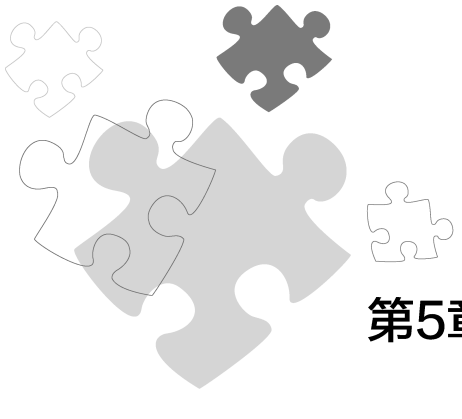


第5章　程序设计与算法导论

计算问题的求解离不开计算方法和计算工具的支撑，而计算的方法就是算法。算法描述了解决问题的方法和步骤，它是计算的灵魂，也是计算机科学的核心。本章从实际生活问题的求解出发，以常见问题为例介绍一些基本算法的设计思路并辅以详细案例分析。然后从程序设计的角度介绍程序设计涉及的基本概念、基本思想和方法，使得读者对程序设计过程有一个初步的了解。最后介绍数据结构的基础知识。

对于初学者，学习程序设计的目的不仅是学习某一种特定的程序设计语言，也不是掌握程序设计语言的语法规则，更重要的是学习算法，并能够应用程序设计语言解决实际问题。

5.1　算法及其描述

计算机求解问题的关键之一在于算法，算法是利用计算机来求解问题的关键。本节将简要介绍算法的基础知识，包括算法概念、算法特征、算法描述、算法评价以及常见问题的算法描述。掌握了这些基础知识，将为利用计算机求解问题提供良好的基础。

5.1.1　计算思维基础与算法基础

计算思维是运用计算机科学的基础概念进行问题求解、系统设计以及人类行为理解等涵盖计算机科学广度的一系列思维活动，由周以真教授于 2006 年 3 月首次提出。2010 年，周以真教授又指出计算思维是与形式化问题及其解决方案相关的思维过程，其解决问题的表示形式应该能有效地被信息处理代理执行。

计算思维融合了数学和工程等其他领域的思维方式，是人类求解问题的一条途径。计算思维的本质是抽象(Abstract)和自动化(Automation)，它反映了计算的根本问题——什么能被有效地自动执行。计算思维吸取了问题解决所采用的一般数学思维方法，现实世界中巨大复杂系统的设计与评估的一般工程思维方法，以及复杂性、智能、心理、人类行为的理解等的一般科学思维方法。其优点是计算思维建立在计算过程的能力和限制之上，由人和机器执行。计算方法和模型使我们敢于去处理那些原本无法由个人独立完成的问题求解和系统设计。

与数学和物理科学相比，计算思维中的抽象显得更为丰富，也更为复杂。数学抽象的最大特点是抛开现实事物的物理、化学和生物学等特性，而仅保留其量的关系和空间的形式，而计算思维中的抽象却不仅仅如此。算法就是对基于计算思维的解决问题的方法的描述。算法代表着用系统的方法描述解决问题的策略机制，它由一系列操作步骤组成，通过这些步骤的自动执行可以解决指定的问题。也就是说，通过一定规范的输入，人或计算机自动执行这一系列操作步骤，在有限时间内获得所要求的输出。如果一个算法有缺陷，或不适合于某个问题，执行这个算法将不会解决这个问题。不同的算法可能用不同的时间、空间或效率来完成同样的任务。

算法中指令描述的是一个计算，当其运行时能从一个初始状态和初始输入（可能为空的）开始，经过一系列有限而清晰定义的状态，最终产生输出并停止于一个终态。一个状态到另一个状态的转移不一定是确定的，包括随机化算法在内的一些算法，包含了一些随机输入。

1. 数学思维与计算思维的关系

例如，求解 SUM＝1＋2＋3＋…＋n，是程序设计课中经常使用的一道题目。

对于这个问题，数学解法与编程解法有很大区别，产生这种区别的原因是数学与计算机在解决问题的方式上有所差异，而这种差异的实质，就是数学思维和计算思维两种思维方式的不同。数学思维的特征是概念化、抽象化和模式化，在解决问题时强调定义和概念，明确问题条件，把握其中的函数关系，通过抽象、归纳、类比、推理、演绎和逻辑分析，将概念和定义、数学模型、计算方法等与现实事物建立联系，用数学思想解决问题。计算思维是按照计算机科学领域所特有的解决方式，对问题进行抽象和界定，通过量化、建模、设计算法和编程等方法，形成计算机可处理的解决方案。

对比后可以发现，数学思维是人的大脑的思维，计算思维同样是人的大脑的思维，但却是在数学思维的基础上解决问题。也就是说，计算思维与数学思维本质上非常相似，只是在实现问题的解决方案时要依靠不同的执行对象。经过数学思维所形成的解决方案，可以单纯依靠人的大脑来实现，而经过计算思维所形成的解决方案，大都可以借助计算工具，通过机器的“自动执行”来实现。

狭义上说，计算思维源于数学思维，两者具有一致性，所不同的是，计算思维在继承数学思维的同时，结合了计算机科学的思想特征，也就是在实际理论的基础上，注重考虑客观环境的条件限制，提出可行方案。

2. 计算思维与算法及程序设计的关系

计算思维是一种抽象的思维活动，算法则是把这种思维活动具象化，描述成具体的方法与步骤。程序设计则是算法在计算机上的正确实现，它是计算思维的最终结果。

例如前面的问题，求解 SUM＝1＋2＋3＋…＋n。通过计算思维可以得到“直接从 1 累加到 n”的解决方案；算法则要考虑采用何种方法、通过何种步骤来实现这个方案，例如，如何输入与输出，怎样用循环实现累加等；程序设计是将算法所描述的方法与步骤转换成计算机所能理解和操作的指令代码，比如使用 for 语句进行循环、用 SUM＝SUM＋i 赋值语句实现累加等，使程序能够在计算机上运行并获得正确结果。

由此看来，数学思维是计算思维的基础，计算思维是解决问题的一种思考方式，算法是对计算思维的具体设计，程序设计则用于实现算法设计。

综上所述，构建计算思维活动的基本要素是“由问题引发思维、由思维产生算法、由算法形成程序”，它是体现计算思维的关键，是人脑的独立思考活动，所形成的解决方案是多样的，并且不受编程语言的限制，也就是我们所说的“一个问题可以有不同的解决方案，一个方案可以有不同的算法设计，一个算法可以用不同的编程语言来实现”。

5.1.2 算法的概念和特征

“算法”一词最早出现在唐朝，当时有《一位算法》《算法》等专著；之后，宋代出现了《算法绪论》《算法秘籍》等书籍；元代出现了《丁巨算法》；明代出现了《算法统宗》；清代出现了《开平算法》《算法一得》《算法全书》等。其中具有代表性的是宋代数学家杨辉的《杨辉算法》，如图 5-1 所示。

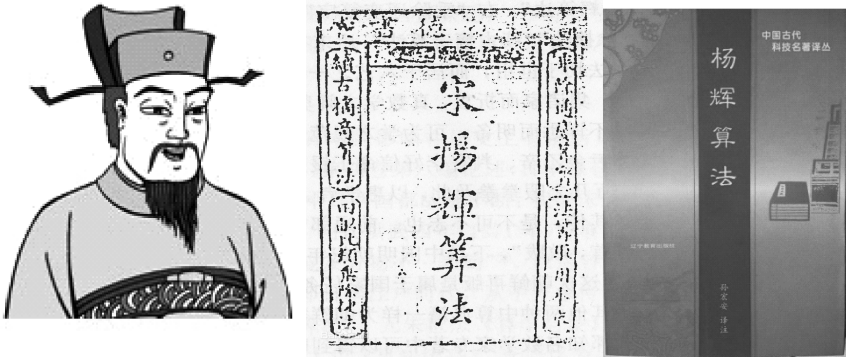


图 5-1 杨辉和杨辉算法

算法的英文名称是 Algorithm，来自于 9 世纪波斯数学家阿尔·花拉子米（Al-Khwarizmi）。算法原为“Algorism”，意思是阿拉伯数字的运算法则，18 世纪演变为“Algorithm”。一般认为，历史上第一个算法是欧几里得算法，即辗转相除法，用于求两个正整数的最大公约数。公元前 3 世纪，古希腊数学家欧几里得在其著作《几何原本》第七卷中阐述了这个算法，如图 5-2 所示，直到现在这个算法还经常使用。

图 5-2 欧几里得

1. 算法的概念

算法（Algorithm）是为解决某个问题而采用的一组明确的、有一定顺序的步骤。它是对问题求解规则的一种过程描述。在算法中要精确定义一组有穷的规则，这些规则规定了解决某一特定类型问题的运算系列，它们指定了相应的操作顺序，以便在有限的步骤内得到所求问题的解答。正如著名计算机教育家科尔曼（Thomas H. Cormen）所说：算法就是任何定义明确的计算步骤，它接受一些值或集合作为输入，并产生一些值或集合作为输出。这样，算法就是将输入转换为输出的一系列计算过程，如图 5-3 所示。

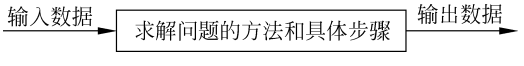


图 5-3 算法定义示意图

一个好的算法是编写程序的模型，因为它能创造计算机程序，其中还包含了程序的精髓。要写出既能正确执行又能提高效率的好程序，算法的学习是不可或缺的。学习算法的同时能提高自己的编程能力。

计算机领域的算法就称为计算机算法，计算机算法可以分为如下两类。

(1) 数值运算算法。用于求解数值，如求一元二次方程的根。

(2) 非数值运算算法。涉及领域较广，主要用于事务管理，如档案管理。

【例 5-1】 欧几里得算法。给定两个正整数 m 和 n，求解其最大公约数，即用辗转相除法能同时整除 m 和 n 的最大正整数。

问题分析：辗转相除法，又称欧几里得算法(EuClidean Algorithm)，它是求两个正整数的最大公因子的算法。设两个正整数为 m 和 n，其中 m、n 不同时为零(设 $m>n$)，输出 m、n 的最大公约数，使用辗转相除法。

算法描述如下。

s1：用 m 除以 n，令所得余数为 r(r 必须小于 n)。

s2：若 r=0，算法结束，输出结果 n；否则继续 s3。

s3：把 n 赋值给 m，把 r 赋值给 n，并返回 s1 继续进行。

例如，求 25 和 15 的最大公约数，计算过程如表 5-1 所示。通过计算得知：余数 r 为 0 时的除数 m 为 5，所以，25 和 15 的最大公约数为 5。

表 5-1　辗转相除法求最大公约数

变　　量	被除数 n	除数 m	余数 r(r=n%m)
第一次值	25	15	10(10!=0)
第二次值	15(n=m)	10(m=r)	5(5!=0)
第三次值	10(n=m)	5(m=r)	0(0==0)

【例 5-2】 计算 1 累加到 n 的和，即 1+2+3+…+n 的和。

问题分析：本题是一个循环累加求和的问题，从第二项开始，每一项和前一项的值相差 1。可以设置两个变量：一个变量为 sum，用来保存每次累加的和，开始取值为 0；另一个变量为 i，用来保存要累加到 sum 的当前项，开始取值为 1。问题的求解过程就是不断地将 i 累加到和 sum 中，每加完一次，i 值加 1，直到 i 的值为 N(N 的值根据需求指定)，最后输出结果 sum。

算法描述如下。

s1：当前项 i 置初值 1，累加和 sum 置初值 0。

s2：判断 i 的值，若 i≤n 则执行步骤 3；否则转 s5 执行。

s3：将 i 加到 sum 中。

s4：i 值加 1，转 s2 执行。

s5：输出结果 sum，算法结束。

例如，求 1 累加到 100 的和，此时 n 的值为 100。

变量 i 的值为 1 时，变量 sum 为 0+1；

变量 i 的值为 2 时，变量 sum 为 0+1+2；

变量 i 的值为 3 时，变量 sum 为 0+1+2+3；

以此类推，变量 i 的值为 100 时，变量 sum 为 0＋1＋2＋3＋…＋100。

通过以上例子可以看出，一个算法由若干步骤构成，这些步骤又按照一定的顺序执行。算法是计算机科学中最具有方法性质的核心概念，是计算机科学领域的基石，被誉为计算机科学的灵魂。

2. 算法的特征

一个算法应该具有以下五个重要的特征：

(1) 有穷性(Finiteness)。一个算法必须在执行有限运算步后终止，每一步必须在有限时间内完成。实际应用中，算法的有穷性应该包括执行时间的合理性。

实际编程时经常会出现死循环的情况，这就是不满足有穷性。当然，这里有穷的概念并不是纯数学意义的，而是指实际应用中的合理范围。如果让计算机去执行一个历时上千年才结束的算法，这虽然是有穷的，但超过了合理的限度，人们认为它不是有效算法。

(2) 确定性(Definiteness)。算法的每一步骤必须有确切的含义，对每一种可能出现的情况，算法都应给出确定的操作，不能有多义性。算法在一定条件下，只有一条执行路径，相同的输入只能有唯一的输出结果。

例如，计算分段函数

$$f(x)=\begin{cases}1 & (x>100)\\ \\ 0 & (x<10)\end{cases}$$

算法描述如下。

s1：输入变量 x。

s2：若 x 大于 100，输出 1。

s3：若 x 小于 10，输出 0。

则算法在异常情况下(10≤x≤100)，执行结果是不确定的。

(3) 零个或多个输入(Input)。输入是指在执行算法时需要从外界获得的必要信息。尽管对于绝大多数算法来说，输入参数都是必要的，但对于个别情况，如打印“**********”这样的内容，不需要任何输入参数，因此，算法的输入可以是零个。

(4) 一个或多个输出(Output)。一个算法有一个或多个输出，算法的目的是为了求解，这里的“解”就是输出。输出可以是打印、显示、磁盘输出等。没有输出的算法是没有意义的。

有些算法可以直接使用别人设计好的，此时只需按照算法的要求进行必要的输入即可得到输出结果。但是，对于程序设计人员来说，必须要会自己设计算法，并且根据算法编写程序。

(5) 有效性(Effectiveness)。算法中执行的任何计算步骤都可以被分解为基本的可执行的操作步骤，即每个计算步骤都可以在有限时间内完成。每一个步骤都应当能有效地执行，并得到确定的结果。例如，如果算法中有除法操作，由于 0 不能作为除数，若不能排除这种情况，这个算法就是无效的。

5.1.3 算法的描述

算法描述(Algorithm Description)是指对设计出的算法，用一种方式进行详细的描述，以便与人交流。算法可采用多种描述语言来描述，各种描述语言在对问题的描述能力方面

存在一定的差异。描述算法的方法有很多，常用的有自然语言、传统流程图、N-S 流程图、伪代码和计算机语言等，但描述的结果必须满足算法的五个特征。下面通过一个实例分别对这些算法描述工具进行介绍。

1. 用自然语言描述算法

自然语言就是人们日常使用的语言，可以是英语、汉语、日语等。用自然语言描述算法通俗易懂，但文字冗长，容易出现歧义。

【例 5-3】 用自然语言描述并计算 1000 以内的所有奇数和，即 1＋3＋5＋7＋…＋999 的和。

问题分析：本题是一个循环累加求和的问题，从第二项开始，每一项和前一项的值相差 2。可以设置两个变量：一个变量为 sum，用来保存每次累加的和，开始取值为 0；另一个变量为 i，用来保存要累加到 sum 的当前项，开始取值为 1。问题的求解过程就是不断地将 i 累加到和 sum 中，每加完一次，i 值加 2，直到 i 超过 999 为止，最后输出结果 sum。

用自然语言描述例 5-3 的算法如下。

s1：当前项 i 置初值 1，累加和 sum 置初值 0。

s2：判断 i 的值，若 i＜1000 则执行 s3；否则转 s5 执行；

s3：i 加到 sum；

s4：i 值加 2，转 s2 执行；

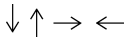
s5：输出结果 sum，算法结束。

用自然语言描述分支和多重循环等算法时，很不方便，容易出现错误。因此，除了比较简单的算法外，一般不用自然语言描述算法。

2. 传统流程图描述算法

流程图是用不同的几何图形表示不同的操作，用流程线表示算法执行的流向。美国国家标准化协会规定了一些常用的流程图符号及其功能，如表 5-2 所示。

表 5-2　常用的流程图符号及其功能

流程图符号	符 号 名 称	符 号 功 能
	起止框(圆弧形框)	表示算法的开始或结束
	处理框(矩形框)	表示一般的处理功能
	判断框(菱形框)	表示对给定的条件进行判断，决定执行后面的哪个操作
	输入输出框(平行四边形框)	表示算法的输入或输出
	流程线(指向线)	表示算法执行的流向
	连接点(圆圈)	表示将画在不同位置圈中标志相同的流程连在一起
	注释框	对流程图中某些框的操作做必要的补充说明，帮助阅读的人更好地理解流程图，不反映流程和操作

【例 5-4】 用传统流程图描述并计算 1000 以内的所有奇数和，即 1＋3＋5＋7＋…＋999 的和。

用传统流程图描述例 5-4 的算法如图 5-4 所示。

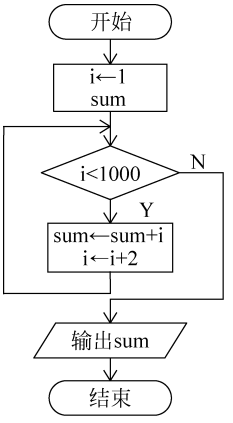


图 5-4　例 5-4 算法的传统流程图

用传统流程图描述算法形象直观、容易理解，但占用篇幅较大，流程指向随意，较大的流程图不易读懂，画流程图既费时又不方便。

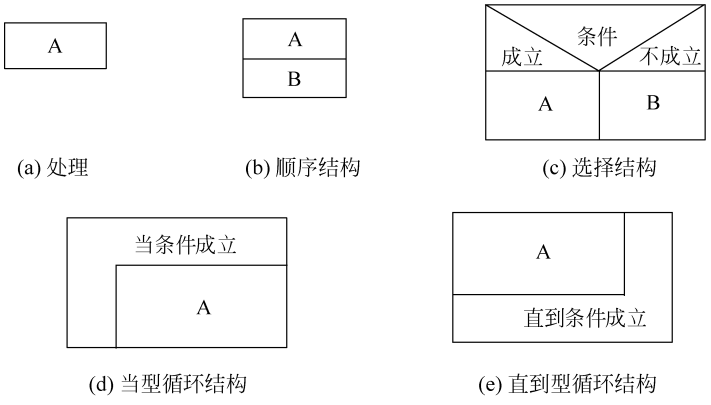
3. N-S 流程图描述算法

1973 年，美国学者 I. Nassi 和 B. Shneiderman 提出了一种新的流程图。在这种流程图中，完全去掉了带箭头的流程线，全部算法写在一个矩形框内，在该框内还可以包含其他的从属结构，这种流程图就是 N-S 流程图。如图 5-5 所示是 N-S 流程图的基本结构。

【例 5-5】 用 N-S 流程图描述并计算 1000 以内的所有奇数和，即 1＋3＋5＋7＋…＋999 的和。

用 N-S 流程图描述例 5-5 的算法如图 5-6 所示。

用传统流程图和 N-S 流程图表示算法直观易懂，但画起来比较费事，在设计一个算法时，可能需要反复修改，而修改流程图是比较麻烦的。



(a) 处理　(b) 顺序结构　(c) 选择结构

(d) 当型循环结构　(e) 直到型循环结构

图 5-5　N-S 流程图结构

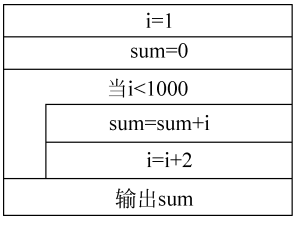


图 5-6　例 5-5 算法的 N-S 流程图

4. 伪代码描述算法(类语言)

伪代码是用介于自然语言和计算机语言之间的文字和符号来描述算法，它与一些高级编程语言(如 C 语言)类似，但是不需要遵循编写程序时的严格规则。伪代码用一种从顶到底、易于阅读的方式表示算法，结构清晰、代码简单。

【例 5-6】 用伪代码描述并计算 1000 以内的所有奇数和，即 1＋3＋5＋7＋…＋999 的和。

用伪代码描述例 5-6 的算法如下：

```
begin(算法开始)
    i = 1;
    sum = 0;
    当(i < 1000)
```

```
    {
        sum = sum + i;
        i = i + 2;
    }
    输出 sum
End (算法结束)
```

伪代码具有高级语言的一般语句结构，撇掉语言中的细节，以便把注意力主要集中在算法处理步骤本身的描述上。伪代码不拘泥于具体语言的语法结构，以灵活的形式表现被描述对象，可将注意力集中在处理步骤中。例如类 Pascal 语言、类 C 语言。在实际工程中描述算法常用伪代码形式，不仅直观、方便、有利于交流，而且在设计算法时能较好地考虑算法执行时的动态性。

5. 计算机高级语言

高级语言就是利用计算机程序设计语言直接表达算法，是可以在计算机上直接运行的源程序。高级语言描述的算法具有严格、准确的优点，但用于算法描述时，也有语言细节过多的弱点。

一个具体的算法使用计算机程序设计语言进行描述的过程，实际上就是程序设计的过程，其最终产物就是计算机程序。

【例 5-7】 用 C 语言计算 1000 以内的所有奇数和，即 1＋3＋5＋7＋…＋999 的和。

用 C 语言编程实现例 5-7 的算法如下：

```
void div()                          //算法的函数定义
{
    int i = 1,sum = 0;              //定义变量
    while(i < 1000)                 //判断 i 的值,若 i < 1000 则累加求和并使 i 的值增加 2
    {
        sum = sum + i;
        i = i + 2;
    }
printf(" % d\n",sum);               //输出结果

}
```

说明：如果需要运行此程序，只要编写主函数 main()，并在主函数中调用此自定义函数即可。

上面讲述的各种算法描述方法都是既有优点又有缺点。自然语言简单易懂，但不能清晰描述逻辑关系；流程图直观、清晰，但不方便数据处理；伪代码程序语言严谨，但语言细节描述过多。使用时可以根据实际情况进行选择。

5.1.4　算法的评价

解决某一问题存在多种算法，为了有效地进行解题，不仅需要保证算法正确，还要考虑算法的质量，通过深入理解问题本质及可能的求解技术，寻求优化算法以便更高效和方便地解决问题。一个优秀的算法一般有以下几个评价标准。

1. 正确性

正确性的基本要求是对输入数据能够得出满足要求的结果并且停止；对一切合法输入

都可以得到符合要求的解。如果一个算法对于某些输入数据要么不会停止，要么在停止前给出的不是预期的正确结果，它就是错误算法。

在很多应用领域中，算法的正确性至关重要。因为算法的错误而造成重大损失甚至灾难的例子并不鲜见。如20世纪60年代，由于飞行控制计算机程序中的一个错误导致发射到金星的美国太空船水手号不幸失事，损失重大。

2. 可读性

算法的可读性是指一个算法可供人们阅读的容易程度。一个算法可读性的好坏十分重要，如果一个算法比较抽象，难以理解，那么这个算法就不易交流和推广，对于修改、扩展、维护都十分不利。所以在写算法时，要尽量写得简明易懂。算法首先是用于人们的阅读与交流，其次才是为计算机执行。一个合格的算法应该具备良好的可读性，算法简单则程序结构也会简单，这便于程序调试。

3. 健壮性

健壮性就是指当输入的数据非法时，算法也会做出相应的判断，而不会因为输入的错误造成程序瘫痪。一个合格的算法应具有一定的容错处理能力，即输入非法数据或错误操作时给出提示，而不是中断程序执行；并可以返回表示错误性质的值，以便程序进行处理。

4. 时间复杂度

通常认为，通过统计算法中基本操作重复执行的次数，就可以近似得到算法的执行效率，用$O(n)$表示，称为时间复杂度。

影响算法时间复杂度的因素很多，主要包括以下几个方面。

(1) 问题规模的大小。例如，求10的阶乘和求100的阶乘所花费的时间是有明显差异的，显然求100的阶乘所花费的时间要比求10的阶乘花费的时间多得多。

(2) 源程序编译功能的强弱以及经过编译后产生的机器代码质量的优劣。

(3) 机器执行一条目标指令所需要的时间。这个因素与计算机系统的硬件息息相关，随着硬件技术的提高，硬件性能越来越好，执行一条目标指令所花费的时间也会相应地越来越少。

(4) 程序中语句的执行次数。这个因素与算法本身有直接关系，一个好的算法应该使程序中语句执行次数尽量少。

由于同一个算法使用不同的计算机语言实现的效率不相同，使用不同的编译器效率也不相同，运行于不同的计算机系统中效率也不相同，因此使用前三个因素来衡量一个算法的时间复杂度通常是不恰当的。通常使用第四个因素，即程序中的语言的执行次数来作为一个算法的时间复杂度的度量。

5. 空间复杂度

空间复杂度指的是算法程序在执行时所需要的存储空间。空间复杂度可以分为以下两个方面：

(1) 程序的保存所需要的存储空间资源，即程序的大小。

(2) 程序在执行过程中所需要消耗的存储空间资源，如中间变量等。

需要说明的是，时间复杂度和空间复杂度共同决定算法的效率，但时间效率和空间效率往往是矛盾的。在算法设计中很多情况下可以“以空间换时间，以时间换空间”。目前，随着计算机硬件的发展，因空间所限影响算法运行的情形较为少见，空间复杂度已经不再显得那

么重要。因而在设计算法时，应把降低算法的时间复杂度作为首要考虑的因素。

5.1.5　常见问题的算法描述

本节通过最值问题、排序问题和搜索问题这几类典型问题来说明计算机求解问题的思路及具体算法。

1. 最值问题及算法描述

(1) 什么是最值问题

每次期末考试结束后，学生除了关心自己的成绩外，是不是对班级或专业排名和最高成绩也很感兴趣呢？无论在日常生活中还是在科学研究中，最值问题都是普遍的应用类问题。例如在班级中找最高的、最矮的，某一科成绩最好的、最差的，在一组数据中找最大数、最小数等。

最值问题是指任意文件(或数据集合)按照指定的关键字寻找最大、最小数据的过程。如在 Excel 中，通过 MAX、MIN 函数可以求出一组数据的最大值、最小值，如图 5-7 所示。怎样设计算法来求解最值问题呢？下面先从简单问题入手。

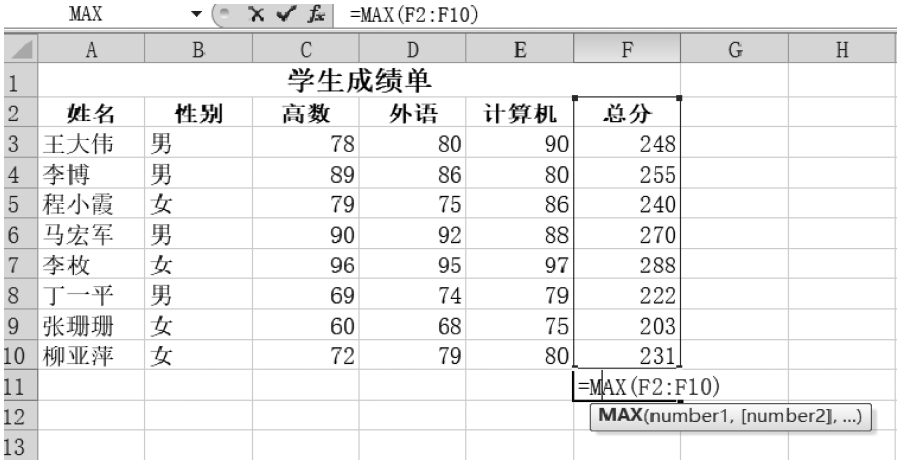


图 5-7　Excel 中求最大值问题

(2) 简单问题求最值

【例 5-8】 计算两个数的最大值(或最小值)。

问题描述：任意给定两个数，找出其中的最大者。

问题分析：定义两个变量 a 和 b 并分别赋予确定值再定义一个和 a、b 同性质的变量 max，用于存放最大值，如果 a>b，就将 a 赋予 max，否则就把 b 赋值给 max，这样最后 max 中存放的就是最大值。

算法 N-S 流程图如图 5-8 所示。

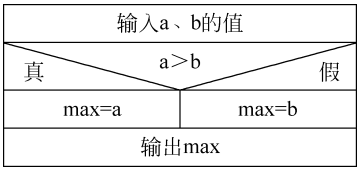


图 5-8　求两个数最大值的 N-S 流程图

这是最简单的两个数求最大值的方法，求最小值的方法与求最大值的方法类似。如果是更多数，如 4 个数求最值，算法如何修改？

【例 5-9】 计算 4 个数的最大值(或最小值)。

问题描述：任意给定 4 个数，找出其中的最大者。

问题分析：定义 4 个变量 a、b、c 和 d 并分别赋予确定值再定义一个和 a、b、c、d 同性质的变量 max，用于存放最大值。假设 a 最大，将 a 的值赋予 max 再比较 max 和 b 的值，如果 max＜b，就将 b 的值赋予 max，继续比较 c 和 max 以及 d 和 max，这样最后 max 中存放的就是最大值。

算法描述如下。

s1：输入 a、b、c、d 的值。

s2：假设 a 的值最大，则 max＝a。

s3：比较 max 和 b，若 max＜b，则 max＝a。

s4：比较 max 和 c，若 max＜c，则 max＝c。

s5：比较 max 和 d，若 max＜d，则 max＝d。

s6：输出最大数 max 的值，算法结束。

几个数求最值，通过比较就可以解决问题，如果是更多若干数据该如何处理？

（3）复杂问题求最值

【例 5-10】 求信息技术与计算思维导论课程，期末考试的最高分。

问题描述：依次输入班级的每一位学生的成绩，找出最高分，当所有人的成绩输入完毕，显示求得的最高分。

问题分析：定义两个变量 m 和 max，m 用来计数，初始值为 1 指向第一个学生，max 用来存放最大值，初始值为 0。先比较 max 和第一个学生的成绩，然后和第二个学生比较，依次类推。如果小于当前比较的学生，就把当前学生的成绩存入到 max 中，当 m 大于班级人数时结束循环，这样最后 max 中存放的就是最大值。

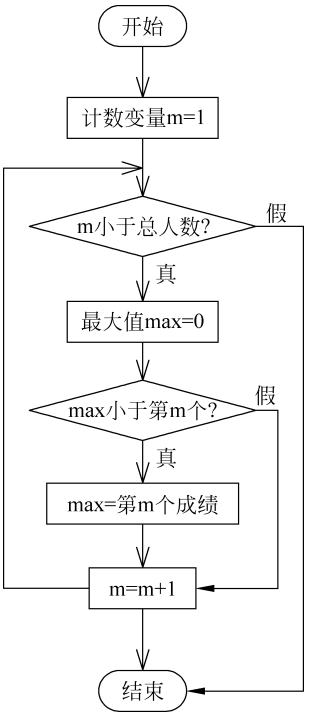


图 5-9 循环求最大值的传统流程图

算法传统流程图如图 5-9 所示。

2. 排序问题及算法描述

（1）什么是排序

排序是人们在处理数据时常见的问题，其本质是将杂乱无章的一组数据元素，通过一定的方法按照某种规则进行排列的过程。例如，学生成绩按降序排列名次、2019 年某流行歌曲排行榜、Excel 中的排序（如图 5-10 所示）等都属于排序问题。许多杂乱的问题的求解中都包含排序问题，排序和查找（下一个问题详细介绍）是计算机数据处理和问题求解中的重要操作，在计算机科学中占有重要地位。

（2）常见的排序算法

计算机编程中通常能有多种不同的算法可以用来完成给定的排序任务，如冒泡排序、选择排序、插入排序、快速排序、堆排序、归并排序等。在不同的情况下，每种算法都有自己的优缺点。在实际应用中，可以根据需要结合具体的问题选择合适的排序算法。本书主要介绍冒泡排序和选择排序。

① 冒泡排序。冒泡排序的基本思想：在每一轮（也称为趟）排序时比较相邻两个数据元素，如果次序不对，则将两个

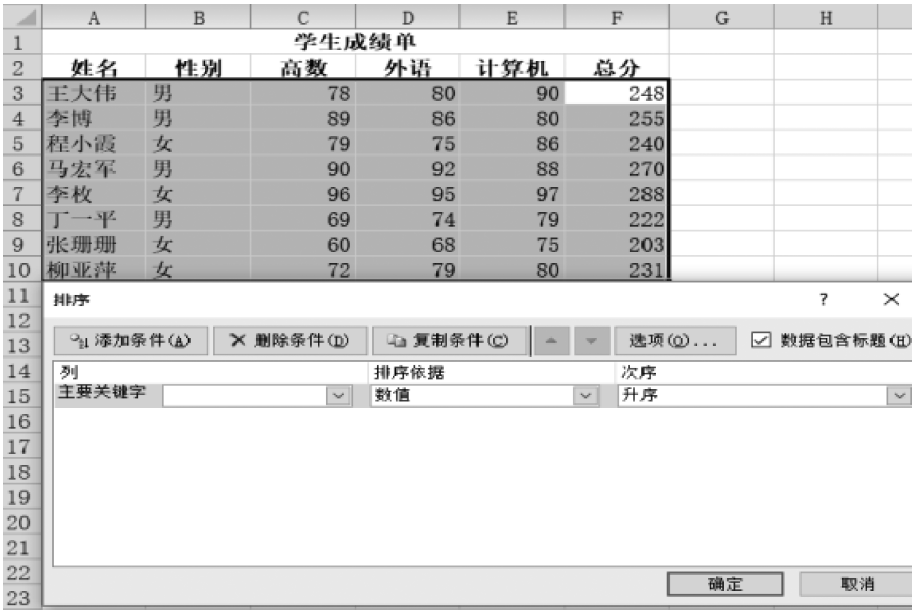


图 5-10　Excel 中的排序

元素位置互换，一趟比较结束时较小的数上浮，较大的数沉底。有 n 个数排序则进行 $n-1$ 趟上述操作。每一趟比较，都有较小的元素向上浮起，犹如冒泡。冒泡排序的具体思路和过程见 5.3 节程序案例。

② 选择排序。选择排序的基本思想：每趟从待排序的数据元素中找出最小(或最大)的元素，将其和第一个元素交换，然后，从剩下的 $N-1$ 个数中找出最小(或最大)的元素，将其和第二个元素交换，依此类推，直到所有的数据均有序为止。

【例 5-11】 输入 10 个数，利用“选择法”进行升序排序并输出。

选择排序算法如下。

s1：首先通过 $n-1$ 次比较，从 n 个数中找出最小的，将它与第一个数交换——第一趟选择排序，结果最小的数被安置在第一个元素位置上。

s2：再通过 $n-2$ 次比较，从剩余的 $n-1$ 个数中找出次小的数，将它与第二个数交换——第二趟选择排序。

s3：重复上述过程，共经过 $n-1$ 趟排序后，排序结束。

(3) 排序算法比较

选择排序和冒泡排序都是经常使用的排序算法，它们之间有以下共同点和不同点。

① 共同点

算法简单，操作方便；每一趟比较仅使一个数确定其所在数列中的位置。

② 不同点

选择排序每一趟比较中找最小元素，一趟比较结束后需要交换位置；冒泡排序在每一趟相互比较后，次序不对就交换位置。

③ 复杂度分析

冒泡排序是最简单的排序算法，运行效率低，时间复杂度在最好情况下，元素是有序的，共比较 $n-1$ 次，无须交换；在最坏情况下，元素逆序排列，共需做 $n(n-1)/2$ 次比较和交换；时间复杂度为 $O(n^2)$。

选择排序算法由一个双层循环控制，算法的时间复杂度由输入规模（也就是元素个数 n）决定，时间复杂度是 $O(n^2)$。

3. 搜索问题及算法描述

（1）什么是搜索问题

搜索问题是人们工作和生活中经常遇到的问题，例如，在学生库中找一个学生的信息，到图书馆查找资料，在网上使用搜索引擎搜寻信息等。Word 中的“查找”“替换”与“定位”也属于搜索问题。不管搜索对象如何，其所使用的搜索方法在思想上是相似的。同时，搜索问题也是许多问题的子问题，在许多复杂问题中都包含着对象搜索，因此对搜索问题求解算法的研究具有非常重要的作用。

搜索（searching）问题就是在给定的一个文件（或数据集合）中，按照指定的关键字寻找该关键字或者寻找包含该关键字的记录。搜索问题通常又称为查找或检索，结果有两种，第一种情况是找到了包含关键字值的记录，称为查找成功；第二种情况是没有找到包含关键字的记录，称为查找失败。常见搜索方法有顺序搜索、二分搜索、枚举法、广度优先搜索、深度优先搜索等。

（2）常见搜索算法

① 顺序查找

【例 5-12】 从键盘输入一个整数，在数组中查找是否存在该数，若存在输出其在数组中的位置。

问题分析：需要遍历整个数组，即顺序查找。设 flag＝0 表示数组中没有要找的数，flag＝1 表示数组中有要找的数。

下面以在数据表中 11，22，33，44，55，66，77，88 中搜索 44 为例，说明顺序查找的基本思想，如图 5-11 所示。

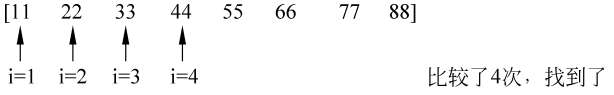


图 5-11 顺序搜索

图 5-12 给出了顺序查找算法的 N-S 流程图。

顺序搜索算法的优点是算法简单，对列表的结构无任何要求，但是查找效率低。对无序数据搜索时，需要对整个无序数据进行遍历，如果数据表规模很大，可以将表搜索分解成多个任务，并行计算，这就是并行顺序搜索，它是一种效率较高的顺序搜索方式。本节对并行搜索不再详述，大家可以查找相关资料学习。

② 二分查找

二分查找也叫折半查找，是一种在有序数据表中查找某一特定元素的搜索算法。它针对有序的数据表，充分利用了元素之间的次序关系，采用分块完成搜索任务，可以有效提高查找效率。其基本思想是不断将数据表对半分割，每次取中间元素和查找元素进行比较；

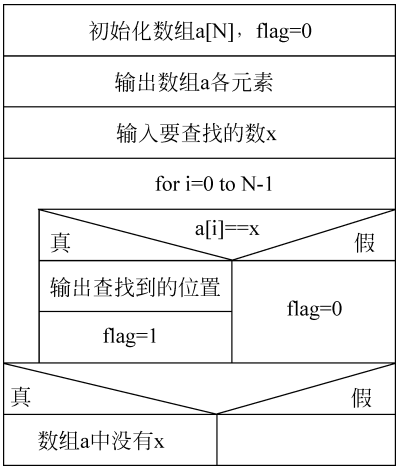


图 5-12　顺序查找算法的 N-S 流程图

如果匹配成功则查找成功，并给出查找元素的位置；如果匹配不成功，则继续进行二分查找；如果查找到最后一个元素仍然没有匹配成功，则查找元素不在列表中。

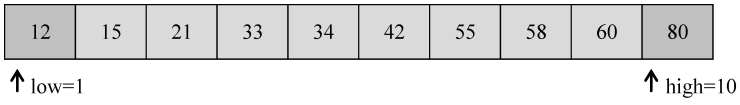
二分查找算法如下。

首先从序列的中间元素开始，如果中间元素正好是要查找的元素，则查找成功，搜索结束；如果要搜索的元素大于或者小于中间元素，则在数组大于或小于中间元素的那一半中查找，而且与开始一样从中间元素开始比较。这种搜索算法每一次比较都使搜索范围缩小一半。

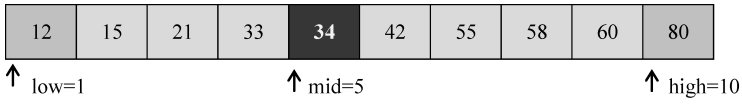
【例 5-13】　利用二分法查找“58”是否在有序表{12，15，21，33，34，42，55，58，60，80}中。

搜索算法如下。

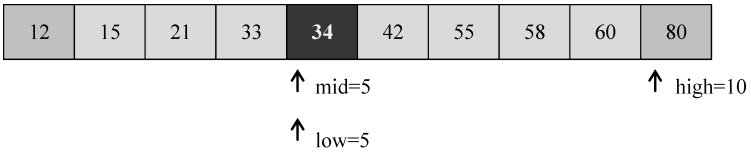
s1：第 1 次二分，设置左端点位置 low＝1，设置右端点位置 high＝10。



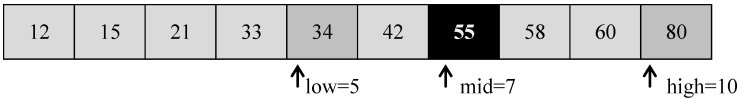
s2：计算中位数 mid＝(low＋high)/2＝(1＋10)/2＝5.5，中位数 5.5 取整为 5，将第 5 个元素(34)与查找元素(58)比较；由于 34＜58，因此查找元素在第 5 个元素之后(列表右边)。



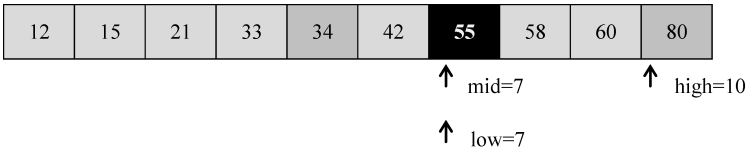
s3：第 2 次二分，将 low 位置移到第 5 个元素。



s4：计算中位数 mid=(low+high)/2=(5+10)/2=7.5，中位数 7.5 取整为 7，mid 为第 7 个元素(55)；将第 7 个元素(55)与查找元素(58)比较；由于 55<58，因此查找元素在第 7 个元素之后(列表右边)。



s5：第 3 次二分，将 low 位置移到第 7 个元素。



s6：计算中位数 mid=(low+high)/2=(7+10)/2=8.5，中位数 8.5 取整为 8，mid 为第 8 个元素(58)；将第 8 个元素(58)与查找元素(58)比较；由于 58 等于 58，元素匹配成功，输出找到的元素(58)和它的位置(8)。

图 5-13 所示给出了二分查找算法的传统流程图。

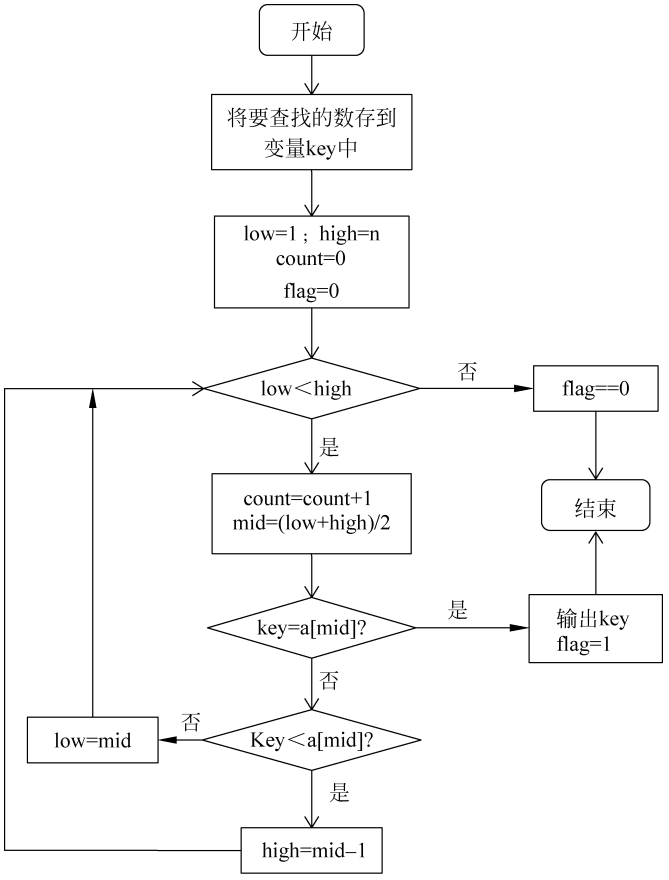


图 5-13　二分查找算法流程图

③ 穷举法

穷举法也称枚举法，基本思想是根据题目的部分条件确定答案的大致范围，并在此范围内对所有可能的情况逐一验证，直到全部情况验证完毕。若某种情况验证符合题目的全部条件，则为本问题的一个解；若全部情况验证后都不符合题目的全部条件，则本题无解。

【例 5-14】 我国古代数学家张丘建在《算经》中提出的数学问题：鸡翁一值钱五，鸡母一值钱三，鸡雏三值钱一。百钱买百鸡问题，即一百个铜钱买了一百只鸡，其中，公鸡一只5钱、母鸡一只3钱、小鸡一钱3只，问一百只鸡中公鸡、母鸡、小鸡各多少只？

问题分析：设公鸡、母鸡、小鸡的数量分别为 x、y、z，则，可以用下面方程组来求解：

$$\begin{cases} 5x+3y+\dfrac{1}{3}z=100 \\ x+y+z=100 \end{cases}$$

上面方程组中，变量个数大于算式的个数，无法直接求解。要想求得 x、y、z 的值，可以将所有可能的 x、y、z 的值带入两个算式中看是否同时满足要求，称这种方法为穷举法。穷举法最关键的是确定穷举的范围，然后在此范围内，对所有可能的情况进行逐一验证，直到所有情况验证完毕为止，若验证完毕未找到符合条件的值，则无解。通过分析，可以得到 x、y、z 的取值范围：x(0～20)、y(0～33)、z(0～100)。

算法设计：设公鸡为 x 只，母鸡为 y 只，小鸡为 z 只。如果全部买公鸡最多可以买 100/5＝20 只，即 x 的取值范围是 1～20；如果全部买母鸡最多可以买 100/3＝33 只，即 y 的取值范围是 1～33；如果全部买小鸡最多可以买 100×3＝300 只，但是题目规定买 100 只鸡，所以 z 的取值范围是 1～100。由此可见，此案例的约束条件为：x＋y＋z＝100 且 5x＋3y＋z/3＝100。

本题可以用三层嵌套的循环来实现，算法的 N-S 流程图如图 5-14 所示。

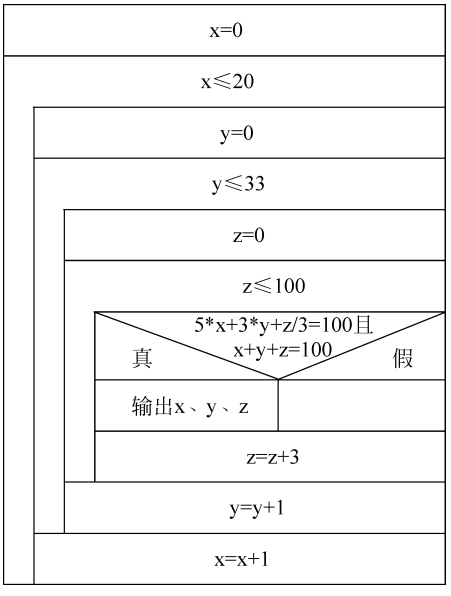


图 5-14　百钱买百鸡问题的 N-S 流程图

(3) 搜索算法比较

顺序搜索、二分搜索和穷举法是大家常见的比较简单的搜索方法,通过上面的例子大家可以看到以下共同点和不同点。

① 共同点

算法简单,操作方便,都是在数据列表中查找某个元素并确定这个元素在列表中的位置。

② 不同点

顺序搜索针对无序数据表进行遍历,二分查找待查找列表必须为有序表,顺序搜索和二分搜索找满足条件的一个值,穷举法是搜索出满足条件的多个值。

③ 复杂度分析

顺序搜索算法的优点是算法简单,对列表的结构无任何要求。对无序数据搜索,需要对整个无序数据表进行遍历,如果数据表规模很大,查找效率较低。算法时间复杂度为$O(n)$。二分查找比较次数少,查找速度快,平均性能好,但待查列表必须为有序表。算法在最坏情况下时间复杂度为$O(\log_2 n)$。穷举法搜索列举出问题的所有可能解,并用约束条件逐一判定,找出符合约束条件的所有解,算法的效率低。

5.2 程序和程序设计

如前所述,用计算机求解任何问题,首先必须给出解决问题的方法和步骤,也就是算法。再按照某种语法规则编写计算机可执行的程序,交给计算机去执行,这个过程就是程序设计的过程。编写程序所使用的语法规则的集合就是程序设计语言。程序设计语言可以是机器语言、汇编语言,也可以是高级语言。本节首先简要介绍计算机程序的基本概念和程序设计语言的演变过程,之后说明语言处理程序及程序设计的详细步骤,最后以 C 语言为例,阐述高级语言程序设计的基本方法以及结构控制。

5.2.1 计算机程序的概念

计算机程序是指用某种计算机程序设计语言编写的解决某个问题或完成某项任务的指令序列。如果用一个等式来表示程序的概念,可表示为:

程序=算法+数据结构+程序设计方法+语言工具和环境

其中算法即操作步骤,解决“做什么”和“怎样做”的问题,是程序的灵魂;数据结构是对数据的描述,即程序中要指定的数据类型和数据的组织形式;程序设计方法即编程所采用的适合的方法;语言工具和环境是编写程序所采用的编程环境及编程语言。

例如,从键盘输入两个数,求它们的和,并输出。

用 C 语言可编写出以下程序代码:

```
#include <stdio.h>                    /* 将文件 stdio.h 包含到本程序中 */
void main()
{
    int a, b, sum;                    /* 定义 3 个整型变量,用于存放输入的两个数及其和 */
```

```
    scanf("%d, %d", &a, &b);          /* 从键盘输入两个数,分别存放到变量 a 和变量 b 中 */
    sum = a + b;                      /* 对变量 a 和变量 b 求和,结果存放到变量 sum 中 */
    printf("%d\n", sum);              /* 输出和 sum,后换行 */
}
```

程序运行结果：

```
3,6↙(说明：↙表示从键盘上按 Enter 键)
9
```

该程序的第一行 main()说明这是一个主函数，它指出一个程序的开始地址。

程序中的“{”和“}”分别标识程序段落的开始和结束。程序中用/* …… */表示的内容是注释部分，目的是增加程序的可读性，对程序的编译和运行并不起作用。

通常，命令型程序以一组说明语句开始，它们描述程序要操作的数据，在此之后是命令型语句，它们描述要执行的算法，注释语句按需要分布在程序各处。

5.2.2　程序设计语言

程序设计语言即编程语言(programming language)，是用于书写计算机程序的语言，它是人与计算机进行交流的工具。要了解程序设计语言先要了解指令和程序的概念。指令是指示计算机执行某种操作的命令，程序是一系列按一定顺序排列的指令，程序设计语言是用于编写计算机程序的语言。执行程序的过程就是计算机的工作过程。从发展历程来看，程序设计语言可以分为以下四大阶段，每一个阶段都大大提高了程序设计的效率。

1. 第一代程序设计语言，机器语言——面向机器的语言

机器语言是最底层的计算机语言。在用机器语言编写的程序中，每一条指令都是“二进制”(0 和 1)形式的指令代码，计算机硬件可以直接识别，机器语言又称为二进制语言。例如，要计算 15＋10，用机器语言编写的程序如下：

```
10110000 00001111                 把 15 放入累加器 A 中
00101100 00001010                 10 与累加器 A 中的值相加,结果仍放入 A 中
11110100                          结束,停机
```

由于机器语言程序是直接针对计算机硬件编写的，因此，它的执行效率比较高。对于不同的计算机硬件(主要是 CPU)，其机器语言是不相通的，使用一种计算机的机器指令编写的程序，不能在另一种计算机上执行。使用机器语言编写程序，编程人员要熟记所用计算机的全部指令代码和代码的含义，因而，用机器语言编写程序的难度较大，容易出错，而且程序的直观性较差。除了计算机生产厂家的专业人员外，绝大多数的程序员已经不再使用机器语言。

2. 第二代程序设计语言，汇编语言——面向机器的语言

为了便于理解与记忆，人们采用能“帮助记忆”的英文缩写符号(称为指令助记符)来代替机器语言中的指令代码，这种语言称为汇编语言或符号语言。例如，要计算 15＋10，用汇编语言编写的程序如下：

```
MOV  A,15                         把 15 放入累加器 A 中
ADD  A,10                         10 与累加器 A 中的值相加,结果仍放入 A 中
HLT                               结束,停机
```

由于汇编语言采用了助记符，因此，它比机器语言直观，更容易理解和记忆，但是，计算机不能直接识别用汇编语言编写的程序。此外，汇编语言也依赖于计算机硬件，因此，程序的可移植性较差。

汇编语言不像其他大多数的程序设计语言一样被广泛用于程序设计。它通常被应用在底层、硬件操作和要求高的程序优化的场合。驱动程序、嵌入式操作系统和实时运行程序都需要汇编语言。

3. 第三代程序设计语言，高级语言——面向过程、面向对象的语言

机器语言和汇编语言是低级语言，两种语言都是面向机器的。高级语言基本脱离了机器的硬件系统，具有良好的通用性和可移植性。高级语言的指令接近自然语言和数学公式，编写的程序称为源程序，高级语言编写的程序计算机也不能直接执行，需要翻译成目标程序（二进制程序）才能执行。高级语言并不是特指某一种具体的语言，而是包括很多编程语言，如流行的 Java、C、C++、C＃、Pascal 等，这些语言的语法、指令格式都不相同。例如，要计算 15＋10，用 C 语言编写的程序如下：

```
int a;                          定义变量 a
a = 15 + 10;                    将 15 和 10 相加,结果存放到变量 a 中
printf(" % d\n", a);            输出结果
```

高级语言有面向过程的语言和面向对象的语言两种。面向过程的语言是以过程或函数为基础的，在编写程序时需要具体指定每一个过程的细节，这种语言对底层硬件、内存等操作比较方便，C 语言就是面向过程的语言。面向对象的语言是一切操作都以对象为基础，它是由面向过程的语言发展而来的，但正是它的这个特性使得面向对象的语言对底层的操作不是很方便，如 Java 语言。可以说面向过程就是什么事都自己做，面向对象就是什么事都指挥对象去做。当程序规模小时用面向过程的语言还可以，当程序规模比较大时用面向对象的语言比较方便。

4. 第四代程序设计语言（Fourth Generation Language，以下简称 4GL）

前几代语言都需要编程指出怎么做（运行步骤），4GL 在一定程度上只需要说明做什么（目的），不需要写出怎么做的过程，因此可大大提高软件生产率，成为面向数据库应用开发的主流工具，如 Oracle 应用开发环境、SQL Windows、Power Builder 等。虽然 4GL 具有很多优点，但也存在严重不足：

（1）4GL 虽然功能强大，但在其整体能力上却与 3GL 有一定的差距。

（2）由于缺乏统一的工业标准，4GL 产品花样繁多，用户界面差异很大，与具体的机器联系紧密，语言的独立性较差，影响了应用软件的移植与推广。

（3）4GL 主要面向基于数据库应用的领域，不适合科学计算、高速的实时系统和系统软件开发。

5.2.3 常用计算机语言介绍

与机器语言和汇编语言不同，高级语言是面向用户的，它包括很多种编程语言。目前，高级语言种类已达数百种，下面介绍几种常用高级语言。

1. FORTRAN 语言

FORTRAN(Formula Translation)语言意为“公式翻译”,它是为科学、工程问题或企事业管理中的那些能够用数学公式表达的问题而设计的,其数值计算的功能较强。

FORTRAN 语言是世界上第一个被正式推广使用的高级语言。它是 1954 年被提出来的,1956 年开始正式使用,至今已有六十多年的历史,但仍经久不衰,它始终是数值计算领域所使用的主要语言。

2. C 语言、C++语言与 Visual C++

C 语言是 1973 年由美国贝尔实验室研制成功的。它最初是作为 UNIX 操作系统的主要语言开发的,并在发展过程中做了多次改进。1977 年出现了不依赖具体机器的 C 语言编译文本,使 C 语言移植到其他机器上的工作大大简化,也推动了 UNIX 操作系统在各种机器上的应用。随着 UNIX 操作系统的广泛使用,C 语言迅速得到推广。1978 年以后,C 语言成功地应用在大、中、小、微型计算机上,成为独立于 UNIX 操作系统的通用程序语言。C 语言表达简洁,控制结构和数据结构完备,具有丰富的运算符和数据类型,可移植性强,编译质量高。作为高级语言,C 语言还具有低级语言的许多功能,可以直接对硬件操作,例如对内存地址的操作、位的操作等,因此用 C 语言编写的程序可以在不同体系结构的计算机运行。用 C 语言不仅能编写出效率高的应用软件,也适用于编写操作系统、编译程序等系统软件。C 语言已成为应用最广泛的通用程序设计语言之一。

C++语言是在 C 语言的基础上发展起来的面向对象的通用程序设计语言。C++语言于 20 世纪 80 年代由贝尔实验室设计并实现。C++语言是对 C 语言的扩充,扩充的内容绝大部分来自其他著名语言(如 Simula、ALGOL68、Ada 等)的最佳特性。它既支持传统的面向过程的程序设计,又支持面向对象的程序设计,而且运行性能高。C++语言与 C 语言完全兼容,用 C 语言编写的程序能方便地在 C++环境中重用。因此 C++语言迅速流行,成为当今面向对象的程序设计的主流语言之一。就面向过程编程而言,C++和 C 几乎是一样的。由于 C 语言相对简单,各大高校都将 C 语言作为一门重要的课程,学习了 C 语言,再去学习 C++,就会达到事半功倍的效果。

Visual C++是微软公司的 Visual Studio 开发工具箱中的一个 C++程序开发包。Visual Studio 提供了一整套开发 Internet 和 Windows 应用程序的工具,包括 Visual C++、Visual Basic、Visual FoxPro 以及其他辅助工具,如代码管理工具 Visual Source Safe 和联机帮助系统 MSDN。Visual C++包中除包括 C++编译器外,还包括所有的库、例子和为创建 Windows 应用程序所需要的文档。从最早期的 1.0 版本发展到 6.0 版本,Visual C++有了很大的变化,在界面、功能、库支持方面都有许多增强。Visual C++6.0 版本在编译器、MFC 类库、编辑器以及联机帮助系统等方面都比以前的版本有了较大改进。Visual C++是一种大型语言,其功能、概念和语法规定都比较复杂,要深入掌握它需要花较多的时间,尤其需要有较丰富的实践经验,一般使用 Visual C++编程的主要是软件专业人员。

3. Java 语言

Java 语言是由 Sun 公司开发的一种新型的跨平台分布式程序设计语言。Java 语言以其简单、安全、可移植、面向对象、多线程处理和动态等特征引起世界范围的广泛关注。

Java 是一门面向对象编程语言,不仅吸收了 C++语言的各种优点,还摒弃了 C++中难以理解的多继承、指针等概念,因此 Java 语言具有功能强大和简单易用两个特征。Java 语言

作为静态面向对象编程语言的代表，极好地实现了面向对象理论，允许程序员以优雅的思维方式进行复杂的编程。

狭义上 Java 是一种编程语言，它既可作为一种通用的编程语言，又可用来创建一种可通过网络发布的、动态执行的二进制“内容”。广义上 Java 不仅是一种编程语言，还包括一个客户机/服务器模式下的开发和执行环境，具有完全的平台无关性。它基于 C++，同时又抛弃了 C++中的非面向对象和容易引起软件错误的地方，因此是一种简单而稳定的语言。

4. BASIC 语言与 Visual Basic 语言

BASIC(Beginners' All-purpose Symbolic Instruction Code)意思就是“初学者通用符号指令代码”，是一种设计给初学者使用的程序设计语言。BASIC 语言最初是在 20 世纪 60 年代初期研制的一种交互式语言，它是一种应用广泛的计算机高级语言，特点是易学易用，人机对话能力强，非常适合于初学者。

1991 年 4 月，Visual Basic 1.0 for Windows 版本发布，许多专家把 VB 的出现当作是软件开发史上的一个具有划时代意义的事件。Visual Basic 意为“可视的 BASIC”，即图形界面的 BASIC。它是用于 Windows 系统开发的应用软件，可以设计出具有良好用户界面的应用程序。此后，微软公司又相继推出了 Visual Basic 2.0 到 Visual Basic 6.0 多种版本，之后又推出 Visual Basic 6.0 中文版。VB 6.0 作为 Microsoft Visual Studio 6.0 工具套件之一，提供了图形化、ODBC 实现整合资料浏览工具平台，提供了与 Oracle 和 SQL Server 的数据库链接工具。VB 6.0 的 Web 开发特性可以使得开发人员以更方便、组件式的方法，开发各种 HTML 和动态 HTML 的应用程序。这些新特性使得 VB 6.0 成为建立可扩展的企业应用开发平台的理想选择。2001 年，VB. NET 发布，由于使用了新的核心和特性，很多 VB 的程序员都要改写程序。2005 年 11 月 7 日，VB. NET 2005(v8.0)发布，它可以直接设计出 Windows XP 风格的界面，但是其编写的程序占用内存较多。2010 年 4 月，VB. NET 2010(v10.0)发布。

5.2.4 语言处理程序

除了机器语言外，其他使用任何软件语言书写的程序都不能直接在计算机上执行，都需要对它们进行适当的处理。语言处理系统的作用是把用软件语言书写的各种程序处理成可在计算机上执行的程序，或最终的计算结果，或其他中间形式。

语言处理系统因所处理的语言即处理方法和处理的过程不同而不同。但对任何一种语言来说，通常都包含一个翻译程序，这种翻译程序也称为语言处理程序。被翻译的汇编语言或高级语言程序称为源程序，翻译后生成的低级语言程序称为目标程序。

语言处理程序是将用程序设计语言编写的源程序转换成机器语言的形式，以便计算机能够运行。翻译程序除了要完成语言间的转换外，还要进行语法、语义等方面的检查。按照不同的源语言、目标语言和翻译处理方法，可把翻译程序分成若干种类。从汇编语言到机器语言的翻译程序称为汇编程序，从高级语言到机器语言或汇编语言的翻译程序称为编译程序。按源程序中指令或语句的动态执行顺序，逐条翻译并立即解释执行相应功能的处理程序称为解释程序。

1. 汇编程序

由于汇编语言的指令与机器语言的指令大体上保持一一对应的关系，汇编算法采用的

基本策略是简单的。汇编过程就是对汇编指令逐行进行处理，翻译成计算机可理解的机器指令。通常采用两遍扫描源程序的算法：第一遍扫描源程序根据符号的定义和使用，收集符号的有关信息到符号表中；第二遍利用第一遍收集的符号信息，将源程序中的符号化指令逐条翻译为相应的机器指令。处理的步骤如下：

（1）指令助记符操作码翻译成相应的机器操作码。

（2）把符号操作码翻译成相应的地址码。

（3）把操作码和操作数构造成机器指令。

2. 编译程序

编译方式的工作过程是：将用高级语言编写的源程序输入计算机，然后调用编译程序把源程序整个翻译成机器指令代码组成的目标程序，再经过连接程序连接后形成可执行程序，最后执行得到结果。采用编译方式执行一般效率高，高级语言大多采用编译方式。目前微型机高级语言 FORTRAN、Pascal、C 等都属于编译方式，BASIC 语言有解释方式的，也有编译方式的。图 5-15 是编译方式的工作过程示意图。

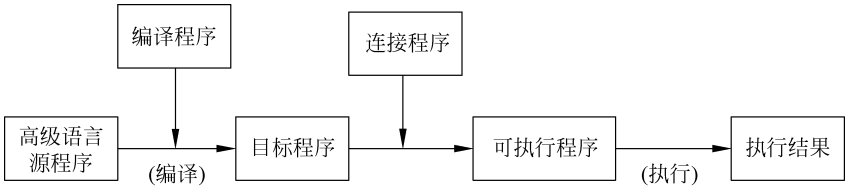


图 5-15　编译方式的工作过程

编译程序对源程序进行翻译的方法相当于“笔译”。在编译程序执行的过程中，要对源程序扫描一遍或几遍，最终生成一个可在具体计算机上执行的目标程序。由于源程序中的每个语句与目标程序中的指令通常具有一对多的对应关系，所以编译程序的实现算法较为复杂。但通过编译程序的处理可以一次性地产生高效运行的目标程序，并把它保存在磁盘上，以备多次执行。因此，编译程序更适合于翻译规模大、结构复杂、运行时间长的大型应用程序。

编译程序多遍扫描并分析源程序，然后将其转换成程序。通常编译程序在初始处理阶段建立符号表、常数表和中间语言程序等数据结构，以便在分析和综合时引用和加工。源程序的分析是经过词法分析、语法分析和语义分析三个步骤完成的，分析过程中发现有错误，给出错误提示。目标程序的综合包括存储分配、代码优化、代码生成等步骤，目的是为程序中的常数、变量、数组等数据结构分配存储空间。

随着高级语言在形式化、结构化、智能化和可视化等方面的发展，编译程序也随之向自动程序设计和可视化程序设计的方向发展。这样，可为用户提供更加理想的程序设计工具。

3. 解释程序

解释程序是对源程序的翻译方法，相当于两种自然语言间的“口译”。解释方式的工作过程是边解释边执行，即把高级语言源程序输入计算机后，解释程序对它进行逐句扫描、逐句翻译、逐句执行。这种对源程序逐句执行的工作方式显然便于实现人机对话。解释程序结构简单、易于实现，但效率较低。图 5-16 是解释方式工作过程示意图。

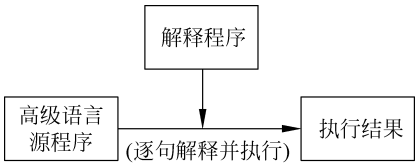


图 5-16　解释方式的工作过程

5.2.5 程序设计的步骤

计算机程序设计是根据特定问题，用计算机语言设计、编制、调试和运行程序的过程。程序设计的步骤一般要经过问题分析和建立模型、算法设计、编写程序、调试运行和编写程序文档多个步骤。

1. 问题分析和建立模型

问题分析是程序设计的第一步，其任务是对于给定的一个问题，要清楚这个问题要完成什么样的功能，哪些条件是已知的，哪些是要求解的，已知条件和要求的结果之间存在什么样的关系等，最终建立数学模型并确定数据的组织形式以及解决问题的方案。

2. 算法设计

在明确了给定的问题后，要确定采用的具体算法，并选择一种合适的算法描述工具进行描述，这一步是程序设计的关键，决定程序执行的效率。

3. 编写程序

编写程序就是根据确定的数据结构和算法，按照某种程序设计语言的语法、语义和规则实现算法的每一个步骤。最终的程序要保证语法和逻辑都是正确的，才能得到正确的结果，这一步需要上机验证。

4. 调试运行

将编写的源程序输入到计算机调试和运行，以便找出和修改程序中的语法错误、运行错误、逻辑错误等，测试程序是否达到预期结果。

5. 编写程序文档

许多程序是提供给别人使用的，如同正式的产品应当提供产品说明书一样，正式提供给用户使用的程序，必须向用户提供程序说明书。内容应包括：程序名称、程序功能、运行环境、程序的装入和启动、需要输入的数据，以及使用注意事项等。

【例 5-15】 已知圆半径为 5，求圆面积。

(1) 问题分析。根据半径求圆面积公式，可以借助数学公式完成。

(2) 算法设计。

首先确定数据结构与数学模型。

数据结构：本问题可以设计一个变量空间 r 存储半径的值，一个变量空间 s 存储面积的值。

数学模型：使用求面积公式 $s=\pi r^2$。

设计算法：求圆面积算法描述如下。

s1：输入半径 r。

s2：依据圆面积公式求圆面积 s。

s3：输出圆面积 s。

(3) 编写程序

用 C 语言编写的程序如下：

```
#include <stdio.h>                    /*将文件 stdio.h 包含到本程序中*/
#define PI 3.14                       /*宏定义*/
void main()
```

```
{
    int r;                          /* 定义整型变量 r,用于存放输入的半径 */
    float s;                        /* 定义单精度型变量 s,用于存放圆的面积 */
    printf("输入变量 r 的值:");
    scanf("%d", &r);                /* 从键盘输入数据,存放到变量 r 中 */
    s = PI * r * r;                 /* 使用求面积公式得圆面积,结果存放到变量 s 中 */
    printf("%f\n", s);              /* 输出面积 s,后换行 */
}
```

5.2.6　程序设计的控制结构

1. 结构化程序设计

结构化程序设计的思想是把一个程序分成若干互相独立的模块，这样在设计程序时，只要各个模块设计正确，就可以保证整个程序也肯定设计正确。如果将来要对某个模块进行修改，也不会引起对整个程序的修改，因此，结构化程序设计的思想越来越深入人心。

结构化程序设计的基本要点如下。

(1) 自上而下，逐步细化

逐步细化总是和自上而下结合使用，将复杂的大问题逐步分解成多个简单的小问题，再将问题求解逐步具体化。

(2) 模块化设计

模块化是结构化程序设计的重要原则，一个程序是由一个主控模块和若干子模块组成的，主控模块用来完成某些公共操作及功能选择，而子模块用来完成某项特定功能，如图 5-17 所示。

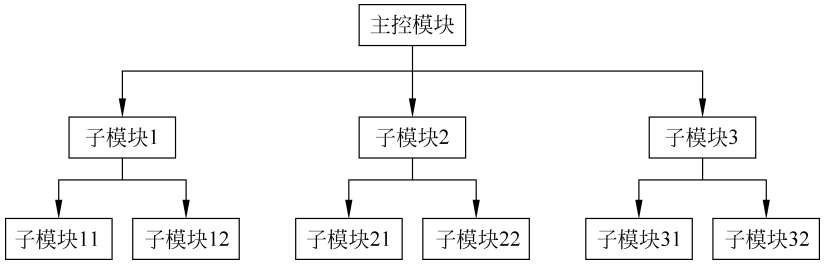


图 5-17　结构化程序设计思想

(3) 结构化编码

在设计好一个结构化的算法之后，还需进行结构化编码，将已经设计好的算法用某种具体的计算机语言来表示，编写出能在计算机上进行编译与运行的程序。

结构化程序设计把程序的结构规定为顺序结构、选择结构和循环结构三种基本结构，限制使用 GOTO 语句。因此，系统中每个模块的功能实现应由上述三种基本结构组成。模块的划分应当遵循以下三条基本要求。

① 模块的功能在逻辑上尽可能单一化、明确化，最好做到一一对应，这称为模块的凝聚性。

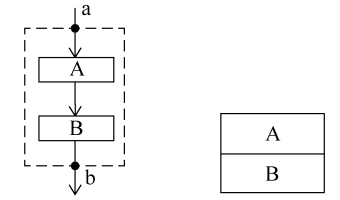
② 模块之间的联系及相互影响尽可能少，对于必需的联系都应当加以明确的说明，这称为模块的耦合性。

③ 模块的规模应当足够小，以便编程和调试易于进行。

尽管结构化程序设计是一种应用非常广泛的程序设计方法，但随着计算机技术飞速发展以及计算机应用的日益广泛，需要研制的系统愈来愈复杂，这种方法也随之暴露出一些不足之处。首先，结构化程序设计是面向过程的程序设计方法，它把数据和对数据的处理过程类型分离为相互独立的实体，它的程序结构是“数据结构＋算法”，若要修改某个数据结构，就需要改动涉及此数据结构的所有处理模块，所以当应用程序比较复杂时，容易出错，难以维护。其次，结构化程序设计方法仍然存在与人的思维方式不协调的地方，所以很难自然、准确地反映真实世界。

2. 程序设计的控制结构

下面通过几个连贯性实例来讲解顺序结构、选择结构和循环结构这三种基本结构。



(a) 传统流程图　　(b) N-S流程图

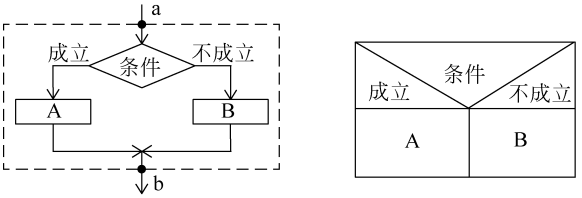
图 5-18　顺序结构流程图

(1) 顺序结构

顺序结构是最简单的算法结构，只要按照解决问题的顺序写出相应的语句即可，它的传统流程图和 N-S 流程图如图 5-18(a)和图 5-18(b)所示。顺序结构的执行顺序是自上而下，依次执行，即执行完 A 操作接着执行 B 操作。

(2) 选择结构

选择结构又称为分支结构，它的传统流程图和 N-S 流程图如图 5-19(a)和图 5-19(b)所示。选择结构的执行顺序是给定的条件成立时执行 A 操作，不成立时执行 B 操作。



(a) 传统流程图　　(b) N-S流程图

图 5-19　选择结构流程图

(3) 循环结构

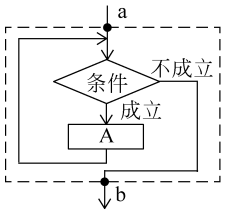
循环结构又称为重复结构，就是在达到指定条件前，反复执行某一部分操作。循环结构分两类：当型循环(while)结构，传统流程图和 N-S 流程图如图 5-20(a)和图 5-20(b)所示，当给定的条件成立时，反复执行 A 操作，直到条件不成立为止；直到型循环(until)结构，传统流程图和 N-S 流程图如图 5-21(a)和图 5-21(b)所示，反复执行 A 操作，直到给定的条件成立为止。

顺序、选择和循环三种基本结构是结构化程序设计的三大基本结构，它们具有以下共同的特点：

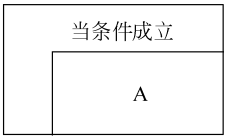
(1) 只有一个入口 a 和一个出口 b。

(2) 结构内的每一部分都有机会被执行。

(3) 结构内不存在“死循环”(无终止的循环)。

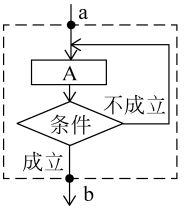


(a) 传统流程图

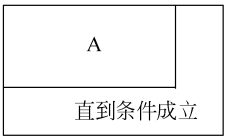


(b) N-S流程图

图 5-20　当型循环结构流程图



(a) 传统流程图



(b) N-S流程图

图 5-21　直到型循环结构流程图

3. 数据成分和程序基本结构的实现(举一个例子)

(1) 数据成分

数据是程序操作的对象,具有名称、类型、作用域等特征。高级语言在使用数据前要对其特征加以说明。数据名称由用户通过标识符命名,数据类型说明数据需要占用存储单元的多少、存放方式以及运算合法性,作用域说明数据可以使用的范围。

以 C 语言为例,其数据类型可分为基本类型、构造类型、指针类型、空类型和用户自定类型。其中,基本类型包括整型、实型、字符型、枚举型;构造类型包括数组类型、结构体类型等。例如,在程序中数值型数据是经常要使用的数据,其值是一个数,数值有范围和精度要求,若它的取值范围是一个整型数,则在程序的数据成分中可写成:

```
int x;
```

它定义了 x 为整型变量。

(2) 三种基本结构的实现

为了实现上述三种基本结构的功能,在程序设计语言中都有相应的语句与它们的功能对应,下面以 C 语言为例,介绍与上述三种基本结构对应的部分语句。

① 赋值语句。赋值语句是一种对应于顺序结构的语句,其表示形式如下:

```
变量名 = 表达式;
```

赋值运算符有两类:简单赋值运算符和复合赋值运算符,这些运算符的功能及运算规则如表 5-3 所示。

表 5-3　赋值运算符

运　算　符	示　　例		功　能
=	a=3		将数值 3 赋值给变量 a
+=	a+=b	等价于 a=a+b	将变量 a 与变量 b 相加的和重新赋值给变量 a
-=	a-=b	等价于 a=a-b	将变量 a 与变量 b 相减的差值重新赋值给变量 a
*=	a*=b	等价于 a=a*b	将变量 a 与变量 b 相乘的积重新赋值给变量 a
/=	a/=b	等价于 a=a/b	将变量 a 与变量 b 相除的商重新赋值给变量 a
%=	a%=b	等价于 a=a%b	将变量 a 与变量 b 相除的余数重新赋值给变量 a

② if 语句。if 语句是一种对应于选择结构的语句,其中双分支语句的表示形式如下:

```
if(表达式)
```

```
    语句 1;
else
    语句 2;
```

其中，语句 1 和语句 2 可以是一条语句，如果是多条语句，应写成复合语句的形式。它的功能是当条件表达式的值为真时，执行语句 1；否则执行语句 2。

双分支 if 语句的执行过程如图 5-22 所示。

③ while 语句。由 while 语句构成的循环称为“当型”循环，其一般形式为：

```
while(表达式)
{
    语句序列;
}
```

其中，表达式是循环条件，可以是任意合法的表达式。语句序列构成循环体。当语句序列只有一条语句时，其外侧的一对“{}”可以省略。执行过程如下：

s1：首先计算表达式的值。

s2：其次若表达式的值为真，执行循环体语句，然后返回步骤 s1。

s3：最后若表达式的值为假，退出循环，接着执行循环结构后面的语句。

while 循环结构执行的流程如图 5-23 所示。

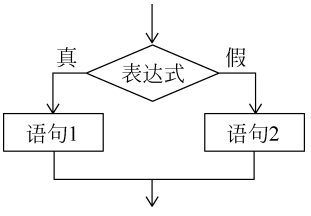


图 5-22　双分支 if 语句的执行过程

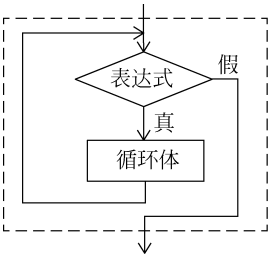


图 5-23　while 循环结构流程

④ for 语句。for 语句是三种循环结构中功能最强，使用最灵活、最广泛的一种循环语句，既可以用于循环次数已知的情况，又可用于循环次数未知的情况，其一般形式为：

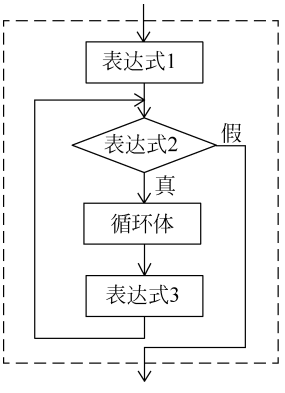


图 5-24　for 循环结构流程图

```
for(表达式 1;表达式 2;表达式 3)
{
    语句序列;
}
```

执行过程如下：

s1：计算表达式 1 的值。

s2：计算表达式 2 的值，若其值为真，执行循环体语句，然后执行步骤 s3；若其值为假，退出循环，继续执行循环结构后面的语句。

s3：计算表达式 3 的值，然后回到步骤 s2 继续执行。

for 循环结构执行的流程如图 5-24 所示。

为了便于理解程序的基本组成成分，以下给出两个 C 语

言程序的例子。

【例 5-16】 编程实现输入两个数，按由大到小的顺序输出。

程序代码如下：

```
#include <stdio.h>
void main()
{
  float x,y,t;
  scanf("%f %f",&x,&y);
  if (x<y)
  { t=x;x=y;y=t;}
  printf("%5.2f,%5.2f",x,y);
}
```

【例 5-17】 编程计算 1＋2＋3＋…＋100 的和。

程序代码如下：

```
#include <stdio.h>
void main()
{
  int i,sum=0;
  for( i=1;i<= 100;i++)
      sum=sum+i;
  printf ("%d",sum);
}
```

5.3 程 序 案 例

排序是人们在处理数据时常见的问题，其本质是将杂乱无章的一组数据元素，通过一定的方法按照某种规则进行排列的过程。计算机编程中通常能有多种不同的算法可以用来完成给定的排序任务。本节通过冒泡排序这个典型案例来分析其算法、编程、结构、运行过程和结果。

5.3.1 算法设计

冒泡排序的基本思想：在每一轮(也称为趟)排序时比较相邻两个数据元素，如果次序不对，则将两个元素位置互换，一趟比较结束时较小的数上浮，较大的数沉底。有 n 个数排序则进行 $n-1$ 趟上述操作。每一趟比较，都有较小的元素向上浮起，犹如冒泡。

【例 5-18】 输入 10 个数，利用“冒泡法”进行升序排序并输出。

下面以 6 个数 9,3,2,8,6,4 为例，说明“冒泡法”排序的基本思想。

冒泡法排序算法如下。

s1：比较第一个数与第二个数，若为逆序，则交换；然后比较第二个数与第三个数；依次类推，直至第 $n-1$ 个数和第 n 个数比较为止。第一趟冒泡排序，结果最大的数被安置在最后一个元素位置上。如图 5-11 所示是第一趟比较的过程和结果。其中，带“ ”的数是要比较的两个数，带“ ”的数是已排好序的数。

第一趟比较的过程和结果如图 5-25 所示。

第1次	第2次	第3次	第4次	第5次	结果
9	3	3	3	3	3
3	9	2	2	2	2
2	2	9	8	8	8
8	8	8	9	6	6
6	6	6	6	9	4
4	4	4	4	4	9

图 5-25　第一趟比较的过程和结果

s2：对前 $n-1$ 个数进行第二趟冒泡排序，结果使次大的数被安置在第 $n-1$ 个元素位置。如图 5-12 所示是第二趟比较的过程和结果。其中，带“ ”的数是要比较的两个数，带“ ”的数是已排好序的数。

第二趟比较的过程和结果如图 5-26 所示。

第1次	第2次	第3次	第4次	结果
3	2	2	2	2
2	3	3	3	3
8	8	8	6	6
6	6	6	8	4
4	4	4	4	8
9	9	9	9	9

图 5-26　第二趟比较的过程和结果

s3：重复上述过程，共经过 $n-1$ 趟冒泡排序后，排序结束。图 5-27 是冒泡法排序过程示意图。

原始数据	9	3	2	8	6	4
第一趟比较	3	2	8	6	4	9
第二趟比较	2	3	6	4	8	9
第三趟比较	2	3	4	6	8	9
第四趟比较	2	3	4	6	8	9
第五趟比较	2	3	4	6	8	9

图 5-27　比较的过程和结果

第一趟比较、调整：两两比较 5 次，找到最大的数 9。

第二趟比较、调整：两两比较 4 次，找到第 2 大的数 8。

第三趟比较、调整：两两比较 3 次，找到第 3 大的数 6。

第四趟比较、调整：两两比较 2 次，找到第 4 大的数 4。

第五趟比较、调整：两两比较 1 次，找到第 5 大的数 3。

剩下的一个数 2 自然是最小的，经 5 轮比较、调整，所有数据都排好序。

下面对例 5-11 引申思考。

引申思考：现有 N 个数，利用“冒泡法”进行升序排序并输出。

问题分析：对于 N 个数需要 $N-1$ 轮比较、调整，第 i 轮两两比较的次数为 $N-i$ 次，可以利用二重循环实现。为了使比较的轮数和每轮比较的次数与数组下标一致，存放 N 个数可定义数组为 int a[$N+1$]，使用 a[1]～a[N]存放 N 个数，a[0]不用。算法 N-S 流程图如图 5-28 所示。

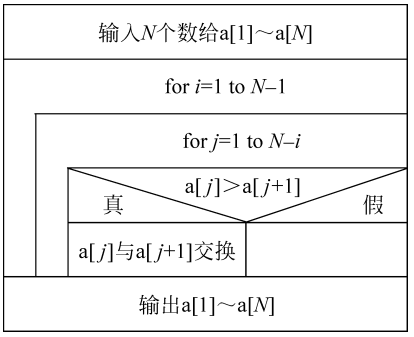


图 5-28　冒泡法排序算法的 N-S 流程图

5.3.2　程序设计和调试运行

1. 程序设计

输入 10 个数，利用“冒泡法”进行升序排序并输出。根据算法描述和 N-S 流程图给出以下源程序：

```
#define N 10
void main()
{
    int a[N+1],i,j,t;
    printf("input %d numbers:\n",N);
    for(i=1;i<=N;i++)              /* 用a[1]到a[10]存放10个数,a[0]不用 */
        scanf("%d",&a[i]);
    for(i=1;i<=N-1;i++)            /* N个数,共比较N-1轮 */
        for(j=1;j<=N-i;j++)        /* 第i轮中,两两比较N-i次 */
            if(a[j]>a[j+1])
                { t=a[j];a[j]=a[j+1];a[j+1]=t;}   /* 交换两个数 */
    printf("the sorted numbers:\n");
    for(i=1;i<=N;i++)
         printf("%d ",a[i]);
}
```

2. 调试运行

下面具体介绍利用 Visual C++6.0 上机运行“冒泡法”排序源程序的操作过程。

(1) 创建工程

启动 Visual C++6.0 环境后，单击菜单“文件”→“新建”，弹出“新建”对话框，显示如图 5-29 所示的“新建”对话框。在左侧的“工程”选项卡中选择 Win32 Console Application 项，然后在右侧的“工程名称”框中输入工程名称，如 FirstC，单击“位置”右方按钮 ...，可以

更改工程保存的位置，在下方的“平台”中勾选 Win32 复选框。

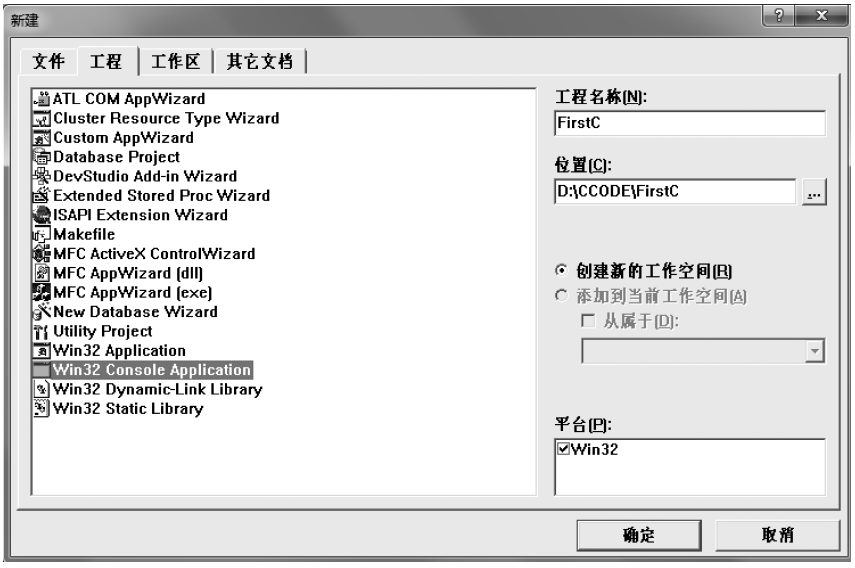


图 5-29 “新建”对话框

单击“确定”按钮，弹出 Win32 Console Application 向导对话框。在该对话框中选择要创建的控制台程序类型，单击选择“一个空工程”项，如图 5-30 所示。

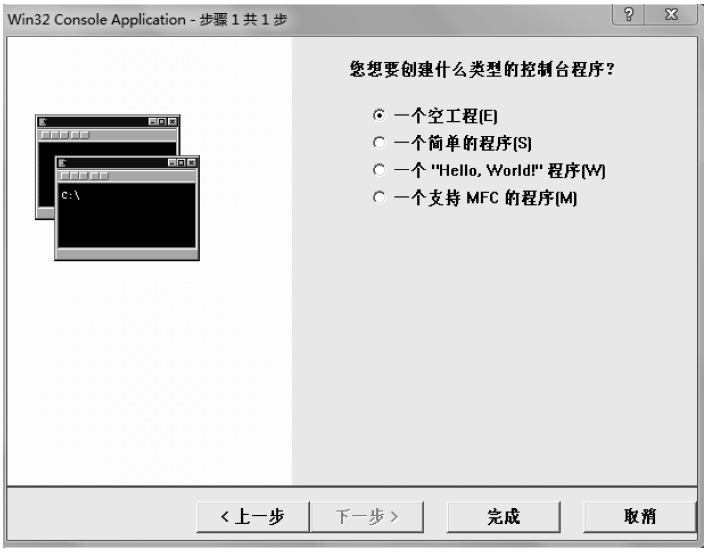


图 5-30 Win32 应用程序向导

单击“完成”按钮，将弹出“新建工程信息”对话框，该对话框中显示所创建的新工程的相关信息。单击“确定”按钮，便完成了工程的创建。

(2) 添加文件

单击菜单项“文件”→“新建”，弹出“新建”对话框，在“文件”选项卡中选择 C++ Source File 项，在右边的“文件名”框中输入文件名，如 FirstC.c，勾选上面的“添加到工程”复选框，

如图 5-31 所示。单击“确定”按钮，即为工程添加了名称为 FirstC 的 C 源文件。

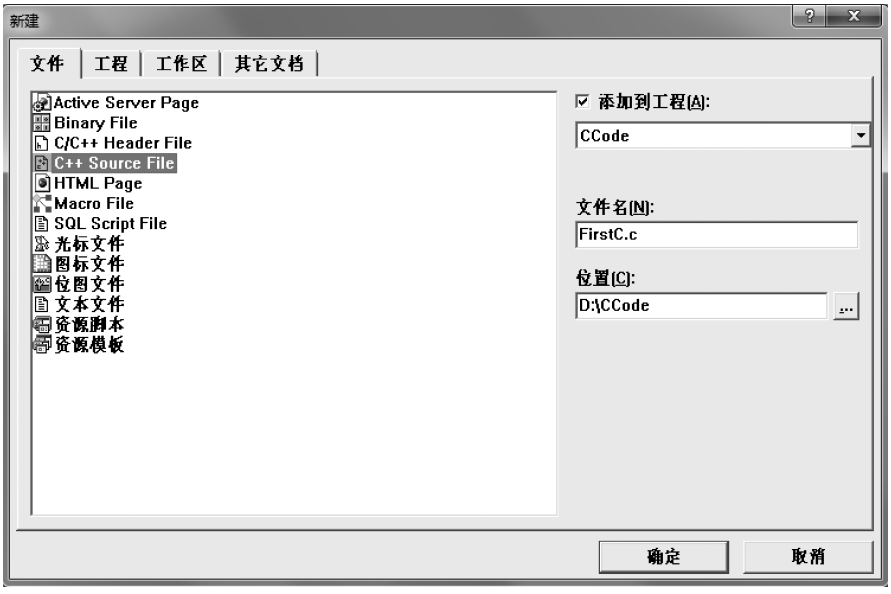


图 5-31　添加文件到工程

（3）编辑、编译、链接与运行

如图 5-32 所示，在右侧的代码编辑区输入“冒泡法”排序的源代码，单击菜单项“组建”→“编译[FirstC.c]”完成编译过程，单击菜单项“组建”→“组建[FirstC.exe]”完成链接过程，编译、链接成功后，单击菜单项“组建”→“执行[FirstC.exe]”执行程序并查看结果，如图 5-33 所示。

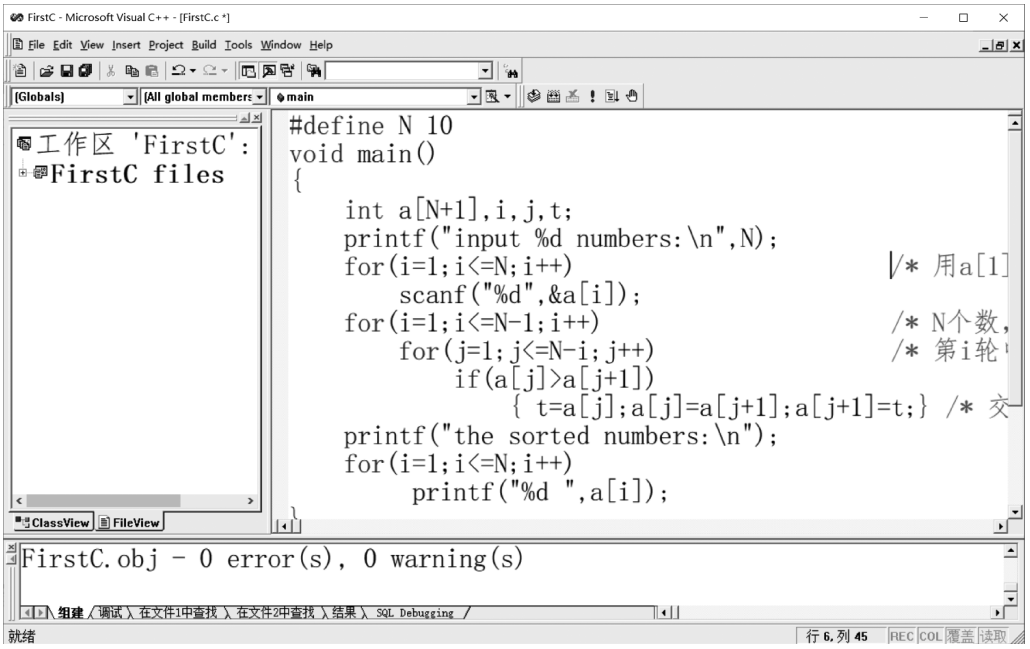


图 5-32　源程序的编辑、编译、链接与运行操作界面

图 5-33　程序的运行结果

5.4　数 据 结 构

数据结构是计算机科学的基石，也是计算机科学与技术专业的核心课程。程序设计的关键问题之一是如何高效地组织和描述数据。不了解施加在数据上的算法就无法决定如何构造数据；反之，算法的设计和选择在很大程度上依赖于作为基础的数据结构，二者相辅相成。N. Wirth 的名言“算法＋数据结构＝程序”精辟地概括了三者之间的关系。

数据的结构分为逻辑结构和物理结构。逻辑结构反映数据成员之间的逻辑关系，而物理结构反映数据成员在计算机内部的存储安排。数据结构主要研究数据的各种逻辑结构和物理存储结构，以及对数据的各种操作（或算法）。通常，算法的设计取决于数据的逻辑结构，算法的实现取决于数据的物理存储结构。

5.4.1　基本概念

在系统学习数据结构知识之前，首先应对一些基本概念和术语赋予确切的含义。

数据（Data）。它是人们利用文字、数字及其他符号对现实世界的事物及其活动所做的描述，即能够被计算机识别、存储、加工处理的信息载体，它是计算机程序加工的原料。在计算机科学中，数据的含义非常广泛，我们把一切能够输入到计算机中并被计算机程序处理的信息，包括数值、字符、文字、图形、图像语音等都称为数据。

数据类型（Data Type）。数据的定义域，是一个值的集合以及定义在这个值集上的一组操作。常见的数据类型有字符型、整数型、逻辑型、数组、集合、记录等。变量是用来存储值的所在处，它们有名字和数据类型。变量的数据类型决定了如何将代表这些值的位存储到计算机的内存中。在声明变量时也可指定它的数据类型。所有变量都具有数据类型，以决定能够存储哪种数据。

数据项（Data Item）。数据项是数据的不可分割的最小单位，是数据记录中最基本的不可分的有名数据单位，是具有独立含义的最小标识单位。数据项可以是字母、数字或两者的组合。通过数据类型（逻辑的、数值的、字符的等）及数据长度来描述。数据项用来描述实体的某种属性。

数据元素（Data Element）。它是数据的基本单位，由数据项组成。在计算机程序中通常作为一个整体进行考虑和处理。有时一个数据元素可由若干数据项组成，例如，一本书的书目信息为一个数据元素，而书目信息的每一项（如书名、作者名等）为一个数据项。同类数据元素的集合称为数据对象。

数据结构(Data Structure)。它是相互之间存在着一种或多种关系的数据元素的集合和该集合中数据元素之间的关系组成,记为 Data_Structure=(D,R),其中 D 是数据元素的集合,R 是该集合中所有元素之间的关系的有限集合。

例如,在学生成绩排序问题中有 5 条记录(表中的一行称为一条记录),即有 5 个数据元素,每个数据元素包括 3 个数据项:学号、姓名、成绩。其中学号、姓名为字符型数据,成绩为整数型数据。所有学生记录放在一起构成的集合(表)即为数据对象。

数据结构研究的内容包括数据的逻辑结构、存储结构以及基本数据操作。

1. 数据的逻辑结构

数据的逻辑结构指数据元素之间的逻辑关系,其中的逻辑关系是指数据元素之间的前后关系,它与数据在计算机中的存储位置无关。数据的逻辑结构分为三类:

(1) 线性结构。数据之间存在前后顺序关系,除第一个元素和最后一个元素外,其他节点都有唯一一个前驱和一个后继节点(一对一关系),包括数组、链表、栈和队列等,如图 5-34 所示。

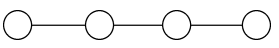


图 5-34　线性结构

(2) 树形结构。数据之间存在顺序关系,除了一个根节点外,其他节点都有唯一一个前驱节点,且可以有多个后继节点(一对多关系),如图 5-35 所示。

(3) 网状结构。每个节点都可以有多个前驱和多个后继节点(多对多关系),如图 5-36 所示。

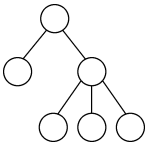


图 5-35　树形结构

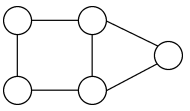


图 5-36　网状结构

例如,排好序的学生成绩之间构成线性结构,企事业单位各部门之间的隶属关系是树形结构,旅行商问题中城市之间的公路连接则为网状结构。

2. 数据的存储结构

数据的存储结构指数据的逻辑结构到计算机存储器的映像。它有多种方式,顺序存储结构和链式存储结构是两种最主要的存储方式。

(1) 顺序存储结构

顺序存储结构是把一组节点存放在地址相邻的存储单元里,节点间的逻辑关系用存储单元的自然顺序关系来表达,即用一块连续存储区域存储线性数据结构。

顺序存储结构的优点在于:数据之间逻辑相邻、物理相邻,可随机存储任一数据元素,存储空间较紧凑。其缺点在于:每次插入或者删除数据元素都需要移动大量数据,且顺序存储结构需要事先分配好内存空间,有可能不需要那么多内存空间而造成内存空间损失,也有可能内存空间分配不够,造成数据缺失。

(2) 链式存储结构

链式存储结构是在节点的存储结构中附加指针域(指针字段)来存储节点间的逻辑关系。链式存储结构中数据节点包括两部分,即数据域(数据字段)和指针域(指针字段),如

图 5-37 所示。

data：数据域，存放节点本身的值。

next：指针域，存放节点的直接后继节点的地址。

图 5-37 链式存储结构中节点的组成

链式存储结构打破了计算机存储单元的连续性，可以将逻辑上相邻的两个数据元素存放在物理上不相邻的存储单元中。其特点如下：

① 节点中除数据外，还有表示链接信息的指针域，因此与顺序存储结构相比，占用的存储空间更大。

② 逻辑上相邻的节点物理上不一定相邻，可用于线性表、树、图等多种逻辑结构的存储。

③ 插入、删除等操作灵活方便，不需要大量移动节点，只需修改节点的指针值即可。

3. 基本数据操作

数据的每一种逻辑结构都有相对应的基本运算或操作，主要包括查找（检索）、排序、插入、更新及删除等。例如，从学生表中删除一个学生记录，向家族树中插入一个家庭成员等。数据的运算定义在数据逻辑结构上，而其运算的具体实现要在存储结构上进行。

数据类型这一概念最早出现在高级程序设计语言中，它与数据结构密切相关，用于描述程序操作对象的特性。在传统的程序设计语言中，所提供的数据类型即反映了其数据结构。无论采用哪种高级语言编写程序，其中的每个变量、常量以及表达式都有其确定的数据类型。简单的数据结构可以用单一的标准数据类型（如整型、实型和字符型等）来定义，而复杂的数据结构（如数组、记录、指针等）需要用简单的数据结构复合而构成，在此基础上还可以得到更为复杂的数据结构。各种数据类型明显或隐含地规定了在程序执行期间变量或表达式的取值范围及其允许的操作。由此可见，数据类型是一个值的集合和定义在这个集合上的一组操作的总称。数据类型也可以说是某种程序设计语言中已实现的数据结构。

近些年使用的面向对象语言（如 C++语言）则根据抽象数据类型的理论，在程序中可以将数据结构的逻辑构成和它的运算操作一并定义，封装成一个整体作为一类对象。在程序使用时，即可对相应对象类的变量进行调用。

5.4.2 线性表

下面简要介绍几种常用数据结构。

线性表（Linear List）是由 n 个数据元素构成的有限序列（a_1，a_2，…，a_n），即按照一定的线性顺序排列而成的数据元素的集合。线性表是最简单最常用的一种线性结构。该结构上的基本操作包括对元素的查找、插入和删除等。

例如，学生成绩按高低排序问题中，可将学生成绩按线性表组织和存储。

数组、链表、栈和队列是最常用的线性表。

1. 数组

它是 n 个类型相同的数据元素构成的序列，它们连续存储在计算机的存储器中，且数组中的每个元素占据相同的存储空间。

对数组的描述通常包含下列 5 种属性。

数组名称：声明数组第一个元素在内存中的起始地址。

数组下标：元素在数组中的储存位置。

数组类型：声明此数组的类型，它决定数组元素在内存所占有的存储空间的大小。

数组元素个数：是数组下标上限与数组下标下限的差加 1。

维度：每一个元素所含数据项的个数，如一维数组、二维数组等。

对数组的常见操作包括插入、删除、排序、查找等。

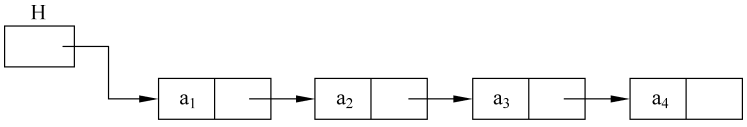
如果数组中存放的是串，它的典型操作与数字数组不同。串是字母表中的字符序列，并以一个特殊字符来标识串的结束。由 0 和 1 组成的字符串称为二进制串或比特串。串的常见操作包括计算串长度，按照字典序比较两个串的优先顺序，以及连接两个串（由两个给定的字符串构造一个新串，将第二个串附加在第一个串的尾部）。

2. 链表

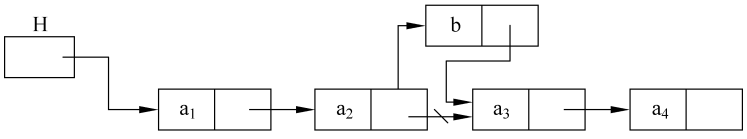
用链式存储结构存储的线性表称为链表。根据链接方式的不同，链表可分为线性链表、双向链表和循环链表等。

（1）线性链表（单链表）

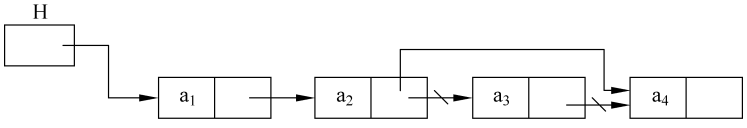
线性链表就是链式存储的顺序表，其节点中只含有一个指针域，用来指出其后继节点的存储位置。线性链表的最后一个节点无后继节点，它的指针域为空（记为 NULL 或 0）。另外，要设置表头节点，指向线性表的第一个节点。单链表及其插入和删除操作如图 5-38 所示。



(a) 由a_1～a_4元素组成的链表



(b) a_2和a_3之间插入一个元素b



(c) 在原链表中删除元素a_3

图 5-38　单链表及其插入、删除运算

（2）双向链表

双向链表是指链表中每个节点有两个指针：左指针和右指针。左指针指向前趋，右指针指向后继。

（3）循环链表

循环链表是指在链表中增加一个表头节点，链表的最后一个节点的指针域不为空，而是指向表头节点。

为了访问链表中的某个特定元素，可以从链表的第一个元素开始，沿着一系列指针前进，直到访问到该特定元素为止。所以，访问单链表中的元素需要的时间依赖于该元素在链

表中所处的位置。其优点是链表不需要事先分配任何存储空间，并且通过重新链接一些相关指针，使插入和删除操作效率非常高。我们可以把链表想象成火车，有多少人就挂多少节车厢，人多了就向系统多要一个车厢；人少了就把车厢还给系统。链表也是一样，有多少数据就用多少内存空间，有新数据加入就向系统再要一块内存空间；而数据删除后，就把空间还给系统。

3. 栈

栈(堆栈)是一种操作上受限的特殊线性表，它只允许在一端进行插入和删除操作。允许进行操作的一端称为栈顶，栈顶的位置是随插入和删除操作动态变化的。不允许操作的一端称为栈底。这种结构的特点是“后进先出”，因此栈也被称为“后进先出表”。

与线性表类似，栈的存储结构可以采用顺序存储结构，也可以采用链式存储结构。采用顺序结构存储的栈称为顺序栈，采用链式结构存储的称为带链接的栈。无论采用哪种存储方式，对栈的操作只能在栈顶进行。栈的逻辑结构如图 5-39 所示。

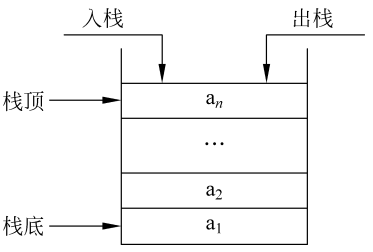


图 5-39　栈的逻辑结构示意图

栈的基本运算包括入栈运算、出栈运算等。入栈运算是指在栈顶插入一个新元素。入栈运算的步骤是：将栈顶指计加 1，然后在栈顶插入新元素。出栈运算是指从栈顶取出一个元素。出栈运算的步骤是：取出栈顶元素，并赋给一个指定的变量，然后将栈顶指针退 1。

4. 队列

队列也是一种操作上受限的特殊线性表，它和堆栈所受的限制不同，只允许在表的一端进行插入操作，在另一端进行删除操作。队列中允许插入的一端称为队尾，允许删除的一端为队首。因此，这样的队列又称为“先进先出表”。队列的示意图如图 5-40 所示。

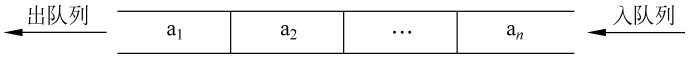


图 5-40　队列示意图

由于队列也是一种特殊的线性表，因此队列的存储结构可以采用顺序存储结构，也可以采用链接存储结构，采用顺序存储结构的队列称为顺序队列，采用链接存储结构的队列称为带链接的队列。无论采用哪种存储方式，对队列的操作只能限制在队尾插入和队首删除。

为了充分利用空间，在实际应用中，队列的顺序存储结构一般采用循环队列的形式。所谓循环队列，就是将队列存储空间的最后一个位置指向第一个位置，从而使顺序队列形成逻辑上的环状空间。在循环队列中，当存储空间的最后一个位置已被使用而要进行入队运算时，只要存储空间的第一个位置空闲，便可将元素加入第一个位置，即将存储空间的第一个位置作为队尾。

队列的基本运算有入队运算和出队运算。入队运算是指在队尾插入一个元素，入队运算的步骤是：首先将队尾指针加 1；然后将新元素插入到队尾指针指向的位置。出队运算是指在队首删除一个元素，出队运算的步骤是：将队首指针减 1；然后将队首指针指向的元素赋给指定的变量。

5.4.3　树

1. 树的定义和相关术语

树(tree)是由 $n(n\geqslant 0)$个有限节点组成一个具有层次关系的集合。有一个特定的节点称为根(root),其余的节点分为 $m(m\geqslant 0)$个互不相交的有限集合,其中每一个集合自身又是一棵树,称为根节点的子树。

例如,树的示意图如图 5-41 所示,树中的 A 节点是根节点,这棵树共有 13 个节点。

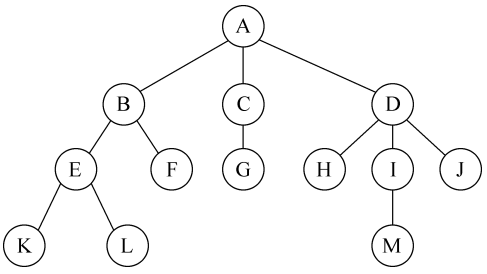


图 5-41　树的示意图

下面介绍树形结构的常用术语。

根节点：树形结构的最高层次的节点称为根节点,这个节点无直接前趋节点。

子树：除根节点以外的其余节点可分为若干互不相交的有限集,其中每一个集合本身又是一棵树,将它们称为子树。

双亲节点或父节点：若一个节点含有子节点,则这个节点称为其子节点的父节点。图 5-41 中,A 是 B、C、D 的父节点。

孩子节点或子节点：一个节点含有的子树的根节点称为该节点的子节点。图 5-41 中,B、C、D 是 A 的子节点。

兄弟节点：具有同一个父节点的子节点互称为兄弟节点。图 5-41 中,B、C、D 互为兄弟节点。

节点的度：一个节点的子树的个数。图 5-41 中,节点 A 的度为 3,节点 B 的度为 2。

树的度：一棵树中,最大的节点的度称为树的度。图 5-41 中,树的度为 3。

叶子节点：度为 0 的节点。图 5-41 中,K、L、F、G、H、M、J 是叶子节点。

节点的层数：根节点的层数为 1,从根节点到某节点的层数称为该节点的层数。从根开始定义起,根为第 1 层,根的子节点为第 2 层,以此类推。图 5-41 中,节点 A 的层数是 1,节点 B 的层数是 2,节点 M 的层数是 4。

树的高度或深度：树中所有节点的层数最大值。图 5-41 中,树的深度为 4。

2. 二叉树

二叉树是树形结构的一种重要类型,它的每个节点最多有两个子节点,且有先后次序。由于对二叉树的操作算法简单,而任何树都可以转化为二叉树处理,所以二叉树在树结构的实际应用中起着重要的作用。

(1) 二叉树的定义

二叉树是 $n(n\geqslant 0)$个节点的有限集合。它或者为空集,或者是由一个根节点加上两棵

互相不相交的左子树和右子树组成，并且左子树和右子树都是二叉树。二叉树的定义是一个递归定义。二叉树是一种特殊的有序树，二叉树的子树有左、右之分。逻辑上二叉树有5种基本形态：

空二叉树——如图5-42(a)所示；

只有一个根节点的二叉树——如图5-42(b)所示；

只有左子树——如图5-42(c)所示；

只有右子树——如图5-42(d)所示；

完全二叉树——如图5-42(e)所示。

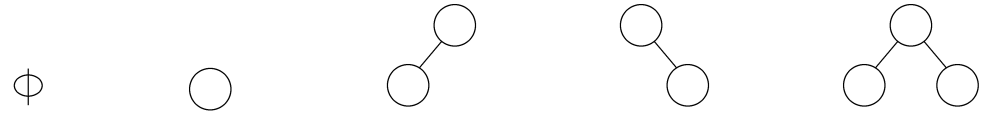


图5-42　树的示意图

(2) 二叉树的存储

二叉树有两种存储结构：顺序存储结构和链式存储结构。

顺序存储结构借用数组将二叉树中的数据元素存储起来，此方式只适用于完全二叉树，如果想存储普通二叉树，需要将普通二叉树转化为完全二叉树。使用数组存储完全二叉树时，从数组的起始地址开始，按层次顺序从左往右依次存储完全二叉树中的节点。当提取时，根据完全二叉树的相关性质，可以将二叉树进行还原。

采用链式存储结构存储二叉树，就非常容易理解了。根据每个节点的结构，至少需要3部分组成，如图5-43所示。

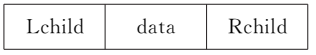


图5-43　二叉树链式存储结构的节点结构

Lchild代表指向左孩子的指针域；data为数据域；Rchild代表指向右孩子的指针域。使用此种节点构建的二叉树称为“二叉链表”。如图5-44所示的就是二叉树的链式存储结构。

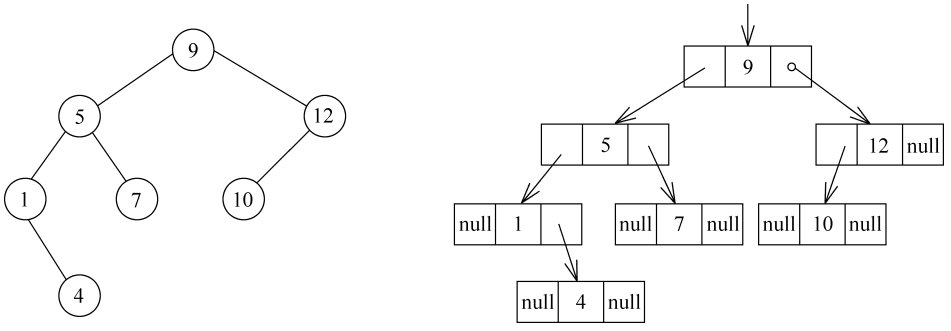


图5-44　二叉树的链式存储结构

(3) 二叉树的遍历

遍历是对树的一种最基本的运算。所谓遍历二叉树，就是按一定的规则和顺序走遍二叉树的所有节点，使每一个节点都被访问一次，而且只被访问一次。由于二叉树是非线性结构，因此，树的遍历实质上是将二叉树的各个节点转换成为一个线性序列来表示。

根据访问节点的次序不同，遍历表达法有 3 种方法：先序遍历、中序遍历、后序遍历。

① 先序遍历。先访问根，然后按先序遍历左子树，最后按先序遍历右子树。

② 中序遍历。先访问左(右)子树，然后访问根，最后访问右(左)子树。

③ 后序遍历。先按后序遍历左子树，然后按后序遍历右子树，最后访问根。

例如，对于图 5-45 中的二叉树，采用上述 3 种遍历次序节点序列如下。

先序遍历为 ABDECF；

中序遍历为 DBEAFC；

后序遍历为 DEBFCA。

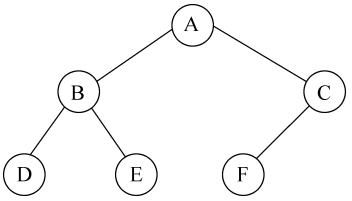


图 5-45　二叉树

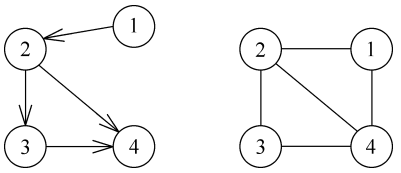


图 5-46　有向图和无向图示意图

5.4.4　图

1. 图的定义

图是由一组数据元素(称为顶点)的有限集和描述顶点间相互关系的边(或弧)的有限集组成。它是一种网状结构。

图分为有向图和无向图。在有向图中，顶点与顶点之间的连线是具有方向的，这样的线称为弧；在无向图中，顶点与顶点间的连线是没有方向的，这样的连线称为边。有向图和无向图的示意如图 5-46 所示。

2. 图的存储结构及运算

(1) 图的存储结构

图是比线性表和树更为复杂的一种数据结构，它是非线性的。由于图中任何两个数据元素之间都有可能存在关系，所以对图要进行运算，一定要采用合适的存储结构。图也有顺序存储结构和链式存储结构，常用的图的存储结构有邻接矩阵、邻接表、邻接多重表等。

(2) 图的遍历

图的遍历是图形结构的重要运算，它是指从图的某一个节点开始按照某种顺序依次访问图中的所有节点，而且每个节点只访问一次。

图的遍历要考虑如下问题：

① 任意一个顶点都可作为第一个被访问的节点。

② 若有回路存在，那么一个顶点被访问之后，有可能沿回路又回到该顶点。

③ 非连通图中，从一个顶点出发，只能访问它所连通的所有顶点，怎样选取下一个出发点以便访问其余的连通顶点。

④ 一个顶点若和其他多个顶点相连，访问它以后怎样选取下一个要访问的顶点。