梅河口市第五中学信息学竞赛培训材料

第 4 次笔记

对应章节：Chapter 3

说明：我们根据需要发布参考笔记，这部分内容作为对教材的说明解释或者额外补充，教材正确合理的部分不作赘述。对于竞赛，语言深度不要求很深，那些不必掌握的知识，笔记中会有注明。

---

1. 综述

第三章的主题为“控制流”，内容是 C 语言程序设计中基本的控制结构。C 语言是面向过程的程序设计，换句话说是“结构化”的，结构化的含义是，采用自顶向下逐步求精的设计方法和单入口、单出口的控制结构。1965 年由 E.W.Dijkstra 提出，C.Bohm 和 G.Jacopini 证明的一个定理是，只用三种基本控制结构，就能够实现任何单入口单出口的程序，它们是：顺序结构（最基本的结构），分支(选择)结构（if、if-else、switch），和循环结构（while、do-while、for）。前三章掌握后，你已经可以写大量有用的程序了，我们关于语言的介绍也完成了近一半，你会发现从现在开始我们逐渐接触到更多算法部分。

本讲条目 5 对应的测试代码已上传至群文件。

2. 关于条件判断

在语句中的判断，规定 0 为“假”，非 0 为“真”（而不是 1 为真），如果 ptr 是一个指针（马上会介绍，稍提前一点点，ptr 假设为 int *类型），判断 ptr 是不是有效的指针可以写：

```
if (ptr)
```

相当于：

```
if (ptr != NULL)
```

把 ptr 换成 a（int 型变量），那么 if (a)的作用是如果 a 为 0 跳过，否则执行。同理，a 替换为表达式也是一样的，前几讲我们说过表达式是有值的，比如 1 > 2 这个表达式的值为 0。再同理，表达式也可以是调用的一个函数（必须有返回值），比如函数 isspace()和 isdigit()，函数返回 0 则跳过，否则则执行。在 while、for 等中出现的判断也是如此。有一类循环（永真循环）可能会用到但比较危险，它长成这个样子：

```
while (1)
    ...
for (;;)
    ...
```

这时循环中必须有（保证能执行到的）break 语句或 return 语句，否则这将是一个死循环（表面上看程序会变成‘未响应’的状态）。

3. 易错之处

语句写错是最愚蠢的错误，我们说过，竞赛考的是算法，如果语法出错的话那么太亏了。有些错误编译时会报出来，比如 do-while 语句中 while(…)后漏写分号，break 和 continue 语句后漏写分号。有些错误编译时不会报（因为符合语法体系，尽管不被期待），比如 for(…);这个分号，再比如 switch 语句中某一个 case 漏写 break;语句，调试期也需要很长时间才能发现。逗号表达式是合法的，但依赖求值次序的表达式间尽量不要使用（在 for 循环中已经规定了从左到右的顺序可以使用），goto 语句也是合法的，在跳出多重循环中尤为有用，但我们的题目基本不会涉及，goto 会破坏程序结构，请尽量不要使用。更多易错之处，比如我们提了 N 次的判断等号为==赋值为=这种，请时刻警惕。

4. 二分法之二分查找

我们不断地追求程序运行的高效率，提升效率的策略之一就是让计算机“跳过”无用的部分，教材 3.3

节（47 页）中下部写了一个很有用的二分查找范例，它的整体思想是，对于有序数组，每次都判断要查找的值与 low 到 high 区间的中间值（程序中 int 与 int 的除法返回整数，在前面提过）的关系，如果相等，表明找到可以返回。如果大，则在前半部分找（忽略了更大的一半所以效率更高），小，则在后半部分找。当 low 超过 high（也就是找到了两端）仍然没发现时即为不存在。二分思想可以显著降低复杂度，我们后面还会遇到。

5. 排序算法

唉…是时候讲解一些排序算法了……

排序，初学者最难理解的部分之一，算法导论的第一个例子，说实话我是真不想写这部分——这也从侧面反映出，我和我的同学们在一学期的学习之后，对排序算法的掌握仍然有模糊之处。

也因此，我强烈建议你在学习该部分时准备一张纸或者一个程序，举一些例子输入给待学习的算法，从而真正掌握它们排序的过程和优劣。

a) 插入排序

```
void InsertionSort(int val[], int len)
{
    int cur;
    for (cur = 1; cur < len; cur++)
    {
        int key = val[cur], i;
        for (i = cur - 1; i >= 0 && val[i] > key; i--)
            val[i + 1] = val[i];
        val[i + 1] = key;
    }
}
```

这只是一个示例版本，请**掌握和理解算法而不要背诵代码**，你可以用自己偏爱的循环方式写出自己风格的代码。但再怎么写，原理都是一样的。插入排序的思想是，如果我手里有几张已经排好序的牌和一张新牌，那么我将新牌插入到有序牌中保持有序牌的有序，并重复这个过程。换句话说，函数中最外层的 for 循环每次执行时都能保证 cur 之前是一个有序序列（对于第一次执行，cur 之前只有一个数，一个数当然也是有序的），那么经过一次遍历之后整个数组就都变成有序的了。如果我们用数组{4, 5, 3, 1, 8, 2}进行测试，那么每次 for 循环执行之后数组是这个样子：

```
4 5 3 1 8 2
3 4 5 1 8 2
1 3 4 5 8 2
1 3 4 5 8 2
1 2 3 4 5 8
```

b) 冒泡排序

插入排序总体上保证了“前面的序列有序”，冒泡很相似，它希望经过两两比较，每次找出最大或最小的键值并通过两两交换将其移到一端（就像水中气泡一样冒上去），参考代码如下：

```
void BubbleSort(int val[], int len)
{
    int i, j;
    for (i = len - 1; i > 0; i--)
        for (j = 0; j < i; j++)
            if (val[j] > val[j + 1])
            {
                int temp = val[j];
                val[j] = val[j + 1];
                val[j + 1] = temp;
            }
}
```

还是一样，冒泡排序写法不局限于上面一种，掌握原理写出自己风格的即可，我们也不局限于交换相邻两数，可以遍历的时候与有序端端点交换（反正都不影响交换次数）。请先冷静思考：**这段冒泡排序是把（大还是小）数往（前还是后）冒？两层循环的控制意义分别是什么？**

我们仍然用数组{4, 5, 3, 1, 8, 2}进行测试，那么每次交换两数后数组是这个样子：

```
4 3 5 1 8 2
4 3 1 5 8 2
4 3 1 5 2 8
3 4 1 5 2 8
3 1 4 5 2 8
3 1 4 2 5 8
1 3 4 2 5 8
1 3 2 4 5 8
1 2 3 4 5 8
```

我知道此刻你可能晕晕的，上面列出的是所有**交换**操作，那么每次外层循环结束之后数组中元素是什么样子呢？**请先独立思考，在上面的框中圈出你认为的答案。**

事实上，每次外层循环结束后数组元素的值为：

```
4 3 1 5 2 8
3 1 4 2 5 8
1 3 2 4 5 8
1 2 3 4 5 8
1 2 3 4 5 8
```

现在我们可以回答刚开始的问题，对于上面这段代码，它每次将大的数值往后冒，对于每次交换操作，保证相邻两数下标小的比下标大的（左边的比右边的）小，这将使大数不断向后移动，因此外层循环保证每次执行之后 i 往后的序列是有序的且是最大的(看上面一共 5 行的表即可发现)。大数向后移动的同时，相对小的数被交换而前移，但相对小和前面的数的关系不确定，因此 i 前面的序列仍然是需要排序的，内层循环的作用就是这一点。

※根据上面的分析独立思考：用冒泡排序的方式给元素个数为 n 的数组排序，外层循环执行多少次？对于第 i 次外层循环，内层循环执行多少次？

这个问题很重要，请一定完成↑↑↑

冒泡排序虽然不是最快的，却是最好写的，请熟练掌握。

c) Shell 排序

铺垫了这么多，其实就是想解释教材中出现的 Shell 排序（代码见 3.5 节第 51 页中部）。仍然用数组{4, 5, 3, 1, 8, 2}进行测试，那么每次交换两数后数组是这个样子：

```
1 5 3 4 8 2
1 5 2 4 8 3
1 2 5 4 8 3
1 2 4 5 8 3
1 2 4 5 3 8
1 2 4 3 5 8
1 2 3 4 5 8
```

很神奇的事情是 Shell 排序交换了 7 次而冒泡排序交换了 9 次，可见 Shell 排序效率是更高一筹的。请观察发生交换的两个数的位置，正因为 gap 是二分再二分逐渐减至 1 的，每次二分都保证了相距 gap 的元素之间左小于右（换言之，大数飞过了 gap 的距离被移到后面而不是每次相邻两数间移动），当 gap 减至 1 时进行了和冒泡相似的操作，由于做了前期准备，所以交换次数要少一些。现在看书上给出的解释应该可以理解了。

d) 未完待续，多种精彩的排序算法等着你~

————————————————（所有内容到此结束）————————————————