

《数据结构》上机实验报告

第 1 次上机

学号：

姓名： yyh

学院： 信息科学学院

专业： 计算机科学

教师：

日期： 2018.9.12

1. 实验要求

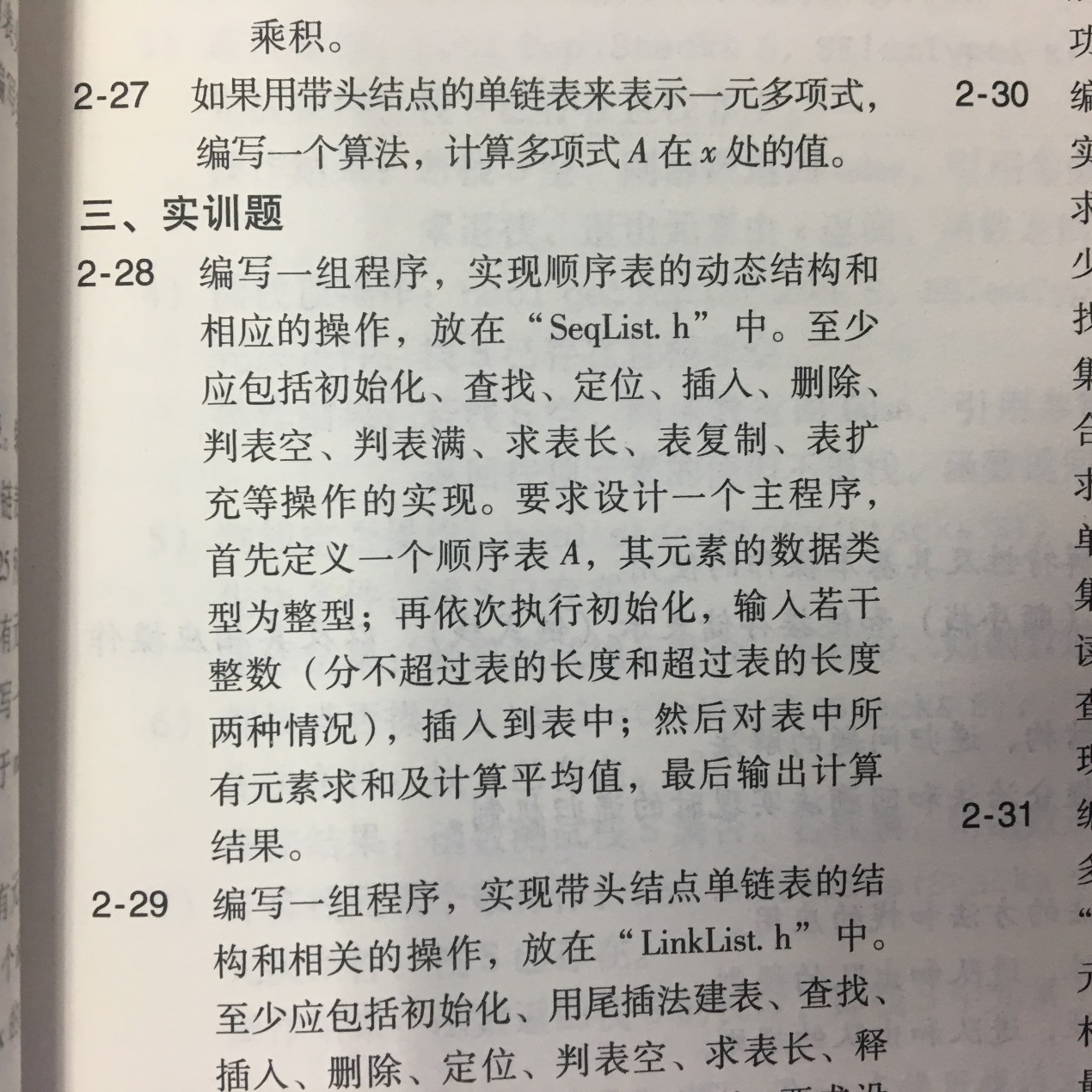
1.上机之前应做好充分准备，认真思考所需的上机题目，提高上机效率。

2.独立上机输入和调试自己所编的程序，切忌抄袭、拷贝他人程序。

3.上机结束后，整理出实验报告。书写报告时，重点放在实验的方法、思路以及总结反思上，以达到巩固课堂学习、提高动手能力的目的。

1. 实验内容

课本练习2-28



1. 实验步骤（写出问题分析或者算法思路）

思路：因为涉及到数组大小的扩充，所以考虑使用顺序表的动态结构定义。

课上和ppt里已经有关于动态结构初始化、查找、插入、删除的思想方法和具体实现，这次也借鉴了很多。分别见InitList, Find, Inset和Remove函数。

定位使用的是类似普通静态数组的用下标直接访问的方法，但是要注意不要越界访问（需要特判）。见Locate函数。

判表空、判表满和求表长，关注n就好了，一则判断n是否为0，二则比较n和maxSize的大小，三则就是n。见isFull、isEmpty和len函数。

表复制，考虑到maxSize可能比initsize大，所以不能直接使用initail函数初始化复制体，需要更改maxsize。见Copy函数。

表扩充，主要思想就是新建一个更大尺寸的表存有原始数据，然后复制大的新表给原表。新建大表和表的初始化类似，但要更改maxSize，然后要把原表的值赋给新表，最后Copy一下新表到原表就可以了。见Expansion函数。特别注意的是，我把扩充函数整合到了插入（insert）函数里，所以insert可以随意插入，一般不会出现插不进去的情况。

表求和，与数组求和类似，就是遍历访问，返回结果。见Sum函数。

1. 程序清单（源程序代码等）

Seqlish.h:

1. #include<malloc.h>
2. #include<stdio.h>
3. #define initSize 100            //最大允许长度
4. **typedef** **int** DataType;   //元素的数据类型
6. **typedef** **struct**
7. {
8. DataType \*data;          //存储数组
9. **int** n;                   //当前表元素个数
10. **int** maxSize;         //表的最大长度
11. } SeqList;
13. **void** InitList (SeqList& L)
14. {
15. L.data=(DataType\*)malloc(initSize\***sizeof**( DataType));
16. **if** ( L.data == NULL)
17. {
18. printf ("存储分配失败!\n");
19. exit (1);
20. }
21. L.n = 0;  L.maxSize = initSize;
22. }
24. **int** Find ( SeqList& L, DataType x )
25. {
26. **for** ( **int** i = 0; i < L.n; i++ )
27. **if** ( L.data[i] == x ) **return** i;     //查找成功
28. **return** -1;                    //查找失败
29. }
31. **bool** Insert ( SeqList& L, DataType x, **int** i )
32. {
33. **bool** Expansion(SeqList &L);
34. //在表中第 i (1≤i≤n+1) 个位置插入新元素 x
35. **if** ( L.n == L.maxSize )
36. Expansion(L);   //表满则进行扩充
37. **if** ( i < 1 || i > L.n+1 )
38. **return** **false**;
39. **for** ( **int** j = L.n-1; j >= i-1; j--)
40. L.data[j+1] = L.data[j];
41. L.data[i-1] = x;        //实际插在第i-1个位置
42. L.n++;
43. **return** **true**;    //插入成功
44. }
46. **bool** Remove ( SeqList& L, **int** i, DataType& x )
47. {
48. //在表中删除第 i 个元素，通过 x 返回其值
49. **if** ( L.n > 0 && i > 0 && i <= L.n )
50. {
51. x = L.data[i-1];
52. **for** (**int** j = i; j < L.n; j++)
53. L.data[j-1] = L.data[j];
54. L.n--;
55. **return** **true**;    //删除成功
56. }
57. **else** **return** **false**;                 //删除失败
58. }
60. **bool** isEmpty(SeqList& L)
61. {
62. **if** ( L.n ==0 )
63. **return** **true**;    //表空
64. **else** **return** **false**;
65. }
67. **bool** isFull(SeqList& L)
68. {
69. **if** ( L.n == L.maxSize)
70. **return** **true**;    //表满
71. **else** **return** **false**;
72. }
74. **int** Len(SeqList& L)
75. {**return** L.n;}
77. **void** Copy(SeqList &L1,SeqList &L2)  //把L2复制到L1
78. {
79. InitList(L1);
80. L1.n=L2.n;
81. L1.maxSize=L2.maxSize;
82. L1.data=(DataType\*)malloc(L1.maxSize\***sizeof**( DataType));
83. **for**(**int** i=0;i<L2.n;i++)
84. {
85. Insert(L1,L2.data[i],1+i);
86. }
87. **return** ;
88. }
90. **bool** Expansion(SeqList &L)
91. {
92. **int** tem=L.maxSize;
93. tem+=initSize;  //每次加initSize的大小
95. SeqList L2;
96. L2.data=(DataType\*)malloc(tem\***sizeof**( DataType));
97. **if** ( L2.data == NULL)
98. {
99. printf ("扩充存储分配失败!\n");
100. exit (1);
101. }
102. **for**(**int** i=0;i<L.n;i++)
103. L2.data[i]=L.data[i];
105. //free(L.data);
106. L.data=(DataType\*)malloc(tem\***sizeof**( DataType));
107. **if** ( L.data == NULL)
108. {
109. printf ("扩充存储分配失败!\n");
110. exit (1);
111. }
112. **for**(**int** i=0;i<L.n;i++)
113. L.data[i]=L2.data[i];
114. L.maxSize=tem;
115. **return** **true**;
116. //free(L2);
117. }
119. **int** Sum(SeqList L)
120. {
121. **int** sum=0;
122. **for**(**int** i=0;i<L.n;i++)
123. sum+=L.data[i];
124. **return** sum;
125. }

main.cpp:

1. #include "SeqLish.h"
2. #include<iostream>
3. **using** **namespace** std;
4. **int** main ()
5. {
6. SeqList A,B;
7. **int** n,tem;
8. freopen("test.txt","r",stdin);
9. **while**(~scanf("%d",&n))
10. {
11. InitList(A);
12. **for**(**int** i=0;i<n;i++)
13. {
14. scanf("%d",&tem);
15. Insert(A,tem,i+1);
16. }
17. Copy(B,A);
18. printf("A=%f\n",1.0\*Sum(A)/A.n);
19. printf("B=%f\tB是A的复制\n",1.0\*Sum(B)/B.n);    //B是A的复制
20. }
21. **return** 0;
22. }

test.txt:

5

1 2 3 4 5

10

1 2 3 4 5 6 7 8 9 10

200

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100 101 102 103 104 105 106 107 108 109 110 111 112 113 114 115 116 117 118 119 120 121 122 123 124 125 126 127 128 129 130 131 132 133 134 135 136 137 138 139 140 141 142 143 144 145 146 147 148 149 150 151 152 153 154 155 156 157 158 159 160 161 162 163 164 165 166 167 168 169 170 171 172 173 174 175 176 177 178 179 180 181 182 183 184 185 186 187 188 189 190 191 192 193 194 195 196 197 198 199 200

300

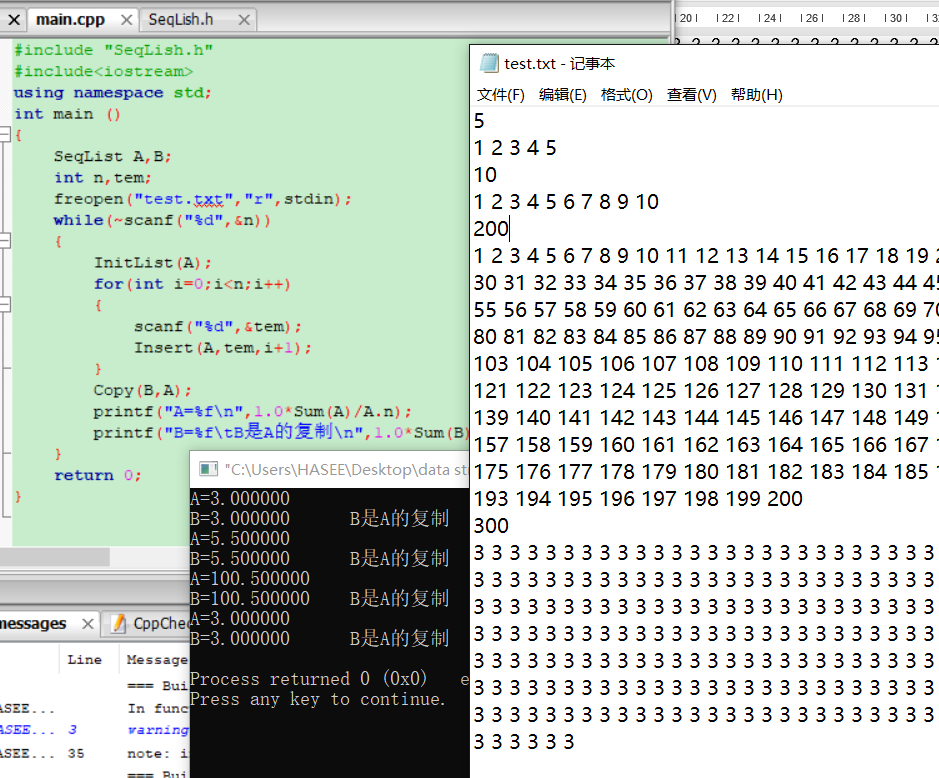
3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3

1. 运行结果（程序运行时的结果说明或运行截图等）

使用了freopen函数来读入测试数据“tset.txt”。

同时测试了一下Copy函数，注意第三个和第四个样例是超过了初始大小100的。

输出的值即为平均数。



1. 总结（实验中遇到的问题、取得的经验、感想等）

1 使用应用型参数&可以直接对原变量（数组、指针等）进行修改。

引用型参数“&”是把形参 L 看作是实际变量（一个表）的别名，在函数体内对 L 的操作将直接对实际变量的操作。使用它可以不必创建实际变量的副本空间，可直接从实际变量得到操作结果，还可在函数体内像普通变量那样对 L 操作，使得操作简单。

2 动态顺序表的插入、删除、扩充、求和、查找都是O(n)的。但定位和修改值却是O（1）的，可以用在经常需要定位和修改的地方，比如成绩簿，知道学号（下标）查战绩和改成绩等。

3 在表的扩充中，可以考虑使用memcpy函数来加快运行速度。

4 表的初始最大值并不是越大越好、每一次扩充的大小也不是越大越好，因为可能会造成空间的浪费。但也不是越小越好，因为可能在相同空间大小需求下，单此越小扩充空间会进行更多次扩充操作，从而造成时间上的浪费。要空间还是时间，这里就需要实践来指导了。

5 每一次读入新数据前，一定要进行初始化。