# 第一章　程序设计和 C 语言

在开始学习程序设计时，初学者首先遇到的问题可能是：“什么是程序”？“什么是程序设计语言”？本章首先讨论这方面的问题，以期帮助读者在比较直观的基础上建立起对程序、程序设计、程序设计语言的基本认识。而后将简单介绍本书中讨论程序设计问题时所用的一种程序设计语言——C 语言，并通过一个简单实例介绍 C 语言程序的一些基本情况和有关概念。最后介绍了程序设计中必然要遇到的一些问题。

## 1.1　程序和程序语言

*程序*一词也来自生活，通常指完成某些事务的一种既定方式和过程。从表述方面看，可以将程序看着对一系列动作的执行过程的描述。日常生活中也可以找到许多“程序”实例。例如，一个学生早上起床后的行为可以描述为：

1，起床；
2，刷牙；
3，洗脸；
4，吃饭；
5，早自习；

这是一个直线形程序，是最简单形式的程序。描述这种程序的就是给出一个包含其中各个基本步骤的序列。如果按顺序实施这些步骤，其整体效果就完成了该项事务。

现在考虑另一个复杂些的过程：到图书馆借教学参考书。这一常见过程可以描述为：

1，进入图书馆；
2，查书目；
3，填写索书单；
4，交图书馆工作人员取书；
5，如果该书已经借出，读者可以有两种选择：
　5.1，回到第2步（进一步查找其他参考书的书目）；
　5.2，放弃借书，离开图书馆；
6，（工作人员找到了要借的书）办理借书手续；
7，离开图书馆。

这个程序比前一个复杂得多。可以看到，这一程序不是一个平铺直叙的动作序列，其中步骤更多，还出现了分情况处理和可能出现的重复性动作。

如果仔细探究这个实例，我们还可以可以看到，这一程序还可以进一步细化。例如可以找出许多上面描述中未处理的情况，例如：若查找图书目录时没有找到所需书目；填写好索书单后已经到了下班时间；借书时发现自己没有带借书证；工作人员查到该读者的借书册数已经达到限额，或发现该读者有逾期未还的图书，因此拒绝借出等等。

由这些现实生活中的例子，可以初步看到程序的一些直观特征。现实生活中有许多程序性活动，当我们身处其中时，通常需要按部就班地一步步完成一系列动作。对这种工作（事物、活动）过程的细节动作描述就是一个“程序”。

在一个程序描述中，总有一批预先假定的“基本动作”，这些基本动作是执行程序者能够理解和直接完成的。例如，在有关借书的程序描述中将“查图目”作为一个基本动作。如果执行者不知道如何查书目，那么，在给这种读者的程序描述中，就需要将“查书目”的动作进一步细分，描述查书目的细节过程。

一个程序总有开始与结束。在执行此程序的过程中，动作者（无论是不是人）需要按照程序的描述执行一系列的动作。在达到结束位置时工作就完成了。

本书中将要深入讨论的计算机程序同样具有这些特征。

### 程序与程序设计

日常生活中程序性活动的情况与计算机里的程序执行很相似。这一情况可以帮助我们理解计算机的活动方式。当然，人们日常生活中的程序性工作中有更多变数，许多事情并不要求完全按程序做，可以有许多“灵活性”。而计算机对程序的执行则完全是严格而一丝不苟的，一步步按程序中的指令办事，一点“商量”的余地也没有。

计算机是人类发明的一种自动机器，它能完成的工作就是计算。计算机的最基本功能是可以执行一组基本操作，每个操作完成一件很简单的计算工作，例如整数的加减乘除运算等等。为使计算机能按人的指挥工作，每种计算机都提供了一套指令，其中的每一种指令对应着计算机能执行的一个基本动作。

作为一种看得见、摸得着的物理实体，计算机的基本原理很简单，其最本质特征是它们不仅能按指令工作，而且能自动地按程序（作为计算机能执行的基本动作序列）工作。因此，人与计算机交流的基本方式就是提供要求它执行的程序。在命令计算机去执行某个程序之后，计算机就会按照程序的规定，一丝不苟地执行其中的指令，直至程序结束。

计算机是一种通用的计算机器，加上一个或一组程序后，它就会变为处理某个专门问题、完成某种特殊工作的专用机器。这种通用性与专用性的统一非常重要。这样，一方面，计算机可以在大工厂里采用现代化生产方式大量制造；另一方面，通过运行不同程序，一台计算机可以在不同时候处理不同问题，甚至同时处理许多不同的问题。这就是计算机威力的真谛。人们描述（编制）计算机程序的工作被称为*程序设计*或者*编程*，这种工作的产品就是程序。由于计算机的本质特征，从计算机诞生之初就有了程序设计工作。

今天，计算机的发展及其在各领域的广泛应用，它对人类社会生活各方面的深刻影响已经是人所共知的事实了。计算机之所以能产生这样大的影响，其原因不仅在于人发明并大量制造了这样一种令人敬畏的奇妙机器，更重要的是人们为计算机开发出了数量宏大、五彩缤纷、能指挥计算机完成各种简单或复杂工作的程序。目前正在使用的计算机没有多少种，而正是数量繁多、功能丰富多彩的程序给了计算机无穷无尽的“生命力”。

由上面介绍还可以看到程序和程序描述的另外一些问题。例如，对于上面学生借书的例子，我们提出了许多需要考虑的细节。如果需要用计算机去处理问题，写程序时就必须精确描述所有动作的细节过程，不能有一点含糊其词的地方。

### 程序设计语言及其发展

要说明在一个程序的进行中需要做些什么，就需要仔细给出这一程序性活动的一步步的细节过程，需要描述程序运行中的各种动作及其执行的顺序。为做到这些，就需要有某种适当的描述方式。一套描述方式就形成了一个*语言*。

语言一词通常指人生活工作中使用的自然语言，如汉语、英语等。这些语言随着人类发展进步而自然形成，是人互相间交流信息的工具和媒介。人们用口头语言向别人传播见闻、表达看法和想法。用书面语言写文章、书籍，以实现更大范围中的信息交流。在前面所描述的现实生活中的程序实例中，我们就是用汉语作为描述程序的语言，所描述的程序是为了给人看，要人去做的。

为了与计算机交流，指挥它工作，同样需要有与之交流的方式，需要一种意义清晰、人用起来比较方便、计算机也能处理的描述方式。也就是说，需要有描述程序的合适语言。可供人编程序用的语言就是*程序设计语言*，这是一类人造语言。程序设计语言也常被称为*编程语言*，本书中常常简称为*程序语言*，在上下文清楚之处简称为*语言*。

有人可能说：小学生就学数学，数学的一个基本部分是计算，小学生已经会用数学方式（或说，用数学*语言*）描述计算过程，程序设计语言有什么特殊之处吗？确实有！程序语言

的一个突出特点就在于不仅人能懂得和掌握它，能用它描述所需的计算过程，而且计算机也可以“懂得”它，可以按程序语言给出的关于计算过程的描述去行动，完成人所需要的计算工作。程序设计语言是人描述计算的工具，也是人与计算机交流信息的媒介：通过用程序语言写程序，人能指挥计算机完成各种特定工作，完成各种计算。

计算机诞生之初，人们只能直接用二进制形式的机器语言写程序。对于人的使用而言，二进制的机器语言很不方便，用它书写程序非常困难，不但工作效率极低，程序的正确性也难以保证，发现有错误也很难辨认和改正。下面是一台假设计算机上的指令系列：

```
0000000010000000001000   -- 将单元1000的数据装入寄存器0
0000000010001000001010   -- 将单元1010的数据装入寄存器1
0000010100000000000001   -- 将寄存器1的数据乘到寄存器0原有数据上
0000000010001000001100   -- 将单元1100的数据装入寄存器1
0000010000000000000001   -- 将寄存器1的数据加到寄存器0原有数据上
0000001000000000001110   -- 将寄存器0里的数据存入单元1110
```

这里想描述的是计算算术表达式 a × b ＋ c（这里的符号 a、b、c 分别代表地址为 1000、1010 和 1100 的存储单元），而后将结果保存到单元 1110 的计算过程（程序）。

一个复杂程序里的指令可能有成百万、成千万条或者更多，程序中的执行流程错综复杂，在二进制机器指令的层面上理解复杂程序到底做了什么，很容易变成人力所根本不能及的事情。为缓解这一问题，人们发展了符号形式的，使用相对容易些的汇编语言。用汇编语言写的程序需要用专门软件（汇编系统）加工，翻译成二进制机器指令后才能在计算机上使用。下面是用某种假想的汇编语言写出的程序，它完成与上面程序同样的工作：

```
load 0 a   -- 将单元a的数据装入寄存器0
load 1 b   -- 将单元b的数据装入寄存器1
mult 0 1   -- 将寄存器1的数据乘到寄存器0原有数据上
load 1 c   -- 将单元c的数据装入寄存器1
add  0 1   -- 将寄存器1的数据加到寄存器0原有数据上
save 0 d   -- 将寄存器0里的数据存入单元d
```

汇编语言的每条指令对应于一条机器语言指令，但采用了助记的符号名，存储单元也用用符号形式的名字表示。这样，每条指令的意义都更容易理解和把握了。但是，汇编语言的程序仍然完全没有结构，仅仅是许多这样的指令堆积形成的长长序列，是一团散沙。因此，复杂程序作为整体仍然难以理解。

1954 年诞生了第一个高级程序语言 FORTRAN，宣告程序设计新时代的开始。FORTRAN 采用完全符号化的描述形式，用类似数学表达式的形式描述数据计算。语言中提供了有类型的变量，作为存储的抽象模型。此外它还提供了一些流程控制机制，如循环和子程序等。这些高级机制使编程者可以摆脱许多具体细节，方便了复杂程序的书写，写出的程序也更容易阅读，有错误也更容易辨认和改正。FORTRAN 语言诞生后受到广泛欢迎。

### 高级语言及其实现

高级程序语言更接近人所习惯的描述形式，更容易被接受，也使更多的人能（并乐于）加入程序设计活动中。用高级语言书写程序的效率更高，这使人们开发出更多应用系统，反过来又大大推动了计算机应用的发展。应用的发展又推动了计算机工业的大发展。可以说，高级程序设计语言的诞生和发展，对于计算机发展到今天起了极其重要的作用。

从 FORTRAN 语言诞生至今，人们已提出的语言超过千种，其中大部分只是试验性的，只有少数语言得到了广泛使用。随着时代发展，今天绝大部分程序都是用高级语言写的。人们也已习惯于用程序设计语言特指各种高级程序语言了。

在高级语言（例如 C 语言）的层面上，描述前面同样的程序片断只需一行代码：

```
d = a * b + c;
```

这表示要求计算机求出等于符号右边的表达式，而后将计算结果存入由 d 代表的存储单元中。这种表示方式与人们所熟悉的数学形式直接对应，因此更容易阅读和理解。高级语言程

序中完全采用符号形式，使人可以摆脱难用的二进制形式和具体计算机的细节。此外，高级语言中还提供了许多高级的程序结构，供编写程序用于去组织复杂的程序。与机器语言和汇编语言的程序相比，情况确实大大改观了。

当然，计算机也不能直接执行高级语言描述的程序。人们在定义好一个语言之后，还需要开发出一套实现这一语言的软件，这种软件被称作*高级语言系统*，也常被说成是这一高级语言的*实现*。在研究和开发高级语言的工作过程中，人们也研究了各种实现技术。高级语言的基本实现技术是编译和解释，下面是两种方式的简单介绍：

1. 采用编译方式实现高级语言：人们首先针对具体语言（例如 C 语言）开发出一个翻译软件，其功能就是将采用该种高级语言书写的程序翻译为所用计算机的机器语言的等价程序。用这种高级语言写出程序后，只要将它送给翻译程序，就能得到与之对应的机器语言程序。此后，只要命令计算机执行这个机器语言程序，计算机就能完成我们所需要的工作了。
2. 采用解释方式实现高级语言：人们首先针对具体高级语言开发一个解释软件，这个软件的功能就是读入这种高级语言的程序，并能一步步地按照程序要求工作，完成程序所描述的计算。有了这种解释软件，只要直接将写好的程序送给运行着这个软件的计算机，就可以完成该程序所描述的工作了。

目前在实际程序设计工作中，最常用的是第一种实现方式，本书后面还会进一步介绍这种实现方式的一些具体情况。

随着计算机科学技术的发展，人们不断提出新的程序语言，老的语言被逐渐淘汰。仍在使用的老语言也在急剧变化。以 FORTRAN 语言为例，它在过去 40 多年里经过了四五次大改版，目前最新版本（FORTRAN 90）与开始的 FORTRAN 相比早已面目全非了。其他历史较长的语言也都如此。推动语言发展的因素很多，一个重要原因是人们对程序工作的新认识。随着程序设计实践越来越丰富，人们对程序设计工作应该怎样做，需要什么东西去描述等不断产生新认识。推动语言发展的另一原因是应用的发展。新应用领域常对描述工具提出新要求，这些认识和要求促使人们改造已有的语言，或者提出新语言。

世界上目前使用较广的语言有 FORTRAN、C、C++、PASCAL、Ada、Java 等，这些语言通常被认为是常规语言，因为它们有许多共同性质。还有一些语言比较特殊，在形式、编程方式等方面与常规语言差别很大，互相间也常常大相径庭。这些非常规语言各有各的特点或应用领域，甚至有特殊的使用人群。这类语言如 LISP、Smalltalk、PROLOG、ML 等。它们虽然不如常规语言使用广泛，但也非常重要，都曾在程序语言或计算机的发展历史上起过（有些仍在起着）极其重要的作用。

### 程序及其抽象层次

对于程序的描述而言（无论是前面给出的日常生活实例中的非形式的“程序”，还是希望计算机去执行的程序），还有几个重要的问题值得提出：

可以写在程序里的基本“指令”包括哪些？洗脸、查书目等是前面日常生活中的“程序”里的基本动作。在编写需要计算机去执行的程序时，其中的基本动作应该是计算机能完成的事项。例如，要写前面那样的二进制代码形式的程序，就需要按照具体计算机指令系统的规定写出一条条指令。用一种高级语言写程序时，同样必须基于这种语言所支持的基本功能（基本命令等等）。学习用高级语言写程序，一个最基本的方面就是了解语言所提供的基本功能，了解它们的描述形式和所完成的操作效果。

对于描述程序的语言有什么要求？描述前面有关日常生活的“程序”时采用的是汉语，是我们日常用于人际交流的“自然语言”。自然语言词汇丰富，有极强的表达能力。但在许多地方要依赖于接受者的知识和常识。如果，要求孔子按照前面给出的程序到图书馆去借书，

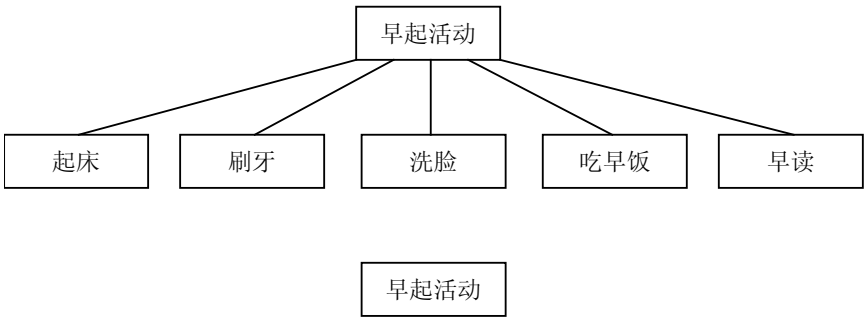
他不可能理解其中的动作，也无法执行它去把书借来。自然语言的描述常常很不精确，许多疏漏需要靠接受者用自己的知识去填补。这种情况可以大大提高信息的传递效率，但也带来误解的可能性。送给计算机使用的程序必须采用计算机能处理的记法形式，其描述工具（语言）必须是精确的，无歧义的。程序设计语言都必须满足这些要求。

一个程序可能在不同的层次上描述。看看有关刷牙的例子。前面只用一个词描述这一动作。但如果仔细想想，刷牙也是一个很复杂的过程。例如，我们还可以进一步将其分解描述为取杯子、装水、取牙刷、挤牙膏、漱口、刷牙、清洗牙齿等一系列细节动作。还可以进一步将这一层面上的每个动作分解为一系列的肌肉伸缩动作。

应当将程序的细节分解到哪个层次，一方面要看程序语言所提供的基本功能。此外，程序的描述方式也要照顾到人的需要。复杂的程序可能需要成千上万，甚至成百万或千万行高级语言代码。简单地在高级语言基本层面上描述程序同样会显得层次太低，使程序的意义难以把握，难以保证它能实现所预想功能，难以修改程序去满足新的需要，如此等等。因此，在开发复杂的程序时，我们需要提供更高的描述层次，将程序的功能在各个层次上分解描述。就像我们看到极长的一系列有关肌肉伸缩动作的描述，很难理解这里所做的是刷牙一样。随着程序变得越来越复杂，其组织结构问题也变得更加重要了。

还是用一个生活中的例子来说明问题。对于学生早上起床后的活动，首先应该在很高的层次上描述，就像前面所给出的：

1，起床；
2，刷牙；
3，洗脸；
4，吃饭；
5，早自习；



早起活动

这样就把一个复杂的程序分解成为若干相对简单一些的部分了。如果需要进一步细节化，那么我们就降到下一个细节层次，将一个高层动作分解为一系列低层的基本动作。例如，可能将“吃早饭”这一高层动作分解为下面动作序列：

拿饭碗；
去食堂；
排队买饭；
吃饭；
刷碗；
离开食堂；

必要时再做进一步分解。例如，将“排队买饭”分解为“排队、选饭、选菜、付款”等。在这种分解描述的过程中，我们还应该保留前面构造出的抽象描述的层次。这种层次结构不但有利于人们理解程序的细节过程，也有利于发现程序中的错误，还能使所得程序易于根据需要去修改。例如，学校的食堂改为快餐份饭，由于整个程序被按照分解为一些具有逻辑独立性的部分，修改起来也就更容易了。

编程序时所需要掌握的恰恰就是这种工作方式。我们需要从问题的要求出发，从高层开始设计程序，并逐步分解程序功能。当将程序所需功能分解到一定的细节程度之后，就可以借助于程序语言的结构，描述程序工作中的细节步骤了。本书将不断讨论这方面的问题。在学习程序设计的过程中，也必须学习分析和构造程序的正确方法。

### 具体语言和程序设计

本章下面几节将从日常生活的实例和一般性介绍转到更具体的问题。由于本书的主要目标是讨论程序设计问题，因此就需要选用一种合用的程序设计语言。为了更好地讨论程序设计中的各种问题，各种基本程序设计教科书都选用某种高级语言作为工作媒介。本书选用 C 语言作为讨论程序设计的工作语言，因为 C 语言能较好地服务于我们的目标。

在后面各章中，我们将逐步展开有关程序设计的讨论：从最基本的数据描述和简单表达式开始，讨论如何写出最简单的一步程序。而后讨论程序的基本流程结构，以及如何利用这些结构解决更复杂的计算问题。书中很早就开始介绍程序的组织问题，以及 C 语言为此而提供的基本机制——函数。在研究了许多基本程序设计问题和技术之后，书中的讨论转向数据的组织问题，因为在更复杂的计算问题中，需要由程序处理的数据也更加复杂多样，需要采用适当的方式将它们组织起来。

在开始对程序设计和 C 语言的系统讨论之前，我们将用本章下面部分介绍一些基本情况，作为后面系统讨论的导引。第 1.2 节将简单介绍 C 语言的历史发展，第 1.3 节从一个简单程序例子出发，介绍 C 语言的一些基本情况，第 1.4 节介绍程序设计过程中的一些重要概念和问题。第 1.4 节的内容具有一般性，完全理解有关内容需要有一定的程序设计实际经验。将这一部分放在这里，只是希望读者对有关情况有一点初步了解。

## 1.2 C 语言简介

C 语言是贝尔实验室 Dennis Ritchie 在 1973 年设计的一种程序设计语言，其目的是用于写操作系统和系统程序，初期用在 PDP-11 计算机上写 UNIX 操作系统。70 年代后作为 UNIX 的标准开发语言，C 语言随着 UNIX 系统流行而得到越来越广泛的接受和应用，80 年代后它被搬到包括大型机、工作站等的许多系统上，逐渐成为开发系统程序和复杂软件的一种通用语言。随着微机的蓬勃发展、处理能力的提高和应用的日益广泛，越来越多的人参与微机应用系统的开发工作，需要适合开发系统软件和应用软件的语言。C 语言能较好满足人们的需要，因此在微机软件开发中得到日益广泛的应用，逐渐成为最常用的系统开发语言之一，被人们用于开发微型机上的各种程序，直至非常复杂的软件系统。

在使用最多的以 Intel 及其兼容芯片为基础的微机上，也有许多性能良好的商品 C 语言系统可用。包括 Borland 公司早期的 Turbo C 和后续 Borland C/C++系列产品；Microsoft（微软）公司的 Microsoft C 和后续 Visual C/C++系列产品。还有其他 C/C++语言系统产品，使用较广的有 Watcom C/C++和 Symantic C/C++等。此外还有许多廉价的和免费的 C 语言系统。其他微型机也有多种 C 语言系统。各种工作站系统大都采用 UNIX 和 LINUX，C 语言是它们的标准系统开发语言。各种大型计算机上也有自己的 C 语言系统。

### C 语言的特点

C 语言之所以能被世界计算机界广泛接受是由于其自身的特点。总体上说，因其设计把直到 70 年代人们对于程序语言的认识和开发复杂系统程序（例如操作系统等）的需要成功地结合起来。C 语言的主要特点包括：

C 语言比较简单，是一个比较小的语言。学习时入门相对容易，知道很少东西就可以开始编程。C 语言很好地总结了其他语言提出的程序库概念，把程序设计中需要的许多功能放在程序库（称为*标准函数库*）里实现，如输入输出功能等。这就使语言本身比较简单，编译程序的实现比较容易。人们早已用 C 语言写了它自己的编译程序，这种程序很容易移植到各种不同计算机上，促进了 C 语言的传播。

C 语言提供了丰富的程序机制，包括各种控制机制和数据定义机制，能满足构造复杂程

序时的各种需要。方便易用的函数定义和使用机制使人可以把复杂程序分解成一个个具有一定独立性的函数，以分解程序的复杂性，使之更容易控制和把握。

C 语言提供了一套预处理命令，支持程序或软件系统的分块开发。利用这些机制，一个软件系统可以较方便地先由几个人或几个小组分别开发，然后再集成，构成最终系统。这种工作方式对于开发大软件系统是必需的。人们用 C 语言开发了许多规模很大的系统。

C 语言的另一特点是可以写出效率很高的程序。人们之所以在一些地方继续用汇编语言，就是因为高级语言写出的程序效率低一些。这样，开发效率要求特别高的程序时就只能使用汇编语言。C 语言的基本设计使得用它开发的程序具有较高的效率，它还提供了一组比较接近硬件的低级操作，可用于写较低级、需要直接与硬件打交道的程序或程序部分。这些使 C 语言常被用作汇编语言的“替代物”，大大提高了开发低层程序的效率。

C 语言的工作得到世界计算机界的广泛赞许。一方面，C 语言在程序语言研究领域具有一定价值，它引出了不少后继语言，还有许多新语言从 C 语言中汲取营养，吸收了它的不少东西。另一方面，C 语言对计算机工业和应用的发展也起了很重要的推动作用。正是由于这些情况，C 语言的设计者获得世界计算机科学技术界的最高奖——图灵奖。

### C 语言的发展和标准化

在设计 C 语言时，设计者主要把它作为汇编语言的替代品，作为自己写操作系统的工具，因此更多强调的是灵活性和方便性。语言的规定很不严格，可以用许多不规矩的方式写程序，因此也留下了许多不安全因素。使用这样的语言，就要求编程序者自己注意可能的问题，程序的正确性主要靠人来保证，语言的处理系统（编译程序）不能提供多少帮助。随着应用范围的扩大，使用 C 语言的人越来越多（显然其中大部分人对语言的理解远不如设计者），C 语言在这方面的缺点日益突出起来。由此造成的后果是，人们用 C 语言开发的复杂程序里常常带有隐藏很深的错误，难以发现和改正。

随着应用发展，人们更强烈地希望 C 语言能成为一种更安全可靠、不依赖于具体计算机和操作系统（如 UNIX）的标准程序设计语言。美国国家标准局（ANSI）在 80 年代建立了专门小组研究 C 语言标准化问题，这个工作的结果是 1988 年颁布的 ANSI C 标准。这个标准被国际标准化组织和各国标准化机构接受，同样也被采纳为中国国家标准。此后人们继续工作，1999 年通过了 ISO/IEC 9899:1999 标准（一般称为 C99）。这一新标准对 ANSI C 做了一些小修订和扩充。有关情况将在本书的最后做一些介绍。

语言改造非常困难。虽然人们已认识到原来 C 语言中的一些东西不好，最终应该丢掉。但已有的东西，主要是新标准出现前人们已开发的各种程序和软件，是一笔巨大财富，不能轻易丢掉，彻底改造要耗费极大人力和物力，也不可能做到。另外，一批老用户已养成习惯，不可能在一朝一夕改变。因此，即使想建立一个新标准，也要尽可能保持与原形式的兼容性，ANSI C 标准基本上容许原形式的 C 程序，作为对现实的让步。但新标准中强调：旧事物终将被抛弃，希望写程序的人尽量不要再使用它们。

今天学习 C 语言和程序设计，理所当然应该采用新的形式，不应该学习那些过时的东西。原因主要有两条：这些旧东西终归将被抛弃，养成使用它们的习惯将来还要改，那时将更加费时费力，也毫无意义；这些过时的东西确实不好，虽然有时用它们能少写几个字符，但往往会阻碍编译系统对程序的检查。人很容易犯错误，在从事写程序这种复杂工作时尤其如此。阻止编译检查就是拒绝计算机帮助，其实际后果无法预料，可能代价惨重，后来要为程序中实际存在的隐藏错误耗费更多的时间和精力。

### 教材和参考书

关于 ANSI C 语言的参考书，除了 ANSI C 语言标准本身外，最重要的是 C 语言设计者 B. W. Kernighan 和 D. M. Ritchie 合著的《C 程序设计语言（第二版）》。英文版已由清华出版

社作为“大学计算机教育丛书（影印版）”出版。书中不仅包括有关 C 程序设计的讨论，还附了许多材料，包括改写的语言参考手册和对标准库的详细介绍。另有一个附录总结了 ANSI C 与过去的 C 之间的差异，供熟悉老式 C 语言的人参考。但是该书假定读者在阅读前已经有程序设计经验，并至少熟悉一种程序语言，因此不适合作为初学程序设计的教材。该书正式中译本已于 2001 年由机械工业出版社出版。

本书中的讨论都基于 ANSI C 标准语言，各章里提出许多使用 C 语言写程序时应注意的问题，其中有些就是针对 C 语言的过时形式。目前市场上仍然有不少书籍以老的 C 语言形式作为程序例子，其中有些说法和解释也已过时，请读者注意。

与学习写程序有关的问题不仅包括采用 C 语言的什么形式，还包括理解运用人们在几十年的程序设计工作所总结出的许多经验，许多情况下的具体程序写法（所谓*习惯用法*），程序书写的形式等。对阅读本书的读者提出的建议是：在学习程序设计的过程中，不仅要学习程序语言和程序设计的方法，而且应该注意学习如何写程序，应该注意养成良好的程序设计习惯。这些也是学习程序语言与程序设计过程中应当特别重视的一个方面。书中许多地方提出了对这方面的建议，希望读者一定要重视。

## 1.3 一个简单的 C 程序

本书将从下章开始详细讨论各种 C 语言结构，讨论程序设计的各方面情况和问题。在开始这些讨论之前，先请读者看一个简单的程序例子，看看用 C 语言写出的程序是什么样子。然后从它出发解释一些程序开发中的问题。下面是一个简单程序：

```
#include <stdio.h>

int main () {
    printf("Good morning!\n");
    return 0;
}
```

用 C 语言写的程序简称为 C *程序*。上面这个简单程序可分为两个基本部分：第一行是个特殊行，说明程序用到 C 语言系统提供的标准功能，为此要参考标准库文件 `stdio.h`，有关细节在第五章介绍。空行下的几行是程序基本部分，描述程序所完成的工作。该程序的意义就是产生一行输出“`Good morning!`”，详细情况在下面讨论。

### C 程序的加工和执行

C 语言是高级程序语言，用 C 语言写出的程序通常称作*源程序*，人容易使用、书写和阅读，但计算机却不能直接执行，因为计算机只能识别和执行特定二进制形式的机器语言程序。为使计算机能完成某个 C 源程序所描述的工作，就必须首先把这个源程序（如上面简单例子）转换成二进制形式的机器语言程序，这种转换由 C 语言系统完成。由源程序到机器语言程序的转换过程称为“C 程序的加工”。每个 C 语言系统都包含了加工 C 源程序的功能，包括“编译程序”、“连接程序”等，系统里还可能有一些其他程序或功能模块。

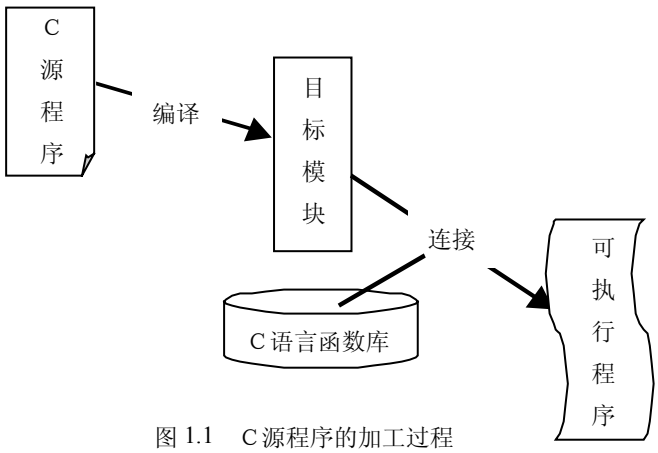



图 1.1 C 源程序的加工过程

程序加工通常分两步完成：第一步，由编译程序对源程序文件进行分析和处理，生成相应的机器语言目标模块，由目标模块构成的代码文件称为*目标文件*。目标文件还不能执行，因为其中缺少 C 程序运行所需要的一个公共部分，C 程序的*运行系统*。此外，一般 C 程序里都要使用函数库提供的某些功能。例如前面例子用到标准函数库的一个输出函数（printf 是该函数的名字）。为构造出完整的可以运行的程序，还需要第二步加工，*连接*。这一工作由连接程序完成，将编译得到的目标模块与其他必要部分（运行系统、函数库提供的功能模块等）拼装起来，做成可执行程序。图 1.1 说明了 C 程序的基本加工过程。

对前面简单 C 程序例子进行加工后，就能得到一个与之对应的、可以在计算机上执行的程序。启动运行这个可执行程序，将能看到它的执行效果。这个程序的执行将得到一行输出，通常显示在计算机屏幕上，或者图形用户界面上的特定窗口里：

```
Good morning!
```

如果修改程序，将双引号里一串字符换成其他内容，就可以让它输出那些内容。例如：

```
#include <stdio.h>

int main () {
   printf("Hello, world!\n");
   return 0;
}
```

这一程序加工后执行，就会输出：

```
Hello, world!
```

C 程序加工过程的启动方式由具体 C 语言系统确定，具体情况请查看有关系统的手册。有时需要直接用操作系统命令形式启动各种基本加工工作（启动编译系统、连接系统等）。先用一个命令要求编译源程序，再用另一个命令做连接。其中除了要把源程序文件名作为命令参数外，还常常需要给一些其他参数。这些命令的书写形式比较复杂，使用不太方便。此外，为了输入、编辑和修改程序，还需要另用一个编辑系统。

### 集成开发环境

为方便程序开发，随着技术进步，人们开发了一类称作*集成式程序开发环境*（Integrated Development Environment，IDE）的软件，许多厂家已把 IDE 作为其语言处理系统的基本部分。IDE 把编程过程中需要的所有软件集成在一起统一管理和使用。IDE 系统一般采用窗口菜单技术，提供了专供编程用的编辑环境，通过菜单方式提供编译、连接、以及启动可执行程序的命令。利用 IDE 写程序，开发过程中的各种工作都很方便，能提高编程工作的效率。各种微机 C 语言系统都包括很完善的 IDE。需要提醒读者，即使编程序时使用的是 IDE，上面讲述的程序加工过程并未改变，不同的只是启动加工命令的方式。

随着时间推移，IDE 也在不断改进和发展，逐渐发展成一类复杂软件系统。使用好这种系统也需要学习，掌握了 IDE 的功能，才能更好地发挥其作用。IDE 虽然能使编程工作更方便，但它没有也不可能改变这个工作的实质。编好程序的最基本因素仍然是人。

### 程序格式

实际的 C 程序可能比前面的简单例子长得多。一般说，一个 C 程序由一系列可打印（可显示）字符构成，人们一般用普通编辑器编写程序，或者用专门的程序开发系统写程序，修改程序。组成程序的字符序列通常被按照人阅读方便的形式分为一些行（就是在字符序列中插进一些换行符），每行长度不必相同。注意，上面把花括号内的部分看作下一层次内容，后退几格写出，就是希望程序的表面形式能较好反映程序的内在层次结构。

C 语言是一种“自由格式”语言，除了若干简单限制外，写程序的人完全可以根据自己的想法和需要选择程序格式，确定在哪里换行，在哪里增加空格等。这些格式变化并不影响

程序的意义。没规定程序格式并不说明格式不重要。程序的一个重要作用是给人看，首先是写程序的人自己要看。对于阅读而言，程序格式非常重要。在多年程序设计实践中，人们在这方面取得了统一认识：由于程序可能很长，结构可能很复杂，因此程序必须采用良好格式写出，所用格式应很好体现程序的层次结构，反映各个部分间的关系。

关于程序格式，人们普遍采用的方式是：（1）在程序里适当加入空行，分隔程序中处于同一层次的不同部分；（2）同层次不同部分对齐排列，下一层次的内容通过适当退格（在一行开始加空格），使程序结构更清晰；（3）在程序里增加一些说明性信息，这方面情况后面介绍。上面程序例子的书写形式符合这些看法。

开始学习程序设计时就应养成注意程序格式的习惯。虽然对开始的小程序，采用良好格式的优势并不明显，但对稍大一点的程序，情况就不一样了。有人为了方便，根本不关心程序的格式，想的只是少输入几个空格或换行，这样做结果是使自己在随后的程序调试检查中遇到更多麻烦。所以，这里要特别提醒读者：注意程序格式，从一开始写最简单的程序时就注意养成好习惯。目前多数程序设计语言（包括 C 语言）都是自由格式语言，这就使人能够方便地根据自己的需要和习惯写出具有良好格式的程序来。

## 1.4　程序开发过程

本节讨论程序开发过程，包括程序调试（Testing）和排除错误（简称排误，Debugging）等方面的问题。调试和排误是程序实现的必经阶段。这节要讨论许多问题。在读者开始学习程序设计时，下面讨论的一些情况可能难以完全明白，因为现在还缺乏程序设计实践。但是这些问题确实需要说明。我们把有关讨论集中在这里，希望读者能在学习了后面章节，做了些程序后再回来重读这些说明，这样反复几次，事情就能弄清楚了。

**程序的开发过程**

用计算机解决问题的过程可以用图 1.2 描述，这种过程大致是：

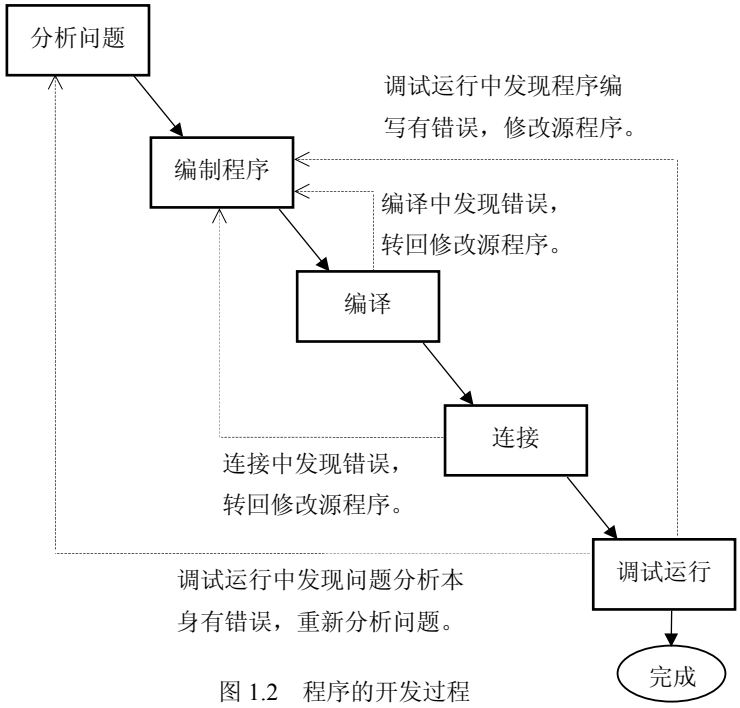



图 1.2　程序的开发过程

1. 分析问题，设计一种解决问题的途径。
2. 根据所设想的解决方案，用编辑系统（或 IDE）建立程序。
3. 用编译程序对源程序进行编译。正确完成就进入下一步；如发现错误，就需要设法确定错误，回到第 2 步去修改程序。
4. 反复工作直到编译能正确完成，编译中发现的错误都已排除，所有警告信息都已处理，一些排除，其余已弄清不是错误，这时就可以做程序连接。如果连接发现错误，就需返回前面步骤，修改程序后重新编译。
5. 正常连接产生了可执行程序后，可以开始程序的调试执行。此时需要用一些实际数据考查程序的执行效果。如果执行中出了问题，或发现结果不正确，那么就要设法确定错误

原因，回到前面步骤：修改程序，重新编译，重新连接，等等。
重复进行上述过程，直到确信程序正确为止。在上面论述中的一些术语见下面解释。

### 程序错误

关于排除程序错误的术语 Debugging 还有一个故事。在美国计算机发展早期，有一天一台计算机出故障不能运行了。经仔细检查，人们发现计算机里有一个被电流烧焦的小虫（bug），它造成了电路短路，是这次故障的祸根。从此，检查排除计算机故障的工作就被称为 Debugging，就是“找虫子”。后来人们也这样看待和称呼检查程序错误的工作。

实际上，对程序设计而言这个词并不贴切。因为程序中的错误都是编程者所犯的错误，并没有其他客观原因，也没有虫子之类的小东西捣乱。学习程序设计首先应该认清这一情况。所谓排除程序错误，也就是排除自己在程序设计过程中所犯的错误，或说是消除自己写在程序里的错误。初学者在遇到程序问题时，往往倾向于认为所用系统或者计算机有问题，常会说“我的程序绝没有错，一定是系统的毛病”。而有经验的程序员都知道，如果程序出了错，基本上可以肯定是自己的错，需要仔细检查程序去排除它们。

程序的错误可以分为两大类，一类是程序书写形式在某些方面不符合程序语言要求。对于这类错误，语言系统在加工程序的过程中能够检查。另一类是程序书写形式本身没错，加工过程能正常完成，产生可执行程序，但或是程序执行中出了问题，或是计算结果（或执行效果）不符合我们的需要。排除程序错误的目的就是要消除这两类错误。

### 程序加工，有关错误的排除

如果语言系统在程序加工过程中能查出错误，编译程序或连接程序就会产生出错信息。通常语言处理程序每发现一个错误就产生一个错误信息行，指明发现错误的位置（例如发现错误的源程序行编号等）和所确认的错误类型，信息行里还可能包括其他信息，供人们检查程序时参考。不同 C 语言系统在检查错误的能力、产生出错信息的形式等方面可能有许多不同。但无论如何，每当系统给出了出错信息，我们都应该仔细阅读，检查错误信息所指定位置附近的源程序代码，找到真正的错误原因并予以排除，然后再继续下去。

编译程序能发现的错误（编译错误）主要有两类：

1. 局部语法错误，如缺少必要的符号（常见的如缺少分号、括号），组合符号拼写不正确等。对这些错误，编译程序都能给出发现错误的位置，但给出的错误原因有时未必正确。编译程序是一个个字符地检查源程序，如果检查到某一点能确定程序有问题，就把这里作为发现错误的位置。因此，源程序中实际错误或是出现在编译程序指定的出错位置，或是在这个位置之前。应当从这里开始向前检查，设法确定错误原因。有些错误可能到很远以后才被编译程序发现，也就是说，实际错误可能出现在编译程序所指位置前面很远的地方。还有一个问题值得注意：有时一个实际错误会导致编译程序产生许多出错信息行。这是因为源程序错误可能使编译程序进入某种非正常的状态，致使它参数一系列出错信息。经验告诉我们，排除程序错误的基本原则是：每次编译后集中精力排除编译程序发现的第一个错误。如果无法确认后面错误，就应当重新编译检查。排除一个错误可能消除掉许多出错信息行。
2. 程序里上下文关系方面的错误。程序里的许多东西有前后照应问题，例如，要用的东西必须先有定义。如果编译中发现某些东西无定义，就会指出这个错误。这种错误也常因为名字拼写有误，有时确实是忘记定义了。这些都比较容易检查和纠正。

编译程序发现错误时总能提供出错位置信息，这种信息非常重要。为帮助人发现程序中的问题，许多编译程序还做一些超出语言定义的检查。如果发现程序有可疑之处，它会提供警告信息（warning）。这种信息未必表示程序有错误，但也很可能是真有错误。经验告诉我

们，对警告信息绝不能掉以轻心，警告常常预示着隐藏较深的实际错误，必须认真地一个个弄清其原因。只有那些能确认没有问题的警告，才可以让它们留在那里。

连接程序也可能检查出一些错误，称为*连接错误*。连接错误都是有关目标模块间，或目标模块与程序库、运行系统之间关系方面的问题。例如，若在前面简单程序里不慎把 `main` 写成 `mian`，编译时不会发现问题，连接时会得到一个错误信息，意思是说：连接中没找到名字为 `main` 的函数。出问题的原因是 C 程序运行系统要用这个函数去启动程序。而在我们的程序里没有这个函数（因为名字写错了）。连接程序发现的错误通常都与名字有关，此时它只能指出发现了关于哪个名字的问题，却无法指出有关错误在源程序里出现的位置。对于小程序，这种错误很容易排除。程序大时可以利用编辑器的字符串查找功能。

### 程序运行中的错误

完成了程序加工，生成可执行程序之后，下步工作应是试验性地运行程序，检查运行情况，看它是否正确实现了所需功能。程序运行中也可出错，出错情况可能有多种：

1. 程序执行中可能违反了系统环境的基本要求，例如试图执行某种非法操作。这时会出什么问题，完全由程序及其运行所在的操作系统决定。在检查严格的系统里，这种程序通常会因为违规而被强行终止，操作系统可能给出出错信息。在控制不严或者完全没控制的系统（例如微机的 DOS 系统）里，程序的这种问题多半会导致系统死机，或出现其他不正常现象。这种程序错误往往很隐蔽，需要仔细检查才能发现。在写 C 程序时不注意，就容易写出这种程序，这是 C 语言的一个重要缺点。在本书后面的讨论中，也特别注意提醒读者在哪些地方需要小心。
2. 由于编程错误，致使程序在执行中进入某种不能结束的状态，一般称“进入死循环”，也就是无休止地重复执行某段指令而无法停止。这种程序在启动后长时间没有反应，或是在执行中不断输出类似信息（如果死循环里有输出命令）。当然，长时间无反应未必说明程序进入了死循环。如果程序里要求键盘输入，执行到达这里程序就会进入等待，直到人由键盘输入信息后才继续下去，这是正常情况。另一方面，有的程序确实需要运行较长时间。对程序是不是真正进入了死循环，还需要仔细分析和判断。
3. 程序在执行中因为出现某些情况，无法继续下去而停止。这时会给出运行中的动态错误信息。例如算术运算中把 0 作为除数，这将使程序无法继续执行，只能停止。
4. 还有一种情况：程序能执行到结束，并不出错，但是产生的效果却不合要求，或者最后结果不正确。这种错误属于一般性的语义错误，也是程序编制方面有问题。

编程中出错是常见问题，调试和排除错误是程序设计（开发）过程中必需的工作阶段。

### 动态运行错误的排除

人们常把程序错误分为两类：一类是*静态错误*，通过静态检查源程序可以清楚地看到它们。编译程序、连接程序能发现的错误都属于这一类。系统在加工中发现错误给出信息后，人比较容易通过检查有关位置的上下文，确定错误原因和改正方法。要想找出这种错误，需要熟悉 C 语言的规定，包括各种结构形式和上下文关系方面的规定等。

另一类称为*动态运行错误*，出现在程序执行中，确认和纠正都更困难。仅从程序代码，数据情况与得到的结果去设法弄清原因，需要更多的分析和思考。在发现动态运行错误后，首先还是应该分析错误的现象和程序代码，考虑出现错误的可能性，逐步排除疑点。

在发现程序错误的疑点后，应该通过适当选择试验运行中提供的数据，设法确认所得到的认识。还可以设法找出导致错误的最简单数据。经过一系列试验和仔细分析，简单程序中的大部分错误都可能直接确认和更正。

如果无法直接确定错误原因，那么可能就需要采用动态检查技术。进行动态错误检查的基本方法是检查程序执行的中间过程（中间状态）。人们最常用的一种方式是在有疑问的地

方插入一些输出语句，让程序在执行中输出一些变量的值。通过检查关键性变量的变化情况，常常可以发现导致程序错误的线索。

C 语言系统通常都为程序动态检查提供了支持。尤其是各种集成式开发环境，它们都对程序的动态检查提供了强有力的支持。这方面的功能通常包括追踪、监视、设置断点、中断执行等。在以调试方式执行程序时可以使用这些功能。这里做些简单介绍：

1. 追踪：一般程序执行是通过一个启动命令，程序启动后就无约束地自动进行直至结束（可能是被强行终止或是自己终止），或进入死循环。对程序进行追踪，指的是以有控制的方式执行程序。例如要求它一个个语句地执行（单步执行），要求它执行到某处暂时停下来（中断执行）等。这样就可以通过各种方式检查程序的中间状态，以便更容易发现错误的根源。目前各种集成开发环境都提供了许多追踪及检查功能。
2. 监视：指在程序追踪过程中始终关注程序里某些变量的变化情况。
3. 设置断点：在开始追踪前，可以在程序里标出一些位置，要求程序执行到达这些位置时停下来，以便做进一步检查。程序在断点暂停后，可以按命令继续执行下去，或从执行中退出。如果确实发现错误，显然应该让程序停下来，修改后重新编译。
4. 中断执行：当发现（或认为）程序进入了非正常状态，或在程序执行中需要检查中间状态时，可以中断程序执行。在调试中可以给程序发中断命令，程序接到中断命令就会停在当时执行点，但还处于执行状态中。这样就可以检查执行现场的各种情况。

强有力的集成开发环境对编程而言确实是一个好条件，在学习程序设计的过程中，逐步了解和掌握所用工具也非常重要。目前有很多商品的集成开发环境，微机上的语言系统一般都以这种环境作为主要部分。不同开发环境虽然各有特点，但在对程序开发和调试的支持方面差别不大，掌握一个就可以触类旁通，学习使用其他系统时不会遇到很大困难。

应当看到，再好的集成开发环境也只是一个好工具，正确熟练地使用它们，能帮助编程者发现程序错误的线索，但确认和改正错误则必须依靠人动脑动手。另一方面，不能因为有了集成开发环境就不注意程序的写法了。人们在程序设计实践中认识到，良好、正确的编程习惯和方式是至关重要的，不可替代的。

也应看的事物的另一面：好的集成开发环境并不能造就优秀的程序工作者。现在的程序开发环境越来越强大，但我们却常能看到许多用高级工具的人编出的程序质量很差。编好程序，最重要的还是要有对这一工作过程中的规律性的理解，以及相当的程序设计经验。程序并不是代码的堆积，编程中最重要的一个方面是程序的设计和组织，程序越大，这方面工作的地位和作用就越明显。本书后面还要通过讨论和例子反复强调这一问题。

关于调试，还有一个重要问题。荷兰计算机科学家（图灵奖获得者）Dijkstra 有一句名言：调试可以确认一个程序里有错误，但是不能确认其中没有错误。一个程序是否正确，这是一个非常深刻的很难回答的问题，关于这个问题，既有许多理论研究，也有许多实际的方法研究。在进入程序设计这个世界之前，请大家首先记住这一点。

## 1.5 问题与程序设计

有了适用的程序语言，我们该如何着手编写程序呢？程序设计是一种智力劳动，编程序就是解决问题。初学程序设计时写的是很简单的程序，与做一道数学应用题或物理练习题有类似之处。编程序时面对的是一个需要解决的问题，要完成的是一个符合题目要求的程序。一般说，解决问题的过程可分为三步：第一步是分析问题，设计一种解决方案；第二步是通过程序语言严格描述这个解决方案；第三步，在计算机上试用这个程序，运行它，看是否真能解决问题。如果在第三步发现错误，那么就需要仔细分析错误原因，弄清后退到前面步骤去纠正错误。如果发现程序有问题，那就要修改它，然后重新编译运行和检查；如果发现求

解方案有误，那就需要修改方案，重编程序，……。请重看图 1.2。

这个工作过程的第一步与在其他领域里解决问题类似，只是考虑问题的基础不同。在程序设计中，我们需要从计算和程序的观点出发，这将引出许多新问题，是本书讨论的一个重要方面。第二和第三步是程序设计的特殊问题。由于程序语言的各种结构有明确定义的功能，把头脑中形成的解决方案变为程序语言描述，往往也不是直截了当的，需要经过仔细考虑和规划。另一方面，程序语言有严格规定的形式，把想清楚的程序用符合规定的形式写出来，也需要做不少工作，在这个过程中也可能犯错误。前面关于程序中可能的错误与排除的讨论，主要关注第二步和第三步之间的小循环，这个方面有许多新东西需要学习。至于发现问题解决方案有错误，则需要根据对所发现错误的深入分析。

在程序设计领域里，在解决小问题与解决大问题，为完成练习而写程序与为解决实际应用而写程序之间并没有截然的鸿沟。开发大的实际程序或软件系统，增加的主要是一前期工作：首先要把问题分析清楚，弄明白到底要做什么。这方面还需要进一步学习。

本课程中涉及的东西很多，包括知识的记忆和灵活掌握，解决问题的思维方法，具体处理的手段和技巧，还有许多实际工作和操作技能问题。我们把几个重要方面列在这里：

1. 分析问题的能力，特别是从计算和程序的角度分析问题的能力。应逐渐学会从问题出发，通过逐步分析和分解，把原问题转化为能用计算机通过程序方式解决的问题。在此过程中构造出一个解决方案。这方面的深入没有止境，许多专业性问题都需要用计算机解决，为此，参与者既需要熟悉计算机，也需要熟悉专业领域。将来的世界特别需要这种兼容并包的人才。虽然课程和教科书里的问题很简单，但它们也是通向复杂问题的桥梁。
2. 掌握所用的程序语言，熟悉语言中各种结构，包括其形式和意义。语言是解决程序问题的工具，要想写好程序，必须熟悉所用语言。应注意，熟悉语言绝不是背诵定义，这个熟悉过程只有在程序设计的实践中才能完成。就像上课再多也不能学会开车一样，仅靠看书、读程序、抄程序不可能真正学会写程序。要学会写程序，就需要反复地亲身实践从问题到程序的整个过程，开动脑筋，想办法处理遇到的各种情况。
3. 学会写程序。虽然写过程序的人很多，但会写程序、能写出好程序的人就少得多了。经过多年的程序设计实践，人们对什么是“好程序”有了许多共同认识。例如，解决同样问题写出的程序比较简单就是一个追求。这里可能有计算方法的选择问题，有语言的使用问题，其中需要确定适用的程序结构等。除了程序本身是否正确外，人们还特别关注写出的程序是否具有良好的结构，是否清晰，是否易于阅读和理解，当问题中有些条件或要求改变时，它们是否容易修改程序去满足新的要求等等。后面许多章节里会反复提到这些问题。
4. 检查程序错误的能力。初步写出的程序常会包含一些错误。虽然语言系统能帮我们查出其中的一些，并通告发现错误的位置，但确认实际错误和实际位置，确定应如何改正，这些永远是编程者自己的事。对于系统提出的各种警告，系统无法检查的错误等的认定就更要依靠人的能力。这种能力也需要在学习中有意识地锻炼。
5. 熟悉所用工具和环境。程序设计要用一些编程工具，要在具体计算机环境中进行，熟悉工具和环境也是这个学习中很重要的一部分。目前大部分读者可能要用某种集成开发环境做程序实习，熟悉这种环境的使用也很重要，可能大大提高我们的工作效率。

### 本章讨论的重要概念:

程序，程序语言，机器语言，汇编语言，高级语言，FORTRAN，程序设计语言，C 语言，ANSI C 标准，C 程序，自由格式语言，程序的格式，程序的加工，程序执行，源程序，编译，目标模块和目标文件，连接，可执行程序（文件），集成式程序开发环境，程序调试，程序排误，出错信息，语法错误，上下文关系，警告信息，连接错误，死循环，语义错误，静态错误，动态运行错误，追踪，监视，断点，中断执行。

## 练习

1. 设法找一找有关程序语言发展的书籍或者文章，或者计算机辞典的有关条目，读一读，了解程序语言的历史、发展、现状等方面的情况。
2. 设法找到 ANSI C 标准或者中国国家标准 GB/T 15272-94《程序设计语言 C》，浏览这些标准的目录，了解在定义一个程序语言时需要说明哪些东西。
3. 熟悉自己学习 C 语言程序设计时准备使用的编译系统或者集成开发环境，了解这个系统的基本使用方法、基本操作（命令式或窗口菜单的图形界面方式），弄清楚如何取得联机帮助信息。设法找到并翻阅这一系统的手册，了解手册的结构和各个部分的基本内容。了解在该系统中编一个简单程序的基本步骤。
4. 输入本章正文中给出的简单 C 程序例子（注意程序格式），在自己所用的系统中做出一个 C 语言源程序文件；对这个源程序进行加工，得到对应的可执行程序；运行这个程序，看一看它的效果（输出了什么信息等）。
5. 在第 4 题完成的程序里各种地方随便加入一些空格、制表符、换行等字符，看看在什么情况下会出现问题，什么情况下不会。如果出错，请仔细考察系统产生的出错信息和自己所做修改的关系，学习阅读系统的错误信息行。
6. 对第 4 题完成的正确程序中随意做一点修改（正文中提出的错误例子，或者其他修改），看看编译加工过程中会得到什么信息。弄清错误信息行和自己所做修改的联系，随后重新把程序修改正确。重复上述步骤。
7. 修改上述程序里位于两个双引号之间的字符序列，看看程序的输出发生了什么变化。修改程序使它能输出你想要的东西。
8. 如果你所用的系统中可以用中文，在上述程序的一对双引号之间写一句中文，看看程序能否正确加工，程序的输出是什么，能否正确输出中文信息。