《数据结构》上机报告

2018 年 9 月 24 日

姓名： 赵得泽 学号： 1753642 班级： 电子 2 班 得分：

<table>
<tr><td>实验题目</td><td colspan="2">顺序表</td></tr>
<tr><td>问题描述</td><td colspan="2">顺序表是指采用顺序存储结构的线性表，它利用内存中的一片连续存储区域存放表中的所有元素。可以根据需要对表中的所有数据进行访问，元素的插入和删除可以在表中的任何位置进行。</td></tr>
<tr><td rowspan="2">基本要求</td><td colspan="2">1. (p1)实现顺序表的基本操作，包括顺序表的初始化、第 i 个元素前插入一个新的元素、删除第 i 个元素、查找某元素、顺序表的销毁。
2. (p2)实现对有序表（非递减）插入元素、删除元素、查找元素的功能
3. (p3)实现两个有序（非递减）表合并生成新的有序表的功能
4. (p4)顺序查找顺序表，删除元素 e（删除所有值为 e 的元素）。
5. (p5)删除顺序表中所有多余的元素（即没有相同的元素）。</td></tr>
<tr><td>已完成基本内容（序号）：</td><td>1，2，3，4，5</td></tr>
<tr><td rowspan="2">选做要求</td><td colspan="2"></td></tr>
<tr><td>已完成选做内容（序号）</td><td></td></tr>
<tr><td>数据结构设计</td><td colspan="2">

```
typedef struct
{
    int *elem;
    int length;
}SeqList;
SeqList SL;
```

本次实验的数据结构主要是线性表中的有序表的构建，在构造的结构体中有表长和动态数组 2 个成员，通过该结构体实现顺序存储。顺序存储结构是借助元素在存储器的相对位置来表示数据元素之间的逻辑关系。顺序表在内存中是用一组连续的存储单元一次存储线性表的各个元素，因为它是一种随机存取的存储结构，故通过确定起始存储位 a1,那么它里面的任意一个元素都可以随机存取。那么它的插入删除查找操作就可以依此进行。</td></tr>
<tr><td>功能(函数)说明</td><td colspan="2">

```
/*****************************************************
功能：初始化顺序表
输入参数：x
说明：利用循环，每输入一个数，将其存入顺序表，表长度加1
*****************************************************/
void InitList(SeqList &SL,int n)
{
    int x, i=0;
    SL.elem = new int[MAX_L];
```

</td></tr>
</table>

```
    while(i<n)
    {
        cin >> x;
        SL.elem[i++] = x;
        SL.length++;
    }
}
/*************************************************
功能：删除元素
输入参数：e
输出值：若找到e,输出其所在位置；否则，输出-1；
说明：若找到删除的元素e,后面的元素依次向前移动1，
表长减1；否则就返回-1。
*************************************************/
void DelList(SeqList &SL, int e)
{
    int i, flag = 0, j;
    for (i = 0; i < SL.length; i++)
    {
        flag = 0;
        if (SL.elem[i] == e)
        {
            j = i + 1;
            flag = 1;
            break;
        }
    }
    if (flag == 1)
    {
        for (i = j; i < SL.length; i++)
            SL.elem[i - 1] = SL.elem[i];
        SL.length--;
        cout << j << endl;
    }
    if (flag == 0)
        cout << -1 << endl;}
/*****************************************
功能：找到元素并将其删除
输入参数：e
输出值：若是找到，将其删除并输出新表；否则
输出-1；
说明：利用循环，一旦找到该元素的位置，结束循环，
标记位置，flag=1，进行删除操作；否则，输出-1
*****************************************/
```

```
void DelList(SeqList &SL, int e)
{
    int i, flag = 0, j;
    for (i = 0; i < SL.length; i++)
    {
        flag = 0;
        if (SL.elem[i] == e)
        {
            j = i + 1;
            flag = 1;
            break;
        }
    }
    if (flag == 1)
    {
        for (i = j; i < SL.length; i++)
            SL.elem[i - 1] = SL.elem[i];
        SL.length--;
        cout << j << endl;
    }
    if (flag == 0)
        cout << -1 << endl;
}
/*******************************************************
功能：任意位置插入元素
输入参数：位置i，元素e
输出值：若插入位置合适就输出插入后的顺序表；否则，输出-1
说明：若不是插在表尾，将第i个及以后的元素向后循环移动1；
若插在表尾，则不进行循环，直接进行赋值即可；
*******************************************************/
void  InsertList(SeqList &SL, int e, int i)
{
    if (i<1 || i>SL.length + 1)
        cout << -1 << endl;
    else
    {
        for (int j = SL.length - 1; j >= i - 1; j--)
            SL.elem[j + 1] = SL.elem[j];
        SL.elem[i - 1] = e;
        SL.length++;
    }
}
/*************************************************
功能：在有序表中插入e使其还是有序表
```

```
说明：在一个有序表中，将该元素分别于表中元素比较，直到
e小于等于表中元素，将其插入即可，表长加1；插入的情况分析同上
**************************************************/
void  InsertList(SeqList &SL, int e)
{
    int i = 0;
    while (SL.elem[i] > 0 && SL.elem[i] < e)i++;
    if (i < SL.length)
    {
        for (int j = SL.length - 1; j >= i - 1; j--)
            SL.elem[j + 1] = SL.elem[j];
        SL.elem[i] = e;
    }
    else
        SL.elem[i] = e;
    cout << i + 1 << endl;
    SL.length++;
}
/*****************************************
功能：排序
说明：利用冒泡法将顺序表排序，以有序输出
*****************************************/
void SortList(SeqList &SL)
{
    int temp;
    for (int i = 0; i < SL.length - 1; i++)
        for (int j = 0; j < SL.length - i - 1; j++)
            if ((SL.elem[j] > SL.elem[j + 1]) && SL.elem[j + 1] > 0)
            {
                temp = SL.elem[j];
                SL.elem[j] = SL.elem[j + 1];
                SL.elem[j + 1] = temp;
            }
}
/**************************************************
功能：将两个有序表合并为一个有序表
说明：比较La、Lb首元素的大小，小者存入Lc，利用循环，
若La[i]<L[j],L[i]插入Lc；反之，Lb[j]插入，j++;
直到某个顺序表结束，再将另一个顺序表剩余元素插入Lc即可
**************************************************/
void CmbList(SeqList &La, SeqList &Lb, SeqList &Lc)
{
    Lc.elem = new int[MAX_LC];
    int a = 0, b = 0;
```

```
    while (a < La.length&&b < Lb.length)
    {
        if (La.elem[a] < Lb.elem[b])
        {
            InsertList(Lc, La.elem[a]);
            a++;
        }
        else
        {
            InsertList(Lc, Lb.elem[b]);
            b++;
        }
    }
    while (a < La.length)
    {
        InsertList(Lc, La.elem[a]);
        a++;
    }
    while (b < Lb.length)
    {
        InsertList(Lc, Lb.elem[b]);
        b++;
    }
}
/*************************************************
功能：去重
说明：建立辅助数组only，利用两层循环将，外层循环
用于SL.elem顺序表的遍历，内层循环用于only数组的遍历，
从而得到only数组只有只出现一次的元素。
*************************************************/
void Rev_same(SeqList &SL)
{
    int *only;
    only = new int[MAX_L];
    int k = 0, t = 1, flag = 0, i, j;
    only[0] = SL.elem[0];
    for (i = 1; i < SL.length; i++)
    {
        flag = 1;
        for (j = 0; j < t; j++)
            if (only[j] == SL.elem[i])
            {
                flag = 0;
                break;
```

```
            }
        if (flag == 1)
            only[t++] = SL.elem[i];
    }
    for (int i = 0; i < t; i++)
        cout << only[i] << " ";
    delete[]only;
}
/*********************
功能：打印顺序表
说明：利用循环将其输出
**********************/
void PrintList(SeqList &SL)
{
    for (int i = 0; i < SL.length; i++)
        if (SL.elem[i] != 0 && SL.elem[i] > 0)
            cout << SL.elem[i] << " ";
    cout << endl;
}
/*功能：销毁顺序表*/
void ClrList(SeqList &SL)
{
    delete[]SL.elem;
}
```
 |
| **开发环境** | WIN10，visual studio 2017，C++语言程序设计 |
| **调试分析** | Problem-1：
4
10 20 40 30
1 25
1
40
10 20 40 30
25 10 20 40 30
10 20 40 30
3
Problem-2:
10 20 40 30 68 39 0
50
30
6
3
10 20 39 40 50 68
23 54 45 30 30 30 67 0
50
30
6
2
23 30 30 45 50 54 67
Problem-3:
1 3 5 7 30 43 0
2 4 8 9 23 45 60 90 0
1 2 3 4 5 7 8 9 23 30 43 45 60 90
Problem-4: |

	7 10 30 34 43 30 56 30 30 10 34 43 56 Problem-5: 8 1 3 2 2 5 4 3 4 1 3 2 5 4
心得体会	本次实验主要是实现顺序表的创建，插入，删除，查找，排序输出，销毁等操作。还有许多其他功能的实现，比如将有序表进行合并依然为有序表，顺序表的去重。这些实例都是以顺序表的基本操作为基础进行应用和整改，从而得到相应的结果。我觉得比较重要的就是删除、插入时对表头和表尾元素的考虑，对这种特殊情况做出特定的分析，看看究竟需不需要另作讨论，或者如何设计变量的取值才能让其成为一般情况，不需大费周折分类讨论，简化代码，这是需要认真思考和实践的地方。接下来就是如何使时间复杂度降低，有效提高算法运行效率，比如顺序表的合并，我们完全可以将两个有序表直接合并，然后利用排序算法处理就得到了新的有序表，但是这样的话就大大提高了算法的时间复杂度，所以可利用一次遍历逐个比较的方式进行排序，这样就极大地提高了效率。总之，对算法的设计要综合考虑，保证准确性和可读性以及运行效率。