第一次研讨课报告

一、 静态结构与动态结构的本质区别

1、 研讨

静态结构是程序段中已经定义好的，不是临时开辟。**静态数据结构的特点是由系统分配固定大小的存储空间，以后在程序运行的过程中，存储空间的位置和容量都不会再改变。**

动态结构是需要时分配，可在使用结束后销毁。根据不同的需要进行分配。动态数据结构不确定总的数据存储量，而是为现有的每一个数据元素定义一个确定的初始大小的空间，若干个数据元素分配若干个同样大小的空间；当问题的数据量发生变化时，数据的存储空间的大小也发生变化。如果数据量增加，就重新向系统申请新的空间；如果数据量减少，就将现有的多余的空间归还给系统。

2、 课后思考

所谓动态数据结构，是指可以随着程序的执行二动态的改变大小和形状的一类数据结构，如链表、树和图等。动态结构的数据，在编译时无法预先规定他们所需要分配的存储空间大小，只有在运行时进行动态存储分配。

静态数据结构，数据所需存储空间是一个固定的有限区域，可在程式说明中显式规定，在编译时静态进行存储分配。

凡是可以用指针动态实现的数据结构，都可以利用数组静态的模拟实现。有时也把这种利用数组静态模拟实现了的动态结构称为半静态数据结构。**当然，半动态结构中也包含了可变数组和变长记录等部**

分采用静态分配、部分采用鼎泰分配的数据结构。如静态链表，二叉树的静态二叉链表表示法，树和森林的双亲表示法，哈夫曼算法的静态实现等。

二、 对于单链表，使用头结点的优点与缺点

1、 研讨

优点：

a) 头结点是为了操作的统一与方便而设立的，放在第一个元素结点之前，其数据域一般无意义。**也可以将头结点的数据域存放链表的长度或者用作监视哨。**

b) 使用头结点后，对第一个元素结点前插入结点或是删除第一个结点，其操作与对其他结点的操作统一。

缺点：

c) 对于大量的链表而言，头结点可能占据了相当的存储空间。

d) 若使用头结点的数据域存储链表的长度，可能带来的问题是链表的数据类型并非是整型，可能要额外进行一次变量的定义。

2、 课后思考

单链表头结点总结:

a) 防止单链表是空的而设的.当链表为空的时候,带头结点的头指针就指向头结点.如果当链表为空的时候,单链表没有带头结点,那么它的头指针就为 NULL.

b) 是为了方便单链表的特殊操作,插入在表头或者删除第一个结点.

这样就保持了单链表操作的统一性!

c) 单链表加上头结点之后，无论单链表是否为空，头指针始终指向头结点，因此空表和非空表的处理也统一了，方便了单链表的操作，也减少了程序的复杂性和出现 bug 的机会。

d) 对单链表的多数操作应明确对哪个结点以及该结点的前驱。不带头结点的链表对首元结点、中间结点分别处理等；而带头结点的链表因为有头结点，首元结点、中间结点的操作相同 ,从而减少分支，使算法变得简单 ,流程清晰。**对单链表进行插入、删除操作时**，如果在首元结点之前插入或删除的是首元结点，不带头结点的单链表需改变头指针的值，在 C 算法的函数形参表中头指针一般使用指针的指针(在 C+ +中使用引用 &)；而带头结点的单链表不需改变头指针的值，函数参数表中头结点使用指针变量即可。

三、 学生成绩管理：按学号顺序输入，建立成绩表；将其按学号从大到小逆置。可以采用哪些数据结构，如何做，算法复杂度分析。

1、 研讨

a) 线性表的顺序存储：数组

i. 可以申请足够大小的数组进行数据的存储。然后将数组以学号从大到小排序。基本上处于$O(n\log n)$到$O(n^2)$之间。

ii. 可以申请一个较大的数组，如大小为 9999 的数组。每一个学

生的学号对应的就是数组元素的下标，进行成绩输入的时候，将对应的学生成绩直接输入进对应的下标元素中。最后从头开始遍历数组，对于空的数组直接跳过。复杂度为$O(n)$。

b) 线性表的链式存储：链表

使用链表同样可以完成这个操作。

使用单链表的头插法，每读入一个数据就对进行一次头插操作，与之后的每一个数据进行比较。复杂度为$O(n)$。

也可以在全部输入完成后在进行排序，后续会进行排序算法的总结。

c) 栈

使用栈可以较为轻松的解决这个问题。因为本身输入就是顺序输入的，最后是要逆序将其输出。我们可以直接将数据输入栈中，然后根据后进先出的特点直接将其 Pop 出即可完成操作。复杂度为$O(1)$。

2、 课后思考

本题实际上涉及到不同数据结构的排序算法，接下来我们对排序算法进行一定的总结：

i. 冒泡排序

冒泡排序是最简单的排序之一了，其大体思想就是通过与相邻元素的比较和交换来把小的数交换到最前面。较为常见，复杂度为$O(n^2)$。

```
class Solution(){
public:
    void bubbleSort(int arr[],int len){
        int i,j;
        int Temp=0;
        int compareRange=len-1;
        for(i=0;i<len;i++){
            for(j=1;j<compareRange;j++){
                if(arr[j-1]>arr[j]){
                    Temp=arr[j-1];
                    arr[j-1]=arr[j];
                    arr[j]=Temp;
                }
            }
            compareRange--;
        }
    }
};
```

ii. 选择排序

选择排序的思想其实和冒泡排序有点类似，都是在一次排序后把最小的元素放到最前面。但是过程不同，冒泡排序是通过相邻的比较和交换。而选择排序是通过对整体的选择。复杂度为$O(n^2)$。

```
    class Solution{
    public:
        void selectSort(int arr[],int len){
            int i,j,temp=0;
            for(i=0;i<len-1;i++){
                for(j=i;j<len-1;j++){
                    if(arr[j]>arr[j+1]){
```

```
                    temp=arr[j];
                    arr[j]=arr[j+1];
                    arr[j+1]=temp;
                }
            }
        }
    }
};
```

iii. 希尔排序

希尔排序是插入排序的一种高效率的实现，也叫缩小增量排序。简单的插入排序中，如果待排序列是正序时，时间复杂度是 O(n)，如果序列是基本有序的，使用直接插入排序效率就非常高。希尔排序就利用了这个特点。基本思想是：先将整个待排记录序列分割成为若干子序列分别进行直接插入排序，待整个序列中的记录基本有序时再对全体记录进行一次直接插入排序。希尔排序的复杂度在$O(n\log n)$到$O(n^2)$之间。

```
class Solution{
public:
    void shellSort(int arr[],int len){
        int j=0,tmp=0;
        for (int d = len/2; d >0 ; d/=2) {//d 是增量，也是排序时的分组数。
            for (int i = d; i <len; i++) {//0～d-1 是各分组的第一个元素，作为初始时插入排序的有序序列。
                j=i-d;      //得到 i 所在的分组中，其前一个元素(有序的)
                tmp=arr[i];
                while (j>=0&&arr[j]>tmp){
                    arr[j+d]=arr[j];
                    j-=d;
                }
                arr[j+d]=tmp;
            }
        }
    }
};
```

iv. 快速排序

冒泡排序是通过相邻元素的比较和交换把最小的冒泡到最顶端，而快速排序是比较和交换小数和大数，这样一来不仅把小数冒泡到上面同时也把大数沉到下面。快速排序是不稳定的，其时间平均时间复杂度是$O(n\log n)$。

```
Class Solution{
public:
    quickSort(int[] arr,int l,int r){
      if(l>=r)
                return;
     int i = l; int j = r; int key = a[l];//选择第一个数为 key
     while(i<j){
         while(i<j && a[j]>=key)//从右向左找第一个小于 key 的值
             j--;
         if(i<j){
             a[i] = a[j];
             i++;
         }
         while(i<j && a[i]<key)//从左向右找第一个大于 key 的值
             i++;
         if(i<j){
             a[j] = a[i];
             j--;
         }
     }
     //i == j
     a[i] = key;
     quickSort(a, l, i-1);//递归调用
     quickSort(a, i+1, r);//递归调用
     }
};
```

v. 插入排序

插入排序不是通过交换位置而是通过比较找到合适的位置插入元素来达到排序的目的的。简单插入排序的复杂度为$O(n^2)$。

```
Class Solution{
public:
    straightInsertSort(int[] arr,int len){
        for (int i = 2; i <=len ; i++) {
           arr[0]=arr[i];
           int j;
           for(j=i-1;arr[0]<arr[j];--j){
               arr[j+1]=arr[j];
           }
           arr[j+1]=arr[0];
       }
    }
};
```

四、 医院看病排队系统

本题的界限比较宽泛，并未给出很具体的要求，所以在题目的解答上多种多样。并且由于此题为实际问题，有一些实际情况的未成文的约束。

1、 约束条件

a) 同一个队列中不能插队

b) 一旦进入一条队列，不能更改

2、 多窗口管理系统

设立 M 个急诊窗口，N 个普通窗口。对于每个病人设立一个病情参数α，α与两个参数有关：初始的病情β和等待时间γ。我们可以对不同的病情设置不同的参数值。当一个患者进入医院后，我们会对该患者的病情参数α做出判断，如果达到急诊要求，就进入 M 个窗口的排队序列中；否则进入 N 个窗口的排队序列中。每个窗口使用队列的数据结构，新加入的病人将自动进入到排队等待时间较少的队伍中。

3、 另有同学提出可使用医院的动态分配进行规划。虽然并未实质性的解决排队问题，但对医院的管理模式提供了一定的思路。主要可以解决科室与诊室的开放与数量的管理。因为每一个诊室的看病时间是不可缩短的，我们尽可能的要减少每个人的等待时间。当一位患者进入医院后即开始计时，当患者结束看病后终止。在其中耗费的所有时间记为看病时间τ。对于不同的诊室不同的科室，有着不同数量以及不同值的τ。因为医院的场地有限，可根据这些大量收集得到的τ值进行科室与诊室的安排。

五、 建议

1、 本次题目中部分题目并未有着严格的要求，不知道是否有意为之。建议之后的题目可以稍加约束，毕竟研讨课的时间有限，题目太宽泛有可能同学之间有理解误差，组与组之间不能很好的沟通。

2、 本次题目有涉及到一些基本的问题，如静态结构动态结构等。这些问题应当掌握，但可能大家的想法都很类似，也很难有创新的点产生。建议可提出一些答案更多元的问题。