HOPE

高级C程序设计技术与实例

本书集中讨论了几个重要程序，包括C与DOS的接口，C与DIOS的接口，C如何访问整个内存。书中附有大量的实例与答案，是帮助您成为[illegible]序员的教科书。

中国科学院希望高级电脑技


HOPE

高级C程序设计技术与实例

中国科学院希望高级电脑技术公司

一九九一年三月

■北京市新闻出版局
■准印证号：3316—90316
■订　　购：北京8721信箱资料部
■邮　　码：100080
■电　　话：2562329
■乘　　车：320、302、332路车至海淀黄庄下车
■办公地点：希望公司大楼一楼往里走101房间

序　言

本书有两个目标。一是帮助你学到更多C语言的知识，二是帮助你学到更多关于IBM PC及类似机种的知识。由于学习C的一个趣事是用C开发计算机的全部功能，因此两个目标是相互关联的。然而它并非C语言的唯一好处所在，而本书前半部的大多处讨论了与固定计算机无关的高级C的问题。

本书研究的一些问题是：

※ I／O问题和解决方法
※ 指针的使用
※ 位操作
※ 内存操作
※ 标准库函数

第二部分研究与IBM的连接，包括：

※ C和DOS的接口
※ C和BIOS的接口
※ IBM PC之Intel 8088处理器的内存分段机制
※ 怎样用C访问整个内存

高级C语言也引入汇编语言并指示建立C程序可以使用的汇编语言模块。简言之，本书覆盖了大多数有趣而实用的问题。

可以进一步帮助你的是，本书提供了许多例子、题目及答案。我们提供的例子都相当短小进而鼓励你录入它并实践之，因为这是学习编程技术和概念的最有效途径。我们希望你赞成这一忠告而进入高级C的多彩世界。

另外，本书还附有包含其中全部源程序的磁盘，它们均通过测试，可直接运行。该盘可到希望公司资料部拷贝。

作　者

目　　录

第一章　简　介

在你学完C的基本知识后，下一步是什么呢？你应学习什么主题和技术呢？本书就汇集若干关键课题来满足你的需要。掌握了它们，你就可以学会基本知识而步入高级程序设计的世界。

在本书第一部分，我们涉及的高层次主题包括指针的使用，函数设计，正文和二进制文件的输入输出，位操作和如何编制遇意外输入不会失控的程序。我们还指出如何充分利用库函数及如何有效地使用内存。

这些主题是各种计算机系统中所共有的，因而均围绕着标准C。然而，C的许多高层次课题探讨开发特定计算机的资源。因此，在本书第二部分，我们将涉及到计算机硬件的难题并指出C语言如何与IBM系列微机有机地结合。你将学到访问ROM例程的方法，学会使用DOS系统调用，使用I／O端口，控制显示屏幕，访问全部PC的内存，编写基本的汇编语言子程序并嵌入你的C程序。其细节问题是针对IBM的，但原理适用于任何计算机系统。

知道程序语言如何工作及你可以用它做什么是令人振奋的。但是当学完这些后，你就应探索未知领域。或许你不很适应语言的某些特征如指针或位操作符。或许你想知道某些库函数如低级I／O函数或内存定位函数。或许你遇到编程问题如面对不于善合作的用户也给出良好的交互输入输出响应。或许你想更直接地控制显示屏幕。或许你想多了解C与指定操作系统的接口及C与汇编互连。总之，要提高有关C的知识，本书可以提供极大的帮助。

1.1　选择主题

本书涉及的许多问题是与具体计算机系统无关的，而有些语言特征依赖于特定的系统。例如C与UNIX的接口和C与DOS的接口是不同的。如果我们只选择前一部分主题，那就漏掉了许多有用的东西，反之则意味着使我们局限于特定的系统。

我们的折中办法是：前半部分讨论通用性主题，后半部集中讨论C与IBM PC／XT／AT环境之结合，并使用前半部讲到的东西。（关于C与UNIX的接口请参阅Advanced UNIX—A Programmer's Guide〈by Stephen〉的后半部）

主题的不同选择使得观察C语言的角度也发生变化。前半部分讨论通用C，即对不同的环境都大致不变的C语言。我们强调兼容性，它也是C的特点之一。后半部分讨论针对IBM的C，其程序专门开发IBM PC的功能及特性。我们又强调效率及灵活性，它们也是C的特点。兼容性和特殊性这两方面都十分有用同时又都很重要。

用C来开发IBM—PC需要对PC有详细的理解。这自然使本书涉及到诸如端口的使用、内存的分段机制、处理机使用寄存器等等一般语言书不涉及的东西。所以这是一本关于C和IBM PC设计的书。

1.2　预备知识

所谓高级C语言并非是说本书只针对高级C程序员。我们是要帮助你员。自然，我们假定你已学过C，会编程序、调用函数、说明变量、

量符、使用类型和数组等等。我们假定你已熟悉或至少接触过C的基本概念。

我们将复习更复杂些的东西如指针、函数、存贮类、数组和结构，并在行文中提醒你注意一些要点。

本书还将提供一些如IBM PC内存分段或基本的汇编语言知识，做到自成系统。

1.3 本书的方式

介绍高级主题有多种方法。一种是集中讨论几个重要程序，其中涉及到许多技能和知识。这有助于展现大程序的组织方法及展现实际问题。缺点是各种概念相互缠绕导致问题复杂化，另外，这种方法不能提供多样化的问题。

这里采用的方法更具有教程的性质。我们用一小段程序解释一个概念，这样有助于理解和上机实验。此法可以使我们接触许多例子，产生各种思路和办法。短小的例子有时会降低实用性，但这不影响学习的进程。

将错误检查程序缩小是使例子保持短小的一种方法，当然，讨论错误检查概念时除外。错误检查是编制正式程序的重要组成，但却会影响主要概念的清晰度。你可以将你感兴趣的例子加上错误的检查，这是一个练习的机会。

我们同时还提出问题让你解答，你可在每章后面获得答案。问题的形式是：

问题1—1 你将在哪里找到问题的答案？

大多数章节的末尾提供练习（但无答案）。

我们建议你自己解决问题和做练习。这将有助于你检查你的理解度并会发展你的理解能力。最有效的办法大概是重新构造例题，通过观察修改后的效果也检查你的理解能力。许多人认为多动手学习程序设计胜于只看书。

1.4 所需之硬软件

你应有一个C编译器来使用计算机系统。本书大致一半的例子可在大多数系统上运行，包括运行UNIX的计算机。另一例子针对IBM PC，你需要与PC兼容的机器及相应编译器。

一、机种机型

本书与机器有关的程序是在IBM PC上编制和测试的。有些程序使用了ROM中的例程。这些程序要在使用IBM ROM或与之兼容的机器上运行。其它程序使用了DOS例程。IBM的产品使用了PC—DOS而其它许多产品用的是MS—DOS，本书例程均可在两者上通过。有些程序需要DOS2.0以上的版本，还有些使用了特殊的端口地址和内存显示区。运行这些程序的机器需要在这些方面兼容。而许多兼容机在这些方面是和IBM一致的。

为简明起见，在提到IBM PC，XT，AT和兼容机时，我们统称IBM PC。

二、关于编译器

我们使用IBM和Microsoft C编译器（两者基本相同）。许多例子几乎无需修改即[illegible]PC编译器上使用，但有些例子使用了IBM和Microsoft独有的库函数和定义。[illegible]些例子无法通过其它编译器，因为有可能用类似的东西取代之。要点[illegible]的作用，然后就可以用其它编译器再现之。

1、IBM和Microsoft编译器

有必要了解一下两者的背景而掌握它们同其它编译器的不同。

Microsoft 最初使用Lattice版本的编译器，发展到3.0版后变成Microsoft版本，同时IBM经过小的变动后又成为IBM C编译器的1.0版。Microsoft还开发了MASM汇编器，成为IBM PC的标准汇编器。我们将在后两章里使用它。

使用这两种编译器的原因在于，它们背后有两个工业巨人的支持而有可能成为工业标准。它们也的确是好的编译器，产生快而兼容的代码。它们功能齐全，提供几种内存模式，可生成几种处理器代码。其它好的编译器也有，但是我们不在此争论谁的最好。我们使用IBM和Microsoft C是因为它们的工业地位及良好的功能。

2、编译器的区别

所有的编译器原则上应实现基本的C语言，但实际情况不然。大多数不同之处在于库函数及对IBM PC环境的调整。不同编译器的字符串处理函数不同。其它不同在于命令及C与IBM接口的设计函数之处。比如，有些编译器没有全部访问内部BIOS和DOS例程，而且接口函数的命名和参数形式也不同。

IBM PC中的8088处理器采用内分段机制。虽然PC可以使用上至640K内存，但大多数编译器只使用一段64K内存放代码，另一段放数据。IBM和Microsoft C编译器提供几种方式克服这一弱点，实现整个内存的访问。

另外，在汇编语言嵌入到C程序的方式上，IBM和Microsoft与其它编译器也不尽相同。

这些不同点对你使用其它编译器有何影响呢？我们感到主要还是要理解基本原理，理解特定函数调用的执行情况。在此基础上再掌握其它编译器就可以运用自如了。

三、IBM和Microsoft C编译的系统需求

这些编译器都较大，需256K内存以上，两个双面磁盘驱动器，Microsoft C需要DOS 2.0版以上，IBM C需DOS 2.1版以上。

这些是最小的需求，但内存增加并使用硬盘就更快更方便。

我们在附录C中列出了基本编译规则，讨论了IBM和Microsoft C的微小差别。但两者是基本一致的，可用IBM C简称两者。

1.5 小结

程序设计是令人入迷的，有时会产生嗜好。我们希望本书能满足你的这种渴望。请勿忘做练习。

本章答案

1—1 这里。

第二章　高级C语言预习

初级C和高级C之间没有明显的界线，但是大多数程序设计人员认为指针和有效使用函数属于高级概念。鉴于本书多处涉及到指针和函数，所以就以对指针和函数的总结作为本书的开头。我们还顺带谈及数组，字符串和结构。本章为你复习、巩固及扩充这方面的知识。

2.1 指针基础

C程序员的一个重要的工具是指针。它主要用于：

构造能够修改在调用程序中的变量的函数。

在处理数组的函数中动态地获得内存。

构造链表等数据结构。

首先用几句话解释什么是指针及它如何工作。什么是指针？它表示内存地址的变量或表达式。你可以将地址视为指示一个无论其中存有何值的内存单元，——因此叫指针。

一、指针常数和地址操作符（&）

C有两种表示指针的基本方式。一种是将地址操作符&放置变量前。结果表示变量的地址。因此如果x是一变量，&x即是变量的地址。例如以下程序片断：

```
int fowls=5;
printf( "The value %d is stored at location %u/n" , fowls, &fowls) ;
```

地址以无符号整形存贮，所以用%u格式输出。这个片断的输出如下：

```
The value 5 is stored at loction 3620
```

这里，&fowls是一指针常数。我们可以将fowls的值变成4或10或其它数，但新值仍存于相同的地址3620。我们可以把fowls视为数值5的标识符，而&fowls是地址3620的标识符。

地址数值的精确含义有赖于具体计算机。当在IBM PC上使用小模式时，指针的数值代表相对数据段首址的偏移量。因此，数值3620不是绝对地址，而对存放数据的内存块首址的相对地址。我们将在第六章再讨论这些概念。然而，对具体数据感兴趣的只是计算机，不是编程人员。我们用符号表示地址来编制程序。

二、指针变量和间接值操作符（*）

第二种表示指针的方式是说明一个指针变量。这种变量可以赋予地址值。例如，如果p是适当的指针变量，我们可将fowls的地址赋给它：

```
= &fowls;/★ assign an address to a pointer variable ★/
```

概念上的区别是：&fowls 所表示的值在程序执行期间不变，而pt 则地址；&fowls是常数，pt是变量。

问题2—1　　在上述赋值之后，pt的值是多少？

C的间接值操作符（*）用来表示指针所指示的数值。因此在上述例子中，pt是fowls的地址，*pt是存于此地址中的值（最近一次出现是5）。

为讨论指针变量的定义，先详尽地考察变量的地址。现有如下说明：

```
char letter;
int boxcars;
float taxes;
```

变量letter，类型为char，占一个字节，其余两变量均各占一个以上字节。但是这些都不会导致它们的地址有任何区别。地址是指数据项首字节的位置。因此，所有地址都一样，无法根据各自的地址来确定是char型还是float型。

然而，程序中需要知道一个给定指针所指示的数据类型。比如，如果间接操作符（*）给出了指针指示的数值，那么我们必须知道它占多少字节，是何类型。因此，一个指针同时表示它是一个指针及它指向的数据类型。C说明数据的指针的例子如下：

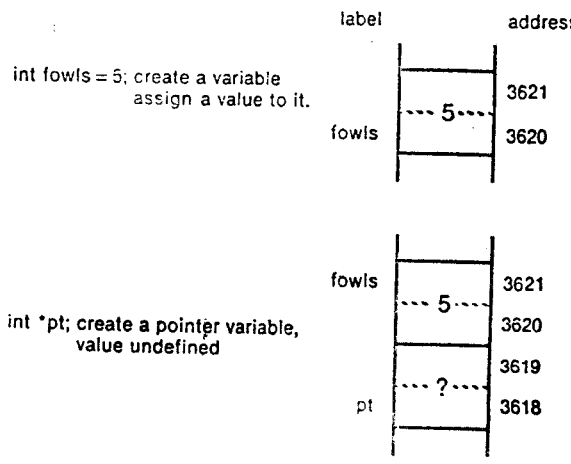
```
char ★pc;  /★ pc is a pointer to type char★/
int ★pi;   /★ pi is a pointer to type int  ★/
float ★pf;  /★ pf is a pointer type float  ★/
```

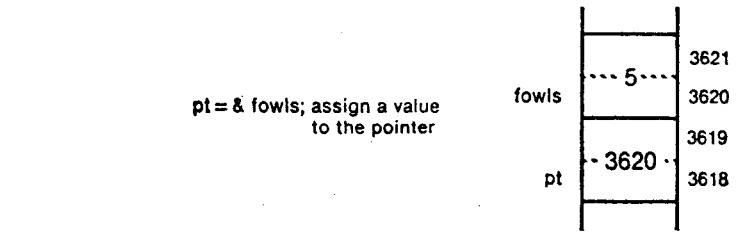
这些说明创建的变量用来存放所示数据类型的数据地址。这些说明了*操作符后就表示pc，pi和pf是指针，（反之，非指针变量则无需使用*操作符）。每个变量都获得足够内存来贮存一个地址。说明的其余部分表示所有地址处的数据类型。重要的是认识到，PC是指针，*PC不是。程序使用指针时用PC，使用指针指示的值时用*PC。

问题2—2　　在上例中怎样比较分配给pf的存贮空间与给pd的空间？

图2—1表示如何给一个指针赋值。

Figure 2-1
Assigning a Value to a Pointer-to-Int





问题2—3　　在图2—1中，&pt是什么？

再看另一例。假如有如下语句：

```
float net;
float gross=650.00;
float taxes=100.00;
float ★pf;
pf=&taxes;
```

这样，＊pf的作用与taxes相同。

```
net=gross-★pf;  /★ same as net=gross-taxes;  ★/
printf( "Taxes are %.2f\n", ★pf );
★pf=112.0       /★ same same as taxes=1120 ★/
```

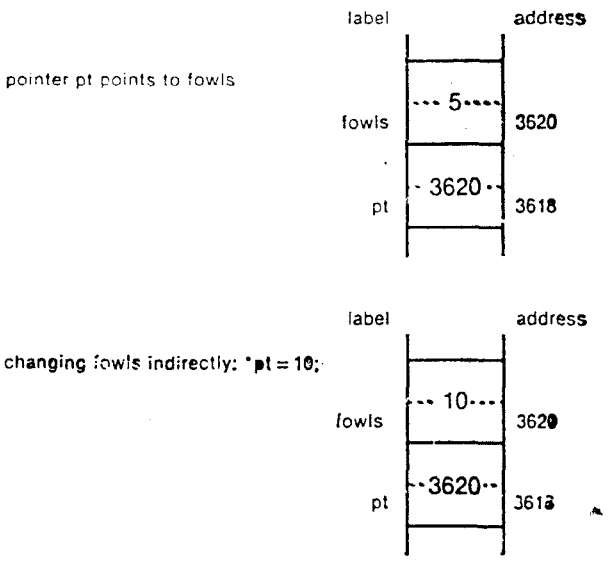
最后一句可以这样看：找到变量pf中存的地址，再将112.0存入该地址处。

同样，pf可以代替&taxex使用：

```
scanf( "%f", pf );/★ same as scanf( "%f", &taxes); ★/
```

简言之，表示地址时，pf和&taxes是等价的，表示值时，＊pf和taxes是等价的。图2—2表示用指针改变变量的值。

Figure 2-2
Changing Values Indirectly



三、指针应先赋值后使用

一个指针应先与一个特定地址相连，再使用 * 操作符。下列代码将出现问题：

```
int ★pi;
★pi=1492;      /★uh-uh ★/
```

它表示将数1492放于pi所指示的内存处。但还未给pi赋予地址，所以程序将视pi中的随机值作地址。这里就会将数1492放到或企图放到可能是很不合适的位置。所以在使用 * 操作符前应确保给指针赋一地址。

四、指针运算

你已知道怎样产生一个指针常数和指针变量。你还可以给指针进行加减操作，让我们看这样一例：

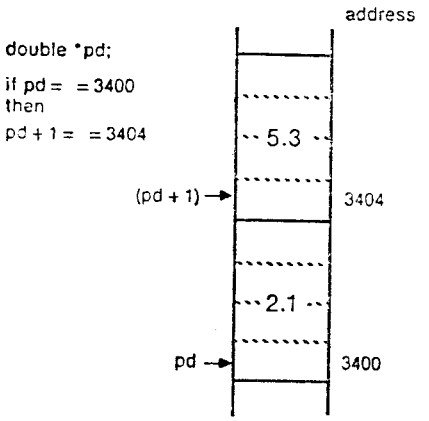
```
#include<stdio.h>
main( )
 {
char ch;
float f1;
printf( "&ch is %u and &ch+1 is %u\n", &ch, &ch+1);
printf( "&f1 is %u and &f1+1 is %u\n", &f1, &f1+1);
}
```

其运行结果如下：

```
&ch is 4048 and &ch +1 is 4049
&f1 is 4050 and &ch +1 is 4054
```

给char地址加1，其地址量增1，给float型地址加1，其地址量增4（这是对在使用4字节float的系统中）。一般地，给指针加1，指针值递增相应数据类型的一个单位量。这为数组处理提供了方便。先使指针指向数组的第一个元素，然后如图2—3，使用指针加法获取数组其余元素的指针。后面会有这样的例子。

Figure 2-3
Pointer Addition



还可以做指针减法。给指针减1则使其指向相应数据类型的前一个单元。

最后，指针运算可以得出两指针差值。这时，两指针需为同类型，差值代表数值类型的单元数。比如，P1和P2都指向4字节整数，则P1—P2是实际数值差值除以4。又比如，P1指向数组开头，P2指向数组中某一值，其差值为两元素之间的间隔数。

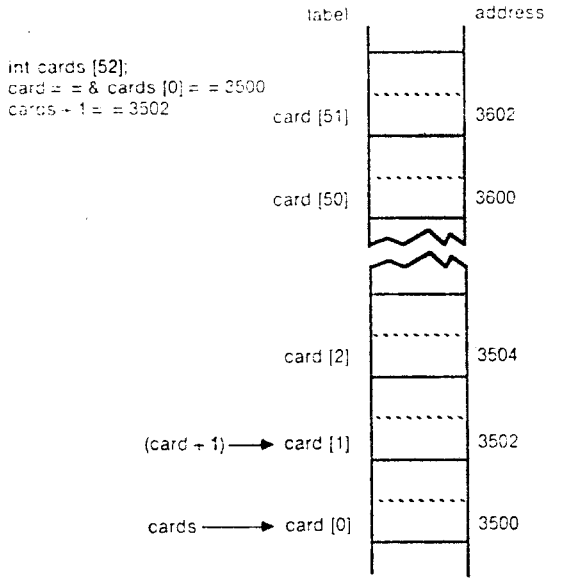
五、指针、数组名和函数名

在C中，数组名就是指向数组头元素的指针。假设有如下说明：

```
int cads [52] ;
```

这时，cards是指向cards [o] 的指针。就是说，它与&cards [o] 相同。注意，数组名是指向第一个元素而不是整个数组。这是说，cards的类型是指向int，而不是指向52个整数的数组。比如，cards + 1 指向下一个整数cards [1]，而不是下一个52个整数构成的块。见图2—4。

Figure 2-4
Array Name Is a Pointer to the First Element in an Array



假定有一二维数组：

```
int decks〔10〕〔52〕;
```

可将其视为数组的数组。这样，decks是有10个元素的数组，每个元素又是由52个整数组成的数组。由于数组名指向其第一个元素，所以decks指向第一个52整数数组，而decks+1指向下一个52整数组。

再之，因为decks是数组的数组，decks〔o〕本身是数组，这样，decks〔o〕就是数组名，也即指向自己头元素（整数）的指针，此头元素是decks〔o〕〔o〕。因此，和前面的cards一样，decks〔o〕是指向整型的指针。这就是说，decks〔o〕+1指向decks〔o〕〔o〕之后的整数，即decks〔o〕〔1〕。此概念图示于图2—5。