定位函数 LocateElem( L, e, compare( ) )

初始条件：线性表 L 已存在，e 为给定值， compare( ) 是元素判定函数。

操作结果：返回 L 中第 1 个与 e 满足关系 compare( ) 的元素的位序。 若这样的元素不存在，则返回值为 0。

遍历线性表 ListTraverse(L, visit( ))

初始条件：线性表 L 已存在。 visit( ) 为某个访问函数。

操作结果：依次对 L 中每个元素调用 函数visit( )。一旦 visit( )失败， 则操作失败。

线性表置空 ClearList( &L )

初始条件：线性表 L 已存在。

操作结果：将 L 重置为空表。

改变第 i 个数据元素的值 PutElem( &L, i, e )

初始条件：线性表 L 已存在，并且 1≤i≤ListLength(L) 。

操作结果：L 中第 i 个元素赋值 e 。

插入数据元素 ListInsert( &L, i, e )

初始条件：线性表 L 已存在，且 1≤i≤ListLength(L)+1 。

操作结果：在 L 中的第 i 个元素之前插入 新的元素 e，L 的长度增1。

删除数据元素 ListDelete( &L, i, &e )

初始条件：线性表 L 已存在并且非空， 且1≤i≤ListLength(L) 。

操作结果：删除 L中的第 i 个元素，并且用 e 返 回其值，L 的长度减1。

} ADT List

1. 本程序包含函数：

typedef struct lnode

{

int num;//序号

int password;//密码

int sign;//是否已出表的标志

struct lnode \*next;

}lnode, \*linklist;

lnode \*create(int n)

void run(lnode \*head,int n,int m)

1. 详细设计
2. #include<stdio.h>
3. #include<stdlib.h>
4. typedef struct lnode
5. {
6. int num;//序号
7. int password;//密码
8. int sign;//是否已出表的标志
9. struct lnode \*next;
10. }lnode, \*linklist;
12. lnode \*create(int n)
13. {
14. lnode \*head,\*p,\*z;
15. head=(lnode \*)malloc(sizeof(lnode));
16. z=head;
17. for(int i=1;i<=n;i++)
18. {
19. p=(lnode \*)malloc(sizeof(lnode));
20. p->num=i;//赋编号
21. printf("第%d个人手中的密码为：",i);
22. scanf("%d",&(p->password));//赋密码
23. p->sign=1;//赋标志，初始为1，可读，若为0，则跳过，表示已出表
24. p->next=z->next;//添加的p结点的指针勾连到head的指针
25. z->next=p;//head的指针指向p
26. z=p;//尾结点变更
27. }//n个人
28. z->next=head;
29. return head;
30. }
32. void run(lnode \*head,int n,int m)
33. {
34. lnode \*p;
35. int i;
36. p=head;
37. while(n>=1)
38. {
39. i=1;
40. while(i<=m)
41. {
42. p=p->next;//后移一位
43. if(p->sign==1)i++;//标志为1，读
44. }
45. printf("%d ",p->num);
46. m=p->password;//获得新的m值
47. p->sign=0;//出列后，赋0，不读
48. n--;
49. }
50. }
52. int main()
53. {
54. int n,m;//n个人，数到m的人出局
55. linklist head;
56. printf("共有几个人：n=");
57. scanf("%d",&n);
58. if(n<=0)printf("n的数值存在问题！");
59. printf("初始报数上限值：m=");
60. scanf("%d",&m);
61. if(m<=0)printf("n的数值存在问题！");
62. head=NULL;
63. head=create(n);
64. printf("出列顺序：");
65. run(head,n,m);
66. printf("\n");
67. return 0；}

4.调试分析

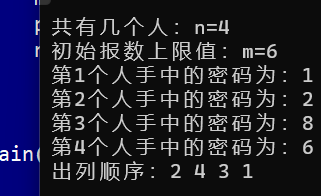
对设计和编码的讨论和分析。该程序实现了城市链表的操作。分析程序代码的质量，主要从以下几个方面考虑。 

* 正确性。在一定的数据范围内，该程序能实现所需功能，所以正确性是没有问题的。
* 健壮性。在一定的数据输入范围内，该程序能较好的实现链表的操作。但是如果输入数 据非法，该程序还是可能会产生一些预想不到的输出结构，或是不做任何处理。所以， 该程序的健壮性有待进一步的提高。要综合考虑一些情况，当输入有误时，应返回一个 表示错误的值，并中止程序的执行，以便在更高的抽象层次上进行处理。

5.使用说明

按照屏幕提示，将数据放入对应文件中，按下ENTER键，结果放在文件“约瑟夫环结果.txt”中。

1. 测试程序的运行结果



1. 心得体会

通过解决约瑟夫环问题，可以更好地理解循环链表及其应用。同时，这个问题也要求我们优化时间复杂度和空间复杂度，提高程序的运行效率。因此，解决这个问题既考察了我们的数据结构与算法知识，又锻炼了我们的编程能力。

在解决约瑟夫环问题时，我们需要使用循环链表的数据结构，并通过计算每一次的出队位置，递推得到最终的出队顺序。通过这种算法思想，我们可以将该问题的时间复杂度优化到 O(n)，极大地提高了解决问题时的效率。

总结：代码虽然简短，但充分利用了递归思想，非常有益。

附录：源程序文件清单

各程序源代码文件随本实验报告电子版一起打包，存放在文件夹约瑟夫环中。

文件清单如下：

约瑟夫环.c…………………………………………约瑟夫环过程的模拟