C 语 言
图形用户界面编程指南

计算机程序设计语言系列丛书

候 阳
迪 克 编著

学苑出版社

计算机程序设计语言系列丛书

C语言图形用户界面编程指南

侯阳迪克 编著
希望 审校

学苑出版社
1993.

(京)新登字 151 号

内 容 提 要

本书是关于创建和管理可移植、独立的图形用户界面(GUI)的 Turbo C 编程指南。图形用户界面的各个成员,如窗口、菜单、控制、位图、资源、屏幕字模、鼠标接口等等,都用 Turbo C 写成。书中给出的 GUI 库函数完整源代码,既是编写实用的图形应用程序的基础,同时又是创建自己风格的图形用户界面的较好样板。

欲购本书者,请与北京 8721 信箱联系,邮政编码 100080,电话 2562329。

计算机程序设计语言系列丛书

C 语言图形用户界面编程指南

编　　著:侯 阳 迪 克
审　　校:希 望
责任编辑:徐建军
出版发行:学苑出版社　　邮政编码:100032
社　　址:北京市西城区成方街 33 号
印　　刷:北京四季青双青印刷厂
开　　本:787×1092　1/16
印　　张:27　　字 数:627 千字
印　　数:1—5000 册
版　　次:1993 年 11 月北京第 1 版第 1 次
ISBN 7—5077—0807—1/TP・18
本册定价:25.00 元

学苑版图书印、装错误可随时退换

目　录

简　介

大多数应用程序的用户界面大约占全部工作量的四分之三．如果不是这样，那只是因为该应用程序不需要复杂的用户界面．功能很强但难以使用或容易使人迷惑的软件实际上很难做什么事情．

在 PC 环境中编写应用程序的一个重要方面是 PC 不预先假设很多．它不包括可以为你的应用程序调用的内部用户界面，也正因为如此，你可以非常自由地构造你认为合适的软件．当然，有人会争辩说这是一个省略而不是一个功能；确实，它给你实现你自己思想的自由．但是也要求你为每一个应用程序编写用户界面．

图形用户界面（Graphical User Interface，简称 GUI）——菜单，鼠标键和对话盒——是要发明的重要努力．大的程序，象 Microsoft Windows，需要一组程序员来编写它们．但是遗憾的是，大的用户界面通常预先假设要在其下运行的应用程序．为 Windows 编写应用程序的程序员频繁地调用图形用户界面设计人员的成果，就象编写 Windows 本身一样．

本书将为你提供一个可管理范围内的图形用户界面的代码．你将能够理解其功能到这样的程度，即当需要时可以很容易地修改而为你使用．当然你绝对不会有困难去编写使用它们的程序．使用本书中的界面，你应该能够编写中型到大型应用程序．除非实在是要修改该用户界面，否则不用考虑界面，只是简单地调用它们．

Macintosh 的贡献

在这本书中，有许多源于对 Macintosh 的参考，而且有些思想就是从 Macintosh 借用的．因此，Macintosh 的发明者就会想为什么这些想编写类似 Macintosh 的应用程序的人为什么不简单地买一台 Macintosh 呢？这里我只能说，Macintosh 和 PC 各自在某些方面做得很好，各自都有大量的爱好者．

Mac 基于一些非常好的思想，而且是缺乏经验的人们使用计算机的用户界面的概念．谁也不能忽略 Mac 的发明，而不知羞耻地说这完全是他自己的努力而没有借用任何人的思想．

另外，人们总是讨论他们使用的软件的结果，而不是该软件在其上运行的计算机．如果你有一个可以实现的计划去保存热带雨林、减除全球变热、使一加仑普通汽油可以跑 600 公里或者从我们生活中去除商业电视，人们要问的问题不会是"你是在 Mac 上还是在 PC 上做的这项工作？"．

在学习 Macintosh 的基本概念和大量模拟其细节之间有着非常明显的区别．编写跟 Macintosh 毫无区别的应用程序是不道德的，而且也是很愚蠢的．当用本书中的图形用户界面开发应用程序时，你会很自然地会有把它引用到你的程序设计项目中去的想法，而 Mac 应用程序的作者可能没有这些想法．

另外，本书中与 Mac 有关的部分大部分集中于讨论程序设计概念，而不是集中在软

件的外观或运行上.

绘图程序

一个人不可能孤立地开发一个用户界面——如果这样做的话，你至少不能找出大部分错误．除了编写在本书中讨论的代码，我还编写了 Desktop Paint（在该用户界面下运行的应用程序），而且在我的朋友之间共享这些软件，他们帮助我找出了许多错误而且做了许多改进．他们除了帮助找出了最初版本中的几个剩余错误，而且还帮助使其更加直观和用户友好．

Desktop Paint 本身频繁地在全书中出现，因为它可以用于那些你可能使用图形用户界面库的一些东西的较好例子，即使你从来没有接触过绘图程序．另外，本书中的所有演示都是用 Desktop Paint 创建的．

第一章　基本 GUI 概念

图形用户界面并不是新事物，你可以在法国南部洞穴的壁上、埃及金字塔内和大多数有原始绘图的国家找到它们．远比文字更加通用，图形甚至可以被无法理解等同文字的人所理解．

图形化通讯可以很典型地从还没有更好手段通讯的原始人类、需要透过语言界限通讯的人们以及认为其大部分用户不愿意读手册的计算机公司中找到．

从最简单的意义来讲，一种书写语音语言——如英语——是一种代码．下面的记号表示声音，而不表示意义，而且单独的各个声音是没有意义的．只有那些知道关键（即说这种语言的人）才可以阅读这种文字和理解它们．

这也适用于英语的子集，比如被用于计算机术语的子集．不知道串行口（serial port）是什么东西，即使英语说得很流利的人也会感到这很容易忘记．

如果你不懂这种代码，这种情况就变得更加容易充分理解．图 1.1 是可读性很强的语音通讯的一个例子．

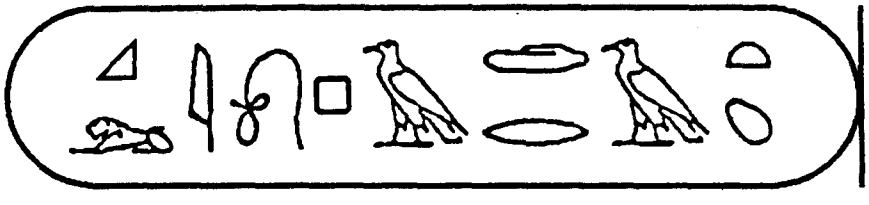

图 1.1：一个较老图形用户界面

另外，从历史角度来看，象形文字是最难翻译的，因为它们有时候只是语音——当然，它们看上去是图形．名字被语音地书写，而任何其他东西则都被图形地表示．另外你还得注意，图 1.2 中的翻译只是大概的，因为语音象形文字表示与我们的不同的一套语音．（例如，埃及的象形文字没有元音．）

图形看上去要远比语音更加一般化，至少它们可以被正确使用．但是，在处理与计算机有关的概念时，软件设计人员通常遇到这样两个问题：许多与计算机有关的思想不能直观地表示；如果可以，但是有许多图形早就已被苹果计算机公司拥有版权．

用图形通讯还有一个潜在的缺限．你可能认为图形的解释通常在某种程度上是有文化偏向的．例如，对于我们来讲，Macintosh 屏幕上的小垃圾箱表示废弃的东西．但是对于生活在三千年前的埃及人来说，它可能表示保存东西的一个罐——正好相反．

图形用户界面是与计算机操作有关的基于信息的现象的直观表示的实时应用．图形用户界面通常使有些程序员感到气恼，因为他们觉得所有这些图形占用大量内存和处理器时间，而实际上根本没有做什么事情，只是让有些人可以不击键地运行计算机．但是，对于地球上的其他人来讲，图形用户界面使得计算机看上去更象某些熟悉的东西——如桌面，或垃圾箱．

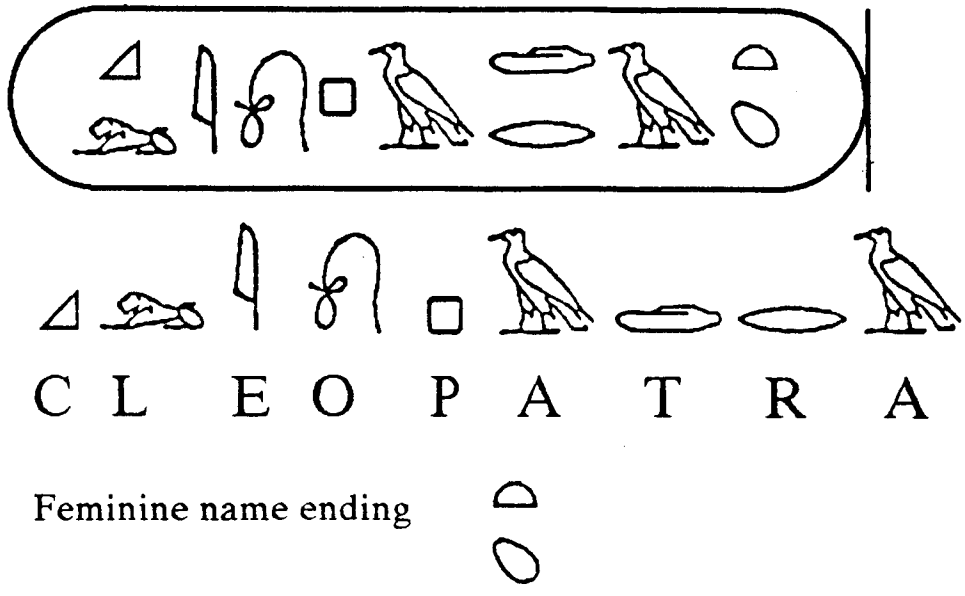

图 1.2：带有文字替代的较老图形用户界面

但是这里微型计算机的悲惨事实是大多数使用它们的人们确实相信计算机包含桌面和垃圾箱．虽然许多使用计算机的人被迫键入 DOS 命令，但是很少有人实际理解计算机正在做什么．

但是不管怎么说，大多数人还是要求一个友好的图形用户界面．在 Macintosh 环境中工作的程序员不会有什么困难，因为该界面已被硬件固化，但是也无法避免它．但是 PC 的情况就完全不同了：它没有固有的高级图形．当从盒子里拿出来的时候，PC 是不知道如何画菜单、开窗口或驱动鼠标的．

在 PC 上编写带有图形用户界面的程序时，会遇到许多潜在的令人迷惑的问题．有些第一代 PC 仍然存在，它们相当相当慢．大多数较新的机器至少有一些扩展或扩充内存，但是你不能假设它们全部都有．有些机器有非常标准化的 EGA 或 VGA 显示卡，但是许多仍然使用较老的单色显示卡——神话般的 Hercules 图形卡或其他杂牌．大多数用户有带有硬盘的机器，但是其他——尤其是那些低级膝上型机器的主人——仍然只使用软驱．最后，8086 系列的处理器做很多事情是非常快的，但是遗憾的是图形操作不在其中．

虽然 PC 没有内嵌的图形用户界面，但是有大量的第三方图形用户界面软件包．其中最常遇见的毫无疑问是 Windows 3，但提供许多诸如多任务、外部设备管理等等功能．但是，Windows 是巨大的；而且你如果编写一个在 Windows 下运行的程序，则必须预先假设将要使用它的每一个人都已经装了 Windows．图 1.3 显示 Windows 3．

Windows 3 在使用 80286 的机器上实际上是毫无用处的，而且从实用角度来讲，至少需要 2MB 内存．即使使用当前的 Windows 软件开发工具包，Windows 应用程序开发仍然是一个漫长而且乏味的工作．

另外还有一些不常使用的类似 Windows 的软件包，每一个功能都要差一些，因此用得也就少一些．其中有时使用的一个是 Digital Research 公司的 GEM 软件包．除了作为单独图形用户界面和 GEM 自己的应用程序一起交付，它的一个修改后的版本构成流行的

Ventura 桌面印刷系统的基础。图 1.4 是来自一个基于 GEM 的应用程序的屏幕。

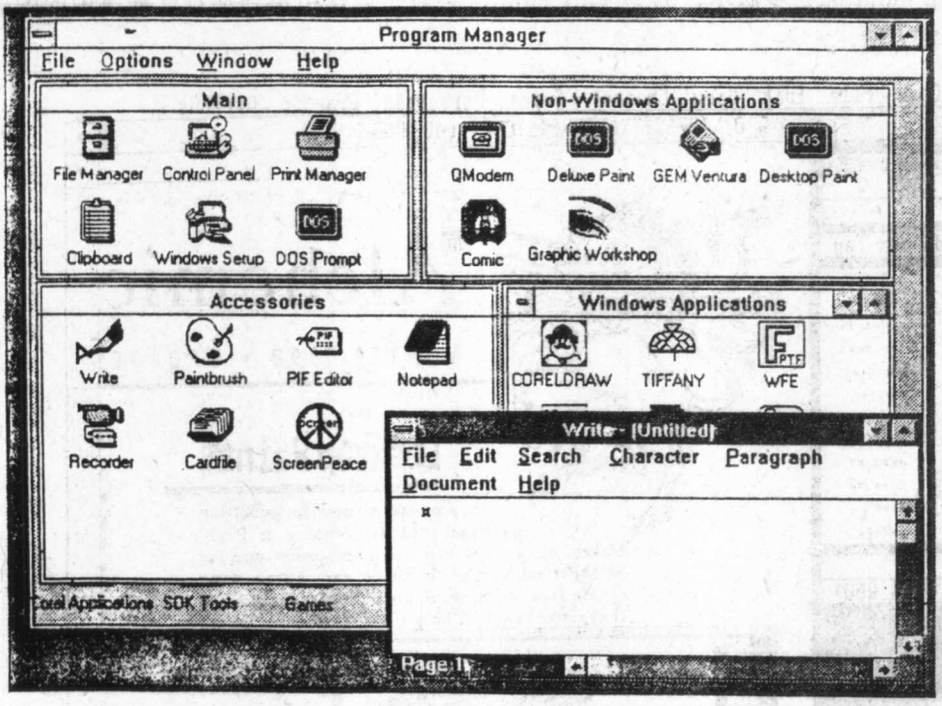

图 1.3: Microsoft 的 Windows 3 环境

跟 Windows 一样，在 GEM 下运行的应用程序也都得预先假设它们的所有用户都拥有 GEM。GEM 应用程序的软件开发可能要比 Windows 程序的限制少一些——你可以使用几种语言实现中的一种，而不象 Windows 你只能使用 Microsoft C 编译程序——但是每次当你要运行你正在工作的程序时，你必须先启动 GEM。

这两个大的工作环境的下一个问题是可连接的图形用户界面库，而且确实只有很少几个。有一些简单函数库，它们提供创建和管理鼠标驱动的用户所需的所有工具。它们与跟随编译程序而来的库一样。你可以连接函数来画和管理菜单，开窗口，等等。这些例子包括 Zinc、TEGL 和来自 Island Systems 的软件包。

使用一个界面库而不是象 Windows 那样完整的图形环境的优点是最终的应用程序可以独立于第三方代码而运行——用户不必为运行该应用程序而去买任何额外的软件包。其次，开发这样的应用程序更加容易，你通常可以在一个集成开发环境——如 Turbo C——中工作。最后，用专用图形用户界面库编写的程序通常需要很少的硬件资源。当然，这样的程序没有 Windows 的多任务和其他功能。

因此最好的方法是创建一个你自己的图形用户界面库，这也正是本书所描述的。因为虽然商用用户界面库使用很方便而且通常功能很强大，但是它们很可能产生一些非常巨大的程序。而且它们限制你按库的设计者所设计的方式运行，而这有时候可能与你所有的方式并不相同。

图 1.5 演示一些来自本书的图形用户界面的屏幕。它们来自一个称作 Desktop Paint

的工作应用程序，它就是用本书中的图形用户界面开发的．你可以在第十一章中找到 Desktop Paint 的初步版本，名为 Tiny Paint．它对于创建引用到你自己程序中的图标和位图非常方便．

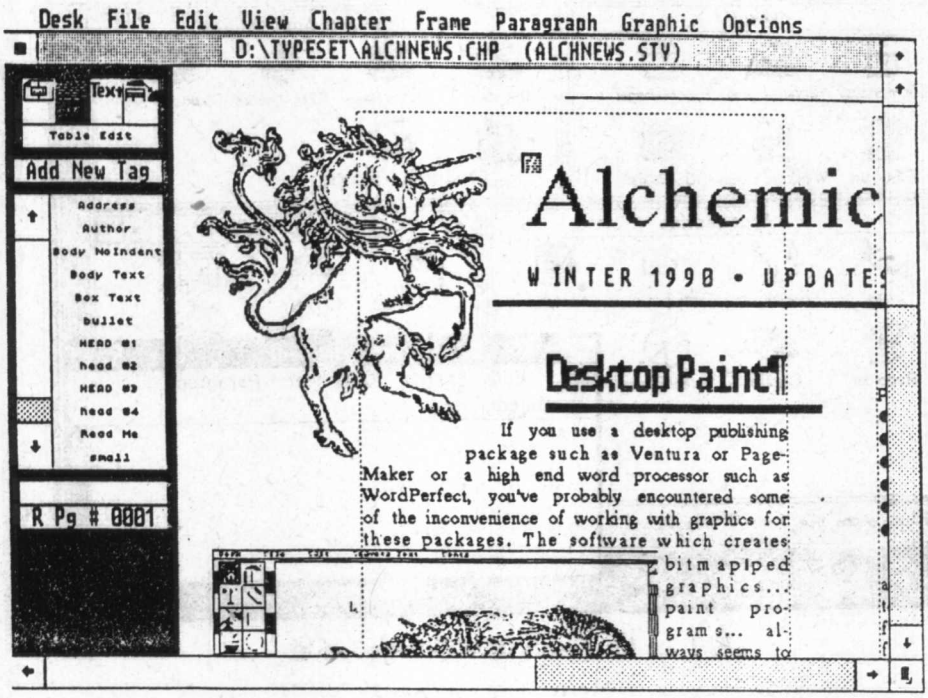

图 1.4: 一个 GEM 应用程序，Ventura Publisher

如果你自己编写一个用户界面，你可以使它复杂到任何程度，而不必增加任何你的应用程序不需要的成分．

1.1 硬件和软件要求

本书将讨论在 PC 环境中编写你自己的图形用户界面的细节．本书中的代码全部都是用 Borland 公司的 Turbo C 2.0 写的．它们也可以在 Turbo C++下运行，只有稍微作一些修改（见各有关部分的注释）．在其他 C 编译程序下运行则要作一些稍大的修改．原因在后面代码的讨论中会更清楚一些，因为一个实用的图形用户界面要求与所使用的语言的内部有非常深入的接触．书中的汇编程序可以用 Microsoft 的 MASM 或 Borland 的 TASM 处理．

本书中的代码没有什么真正的技巧，但是在你试图理解它之前应熟悉 C 程序设计的基本知识．

本书中的所有程序都应被用大模式（large model）编译．在编译之前应关闭 Turbo C 的大小写敏感连接选择项（case-sensitive link option）和 ANSI C 的扩展（ANSI C extention）．

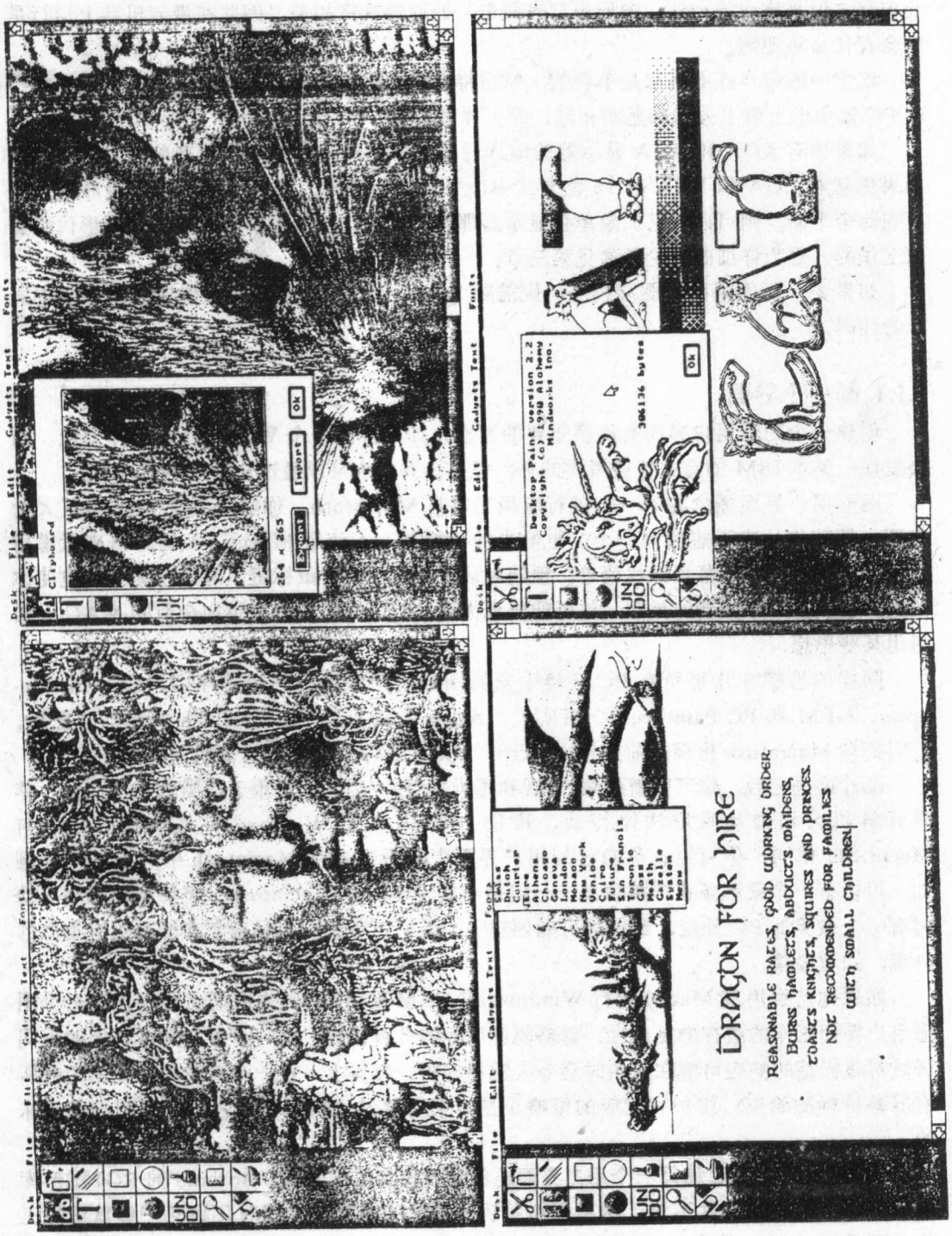

图 1.5: 图形用户界面的例子

你可以在几乎任何类型的 PC 上运行，即使是非常慢的基于 8088 的机器。不需要扩

充内存．但是你应该记住，图形用户界面是与处理器紧密相关，因此在慢的机器上运行是不会有什么乐趣的．

本书中的用户界面将都是单色的，原因将在后面讨论．由于这个原因，它在任何流行的 PC 显示卡上看上去将都基本相同，除了屏幕大小．

如果你有 EGA 和 VGA 显示器的编程经验，你可能知道这些卡没有单色方式．但是如果你忽略 EGA 和 VGA 的 16 色方式中位平面切换逻辑，高效率地把图形缓冲区同时写到四个平面，则可以使它们象单色显示器那样工作．这就是用户界面的用户图形代码如何工作的．更为详细的讨论请参见第三章．

如果要一起使用鼠标驱动程序，则需要 EGA、VGA 或 Hercules 卡以使用本书中所示的代码．

1.1.1 画一个界面

创建一个图形用户界面有许多重要的考虑．如果你买一个 Windows 软件开发包，则会发现一整本 IBM 的有关这些考虑的书；但是该界面也可以通过几段而得到．

图形用户界面概念的第一个流行应用当然是 Macintosh．你实际上从 Mac 学了大量东西，即使你从来不想拥有一个．更恰当的可能是，人使用 Macintosh 的术语来考虑用户界面——它们在屏幕顶部找菜单，预期对话有小盒子来按鼠标键，等等．为了不迫使你的用户去熟悉一个全新的界面，你可能要为你自己的用户界面而从更加全面的 MAC 元素借用某些思想．

同样的思想在几乎所有 PC 环境中支持图形用户界面的应用程序中出现，如 Windows、GEM 和 PC Paintbrush．事实上，Apple 并没有发明 Macintosh 所基于的思想；它们是在 Macintosh 出现的前几年由 Xerox 为其自己的 Star 微型计算机而开发的．

说过这些之后，你可能就理解"外观和感觉"问题，因为它适用于这类软件．例如，苹果计算机公司鼓动采取法律行动，声称 Microsoft 仅 Windows 在外观和感觉上与 Macintosh 相同．你可以在你自己的用户界面中使用嵌入在 Macintosh 中的思想——例如，可以有一个菜单条在屏幕顶部——但是不要完全照抄 Macintosh．坚持要使程序启动时有一个微笑的 PC 外观，最好尽可能创建一个最初的图标，而且避免在屏幕上有类似的外观，如垃圾箱．

如果你已经用过 Mac 或运行 Windows 或 GEM 的 PC，则你可能已经知道使一个图形用户界面工作的潜在的复杂性．忽略驱动鼠标的低级机制，在一个复杂的屏幕上的不同区域对鼠标键的响应可能在开始时会令人觉得挫折．事实上，在一个图形用户界面中对所有屏幕目标的绘制、维护和响应的策略正是本书所要讨论的．相比之下，图形则是乏味的．

除此以外，为 PC 编写一个用户界面还包括一些实用的问题．Borland 的 BGI 图形库确实不错，但是它速度不快，达不到你所认为需要的速度．它同时还缺乏一些重要的工具，而且 Turbo C 的内存管理机制是非常初级的．你最好从头编写一个完整的图形包，就象构成 Macintosh 基础的 QuickDraw 包一样；当然做这些工作就超出本书范围了．

为了得到与使一个图形用户界面在 PC 上工作有关的问题的感觉，让我们从一个包含几个概念的例子开始．图 1.6 演示当一个程序要打开一个文件时该界面使用的对话盒．

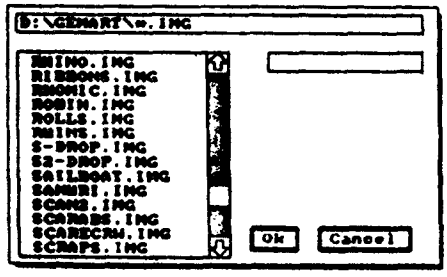

图 1.6: Choose File 对话盒

对于这时的程序，有关的部分是:

```
chooseFile(path, name, 48, 48, drivemap);
```

参量 path 是持有对于可能包含要打开的文件的目录的路径的串。参量 name 将包含所选择的文件名，如果 chooseFile 函数返回一个真值。下两个参量则是表示被该调用所调用的对话的左上角。最后一个参量 drivemap 是一个包含合法驱动器符的串。

为了创建这样一个对话，用户界面必须做一些事情，包括在屏幕上打开一个窗口，画窗口边框，用空白填充屏幕内部，并画阴影。而且还需要保存被该窗口覆盖的区域，以便在窗口关闭时恢复该区域。

窗口中的每一样东西都是一个对象。在一个理想的用户界面中，可能会规定只有对象才能被放在窗口中——但是并不总是能够实际这样做的。对象并不是被画到而是被增加到窗口中的。

这个窗口中的对象是顶部的文本字段对象（包括文件路径）。它下面和右面的编辑字段对象（包括所选择文件的名字），左边的清单对象（包括可用的名字），清单右边的滚动条对象，以及该窗口右下角的两个按钮对象（分别包括文本"Ok"和"Cancel"）。

保持对这个窗口中所有可以用鼠标键选择的东西的管理，而不是一团迷糊，则要求它们被作为对象来处理，而不是图形，与象 Corel Draw 或 AutoCAD 那样的绘图程序操作对象绘制的方法相同。对象的实际性质将在后面讨论。

为了打开一个窗口，你要创建一个类型为 WINDOW 的对象，并告诉用户界面把它增加到屏幕上。随后，创建一个类型为 TEXTFIELD 的对象并把它增加到该窗口中。另外还有类型分别为 EDITFIELD、LIST、SCROLLBAR 和 BUTTON 的对象。

TEXTFIELD 对象和 EDITFIELD 对象之间的区别是后者用户编辑盒中的文本。

把一个对象增加到窗口中的过程包括把它画在屏幕上，并告诉窗口如何在有鼠标键被按下时找到它，同时也就出现了与其被按下鼠标的位置有关的一些问题。

组成一个 WINDOW 对象的东西中有一个是对象指针。当该窗口被最初打开时，它被赋值为 NULL（即它不指向任何对象）。如果你随后把一个 TEXTFIELD 对象增加到该 WINDOW 对象中，则该 WINDOW 对象中的对象指针将指向 TEXTFIELD。而 TEXTFIELD 对象中的对象指针则指向 NULL。

如果你随后把一个 BUTTON 对象增加到 WINDOW 对象中，则 WINDOW 将指向

TEXTFIELD，TEXTFIELD 指向 BUTTON，而 BUTTON 指向 NULL．增加另一个对象到 WINDOW 则将使 BUTTON 指向它，以此类推．

这是窗口管理中的一个基本思想，被称作链表（LINKED LIST）．如果你知道一个窗口在什么地方，则可以通过顺序查找链表直到找到所要的对象来定位任何对象．

除了一个对象指针，每一个对象还有一个类型，比如 WINDOW，BUTTON，等等．通过顺序查找链表而找到一个对象，你就可以读取其类型．

最后，每一个对象被其在屏幕上的位置（被称作边框）而定义．通过定义，就不会出现同一个窗口中两个对象重叠的情况．这样，如果鼠标键被在该窗口内按下，你则可以找到鼠标键被按下的对象，找出该对象的类型，以及该类型的对象中哪一个是实际所选择的．

被在其内按下鼠标键和标识之后，一个对象首先必须被跟踪，然后被响应．跟踪一个对象包括做任何使一个对象看上去确实被激活所需要的工作，这样你的程序的用户知道发生了一些事情．BUTTON 对象被通过颠倒颜色直到释放鼠标键来跟踪．EDITFIELD 对象被通过允许用户键入新的文本或修改原有文本来跟踪．TEXTFIELD 对象则被不作任何事情地跟踪（因为对于 TEXTFIELD 你只能观察）．

最后，当一个特定对象被激活时，你的程序必须做所有有关的事情．

1.2 对象的性质

图形用户界面使用比任何其他类型程序多得多的特殊化的 struct 定义．所有对象都被作为结构处理，即 typedef 结构．

使用界面工作需要包括两个最简单的结构：

```
typedef struct {
    int x, y;
    } POINT;

typedef struct {
    int left, top, right, bottom;
    } RECT;
```

POINT 对象定义屏幕上的一个点，其中点（0，0）是屏幕的左上角．通常把一个 POINT 对象和鼠标的位置联系起来．你将在下一章中看到，如果 p 是一个类型为 POINT 的对象，则这将返回鼠标在其中的位置：

```
MouseLoc(&p);
```

函数 MouseLoc 是用户界面库的成员．

RECT 结构定义屏幕上的一个矩形区域．例如，对象的边框被定义为一个 RECT．

下面是一个 WINDOW 对象．注意在它里面的 RECT 对象定义其边框．另外，back 是指向定义在画窗口之前窗口后面图象的缓冲区的指针．

```
typedef struct {
    OBJECTHEAD head;
    RECT frame;
    char *back;
    } WINDOW;
```

窗口的 OBJECTHEAD 元素显然也是一个结构:

```
typedef struct {
    unsigned int type;
    void *next;
    } OBJECTHEAD;
```

事实上这是前面提到的对象指针。每一个与窗口有关的对象类型都从一个 OBJECTHEAD 元素开始。如果 p 指向一个未知类型的对象，则可以这样来找出该对象到底是什么:

```
n = (OBJECTHEAD *)p->type;
```

可能有一些预定义的常量出现在 OBJECTHEAD 的 type 元素中。

OBJECTHEAD 的 next 元素指向链表中的下一个对象，或者当它是该链表中的最后一个对象时指向 NULL。

1.2.1 内存管理

在创建窗口中最大的考虑之一是当窗口不再需要时如何关闭它们。为了创造窗口从屏幕上突然消失的印象，原来在被占区域的所有细节都必须被重画。你可以遵循下面两种途径之一。

关闭窗口的最简单的方法，而且也是本书所采用的方法，包括在画窗口之前把窗口将要占据的屏幕区域的图象拷贝到内存中，然后当关闭窗口时重新取代它。这在 Turbo C 下用 getimage 和 putimage 函数做十分容易。WINDOW 对象中的 back 指针指向保存被窗口占据区域图象的缓冲区。

这种方法有几个缺陷，主要是内存管理。每次你用这种方法打开一个窗口时，你必须申请一个相当的内存块来适合背景图象。如果你一个叠着一个地同时打开多个窗口，则每一个都需要一个背景缓冲区。

如果你的程序是在 16 色显示器上运行，比如 EGA 和 VGA 卡，你可能很容易就碰到要打开的窗口太大的问题。Turbo C 的 getimage 函数只能处理小于一个段的图象块，即它们必须少于 64K。这样，你只能打开只有整个 EGA 屏幕一半大小的窗口。

putimage 函数在 16 色方式中十分慢，而它负责使窗口突然消失。有人可能会说你实际上在任何情况下都不应创建这样大的窗口。但是，每三章将演示如何这样做。

关闭窗口的另一种方法是重新产生该窗口下面的所有屏幕对象（巧的是这正是 Mac 如何处理关闭窗口的）。因为在 Mac 上可以不在正在运行的主程序的控制之下弹出一个窗口，Mac 使用一个系统信号（或事件）来允许一个正要关闭的窗口发信号给所有在它之