● 软件工程系列丛书

SEI
SOFTWARE ENGINEERING INSTITUTE
BUAA

北京航空航天大学出版社

金茂忠 主编

C

程序设计高级教程

C 程序设计高级教程

金茂忠　主编

北京航空航天大学出版社

内 容 简 介

《C 程序设计高级教程》根据国外 1988 年出版的最新资料改编而成，它是 C 程序设计从初级阶段到高级阶段必读的后续课程。每一个想真正掌握 C 语言的软件开发人员将从本书中获得极为丰富和极有价值的资料。人们渴求得到的这些知识是迄今为止几乎所有关于 C 教材中都未曾涉及的。

本书主要内容有：

· 详细、全面地讨论了 C 语言最难掌握的结构、联合和指针等各种高级处理技巧。

· 以各种编程实例十分详尽和清晰地介绍了 C 标准程序库、标准 I／O 程序库和 UNIX 系统调用方法。

· 教会你使用 CURSES 来编写有关终端屏幕控制和多窗口处理的程序。

· 让你彻底掌握用 Make 工具来维护、管理整个程序系统。

· 全面介绍了 line、ctrace 和 sdb 等强有力的 C 排错工具的功能和使用方法。本书附有大量编程实例，每章后均有习题。

本书即可作为计算机专业本科生和研究生的教材，也可作为计算机软件人员、广大科技工作者的参考资料。

C 程序设计高级教程

C CHENGXUSHEJI GAOJI JIAOCHENG

金茂忠　　主编

责任编辑　罗燕京

*

北京航空航天大学出版社出版

北京航空航天大学软件工程研究所微机输入排版、激光打印

地质出版社胶印厂印刷

新华书店总店科技所发行　　各地新华书店经售

*

787×1092　1／16　印张：25.5　字数：566 千字

1990 年 6 月第一版　1990 年 6 月第一次印刷

印数：0001—5000　　定价：11.20 元

ISBN 7-81012-192-8／TP·035

前　言

C语言不仅早已成为计算机系统软件开发者和软件工程研究人员的挚友，而且正在各种应用软件开发人员心目中成为一颗冉冉升起的明星。确实，只要列举以下事实，就足以说明C语言在计算机世界的重要地位。

·全世界唯一能称为操作系统工业标准的*UNIX*系统，从只有一万多行代码的核心到由极其丰富的实用程序（达数十万行代码）组成的*UNIX*程序设计环境，几乎都是用C语言实现的。

·在拥有最广大用户的*PC-DOS*上，用C语言开发的软件已比比皆是，其中包括大名鼎鼎的*dBASE*Ⅲ和*AUTOCAD*系统。

·日本通产省和民间产业界通力协作，投资达250亿日元，历时5年（1985-1989）之久研究开发了∑系统（软件生产工业化系统）。该系统由设在东京的∑中心和遍布日本全国的成千上万台∑工作站通过计算机网络构成一个∑软件工程环境。预计该系统在1990年投入使用后可显著改善日本软件生产的效率和提高软件产品的质量。这个令人瞩目的∑系统，其全部软件指定要用C语言进行开发。

·现在世界上生产的计算机，小到微型机，大到象*Cray-1*这样的巨型机，几乎无一不配置有C语言。

由此可见，C语言之所以受到广大软件工程技术人员的高度重视和青睐不是没有道理的。很多读者也都希望能尽快地掌握用C语言来开发软件。

目前各种C语言的著作已出版不少。但人们一直翘首以待有一本关于C语言高级编程技巧的书。因为对于一名想熟练和深入掌握C语言编程的人，只有初步的C语言知识显然是很不够的。

本书是我们根据国外1988年出版的最新资料改编而成的，它是C程序设计从初级到高级阶段必读的后续课程，每一个C语言的高级编程人员都将从本书获得十分丰富和有价值的资料。而这些C语言高级编程知识却是迄今为止出版的大量有关C的教材中所未曾或很少涉及的。

本书主要内容包括：

·详细、全面地讨论了C语言最难掌握的结构、联合和指针等高级处理技巧。

·以各种实例详尽和清晰地介绍了C标准程序库、标准*I/O*程序库和*UNIX*系统调用方法。

·用*CURSES*来编写有关终端屏幕和多窗口处理的程序。

·使用*Make*工具来维护、管理整个程序系统。

·全面介绍了*lint*、*ctrace*和*sdb*等强有力的C排错工具的功能和使用方法。

本书内容丰富、讲解深入详细，取材新颖实用、并注重实践和应用。书中列举的大量编程实例和在各章后面附有的习题对读者无疑是十分有益的。

参加本书编写工作的有金茂忠、张子让、罗燕京、刘超、晏海华、田蕾　闫龙

翔、谢小兵和栗更等同志。全书由金茂忠负责审阅定稿。由于编写时间仓促、水平有限，难免有疏漏不当之处，欢迎广大读者给以批评指正。

最后，我们想借此机会特别感谢周伯生教授，没有他的鼓励、帮助和卓有成效的支持，本书是难以在短时间里和读者见面的。为使本书能尽早奉献给广大读者，许多同志为此付出了辛勤劳动，在此谨向他们致以深切的感谢！

金茂忠 *1989.12.*

于北京航空航天大学软件工程研究所

目　　录

第一章 简 介

本书是C程序设计语言由初级到高级阶段的后续课程，目的在教会你有关C程序设计语言的各种较深入的课题。本书假设你已学过初级的C程序设计语言（如"Programming in C"，Hayden Books，1983），或已具有同样的经验。

在C程序语言中，结构（structure）和指针（pointer）的处理是最困难的部分，我们将在第二章进行详细、全面地讨论；其中除了基本课程的复习外，对有关结构和指针的高级处理技巧，也都将详尽地一一介绍。第二章有关结构的介绍，重点放在结构的运算、结构与函数的关系、结构数组和较复杂的结构等方面。

第二章有关指针的介绍将从概念和实际设计的观点，对指针处理运算的事先准备工作先做一番回顾，然后再介绍指针与函数的关系，指向结构的指针（pointer to structure），指向数组的指针（pointer to array）和字符串等。由于字符数组、字符串常数和字符指针经常被混淆，所以本节特别强调它们彼此的不同。接着是指针算术的详细描述、指针与数组的关系、指针在较复杂的数据结构〔如链表（linked list）和树结构（tree）〕中的作用、二维数组和指针数组、指针的指针和指向函数的指针；最后以一个实际的例子——派遣表（dispatch table）的设定，来说明指向函数指针的功能。

第3~5章包括了在UNIX系统下所提供的各种C程序库，其中第三章与第四章讨论标准C程序库和标准输入输出程序库，这些程序库也会出现在很多非UNIX系统上；第五章介绍UNIX的系统调用，该章所介绍的函数是UNIX系统核心的一部分，因此可能无法在非UNIX系统上执行。

经验表明，由于程序设计人员不知道各种程序库中有哪些可用的函数或不知如何使用，因此常常很辛苦地设计了一些已经由别人开发完成的函数，这种最容易出错、也是最浪费时间与精力的事都必须尽量避免。第3-5章将对标准C程序库、标准输入输出程序库和UNIX系统界面程序库的众多常用函数（甚至非常用函数）做一系统、全面而详细地介绍。

每一章里的例程都是根据我们所提供函数的类型而归类在一起的。例如在标准C程序库一章中，你会发现函数的标题是字符测试、数据转换、字符串处理、存贮器存取、动态存贮分配、日期与时间的转换、数据的加密（encryption）、口令（password）和分组文件（group file）的处理、进程（process）的控制、表格和树结构的处理等。在大部分的例程（routine）说明之后，都会接着一个完整而可执行的程序范例，以便使读者更加明了这些例程在实际上是如何使用的。

第四章介绍标准输入输出函数库。首先对它做一概要介绍，然后回顾"标准输入"和"标准输出"概念，最后是详细介绍函数如何将数据写到标准输出上，以及如何从读取数据。第二节中将对printf和scanf做一完整的说明。下一节是有关文件处理介绍，包括随机输入输出（random I/O）、临时文件（temporary file）的建立

达命令行（command lines）给 UNIX 系统 shell 执行、控制缓冲区的管理策略、错误处理和各方面信息的取得等.

第五章讨论 UNIX 系统调用，一开始是对这些系统的调用做一概括的介绍，然后详细说明各种调用，依次为打开和关闭一个文件的输入输出调用，读取和写入数据，执行随机输入输出，建立及使用管道（pipe）以及控制终端输入输出.

输入输出介绍完后，紧接着是文件处理调用，包括文件的建立，删除、连接和文件属性的更改等. 然后再说明进程控制例程，包括以 fork 来产生新的进程，以 exec 来执行程序，两个程序通过管道来传送数据，取得和设置进程的信息等. 第五章末尾，以介绍信号处理例程和其他各种系统信息例程做为结束.

有关第三到第五章内容的最佳参考资料是"The UNIX Programmer's Reference Manual"（AT&T 贝尔实验室），本书并非要取代这本参考手册，而是教你使用很多归纳在该手册里的函数，你可以拷贝一份手册的内容以供阅读本书时参考. 在本书的第三章开头将介绍手册的组织章节，如何找到其中函数的说明，以及如何解释所找到的函数说明.

不幸，并没有任何标准的方法可以告诉终端装置如何去执行有关硬件的函数，如屏幕的清除或将光标移到屏幕的左上角（Home）等. 到目前为止，每一种不同的终端都只认识本身所设定的唯一代码来做这些函数. 例如要在 Digit 的 VT-52 终端上清除屏幕，你必须送出（Escape）H 字符，而在 Hewlett-Packard VT52 终端上，你必须送出（Escape）H 和（Escape）J 键等字符. 当程序设计员想设计一个以屏幕为导向的应用程序时，如屏幕编辑程序、试算表（spreadsheet）或以功能表为驱动的系统，如果希望这个程序可以允许用户在任何可能类型的终端上运行，那就必须考虑到这将和编写一独特的代码清除屏幕一样，会是一件庞大的工作.

庆幸的是，UNIX 系统已经采用一种称为 terminfo 的数据库作为共同的约定，用来描述如何从各种不同类型的终端中来执行各种不同的函数①. curses 程序库具有各种知道如何在该数据库上运算的例程，因此要清除屏幕时，您并不需要知道该送什么代码给终端，而只要调用 curses 数据库中的 clear 例程即可，它会自动送出所需要的代码给终端装置. 在第六章会教你如何利用 curses 程序来设计与终端无关的程序，目前很多 UNIX 的应用程序都利用到这个程序库. 如果你的应用程序是以屏幕为导向，那么你将发现 curses 程序库是非常有用的.

第七章将讨论 make 这个命令，它使你能够很容易地管理程序的建立，特别是那些被分割成很多不同文件的程序. make 命令会记录哪些文件被更改过，而以尽可能少的工作自动下达命令重新产生程序. 例如，假设你的 C 程序被分割成五个文件，当你编辑其中两个文件时，make 例程会知道另外三个文件的目标文件（object files）是对的，而只编译（compile）这两个文件，然后再将所建立的两个目标文件和另外三个目标文件连接（link）成一个新的可执行文件. 以上这种方式可省去你花脑筋去记住哪些文件被修改过和哪些文件需要重新编译的麻烦. 在你的程序被分成很多文件，且要依赖其它文件，如包含文件（include file）和程序库的情况下，make 例程会变得非常有用.

要程序在第一次执行时就完美无缺几乎是不可能的，所幸的是，在 UNIX 系统下有

---

①该数据库在System V以前的AT&T UNIX、XENIX和BSD系统上，称之为termcap.

各种各样的排错程序可用来对C程序进行排错，在本书的最后一章将会对这些排错程序进行详尽地说明，在这部分里你会学到lint程序，它是编译程序之一，但比C编译程序更详尽地检测你的C源程序，并能检查出潜伏的错误和非可移植性代码的使用，然后你将发现C预处理程序（preprocessor）是如何有效地处理在程序中加入排错语句的．利用上述技巧，可以很简单地给cc命令一个选择项，这样就能编译你所有的排错代码．另外，你也会学到如何设置你的程序，以便在运行时能得到各种程度的排错输出结果．

在UNIX system V的第二版中加入了ctrace命令，它可使程序执行时自动输出跟踪结果，你将学会该命令的使用方法，以便很容易地跟踪程序的执行．

本章最后介绍sdb，它是功能最强的一个工具．它是一个交互式的"符号"(symbolic)排错程序，可以跟踪程序的运行，并且可以在程序运行时检查和设置变量的值．它所具备的符号性质允许你使用标准C的符号来检查结构的成员、数组的元素、字符串和变量等．唯一的缺点是并非所有的UNIX系统都提供该程序．

附录中将教你当拥有一台未描述的终端而要使用curses程序库时，应如何填写termcap和terminfo项目，另外在附录B中归纳了所有在curses程序库中的函数以供参考．

我们建议你先阅读第二章的结构和指针，它是以后各章的基础．第三章到第五章是有关程序库的介绍，最好依次阅读．第六章到第八章分别为独立的课题（curses、make和debugging），可以按任意次序阅读，即在其他各章节前后阅读皆可．

在介绍如何使用程序库的函数或交互式排错程序时，如sdb程序，我们认为用例子来说明是最佳的方式，因此读者不必对本书中许多实际例子感到惊异，应该在你的系统上测试这些例子，并对程序中可选择的项目做适当的修改，以检测其各种功能，增加这些程序的使用性．每章末尾都附有习题，你要试着回答问题，以检验你对本书了解的程度．

本书中所列全部程序的源代码，均可通过USENET网络利用电子邮件的传递从Pipeline Associates, Inc.免费获得．任何用户要得到这些程序，只要简单地发送一个UNIX邮件给下列地址中的一个即可：

```
ihnp4!bellcore!phw5!topics
harpo!bellcore!phw5!topics
```

在行之前加上

```
SEND_PROGRAMS_TO:
```

程序便会自动以UNIX电子邮件的方式传送到行后面所指定地址的shell archive（里面含有如何取用它的资料），其中所有地址的指定都必须相对于ihnp或harpo才行。

下面的例子将使程序传送到用户joe系统上的ihnp4!ucbvax!galaxy:

```
$ mail ucbvax!ihnp4!bellcore!phw5!topics
SEND_PROGRAMS_TO: ihnp4!ucbvax!galaxy!joe
.
$
```

请注意：地址的声明必须采用文字表示（literal），所以如下行所示的地址是没有用的:

```
joe@outer.space.UUCP
```

本书虽源于 UNIX system V 第二版，但其大部分程序均可以不经修改即能在 XENIX Ⅲ、XENIX V 和伯克利 BSD UNIX 系统上运行；有些 UNIX 系统的界面程序和 curses 一章所提到的程序，可能需要做些稍微的修改才能在 XENIX 和 BSD 系统上运行。如果你在其他不同的操作系统上开发 C 程序，本书中的许多内容仍然适用。第二章内容全部适用，第三章和第四章有许多函数可在你的系统上使用，第五到第八章虽以 UNIX 系统为主，但是有一些例程和程序在你的系统上照样可以运行。

第二章　结构和指针

本章将详细介绍有关结构和指针的各种问题，包括如何定义结构和指针的变量、如何对该变量设定初值，如何对结构和指针做各种运算，如何与函数一起使用以及如何定义和运算结构及指针数组。

本章第二部分则讨论 C 程序设计语言中最困难的部分——指针。你将学会如何定义指针，如何间接地取得它们所指向的内容，对指针可进行何种算术运算，如何使用指向数组的指针、指向结构的指针、指向指针的指针以及指向函数的指针，还有如何利用指针产生较复杂的数据结构，例如链表和派遣表。

2.1 结构

2.1.1 定义

结构（structure）是一些可整体引用的变量所组成的集合，它与数组不同，因为结构中的元素（在结构中习惯称为"成员"（member））不象数组中的元素必须具有相同的类型，并且这些元素的引用方式也不相同。

当你的程序要使用结构变量以前，必须先告诉 C 编译程序这个结构的"长相"。这就涉及到定义结构的成员及其数据类型。

结构定义的一般形式如下：

```
struct sname {
    member-declaration
    member-declaration
    ...
};
```

以上定义了一个称为 sname 的结构及其成员。每个成员的声明——member-declaration 的一般格式如下：

```
type member-name;
```

一旦对 C 编译程序定义了结构，就可以继续对该结构类型声明变量。要注意的是，结构定义只是告诉 C 编译程序该结构的"长相"，而不会给它配置任何存储单元，一直要到结构变量声明以后，才会保留空间给该变量。

例如，假定希望在程序中储存好几个日期数据，那么这时定义一个称为 date 的结构来储存这些日期是再好不过了。假设程序中，日期要以三个整数来表示年、月、日，则这样的结构可定义如下：

```
struct date {
    int month;
    int day;
    int year;
};
```

这个称为 date 的结构共有三个成员：一个是称为 month 的整数，一个是称为 day 的整数，一个是称为 year 的整数。需要再次提醒的是：这时只是告诉 C 编译程序 date 结构的长相，而未保留任何空间；也就是说，你只是为 date 结构定义了一个如图 2–1 所示的模板（template）。

```
struct date {
   int month;
   int day;
   int year;
};
```

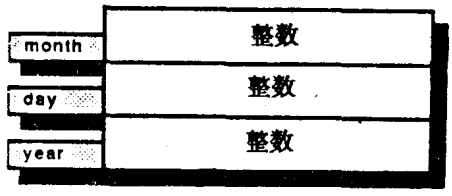

图 2–1　定义一个结构

上图模板的左边列有三个成员的名称，这些名称只是告诉 C 编译程序，当结构中的三个整数被引用时，该如何选取。例如，用名称 month 时，该选取那一个整数，用名称 day 和 year 时，各该选取那个整数。这些名称并不会和结构变量中的值一并储存起来，它们只在编译你的程序时才存在。

2.1.2 变量声明

当你告诉 C 编译程序有关 date 结构的长相后，就可以接着声明该结构类型的变量：

```
struct date today;
```

上述声明通知编译程序要保留一块空间给名为 today 的变量，它的类型是 struct date，如图 2–2 所示。

```
struct date today;
```

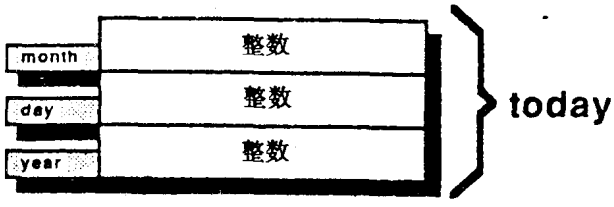

图 2–2　声明一个结构变量

2.1.3 结构变量的赋值

声明了变量以后，便可将值存入。当要把值赋给结构变量中的某一成员时，需用下面的格式：

```
variable.member = value
```

结构变量后面要跟一个运算符“.”，然后再紧接着该结构中合法的成员名称；例如本例中的 date 结构：结构变量是 today，合法的成员名称有 month、day 和 year。

因此若要把 1987 年 3 月 13 日存入 today 变量中，就必须写如下三个语句：

```
today.month = 3;
today.day = 13;
today.year = 1987;
```

现在 today 变量中的三个成员将被设定成图 2-3 所示。

```
today.month = 3;
today.day = 13;
today.year = 1987;
```

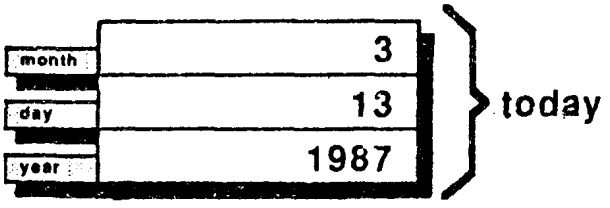

图 2-3　结构变量的赋值

程序 2-1 是一个很短的程序，它把到目前为止所讨论的内容做了一个小结①。

在 main 函数中，today 变量的类型声明为 struct date，并且有三个成员：month、day 和 year，其值依次设定为三个数：3、13 和 1987，然后再把结构中这三个值重新取出传给 printf 函数而显示出来。

请注意在此程序中，date 结构是在 main 函数外面定义的，如果在里面定义，其结果也相同；但若在其它情况下，究竟在何处定义结构将会影响到结果。结构定义和变量声明一样，按照定义的所在位置会产生一个有效的使用范围。假如结构在函数内定义，那么只有该函数知道它的定义，这就是"局部"（local）的结构定义；如果结构在任何函数的外面定义（通常是在文件的开头），那就是"全程"的结构定义，它表示以下所定义的任何函数都可以使用这个结构定义。

程序 2-1

```
struct date {
    int month;
    int day;
    int year;
};
main ( )
{
    struct date today;

    today.month = 3;
    today.day = 13;
    today.year = 1987;
```

①本书中的程序都不对main函数返回值的类型进行声明，而在一些非UNIX系统上，main函数要声明为void，以防止编译程序在编译时误以为 main 函数缺少对返回值的描述。

```
    printf ("%d / %d / %d / \n", today.month, today.day,
                    today.year - 1900);
}
$ a.out
3 / 13 / 87
```

(在 UNIX 系统下，可执行的目标文件名都预设为 a.out.)

2.1.4 结构初值的设定

结构变量可以在它们声明时便设定初值，但只限定是全程或静态（static）变量才行：换句话说，不可以对自动（automatic）结构变量做初值的设定。静态变量与自动变量不同，因为它要经过函数调用才能保持其值，并预设初值为零。请回忆自动数组（象结构一样）不能设定初值。

结构初值设定的一般格式如下：

struct sname variable = {val1, val2, …}; 所以在前面程序的声明中，要对 today 变量设定初值，其写法如下：

```
static struct today = {3, 13, 1987};
```

如上所述，自动结构变量是不能设定初值的，因此这里关键字"static"是必不可少的。如果省略，编译程序在编译时会出现错误信息。

2.1.5 结构的运算

把一个结构变量的值赋给（assign）另一个结构是结构可使用的少数运算之一，但只限定在"类型完全相同"的两个结构变量①。因此如果想把 date 结构的 today 变量内的值拷贝给另一个称为 tomorrow 的结构变量，你只要写

```
tomorrow = today;
```

便可。将整个结构视为一体时，除了把结构传给函数或从函数返回整个结构以外，你不能再对整个结构做其它的事②，因此不要书写如下语句来企图测试两个结构是否相等，因为它根本不能这样运算：

```
if (today = = tomorrow)
  …
```

而必须逐一比较它们的成员：

```
if (today.month = = tomorrow.month && today.day
    = = tomorrow.day && today.year = = tomorrow.year)
  …
```

当你引用结构中某个特定的成员时，这个引用的表达式（expression）类型和引用成员的类型是相同的，例如：

```
today.year
```

---

①该特性在第七版和BSD4.1系统上并未提供。

②该特性在第七版的BSD4.1系统上也未提供。

该表达式的类型就是 year 成员的类型，是个整数，因此 today.year 便可当作一个正常的 int 来使用．例如，可把它传给一个需要 int 类型参数的函数，也可用"++"运算符把它加 1，依此类推，如下述语句：

```
century = today.year / 100+1;
```

会执行一个 today.year 和 100 之间的整数除法（回忆 C 程序语言中的整数除法，其结果仍为整数，但是余数部分将被舍去。）

2.1.6 结构与函数

我们可以把整个结构当作一个参数传给一个函数，做法很简单，只要在调用函数时，把结构变量写在参数序列的位置便可。例如，假定有一个名叫 juliandate 的函数，用来计算一个存在 date 结构中的日期，这时只需用一条语句将整个结构传给函数即可，写法如下：

```
julian = juliandate(today);
```

在 juliandate 函数的头部（header）要进行适当的说明，告诉编译程序该函数需要一个 struct date 类型的参数：

```
int juliandate  (caldate)
struct date caldate;
 {
    int result;
     ...
    return (result);
}
```

上面的声明说明 juliandate 是一个函数，返回值是整数，它需要一个称为 caldate 的参数，该参数的类型是 struct date.

请记住在 C 程序设计语言中，调用函数时是传递参数的值（call by value），因此不论什么时候，传一个结构变量给函数时，函数无法对结构变量本身的值作任何永久性的改变，它只能更改调用函数时所产生的那份拷贝．因此在上面的例子中，juliandate 函数不能对 today 变量的值做任何改变，而只能更改 today 变量的拷贝，这份拷贝在调用函数时已存放到 caldate 变量中．

只要对返回值的类型做正确的声明，便可以从函数中返回整个结构．假定编写了一个称为 nextday 的函数，而其目的是按照一个 date 结构的日期参数，计算出该日期的下一日，我们希望返回整个表示新日期的 date 结构．这个 nextday 函数看起来应该如下：

```
struct date nextday (now)
struct date now;
 {
   ...
   return (now);
}
```

上述声明告诉 C 编译程序 nextday 是一个函数，返回一个类型为 struct date 的值，并需要一个同样类型的参数。这个函数会更改 now 变量，然后执行下面的语句，将更改后的结构传回原来调用的函数．

```
return (now);
```