北京希望电脑公司计算机软件丛书

微机图像格式大全

史文革　编

- 剖析Mac Paint, IMG, PCX, GIF, TIFF等格式
- 提供单色彩色驱动程序
- 点阵，激光，Postscipt打印机驱动
- 利用面向对象程序技巧产生BMP

海洋出版社

TP391.41
SWG/1

微机图像格式大全

史文革　编
文　凯　校

海洋出版社

1996年·北京

034805

内 容 简 介

本书汇编了多种图像文件的格式、标准和 PC 上处理这些图像的原程序以及这些格式之间的相互转换的方法。阅读本书后您可掌握五种最流行的图像文件格式。本书还向您介绍了如何在各种复杂的图形模式下驱动流行的显示卡，以及如何将这些图形打印到各种常用的硬拷贝设备上，譬如打印到 PostScript 这样难以处理的打印机上，最后还介绍了图像的抖动处理技术，并提到了扩展内存、扩充内存和虚拟内存的使用方法和技术，是一本不可多得的参考资料。

JS361/08

微机图像格式大全

史文革 编
文 凯 校

* * *

海洋出版社出版（北京复兴门外大街 1 号）
海洋出版社发行　　北京施园印刷厂印刷
开本：787×1092 毫米 1/16　印张：33.3　字数：700 千字
1992 年 2 月第一版　1996 年 5 月第二次印刷
印数：1—5000 册　定价：35.00 元
ISBN7－5027－3128－8/TP・158

目 录

试读结束：需要全本请在线购买： www.ertongbook.com

第一章　图像技术入门

尽管我们的结论是矛盾的，但人类本质上仍是非常原始的动物。例如，虽然我们已经具备了处理符号和抽象文字的能力，但是我们并不想一直这样做下去。具体地说，面对不需要更多思考的感觉信息，通常比面对一段文字、几行数字或几页公式能更有效地对事物做出鉴别。

只有图像（Visual Image）可能是最易理解的信息形式，因为人类具有相当敏锐的直观辨别力。虽然我们不能极其深入细致地观察我们所看到的东西，也足以使我们完全理解它们代表的意义。大部分抽象文字只是做为视觉的一种近似表达方式。我们之所以常常描写一些事物，是因为我们手边没有摄影机。

观察图1－1，可明显地看到其中一种信息表达方式比另一种更易于完全理解。假设我们接收到的是明信片左边的图像片，想一想，它也的确更易于理解。

无论从哪一方面来讲，计算机图形学都是一种强有力的工具。但据推测，计算机图像的主要引人之处，在于它们允许人们通过计算机直觉理解事物，而不是经过深入思考再去理解。另外，图像就好象模拟物——从整体来看，它们很好地缩短了我们对大量计算机数据的理解过程。

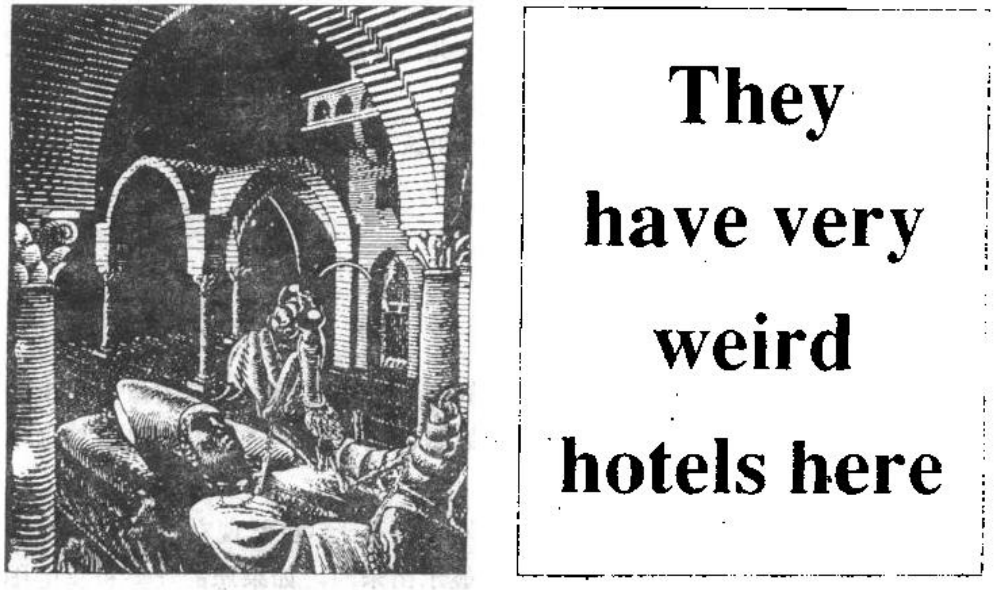

图1－1　表达同一主题的两种方法：
一种是应用图像，一种是应用文字

计算机图形学通常涉及描绘事物的某些程序，各种讨论计算机图形学的书籍常常都会谈到有关描绘直方图、分布曲线图等等的问题。因为商业方面的图形需要这些，而获得它

们又非特别困难，无非是一些程序设计问题。本书中所论及的计算机图像问题是一个多少有点儿不同的领域。

本书主要讨论位图（Bitmaps），即讨论以数字（digital）形式表示，按照像点阵列贮存（在计算机中）的图像，最终在显示器上显示或在打印机上按图形模式打印。而不去论述那些直线和圆——即这些可见目标（能够形成可绘制的位映射图像）是对我们观察事物方法的一种数字模拟（Digital analog）。

处理位图中的绝大多数问题都与特定的软件有关。例如，象MacPaint和PC Paintbrush那样的程序，按照专用格式储存位映射图像。如果您想要在自己的程序中应用这些程序产生的文件，或者先将其转换成其它应用程序要求的不同文件格式，再使用之，那么您就必须弄明白文件的结构，并且一定要有处理这些文件的程序。

如果您想观察图像文件，那么您还得了解如何使用各种流行的PC显示卡的图形模式，不过，实际中又常常缺乏这方面的文字说明资料。同样，要在打印机上打印图像，亦要使用那些更缺乏说明资料的打印图形功能。

目前已涌现出多种图形文件格式。这些格式无外乎有两种来源，一种由基于 PC 的程序产生，另一种由其他应用程序产生（譬如 Apple Macintosh 系统上的 MacPaint 应用程序）。通常后一类程序产生的文件常常被移植到 PC 环境中。本书将详细讨论五种最流行的文件格式：MacPaint、Digital Research 公司的 GEM/IMG、PC Paintbrush PCX、Compu－Serve 公司的 GIF 和 TIFF 文件格式，还有标准化的图像文件格式。

本书还将讨论如何使用各种显示卡和打印机。最后，讨论一些与图像处理有关的领域——最值得注意的是抖动技术问题（Dithering）。

在本书中，应用程序所用的程序码（code）和技术将由想要设计程序的种类来决定。

1.1 图像文件工作原理

位映射图像可以是任何一种能用显示卡的图形模式进行显示的图像。由于是初次讨论这个问题，为了简单起见，我们假定所讨论的图像文件中，储存着一幅与显示卡图形模式下的屏幕一样大小的图像。

我们再进一步假设图1－2中的图像已显示在屏幕上（虽然现在暂时无法理解，但不要紧）。事实上这幅图像是以EGA卡的画面大小显示出来的，如果您的计算机里的图形显示卡是一个CGA卡，那么您就可能看不到整个画面——此时，我们可想象显示卡硬件略为不同（即以为是EGA卡，而不是CGA卡）。

由于受本书条件的限制，这幅图像在这里只能以单色形式显示出来，但在EGA卡上，它很可能是彩色的，我们暂时忽略这种可能性。该图像看起来象是一幅老式版画（它的确是一幅老式版画），这些版画当时在Hercules卡上显示出来，全为单色的。

EGA卡的屏幕面积为640(像点)×350(像点),总共为224 000像点。显示卡上的屏幕数据如一般的程序和数据一样，贮存在缓冲区内，即8－bit bytes字节。在单色图像中，每一个位映射到屏幕上一个像点，因此就形成了术语“位映射式图形显示”。由于八个像点正好对应一个byte，所以可将这一位对应储存到大约28K的磁盘文件中。

图1－2 一幅与EGA卡屏幕同大的图形。注意所有的白色区。

这个程序将非常容易设计，因为如果我们知道EGA卡由段A000H开始存储其图像，那么我们只需将缓冲区内由这个位置开始的28 000 bytes拷贝到一个文件中即可。具体程序如下：

```
FILE *fp;
if((fp = fopen("SCREEN. BIN","wb")) != NULL){
        fwrite(MK_FP(0xa000, 0), 28000, fp);
        fclose(fp);
} else puts("Error creating file");
```

如果到目前为止，您对这一点还不十分清楚，那也不必担心。以后自然会明白的。

这个程序的执行速度非常快，但据反应它也不很经济。为了追究原因，我们不妨回过头来看看这幅图像，由图可见它的大部分是白色。甚至在大部分由独角兽占据的区域中，也有大量白色像点。象这样以直接方式将图像存储于文件内，将会在文件中留下大量冗余数据。

本书后几章研究全彩色图像，它们的“原始”形式大约要占据1/2 兆字节 甚至更多bytes的存储空间。显然，更重要的(或不久就会显得更重要的)是应把注意力放在“存储比要求存储量还要多的数据”这一点上。

独角兽图像的第一行扫描线全是白色像点，它占据80个byte，我们可轻易将其编码成两个或者甚至是一个byte。试想显示图像的程序具有一定的智能，它不只是把图像数据由磁盘文件拷贝到屏幕缓冲区内，而是在某种程度上还予以解码。

这里是一种简单的图像数据编码方案，实际上这种方案和MacPaint文件格式编码方法非常接近。我们假定该图像数据曾经显示在屏幕上，现在贮存在一个文件内，并且预先已经过编码。所讨论的程序将对这一文件进行解码，即重新把文件中的图像显示到屏幕上。当我们开始研究有关图像文件的解码部分时，我们就会发现图像文件压缩相当容易理解。

在本例中，被编码图像文件中的全部数据，不外是下列三种：

- Key bytes: 告诉解码过程如何处理遇到的下一个数据Packet。
- Index bytes: 告诉解码程序目前Packet的长度。
- 数据：是实际待解码的图像数据。

一个 Packet 是一组全部以同一方式处理的 bytes 的集合。Packet 的长度可与 Scan line 同长，然而如果通过使用一组不同类型的 Packet 对该 Scan line 进行更有效的编码，则对应这个 scan line 的 Packet 就会短得多。

本例包含两类Packet。第一类称为“run of byte”Packet，长度为三个bytes。在这类Packet中，第一个byte是0xff，告诉解码程序，正在分析的是一个“run of byte”Packet；第二个byte是一个1到80之间的数字，指出该Packet包含多少byte。虽然一个零长度Packet是无用的，对于本例中的Packet，其长度不会超过Scan line的长度。

第三个byte是实际的数据byte。如果解码程序遇到下列Packet：

```
0xff  0x09  0x55
```

它将把数值为0x55的九个相同byte写到屏幕上。值0x55在图像文件中作为屏幕数据具有一点的用意，因为它会将每一个奇数位设置为1，这样就可在屏幕的对应位置描绘出一个灰度为百分之五十的区域。

第二类的 Packet 处理不能被压缩为“run of byte”Packet 的屏幕区域，称为“字符串”Packet。在这种模型中，一个“字符串”Packet 的长度可从 3 到 82byte。第一个 byte 是 0x00，指出该 Packet 是一个“字符串”Packet。第二个 byte 包含 Packet 内连续的，可被显示到屏幕上的 byte 的数目。Packet 的剩余 byte 包含不能被压缩的数据，这些数据仍按原样被写到屏幕上。

如果解码程序遇到这个Packet：

```
0x00 0x09 0x65 0x12 0xa6 0x77 0x01 0x76 0x69 0xf1 0x98
```

那么，它将把从0x65到0x98的九个bytes“原封不动”地拷贝到屏幕上。

当处理一个Packet时，解码程序直接从文件读取下一个byte，将其看作是紧跟其后的Packet的key byte。

图1－3中的流程图说明了这种简单解码方案的工作过程。本书中决大多数实际的图像文件格式，都有一个与其相应大流程图。

1·1.1 实际考虑

上述假想的图像文件格式有许多缺陷。首先，它包含一些严格说来对所讨论的对象并非必需的数据。例如，可以将key byte和Index byte合成一个byte。因为EGA卡上一个Scan line的长度不会超过80，所以Index byte中的数值亦不会大于这个数，这样就可将key byte的最高有效位作为一个标志，区别该Packet是一“run of byte”Packet，还是一“字符串”Packet，从而构成了一种新的格式。因此，每个Packet可节省一个byte，这对一个具有许多Packet的复杂图像来说是一个不可忽视的储存空间。

我们将发现实际的图像文件格式通常比较长，因为利用上述这种不太高明的编码方案，只能挤压出一小部分byte。

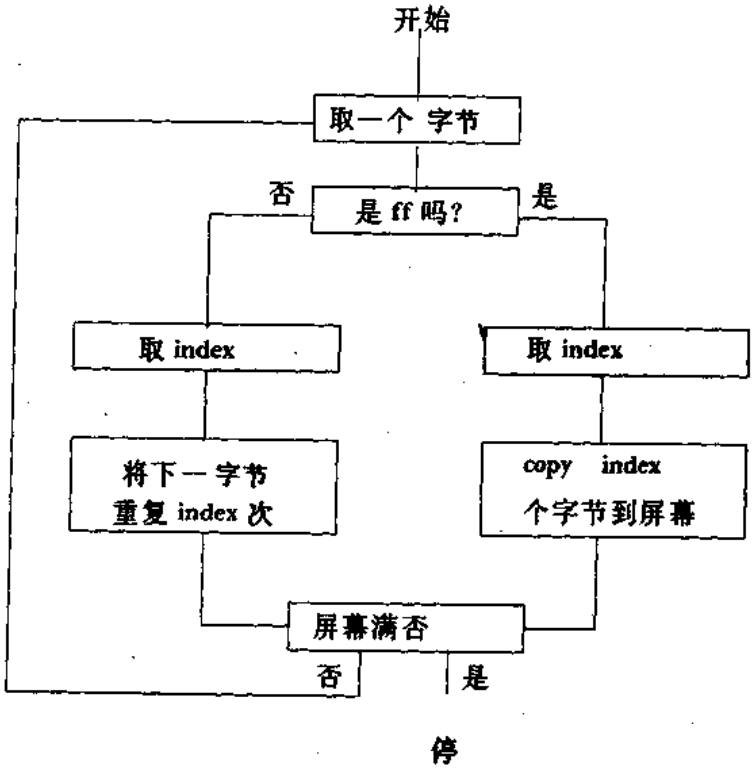

图 1-3 一种假想的图像解码过程

第二个更为实际的问题是这个图像文件格式只适用于 EGA 卡。例如，假设我们试图将它解码到 Hercules 的屏幕缓冲区内，其结果将会是比原来图像更蹩脚的抽象艺术品。Hercules 卡的屏幕大小与 EGA 卡的屏幕大小不同。它还有一个内部缓冲结构，这一点也不同于 EGA 卡，但我们暂时忽略这一问题。

试读结束：需要全本请在线购买： www.ertongbook.com

在这个假想的图像文件中，图像大小就是图像本身的大小，因为解码程序正在与EGA卡一起工作。该文件是无法移植的，即在另一台不同屏幕的计算机上执行的解码程序，不可能对这幅图像进行正确解码，因为图像文件中未包含原始图像的大小数据。

第三个潜在的问题，是这个假想的解码程序无法知道被解码的图像文件的内容究竟是什么。事实上，它可能是WordStar覆盖段（overlay），也可能是用于本书第五章的文字档，或者是patch file to cheat on Arkanoid。解码程序不能区别实际图像文件和其他类型文件，当然可能试图对潜在的错误数据进行解码。在这类情况发生时，解码程序往往会使主机锁死（crash）。

对这些问题的解决办法是在图像文件中装入一个“表头（Heeader）”。一个表头是以公认（agreed）方式构成的少许数据，包含原始图像大小以及可为解码程序识别的“标记”。这种标记可能是一独特的String of bytes。

下列信息可作为一种图像文件表头：

```
typedef struct
        char signature[5];
        int width , depth;
}HEADER;
HEADER myHeader = {"PCTR", 640, 350};
```

假如不论编码什么图像，都要先把这个表头写在文件里，那么图像文件中将包含图像数据本身、识别图像的方式及代表图像实际大小的数据。

实际中绝大多数表头都比这里的表头大，因为它们还要包含一些其他信息。图像文件格式通常会指出这样做的原因。

1.2 C语言程序的设计考虑

对于初学者，这不是一本程序设计书，不过要确实掌握书中各个C语言程序的用法，您就必须在一定程度上好好地理解C语言。

尽管在很大程度上，本书中的汇编语言函数可以原封不动地被采用，但如果您想进一步改进它们，您还得再多掌握一些汇编语言知识。

本书中C语言程序应用Borland的Turbo C交互式环境编写而成。汇编语言模块通过Microsoft的MASM宏汇编程序编译。这些C语言程序实际上比较通用，只要稍做改变，就可移植到其他几乎任一种C语言环境中。差不多任一种支持标准目标文件连接的C语言编译程序都接受汇编语言程序码。

附录A讨论了几个常用函数以及它们分别在Microsoft C和Turbo C中的某些约定。

随着ANSI标准C语言的引进，改变了人们以往编写C语言程序的方法，目前，C语言程序设计越来越受到重视。本书中的C语言程序不使用ANSI标准。

C语言最突出的方面之一，是它的灵活性。根据C语言程序设计方式的不同，编译程序将提供相应的不同程度的保护措施。例如，我们可说明“void”函数，用以阻止使用无实际意义的函数返回值。当然，象这样的ANSI标准函数本书中未出现。

这是一个常用的C语言函数：

```
show_picture(filename)
        char *filename;
{
        /* some code goes here */
}
```

这个函数有可能返回一些有实际意义的值，也有可能不返回。根据缺省值，C语言认为该函数应返回一个“int”。然而，如果您不关心返回的是什么，那么您完全可以忽略该函数的返回值。

在ANSI C环境下，我们可这样写出该函数：

```
void show_picture(filename)
        char *filename;
{
        /* some code goess here */
}
```

如果您试图使用该函数返回的值，那么该函数将引起编译程序的抗议，因为我们已使用“void”说明语句，告诉C编译程序函数的返回值是无意义的。

有那么一些人，他们认为一个善于构思的程序设计人员从来不会犯上述例子中所设定的各种错误。假设您已了解了您所写的函数，那么您就不会试图使用它们那没有实际意义的返回值。如果您与许多程序设计人员一起开发一个大型应用程序，在这项工程中，您负责调用其他设计人员所写的函数，那么上述论点就显得不实际了。

为了控制本书中各个范例程序，使其不致于太复杂、太庞大，应尽可能使用纯C语言编写，而尽量少用ANSI扩充功能。如果您准备使用这些实例，则可以分做两种情况进行处理。第一、若您选择了ANSI扩充功能，则可将ANSI扩充功能增加到这些实例中；第二、恰与前者相反，只需简单地排除编译程序的ANSI扩充特性。图1－4说明了Turbo C 2.0版的设置过程，它们将使ANSI特性不起作用，这样本书中范例程序的编译结果将不会出现ANSI警告错误或致命错误这一类信息。

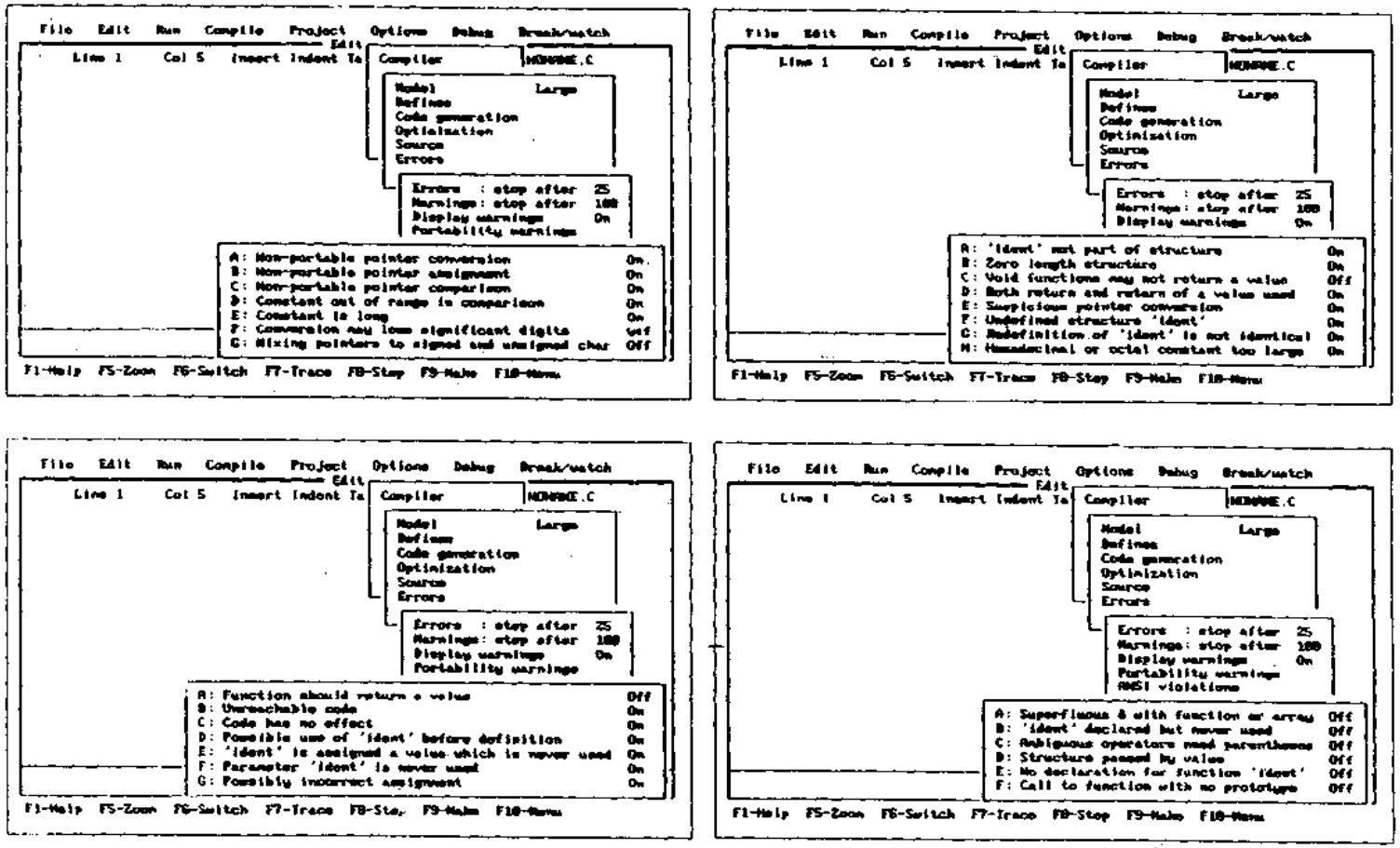

图1－4　如何为本书中的程序建立Turbo 2.0版执行环境

1.2.1 C语言程序设计风格

每一位从事大量C语言程序设计工作的人士，都会逐渐形成自己的程序设计风格。C语言最突出的特点之一是非常灵活，无论是否在一个“while”语句之后紧跟着放置一个括号（curly brackets），以及是否加入一个nested loops等等，它都会接受。因此，实际编写程序时的最基本的原则应是使其易读。本书强烈要求读者能够修改某些程序码，这样，可与您的程序设计风格保持一致。

当我们开始分析大量程序码时，头一条要求就是能够毫无困难的阅读一个程序。许多应用程序，它们的功能并不象普通的位映射式图形显示函数那样简单，因此，它们必定需要大量的程序码。对程序易读性的限制因素之一，是每次能在屏幕上分析的程序码量。

由于C语言格外灵活，从而使得程序设计人员在考虑构成各种复杂表达式时能够拥有多种选择。譬如，可将完成多个任务的程序码压缩到一行，结果是一次可在一个25行或43行的屏幕上读到更多的信息。

缺乏经验的C语言程序设计者常常被这样一些表达式所困扰。例如，应用C语言语句打开一个文件，人们或许是这样做的：

```
FILE *fp;

fp = fopen("PICTURE.MAC", "rb");
if(fp == NULL)puts("Error opening file");
else {
    /* some code goes here */        }
```

对同一表达式进行压缩后，占用的行数就会减少，譬如：

```
FILE *fp;

if(fp = fopen("PICTURE.MAC","rb"))!=NULL){
    /* some code goes here */
}else puts("Error opening file");
```

由于本书在编入有关程序时，已预先假定使用者在过去经历中已具有一定的C语言知识，所以这里大部分实例均使用新版C语言结构。再者，也许您还想用C语言编写方式修改程序。可以说前面所有实例中没有哪一个产生的最终码比另一个更可取。

1.2.2 缓冲区模型的问题

位映射图像是足够大的，即使小型位映射图像也是足够大的，至于大型位映射图像更是巨大的了，巨大的位映射图像需用Cray巨型计算机来进行处理，这种计算机对大多数程序设计人员来说或许太昂贵以至于无法获得，所以在此不予讨论。

再看图1－1，左边图像需41 722 bytes，右边图像只需33 bytes，这样，直观信息传递的潜在缺点之一应是显然的。

近年来PC的缓冲区结构最终还是受到了嘲弄，这里不必再予指责，要指出的是它只可处理长度小于64K的对象，如果长度超过这一界限情况就不同了。放在64K中的对象可以说是驻留在单一缓冲区段内。由于大部分图像文件格式都能够容纳单一缓冲区段容量的图像，所以人们必须假定所有这些图像文件格式可供大型数据对象应用。

大型数据对象是难以处理的。

C语言并没有直接考虑到分段缓冲区结构，因此，当为了处理大型图像而面临着需要各种巨大内存缓冲区时，Turbo C的实现就会出现某些漏洞。要对付这些问题可采取不同的方法，主要取决于您选择哪一种缓冲区模型。

缓冲区模型对于图形学方面的程序设计是非常重要的。

在一个简单的（小型的）C语言程序中，所有数据和程序码都将放入一个64K缓冲区段中。例如，您这样说明一些数据：

```
char buffer[64];
```

并将一个指针（pointer）指向它：

```
char *p = buffer;
```

那么“p”的实际结构将为一 16－bit数字。在微机早期，一个计算机的最大存储量只有64 Kilobytes，指针和整数由于这个缘故常常被非法交换位置。

这个问题在下列任一情况下都可得到解决，第一种情况：在内存中当缓冲区所处的大于64K的缓冲区段远离使用该缓冲区的程序所在段时；第二种情况：将缓冲区扩大，使其大于64K时。在上述任一情况下，一个16位指针将不能正确地对缓冲区进行寻址。

将程序码和数据一起放入单一缓冲区段，这种贮存分配方案称为“小型”缓冲区模型方案。它不仅有前面提到的指针问题，而且万一所有程序大于64K时还会引起另外一些问题。

在一台计算机中，“大型”缓冲区模型要使用32位指针和32位地址。理论上，这种地址机构可以处理 兆字节范围的程序码及兆字节范围的数据－－实际上在一台机器中由于系统的复杂性，只有1/2 兆字节 的缓冲区可供使用，但是人们从不介意这一点。

一个大型缓冲区指针由两个16位数字组成，分别称为“段地址”和“偏移量”。段地址是缓冲区末端与所指定位置之间16－字节的块 的数目。偏移量可在64K缓冲区中直接存

取上述这一位置。段地址和偏移量被当作是无符号整数，在必要时我们将有理由对它们做如此处理。

注意，运用几个整数代表一个大型指针的各组成部分，并不象给一个整数分配一个指针那样简单。前者是可接受的，因为它的实现意味着程序员知道自己在做什么。

如果您迫切想知道为什么段地址以 16 byte 的间隔来变化，那么请注意这一点：65 535(是可存于一个 16 位中的最大数字) 与 16 相乘，得到 1 048 560 或 1 兆字节 。这正好是 8086 系统微处理器基本缓冲区 bus 可寻址的 (addressable) 范围。

如果 C 语言可做“线性寻址”(Linear addressing)，即如果缓冲区直接由长整数来定址，那么本书中得程序处理起来将是十分方便的。其实，巨型缓冲区模型可以运用“指针归一化”(Pointer normalization) 实现线性定址方式和 PC 内部段及偏移地址方式之间的相互转换。然而，如果一个程序每次对缓冲区寻址时都必须经过这种归一化转换过程，就会大大降低系统运行速度。

实际上，使用大型缓冲区定址方法速度更快，虽然这种方法运用起来可能不方便，但是，当那些为数不多的大型缓冲区和地址出现时，可通过人工干预来处理。

当涉及大型缓冲区时，您也许不完全明白为什么大型缓冲区模型会造成如此多的问题。下面这个例子或许有助于您理解这一点。在这个例子中，“p”指向一个特别大的缓冲区基址 (base)：

```
p + = 0xffff;
+ + p;
```

在此运算之后，您也许会认为“p”会指向缓冲区中第0x10000L byte，但事实上并非如此，而是再次指向缓冲区的起点，甚至指向缓冲区起点之前的位置。

在Turbo C语言中，指针操作只改变大型缓冲区模型指针的偏移，因为我们只能对原指针值增加或减去一个整型数，而不能增加或减去一个长整数。这就意味着C语言不允许人们通过使用惯有指针演算法去存取大于64K的邻接缓冲区。

这是一个问题，因为位映射式图形显示程序一定得这样做。

该问题可由使用本书，不时出现的一个小型函数得到解决，这个函数可对指针增加一个长整数：

```
char *farPtr(p, l)    /* return a far pointer p + l */
    char *p;
    long l;
{
    unsigned int seg, off;

    seg = FP_SEG(p);
```


试读结束：需要全本请在线购买： www.ertongbook.com