第1章

数据结构与算法

开场白

质软滑腻的石墨和坚硬无比的金刚石都是由碳原子组成的。由于它们的结构不一样，所以物理性质差异很大，最终导致用途截然不同。人类是由碳元素为中心组成的碳水化合物。如果另一星球上的智能生物拥有和人类相似的结构，唯一不同的只是所有碳原子替换成了半导体元素硅，相信这种外星人会像石头一样坚不可摧，并能穿山下地，或像变形金刚一样威力无比……虽然人类很难改变自身的结构，却可通过不断改变社会和物质的结构来获得进步。宋朝毕昇发明的活字印刷术，利用活字结构改变了雕版的固定结构，推动了整个人类文明的进步；产业结构调整，往往可以带动新一轮的经济发展和社会进步……

所以，不同的结构具有不同的性质，致使物体具有不同的用途和功效。然而，相同的结构，如果处理的方法不一样，又会产生什么样的差别呢？

例如，要移走一座山，愚公当年靠的是“子子孙孙无穷溃也”的几十乃至上百年的努力，而现代人采用炸药和推土机，几个月就可以完成。再比如，求 1～1000 之和，一般小学生采用呆板的累加方法，可能需要算上半天时间；若采用高斯算法来计算，只需半分钟。因此，方法决定了效率。

综上所述，用计算机解决现实问题，需要编写程序，而编写程序时需要考虑两个问题：一是采用什么结构存放数据（即数据结构）；二是采用什么方法和步骤来处理数据（即算法），以便尽快得到正确结果。选择不同的结构和算法，其效率是完全不同的。

事实上，可再用金庸的武侠小说来打个比方。结构就好比是一个练武之人的悟性和体格，算法就好比是其所学武功的门派和师傅传授的方法。郭靖的先天悟性和体格不算优秀，如果只是江南七怪调教，估计难成大气，但幸运的是得到了洪七公等名师的上乘方法调教，通过日积月累的练习，终成大侠；张无忌既是武学奇才（即先天结构很不错），又有高人前辈指点（即算法也很好），所以能迅速成长为少年英雄。

图灵奖获得者、计算机科学家沃思（N. Wirth）说过：“程序 = 数据结构 + 算法”。这里，数据结构指数据与数据之间的逻辑关系，算法指解决特定问题的步骤和方法。

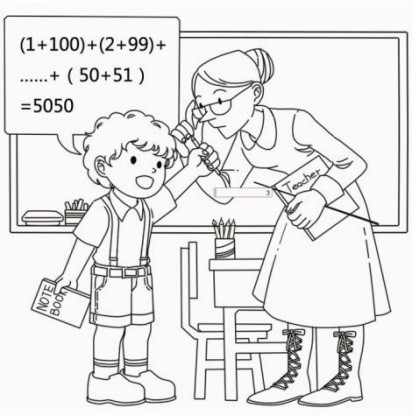
本书通过 9 个有趣案例，帮助大家用典型的结构和算法解决实际问题。

1.1 案例提出——高斯的巧妙解题

【案例描述】

描述一：

1787 年，在德国一所乡村小学的三年级课堂里，数学老师为了惩罚吵闹的学生，出了一道计算题：1+2+3+4+5+…+98+99+100。谁完成了就可以放学回家。



把 100 个数一个一个地加起来，这件事让三年级的小学生来做，是一种考验。不料，老师刚说完题目，班里一位名叫高斯的学生，就把他写好答案的小石板交上去了，而其他同学都还在紧张地、逐一地累加这一大堆数字。起初老师毫不在意，这么快就交上来，谁知道写了些什么呢？后来发现，全班只有一位同学做对了，就是这位最快交卷的高斯。

高斯的解答方法让小伙伴们和老师惊呆了。原来，高斯把这 100 个数从两头往中间一边取一个，配起对来，1 和 100，2 和 99，3 和 98，……，共计配成 50 对，每一对两个数相加，都等于 101，因而结果为 101×50=5050。

请根据时间复杂度分析法，比较高斯和他的小伙伴们的算法优劣。

描述二：

时间穿越到了 2014 年，在德国一所乡村小学的三年级信息技术课堂上，计算机老师为了惩罚吵闹的学生，出了一道编程题：先用文本文件记录 100 个没有规律的数字，然后将文件分发给学生，要求学生利用计算机编程求出它们的和。谁算出正确结果，谁就可放学回家。

编写程序把 100 个没有规律的数一个一个地加起来，这件事让三年级的小学生来做，是一种考验。不料，老师刚说完题目，班里的一位名叫高斯的学生，就把答案报给了老师，而其他同学都还在紧张地、逐一地加着这一大堆数字。

高斯的编程方法同样让小伙伴们和老师惊呆了。原来，高斯是利用数组并借助了一个循环语句，将这 100 个数进行快速求解的。

请对数据的不同结构进行分析，比较高斯和他的小伙伴们的解题优劣。

【案例说明】

在案例描述一中，高斯是通过简单顺序结构，并套用公式实现了 100 个连续整数求和；在描述二中，高斯是通过循环结构，并借助数组实现了 100 个无序数的相加。

【案例目的】

通过用不同的结构来解决同一问题，以及用相同结构的不同算法来解决同一问题，让学习者体会结构和算法对程序效率的影响。

为了比较结构和算法的优劣，先来学习一下相应的知识点吧。

1.2　知识点学习

“ 数据结构 ” 是计算机及相关专业的专业基础课之一，是一门十分重要的核心课程，主要学习用计算机实现数据组织和数据处理的方法。它也为计算机专业的后续课程（如操作系统、编译原理、数据库原理和软件工程等）的学习打下了坚实的基础。

另外，随着计算机应用领域的不断扩大，非数值计算问题占据了当今计算机应用的绝大多数领域，简单的数据类型已经远远不能满足需要，各数据元素之间的复杂联系已经不是普通数学方程式所能表达的、无论是设计系统软件，还是设计应用软件，都需要用到各种复杂的数据结构，因此，掌握好 “ 数据结构 ” 课程的知识，对于提高解决实际问题的能力将有很大的帮助。实际上，一个 “ 好 ” 的程序无非就是选择了一个合理的数据结构和一个高效的算法，而高效的算法很大程度上取决于描述实际问题所采用的数据结构。所以，要想编写出 “ 好 ” 的程序，仅仅学习计算机语言是不够的，还必须扎实地掌握数据结构的基本知识和技能。

1.2.1　数据结构

在了解了数据结构的作用之后，下面介绍数据结构的定义以及相关的基本概念。

1.2.1.1　数据结构相关概念

数据是用符号对现实世界的事物及活动做出的抽象描述，其中，符号可以是文字符号、数字符号以及其他规定的符号。例如，班级点名册上的名字、学号、考勤记录、平时成绩等都是数据。从计算机的角度来说，数据就是能输入到计算机中并且能被计算机处理的符号的集合。例如，201302 班学生数据就是该班全体学生记录的集合。

数据元素是数据的基本单位。例如，201302 班点名册中的每个学生记录都是一个数据元素。数据元素也可称为元素、结点、顶点、记录等，在计算机中通常被作为一个整体来进行考虑和处理。一个数据元素可以由若干个数据项组成。数据项是具有独立含义的最小的数据单位，也称为字段或域。例如，201302 班点名册中的每个数据元素（即学生记录）是由学号、姓名、出勤和平时成绩等数据项组成的。

数据结构是指数据和数据之间的关系，可以看成是相互之间存在着某种特定关系的数据元素的集合。数据结构包括数据的逻辑结构、数据的物理结构和数据的运算 3 个方面。

数据的逻辑结构表示数据之间的逻辑关系，与数据的存储无关，是独立于计算机的；数据的物理结构（即存储结构）是数据元素及其关系在计算机存储器中的存储方式，即物理结构是计算机语言的实现，是逻辑结构在计算机中的存储方式，依赖于计算机语言；数据的运算是施加在数据上的操作，它是定义在数据的逻辑结构之上的，每种逻辑结构都有一组相应的运算。例如，最常用的运算有插入、删除、查找、排序等。数据的运算最终需在对应的存储结构上用算法来实现。

所以，数据结构是一门讨论描述现实世界实体的数学模型（非数值计算）及其之上的运算在计算机中如何表示和实现的学科。

【例 1.1】 表 1.1 所示的学生表中的数据元素是学生记录，每个数据元素由 4 个数据项（即学号、姓名、性别和年龄）组成。试讨论其存储结构。

表 1.1　学生表

学　　号	姓　　名	性　　别	年　　龄
3	张飞	男	18
2	刘备	男	23
14	吕布	男	19
5	貂婵	女	18
26	关羽	男	21
12	小乔	女	17

解：该表中的记录顺序反映了数据元素之间的逻辑关系，用学号标识每个学生记录，这种逻辑关系可以表示为 <3,2>,<2,14>,<14,5>,<5,26>,<26,12>。其中 “ $<a_i,a_{i+1}>$ ” 表示元素 a_i 和 a_{i+1} 是相邻的，即 a_i 在 a_{i+1} 之前，a_{i+1} 在 a_i 之后。

这些数据在计算机存储器中的存储方式就构成了存储结构。通常可以采用 C++ 语言中的结构体数组和链表两种方式实现其存储结构。

存放上述学生表的结构体数组 Stud 定义如下：

```
struct
{
    int no;                    // 存储学号
    char name[8];              // 存储姓名
    char sex[2];               // 存储性别
    int age;                   // 存储年龄
 } Stud[7]={{3, " 张飞 "," 男 ",18},…, {12," 小乔 "," 女 ",17}};
```

数组 Stud 中，每个元素在内存中按顺序存放，即 Stud[i] 存放在 Stud[i+1] 之前，Stud[i+1] 存放在 Stud[i] 之后。

存放学生记录表的链表的结点类型 StudType 定义如下：

```
typedef struct studNode
{
    int no;                    // 存储学号
    char name[8];              // 存储姓名
    char sex[2];               // 存储性别
    int age;                   // 存储年龄
    struct studNode *next;     // 存储指向下一个学生的指针
}StudType;
```

学生表中，每个学生记录采用一个 StudType 类型的结点存储，一个学生结点的 next 域指向逻辑结构中其后继学生对应的结点，从而构成一个链表，其存储结构如图 1.1 所示。

图 1.1　学生的链表存储结构

head 是一个指针，指向学生链表的首结点（name 域为张飞的结点），通过 head 指针可以得到首结点地址。首结点的 next 域存放第二个结点（name 域为刘备的结点）的地址，然后由它得到下一个结点的地址。依此类推，通过指针可以找到任何一个结点的地址。

对于学生记录表这种数据结构，可以进行一系列的运算。例如，增加一个学生记录、删除一个学生记录、查找性别为“女”的学生记录、查找年龄为 18 岁的学生记录等。从前面介绍的两种存储结构可以看出，同样的运算，在不同的存储结构中其实现过程是不同的。例如，查找学号为 5 的学生的姓名，对于 Stud 数组，可以从 Stud[0] 开始比较，Stud[0].no 不等于 5，再与 Stud[1].no 比较，直到 Stud[3].no 等于 5，返回 Stud[3].name。而对于以 head 为首结点指针的链表，从 head 所指结点开始比较，*head->no 不等于 5，从它的 next 域得到下一个结点的地址，再与下一个结点的 no 域比较，直到某结点的 no 域等于 5，返回其 name 域。

由例 1.1 可以得出结论：对于一种数据结构，其逻辑结构是唯一的，而存储（物理）结构可以有多种，并且对于同一运算，不同的存储结构，其实现过程可能不同。这类似于碳元素，其原子的逻辑结构是唯一的，但可以通过不同的组合方式形成石墨和金刚石。

数据结构的描述通常采用二元组表示如下：

$$B=(D,R)$$

其中，B 是一种数据结构，它由数据元素的集合 D 和 D 上二元关系的集合 R 组成。即：

$$D=\{d_i \mid 1 \leqslant i \leqslant n, n \geqslant 0\}$$

$$R=\{r_j \mid 1 \leqslant j \leqslant m, m \geqslant 0\}$$

其中，d_i 表示集合 D 中的第 i 个结点或数据元素；n 为 D 中结点的个数，如果 n=0，则 D 是一个空集，B 也就无结构可言。r_j 表示集合 R 中的第 j 个关系；m 为 R 中关系的个数，如果 m=0，则 R 是一个空集，表明集合 D 中的结点彼此是独立的，它们之间不存在任何关系。

集合 R 是关系 r 的集合。每个 r 代表一个序偶，对于任一序偶 $<x, y>$（$x, y \in D$），把 x 叫做序偶的第一结点，把 y 叫做序偶的第二结点，结点 x 称为结点 y 的前驱结点（简称前驱），结点 y 称为结点 x 的后继结点（简称后继）。

若一个结点没有前驱，则称该结点为开始结点；若一个结点没有后继，则称该结点为终端结点。若有 $<x, y> \in R$，且 $<y, x> \in R$（$x, y \in D$），则用圆括号代替尖括号，即 $(x,y) \in R$，称 (x, y) 为对称序偶。

【例 1.2】采用二元组表示例 1.1 中的学生表。

学生表中共有 6 个结点，依次用 s1 ～ s6 表示，则对应的二元组表示为 $B=(D,R)$，其中：

```
D={s1,s2,s3,s4,s5,s6 }
R={r}
r={<s1,s2>,<s2,s3>,<s3,s4>,<s4,s5>,<s5,s6>}
```

用图形可以形象地表示这种数据结构，如图 1.2 所示，图形中的每个结点对应着一个数据元素，两结点之间的连线对应着关系中的一个序偶。

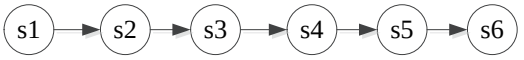


图 1.2　学生表数据结构图示

1.2.1.2　数据的逻辑结构

在不会产生混淆的前提下，常常将数据的逻辑结构简称为数据结构。数据的逻辑结构主要有以下几类。

1．集合

集合是指数据元素同属于一个集合，别无其他关系。

2．线性结构

线性结构中的结点之间是一对一的关系，其特点是开始结点和终端结点都是唯一的，除开始结点和终端结点外，其余结点都有且仅有一个前驱，有且仅有一个后继。顺序表就是典型的线性结构。

3．树形结构

树形结构中的结点之间存在一对多的关系，其特点是每个结点最多只有一个前驱，但可以有多个后继，可以有多个终端结点。二叉树就是一种典型的树形结构。

4．图形结构

图形结构中的结点之间存在多对多的关系，其特点是每个结点的前驱和后继的个数都可以是任意的。因此，可能没有开始结点和终端结点，也可能有多个开始结点和终端结点。

树形结构和图形结构统称为非线性结构。由数据的逻辑结构的相关定义可知，线性结构是树形结构的特殊情况，树形结构是图形结构的特殊情况。本书的一级目录正是按照这种逻辑结构排列的，如图 1.3 所示。

图 1.3　本书一级目录的结构

【例 1.3】有一种数据结构 $B1=(D,R)$，其中：

```
D={ 1,5,8,12,20,26,34}
```

```
R={r}
r={<1,8>,<8,34>,<34,20>,<20,12>,<12,26>,<26,5>}
```

画出其逻辑结构图示。

解：对应的图形如图 1.4 所示。

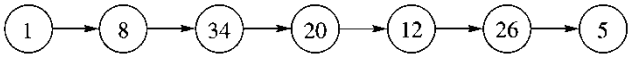


图 1.4　对应 B1 的逻辑结构图示

从图 1.4 看出，每个结点有且只有一个前驱（除第一个结点外），有且仅有一个后继（除最后一个结点外）。其特点是结点之间为一对一的联系，即线性关系。这种数据结构就是线性结构。

【例 1.4】有一种数据结构 $B2=(D, R)$，其中：

```
D={a,b,c,d,e,f,g,h,f,j}
R={r}
r={<a,b>,<a,c>,<a,d>,<b,e>,<c,f>,<c,g>,<d,h>,<d,i>,<d,j>}
```

画出其逻辑结构图示。

解：对应的图形如图 1.5 所示。

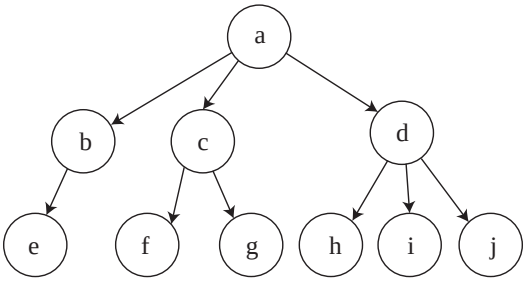


图 1.5　树形结构

从图 1.5 看出，每个结点有且只有一个前驱（除树根结点外），但有多个后继（树叶结点可看作具有 0 个后继结点）。这种数据结构的特点是数据元素之间为一对多联系，即层次关系。这种数据结构就是树形结构。

【例 1.5】有一种数据结构 $B3=(D,R)$，其中：

```
D={a,b,c,d,e}
R={r}
r={(a,b),(a,c),(a,d),(b,c),(b,e),(c,d), (c,e),(d,e)}
```

画出其逻辑结构图示。

解：对应的图形如图 1.6 所示。

从图 1.6 可以看出，每个结点可以有多个前驱和多个后继。其特点是数据元素之间为多对多联系，即图形关系。这种数据结构就是图形结构。

【例 1.6】 有一种数据结构 $B4=(D, R)$，其中：

```
D={48,25,64,57,82,36,75}
R={r1,r2}
r1={<25,36>,<36,48>,<48,57>,<57,64>,<64,75>,<75,82>}
r2={<48,25>,<48,64>,<64,57>,<64,82>,<25,36>,<82,75>}
```

画出其逻辑结构图示。

解：对应的图形如图 1.7 所示。它是一种图形结构，其中，$r1$（对应图中虚线部分）为线性结构，$r2$（对应图中实线部分）为树形结构。

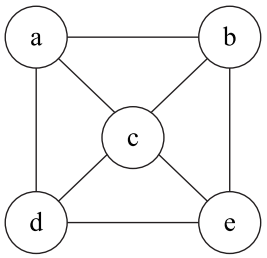


图 1.6　对应 $B3$ 的逻辑结构图示

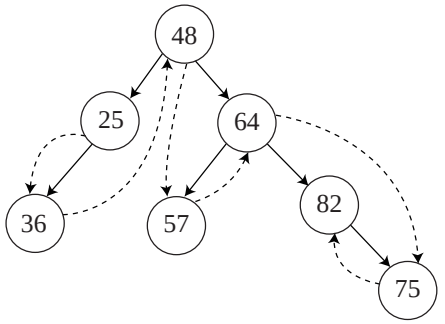


图 1.7　对应 $B4$ 的逻辑结构图示

1.2.1.3　数据的存储结构

在例 1.1 中，学生表采用了数组和链表两种存储方式，前者是典型的顺序存储方法，后者是典型的链式存储方法。归纳起来，在数据结构中有 4 种常用的存储方法。

1. 顺序存储方法

该方法是把逻辑上相邻的结点存储在物理位置上相邻的存储单元里，在算法的 C++ 实现中，顺序存储结构是用数组来描述的。

顺序存储方法的主要优点是节省存储空间，因为分配给数据的存储单元全用于存放结点的数据，结点之间的逻辑关系没有占用额外的存储空间。采用这种方法时，可实现对结点的随机存取，即每个结点作为一个数组元素对应一个下标，由该下标可直接计算出结点的存储地址。例如，对于数组 A 中的数组元素 A[i]，可以通过 *(A+i) 进行存取。顺序存储方法的主要缺点是不便于修改，对结点进行插入、删除运算时，可能要移动一系列结点。

2. 链式存储方法

该方法不要求逻辑上相邻的结点在物理位置上也相邻，结点间的逻辑关系由结点中的指针域来表示。在算法的 C++ 实现中，链式结构要借助于指针类型来描述。

链式存储方法的主要优点是便于修改，在进行插入、删除运算时，仅需修改结点的指

针域，不必移动结点。与顺序存储方法相比，链式存储方法的主要缺点是由于分配给数据的存储单元中有指针域（用来存储结点之间的逻辑关系），故存储空间的利用率较低。另外，由于逻辑上相邻的结点在存储器中不一定相邻，所以不能对结点进行随机存取。

3. 索引存储方法

该方法通常用来存储线性结构。在存储结点信息的同时，还建立附加的索引表，像图书的目录一样。索引表中的每一项称为索引项，索引项的一般形式是：(关键字 , 地址)。其中，关键字可唯一地标识一个结点，地址作为指向结点的指针。这种带有索引表的存储结构可以大大提高数据查找的速度。

索引存储方法的主要优点是可以对结点进行随机访问，在进行插入、删除运算时，只需移动存储在索引表中的结点的存储地址，而不必移动存储在结点表中的结点数据，所以数据修改时效率较高。其缺点是增加了索引表，降低了存储空间的利用率。

4. 哈希（或散列）存储方法

该方法的基本思想是根据结点的关键字，通过哈希函数（也称散列函数）计算出一个值作为该结点的存储地址。

哈希存储方法的优点是查找速度快，只要给出待查结点的关键字，就可以计算出该结点的存储地址。但是哈希存储方法只存储结点的数据，不存储结点之间的逻辑关系，一般只适合要求对数据进行快速查找和插入的场合。

上述 4 种基本的存储方法，既可以单独使用，也可以组合使用。同一种逻辑结构采用不同的存储方法，可以得到不同的存储结构。在实际存储中，需要视具体要求来决定选择使用何种存储结构。

1.2.1.4　数据结构和数据类型

数据类型是和数据结构密切相关的一个概念，容易引起混淆。本节介绍两者之间的差别和抽象数据类型的概念。

1. 数据类型

在 C++ 程序中出现的每个变量、常量或表达式，都需要明确声明它们所属的数据类型。不同类型的变量，其所能取的值的范围不同，所能进行的操作也不同。例如，用 int 声明的变量，在内存中占 2 个字节，能进行加、减、乘、除、求余等运算；用 char 声明的变量，在内存中占 1 个字节，存储的是一个字符，一般不用来做算术运算。

数据类型可分为简单类型和结构类型两种。简单类型中的每个数据（即简单数据）都是无法再分割的整体，如一个整数、实数、字符、指针、枚举量等。结构类型由简单类型按照一定的规则构造而成，一种结构类型中的数据可以分解为若干个简单数据或结构数据，每个结构数据仍可再分。例如，C++ 语言中的数组是一种结构类型，它由多个同一类型的数据组成；结构体也是一种结构类型，它由多个不同类型的数据组成，如例 1.1 中的结构体数组 Stud。

下面回顾一下 C++ 语言中常用的数据类型。

（1）基本数据类型

C++ 语言中的基本数据类型有 int 型、float 型和 char 型。int 型可以有 3 个限定词：

short、long 和 unsigned，分别为短整数、长整数和无符号整数。float 型有 3 种形式：float、double 和 long double，分别为单精度浮点型、双精度浮点型和长浮点型。

（2）指针类型

C++ 语言允许直接对存放变量的地址进行操作。例如，有定义 int i，则 &i 表示变量 i 的地址，也称做指向变量 i 的指针。存放地址的变量称做指针变量。

有关指针的两个操作是：对于定义 int i，则 &i 操作是取变量 i 的地址；对于定义 int *p，其中的 p 是指向一个整数的指针，则 *p 操作是取 p 指针所指的值，即 p 所指地址的内容。

（3）数组类型

数组是同一类型的一组有序数据的集合，包含一维数组和多维数组。数组名用于标识一个数组，下标用于指示一个数组元素在该数组中的顺序位置。

C++ 语言中，数组的下标总是从 0 开始，下标的最大值为数组长度减 1。例如，int a[10] 定义了包含 10 个整数的数组 a，下标范围是 0~9。

（4）结构体类型

结构体由一组称做结构体成员的项组成，每个结构体成员都有自己的标识符。例如：

```
struct  teacher
{
    int no;
    char name[8];
    int age;
}
```

上述语句定义了一个结构体类型 teacher。下面的语句定义了该类型的两个变量 t1 和 t2：

```
struct  teacher  t1,t2;
```

（5）自定义类型

C++ 语言中，允许使用 typedef 关键字来定义等同的数据类型名，例如：

```
typedef char ElemType;
```

上述语句将 char 类型与 ElemType 等同起来。

（6）引用运算符

C++ 语言中提供了一种引用运算符“ & ”。引用即别名，正如“ 赵云 ”字“ 子龙 ”一样，“ 子龙 ”和“ 赵云 ”实乃一人也。当建立引用时，程序用另一个已定义的变量或对象（目标）的名字初始化它，从那时起，引用作为目标的别名使用，对引用的改动实际就是对目标的改动。例如：

```
int a=4;
```

```
int &b=a;
```

上述语句说明变量 b 是变量 a 的引用，b 也等于 4，之后这两个变量同步改变。

引用常用于函数形参中。采用引用型形参，则在函数调用时会将形参的改变回传给实参。例如，有如下函数（其中的形参均为引用型形参）：

```
void Swap(int &x,int &y) // 形参前的 & 符号不是指针运算符，是引用
{
   int tmp=x;
   x=y; y=tmp;
}
```

例如，当用执行语句 Swap(a,b) 时，a 和 b 的值发生了变换。

2．抽象数据类型

抽象数据类型（Abstract Data Type，ADT）是指用户进行软件系统设计时，从问题的数学模型中抽象出来的逻辑数据结构和逻辑数据结构上的运算，不考虑计算机的具体存储结构和运算的具体实现算法。

一个具体问题的抽象数据类型的定义包括数据对象（即数据元素的集合）、数据关系和基本运算三方面的内容。抽象数据类型可用 (*D*,*S*,*P*) 三元组表示。其中，*D* 是数据对象；*S* 是 *D* 上的关系集；*P* 是 *D* 中数据运算的基本运算集。其基本格式如下：

```
ADT 抽象数据类型名
{
   数据对象：数据对象的定义
   数据关系：数据关系的定义
   基本运算：基本运算的定义
}ADT 抽象数据类型名
```

例如，抽象数据类型复数的定义为：

```
ADT Complex
{
  数据对象：
    D={e1,e2|e1,e2 均为实数 }
  数据关系：
    R1={<e1,e2>| e1 是复数的实数部分 ,e2 是复数的虚数部分 }
  基本操作：
    AssignComplex(&Z,v1,v2);          // 构造复数 Z, 其实部和虚部分别赋以参数 v1 和 v2 的值
    DestroyComplex(&Z);               // 复数 Z 被销毁
    GetReal(Z,&real);                 // 用 real 返回复数 Z 的实部值
```

```
    GetImag(Z,&Imag);                    // 用 Imag 返回复数 Z 的虚部值
    Add(z1,z2,&sum);                     // 用 sum 返回两个复数 z1,z2 的和值
} ADT Complex
```

1.2.2 算法

计算机软件的最终成果都是以程序的形式表现的，数据结构的各种操作都是以算法的形式描述的。数据结构、算法和程序密不可分，它们之间的关系是：数据结构 + 算法 = 程序。

1.2.2.1 什么是算法

算法是对特定问题求解步骤的一种描述。简单地说，一个算法就是一个解题方法和步骤，是指令的有限序列，其中每一条指令表示计算机的一个或多个操作。一个算法具有以下 5 个重要的特性：

（1）有穷性

一个算法必须能够（对任何合法的输入值）在执行有穷步之后结束，且每一步都可在有穷时间内完成。

（2）确定性

算法中的每一步都必须有确切的含义，并且在任何条件下都只有一条执行路径，即对于相同的输入，只能得出相同的输出。

（3）可行性

算法中的所有操作都必须足够基本，且可以通过已经实现的基本操作运算有限次得以实现。

（4）有输入

具有 0 个或多个输入。有些输入量需要在算法执行过程中输入，而有的算法表面上可以没有输入，实际上已被嵌入算法之中。

（5）有输出

有一个或多个输出，这些输出与输入之间存在确定关系的量。

注意：算法和程序不同。一方面，程序不一定满足有穷性。如操作系统程序，只要系统能正常工作，就不会停止；没有用户操作，就一直处于动态等待中。另一方面，程序中的指令必须是机器可执行的，而算法中的指令则没有这个限制。所以说，算法代表了对问题的解，而程序是算法在计算机上的特定实现。

【例 1.7】设计一个算法，读入 3 个整数 x、y 和 z 的值，要求从大到小输出这 3 个数。

算法设计如下：

（1）输入 x、y、z。

（2）如果 $x<y$，交换 x、y。

（3）如果 $y<z$，交换 y、z。

（4）如果 $x<y$，交换 x、y。

（5）输出 x、y、z 的值。

（6）算法结束

【例 1.8】判断下列两段描述是否满足算法的特征，如不满足，说明它们违反了哪些特征。

程序段 1：

```
void Test1()
{
    n=2;
    while (n%2==0)
        n=n+2;
    cout<<n<<endl;
}
```

程序段 2：

```
void Test2()
{
    y=0;
    x=10/y;
    cout<<x<<'\t'<<y<<endl ;
}
```

程序段 1 中，算法是一个死循环，违反了算法的有穷性特征；程序段 2 中，算法包含除零错误，违反了算法的可行性特征。

1.2.2.2　算法描述

算法可以使用不同的方法来描述，最简单的是使用自然语言来描述，用自然语言描述算法操作简单、方便阅读，但是不够严谨；算法也可以使用程序流程图、N-S 图等来描述，这样的描述过程简洁明了，但是不能直接在计算机上执行；还可以使用伪代码语言来描述，它是介于程序设计语言和自然语言之间的一种语言，忽略了高级程序设计语言的严格语法规则和描述细节，容易理解。本书采用 C++ 语言来描述，它的优点是类型丰富、语句精炼，编写的程序结构化程度高、可读性强。

下面是常用的用于描述算法的 C++ 语言基本语句：

（1）输入语句

```
cin>> 变量名 1[>> 变量名 2>>…>> 变量名 n];
```

（2）输出语句

```
cout<< 表达式 1[<< 表达式 2<<…<< 表达式 n];
```

（3）赋值语句

```
变量名 = 表达式 ;
```

（4）条件语句

```
if< 条件 >< 语句 >;
```

或者

```
if< 条件 >< 语句 1>else< 语句 2>;
```

（5）循环语句

```
① while< 表达式 >
{
< 循环体语句 >;
}

② do
{
< 循环体语句 >;
}while< 表达式 >;

③ for(< 赋初值表达式 1>;< 条件表达式 2>;< 步长表达式 3>)
{
< 循环体语句 >;
}
```

（6）返回语句

```
return(< 返回表达式 >);
```

（7）定义函数语句

```
< 函数返回值类型 >< 函数名 >(< 类型名 >< 形参 l>,< 类型名 >< 形参 2>,…)
{
   < 说明部分 >;
   < 函数语句部分 >;
}
```

（8）调用函数语句

```
< 函数名 >(< 实参 1>,< 实参 2>,…);
```

【例 1.9】对于例 1.7 的算法，用程序语言描述如下。

```
void Descending()
{
    int x,y,z,temp;
    cout<<" 输入 x,y,z"<<endl;
    cin>>x>>y>>z;
  if (x<y)
  {   temp=x;x=y;y=temp; }        // 交换 x 和 y，使 x ≥ y
    if (y<z)
  {   temp=y;y=z;z=temp; }        // 交换 y 和 z，使 y ≥ z
    if (x<y)
    {  temp=x;x=y;y=temp;}        // 交换 x 和 y，使 x ≥ y
  cout<<x<<'\t'<<y<<'\t'<<z<<endl;
}
```

1.2.2.3　算法分析

要解决一个问题，通常可以设计出若干不同的算法。通过对算法进行分析，可以从中选择较为合适的一种，也可以对现有算法进行改进，以设计出更好的算法。

1. 算法设计的目标

算法首先应该是正确的，其次应该易于理解、编码、调试。除此之外，执行算法所耗费的时间和存储空间也是需要考虑的因素。一个好的算法，应该能有效地使用存储空间和有较高的时间效率。

2. 算法效率分析

一个算法用高级语言实现后，在计算机上运行时所消耗的时间与很多因素有关，如计算机的运行速度、编写程序采用的计算机语言、编译产生的机器语言代码质量和问题的规模等。在这些因素中，前 3 个都与具体的机器有关。除去这些与计算机硬件、软件有关的因素，一个算法所耗费的时间是该算法中每条语句执行时间的总和，而每条语句的执行时间是该语句的执行次数（也称语句频度）与该语句执行一次所需时间的乘积。假设执行每条语句所需的时间均为单位时间，则一个算法的时间耗费就是该算法中所有语句的频度之和。

在一个算法中，进行基本运算的次数越少，其运行时间也就相对越少；基本运算次数越多，其运行时间也就相对越多。所以，通常把算法中包含基本运算次数的多少称为算法的时间复杂度。也就是说，一个算法的时间复杂度是指该算法的基本运算次数。算法中基

本运算次数 $T(n)$ 是问题规模 n 的某个函数 $f(n)$，记作：

$$T(n)=O(f(n))$$

其中，记号“O”读作“大O”，即Order（数量级）的缩写，表示随问题规模 n 的增大，算法执行时间的增长率和 $f(n)$ 的增长率相同。$T(n)$ 称为算法的时间复杂度。例如，一个算法中频度最大的语句执行次数为 $f(n)=5n^2+3n+7$，则 $T(n)=O(n^2)$。

一个没有循环的算法的基本运算次数与问题规模 n 无关，记作 $O(1)$，也称做常数阶。一个只有一重循环的算法的基本运算次数与问题规模 n 呈线性增长关系，记作 $O(n)$，也称做线性阶。其他常用的还有平方阶 $O(n^2)$、立方阶 $O(n^3)$、对数阶 $O(\log_2 n)$、指数阶 $O(2^n)$ 等。各种不同数量级对应的值存在如下关系：

$$O(1)<O(\log_2 n)<O(n)<O(n*\log_2 n)<O(n^2)<O(n^3)<O(2^n)<O(n!)$$

将算法的时间复杂度采用这种数量级形式表示，使得算法效率分析变得非常简单。通常只需要分析影响一个算法时间复杂度的主要部分即可，而不必再对每一步都进行详细的分析。

大O表示法有两个重要规则：加法规则和乘法规则。

（1）加法规则

当两个并列的算法段的时间代价分别为 $f_1(n)=O(g_1(n))$ 和 $f_2(m)=O(g_2(m))$ 时，两个算法段连在一起的时间代价为：

$$f(n,m)=f_1(n)+f_2(m)=O(\max(g_1(n),g_2(m)))$$

（2）乘法规则

如果存在循环嵌套，关键操作应该在最内层循环中。首先，自外向内、层层分析每一层的渐近时间复杂度，然后利用大O表示法的乘法规则来计算其时间复杂度。当两个嵌套时间复杂度的时间代价分别为 $f_1(n)=O(g_1(n))$ 和 $f_2(m)=O(g_2(m))$ 时，那么整个算法段的时间代价为：

$$f(n,m)=f_1(n)*f_2(m)=O(g_1(n)*g_2(m))$$

一个算法执行的基本运算次数常常因输入不同而不同。算法执行的基本运算次数，不仅取决于输入数据元素的个数，即问题的规模，还与输入数据的性质有关。比如，对一组数据进行起泡法排序，如果给定的序列本身是有序的，那么只需要对所有数据进行比较判断，即可得到一个最好的时间复杂度；如果这组数据是逆序的，则所有数据每比较一次，就要产生一次交换，这样就会得到一个最坏的时间复杂度。

【例 1.10】 设计一个算法，求含 n 个整数元素的序列中前 i 个元素的最大值，并分析算法的平均时间复杂度。

对应算法如下：

```
int fun(int a[],int n,int i)
{
    int j,max=a[0];
    s=0;
    for (j=0;j<=i-1;j++)
     if(a[j]>max)
        max=a[j];
   return max;
}
```

分析算法可知，i 的取值范围为 1~n；对于求前 i 个元素的最大值时，需要元素比较 (i-1)-1+1=i-1 次。在等概率的情况下：

$$T(n)=\sum_{i=1}^{n}\frac{1}{n}*(i-1)=\frac{1}{n}\sum_{i=1}^{n}(i-1)=\frac{n-1}{2}=O(n)$$

本算法的平均时间复杂度为 O(n)。

3．算法存储空间分析

一个算法的空间复杂度记作 $S(n)$，是指该算法在运行过程中所耗费的辅助存储空间，也是问题规模 n 的函数。如果一个算法所耗费的存储空间与问题规模 n 无关，记作 $S(n)$=O(1)。

算法所耗费的存储空间包括算法本身所占用的存储空间、算法输入 / 输出所占用的存储空间和算法在运行过程中临时占用的辅助存储空间。算法输入 / 输出所占用的存储空间由算法解决的问题规模所决定，不随算法的改变而改变；算法本身所占用的存储空间与实现算法的程序代码长度有关，代码越长，占用的存储空间越大；算法在运行过程中临时占用的辅助存储空间随算法的不同而不同，有的算法不随问题规模的大小改变，而有的算法占用的辅助存储空间与解决问题的规模 n 有关，随 n 的增大而增大。例如，起泡排序算法在运行过程中临时占用的工作单元数与解决问题的规模 n 无关，即空间复杂度为 O(1)。

1.2.3　数据结构 + 算法 = 程序

N. Wirth 的“数据结构 + 算法 = 程序”公式对计算机科学的影响程度足以媲美物理学中爱因斯坦的质能公式 $E=mc^2$——一个公式展示出了程序的本质。

程序设计是必须认真规划的系统工程，从数据结构的设计到算法的设计，都应在可行性的基础上充分考虑其效率、扩展、异常和可维护性等。

1.3　案例问题解决

1.3.1　1787 年高斯算法——比较算法优劣

【算法思路】

1787 年，高斯的小伙伴们通过累加求解 1~100 的和，而高斯通过公式（即 sum =(1+100)*100/2 进行求解，因此能在较短的时间内求出结果。

【源程序】

```
void main()
{
    int sum=0;                                  // 高斯的算法
```

```
    sum=(1+100)*100/2;
    cout<<sum<<endl ;
    sum=0;                                    // 其他小伙伴的算法
    for(int i=1;i<=100;i++)
        sum+=i;
    cout<<sum<<endl;
}
```

比较两个算法可知，高斯算法的时间复杂度为 O(1)，而其他小伙伴算法的时间复杂度为 O(*n*)。

1.3.2 2014 年高斯算法——比较结构优劣

【算法思路】

对于无序的 100 个数，定义 100 个变量是不现实的，直接累加也很麻烦。采用数组这种数据结构，利用循环来实现，问题就变简单了。

【源程序】

```
void main()
{
    const int N=100
    int a[N]={12,23,34,45,56,76,34,23,67……},sum=0;
    for(int i=0;i<N;i++)
        sum+=a[i];
    cout<<sum<<endl;
}
```

虽然此处高斯算法的时间复杂度为 O(*n*)，而其他同学直接累加的时间复杂度为 O(1)，但采用数组的循环编写程序比用直接累加 100 个变量或用计算器累加 100 个常数要方便得多。更重要的是，前者比后者降低了数据与程序的耦合度，从而更容易理解和维护。

1.4 知识与技能扩展

1. ACM 程序设计大赛

ACM 程序设计大赛是大学级别最高的脑力竞赛，素来被冠以“ 程序设计中的奥林匹克 ”之称。大赛自 1970 年开始，至今已有 40 多年历史，是世界范围内历史最悠久、规模最大的程序设计竞赛。比赛形式是：经过校级和地区级选拔的参赛组，于指定的时间、地点参

加世界级决赛；由 3 个成员组成的小组应用一台计算机解决 6~8 个生活中的实际问题，参赛队员必须在 5 个小时内编完程序，并进行测试和调试。大赛对参赛学生的逻辑分析能力、策略制定能力等方面均具有极大的挑战性。大赛提倡在压力较大的情况下培养学生的创造力、团队合作精神，从而挑选和发掘世界上最优秀的程序设计人才。

2010 年 2 月 5 日, 哈尔滨工程大学承办了 ACM-ICPC 全球总决赛。来自世界 33 个国家和地区的 103 个编程小组共 300 多名大学生计算机编程高手参赛。上海交通大学最终获得了全球总决赛第一名，莫斯科国立大学和台湾大学分获亚、季军，清华大学、中山大学和复旦大学均榜上有名。有美国人评论说：前十名几乎被中国人和俄罗斯人包揽了。

2.《计算机程序设计艺术》

高德纳（Don E. Knuth）所著的《计算机程序设计艺术》被称为程序设计中的圣经。像 KMP 这样令人不可思议的算法，在此书中比比皆是。难怪连 Bill Gates 都说：“ 如果能做对书里所有的习题，就直接来微软上班吧！ ”

对于 Don E. Knuth 本人，一生中获得的奖项和荣誉不计其数，包括图灵奖、美国国家科学金奖、美国数学学会斯蒂尔奖（AMS Steel Prize）以及因发明先进技术荣获的极受尊重的京都奖（Kyoto Prize）等。他写过 19 部书和 160 余篇论文，每一篇著作都能用 “ 影响深远 ” 来形容。Don E. Knuth 也被公认是美国最聪明的人之一。当年他上大学时，常编写各种各样的编译器来挣钱，各类编程比赛，只要他参加，总是第一名。他同时也是世上少有的连续编程 40 多年的程序员之一。他除了是科学与技术上的泰斗之外，更是无可非议的写作高手，其撰写的技术文章堪称一绝，文风细腻，讲解透彻，思路清晰，而且没有学究气。

课 后 习 题

一、单项选择题

1. 数据结构是指（　　）。

A. 数据元素的组织形式　　B. 数据类型

C. 数据存储结构　　D. 数据定义

2. 数据在计算机存储器内表示时，物理地址与逻辑地址不相同的称为（　　）。

A. 存储结构　　B. 逻辑结构

C. 链式存储结构　　D. 顺序存储结构

3. 树形结构是指数据元素之间存在一种（　　）。

A. 一对一关系　　B. 多对多关系

C. 多对一关系　　D. 一对多关系

4. 设语句 “*x*++;” 的执行时间是单位时间，则以下语句的时间复杂度为（　　）。

```
for(i=1; i<=n; i++)
```

```
for(j=i; j<=n; j++)
x++;
```

A. O(1)　　B. O(n^2)　　C. O(n)　　D. O(n^3)

5. 数据在计算机内有链式和顺序两种存储方式。从存储空间使用的灵活性上考虑，链式存储的灵活性比顺序存储要（　　）。

A. 低　　B. 高　　C. 相同　　D. 不好说

二、填空题

1. 数据结构按逻辑结构可分为两大类，分别是_____和_____。

2. 数据的逻辑结构有 4 种基本形态，分别是_____、_____、_____和_____。

3. 一个算法的效率可用_____复杂度和_____复杂度衡量。

三、求下列程序段的时间复杂度

1.
```
x=0;
for(i=1;i<n;i++)
    for(j=i+1;j<=n;j++)
        x++;
```

2.
```
x=0;
for(i=1;i<n;i++)
    for(j=1;j<=n-i;j++)
        x++;
```

3.
```
int i,j,k;
for(i=0;i<n;i++)
    for(j=0;j<=n;j++)
    {  c[i][j]=0;
       for(k=0;k<n;k++)
          c[i][j]=a[i][k]*b[k][j]
    }
```

4.
```
 i=n-1;
while((i>=0)&&A[i]!=k))
  j--;
return (i);
```

上 机 实 战

1. 编写一个程序，计算任意输入的正整数的各位数字之和，并分析算法的时间复杂度。

2. 编写一个程序，打印九九乘法表，并分析算法的时间复杂度。

课堂微博：