第1章　程序设计基础与编程规范

本 章 要 点

- 程序设计的基础知识。
- 程序的基本结构。
- 编程规范。
- 提问的智慧。

1.1　程序设计与编程工具

程序设计（programming），经常被称为编程。那么什么是程序？为什么要学习编程？

1.1.1　程序与程序设计的概念

计算机程序（computer program）是指一组指示计算机执行的指令，通常用某种程序设计工具或语言编写，且由键盘输入的各种字符组合而成，这些字符集合就是所谓的程序代码（code）。通常，程序设计指某种求解算法用某一程序设计语言的具体实现过程。程序设计或编程的目的是以编程方式来求解某一问题，或实现某一功能。

1.1.2　为什么要学程序设计

信息技术飞速发展带来的技术与社会的变革远远超出人们的想象，而且这场巨变仍在进行中。计算机程序设计作为信息技术最基本、最重要的组成部分之一，越来越受到社会的重视。目前不少中小学已经展开了程序设计的学习，很多知名大学已经提升了程序设计在所有专业的人才培养及课程体系中的地位。

据报道，过去五年，斯坦福大学排名超越其他常春藤名校，稳居由哈佛大学制定的全美大学排行榜首位。在学生入学难度和校友捐赠数量两项评价指标中，斯坦福排名第一。入学难度决定生源质量，校友捐赠数量评价毕业生质量。据专家深度分析，斯坦福在计算机、工程技术等学科的领先地位和位于硅谷的地域优势，吸引了无数怀抱创业梦想的青年才俊。调查结果显示，有 90% 的斯坦福在校本科生学习过至少一门计算机程序设计相关课程。

计算机程序设计能力是现代社会人们应具备的基本素质之一。具体来说，学习程序设计的重要性体现在以下几个方面。

1. 学会程序设计能够极大拓展个人使用计算机解决问题的能力

与学习使用其他计算机软件（如操作系统、办公软件、图形图像处理软件等）不同，程序设计方法提供了一种底层的、本质的解决方式，通过算法和编程可以使用计算机解决更为复杂的、个性化的专业领域问题。

2. 学会程序设计是技术创新的根本

学会程序设计，可以很容易将自己的想法付诸实践，在整个过程中会产生无数的创意。编程水平越高，这样的创新体验就越沁人心脾。编程能更好彰显人的个性、魅力与才华。

3. 学习程序设计跟学习语言一样简单

编程就跟我们在生活中学习语言表达一样自然和简单。只要掌握简单的语法，即可进入用计算机程序解决问题的快乐过程。

4. 学习程序设计提升逻辑思维能力

编程是对问题求解过程的代码化表达，要应用大量逻辑分析，编程过程可以锻炼、训练并提高逻辑思维能力。

5. 社会需要越来越多的程序设计者

就业岗位是由社会需要决定的，而高品质的需求主要来源于具有创新性的产品和服务，而这两者都离不开程序设计。不论什么专业，程序设计水平高的同学，就业更容易、薪酬更高。

1.1.3 为什么要学习 C 程序设计

计算机编程语言或开发工具众多，为什么要学习 C 程序设计？

C 语言[①] 自 1970 年（也有文献说 1968 年和 1972 年）由美国贝尔实验室的 Ken Thompson 和 Dennis Ritchie 开发成功以来，近半个世纪过去，C 语言仍然是世界上使用最为广泛的程序设计工具。C 语法简洁、紧凑，使用方便、灵活，得到了从初学者到专业程序设计师的深爱。在主要编程语言排行榜中始终雄居前 3 位。

1983 年，贝尔实验室的 Bjarne Stroustrup 在 C 语言基础上推出了 C++。C++进一步扩充和完善了 C 语言，具有类与对象、封装、继承、重载、多态性等特性，是一种面向对象的程序设计语言。

C 语言是结构化程序设计的代表性语言，有极为丰富的教学、应用开发资源，是学习程序设计的理想选择。

以世界范围内影响最大的程序设计大赛 ACM 为例，C/C++、Java、Pascal 是其仅支持的几种编程语言。

① 由于 C 语言本身就表示一类程序设计语言或工具，本书在表述时大部分时候只称 C。

1.2　程序的基本结构和要素

1.2.1　程序的基本结构

通常，每个程序都包含一个或多个函数，但只有一个主函数 main()，它是程序的入口，整个程序的执行就是从这个函数开始的。没有主函数 main()，程序无法开始。

在由一对花括号（{ }）确认范围的函数内部，一般包含变量声明、变量赋值语句、执行输入输出、运算、函数调用等语句，每条语句由分号“;”结束。

同时，C 本身提供了大量的完成某一类特定功能的标准库函数，如输入、输出函数，常用数学函数，字符串处理函数等。这些函数是系统预先编制好的，它们存放在不同的库文件。这类文件常被放在程序开始，所以又称为“头文件”。当需要某一库函数时，要在程序开头位置作预处理声明，格式是：

#include<头文件名>

注意：编译器在编译预处理命令时，是告诉编译器如何对源程序进行处理，其本身不属于语句范畴，所以预处理命令末尾不加分号“;”。

1.2.2　输入输出

通常，程序通过输入获得要处理的数据，然后对输入数据进行处理，最后输出结果。在 C 中，输入输出是用一系列输入输出函数来完成的。这些内容在后面的章节中单独介绍。

1.3　程序设计一般方法

(1) 从问题的全局出发，将问题转化为数学语言（或代数语言）。
(2) 建立求解算法。
(3) 定义变量，选择库函数或自定义函数。
(4) 按照解决问题的顺序把语句和函数在 main() 里面像积木一样堆起来。
(5) 编辑程序代码、编译、调试、连接、生成可执行文件，运行程序、测试数据。
第 (5) 条的相关内容参见附录 B。

1.4　编 程 规 范

首先，编程是一门艺术，编程是美学的一个分支。

国际知名的算法和程序设计技术先驱、图灵奖获得者 Donald E. Knuth，在早年出版了《计算机程序设计的艺术》[①]。他在写作《计算机程序设计的艺术》和《计算机与排版》时，大量参考了美国出版的《Aesthetic Measures》（《美学标准》）一书，将美学标准引入到程序设计中[②]。

① 此书被评为 20 世纪最有影响力的 20 本书之一。
② 本书的排版使用了 Knuth 设计的 tex 格式。

其次，美观的程序，也更容易理解、便于发现错误、交流、维护，因此，好的程序除了代码本身的美感外，还有明确的实际效用。

在程序设计与开发的长期实践中，人们总结出一套行之有效的写出好程序的方法，这些方法就构成了现在具有指导意义的代码编写规范，简称编程规范。编程规范独立于编程工具和开发环境。

1.4.1 为什么要遵守编程规范

很长时间以来，在程序设计课程的教学过程中，往往只关注程序设计本身的语法和功能实现，编程规范普遍未受到重视。编写规范化的代码在软件工程学中是非常重要的。编程规范很大程度上决定着软件开发与维护的效率。

程序调试是编程过程中最耗时费神的。常见的现象是：学习理解一个算法需要一天，输入代码需要一小时，找到 bug 需要数天。数据显示，一个软件产品的生命周期中 80% 以上的时间和经费用于维护。

不规范的代码可读性差、调试困难、存在各种 Bug 隐患。而编程规范是初学者容易忽略的。编程刚入门时，代码量较少，这一问题不明显。但是，这种不规范编程形成习惯后，当程序代码量变大时，各种无法预知的 Bug 接踵而至，经常给编程者本人、答疑教师带来许多困扰，并造成无谓的时间浪费。

具体地说，养成规范的编程习惯，可以大大提高程序的阅读、理解、交流、调试、维护、升级效率。对于初学者，更有必要从一开始就培养这种习惯。其重要性可用“车同轨、书同文、度同制”作类比。

本书仅列出若干项最基本的编程规范。

1.4.2 编程规范的基本要求

(1) 可读性优先于程序运行效率。

(2) 程序结构清晰，简单易懂，单个函数（包含主函数）的程序不得超过 100 行。

(3) 程序实现的功能，要简单，直截了当，代码精炼，避免多余程序。

(4) 尽量使用标准库函数和公共函数。

(5) 不要随意定义全局变量，尽量使用局部变量。

(6) 使用括号以避免二义性。

1.4.3 标识符命名

程序代码中需要自定义的变量名、数组名、函数名、指针名等标识符，对程序可读性影响较大，规范、统一的标识符名称是编程规范的基本的要求。

标识符的命名要有明确清晰的含义，使其具有自注释功能。规范、统一的标识符使程序具有良好的可读性。

常用的标识符规范有三种：

(1) Pascal（帕斯卡）命名法或驼峰命名法。源自于 Pascal 语言的命名惯例，看上去就像骆驼峰一样此起彼伏，故又称驼峰法。其规则是，标识符一般由两个或两个以上的单词

组成，每个单词的首字母大写，且中间不用下划线，如：

```
MyFirstName
LetterAmount
PrintEmployeePaychecks()
```

驼峰法也称为大驼峰法，以便与下面的小驼峰法区别。

(2) 小驼峰命名法。第一个单词小写，第二个及后面的单词首字母大写。如：

```
myLastName
studentScoreSum
printEmployeePaychecks()
```

(3) 下划线法。将不同单词间用下划线连接，如：

```
my_last_name
student_score_average
print_employee_paychecks()
```

(4) 匈牙利命名法。由一位叫 Charles Simonyi 的匈牙利程序员首创，经常用于面向对象的程序设计，如 Windows 系统及其应用软件的开发。其基本命名规则是：

属性 + 类型 + 对象描述

下面对命名的各部分做一简单介绍。

属性部分：

g：全局变量

c：常量

s：静态变量

对于局部、动态变量，属性部分可省略。

类型部分：

指针：p

函数：fn

长整型：l

布尔型：b

浮点型：f

字符串：sz

短整型：n

双精度浮点型：d

字符：ch（通常用 c）

整型：i（通常用 n）

字节型：by

描述部分：

标识符代表的对象的描述，如 Max（最大）、Min（最小）、Counter（计数）等。

匈牙利命名法举例：

lpszName

由 l+p+sz+Name 组成，表示 32 位字符串（字符串内容为 Name) 指针。

如果标识符中有特殊约定、缩写、专业词，要有对应注释说明。

需要说明的是，有的时候，严格遵守编程规范与使用方便性不能兼得，在实际编程中，往往会根据使用环境做一定的妥协。例如：编程规范禁止诸如 a、b、c、d 这样的单字母标识符（循环变量 i、j、k 除外），但教学中用短程序来解释某一语法概念时，a、b、c、d 等作为标识符是可行的；考虑到数学的一些习惯用法，m、n、x、y、z 也经常可以作为局部变量等。

在本书中，标识符命名采用如下风格：

(1) 对于有应用背景的程序，标识符使用应用问题中该名称的英文或英文缩略词，或英文的组合式，组成时采用小驼峰命名法。如：

nChicken, nRabbit, nHead, nFoot	//鸡兔同笼问题中的鸡、兔、头、脚等的数量
year, month, day, leap	//闰年问题中的年、月、日、闰年
radius, area, circle	//圆半径、面积、周长

(2) 涉及数学问题的程序，一般与数学标记习惯相同，如：

a, b, c	//方程 $ax^2+bx+c=0$ 求解
sum, fact, ave, max, min	//和、阶乘、均值、最大、最小
a_n, a_n_1, a_n_2	//数列 a_n 的第 n、$n-1$、$n-2$ 项

(3) 用于解释语法概念的程序，作如下约定：

i, j, k, ii, jj, kk	//循环变量，数组下标
m, n, nA, nB, nC	//整型变量, nA，nB，nC 表示多个并列变量，下同
f, fA, fB, fC	//浮点型变量
ch, chA, chB, chC	//字符型变量
str, strA, strB, strC	//等字符串
arr, arrA, arrB, arrC	//数组
fun, funA, funB, funC	//函数
p, pA, pB, pC, pp	//指针, pp 用于表示双层指针

(4) 由两单词或多个单词组成的标识符，采用小驼峰法。

1.4.4 缩进

程序块要采用缩进风格编写。缩进可使整个程序看上去层次分明，特别是在嵌套选择、嵌套循环语句中，如果没有缩进，程序将很难理解。

缩进要求采用空格，一个缩进的空格数为 4 个。有的程序编辑器（注意不是编译器）支持 Tab 键，一个 Tab 相当于 4 个空格。但不同的编辑器设定的每个 Tab 键对应的空格数不同，可能会造成程序布局不整齐。DevCPP 软件在编辑器属性中可以设置 Tab 键的空格数。

1.4.5　空行

程序代码中的空行可以认为是一种特殊的注释，相当于普通文档的分段，插入空行使程序像搭积木那样呈模块化。每一个程序模块对应一定的功能，如变量声明、输入、变量赋初值、运算、输出都可以通过插入空行来实现相对独立的模块。

1.4.6　一行只写一条语句

一行只写一条语句，不允许把多个短语句写在一行中。下面是不符合规范示例：

```
length=0; width=0;
```

应如下书写：

```
length=0;
width=0;
```

1.4.7　if、for、while 语句体加括号 { }

在 if、for、while、switch 等语句的语句体（也称语句块），一定要加一对花括号 { }，即使执行语句体只有一行语句，也应该养成加花括号的习惯。

下面是不符合规范的示例：

```
if (i==1)
counter++;
```

规范化的写法如下：

```
if (i==1)
{
    counter++;
}
```

这一条规范有人可能不理解，因为很多时候，尤其是在初学者编写的程序中，这些语句的执行语句体只有一条语句，没必要浪费敲键盘时间。但从大量的经验来看，一旦这种不加花括号的风格养成习惯，当遇到需要执行多行语句时，会因为忘记加花括号而出错。

也有不少程序设计师喜欢将左花括号 { 直接写在 if、for、while 的圆括号“( )”后，而不是另起一行，即：

```
if (i==1) {
    counter++;
}
```

这样做的好处是可以有效避免如下的错误：

```
if (i==1);   //加分号“;”后，使条件 i==1 无效
{
    counter++;
}
```

上面这类错常一旦发生，不容易发现。

1.4.8 每行只声明同一类变量

每行只声明同一类变量是指只将同一类变量放在一行。如循环变量 i，j 单独放一行，数组单独放一行等。

初学者往往喜欢在一行声明全部变量，这会降低可读性，容易出错。当变量较多，并且这些变量为不同的数据类型时，更容易出错。

下面的变量声明是不规范的：

```
int n, array[5], temp, i, j;
```

下面的变量声明是规范的：

```
int n;
int array[50];
int temp;
int i, j;
```

下面是涉及年、月、日、闰年问题时的变量声明：

```
int year, month, day;          //年、月、日变量，放一行声明
int leap=0;                    //是否闰年标识变量
```

1.4.9 函数要先声明后定义

类似于变量使用时的情况，在使用函数时，规范的程序书写顺序是：

(1) 在主函数前声明自定义函数；

(2) 主函数；

(3) 自定义函数。

1.4.10 注释

注释可以增加程序的可读性，特别程序中的关键点、难点。下面是注释的一些规范。

(1) 注释要详略得当、恰到好处。可根据程序使用者或程序的复杂度来决定注释的多少。如果是给初学者看的样例程序，注释可适当详细。一目了然的语句不加注释。

注释不是越多越好，过多的注释会使编辑器的版面显得混乱。

(2) 注释的内容要准确、清晰、简洁，避免二义性；错误的注释不但无益，甚至有害。

(3) 写注释的位置应与其描述的代码接近，一般应放在其上方或右方（对单条语句的注释）的相邻位置，不可放在下面，如放于上方则需与其上面的代码用空行隔开。

(4) 对于标识符（如变量名），如果其命名不是充分自注释的，在声明时都必须加注释。有时，即使这些标识符是自注释的，也推荐加入简要注释，可使阅读程序时的难度降低。

(5) 对典型算法、新算法、原创算法的关键点加注释。

(6) 全局变量要有详细的注释，包括对其功能、取值范围、哪些函数或过程存取它、存取时注意事项等的说明。

(7) 处理过程的每个阶段都有相关注释说明。养成边写代码边注释的习惯，修改代码同时修改相应的注释，以保证注释与代码的一致性。不再有用的注释要删除。

(8) 避免在注释中使用缩写。在使用缩写时或之前，应对缩写进行必要的说明。

(9) 为每个程序加注释。一般写在程序开头，用来说明程序功能、算法、版权、作者、日期等。

(10) 在每个自定义函数头部加注释，说明功能、参数、返回值。

DevCPP 的编译器同时支持两种形式的注释。

C 注释的格式为：

```
/* 注释文字 */
```

C++注释的格式为：

```
//注释文字
```

可根据注释的内容、长度、位置来决定采用哪种注释。

1.4.11　函数返回类型与 return 语句不缺省

声明和定义函数时，都要列出函数返回类型。不要使用系统默认。

除非函数返回类型为 void，否则都应在函数体内有 return 语句。这一点在自定义函数时比较重要。本书所有例题的主函数 main() 都有返回类型（绝大部分时候是 int），main() 内都有 return 语句，为的是培养这种良好的习惯。

1.4.12　例 1-1：鸡兔同笼

设在一只笼子里关着鸡和兔子，从键盘输入鸡和兔的总只数、总脚数。若有解，输出鸡和兔各多少只？若无解，则输出：无解。

现在用 C 程序求解鸡兔同笼问题，来体会编程规范对程序可读性、美感的影响。编程中用到的选择控制、循环控制语句会在后面的章节中介绍。

先来看比较规范的程序。

```
01  //ch1_1A.cpp
02  //编程规范示例 —— 鸡兔同笼
03
04  # include<stdio.h >
```

```
05  int main()
06  {
07      int nHead, nFoot;   //分别表示鸡兔总只数、总脚数
08      int nChicken, nRabbit;   //分别表示鸡、兔只数
09      int flag=0;   //有、无解标志，若 flag=1, 找到解; 若 flag=0，无解
10
11      printf("输入鸡兔总只数：");
12      scanf("%d", &nHead);
13      printf("输入鸡兔总脚数：");
14      scanf("%d", &nFoot);
15
16      //嵌套循环穷举所有可能，按条件找正确结果
17      for (nChicken=0; nChicken<=nHead; nChicken++)
18      {
19          for (nRabbit=0; nRabbit<=nHead; nRabbit++)
20          {
21              if (nChicken+nRabbit==nHead
22                      && 2*nChicken+4*nRabbit==nFoot)
23              {
24                 printf("鸡 =%d\n",nChicken);
25                 printf("兔 =%d\n",nRabbit);;
26                 flag=1;    //找到解标记
27
28                 //若找到解，解有唯一性，主函数返回 1，程序结束
29                 return 1;
30              }
31
32          }
33      }
34
35      //其他情况：无解
36      if (flag==0)
37      {
38         printf("无解 \n");
39      }
40
41      return 0;
42  }
```

对于本题，如果有解则有唯一解，程序第 29 行表示一旦找到解，主函数立即返回 1（也可以是其他整数），程序结束。这样可以减少程序不必要的运行时间，虽然节省的时间微不足道，但这样对优化程序、提高逻辑思维能力是十分必要的。如果没有第 29 行程序，程序依然可正确运行。

程序运行结果：