中北大学软件学院

**实验报告**

|  |  |
| --- | --- |
| 专 业： | 软件工程 |
| 课程名称： | 数据结构与算法 |
| 班 级： | 21130420 |
| 学 号： | 2113042015 |
| 姓 名： | 程楚晋 |
| 辅导教师： | 马巧梅 |

2022年3月制

成绩：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 实验时间 | 2022年3月29日8时至10时 | 学时数 | 2学时 | |
| 1.实验名称  线性表的基本操作及应用 | | | | |
| 2.实验目的  （1）掌握单链表的创建，插入、删除、查找和打印算法；  （2）运用线性表解决线性结构问题。 | | | | |
| 3.实验内容  **基本要求：**  （1）实现单链表的创建；（2）实现单链表的插入；（3）实现单链表的删除  （4）实现单链表的查找；（5）实现单链表的显示；  **选作内容：**  两个线性表合并算法的实现。已知顺序表LA和LB中的数据元素按值非递减有序排列，现要将LA和LB归并为一个新的顺序表LC，且LC中的数据元素仍按值非递减有序排序。例如：LA＝（3,5,8,11） LB＝（2,6,9,15,20）。 | | | | |
| 4.线性链表的逻辑状态（此处仿照书上P28图2.6画出所建链表的逻辑状态）    通常我们把链表画成由箭头相链接的序列，结点之间的箭头表示链域中的指针，任何两个元素之间没有固定的联系。然而，每个元素的储存位置都包含在其直接前驱结点的信息之中。因此，单链表是非随机存取的储存结构。 | | | | |
| 5.实验源代码   1. #include<stdio.h> 2. #include<stdlib.h> 3. #define OK 1 4. #define ERROR 0 5. **typedef** **int** Status; 6. **typedef** **float** ElemType; 7. **typedef** **struct** LNode{ 8. ElemType data; 9. LNode \*next; 10. }LNode , \*LinkList; 11. //或者可以这样写 12. /\*struct LNode{ 13. ElemType data; 14. struct LNode \*next; 15. }; 16. typedef struct LNode LNode; 17. typedef LNode \*LinkList;\*/ 19. //创建的单链表，头插法 20. **void** CreatList\_L(LinkList &L,**int** n) //输入n个元素的值 21. { 22. LNode \*p;  **int** i; 23. L=(LinkList)malloc(**sizeof** (LNode)); 24. L->next=NULL;  //先建立一个带头结点的空链表 25. **for**(i=n;i>0;--i) 26. { 27. p=(LinkList)malloc(**sizeof** (LNode)); 28. scanf("%f",&p->data);   //已经带了输入，所以在主函数里面不要再写输入数据 29. p->next=L->next; 30. L->next=p; 31. } 32. } 33. //尾插法 34. **void** CreateListTail(LinkList &L,**int** n) 35. { 36. LinkList  r=L=(LinkList)malloc(**sizeof** (LNode)); 37. L->next=NULL; 38. **for**(**int** i=1;i<=n;++i) 39. { 40. LinkList s=(LinkList) malloc(**sizeof** (LNode)); 41. scanf("%f",&s->data); 42. s->next=NULL; 43. r->next=s; 44. r=s; 45. } 46. } 47. //单链表的插入 48. **int** Listlnsert\_L(LinkList &L, **int** i, ElemType e) 49. {     //带头结点的单链表L中第i个位置之前插入元素e 50. LNode \*p,\*s; 51. **int** j=0; 52. p=L; 53. **while**(p&&j<i-1) 54. { 55. p=p->next;  ++j; 56. } 57. **if**(!p || j>i-1) **return** ERROR; 58. s=(LinkList)malloc(**sizeof**(LNode));  //生成新的结点 59. s->data=e;         //使新结点数据域的值为e 60. s->next=p->next;    //将新结点插入到弹量表L中 61. p->next=s;       //修改第i-1个结点指针 62. **return** OK; 63. } 64. //取第i个元素 65. Status GetElem\_L(LinkList L,**int** i,ElemType &e) 66. {   //L是带头节点的链表的头指针，以e返回第i个元素 67. LNode \*p;  **int** j=1;  //j是计数器 68. p=L->next; 69. **while**(p && j<i) 70. { 71. p=p->next;    ++j; 72. } 73. **if**(!p||j>i) 74. **return** ERROR; 75. e=p->data; 76. **return** OK; 77. } 78. //单链表的删除 79. **int** ListDelete\_L(LinkList &L,**int** i,ElemType &e) 80. {   //删除第i个元素，并由e返回其值 81. LNode \*p, \*q; 82. **int** j = 0; 83. p = L; 84. **while** (p->next && j < i - 1) { 85. p = p->next; 86. ++j; 87. } 88. **if** (!(p->next) || j > i - 1) **return** ERROR;   //删除位置不合理 89. q = p->next;  //用指针q指向被删除的结点 90. p->next = q->next;  //删除第i个结点 91. e = q->data;     //取出第i个结点的数据域值 92. free(q);      //释放第i个结点，避免出现空结点 93. **return** OK; 94. } 95. **void** MergeList\_L(LinkList &La,LinkList &Lb,LinkList &Lc) 96. // 或者void MergeList\_L(LinkList La,LinkList Lb,LinkList Lc) 97. {    //归并La和Lb得到新的单链表Lc,Lc的元素也按非递减排列 98. LNode \*pa,\*pb,\*pc; 99. pa=La->next;  pb=Lb->next; 100. Lc = pc = La;   //用La的头节点作为Lc的头节点 101. **while**(pa && pb) 102. { 103. **if**(pa->data<=pb->data) 104. {    //如果pa->data<=pb->data 105. pc->next=pa;    pc=pa;   pa = pa->next; 106. } 107. **else**{ 108. pc->next=pb;  pc=pb;   pb = pb->next; 109. } 110. pc->next = pa?pa:pb;  //插入剩余段 111. free(Lb); 112. } 113. } 114. //单链表的输出 115. **void** List\_printf(LinkList &L) 116. { 117. LNode \*p=L->next; 118. **while**(p) { 119. printf("%f ", p->data);   //输出插入的元素 120. p=p->next; 121. } 122. } 124. **int** main() 125. { 126. **int** i,j; 127. LinkList LA,LB,LC; 128. ElemType a,b,e; 129. scanf("%f",&a);    //创建的单链表的长度 130. CreatList\_L(LA, a);    //创建一个单链表 131. scanf("%f",&b); //插入链表中的元素个数 132. scanf("%d",&i);  //插入位置 133. **for**(j=0;j<b;j++) {    //插入的元素 134. scanf("%f", &e); 135. Listlnsert\_L(LA, i, e); 136. } 137. List\_printf(LA); 138. /\*  LNode \*p=LA->next;    //或者写LinnkList p=L->next; 139. while(p)  // 或者while(p!=NULL) 140. { 141. printf("%f ", p->data);   //输出插入的元素 142. p=p->next; 143. }   //此方法为链表的输出方法    \*/ 144. printf("\n"); 145. printf("输入要查找的元素："); 146. scanf("%d",&i); 147. GetElem\_L(LA,i,e); 148. printf("查找的第%d个元素是%f\n",i,e); 149. printf("输入要删除的元素位置"); 150. scanf("%d",&i); 151. ListDelete\_L(LA,i,e); 152. printf("删掉的元素是%f",e); 153. printf("\n"); 154. scanf("%f",&a);    //LB的链表长度 155. CreateListTail(LB, a);   //用尾插法创建单链表LB 156. printf("输出单链表LB\n"); 157. List\_printf(LB); 158. printf("\n"); 159. printf("输出归并后的单链表LC\n"); 160. MergeList\_L(LA,LB,LC); 161. List\_printf(LC); 162. **return** 0; 163. }   代码运行结果： | | | |
| 6.实验结论及心得  1.单链表的查找中，不能用p++因为其储存并不是连续的，而应该用list-printf函数的形式进行输出：  2.插入时，指针的顺序不能更改：  3.每次删除时，要free掉删除的空间，防止出现空指针，节省空间：  4.单链表的创建时，有两种方法，一种是头插法，头插法的好处是，插入前去就是L，指针使用的少，但是插入后，插入顺序和单链表顺序是倒序排列的。另一种是尾插法，输入序列和生成序列顺序一致：  5.单链表的操作有很多，比如，在创建单链表的时候同时也可以进行排序操作，可以使输入的数据按照顺序排列，也可以进行统计单链表长度等操作。 | | | |