李金山 著

C编程艺术

C BianCheng YiShu

山西出版集团 ● 山西人民出版社

- C程序的框架结构
- 接口函数
- 运算符及其多态性
- 输出和输入的关系
- 中英文录入测试系统的设计与实现

李金山 著

C BianCheng YiShu

C编程艺术

山西出版集团
山西人民出版社

图书在版编目(CIP)数据

C编程艺术/李金山著.—太原:山西人民出版社,2007.4
ISBN 978-7-203-05796-3

Ⅰ.C... Ⅱ.李... Ⅲ.C语言—程序设计 Ⅳ.TP312

中国版本图书馆CIP数据核字(2007)第040836号

C编程艺术

著　　者:李金山
责任编辑:员荣亮
装帧设计:王云翠

出 版 者:山西出版集团·山西人民出版社
地　　址:太原市建设南路21号
邮　　编:030012
电　　话:0351-4922220(发行中心)
　　0351-4922208(综合办)
E-mail: fxzx@sxskcb.com
　　web@sxskcb.com
网　　址:www.sxskcb.com

经 销 者:山西出版集团·山西人民出版社
承 印 者:太原市力成印刷有限公司

开　　本:787mm×1092mm
印　　张:12.125
字　　数:300千字
印　　数:1-300册
版　　次:2007年4月第1版
印　　次:2007年4月第1次印刷
书　　号:ISBN 978-7-203-05796-3
定　　价:36.00元

引 子

150 亿年前，诞生了我们的宇宙；100 亿年前，诞生了银河系；50 亿年前，诞生了太阳；46 亿年前，诞生了地球；40 亿年前，诞生了人类的始祖单细胞；450 万年前，诞生了类人猿，他们会用十个指头（十进制）计算打死老虎的数量，但他们的经验只能保存在内存（大脑）中，不能另存到外存（甲骨、竹子、绸缎等），死之前若不能言传身教给近亲，那他们积累的知识将会随之东流去；10 万年前，诞生了人这种动物；5 千年前，智力超常的人琢磨出了结绳记事等记忆棒或用石头、木棒等作为筹算和记忆的工具；960 年前，北宋的劳动人民“鼓捣”出了算盘及其一整套软件（口诀）。

1614 年，苏格兰数学家 John Napier 发明对数和骨制可进行四则运算的计算尺；1621 年，英国数学家 Willian Oughtred 根据对数原理发明计算尺，为两次工业革命做出了贡献，并成为当时工程师的身份符号；1642 年，法国哲学家帕斯卡（Pascal Blaise）发明加法机；1673 年，微积分发明者德国的莱布尼兹设计出一台能进行四则运算的手摇计算机，并将其克隆品送给康熙；1800 年，意大利的伏特发明电池；1822 年，剑桥大学数学教授查尔斯·巴贝奇（Charles Babbage）发明差分机。1834 年，他开始设计采用寄存器来存储数据的分析机。拜伦的女儿，世界上第一位程序员 Ada（美国军用语言，法国人 Jean Ichbiah1979 年中标）负责给其编程；1831 年“文盲”法拉第发现磁能生电，并造出首台发电机；1847 年，英国数学家乔治·布尔创立逻辑代数，为开关逻辑元件和逻辑电路的设计铺平了道路；1868 年，美国新闻工作者克里斯托夫·肖尔斯发明 QWERTY 键盘；1888 年，美国人赫尔曼·霍勒斯（1896 年创立制表机公司，1911 年又创建 CTR（计算制表记录）公司，1924 年，托马斯·沃森把 CTR 更名为 IBM）发明了制表机，采用穿孔卡片进行数据处理，并用电气控制技术取代了纯机械装置。

1904 年，英国人弗莱明（J. Fleming）发明真空电子二极管；1906 年，美国人德弗雷斯特发明电子三极管，真正标志着人类进入电子和无线通信时代；1937 年，德国的康拉德·朱斯（Konrad Zuse）制造出 Z-1 机电式计算机，这是第一台采用二进制的计算机，并于 1946 年发明计算机语言 Plankalkul；1938 年，信息论创始人美国数学家申农发表《继电器和开关电路的符号分析》，首次阐述了如何将布尔代数运用于逻辑电路，奠定了开关电路的理论基础；1940 年，控制论创始人美国数学家维纳提出计算机应该是数字式而不是模拟式，以电子元件构成并尽量减少机械装置，采用二进制而不是十进制，能存储数据。维纳 1948 年写的《控制论》对计算机和 AI 的发展产生深刻影响；1943 年 10 月，英国的图灵攻关组推出巨人（Colossus）计算机，用于破译德军密码；1944 年，美国数学家艾肯（H.Aiken）在 IBM 的资助下，研制成功机电式计算机 Mark-I，在哈佛大学正式运行。艾肯说，Mark-I 的运行使巴贝奇的梦想变成现实。它采用继电器而不是齿轮传动装置，以穿孔纸带传送指令。de-bug、千年虫、世界上第一个将高级符号语言变换为机器语言的编译器 A-0（1952）以及 COBOL（1959）语言的始作俑者，海军少将 Grace Hopper 负责给 Mark 机编程。

1945 年底采用十进制的 ENIAC 诞生，1946 年 2 月 15 日，ENIAC 做了第一次公开表演；1947 年 12 月 23 日，贝尔实验室的肖克利（William B. Shockley）、布拉顿（John Bardeen）、巴丁（Walter H. Brattain）创造出晶体管，一起荣获 1956 年诺贝尔物理奖；1949 年英国剑桥大学数学家 Wilkes 推出世界上第一台能存储程序的计算机 EDSAC，输入输出仍是纸带；1951 年 6 月 14 日埃克特和莫克利再次合作制造的 UNIVAC 是世界上第一个进行批量生产的计算

机，从而使电脑走出实验室，人类社会进入计算机时代；1952 年 1 月由计算机之父冯·诺依曼设计的 EDVAC 问世，它比 ENIAC 速度快 10 倍；1953 年，IBM 公司推出 IBM701，在商战中击败 UNIVAC，从而奠定了 IBM 在计算机产业界的霸主地位；1958 年，美国物理学家基尔比和 Intel 创始人诺伊斯同时发明集成电路；1971 年 1 月，Intel 公司的霍夫研制成功世界上第一块 4 位微处理器芯片，标志着微机时代从此开始；1981 年 8 月 12 日，现在司空见惯的个人电脑（微机最初的称呼）的鼻祖 IBM PC 机从 IBM 公司的流水线上缓缓涌入市场；2005 年 11 月，IBM 研制的“IBM 蓝色基因/L”巨型机峰值速度达到 367 万亿次/秒；2007 年 1 月英特尔推出酷睿 2 四核处理器，含有 5.8 亿多个晶体管，主频为 2.4GHz。

给计算机编程序最初使用机器语言，20 世纪 50 年代中期开始使用汇编语言；1957 年世界上第一个高级语言 FORTRAN（FORmula TRANslating）在 IBM 诞生；1959 年，商用编程语言 COBOL 出现；1960 年，ALGOL60 推出，首次引进了局部变量和递归的概念。ALGOL60 离硬件较远，不宜用来编写系统软件；1963 年，英国剑桥大学在 ALGOL60 的基础上推出了 CPL（Combined Programming Languange）语言，它要靠硬件近一些；1967 年剑桥大学的 Matin Richards 在 CPL 的基础上推出了 BCPL（Basic Combined Programming Languange）语言；1969 年，贝尔实验室的肯尼斯·汤普森和丹尼斯·里奇用汇编语言开发出 UNIX。1970 年，Ken Thompson 在 BCPL 基础上设计出了接近硬件的 B 语言，并用 B 语言重写了 UNIX；B 语言功能有限，1972 年，D.M.Ritchie 在 B 语言的基础上设计出了 C 语言。1973 年，K.Thompson 和 D.M.Ritchie 合作，用 C 重写了 UNIX（UNIX 第 5 版）。

C 语言是开发 UNIX 的“副产品”，它将汇编语言可以直接对硬件进行操作和高级语言的可读性强两个优点融于一身，因此被称为可移植的汇编语言或中级语言，写系统软件是 C 的长项。如，UNIX（及其变种 Linux、BSD、Solaris、MacOS X）、Windows 的 API、TCP/IP 协议栈、VFP、JVM、Photoshop、MatLab、ODBC、Ruby、多数 C 编译器以及一些黑客工具等都是采用 C 开发的；1980 年，贝尔实验室的 Bjarne Stroustrup 博士（1993 年获 Grace Murray Hopper 奖，ACM 院士）及其同事对 C 语言进行了扩充，称为“带类的 C”，1983 年更名为 C++。C 语言是 C++的子集，C 代码不经修改就可以在 C++环境中编译。在 C 基础上发展起来的语言还有 Java（1995 年由 SUN 公司的 Bill Joy 和 James Gosling 研制）、C#（2002）等。

Anders Hejlsberg（微软.NET 首席架构师，Delphi、J++、C#之父）用汇编语言撰写了 Turbo Pascal 编译器，并且和 Nicklaus Wirth 的学生 Philipe Kahn 在 1983 年创办 Borland 公司。Borland 公司在 1987 年发布 Turbo C 1.0；1990 年发布 Turbo C/C++1.0（也被称为 Borland C/C++）；1992 年发布 Borland C/C++3.0；1993 年发布 Borland C++4.0；1997 年发布 C++Builder 1.0；2003 年发布 Borland C++Builder X。本书所有源代码均在 TC++3.0 下调试通过。

本书是作者多年从事 C 语言、数据结构、软件工程、操作系统教学实践和实际工程项目经验的一个总结，全书采用实例加注释的思维模式，以满足一般人由特殊到一般、从实践到理论、从具体到抽象的自然思维习惯。最后给出一个作者开发的案例，以起立竿见影的作用。

感谢山西人民出版社的责任编辑员老师，他的指点促使我把独享的暂存于我的人脑中的数据另存到了纸介质上。感谢我的朋友和家人，你们的爱和支持是本书得以完成的保障。感谢 baidu 和 Internet 上的网友，没有瞬间即可获得的你们的数据，本书的完成恐怕还得些时日。

限于作者的经验和水平，书中难免有错误和不足，欢迎读者和各位专家批评指正。

李金山

2007 年 3 月于太原寓所

试读结束，需要全本PDF请购买 www.ertongbook.com

目　录

第一章 C 程序的框架结构 1

1.1 用 C 写程序的思维框架 1
1.2 变量面面观 6
1.3 常量的 5 种表示法 15
1.4 C 程序内存框架结构 17

第二章 接口函数 19

2.1 printf()、fprintf()、scanf()以及 fscanf()库函数 19
2.2 getchar()、putchar()、getc()以及 putc()宏和 gets()、puts()库函数 25
2.3 getch()、getche()、cputs()、windows()以及 bioskey()库函数 26
2.4 fgets()、fputs()以及 fopen()库函数 28
2.5 fread()、fwrite()以及 fflush()库函数 33

第三章 运算符及其多态性 38

3.1 *号的 9 种用法 38
3.2 &号的 6 种用法 62
3.3 {}号的 6 种用法 71
3.4 ;号的 4 种用法 82
3.5 :号的 5 种用法 85
3.6 ()号的 5 种用法 90
3.7 –号的 7 种用法 93
3.8 <和>号的一些用法 96
3.9 下标运算符[]及二维数组的 5 种表示法 99
3.10 其他运算符 106
3.10.1 +号的 6 种用法 106
3.10.2 ,号的 3 种用法 106
3.10.3 %号的 4 种用法 108
3.10.4 #号的 4 种用法 111
3.10.5 ?号的 3 种用法 112
3.10.6 |号的用法 117

第四章 输出和输入的关系 123

第五章 中英文录入测试系统的设计与实现 133

5.1 系统需求分析 133
5.2 系统设计 135
5.3 系统实现 137
5.4 系统测试 181
5.4.1 系统兼容性测试 181
5.4.2 系统测试用例 181
5.4.3 系统存在的问题 183

附录 I 历届图灵奖获得者简介 184

附录 II 杂感四篇 186

参考文献 188

第一章 C 程序的框架结构

计算机系统由硬件系统和软件系统构成，硬件由 IBM、HP、Dell、Acer 以及联想这样的工厂制造出来，叫裸机。软件是由微软、Oracle 以及用友等公司的程序员用计算机语言编写出来的，叫程序。程序和编写该程序的文档一起构成软件系统。裸机安装上操作系统就构成了第一层虚拟机，计算机的软硬件资源由操作系统来管理。在操作系统之上再安装应用软件，就构成了第二层虚拟机，用户一般是与这层计算机打交道的。应用软件提供一个输入数据的界面，用户通过这个界面将要加工的原始数据（文字、数值、声音、图片、图形、动画、电影等）输入给这个软件，然后它的加工部分对数据进行变换，最后将计算结果输出给用户。

软件是被记录下来的思维的逻辑推理过程，是计算机硬件的“学历”，是计算机系统的灵魂。没有安装软件的裸机就像一台接收不到电视信号的电视机一样，其“本事”不能被挖掘出来。因此，硬件的价值与安装到其上的软件的功能成正比，人借助软件间接让硬件“劳动”。

计算机硬件的制造现在已经实现批量生产了，但软件的开发还需要人来写，目前还不能实现让计算机来编写软件。用来写软件的计算机语言多达 2000 种，C 语言能在众多计算机语言中脱颖而出，靠的就是其既能写系统软件又能写应用软件的特长。系统软件一般直接与硬件打交道，编写难度大。应用软件是用来解决特定领域的数据加工问题的，现在通常的做法是用 C 开发一个系统软件，然后用这个系统软件开发应用软件。例如用 C 开发出 VFP 系统，用户使用 VFP 系统开发其领域的应用软件，如考场安排系统、成绩管理系统以及售票系统等。

1.1 用 C 写程序的思维框架

下面给出第一个 C 程序，命名为 c1.c，C 源程序的扩展名为 c。

```
main() //main 函数的功能是调用其他函数，给其传递数据，并接收其传来的数据。
{void drawsomething(char,int);        //对该函数进行声明
 drawsomething(' ',2);
 drawsomething('#',1);printf("\n"); //括号内参数是实参
 drawsomething(' ',1);
 drawsomething('#',3);printf("\n"); //实参间用逗号分隔
 drawsomething(' ',0);
 drawsomething('#',5);printf("\n"); //字符常量用单撇括起
} //分号是语句的标志，少了它就变成表达式了。
 /*下面函数的功能是输出 n 个由参数 c 指定的字符*/
void drawsomething(char c,int n) //c 和 n 是形式参数，简称为形参，char 即 character。
{int i; //定义变量 i 是 integer 型，C 规定变量必须先定义，后使用。
 for(i=0;i<n;i++) printf("%c",c); //循环 n 次，每次输出一个字符。
```

} //for 循环中的分号是分隔符，第三章会讲到，同一符号在不同上下文中含义迥然。

由/*和*/括起来的部分是 C 和 C++的多行注释语句，//是 C++的行注释语句，本书所有程序都是在 Turbo C++3.0 IDE（Integrated Development Environment）下调试和编译。注释语句也叫内部文档，一般分两种：一种是放在源码的最前面，对该程序进行整体注释，介绍其功能、I/O 接口中参数的含义、权限要求、算法的简单描述、调用了哪些函数、编者、创建日期、修改日期以及审核日期等；另一种是行间注释，对可能引起歧义的语句进行说明。使用注释的目的是为了让读者快速理解程序，注释量一般要占到全部程序代码量的一半左右。

该程序由 main 和 drawsomething 两个函数构成，每个函数完成一个特定的功能。一个程序有且只能有一个 main 函数，main 函数是程序的入口函数，其位置任意。程序总是从 main 函数开始执行，main 函数执行完了，整个程序也就执行完了。所以，在拿到一个要进行数据变换的问题后，首先要做的就是将要解决的问题分解为若干功能单一的函数，以使每个函数都拥有“一技之长”。接下来 main 函数按照输入数据、变换数据、输出数据的顺序调用相应的函数，由其完成具体任务。当然，函数之间可以互相调用，但不能调用 main 函数。这就是用 C 解决数据加工问题的思维框架结构。

通过上面的示例可以看出，一个函数由函数首部和用花括号括起来的函数体两部分组成。

函数首部由函数返回值类型、函数名(类型 形参，类型 形参，…)构成。上面的 main 函数没有形参，叫无参函数，没有返回值类型，默认为 int 型，没有返回值语句，则执行完该函数后系统返回一个随机数。drawsomething 函数的返回值类型是 void，说明不能有返回值。形参是定义函数时的形式参数，必须是变量，实参是调用函数时的实际参数，必须有明确的值。调用时，按从右到左（TC++是这样，但 Fortran 等语言按从左向右的顺序）的顺序将实参的值赋给形参，所以，要求实参和形参的类型、个数、顺序要对称，否则，或者是编译通不过，或者是计算结果不正确。现在让我们故意犯个错误，将 main 函数第 3 行的，2 去掉，编译后给出“too few parameters in call to 'drawsomething' in function main”的提示，将实参 2 改为 2.9，运行结果不变，因为形参 n 是整型，它只要整数部分。

函数体内要先写局部变量定义、被调函数声明，然后再写函数调用、判断、循环、赋值、return 等语句。函数体也可以是空语句，即只有一个函数首部后跟一对花括号，叫空函数。根据 C 的思维框架结构，我们在处理一个大的任务时，首先要做的是将大任务分解为小任务，如此分解下去，直到每个小任务只有一个功能，我们暂时就用这个空函数代表这个小任务，待分解合理了，任务之间的关系理顺了，再把空函数细化为真正的函数，然后上机调试、编译为 EXE 文件，就可以作为软件使用了。如，求两个正整数的最大公约数和最小公倍数，可以分解为如下三个函数：

```
int max(int,int){} //max 函数负责求最大公约数，返回值类型是 int 型。
min(int,int){}     //min 函数负责求最小公倍数，省略了返回值类型。
void main(){}  //负责接收待求数，调用 max 和 min，接收其返回值，最后输出结果。
```

下面是细化后的三个函数：

```
int max(int bcs,int cs) //bcs 是被除数拼音首字母缩写，cs 代表除数。
{int yushu; //yushu 代表余数，zdgys 代表最大公约数，zxgbs 代表最小公倍数。
 while(cs!=0)          //while 循环结构，读作当条件为真时，执行循环体。然后返回到
   {yushu=bcs%cs;      //条件处继续判断，当条件仍为真时，继续执行循环体，如此循环，
    cs=yushu;          //当条件为假时，跳出循环，继续执行后继语句。当循环体语句多
```

```
    bcs=cs;              //于一条时用{}括起来，构成复合语句，其功能相当于一条语句。
  } //下面的 return 语句是上面循环结构的后继语句
 return(bcs); //将加工结果返回到调用处，return 后面的圆括号可以省略。
} //将源代码写成锯齿状的目的是增强程序的可读性，程序的结构层次清晰，一目了然。
 min(int bcs,int cs)     //形参是局部变量，与生活中的临时工类似。
{int zdgys=max(bcs,cs); //在定义局部变量的同时调用 max 函数，其返回值赋之。
 return bcs*cs/zdgys;    //先计算，再返回结果。
}
 void main() //无参函数，且不能有返回值，有返回值也不允许使用它。
{int bcs=26,cs=4;        //定义局部变量的同时，给其赋值。
 int zdgys=max(bcs,cs); //调用 max 函数，并将其返回值给 zdgys。
 int zxgbs=min(bcs,cs); //调用 min 函数，并将其返回值给 zxgbs。
 printf("\nMax=%d,Min=%d\n",zdgys,zxgbs); //输出结果
 return 6; //6 可略，写上此句不算错，读者可编写一个使用此返回值的程序，编译。
}          //时会给出“Not an allowed type in function functionname”的提示。
```

在 TC++系统的 IDE 中输入源码，保存为 c2.c，也可以在类似于 NotePad 这样的文本文件编辑器中输入源码，然后在 TC++系统中打开，编译通过后，生成 c2.obj（Object）二进制码文件，再和库函数 Link 成可独立执行的 c2.exe（execute）机器码文件。可以到 XP 的命令提示符下运行，输出结果为 Max=2,Min=52。

为什么要把具有某一功能的一小段程序称为函数呢？仔细剖析 max 这段程序可以看出，其功能是将传进来的两个正整数（自变量，其所有可能的值构成定义域）经过一定的变换规则加工为最大公约数（因变量，其所有的值构成值域），称这段程序为函数是比较恰当的。

若被调函数处于调用函数之前，或函数的返回值类型是 int 型，声明语句可以省略。上面的 min 函数直接调用了 max 函数就是这个原因。函数的声明也可以集中写在所有函数之前，程序的开始部分。下面的程序 c3.c 就是这样。

```
 long power(int,int);//函数的声明放在所有函数之前，可不必声明而直接调用。
 void main()
{int a=2,b=8; //16 位计算机 int 型变量占 2 字节，取值范围为-32768～32767。
 printf("result is:%ld\n",power(a,b)); //%ld 表示此处放一个长整型数，l 即 long
}                                      //d 即 decimal，\n 表示换行。
 long power(int x,int n) //long int 型中的 int 可省略，该函数的功能是求 X^n。
{int i; //定义局部变量，也叫自动变量，用 auto 修饰，可以省略。
 long result=1;//16 位机 long 型变量占 4 字节，取值范围-2147483648～2147483647。
 for(i=0;i<n;i++) //for 循环结构，循环 n 次
   result*=x;     //与 result=result*x 等价，但前者更简洁。
 return result;
}
```

为何 printf 函数没有声明，也不见其函数体，用户也没有编写这个函数，却可以拿来直接调用呢？原因是它是库函数，不是用户自定义函数。库函数是由 C 系统开发人员事先写好并编译成二进制代码后保存到库文件中，并随系统一起提供的。库函数一般完成和硬件直接

打交道的底层功能，或复杂的计算。前者如输出计算结果的printf函数，将键盘输入的数据送到内存指定单元的scanf函数，清屏函数clrscr等，后者如sqrt、sin函数等。由于printf、scanf、clrscr函数使用频繁，TC++编译系统允许直接使用它们而不必声明，其他库函数在使用前仍需声明，但与用户自定义函数的声明方法略有不同，程序c4.c说明了这种差异。

```
#include<math.h> //包含命令，下面三个库函数首部信息保存在这个头文件中。
void main() //库函数sin()、log10()、sqrt()的返回结果作为printf()的实参
{printf("sin(3)=%lf, ",sin(3));          //求3的正弦，l即long，f即float。
printf("log10(10)=%lf, ",log10(10)); //求log10^10，lf表示双精度数。
printf("5's square root is:%lf\n",sqrt(5)); //求5的平方根
} //输出结果为sin(3)=0.141120，log10(10)=1.000000，5's square root is:2.236068
```

#inlucde<math.h>是编译预处理命令，编译预处理命令（不能叫语句）不是C语言的组成部分，但却是C编译系统的重要组成部分，使用它主要是为了提高程序的可读性。所有编译预处理命令都以#号开头且后面紧跟define、undef、pragma等命令，命令结束处勿写分号。

在正式编译这段程序前，编译预处理程序找到math.h文件（该文件是个文本文件，保存在安装文件夹下include文件夹中），并将其内容复制一份粘贴到该命令处以取代该命令，然后编译器对经编译预处理程序处理后的程序进行编译、连接、生成可执行文件。*.h(h即head)文件一般称为头文件，这些文件中包含有库函数的首部信息和系统定义的常量、类型等。程序中用到库函数时，须将相应的头文件写入到程序的开始部分，以便在进行编译预处理时，将这些文件的内容包含进来。

顺便提一下，<math.h>也可以写做"math.h"。用双撇时，编译预处理程序先去当前文件所在的文件夹下寻找，若找不到，再到系统配置文件所指定的文件夹中查找，查找用户自定义文件一般采用这种方法。用尖括号时，编译预处理程序直接到系统配置文件所指定的文件夹中查找，查找系统自带的文件一般采用这种方法。

让我们写个程序，验证和体验一下上面的话。在XP的命令提示符下发如下命令：

cd\tcplus /*进入根下的tcplus文件夹（folder），文件夹在DOS时代叫目录（directory）。假定C系统安装在此文件夹下。顺便提一下，发cd/?命令可获得cd用法的帮助信息，发cd/?|more可分屏显示，否则当帮助信息大于一屏时，将滚屏显示，用户只能看到最后一部分。发help|more可显示所有DOS内部命令，按↑↓键可浏览已发命令集，按F1～F10会出现什么？读者若感兴趣的话，可一一试验、体验。这些内部命令都可以用C编写出来*/

```
copy con practise.h //con即console，此处代表键盘。将按键内容写入practise.h。
int a=9;
按Ctrl+Z回车或按F6回车 //保存文件
type practise.h回车 //type命令能显示文本文件practise.h的内容。
copy con okay.c //注意DOS文件命名的8.3规则
#include "practise.h"回车 //由于是用户自定义文件，故采用双撇号。
main(){printf("a=%d\n",a);} 按F6回车
type okay.c回车 //显示文本文件okay.c的内容。
bin\tc okay.c回车 //运行当前文件夹下的bin子文件夹下的tc.exe并打开okay.c
```

在TC++IDE下将okay.c制作成EXE文件。这个过程其实包含了编译预处理、编译、连

试读结束，需要全本PDF请购买 www.ertongbook.com

接三个过程。其中，源文件 okay.c 在编译预处理后其内容如下：

```
int a=9; //practise.h 文件中的内容已经复制到此处并替换掉#include "practise.h"
main(){printf("a=%d\n",a);}
```

接着，对编译预处理后的内容进行编译、连接，生成 okay.exe 文件。在 IDE 下选择 File 菜单下的 DOS shell 暂时返回到 DOS，发 okay 回车，可以看到输出结果为 a=9，发 dir okay.* 回车，可以看到 okay.c、okay.obj 和 okay.exe 三个文件。发 exit 返回到 IDE。

也许有些读者会好奇，上面的 cd、copy、type 以及 bin\tc 命令后都带有参数，这是如何实现的？其实这个很简单，我相信你看了下面的“升级版”okay.c 后也会模仿出点什么来。

```
#include "practise.h" //假定该文件保存在 C:\tcplus 下，叫 okay.c。
#include <stdlib.h> //库函数 atoi()的首部信息放在该文件中，此处用尖括号较好。
main(int argc,char **argv) //argv 最初保存 C:\tcplus\okay.exe 串的首址。
{if(argc<2)  //命令行参数间用空格分隔，参数的个数（包括软件名）保存在 argc 中。
 {clrscr(); //清屏库函数
  printf("okay Software Version 1.1 Copyright(c) 2008 LJS.\n");
  printf("Syntax is:okay number e.g. okay 81 then,the output will be 9+81=90");
 }   //若用户只输入 okay 回车，argc 的值是 1，则执行此复合语句（共三条）。
 else //否则执行下面的复合语句（共两条）。复合语句的功能仅相当于一条语句。
 {argv++; //argv 指向下一个参数并记住它的首址，下面的*argv 也是该参数的地址。
  printf("%d+%s=%d\n",a,*argv,atoi(*argv)+a); //%s 表示此处要放一个串
 } //上面的 atoi()库函数将*argv 指向的串（第二个参数）转换为 int 型并返回此值。
}
```

生成 EXE 文件后，在 DOS 下输入 okay 回车。由于参数只有一个，即 C:\tcplus\okay.exe，故 argc 的值是 1，argc<2 为真，执行清屏函数和两个 printf 函数，其输出结果如下：

```
okay Software Version 1.1 Copyright(c) 2008 LJS.
Syntax is:okay number e.g. okay 81 then,the output will be 9+81=90
C:\TCPLUS>
```

在 DOS 下输入 okay 121 回车。此时有两个参数，C:\tcplus\okay.exe 和 121，argc 的值为 2，argc<2 为假，执行 else 部分。argv 被定义为指向指针的指针，我们权且不管什么是指针的指针，总之，argv 最初是指向第一个参数的，argv++后，指向第二个参数，即串 121，库函数 atoi()将串 121 转换为数值 121，然后与 a 的值相加，其输出结果为 9+121=130。

函数的声明就是将函数的首部复制一份，参数可以省略。声明的目的是将函数名、返回值类型、参数类型、个数等信息通知编译系统，以便在调用该函数时编译系统按此进行对照检查。如，将程序 c3.c 声明部分的 long 改为 int，编译时，将给出“Type mismatch in redeclaration of 'power'”的错误提示。

函数在调用时，可以作为表达式出现，如 zdgys=max(bcs,cs)语句，也可以作为语句出现，如 drawsomething('#',1)语句，甚至于以实参的形式出现，如下面的 c5.c 程序所示。

```
maximum(int a,int b) //该函数的功能是返回 a 和 b 中的大数。
{return a>b?a:b;}      //?:是三目运算符，?号前的表达式为真返回 a，否则返回 b。
main()
{int a=10,b=45,c=18,max;
```

```
    max=maximum(a,maximum(b,c));//maximum(b,c)的返回值作为maximum(a,?)的实参
    printf("Max=%d",max);          //输出结果为Max=45，max的值放在%d所在的位置。
}
```

函数直接或间接地调用自身，叫递归调用，如A函数内部又调用了A函数，或A函数调用了B函数，B函数又调用了A函数。程序c6.c演示了直接递归调用的一些用法。

```
void main()
{long add(long);long factorial(long); //函数声明
 long temp=123456789,n=7; //准备数据
 printf("sum=%ld, ",add(temp)); //调用add函数，其返回值是库函数printf的实参。
 printf("%ld!=%ld\n",n,factorial(n));//factorial函数的返回值是printf的实参
}  //输出结果为sum=45，7!=5040
 long add(long n) //add函数的功能是求数n各位数字之和。如1234，1+2+3+4是10。
{long yushu,shang,sum=0;
 yushu=n%10; //求余数
 shang=n/10; //求商，两个整数相除，商是整数。如14/5的结果是2。
 if(n>0)
  sum=yushu+add(shang); //递归调用add
 return sum;
}
 long factorial(long n) //该函数的功能是求n!
{long s;
 if(n==0) s=1;
 else
  s=n*factorial(n-1);//递归调用factorial函数
 return s;
}
```

1.2 变量面面观

程序c6.c的main、add和factorial函数中都有变量n，这是同一个变量吗？不是。函数内部定义的变量包括形参，都是局部变量，它只存在于定义它的函数中，在调用该函数时在内存创建它，该函数执行完，局部变量的空间被释放，局部变量也就不复存在了。所以，上述变量n是同名的三个不同变量。

既然有局部变量，就一定有全局变量（或称为外部变量），全局变量是定义在函数之外的变量，其作用域从定义处开始到本文件末结束。程序c7.c、c8.c、c9.c演示了这两种按作用域（从空间角度）划分的变量之间的一些用法。

```
extern m;     //此处的extern是将外部变量m的作用域扩大到从此处开始到本文件末
void main()   //这是程序c7.c的入口函数
{int m=0,n=5; //定义局部变量m和n
```

```
 m+=2;
 local_public(n++); //调用函数，n 是后加，故先传 n 的值，然后 n 的值加一。
 printf("main m=%d,n=%d\n",m,n); //此处的 m 和 n 都是 main 中的局部变量
 local_public(--n); //调用函数，n 是前减，故 n 先减一，再传 n 的值。
 printf("main m=%d,n=%d\n",m,n); //此处的 m 和 n 都是 main 中的局部变量
}
 local_public(int n)
{m++; //这是全局变量 m
  { //此处的花括号是一复合语句，下行定义的 m 只存在于此复合语句中。
   int m;
   m++; //这是此复合语句中定义的局部变量 m
   n++; //这是形参 n
  } //程序执行到此处，该复合语句中定义的 m 被释放。
  n++; //这是形参 n
  m++; //这是全局变量 m
  printf("local_public m=%d,n=%d\n",m,n); //输出的是全局变量 m 和形参 n
}
 int m=1; //定义外部变量 m，其作用域为从此处开始到本文件末。
 运行结果为：
 local_public m=3,n=7
 main m=2,n=6
 local_public m=5,n=7
 main m=2,n=5
 程序 c8.c 源码如下：
 extern m; //将外部变量 m 的作用域延长至此处
 int fn(int m,int n) //形参 m 和 n 是局部变量，只存在于定义它的 fn 函数中。
{m+=n; //这是形参 m 和 n
 return m++; //这是形参 m，由于是后加，故先返回 m 的值，m 再加一。
}          //下行定义外部变量 m，外部变量也叫全局变量，类似于国营单位的正式工，
 int m=1; //存放在静态区，只要程序没结束，它就一直存在，局部变量存放在动态区。
 void main()
{int n=0; //定义局部变量 n，其作用域是 main 函数。
  {        //局部变量与创建它的函数共生死，全局变量与创建它的程序共存亡！
   int m=1; //此局部变量 m 的作用域是该复合语句。该语句是 m 的出生语句。
   n+=fn(m,m++);//此 n 是 main 中的 n，此 m 是上行的 m。TC++规定实参从右向左传值
  } //程序执行到此处，该复合语句中的 m 在动态区占用的空间被回收，此 m 死亡。
 n+=fn(m,++m); //此行和下行的局部变量 n 是 main 中的 n，m 是外部变量 m。
 printf("%d,%d,%d,%d\n",m,m++,n,n--); // TC++规定实参从右向左传值
} //输出结果为 3，2，6，7
 程序 c9.c 源码如下：
```

```
#define ON 0 //宏定义命令，习惯上宏名用大写，变量用小写，以提高程序的可读性。
int max=0,min=0; //定义两个全局变量并初始化，max 保存最大数，min 保存最小数。
//average 函数返回平均值，并将求出的最大数和最小数分别保存到 max 和 min 中。
double average(int *p,int n)//定义 p 为指针变量，指针变量保存普通变量的地址。
{int i;     //TC++规定 double 型变量占 8 字节，精度 15 位，取值范围约±1.7×10^±308
 float sum; //TC++规定 float 型变量占 4 字节，精度 7 位，取值范围约±3.4×10^±38
 sum=max=min=*p++; //*p++与*p;p=p+1 等价，前者简洁但不易理解，读古文也是这样。
 for(i=1;i<n;i++,p++) //p++读作 p 指向下一个单元
  {if(max<*p) max=*p; //*p 读作 p 指向的单元的值，此处的 p 指向数组 a。
   if(min>*p) min=*p; //min=*p 读作把 p 指向的单元的值取出来赋给 min
   sum+=*p;
  }              //float 型和 int 型混合运算，两者都要先转化为 double 型，计算结果
 return sum/n; //也是 double 型。这样做的目的是确保在计算过程中不丢失数据。
}
 void main()//下行定义并初始化数组 a，数组名 a 是常量且保存数组的首址。
{int a[]={12,32,-3,54,5,-6,37,78,-9,33};
 if(ON)
  printf("average=%lf,max=%d,min=%d\n",average(a,10),max,min);
 else
  printf("max=%d,min=%d,average=%lf\n",max,min,average(a,10));
} //输出结果为 max=78，min=-9，average=23.300000
 请将#define ON 0 中的 0 改为非零，如 1，再运行程序，输出结果如下：
 average=23.300000，max=0，min=0
```

为什么会是这样的结果呢？前面提到过，printf 函数的实参按从右向左传值，当 ON 为 0 时，编译预处理后，源码中的 ON 已经被 0 替换了，这叫宏展开或宏替换，而 0 在 C 系统中被当作判断时，表示假（非零是真），故，程序执行 else 部分的 printf 语句。printf 函数则先调用 average 函数，实参 10 赋给了形参 n，数组 a 的首址赋给了形参 p。average 函数对 p 指向的数组中的数据进行加工，求出最大数、最小数和累加和，并分别保存到 max、min 和 sum 中，最后返回平均值到调用处，作为 printf 函数的实参输出。average 函数执行完毕，局部变量 p、n、i 和 sum 所占空间已被释放，这四个变量已不复存在，而全局变量 max 和 min 依然存在，且保存了最大数和最小数。所以，输出结果正是我们所希望的。现在再来看 0 被改为 1 之后的情况，编译预处理后，源码中的 ON 已经被 1 替换掉了，1 是非零，是真，故，执行 printf("average=%lf,max=%d,min=%d\n",average(a,10),max,min);语句，printf 函数按从右向左的顺序，先输出 min 的值，再输出 max 的值，最后调用 average 函数，求出最大数和最小数，并让 max 和 min 保存，最后返回平均值，原来如此！

此处的宏 ON 起到了开关的作用，两个 printf 语句最终都被编译到了 EXE 文件中，只不过只有一个被执行到了。能不能根据 ON 的值，只将符合条件的 printf 语句编译到 EXE 文件中呢？这样可以去掉多余的代码，使 EXE 文件的字节数减少或实现其他目的。用下面的代码替换程序中的 if 结构即可。

```
#if ON //ON 是宏，在编译预处理之后，ON 会被它代表的串替换掉。
```

```
    printf("average=%lf,max=%d,min=%d\n",average(a,10),max,min);
  #else
    printf("max=%d,min=%d,average=%lf\n",max,min,average(a,10));
  #endif
```

这是条件编译命令，编译预处理程序会根据 ON 的值将两个 printf 中的一个编译到 EXE 文件中。为了加深对条件编译的理解，这里再给出一个让计算机随机出两个两位以内的正整数相加或相乘的程序，要求演示版只能做加法题，零售版都可以做且交错进行。源码如下：

```
  #define RETAIL 1    //这里约定 1 表示演示版，0 表示零售版。
  #include<stdlib.h> //库函数 rand()和宏 randomize()的有关信息放在该文件中
  #include<time.h> //库函数 time()的首部信息放在该文件中，randomize()用到该函数。
  void main() //time()函数能返回自 1970 年 1 月 1 日零时零分零秒到此刻所走过的秒数
  {char s[10]; //定义由 10 个字符型元素组成的一维数组，用来保存用户的答案。
  int num1,num2,result,t; //num1 和 num2 保存随机产生的数
  randomize(); //这是定义在 stdlib.h 中的宏，打开 stdlib.h 文件可看到它的定义如下：
  /*randomize(void) { srand((unsigned) time(NULL));*/
  //NULL 是定义在 stdio.h 中的宏，其值为 0。打开 srand()的帮助信息，对其解释如下：
  //Initializes random number generator.
  //void srand(unsigned seed);
  //Prototype in stdlib.h. srand does not return a value. See also rand randomize
  do //do 循环结构，当 while 后面的表达式为真时，一直执行循环体。
  {result=0; //result 为 0 表示没有做对，为 1 表示做对了，默认为 0。
   num1=rand()%100; //rand 函数返回随机产生的 0～32767 之间的整数。
   num2=rand()%100; //去掉 randomize 语句，可看到每次运行程序产生的随机数都一样。
   #if RETAIL //条件编译命令形式一
    #undef RETAIL //删除宏 RETAIL
   #endif
   #ifndef RETAIL //条件编译命令形式二
    if(1) //演示版
   #else
    if(num1%2) //零售版，产生交错效果。
   #endif
   printf("%d+%d=",num1,num2); //出加法题
   #ifdef RETAIL //条件编译命令形式三
    else
    printf("%d*%d=",num1,num2); //零售版，出乘法题。
   #endif
   gets(s); /*gets 库函数首部信息在 stdio.h 中，功能是将键盘输入的数据存放到数组
s 中，以回车键作为结束标志，但回车符不放到 s 中。用户输入的答案通过它保存到 s 中*/
   t=atoi(s); //atoi 库函数头部在 stdlib.h 中，功能是将字符数组中的字符转换为整数。
   #ifndef RETAIL
```

```
 if(1) //演示版
#else
 if(num1%2) //零售版
#endif
 {if(num1+num2==t) result=1;} //判题，若正确，则让 result 保存 1。
#ifdef RETAIL
 else
 {if(num1*num2==t) result=1;} //零售版，判题。
 #endif
 if(result) //若 result 为 1，则做对了，否则，做错了。
  printf("OK,Enter -1 End.\n"); //做对了
 else
  printf("Wrong!\n"); //做错了
}
while(t!=-1); //勿省略分号，它是 do 结构的一部分。此处的判断若为真，则继续循环。
}//while(){}与 do{}while();结构的唯一区别是：后者“先斩后奏”，前者“先奏后斩”。
```

该程序在 RETAIL 分别取 1 和 0 时，运行结果大相径庭，为什么？只要对两种情况下编译预处理后各自的源码进行分析，谜团自会“不攻自破”。

“局部变量们”是共享内存动态区的，这样做不仅节约了内存空间，而且也解决了不同函数间出现同名变量的问题。而“全局变量们”独享内存的静态区，只要程序不退出，分配给它们的“一亩三分地”就会被它们一直“霸占”着。所以，从时间角度看，动态区的变量“短命”，静态区的变量“长寿”。因此，我们可以将各函数都需要的一些数据保存到全局变量中，但过多地使用全局变量，使函数间的偶合程度增强，降低了函数的独立性，为程序将来的维护（排除用户在使用过程中暴露出来的在测试时没有发现的错误或增加新的功能，卸下过时的功能等）埋下了“祸根”，故全局变量要慎用。

那“程序们”放在什么地方呢？计算机只认识机器码，用 C 写的源码经过编译、连接后生成的机器码是以文件的形式保存在硬盘等存储介质上。运行该程序，就是将这个机器码文件的内容复制一份存放到内存的程序区（也叫代码区），然后 CPU 从代码区取出指令，分析指令，执行指令，再取下一条指令，如此循环，直到程序结束。在程序运行期间，计算部分由运算器负责完成，运算需要的数据存放在内存的动态区或静态区（register 型和常量存放在 CPU 的寄存器中），需要由控制器发出读命令，从指定的内存地址中读出到运算器的寄存器上。计算完毕，控制器发出写命令，将结果写入到指定的内存地址所在的单元中。源码中的变量名在编译后的机器码中都变成了内存地址，源码中使用变量名主要是为了增加程序的可读性，如此而已。

前面程序中在定义局部变量时，都没有写 auto（可译作动态或自动）存储类型说明符，auto 是用来告诉编译系统，这些变量要放到动态区，没写，说明默认的就是它。用 register 存储类型说明符修饰的局部变量存放在 CPU 的寄存器中（比起住在内存的变量兄弟们，此兄挨的“皇帝”近，谁让人家被召唤的勤呢），这样做要比到内存中去读它速度快几个数量级。不过，由于 CPU 中寄存器的数量有限，因此，并不是每个被 register 修饰的局部变量都会被编译系统安排到寄存器上（编译后，被安排到寄存器上的变量名已经替换成了寄存器名）。另

试读结束，需要全本PDF请购买 www.ertongbook.com