**数据结构第一次上机实验报告**

学号：161730126 姓名：李双玖

1. 调试成功程序及说明

**1、**

**题目：**

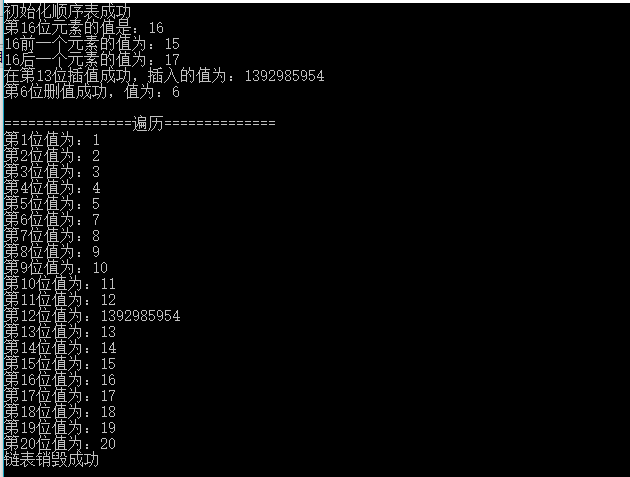
**编程实现书P19 ADT List 基本操作12个：**

**（1）用顺序存储结构实现； （2）用链式存储结构实现；**

1）顺序结构：

算法思想：a、采用动态数组的方式进行顺序储存，定义一个结构体来存储数组的首地址、长度和最大存储空间。当长度超过最大存储空间时，增加最大存储空间并用realloc重新分配空间；b、插入数据时，将该位置之后的数据都向后移一位。

运行结果：



结果分析：成功。插入删除各一次，最后还是20位数。

源程序：

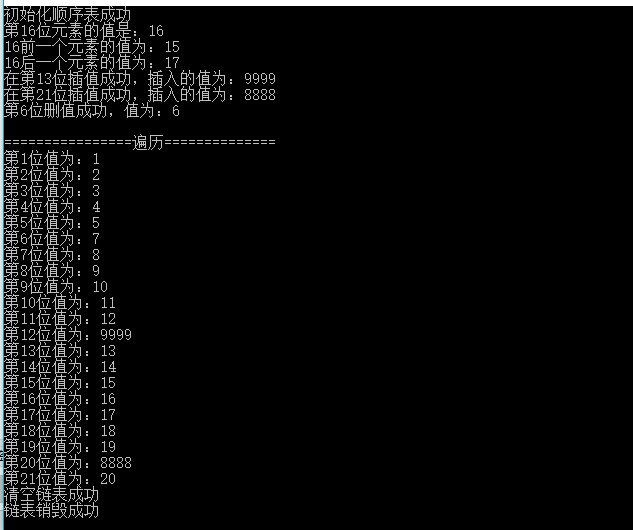
1. #include<iostream>
2. #include<stdlib.h>
3. **using** **namespace** std;
5. #define LIST\_INIT\_SIZE 100
6. #define LIST\_INCREMENT 10
8. // 顺序表头
9. **typedef** **struct** {
10. **int** \*head;
11. **int** length;
12. **int** listSize;
13. }Sqlist;
15. //顺序线性表类
16. **class** CSqList {
17. **public**:
19. //初始化线性表，即创建一个空表
20. **bool** InitList(Sqlist &list) {
21. **if**(list.head = (**int**\*)malloc(LIST\_INIT\_SIZE\***sizeof**(**int**))) {
22. list.length = 0;
23. list.listSize = LIST\_INIT\_SIZE;
24. **return** **true**;
25. }
26. **else** **return** **false**;
27. };
29. //销毁线性表，即free头指针
30. **bool** DestroyList(Sqlist &list) {
31. **if**(list.head) {
32. free(list.head);
33. list.head = NULL;
34. **return** **true**;
35. }
36. **else** **return** **false**;
37. };
39. //清空线性表，即令长度为零
40. **void** ClearList(Sqlist &list) {
41. list.length = 0;
42. };
44. //判断是否为空表
45. **bool** ListEmpty(Sqlist list) {
46. **return** list.length ? **false**:**true**;
47. };
49. //返回线性表的长度
50. **int** ListLength(Sqlist list) {
51. **return** list.length;
52. };
54. //返回线性表中第i个位置的值
55. **bool** GetElem(Sqlist list, **int** i, **int** &e) {
56. **if**(!list.length) **return** **false**;
57. **if**(i <= 0 || i > list.length) **return** **false**;
58. **else** e = list.head[i-1];
59. **return** **true**;
60. };
62. //返回线性表中值与e相等的位置
63. **int** LocateElem(Sqlist list, **int** e) {
64. **if**(!list.length) **return** 0;
65. **for**(**int** i = 0; i < list.length; i++) {
66. **if**(list.head[i] == e) **return** i+1;
67. }
68. **return** 0;
69. };
71. //返回值与cur\_e相同的前一个值
72. **bool** PriorElem(Sqlist list, **int** cur\_e, **int** &pre\_e) {
73. **if**(!list.length) **return** **false**;
74. **if**(list.head[0] == cur\_e) **return** **false**;
75. **for**(**int** i = 1; i < list.length; i++) {
76. **if**(list.head[i]  == cur\_e) {
77. pre\_e = list.head[i-1];
78. **return** **true**;
79. }
80. }
81. **return** **false**;
82. };
84. //返回值与cur\_e相同的下一个值
85. **bool** NextElem(Sqlist list, **int** cur\_e, **int** &next\_e) {
86. **if**(!list.length) **return** **false**;
87. **if**(list.head[list.length - 1] == cur\_e) **return** **false**;
88. **for**(**int** i = 0; i < list.length-1; i++) {
89. **if**(list.head[i]  == cur\_e) {
90. next\_e = list.head[i+1];
91. **return** **true**;
92. }
93. }
94. **return** **false**;
95. };
97. //在线性表的第i位之前插入e值
98. **bool** ListInsert(Sqlist &list, **int** i, **int** e) {
99. **if**(!list.length) **return** **false**;                  //判断表是否为空
100. **if**(i <= 0 || i > list.length) **return** **false**;       //判断i是否有效
101. **if**(list.length == list.listSize) {              //判断表存储空间是否已满
102. **int** \*newBase;
103. **if**(newBase = (**int**\*)realloc(list.head,
104. (list.listSize+LIST\_INCREMENT) \* **sizeof**(**int**))) {
105. list.head = newBase;
106. list.listSize += LIST\_INCREMENT;
107. }
108. **else** {
109. printf("error from ListInsert(): realloc defeat!\n");
110. **return** **false**;
111. }
112. }
113. **for**(**int** j = list.length-1; j >= i-1; j--) {      //将head[i-2]之后的数都向后移一位
114. list.head[j+1] = list.head[j];
115. }
116. list.head[i-1] = e;                             //把e赋给head[i-1]
117. list.length++;
118. **return** **true**;
119. };
121. //删除i位置的值，并用e返回其值
122. **bool** ListDelete(Sqlist &list, **int** i, **int** &e) {
123. **if**(!list.length) **return** **false**;                  //判断表是否为空
124. **if**(i <= 0 || i > list.length) **return** **false**;       //判断i是否有效
125. e = list.head[i-1];                             //把head[i-1]赋给e
126. **for**(**int** j = i; j < list.length; j++) {           //将head[i]之后的数都向前移一位
127. list.head[j-1] = list.head[j];
128. }
129. list.length--;
130. **return** **true**;
131. };
133. //遍历函数
134. **void** ListTraverse(Sqlist list) {
135. printf("\n================遍历==============\n");
136. **for**(**int** i = 0; i < list.length; i++)
137. printf("第%d位值为：%d\n", i+1, list.head[i]);
138. };
139. };

142. **int** main() {
143. CSqList List;//顺序表对象
144. Sqlist list;//顺序表头
146. **if**(List.InitList(list)) printf("初始化顺序表成功\n");
147. **else** printf("初始化顺序表失败\n");
148. **for**(**int** i = 0; i < 20; i++) {
149. list.head[i] = i+1;
150. list.length++;
151. }
153. **int** test = 0;
154. **if**(List.GetElem(list, 16, test)) printf("第16位元素的值是：%d\n", test);
155. **else** printf("访问第16位元素失败\n");
157. **int** test1 = 0;
158. **if**(List.PriorElem(list, test, test1)) printf("%d前一个元素的值为：%d\n", test, test1);
159. **else** printf("访问前一位元素失败\n");
161. **if**(List.NextElem(list, test, test1)) printf("%d后一个元素的值为：%d\n", test, test1);
162. **else** printf("访问后一位元素失败\n");
164. **if**(List.ListInsert(list, 13, 1392985954)) printf("在第13位插值成功，插入的值为：1392985954\n");
165. **else** printf("在第13位插值失败\n");
167. **if**(List.ListDelete(list, 6, test)) printf("第6位删值成功，值为：%d\n", test);
168. **else** printf("第6位删值失败\n");
170. List.ListTraverse(list);
172. **if**(List.DestroyList(list)) printf("链表销毁成功\n");
173. **else** printf("\n链表销毁失败\n");
174. }

2）链式结构：

算法思想：a、创建链表并赋初值，用两个指针，一个开辟空间，一个负责将新结点接入链表并向后移动；b、插入与删除结点，用两个指针，一个指向前一个结点，一个指向要操作的位置，遍历链表直到要操作的位置，然后开始插入删除。

运行结果：



结果分析：成功。插入两次、删除一次，刚好21位数。

源程序：

1. #include<iostream>
2. #include<stdlib.h>
3. **using** **namespace** std;
5. // 链式结点
6. **typedef** **struct** LNode{
7. **int** data;
8. LNode \*next;
9. }LNode, \*LinkList;
11. // 链式线性表类
12. **class** CLinkList {
13. **public**:
15. //初始化线性表，创建一个头结点，头结点的data数据表示链表长度
16. **bool** InitList(LinkList &list) {
17. **if**(list = (LNode\*)malloc(**sizeof**(LNode))) {
18. list->data = 0;
19. list->next = NULL;
20. **return** **true**;
21. }
22. **else** **return** **false**;
23. };
25. //销毁线性表，即free头指针
26. **bool** DestroyList(LinkList &list) {
27. **if**(list) {
28. **if**(ClearList(list)) {
29. printf("清空链表成功\n");
30. free(list);
31. **return** **true**;
32. }
33. **else** {
34. printf("清空链表失败\n");
35. **return** **false**;
36. }
37. }
38. **else** **return** **false**;
39. };
41. //清空线性表，将链表的结点都删除（除头结点之外）
42. **bool** ClearList(LinkList &list) {
43. **if**(list) {
44. LNode \*temp, \*p = list->next;
45. **while**(p) {
46. temp = p;
47. p = p->next;
48. free(temp);
49. }
50. list->data = 0;
51. list->next = NULL;
52. **return** **true**;
53. }
54. **return** **false**;
55. };
57. //判断是否为空表
58. **bool** ListEmpty(LinkList list) {
59. **if**(!list) **return** **false**;         //判断表是否为空
60. **return** list->data ? **false**:**true**;
61. };
63. //返回线性表的长度
64. **int** ListLength(LinkList list) {
65. **if**(!list) **return** 0;             //判断表是否为空
66. **return** list->data;
67. };
69. //返回线性表中第i个位置的结点
70. **bool** GetElem(LinkList list, **int** i, LNode \*e) {
71. **if**(!list || !list->data) **return** **false**;           //判断表是否为空
72. **if**(i <= 0 || i > list->data) **return** **false**;
73. **else** {
74. **int** j = 1;
75. LNode \*p = list->next;
76. **while**(p && j++!=i) p = p->next;
77. e->data = p->data;
78. e->next = p->next;
79. }
80. **return** **true**;
81. };
83. //返回线性表中值与e相等的位置
84. **int** LocateElem(LinkList list, LNode \*e) {
85. **if**(!list || !list->data) **return** 0;
86. LNode \*p = list->next;
87. **int** i = 1;
88. **while**(p) {
89. **if**(p->data == e->data) **return** i;
90. p = p->next;
91. }
92. **return** 0;
93. };
95. //返回值与cur\_e相同的前一个值
96. **bool** PriorElem(LinkList list, LNode \*cur\_e, LNode \*pre\_e) {
97. **if**(!list || !list->data) **return** **false**;           //判断表是否为空
98. LNode \*last = list, \*p = list->next;
99. **int** j = 1;
100. **while**(p) {
101. **if**(j == 1 && p->data == cur\_e->data) **return** **false**;
102. **if**(p->data == cur\_e->data) {
103. pre\_e->data = last->data;
104. pre\_e->next = last->next;
105. **return** **true**;
106. }
107. last = p;
108. p = last->next;
109. j++;
110. }
111. **return** **false**;
112. };
114. //返回值与cur\_e相同的下一个值
115. **bool** NextElem(LinkList list, LNode \*cur\_e, LNode \*next\_e) {
116. **if**(!list || !list->data) **return** **false**;           //判断表是否为空
117. LNode \*next, \*p = list;
118. **int** j = 1;
119. **while**(p) {
120. next = p->next;
121. **if**(j == list->data && p->data == cur\_e->data) **return** **false**;
122. **if**(p->data == cur\_e->data) {
123. next\_e->data = next->data;
124. next\_e->next = next->next;
125. **return** **true**;
126. }
127. p = p->next;
128. j++;
129. }
130. **return** **false**;
131. };
133. //在线性表的第i位插入e值
134. **bool** ListInsert(LinkList &list, **int** i, LNode \*e) {
135. **if**(!list) **return** **false**;         //判断表是否为空
136. **if**(i <= 0 || i > list->data+1) **return** **false**;       //判断i是否有效
138. //防止用户把传进的实参e多次使用
139. LNode \*temp = (LNode\*)malloc(**sizeof**(LNode));
140. temp->data = e->data;
141. temp->next;
143. LNode \*last = list, \*p = list->next;
144. **int** j = 1;
145. **while**(p && j++!=i) {
146. last = p;
147. p = last->next;
148. }
149. last->next = temp;
150. temp->next = p;
151. list->data++;
152. **return** **true**;
153. };
155. //删除i位置的值，并用e返回其值
156. **bool** ListDelete(LinkList &list, **int** i, LNode \*e) {
157. **if**(!list || !list->data) **return** **false**;           //判断表是否为空
158. **if**(i <= 0 || i > list->data) **return** **false**;     //判断i是否有效
159. LNode \*last = list, \*p = list->next;
160. **int** j = 1;
161. **while**(p && j++!=i) {
162. last = p;
163. p = last->next;
164. }
165. e->data = p->data;
166. e->next = NULL;
167. last->next = p->next;
168. free(p);
169. list->data--;
170. **return** **true**;
171. };
173. //遍历函数
174. **void** ListTraverse(LinkList list) {
175. printf("\n================遍历==============\n");
176. LNode \*p = list->next;
177. **int** i = 1;
178. **while**(p) {
179. printf("第%d位值为：%d\n", i, p->data);
180. p = p->next;
181. i++;
182. }
183. };
184. };

187. **int** main() {
188. CLinkList List;//顺序表对象
189. LinkList list;//顺序表头
191. **if**(List.InitList(list)) printf("初始化顺序表成功\n");
192. **else** printf("初始化顺序表失败\n");
194. **int** i = 1;
195. LNode \*temp, \*p=list;
196. **while**(i <= 20) {
197. temp = (LNode\*)malloc(**sizeof**(LNode));
198. temp->data = i;
199. temp->next = NULL;
200. p->next = temp;
201. p = p->next;
202. list->data++;
203. i++;
204. }
206. LNode \*test = (LNode\*)malloc(**sizeof**(LNode));
207. **if**(List.GetElem(list, 16, test)) printf("第16位元素的值是：%d\n", test->data);
208. **else** printf("访问第16位元素失败\n");
210. LNode \*test1 = (LNode\*)malloc(**sizeof**(LNode));
211. **if**(List.PriorElem(list, test, test1)) printf("%d前一个元素的值为：%d\n", test->data, test1->data);
212. **else** printf("访问前一位元素失败\n");
214. **if**(List.NextElem(list, test, test1)) printf("%d后一个元素的值为：%d\n", test->data, test1->data);
215. **else** printf("访问后一位元素失败\n");
217. test->data = 9999;
218. **if**(List.ListInsert(list, 13, test)) printf("在第13位插值成功，插入的值为：%d\n", test->data);
219. **else** printf("在第13位插值失败\n");
221. test->data = 8888;
222. **if**(List.ListInsert(list, 21, test)) printf("在第21位插值成功，插入的值为：%d\n", test->data);
223. **else** printf("在第21位插值失败\n");
225. **if**(List.ListDelete(list, 6, test)) printf("第6位删值成功，值为：%d\n", test->data);
226. **else** printf("第6位删值失败\n");
228. List.ListTraverse(list);
230. **if**(List.DestroyList(list)) printf("链表销毁成功\n");
231. **else** printf("链表销毁失败\n");
232. }

**2、**

**题目：设元素值为整型的线性表L，分别采用顺序结构和链式结构存储，编写程序，实现线性表的就地逆置**

1）顺序结构：

算法思想：将数组中第i位数与第n-1-i位数交换，i从0到n/2，即可完成倒置。算法时间复杂度为T(n/2)。

运行结果：



结果分析：成功倒置

源程序：

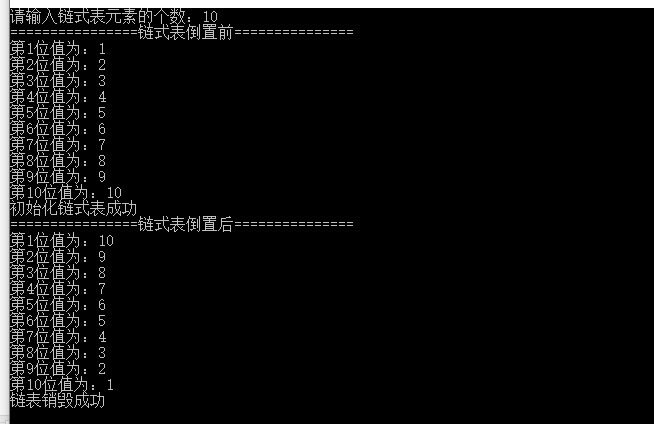
1. #include<iostream>
2. #include<stdlib.h>
3. **using** **namespace** std;
5. //顺序表头结构体
6. **typedef** **struct** {
7. **int** \*head;
8. **int** length;
9. }Sqlist;

12. //遍历函数
13. **void** ListTraverse(Sqlist list) {
14. **for**(**int** i = 0; i < list.length; i++)
15. printf("第%d位值为：%d\n", i+1, list.head[i]);
16. }
18. //初始化线性表
19. **bool** InitList(Sqlist &list, **int** n) {
20. **if**(list.head = (**int**\*)malloc(n\***sizeof**(**int**))) {
21. list.length = 0;
22. **for**(**int** i = 0; i < n; i++) {
23. list.head[i] = i+1;
24. list.length++;
25. }
26. printf("================顺序表倒置前===============\n");
27. ListTraverse(list);
28. **return** **true**;
29. }
30. **else** **return** **false**;
31. }
33. //销毁线性表，即free头指针
34. **bool** DestroyList(Sqlist &list) {
35. **if**(list.head) {
36. free(list.head);
37. list.head = NULL;
38. **return** **true**;
39. }
40. **else** **return** **false**;
41. }

44. **int** main() {
45. Sqlist list;//顺序表头
47. **int** num;
48. printf("请输入顺序表元素的个数：");
49. scanf("%d", &num);
51. **if**(InitList(list, num)) printf("初始化顺序表成功\n");
52. **else** printf("初始化顺序表失败\n");
54. **int** temp = 0;
55. **for**(**int** i = 0; i < list.length/2; i++) {
56. temp = list.head[i];
57. list.head[i] = list.head[list.length-i-1];
58. list.head[list.length-i-1] = temp;
59. }
60. printf("=====================顺表倒置后=================\n");
61. ListTraverse(list);
63. **if**(DestroyList(list)) printf("链表销毁成功\n");
64. **else** printf("链表销毁失败\n");
65. }
66. 链式结构：

算法思想：将第i个结点都插入头结点的下一个即链表的第一个位置（头插法），i从1到n。算法时间复杂度为T(n)。

运行结果：



结果分析：成功置换

源程序：

1. #include<iostream>
2. #include<stdlib.h>
3. **using** **namespace** std;
5. // 链式结点
6. **typedef** **struct** LNode{
7. **int** data;
8. LNode \*next;
9. }LNode, \*LinkList;

12. //遍历函数
13. **void** ListTraverse(LinkList list) {
14. LNode \*p = list->next;
15. **int** i = 1;
16. **while**(p) {
17. printf("第%d位值为：%d\n", i, p->data);
18. p = p->next;
19. i++;
20. }
21. }
23. //初始化线性表
24. **bool** InitList(LinkList &list, **int** num) {
25. **if**(list = (LNode\*)malloc(**sizeof**(LNode))) {
26. list->data = 0;
27. list->next = NULL;
28. LNode \*temp, \*p=list;
29. **for**(**int** i = 1; i <= num; i++) {
30. temp = (LNode\*)malloc(**sizeof**(LNode));
31. temp->data = i;
32. p->next = temp;
33. p = temp;
34. list->data++;
35. }
36. p->next = NULL;
37. printf("================链式表倒置前===============\n");
38. ListTraverse(list);
39. **return** **true**;
40. }
41. **else** **return** **false**;
42. }
44. //销毁线性表
45. **bool** DestroyList(LinkList &list) {
46. **if**(list) {
47. LNode \*temp, \*p = list->next;
48. **while**(p) {
49. temp = p;
50. p = p->next;
51. free(temp);
52. }
53. free(list);
54. **return** **true**;
55. }
56. **else** **return** **false**;
57. }
59. //在线性表的第i位插入e值
60. **bool** ListInsert(LinkList &list, **int** i, LNode \*e) {
61. **if**(!list) **return** **false**;         //判断表是否为空
62. **if**(i <= 0 || i > list->data+1) **return** **false**;       //判断i是否有效
64. LNode \*last = list, \*p = list->next;
65. **int** j = 1;
66. **while**(p && j++!=i) {                            //优化
67. last = p;
68. p = last->next;
69. }
70. last->next = e;
71. e->next = p;
72. list->data++;
73. **return** **true**;
74. }

77. **int** main() {
78. LinkList list;//顺序表头
80. **int** num;
81. printf("请输入链式表元素的个数：");
82. scanf("%d", &num);
84. **if**(InitList(list, num)) printf("初始化链式表成功\n");
85. **else** printf("初始化链式表失败\n");
87. LNode \*next = NULL, \*p = list->next;
88. list->data = 0;
89. list->next = NULL;
90. **while**(p) {
91. next = p->next;
92. **if**(!ListInsert(list, 1, p)) {
93. printf("插值失败\n");
94. **break**;
95. }
96. p = next;
97. }
99. printf("================链式表倒置后===============\n");
100. ListTraverse(list);
102. **if**(DestroyList(list)) printf("链表销毁成功\n");
103. **else** printf("链表销毁失败\n");
104. }

**3、**

**题目：输入正整数n、m（m<n），设有n个人坐成一圈，从第1个人开始循环报数，报到m的人出列，然后再从下一个人开始报数，报到m的人又出列，如此重复，直到所有的人都出列为止。要求用链式结构和顺序结构实现，按出列的先后顺序输出每个人的信息。**

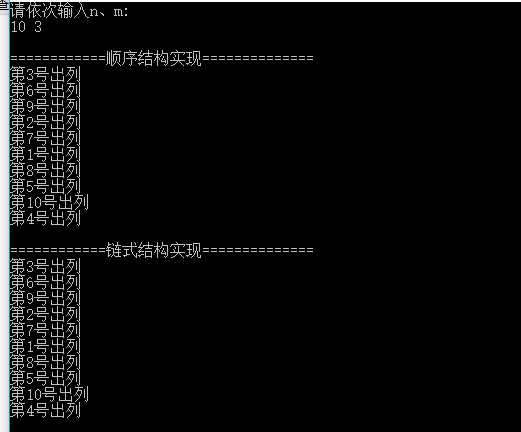
1）顺序结构：

算法思想：定义i、num两个数，i%n表示数组中的位置，num%m表示报数的结果。执行以下步骤：a、让i++，直到length==0；b、当num%m==0时，将该位置上数的值设为0并让顺序表的length--；c、如果该位置上数值为0则num不变，不为0则num++。算法时间复杂度T(n)。

2）链式结构·：

算法思想：采用单链表，让头结点的data部分存储链表的长度，同时定义两个结构体指针，一个指向前一个，一个指向当前。执行以下步骤：a、将两个指针向下移动同时num++，直到链表长度等于0；b、当num%m==0时，将该结点删除；c、当前指针指向NULL即达到链表尾时，将当前指针指向第一个结点，上一个指针指向头结点。算法时间复杂度T(n)。

运行结果：



结果分析：两种方式都成功，与样例结果相同。

源程序：

1. #include<stdio.h>
2. #include<stdlib.h>
3. #include<string.h>
5. // 顺序表头
6. **typedef** **struct** {
7. **int** \*head;
8. **int** length;
9. }SqList;
11. // 指针结点
12. **typedef** **struct** LNode {
13. **int** data;
14. **struct** LNode \*next;
15. }LNode,\*LinkList;

18. **int** main() {
20. **int** m = 0, n = 0;
21. printf("请依次输入n、m:\n");
22. scanf("%d%d", &n, &m);
24. printf("\n============顺序结构实现==============\n");
25. // 定义顺序链表
26. SqList List;
27. List.head = (**int**\*)malloc(**sizeof**(**int**) \* n);
28. List.length = n;
30. // 初始化数组值
31. **int** i = 0;
32. **while**(i < n) {
33. List.head[i] = i+1;
34. i++;
35. }
37. // 进行数组元素出列
38. i = 0;
39. **int** num = 1;
40. **while**(List.length) {
41. **if**(List.head[i%n]) {
42. **if**(num%m == 0) {
43. printf("第%d号出列\n", List.head[i%n]);
44. List.head[i%n] = 0;
45. List.length--;
46. }
47. num++;
48. }
49. i++;
50. }
52. // 释放空间
53. free(List.head);


57. printf("\n============链式结构实现==============\n");
58. // 定义链表
59. LinkList head;
60. // 定义使用到的指针
61. LNode \*p, \*temp;
62. // 创建头结点
63. head = (LNode\*)malloc(**sizeof**(LNode));
64. head->data = n;
66. // 创建链表，并为每个结点设立数值
67. p = head;
68. i = 0;
69. **while**(i++ < n) {
70. temp = (LNode\*)malloc(**sizeof**(LNode));
71. temp->data = i;
72. p->next = temp;
73. p = p->next;
74. }
75. p->next = NULL;
77. // 进行链表元素出列
78. num = 1;
79. LNode \*last = head;
80. **while**(head->data) {
81. **if**(!(last->next)) last = head;
82. p = last->next;
83. **if**(num%m == 0) {
84. printf("第%d号出列\n", p->data);
85. head->data--;
86. last->next = p->next;
87. free(p);
88. }
89. **else** last = last->next;
90. num++;
91. }
93. // 释放头结点
94. free(head);
95. }

**4、**

**题目：学生排队**

算法思想：采用单链表结构进行存储，初始化时每个结点的数据域保存该位置对应学号。定义两个结构体指针，一个指向前一个prior，一个指向当前p。执行以下步骤：a、当用户输入两个数，指针就开始遍历链表，并用j变量来记住当前位置；b、当p指向的结点的数据域与输入学号相等时，将p的next赋给prior的next。然后将p插入 j+移动距离 的位置上，此处用函数来实现。算法时间复杂度T(m)-T(m\*n)。

运行结果：



结果分析：与样例结果相同，成功

源程序：

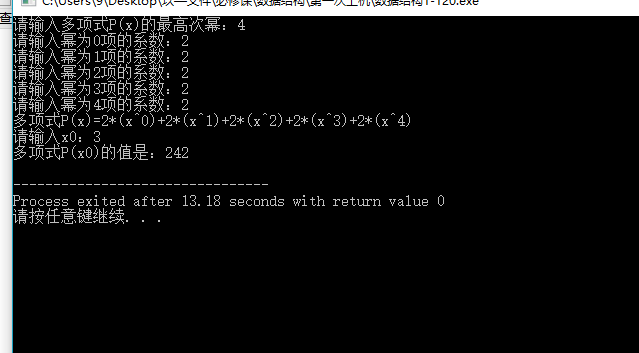
1. #include<stdio.h>
2. #include<stdlib.h>
4. **typedef** **struct** LNode{
5. **int** data;
6. LNode \*next;
7. }\*LinkList, LNode;
9. //在pos位置插入e
10. **bool** insertLinkList(LinkList &list, **int** pos, LNode \*e){
11. **if**(pos>list->data+1 || pos<=0) **return** **false**;
13. **int** i = 1;
14. LNode \*p = list->next, \*prior = list;
15. **while**(p && i<pos){
16. prior = p;
17. p = prior->next;
18. i++;
19. }
20. prior->next = e;
21. e->next = p;
22. list->data++;
23. **return** **true**;
24. }
26. //遍历函数
27. **void** ListTraverse(LinkList list) {
28. printf("\n================遍历==============\n");
29. LNode \*p = list->next;
30. **int** i = 1;
31. **while**(p) {
32. printf("第%d位值为：%d\n", i, p->data);
33. p = p->next;
34. i++;
35. }
36. printf("==================================\n\n");
37. };
39. **int** main(){
40. **int** n = 0, m = 0;
41. scanf("%d%d", &n, &m);
43. //初始化
44. LinkList list = (LNode\*)malloc(**sizeof**(LNode));
45. list->data = 0;
46. LNode \*temp, \*p = list;
47. **for**(**int** i=0; i<n; i++){
48. temp = (LNode\*)malloc(**sizeof**(LNode));
49. temp->data = i+1;                    //将学号赋给结点的数据域
50. p->next = temp;
51. p = p->next;
52. list->data++;
53. }
54. p->next = NULL;
55. ListTraverse(list);
57. //开始排队
58. **int** num = 0, pos = 0;
59. **int** j = 0;
60. LNode \*prior = NULL;
61. **for**(**int** i=1; i<=m; i++){
62. scanf("%d%d", &num, &pos);
63. p = list->next;
64. prior = list;
65. j = 1;
66. **while**(p){
67. **if**(p->data == num){              //找到等于该学号的结点
68. prior->next = p->next;        //将该结点取出
69. list->data--;
70. **if**(!insertLinkList(list, j+pos, p)) {   //再将该结点插入新位置
71. printf("insert error!\n");
72. }
73. }
74. prior = p;
75. p = prior->next;
76. j++;
77. }
78. }
79. ListTraverse(list);
81. //销毁链表
82. p = list;
83. **while**(p) {
84. temp = p;
85. p = p->next;
86. free(temp);
87. }
88. }

**5、**

**题目：求表达式的值，输入ai(i从0到n)、x0和n，输出表达式的在x=x0时的值。**

算法思想：采用顺序结构进行存储，并将各系数存在相对应数组位置中。先设表达式和sum为数组最后一个数乘上x，从数组最后一个数向前遍历，每次将sum乘上x再加数组中前一个数。算法时间复杂度T(n)。

运行结果：



结果分析：结果正确。

源程序：

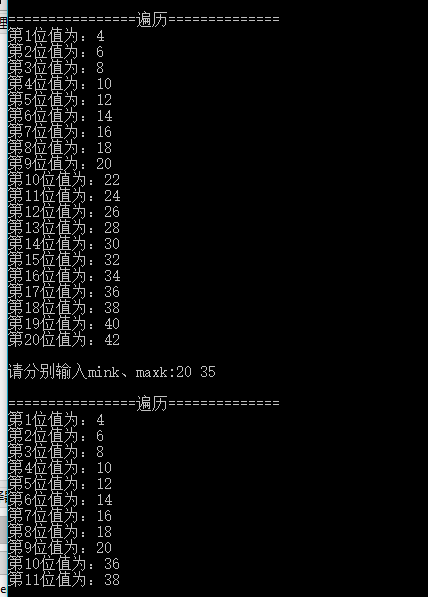
1. #include<stdio.h>
2. #include<stdlib.h>
4. **int** main() {
5. **int** n;
6. printf("请输入多项式P(x)的最高次幂：");
7. scanf("%d", &n);
9. **int** \*a = (**int**\*)malloc(**sizeof**(**int**) \* (n+1));
10. **for**(**int** i=0; i<=n; i++) {
11. printf("请输入幂为%d项的系数：", i);
12. scanf("%d", &a[i]);
13. }
15. printf("多项式P(x)=%d\*(x^%d)", a[0], 0);
16. **for**(**int** i=1; i<=n; i++) {
17. printf("+%d\*(x^%d)", a[i], i);
18. }
19. printf("\n");
21. **int** x0 = 0;
22. printf("请输入x0：");
23. scanf("%d", &x0);
24. **int** sum = a[n];
25. **for**(**int** i=n-1; i>=0; i--) {
26. sum = sum \* x0 + a[i];
27. }
29. printf("多项式P(x0)的值是：%d\n", sum);
30. }

**6、**

**题目：在链表中删除数值在mink和maxk之间的结点。**

算法思想：步骤分为三步：a、在p不为空且p->data<=mink的情况下，向下依次遍历链表结点，并用prior记住p的上一个结点。此次遍历结束后，将prior赋给temp指针。b、接着上一步在p不为空且p->data<maxk的情况下，向下遍历，依旧用prior记住上一个结点。同时在遍历过程中free掉prior指向的结点。c、当此次循环结束后，将p赋给temp->next即可。算法时间复杂度T(n)。

运行结果：



结果分析：正确

源程序：

1. #include<stdio.h>
2. #include<stdlib.h>
4. **typedef** **struct** LNode {
5. **int** data;
6. LNode \*next;
7. }LNode, \*LinkList;
9. //遍历函数
10. **void** ListTraverse(LinkList list) {
11. printf("\n================遍历==============\n");
12. LNode \*p = list->next;
13. **int** i = 1;
14. **while**(p) {
15. printf("第%d位值为：%d\n", i, p->data);
16. p = p->next;
17. i++;
18. }
19. }
21. **int** main() {
22. // 初始化链表
23. LinkList list = (LNode\*)malloc(**sizeof**(LNode));
24. list->data = 0;
25. list->next = NULL;
26. LNode \*p, \*temp;
27. p = list;
28. **int** i = 0;
29. **while**(i<20) {
30. temp = (LNode\*)malloc(**sizeof**(LNode));
31. temp->data = 2\*i+4;
32. p->next = temp;
33. p = p->next;
34. i++;
35. }
36. p->next = NULL;
37. ListTraverse(list);
39. // 输入上下界
40. **int** mink = 0, maxk = 0;
41. printf("\n请分别输入mink、maxk:");
42. scanf("%d%d", &mink, &maxk);
44. // 开始删除元素
45. LNode \*prior;
46. prior = list;
47. p = list->next;
48. **while**(p && p->data<=mink) {
49. prior = p;
50. p = p->next;
51. }
52. temp = prior;
53. **while**(p && p->data<maxk) {
54. prior = p;
55. p = p->next;
56. free(prior);
57. }
58. temp->next = p;
59. ListTraverse(list);
61. //销毁链表
62. p = list;
63. **while**(p) {
64. temp = p;
65. p = p->next;
66. free(temp);
67. }
68. }