《数据结构》上机报告

2018 年 11 月 21 日

姓名：刘思源 学号：1651390 班级：电子三班 得分：

<table>
<tr><td>实验题目</td><td colspan="2">静态链表</td></tr>
<tr><td>问题描述</td><td colspan="2">静态链表（或称为数组链表），用一个一维数组 S 存储链表的所有结点。每个结点包含 data 和 cur 两个域，data 存储数据,cur(称作游标，int 类型)代替指针，存储链表的下一个结点在数组中的相对位置。 数组前两个 s[0]存放链表头指针,s[1]存放闲置链表头指针；S[0].cur 初始为-1，表示空表。S[1].cur 初始为 2，表示第 1 个可分配的结点位置。S[MaxSize-1]=-1 表示无后继结点。初始 S[i].cur=i+1，表示所有未分配结点已形成一个备用链表，每当进行插入时，从备用链表中取得第 1 个结点作为待插入的新结点，删除时则将从链表中删除的结点链接到备用链表中。

静态链表实现线性表的操作与动态链表相似，执行插入和删除操作时不需移动元素，只需修改指针，因此具有链表存储结构的主要优点。 本题练习用静态链表实现线性表的基本操作，包括表的初始化，尾部追加一个元素、第 i 个元素位置前插入一个新的元素、删除第 i 个元素、查找某元素、判表空、判表满、表的遍历。 元素数据类型是字符串。</td></tr>
<tr><td rowspan="2">基本要求</td><td colspan="2">1. (p1) 静态链表实现线性表的操作与动态链表相似，执行插入和删除操作时不需移动元素，只需修改指针，因此具有链表存储结构的主要优点。 本题练习用静态链表实现线性表的基本操作，包括表的初始化，尾部追加一个元素、第 i 个元素位置前插入一个新的元素、删除第 i 个元素、查找某元素、判表空、判表满、表的遍历。 元素数据类型是字符串。</td></tr>
<tr><td>已完成基本内容（序号）：</td><td>1</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>
</table>

数据结构设计

```
typedef struct Node{
    string data;
    int cur;
}*SLinkList, StaticList;
```

功能(函数)说明

1、 静态链表初始化

首先将所有的元素都置为“N/A”。接着将第三个之后的元素全部置为 i+1，将最后一位的指针置为-1。将第二位的活动指针元素置为第一个空结点，如果没有就置为-1。

```
void InitSList(SLinkList&SList, int num, int n)
{
    int i = 0;
    for (i = 0; i<num; i++){
        SList[i].data = "N/A";
    }
    for (i = 2; i < num - 1; i++){
        SList[i].cur = i + 1;
    }
    SList[num - 1].cur = -1;
    SList[0].cur = -1;
    if (num > 2){
        SList[1].cur = 2;
    }
    else{
        SList[1].cur = -1;
    }
   }
```

2、 打印静态链表

每三个元素输出一行，打印静态链表。

```
void Print(SLinkList&SList, int a)
{
    int k = 0;
    for (int i = 0; i<a; i++){
        cout << i << " : " << SList[i].data << " : " << SList[i].cur << "        ";
        k++;
        if (k % 3 == 0 && k != 0) {
            cout << endl;
        }
```

```
    }
    if (k % 3 != 0){
        cout << endl;
    }
}
```

3、 更新目标节点

静态链表并不存在指针，所谓的指针是通过一个静态的数组元素来实现的。每次添加新的元素、删除、插入的时候都要对这个指针进行更新。将目标节点的下一个指针位置赋给活动指针，再将值赋给目标元素。

```
void Chage(SLinkList&SList, int num, string str)
{
    int t = SList[1].cur;
    SList[1].cur = SList[t].cur;
    SList[t].data = str;
    SList[t].cur = SList[num].cur;
    SList[num].cur = t;

}
```

4、 释放元素

首先更新活动指针的指针域，再将目标的数据与置为N/A。

```
void Free(SLinkList&SList, int n)
{
    SList[n].cur = SList[1].cur;
    SList[n].data = "N/A";
    SList[1].cur = n;
}
```

5、 插入操作

如果活动指针的指针域为-1，那么说明链表已满，输出提示条件。否则从链表的头结点开始遍历，一直找到目标位置的结点，进行更新操作。如果目标位置不合法，输出-1，并进行打印。

```
    void Insert(SLinkList&SList, string insert, int num, int &n)
{
    if (SList[1].cur == -1){
        cout << "FULL" << endl;
        return;
    }
    int curpos = 0, s;
    int pos = 0;
    while (curpos != -1 && pos < num - 1) {
        curpos = SList[curpos].cur;
        pos++;
    }

    if (curpos == -1 || pos > num - 1)  {
        cout << "-1" << endl;
```

```
        return;
    }
    Chage(SList, curpos, insert);
    Print_1(SList);
    n++;
}
```

6、 删除操作

如果当前头结点的指针域为-1，说明该链表为空，输出提示语句。如果删除的位置不合法，输出提示语句。接下来，从链表的头结点开始进行遍历，一直找到目标的位置，将前驱结点连接到后继节点，然后对该结点继续进行释放操作。

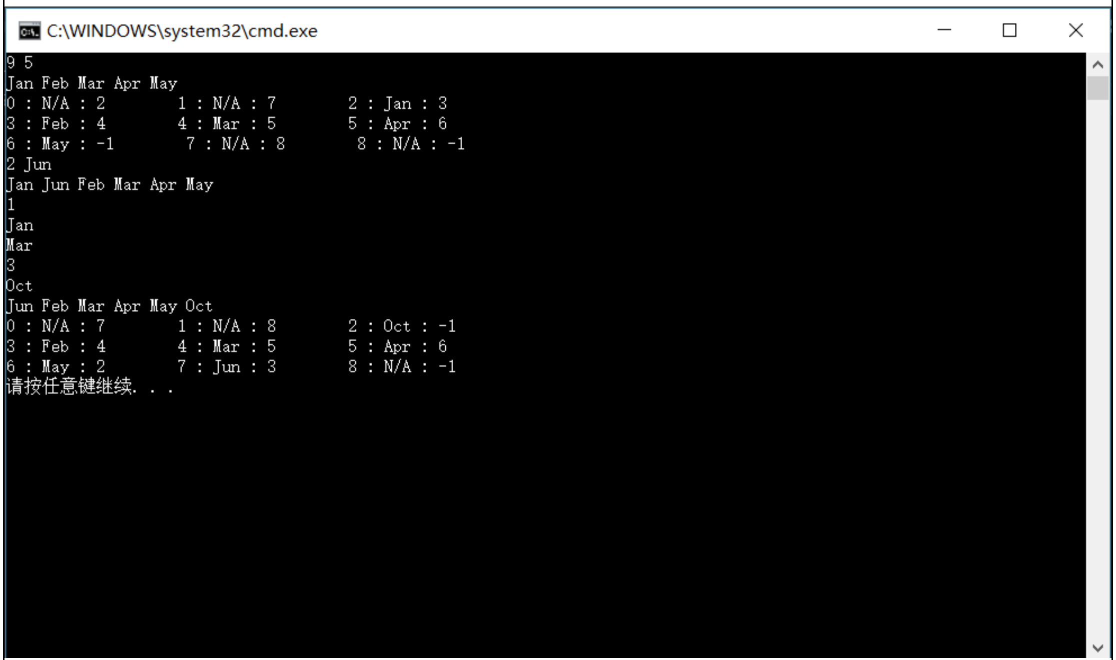
```
void Delete(SLinkList&SList, int a, int &n)
{
    if (SList[0].cur == -1){
        cout << "EMPTY" << endl;
        return;
    }
    if (a<1 || a>n){
        cout << "-1" << endl;
        return;
    }
    int curpos = 0, q = 1;
    while (q<a){
        curpos = SList[curpos].cur;
        q++;
    }
    int t = SList[curpos].cur;
    cout << SList[t].data << endl;
    SList[curpos].cur = SList[t].cur;
    Free(SList, t);
    n--;
}
```

7、 查找操作

从链表的头结点开始进行遍历，一直找到目标的位置。返回第一个位置。

```
int Locate(SLinkList&SList, string e, int(*compare)(string e1, string e2))
{
    int curpos = SList[0].cur;
    int pos = 1;
    while (curpos != -1 && (*compare)(e, SList[curpos].data) == 0) {
        curpos = SList[curpos].cur;
        pos++;
    }

    return curpos != -1 ? pos : -1;
```

	}
开发环境	Visual studio 2017
调试分析	
心得体会	1、 静态链表在插入和删除操作时不需要移动元素，只需要修改游标，从而改进了在顺序存储结构中插入和删除操作需要移动。在少量元素时比较方便。 2、 大量元素的缺点；但并没有解决连续分配存储带来的表长难以确定的问题；并且失去了顺序存储结构随机存取的特性。