第3章

最简单的C程序设计——顺序程序设计

有了前两章的基础，现在可以编写简单的C程序了。要编写出一个正确的C语言程序，需要两个方面的知识：一是根据所解问题的要求，设计出解题的具体步骤，这一步骤称为设计算法；二是用C语言写出程序，以便计算机能正确地执行。这两方面的知识是缺一不可的。也就是说，既要懂得算法，能设计算法，又需要掌握C语言的知识，能够灵活地运用它们写出可供计算机执行的C程序。

本章介绍有关算法的知识，同时也介绍最简单、最基本的C语句，并把这二者紧密结合起来，引导读者编写最简单的C语言程序，为以后逐步深入地学习、编写较复杂的C程序打下初步的基础。

3.1 算法是程序的灵魂

3.1.1 算法的概念

在前面两章中，读者已经清楚地看到：计算机所进行的一切操作都是由程序决定的，程序是由人们事先编写好并输入给计算机的。从前面的程序中可知，一个程序包括以下两个方面的内容：

(1) **对数据的描述**。在程序中要指定数据的类型和数据的组织形式，即**数据结构**(data structure)。

(2) **对操作的描述**。即操作步骤，也就是**算法**(algorithm)。

数据是操作的对象，操作的目的是对数据进行加工处理，以得到期望的结果。打个比方，厨师制作菜肴，需要有菜谱，菜谱上一般应包括：①配料，指出应使用哪些原料；②操作步骤，指出如何使用这些原料，按规定的步骤加工成所需的菜肴，没有原料是无法加工成所需菜肴的。面对同一些原料可以加工出不同风味的菜肴。作为程序设计人员，必须认真考虑和设计数据结构和操作步骤(即算法)。著名计算机科学家沃思(Nikiklaus Wirth)提出一个公式：

数据结构 + 算法 = 程序

这是对面向过程程序的概括。实际上，一个程序除了以上两个主要要素之外，还应当

采用结构化程序设计方法进行程序设计，并且用某一种计算机语言表示。因此，算法、数据结构、程序设计方法和语言工具4个方面是一个程序设计人员所应具备的知识。在设计一个程序时要综合运用这几方面的知识。在这4个方面中，**算法是灵魂，数据结构是加工对象，语言是工具，编程需要采用合适的方法**。

算法是解决“做什么”和“怎么做”的问题。程序中的操作语句，实际上就是算法的体现。显然，不了解算法就谈不上程序设计。本书不是一本专门介绍算法的教材，也不是一本只介绍C语言语法规则的使用说明。本书的目的是使读者通过学习，能够知道怎样编写一个C程序，并且能够编写出不太复杂的C程序。我们将通过一些实例把以上4个方面的知识结合起来，介绍如何编写一个C程序。

首先通俗地说明什么是算法。做任何事情都有一定的内容和步骤。例如，你想从北京去天津开会，首先要去买火车票，然后按时乘坐地铁到北京站，登上火车，到天津站后坐汽车到会场，参加会议；你要买电视机，先要选好货物，然后开票，付款，拿发票，取货，打车回家；要考大学，首先要填报名单，交报名费，拿到准考证，按时参加考试，得到录取通知书，到指定学校报到注册等。这些步骤都是按一定的顺序进行的，缺一不可，次序错了也不行。我们从事各种工作和活动，都必须事先想好进行的步骤，然后按部就班地进行，才能避免产生错乱。实际上，在日常生活中，由于已养成习惯，所以人们并没意识到每件事都需要事先设计出“行动步骤”。例如吃饭、上学、打球、做作业等，事实上都是按照一定的规律进行的，只是人们不必每次都重复考虑它而已。

不要认为只有“计算”的问题才有算法。广义地说，为解决一个问题而采取的方法和步骤，就称为“算法”。例如，描述太极拳动作的图解，就是“太极拳的算法”。一首歌曲的乐谱，也可以称为该歌曲的算法，因为它指定了演奏该歌曲的每一个步骤，按照它的规定就能演奏出预定的曲子。

对同一个问题，可以有不同的解题方法和步骤。例如，求1＋2＋3＋…＋100，即$\sum_{n=1}^{100} n$。有人可能先进行1＋2，再加3，再加4，一直加到100，而有的人采取这样的方法：100＋(1＋99)＋(2＋98)＋…＋(49＋51)＋50＝ 100＋49×100＋50＝5050。还可以有其他的方法。当然，方法有优劣之分。有的方法只需进行很少的步骤，而有些方法则需要较多的步骤。一般说，人们都希望采用方法简单、运算步骤少的方法。因此，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，选择合适的算法。

本书所关心的当然只限于计算机算法，即计算机能执行的算法。例如，让计算机计算1×2×3×4×5，或将100个学生的成绩按高低分数的次序排列，是可以做到的，而让计算机去执行“替我理发”或“煎一份牛排”，是做不到的(至少目前如此)。

计算机算法可分为两大类别：**数值运算算法**和**非数值运算算法**。数值运算的目的是求数值解，例如求圆面积、求方程的根、求一个函数的定积分、判断某年是否闰年等，这些都属于数值运算范围。非数值运算包括的面十分广泛，最常见的是用于事务管理领域，例如图书检索、学生成绩管理、商品销售管理、对一个单位的成员按年龄排序等。目前，计算机在非数值运算方面的应用远远超过了在数值运算方面的应用。由于数值运算有现成的模型，可以运用数值分析方法，因此对数值运算的算法的研究比较深入，算法比较成熟。

人们常常把这些算法汇编成册(写成程序形式),或者将这些程序存放在磁盘上,供用户调用。例如有的计算机系统提供“数学程序库”,使用起来十分方便。而非数值运算的种类繁多,要求各异,难以规范化,因此只对一些典型的非数值运算算法(例如排序算法)作比较深入的研究。其他的非数值运算问题,往往需要使用者参考已有的类似算法,重新设计解决特定问题的专门算法。本书不可能罗列所有算法,只是想通过一些典型算法的介绍,帮助读者了解如何设计一个算法,并引导读者举一反三。

在写程序之前,必须想清楚“做什么”和“怎么做”。“做什么”往往是从题目或任务中可以看出来或整理出来的(例如:求三角形的面积、统计学生成绩等),而“怎么做”则要由程序设计者去思考和设计的。“怎么做”包括两方面的内容:一是要做哪些事情才能达到解决问题的目的;二是决定做这些事情的先后次序。这就是“算法”所要解决的问题。

3.1.2 怎样表示算法

想好一个算法后,应该采用适当的方式表示出来,以便根据它编写程序。

1. 用自然语言表示算法

如本章 3.1.1 节表示的那样。自然语言就是人们日常使用的语言,可以是汉语、英语,或其他语言。用自然语言表示通俗易懂,但文字冗长,容易出现歧义性。自然语言表示的含义往往不大严格,要根据上下文才能判断其正确含义。假如有这样一句话:“张先生对李先生说他的孩子考上了大学。”请问是张先生的孩子考上大学呢还是李先生的孩子考上大学呢?光从这句话本身难以判断。此外,用自然语言来描述包含条件判断和循环的算法,不很方便。因此,除了那些很简单的问题以外,一般不用自然语言描述算法。

2. 用流程图表示算法

流程图是指用来表示各种操作的一些图框。美国国家标准化协会 ANSI(American National Standard Institute)规定了一些常用的流程图符号(见图 3.1),这些符号已为世界各国程序工作者普遍采用。

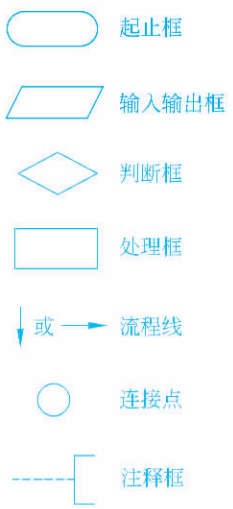

图 3.1

如判断一个数是否为偶数的算法,用流程图表示如图 3.2 所示。图中的菱形框用来判断“m 能否被 2 整除”,如果能,就输出该数“是偶数”,否则,就输出该数“不是偶数”。

输出 1～100 的算法,用流程图表示如图 3.3 所示。

首先,把 1 赋给变量 n,即置 n 的初值为 1,然后判别 n 的值是否小于或等于 10,今 n 的值小于 10,故应执行“输出 n 的值”,输出数值 1;其次,执行 $n=n+1$,即将 n 的值加 1 后再赋给 n,故 n 变成 2 了,然后返回去判断 n 是否小于或等于 10,今 n 的值小于 10,故输出 n 的值(今为 2),再使 n 变成 3,再判断 n 是否小于或等于 10,……,如此反复循环,直到第 10 次,输出完 n 的当前值 10 后,n 变成 11 了,再判断时,n 已大于 10 了,所以不再执此“输出 n 的值”和“使 n 增值 1”的操作,算法结束。

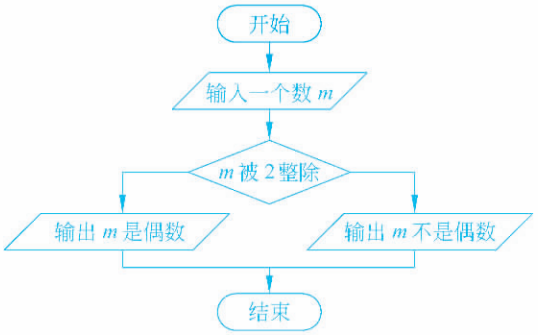

图 3.2

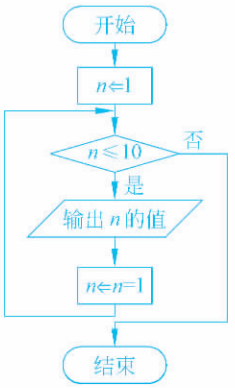

图 3.3

用这种流程图表示算法，直观形象，易于理解。但画图比较麻烦，需要占用较大的纸面面积，而且修改比较困难，现在使用不多了。

3. 用 N-S 流程图表示算法

1973 年美国学者 I. Nassi 和 B. Shneiderman 提出了一种新的流程图形式。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于它的框，或者说，由一些基本的框组成一个大的框。这种流程图又称 **N-S 结构化流程图**（N 和 S 是两位美国学者的英文姓氏的首字母）。这种流程图适于结构化程序设计，而且作图简单，占面积小，一目了然，因而很受欢迎。

N-S 流程图用以下的流程图符号。

（1）顺序结构。顺序结构用图 3.4 所示的形式表示。表示执行完 A 操作后，接着执行 B 操作。

（2）选择结构。选择结构用图 3.5 所示的形式表示。当 p 条件成立时执行 A 操作，p 不成立则执行 B 操作。

图 3.2 可以改用 N-S 流程图表示，如图 3.6 所示。

图 3.4

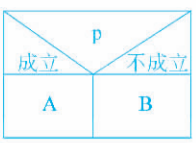

图 3.5

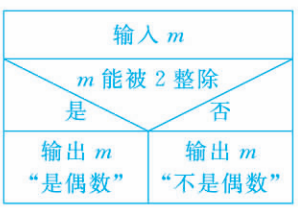

图 3.6

（3）循环结构。循环结构可用图 3.7 所示的形式表示。图 3.7 表示当 p_1 条件成立时反复执行 A 操作，直到 p_1 条件不成立为止。

输出 1～100 的算法，用 N-S 流程图表示如图 3.8 所示。它的流程与图 3.3 相同。

在本章和后续各章中，读者将会了解在程序设计中怎样使用流程图。

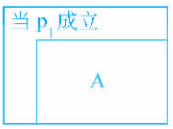

图 3.7

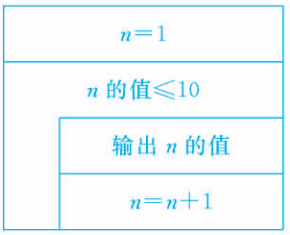

图 3.8

4. 用伪代码表示算法

用传统的流程图和 N-S 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，可能要反复修改，而修改流程图是比较麻烦的。因此，流程图适宜于表示一个算法，但在设计算法过程中使用不是很理想（尤其是当算法比较复杂、需要反复修改时）。为了设计算法时方便，常用一种称为**伪代码**（pseudo code）的工具。

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。它如同一篇文章一样，自上而下地写下来，每一行（或几行）表示一个基本操作。它不用图形符号，因此书写方便，格式紧凑，也比较好懂，便于向计算机语言算法（即程序）过渡。

例如，“输出 x 的绝对值”的算法可以用伪代码表示如下：

```
if x is positive then
  print x
else
  print -x
```

它好像一个英语句子一样好懂，在西方国家用得比较普遍。

也可以用汉字伪代码。例如：

```
若 x 为正
    输出 x
否则
    输出-x
```

也可以中英文混用，例如：

```
if x 为正
  print x
else
  print -x
```

将计算机语言中的关键字用英文表示，其他的可用汉字。

总之，以便于书写和阅读为原则，用伪代码写算法并无固定的、严格的语法规则，只要把意思表达清楚，并且把书写的格式写成清晰易读的形式即可。

在以上几种表示算法的方法中，具有熟练编程经验的专业人士喜欢用伪代码，初学者喜欢用流程图或 N-S 图，因为它比较形象，易于理解。本书主要使用 N-S 图表示算法。

3.2 程序的三种基本结构

在 3.1 节介绍 N-S 流程图时已提到了程序中用到的三种结构：顺序结构、判断结构和循环结构。在本节中再作进一步介绍。

一个程序包含一系列的执行语句，每一个语句使计算机完成一种操作。在写程序时，要仔细考虑各语句的排列顺序，程序中语句的顺序不是任意书写而无规律的。假如一个程序的流程如同图 3.9 那样无规律地跳转，虽然它也能执行并得到正确的结果，但是在阅读这样的程序时，很难清晰地理解其算法。这样的程序是难阅读、难修改、难维护的。写程序应当遵循一定的规律，尽量避免不必要的跳转，最好是使各语句按照从上到下的顺序排列，在执行时也是按从上到下的顺序执行。1966 年，Bohra 和 Jacopini 提出了以下 3 种基本结构，如果用这 3 种基本结构作为算法的基本单元来编写程序，就能实现上面的目的。

(1) **顺序结构**。各操作步骤是顺序执行的。如图 3.10 所示，虚线框内是一个顺序结构，其中 A 和 B 两个框是顺序执行的，即在执行完 A 框所指定的操作后，必然接着执行 B 框所指定的操作。顺序结构是最简单的一种基本结构。

(2) **选择结构**。选择结构又称**判断结构或分支结构**，它根据是否满足给定的条件而从两组操作中选择一种操作。如图 3.11 所示，虚线框内是一个选择结构。此结构中必包含一个判断条件 p(以菱形框表示)，根据给定的条件 p 是否成立而选择执行 A 组操作或 B 组操作。p 所代表的条件可以是“x＜0”或“x＞y”，“a＋b ＜c＋d”等，详见第 4 章。

图 3.9

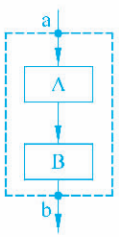

图 3.10

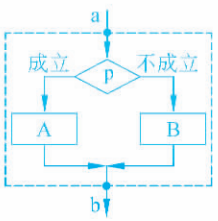

图 3.11

第 1 章例 1.3 中的 if 语句

```
if  (x>y) z=x;              /* 如果满足 x>y 条件,执行 z=x* /
else z=y;                   /* 如果不满足 x>y 条件,执行 z=y* /
```

就是一个选择结构。

注意：无论 p 条件是否成立，A 操作或 B 操作只能执行其中之一，不可能既执行 A 操作又执行 B 操作。无论走哪一条路径，在执行完 A 或 B 之后，算法就结束了。A 或 B 两个操作中可以有一个是空操作，即不执行任何操作，如图 3.12 所示。

(3) 循环结构。它又称重复结构，即在一定条件下反复执行某一部分的操作。图 3.13 所示的就是一种循环结构。执行过程是，当给定的条件 p 成立时，执行 A 操作，执行完 A 后，再判断条件 p 是否成立，如果仍然成立，再执行 A，如此反复执行 A，直到某一次 p 条件不成立为止，此时不执行 A，而脱离循环结构。

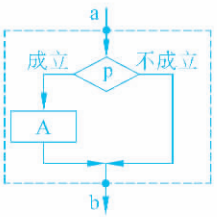

图 3.12

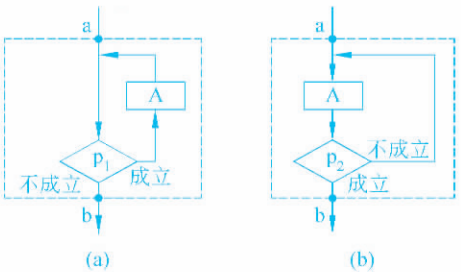

图 3.13

一个良好的程序，无论多么复杂，都可以由这 3 种基本结构组成。用这三种基本结构构造算法和编写程序，就如同用一些预构件盖房子一样方便，程序结构清晰。有人形容这三种基本结构像“项链中的珍珠”一样排列整齐、清晰可见。用这三种基本结构构成的程序称为**结构化程序**。

C 语言提供了实现三种基本结构的语句，如用 if 语句可以实现选择结构，用循环语句(for 语句，while 语句) 可以实现循环结构。凡是能提供实现三种基本结构的语句的语言，称为结构化语言。显然，C 语言属于结构化语言。

本章只介绍能实现顺序结构的语句，它们只执行最简单的操作。

3.3 C 语句综述

和其他高级语言一样，C 语言的语句用来向计算机系统发出操作指令。一条语句经编译后产生若干条机器指令，一个实际的程序应当包含若干语句。从第 1 章已知，一个程序是由若干函数组成的，在一个函数的函数体中一般包括两个部分：**声明部分**和**执行部分**(有的简单的程序可以不包含声明部分，而只有执行语句，如第 1 章中的例 1.1)，执行部分是由语句组成的。根据 C89 标准，C 语句都是用来完成一定操作任务的。声明部分的内容不称为语句，如“int a;”就不是一条 C 语句，它不产生机器操作，而只是对变量的定义。

C 程序结构可以用图 3.14 表示，即一个 C 程序可以由若干个源程序文件(分别进行编译的文件模块)组成，一个源文件可以由若干个函数和预处理命令以及全局变量声明部分组成(关于“全局变量”见第 7 章)，一个函数由数据声明部分和执行语句组成。

C 语句分为以下 5 类。

(1) **控制语句**。控制语句用于完成一定的控制功能。C 只有 9 种控制语句，它们是：

① if()…else…　　(条件语句，用来实现选择结构)

② switch　　(多分支选择语句)

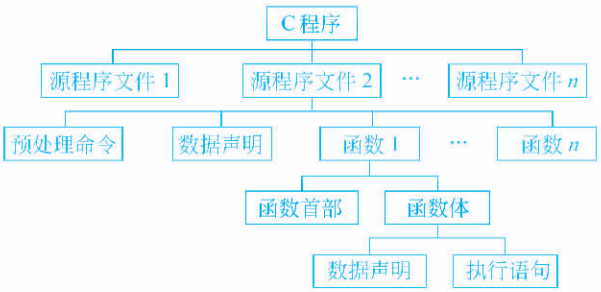

图 3.14

③ for()…　　（循环语句，用来实现循环结构）
④ while()…　　（循环语句，用来实现循环结构）
⑤ do…while()　　（循环语句，用来实现循环结构）
⑥ continue　　（结束本次循环语句）
⑦ break　　（中止执行 switch 或循环语句）
⑧ return　　（从函数返回语句）
⑨ goto　　（转向语句，现已基本不用了）

以上 9 种语句表示形式中的括号“()”表示括号中是一个判别条件，“…”表示内嵌的语句。例如：“if()…else…”的具体语句可以写成：

```
if(x<y)z=x;else z=y;
```

其中 x＞y 是一个判别条件，“z＝x;”和“z＝y;”是语句，这两个语句是内嵌在 if…else 语句中的。这个 if…else 语句的作用是：先判别条件 x＞y 是否成立，如果 x＞y 成立，就执行内嵌语句“z＝x;”；否则就执行内嵌语句“z＝y;”。

（2）**函数调用语句**。函数调用语句由一个函数调用加一个分号构成，例如：

```
printf("This is a C statement.");
```

printf 是一个函数，上面的语句是调用 printf 函数，后面加一个分号。此外没有其他的内容。

（3）**表达式语句**。表达式语句由一个表达式加一个分号构成，最典型的是，由赋值表达式构成一个赋值语句。例如：

```
a=3
```

是一个赋值表达式，而

```
a=3;
```

是一个赋值语句。可以看到，一个表达式的最后加一个分号就成了一个语句。一个语句必须在最后出现分号，分号是语句中不可缺少的组成部分，而不是两个语句间的分隔符号。例如：

```
i=i+1          (是表达式,不是语句)
```

```
i=i+1;          (是语句)
```

任何表达式都可以加上分号而成为语句，例如：

```
i++;
```

是一个语句，作用是使 i 的值加 1。又例如：

```
x+y;
```

也是一个语句，作用是完成 x＋y 的操作，它是合法的，但是并不把 x＋y 的值赋给另一变量，所以它并无实际意义。

表达式能构成语句是 C 语言的一个重要特色。其实函数调用语句也是属于表达式语句，因为函数调用(如 sin(x))也属于表达式的一种。只是为了便于理解和使用，才把函数调用语句和表达式语句分开来说明。由于 C 程序中大多数语句是表达式语句(包括函数调用语句)，所以有人把 C 语言称作“表达式语言”。

(4) **空语句**。下面是一个空语句：

```
;
```

即只有一个分号的语句，它什么也不做。空语句有时用来作流程的转向点(流程从程序其他地方转到此语句处)，也可用来作为循环语句中的循环体(循环体是空语句，表示循环体什么也不做)。

(5) **复合语句**。可以用{}把一些语句括起来成为复合语句。例如下面是一个复合语句：

```
{  z=x+y;
   t=z/100;
   printf("%f",t);
}
```

注意：复合语句最后一个语句中最后的分号不能忽略不写。

C 语言允许一行写几个语句，也允许一个语句拆开写在几行上，书写格式无固定要求。

本章介绍几种顺序执行的语句，在执行这些语句的过程中不会发生流程的控制转移。

3.4 赋值表达式和赋值语句

3.4.1 赋值表达式

1. 赋值运算符

赋值符号“＝”就是赋值运算符，它的作用是将一个数据赋给一个变量。如“a＝3”的

作用是执行一次赋值操作(或称赋值运算),把常量 3 赋给变量 a。也可以将一个表达式的值赋给一个变量。

2. 复合的赋值运算符

在赋值符“=”之前加上其他运算符,可以构成复合的运算符。如果在“=”前加一个“+”运算符就成了复合运算符“+=”。例如,可以有:

a+=3　　等价于　a=a+3

x*=y+8　　等价于　x=x*(y+8)

x%=3　　等价于　x=x%3

以“a+=3”为例来说明,它相当于使 a 进行一次自加 3 的操作。即先使 a 加 3,再赋给 a。同样,“x*=y+8”的作用是使 x 乘以(y+8),再赋给 x。

为便于记忆,可以这样理解:

① a+=b　　(其中 a 为变量,b 为表达式)

② a+=b　　(将有下划线的“a+”移到“=”右侧)

　|__↑

③ a=a+b　　(在“=”左侧补上变量名 a)

注意:如果 b 是包含若干项的表达式,则相当于它有括号。例如,以下 3 种写法是等价的:

① x%=y+3

② x%=(y+3)

　|__↑

③ x=x%(y+3)(不要错写成 x=x%y+3)

凡是二元(二目)运算符,都可以与赋值符一起组合成复合赋值符。有关算术运算的复合赋值运算符有+=,-=,*=,/=,%=。

C 语言采用这种复合运算符,一是为了简化程序,使程序精练;二是为了提高编译效率,能产生质量较高的目标代码。专业人员喜欢使用复合运算符,因为程序显得专业一点。对初学者来说,不必多用,首要的是保持程序清晰易懂。我们在此作简单的介绍,是为了便于阅读别人编写的程序。

本小节内容不要详细讲授,由读者自己阅读,有一定了解即可。

3. 赋值表达式

由赋值运算符将一个变量和一个表达式连接起来的式子称为“赋值表达式”。它的一般形式为

变量 赋值运算符 表达式

如“a=5”是一个赋值表达式。对赋值表达式求解的过程是:先求赋值运算符右侧的“表达式”的值,然后赋给赋值运算符左侧的变量。一个表达式应该有一个值,例如,赋值