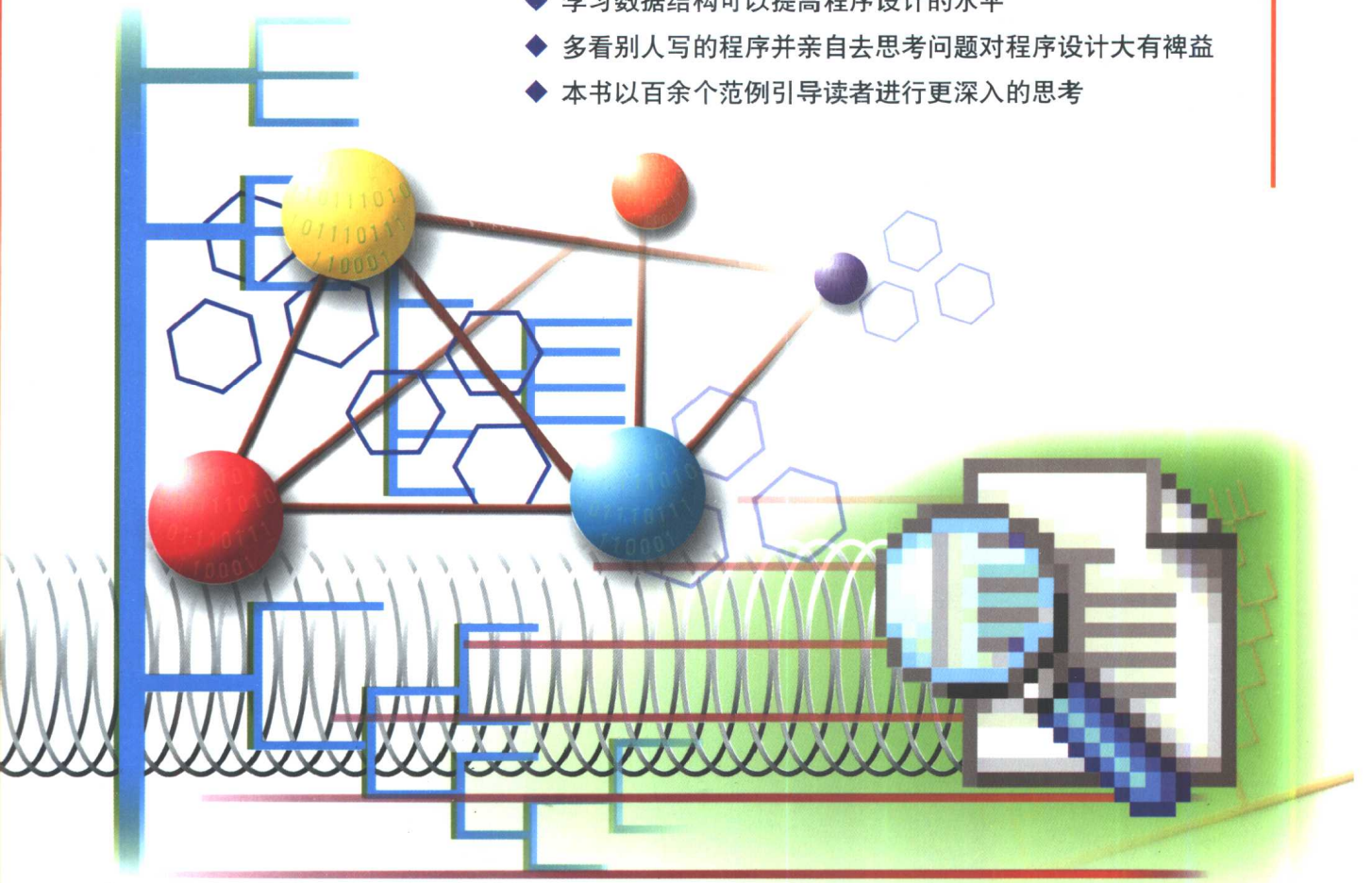
数据结构（C语言版）

- 数据结构是前人在思索问题时想出的解决方法
- 学习数据结构可以提高程序设计的水平
- 多看别人写的程序并亲自去思考问题对程序设计大有裨益
- 本书以百余个范例引导读者进行更深入的思考



黄国瑜　叶乃菁　编著

清华大学出版社
http://www.tup.tsinghua.edu.cn

数 据 结 构

(C 语言版)

黄国瑜　叶乃菁　编著

清 华 大 学 出 版 社

(京)新登字 158 号

内 容 简 介

数据结构包含以下两方面的内容：一是用合适的运算法则来规划程序流程，二是采用简洁的数据结构来表示程序中的数据和变量。

本书以 C 语言为程序设计语言，采用条列式的叙述方式，引导读者循序渐进地掌握堆栈结构、队列结构、树状结构、字符串结构，以及递归设计、排序设计和查找设计等程序设计。全书文字浅显易懂，程序示例简洁明了，是程序设计人员的上乘参考书。

书　　名：数据结构(C 语言版)
作　　者：黄国瑜　叶乃菁
责任编辑：张彦青
出 版 者：清华大学出版社(北京清华大学学研大厦，邮编 100084)
http://www.tup.tsinghua.edu.cn
印 刷 者：北京市清华园胶印厂
发 行 者：新华书店总店北京发行所
开　　本：787×1092 1/16　**印张**：28.25　**字数**：790 千字
版　　次：2001 年 8 月第 1 版　2001 年 11 月第 2 次印刷
书　　号：ISBN 7-302-04509-7/TP・2665
印　　数：5001～10000
定　　价：32.00 元

许多人都会设计程序，但并非每位程序设计者都能写出好的程序。怎样才能编写出好的程序呢？我们强调两方面，一方面是要找出合适的算法来设计程序的流程，另一方面则是采用简洁适用的数据结构来表示程序中的数据和变量。而“数据结构”含括了这两方面的概念，故“数据结构”为程序设计者最重要的基本要求。

“数据结构”这门学科是前人在程序设计方面积累的程序设计经验，“学会基础程序设计，你只能解决程序设计中三成的问题；学会数据结构，你却能解决程序设计中八成的问题。”因此，“数据结构”是信息科学中被认为最重要的学科之一。但是“数据结构”并不容易学习，所以常让初学者望而却步。

为了减少读者在学习“数据结构”上的恐惧，本书在每章节中都附有详细的说明和程序范例，藉此带领读者一步一步地学习“数据结构”，并希望读者在学习本书的过程中能够获得最大的收益。

在本书中我们采用条列步骤的方式，引导读者循序渐进地学习各种不同的数据结构，还使用图形来表示每一步骤的处理操作，再加上浅显易懂的文字说明及程序范例，让读者可以轻轻松松地了解数据结构的概念。

虽然本书在编写过程中力求完美，仍难免有错漏之处，希望各位不吝指正。

WANT 工作室　黄国瑜
Cherish 工作室　叶乃菁

目　录

第1章 数据结构的基本概念

- 何谓数据结构
- 算法
- 程序结构化与设计风格
- 程序分析的方法
- 时间复杂度分析
- 渐近式表示法
- 递归式的复杂度计算

1.1 何谓数据结构

程序设计正如写篇英语作文一样，每个人都懂得英语的语法和基本句型，可是同样的作文题目，每个人写出的作文，却风格各异。同样的道理，程序设计也是一样，程序员都懂得程序的语法与语意，可是相同功能的程序，由不同程序员写出来的程序解法也会不一样。有人写出来的程序，程序效率很高，有人却用最复杂的方法来解决一个简单的问题。

当然程序设计的水平，绝非看了几本程序设计的书，就可以功力大增的。“师父领进门，修行看个人”，同样看完3本程序设计书的两个人，程序设计水平高的，一定是愿意亲自思索问题、多练习的人。记得刚学程序设计时，常听人说程序设计的水平要想提高，除了练好基本功夫外，最重要的还是多看别人写的程序，多亲自尝试去思考问题。从看别人写的程序中，我们可以发现出更多更有效率的解决方法，从亲自思考问题的过程中，我们更可以了解问题的解决方法常常不只一个，如何运用以前解决问题的经验，来解决更复杂更深入的问题，才是最有效的。

而“数据结构”，这门信息科学上的学问，正是前人在思索问题的过程中，所想出的解决方法。一般而言，在学习程序设计一段时间后(练好程序设计的基础以后)，学习“数据结构”便可以大大提高程序设计上水平。如果你只学会了程序设计的语法和语意，你只能解决程序设计上三分之一的问题，而且运用的方法，并不是最有效率的。但如果你学会了数据结构的概念，你却能在程序设计上，运用最有效的方法来解决八成以上的问题。

在程序语言中，“数据类型”是指程序语言中变量所能表示并存储的数据种类，“数据实体”则是指在一种数据类型中的所有可能元素的集合。而“数据结构”，大致上说来，就是数据实体中元素之间的关系，包括数据的表示法和运算。例如：链表(以后章节中会有详细的介绍)，是指一个将具有数据内容和指向下一笔数据指针的节点串连而成的数据结构。堆栈是指各元素具有先进后出(First In Last Out)特性的数据结构，当然还有很多种数据结构与问题的解法。本书会在后面的章节中，介绍几种常见的数据结构和问题的解决方法，帮助读者了解数据结构。

希望读者除了详读本书以外，还能试着用自己的方法，来解决本书所附的范例和习题，相信这会使程序设计水平有很大的提高。

1.2 算法与伪码

在开始这节以前，我们先定义出什么是“算法(Algorithms)”？算法是指为了完成某项特定工作所设计出的一连串用来说明工作是如何被完成的步骤。所有的算法都必须满足下列几个条件：

- 输入：不具有输入数据或具有多个输入数据。
- 输出：具有一个以上的结果输出。
- 定义明确(Definiteness)：每一个步骤的语句必须很明确。
- 有限的步骤(Finiteness)：按照算法所描述的步骤执行，在有限的步骤内，一定会结束。
- 有效率的步骤(Effectiveness)：算法中的每一个步骤必须是基本的指令(即使是用纸和笔也可以完成计算)。

简单的说，算法就是一个具有次序、步骤清楚，最后一定会有执行结束的可执行步骤。程序与算法不同的地方在于上述的第 4 点，算法必须是一个可执行完的步骤，而程序却允许存在死循环(一个具有死循环的程序，我们也称为程序)。

用来写算法的方式有好几种，一般而言，我们可以分为下列4种方式：

- 条列式的步骤：

以条列式的步骤来描述解决问题的方法。

例如：运用顺序查找来查找数据中某一个特定值。

步骤 1：输入数据和欲查找值。

步骤 2：查找数据中第一项。

步骤 3：如果数据全都查找过但未能查找到欲查找值，表示没有查找到数据。

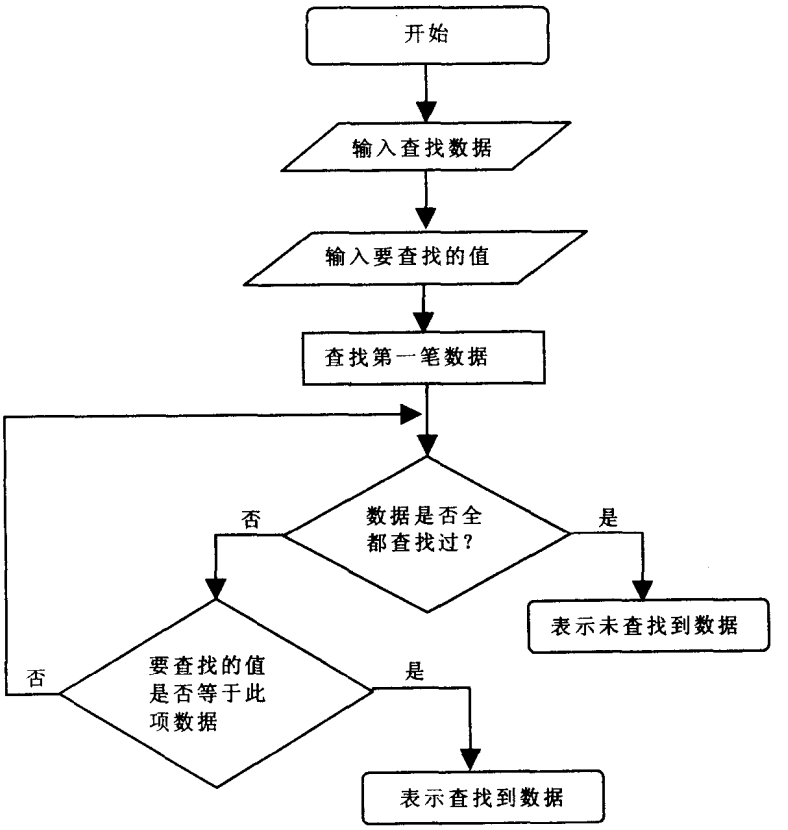
步骤 4：如果欲查找值等于此项数据，表示查找到数据。

步骤 5：如果欲查找值不等于此项数据，查找下一项数据，回到第 3 步继续执行。

- 流程图(Flow Chart)：

以图形符号来描述解决问题的方法。仅适用于小问题，不过现在已很少用。

例如：运用顺序查找来查找数据中某一个特定值。



- 伪码(Pseudo Code)：

以夹杂程序语法和自然语言(如：中文、英文)的形式来描述解决问题的方法。

例如：运用顺序查找法来查找数据中某一个特定值。

```
Procedure  Sequential_Search(Data,KeyValue)
设 I 为 1;
while ( )
{
    if ( 欲查找值等于 Data[I] )
printf("查找到数据。");
    else if ( I > 数据个数 )                    // 数据全查找完
printf("未能查找到数据。");
    I++;
```

```
}
```

- 程序语句：

直接以程序语法来描述解决问题的方法。

例如：运用顺序查找法来查找数据中某一个特定值。

```
/* ============== Program Description ===============*/
/* 程序名称： s_search.c                                */
/* 程序目的： 设计一个顺序查找的程序。                  */
/* Written By Kuo-Yu Huang. (WANT Studio.)             */
/* ================================================== */
int Data[20] =  /* 输入数据数组 */
    {       1,   7,    9,   12,   15,
     16,  20,   32,   35,   67,
     78,  80,   83,   89,   90,
              92,  97,  108,  120,  177};
int Counter = 1;      /* 查找次数计数变量 */

/*-----------------------------------------------------*/
/*顺序查找                                             */
/*-----------------------------------------------------*/
int Seq_Search(int Key)
{
    int    i;                    /* 数据索引计数变量 */

    for ( i=0 ; i<20 ; i++ )
    {
     printf("[%d]",Data[i]);  /* 输出数据 */
     if ( Key == Data[i] )    /* 查找到数据时 */
          return 1;
     Counter++;              /* 计数器递增 */
    }
    return 0;
}

/*-----------------------------------------------------*/
/*主程序                                               */
/*-----------------------------------------------------*/
void main ()
{
    int    KeyValue;                  /* 欲查找数据变量 */

    printf("Please enter your key value : ");
    scanf("%d",&KeyValue);          /* 输入欲查找值 */

    if ( Seq_Search(KeyValue) )      /* 调用顺序查找子程序 */
                                    /* 输出查找次数 */
     printf("\nSearch Time = %d\n",Counter);
    else
     printf("No Found!!\n");         /* 输出没有找到数据 */
}
```

1.3 程序结构化与设计风格

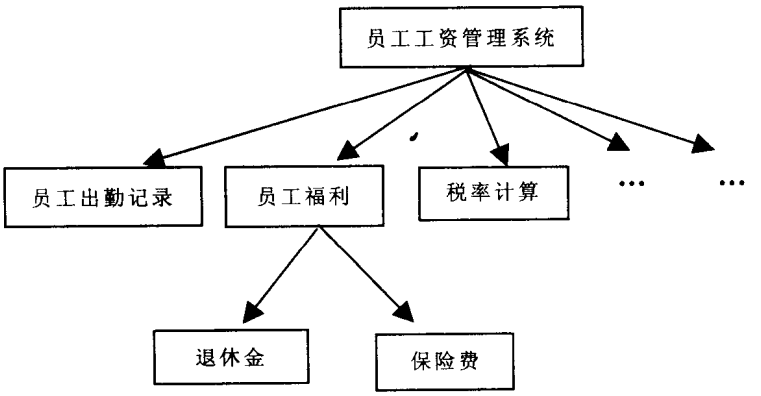
随着程序所要解决的问题愈来愈多时，程序的代码量也会愈来愈大，本来我们只是写写一、两百行的程序，可是当所要解决的问题很大时，程序可能就需要上万行才能解决该问题。一个小程序，在程序写完几个月后，我们再重新阅读时，很轻易的就能再了解当初所写的程序，但是当程序量很大时，如果没有一

个特定的程序编写方式，以后重新阅读时，一定困难重重。这一节我们所要说明的正是程序结构化与设计风格，希望藉此帮助读者建立程序的设计风格。

其实程序员所写的程序并不是一次就能把问题完整的解决，程序员花在程序维护和除错上的时间常常多于程序编写的时间。一个程序如果没有一定的风格，不仅别人看不懂，最后可能连自己也会看不懂，最糟的情况可能还得重写一次，因为重写的时间可能要比看懂原程序的时间短。的确，一份设计风格明确、简洁易懂的程序，可节省程序员在维护、除错上所花的时间，并有助于程序以后的更新与发展。

首先，我们来谈程序的结构化。程序结构化最重要的概念是“模块化(Modularity)”，所谓模块化就是将原来的大问题区分成几个小问题，再从解决小问题的过程中，组合成大问题的解法。对于解决小问题的小程序或许一个程序员就能胜任，但是一个大问题，常常是一群程序员共同努力的成果。所以模块化的概念，可将每个小模块分工给各小组的程序员来开发，最后再组合成一个完整的软件。

例如：一套员工工资管理系统，可能分为几个小模块，如下图所示，每一个方块代表一个模块。



区分为模块以后程序员只要分头去进行各模块的程序编写工作，最后即可完成一套完整的员工工资管理系统。

在此介绍读者两种程序设计发展的方式：

- 由上而下设计(Top-Down Design)：
  由上而下设计法的第一步通常是产生较简短的解法，再一步一步地修正前一个步骤所产生的解法，将前一个大模块区分成更小的模块，解决小模块以组合出大模块的解法。
- 由下而上设计(Bottom-Up Design)：
  由下而上设计法则是先列举出一个大模块中，各个个别的小模块，分别建立出各个模块的解决方案，再应用于大模块中。这个概念随着对象导向程序设计而逐渐受欢迎，因为在对象导向程序设计中，常常是建立一个个独立的对象，再通过小对象来完成大问题。

除了程序的结构化以外，程序的编写风格也是重点，笔者在此提供几个程序编写上的方法，以帮助读者建立起良好的编写风格。

1. 注释

一份良好的程序，除了程序本身外，最重要是要有一份完整的程序说明文件。一份没有注释的程序，宛如是一部天书，常常会让负责维护的程序员，搞不懂原设计者的设计目的。但一份注释完整的程序，除了自己阅读和除错上的方便外，更容易让人了解你的程序，并赞叹你在程序设计上的巧思与创意。

我们再举以前所提的顺序查找法为例：

```
01      /* ================ Program Description ================*/
02      /* 程序名称:  s_search.c                                  */
03      /* 程序目的:  设计一个顺序查找的程序。                    */
04      /* Written By Kuo-Yu Huang. (WANT Studio.)                */
05      /* ======================================================= */
```

```
06      int Data[20] =  /* 输入数据数组 */
07      {    1,   7,    9,   12,   15,
08      16,  20,   32,   35,   67,
09          78,  80,   83,   89,   90,
10                92,  97,  108,  120,  177};
11      int Counter = 1;      /* 查找次数计数变量 */
12
13      /* ------------------------------------------*/
14      /* 顺序查找                                  */
15      /* ------------------------------------------*/
16      int Seq_Search(int Key)
17      {
18          int    i;                        /* 数据索引计数变量 */
19
20          for ( i=0 ; i<20 ; i++ )
21          {
22           printf("[%d]",Data[i]);    /* 输出数据 */
23           if ( Key == Data[i] )      /* 查找到数据时 */
24                return 1;
25           Counter++;                 /* 计数器递增 */
26          }
27          return 0;
28      }
29
30      /* ------------------------------------------*/
31      /* 主程序                                    */
32      /* ------------------------------------------*/
33      void main ()
34      {
35       intKeyValue;                      /* 欲查找数据变量 */
36
37          printf("Please enter your key value : ");
38          scanf("%d",&KeyValue);          /* 输入欲查找值 */
39
40          if ( Seq_Search(KeyValue) )      /* 调用顺序查找子程序 */
41                                          /* 打印查找次数 */
42           printf("\nSearch Time = %d\n",Counter);
43          else
44           printf("No Found!!\n");         /* 打印没有找到数据 */
45      }
```

从此程序中，我们可以发现在程序的开头前 5 行：

```
/* ================ Program Description ================ */
/* 程序名称:  s_search.c                                  */
/* 程序目的:  设计一个顺序查找的程序。                    */
/* Written By Kuo-Yu Huang. (WANT Studio.)                */
/* ======================================================*/
```

我们在注释中写上程序名称、程序目的及作者。程序名称注明了这个程序可以在哪一个文件中找到，往后在查阅或修改时，就可以马上找到该文件进行修改。当然在修改时，如果能在程序开头再注明修改的时间、内容及修改者就更好了。程序目的则注明了这个程序的功用，帮助下一次运行时，无需重新看完程序，即可了解其功用。而程序作者部分，则是帮助以后想看懂这个程序或想修改的人，有问题时可以求助于原作者。

第 13~15 行是子程序的注释格式。

```
/* ------------------------------------------------- */
/* 顺序查找                                          */
/* --------------------------------------------------*/
```

我们在注释中，只需注明这个子程序功用即可，因为程序的主要功用已在程序开头的批注中说明过了。

第 6、11、18、35 行则是程序中变量的注释格式：

```
int    Counter = 1;     /* 查找次数计数变量 */
```

如果可以的话，最好是每一个变量用一行来声明，有人习惯在一行中声明多个变量，如下列这种方式：

```
int    Index,Counter,key,I,j;
```

除非这几个变量是同一功用的变量(如循环计数变量)，否则的话，这种变量声明方式，不仅在注释上不易，在阅读上更显得杂乱。

其余的程序批注，可选择在重要的程序语句后面加上一个注释内容。

2. 变量命名

变量是程序中用于记录输入值或输出结果的一个内存位置，所以变量所象征的意义正如同该变量在程序中所代表的意义。如果我们想要写一个计算学生成绩的程序，程序中需要用户输入学生学号、语文成绩、英语成绩、数学成绩，最后再计算出 3 科的平均成绩。此时如果程序中的变量声明为：

```
int    X;
int    A,B,C;
int    D;
```

我们没有人会看的懂几个变量所代表的意义，就算在程序以初就注明 X 是代表学生学号、A 代表语文成绩、B 代表英语成绩、C 代表数学成绩，D 代表 3 科平均成绩，在程序中，也会很容易忘了什么变量代表什么意义。如果把这 4 个变量声明为：

```
int    StudentNum;          /*   学生学号     */
int    Chinese;             /*   语文成绩     */
int    English;             /*   英语成绩     */
int    Math;                /*   数学成绩     */
int    Average;             /*   3 科平均     */
```

这样是不是比较清楚易懂呢？因为变量的声明通常会在某一个特定的区域(如：程序开头)，如果在变量以后再加上一些批注说明，这样在编写或修改程序以际，变量声明的区域就像是一个小字典，提供给我们所有在此程序中的输入和输出信息。

3. 程序缩排

在程序中经常会用到一些条件式语句或循环结构，在这个语句当中，包含了 C 语言中区块(Block)，也就是一群单一语句的集合，通常是以“{”及“}”来划分。“{”表示区块的开始，“}”表示区块的结束，在区块划分上有两种设计的风格。

一种是把开头的“{”与语句放在同一行，如：

```
for ( i=0 ; i<20 ; i++ )   {
      printf("[%d]",Data[i]);          /* 输出数据 */
      if ( Key == Data[i] )           /* 查找到数据时 */
                return 1;
            Counter++;                /* 计数器递增 */
      }
```

另一种是把开头的“{”不与语句放在同一行，如：

```
for ( i=0 ; i<20 ; i++ )
{
      printf("[%d]",Data[i]);          /* 输出数据 */
      if ( Key == Data[i] )           /* 查找到数据时 */
                return 1;
            Counter++;                 /* 计数器递增 */
}
```

到底哪一种比较好？并没有绝对的答案。不过笔者较偏好于后者，因为其结构清晰，程序也一目了然。