通过在命令行运行 ALGOL 编译器对 FIRST.ALG 文件进行编译，其格式如下：

```
ALGOL FIRST.ALG
```

ALGOL 编译器对这条命令很可能做出这种响应，在显示设备上给出以下提示信息：

```
Line 3: Unrecognized keyword 'ende'.
```

ALGOL 编译器对拼写的检查非常严格，它在这一点上比传统的语文教师更甚。因为输入程序时，误把“end”拼写做“ende”，所以编译器通过提示信息告诉我们程序中有语法错误（syntax error）。当编译器检查到“ende”时，它期待能遇到一个可识别的关键字（keyword），但由于上述错误，编译不能通过。

将程序中的错误改正之后，可以再次执行编译命令。由于系统平台和编译器版本的不同，有时编译器会直接生成一个可执行文件（CP/M 平台下此文件名为 FIRST.COM，MS-DOS 平台下名为 FIRST.EXE）；有时还需要再执行一个步骤才可以完成。不论是哪种情况，最后你都可以在命令行执行 FIRST 程序：

```
FIRST
```

FIRST 程序会对此响应，并显示以下内容：

```
This is my fist ALGOL program!
```

注意，这里还有一个拼写错误：first 被误做 fist！编译器没有检查出这个错误，因此它被称为运行时错误（run-time error）——程序被执行时才出现的错误。

很明显，我们的第一个 ALGOL 程序中，print 语句的功能是把一些信息显示到屏幕上，在本程序中是显示一行文本（从功能的角度来看，该程序与本章开始所给出的汇编程序是等价的）。ALGOL 语言的正式规范中并不包括 print 语句，但我们假设所使用的特定 ALGOL 编译器包括这个便利的工具，它有时候也被称做内部函数（built-in function）。除了 begin 和 end 之外的大部分 ALGOL 语句都要以分号结尾。你可能注意到了 print 语句使用了向右缩进的格式，这并不是必要的，其作用只是为了让程序的结构更加清晰。

假设现在要编写一个用于两个数相乘的程序。每一种程序设计语言都包括变量（variable）的概念。程序中的变量可以是一个字母、一个短的字母序列，也可以是一个单词，由程序员自己决定。变量名实际上对应内存的一个存储单元，但在程序中是通过名

字来访问该存储单元的，而不是直接使用存储单元的地址值。下面的程序定义了三个变量，分别命名为 *a*，*b*，*c*:

```
begin
    real a, b, c;
    a := 535.43;
    b := 289.771;
    c := a × b;
    print ('The product of', a, ' and ', b, ' is ', c);
end
```

real 语句称为声明（declaration）语句，用来指明程序中要定义的变量。在该程序中，变量 *a*，*b*，*c* 被定义为实数（real）类型或浮点数类型（同时，ALGOL 语言也支持使用 integer 关键字来定义整数型变量）。程序设计语言中的变量名通常以字母开头，变量名也可以包括数字，但前提是第一个字符必须是字母。变量名不能含有空格，也不能包含除字母和数字以外的其他大部分字符。通常编译器会规定变量名的最大长度，本章用到的变量一律以单个字母命名。

假如我们使用的特定 ALGOL 编译器支持 IEEE 浮点数标准，则本程序中所定义的三个变量每一个需要 4 个字节的存储空间（采用单精度格式）或 8 个字节的存储空间（采用双精度格式）。

声明语句之后的三个语句是赋值（assignment）语句。在 ALGOL 语言中，赋值语句很容易被识别，因为它的格式很固定，总是在冒号后面跟着一个等号（在大多数计算机语言中，赋值语句通常只包括等号）。赋值语句的冒号左边是一个变量，而等号右边是一个表达式，表达式的计算结果将被赋值给左边的变量。前两条赋值语句指明，变量 *a*，*b* 将分别被赋予一个特定的值；第三条赋值语句指明，将 *a* 和 *b* 的乘积赋值给变量 *c*。

时至今日，我们所熟悉的乘法符号“×”已经不允许出现在程序设计语言中了，因为它没有被包括在 ASCII 和 EBDCIC 字符集中。大多数程序设计语言使用星号（*）来替代它作为程序中的乘号标记。尽管 ALGOL 使用了普遍使用的斜杠（/）作为除法标记，但在该语言仍然可以使用除法标记（÷），该标记用于整数除法，用来指明被除数与除数的倍数关系。ALGOL 还使用了另一个非 ASCII 字符“↑”，该箭头符号用来做乘方运算。

最后的 print 语句用来显示所有变量的值。它包含文本和变量，并以逗号分隔。print

语句的主要工作并不是用来显示 ASCII 码值的，但本程序中却做了更多的工作：将浮点数也转换成了 ASCII 码并显示：

```
The product of 535.43 and 289.771 is 155152.08653
```

接着会执行 end 语句，程序终止并将控制权交还给操作系统。

如果要将另外两个数相乘，则需要做以下工作：修改程序，改变变量的值，重新编译并重新运行程序，这将是一件非常烦琐的工作。为了避免这些重复工作，我们可以借助于另一个内部函数 read。修改后的程序如下：

```
begin
    real a, b, c;
    print ('Enter the first number: ');
    read (a);
    print ('Enter the second number: ');
    read (b);
    c := a × b;
    print ('The product of ', a, ' and ', b, ' is ', c);
end
```

read 语句的功能是读取从键盘键入的 ASCII 码值，并将其转换成浮点数。

循环（loop）是高级语言的重要组成部分。循环使得程序可以对同一个变量的不同取值反复执行相同的操作。假设我们要写一段程序用来计算 3，5，7，9 各自的平方，可以这样编写程序：

```
begin
    real a, b;

    for a := 3, 5, 7, 9 do
            begin
                b := a × a × a;
                print ('The cube of ', a, ' is ', b);
             end
end
```

for 语句将变量 *a* 的值第一次设为 3，然后执行 do 关键字后面的语句。如果 do 后面要执行的语句不止一条（如本例），则必须将它们置于 begin 和 end 之间，这两个关键字定义了一个语句块（block）。第一次循环之后，for 语句会一次为 *a* 赋值 5，7，9 并执行

相同的语句块。

下面的程序中采用了 for 语句的另一种使用方式，这段程序用来计算 3～99 之间所有奇数的立方。

```
begin
    real a, b;
    for a := 3 step 2 until 99 do
            begin
                b := a × a × a;
                print ('The cube of ', a, ' is ', b);
            end
end
```

for 语句将变量 *a* 初始化为 3，并执行 for 后面的语句块。第一次循环结束后，变量 *a* 与 step 关键字后面的增量相加，这里是 2。新得到的 *a* 的值是 5，它将用于第二次执行语句块。变量 *a* 继续增加 2 并用于下一次循环，直到 *a* 的值超过 99，这时 for 循环结束。

一般而言，程序设计语言对语法都有着非常严格的要求。在 ALGOL 60 中，就关键字 for 而言，其语法格式是：for 的后面只能跟一个变量名。而英语中的这种限制宽松的多，单词 for 的后面可以跟所有类型的单词，例如“for example”，“for can”等。尽管编译器是非常复杂的程序，但其所能解释的语言显然要比人类的语言简单得多。

大部分程序设计语言的另一个重要特征体现在条件（conditional）语句的使用。条件语句的特点是，只有当某个条件成立时才会执行另一条对应的语句。在下面的例子中，我们使用 ALGOL 的内部函数 sqrt 来计算一些数的平方根。sqrt 函数的参数不能是负数，因此要在程序中通过条件测试避免这种情况。

```
begin
    real a, b;

    print ('Enter a number: ');
    read (a);
    if a < 0 then
            print ('Sorry, the number was negative.');
    else
            begin
                b = sqrt(a);
                print ('The square root of ', a, ' is ', b);
```

```
            end
end
```

左尖括号（<）是小于号。如果程序的使用者输入的是一个小于 0 的数，if 语句中的判断语句为真，因此第一个 print 语句将会被执行。反之，如果该数大于或等于 0，则 else 关键字后面的语句块则会被执行。

本章目前所用到的变量都是一个变量对应一个值，我们也可以用一个变量对应多个值，数组（array）就是一个很好的选择。在 ALGOL 程序中可以这样声明一个数组：

```
real array a[1:100];
```

该语句定义一个数组变量 ***a***，它可以用来存放 100 个不同的浮点数，这些数被称做数组元素。可以使用数组名加标号的方式来引用数组元素，例如，第一个数组元素是 ***a***[1]，第二个是 ***a***[2]，最后一个是 ***a***[100]。方括号中的数字称做数组下标（index）。

下面的程序用来计算 1 ~ 100 所有数的平方根，将结果保存在一个数组中，然后再通过循环将这些结果显示出来。代码如下：

```
begin
    real array a[1:100];
    integer i;
    for i := 1 step 1 until 100 do
            a[i] := sqrt(i);

    for i := 1 step 1 until 100 do
            print ('The square root of ', i, ' is ', a[i]);
end
```

程序中还定义了一个整型变量 *i*（由于它是 integer 的首字母，经常被程序员用做整型变量名）。第一个 for 循环的执行过程中，每个数组元素被赋值为其下标的平方根；第二个 for 循环执行过程中，数组中的每一个元素被显示出来。

变量的类型有很多，除了我们已经介绍过的实型和整型之外，变量还可以被声明为布尔型（Boolean，该名称是为了纪念第 10 章提到的乔治·布尔）。布尔变量的取值只可能有两种，即 true 和 false。在本章的最后将介绍一个用到布尔数组的例子（这个例子也将用到目前所介绍的大部分内容），来实现一个寻找素数的著名算法——爱拉托逊斯筛法（Sieve of Eratosthenes）。爱拉托逊斯（约公元前 276–196 年）传说是亚历山大图书馆的管

理员，他因准确计算出地球的周长而永载史册。

素数是只能被 1 及其本身整除的一类整数。第一个素数是 2（也是唯一的偶数素数），其他的素数还包括 3，5，7，11，13，17，等等。

爱拉托逊斯方法以 2 开始的整数表开始，因为 2 是素数，因此所有可以被 2 整除的数都被排除掉（即除了 2 之外的全部偶数）。接下来是 3，因为 3 是素数，因此所有能被 3 整除的数也被排除掉。因为 4 在第一个步骤中已经被排除掉，所以下一个要考虑的数是 5，即排除所有 5 的倍数。按这种方式不断循环，最后剩下的都是素数。

下面的 ALGOL 程序用来筛选 2～10,000 之间的所有素数，程序中定义了一个布尔数组，用来对所有的数进行标识。该程序如下：

```
begin
    Boolean array a[2:10000];
    integer i, j;

    for i := 2 step 1 until 10000 do
    a[i] := true;

    for i := 2 step 1 until 100 do
          if a[i] then
                  for j := 2 step 1 until 10000 ÷ i do
                        a[i × j] := false;

    for i := 2 step 1 until 10000 do
          if a[i] then
                print (i);
end
```

第一个 for 循环将数组 ***a*** 的每一个元素的初始值设置为布尔值 true。这里的 true 表示该位置的数是素数，因此现在程序默认所有的数都是素数。第二个 for 循环的范围是 1～100（100 刚好是 10000 的平方根）。在第二个 for 循环中，如果判断条件成立，该数为素数，即 ***a[i]***为 true，则第三个 for 循环则会把该数的所有小于或等于 10000 的倍数（除了其本身）设置为 false，因为这些数都不是素数。最后的 for 循环用来输出所有的素数，这里的判断条件是：若 ***a[i]***为 true，则 *i* 为素数。

程序设计到底是一门科学还是一门艺术呢？这的确是一个有趣的问题，一些人甚至

还为此争论不休：一方面，你或许在大学里系统地学习了计算机科学（Computer Science）课程；另一方面，你又读过如唐纳德·克努斯（Donald Knuth）的名著《计算机编程艺术系列》（*The Art of Computer Programming series*）等著作。然而物理学家理查德·费叶曼（Richard Feynman）曾这样写道："从某种程度上看计算机科学像是一种工程，它的工作范畴是利用一些事物去实现其他事物。"

在程序设计中有一种现象：如果让 100 个人来编写输出素数的程序，你可能会得到 100 个不同的解决方法。就算所有的程序员都使用"爱拉托逊斯筛法"来解决这个问题，其最后所写的程序也不一定与本文所写程序完全相同。如果说程序设计是一门科学，那么就不应该出现如此多的解法，而不正确的方法将会非常明显。偶尔，一个程序设计问题会诱发出极富创造性的火花或洞若观火般的觉察力，这就是所谓的程序设计的"艺术"。但是，程序设计的更多的时候是设计和建造，就像修建一座大桥的过程。

早期的程序设计对编程人员的要求很高，所以很多早期的程序员都是科学家或工程师，他们通常利用 FORTRAN 或 ALGOL 中的数学算法来描述并解决各自领域的问题。回顾程序设计语言发展的整个历程时，我们会发现，人们一直在努力开发一种能为更大范围的人群所使用的语言。

第一个成功地为商务系统所使用的程序设计语言是 COBOL（COmmon Business Oriented Language），今天它仍然被广泛使用。COBOL 于 1959 年开始开发，由美国工业界和国防部组成的委员会发起并实施，它的设计思路受到格瑞斯·霍珀早期编译器的影响。从某些方面来看，COBOL 的设计中渗透了这种思想：使管理人员——可能并不进行实际的编码工作——但他们至少可以看懂程序代码，而且能够检测程序能否完成预定工作（实际上这种情况非常少见）。

COBOL 语言广泛支持读取记录（record）和生成报表（report）。记录是按照统一方式归类整理的信息的集合。例如，保险公司一般会维护一个包括其所售的所有保险信息的大型文件，每一项保险业务称为一条单独的记录。每一条记录包括客户的姓名、出生日期等信息。早期编写的 COBOL 程序，大都是为了处理存储在 IBM 打孔卡片上的 80 列记录而编写的。为了尽量减少孔洞所占用的卡片空间，年份通常设计成 2 位而不是 4 位，随着时间的推移，这个设计的缺陷逐渐显露出来，最终导致在 2000 年出现了著名的"千年虫问题"（millennium bug）。

在 20 世纪 60 年代中期，为了配合 System/360 项目的开发，IBM 同时开发了程序设计语言 PL/Ⅰ（Ⅰ是罗马数字中的 1，因此 PL/Ⅰ的含义是：Programming Language Number One）。PL/Ⅰ的设计者们想要使其融合 ALGOL 的块结构，FORTRAN 语言的数学函数功能以及 COBOL 处理记录和报表的能力，但该语言却远没有达到 FORTRAN 和 COBOL 那样广泛的使用程度。

虽然 FORTRAN，ALGOL，COBOL 以及 PL/Ⅰ都可以应用于家用计算机，但它们对于小型计算机的影响远没有 BASIC 语言那么深远。

BASIC（Beginner's All-purpose Symbolic Instruction Code）由达特茅斯（Dartmouth）大学数学系的约翰·克莫尼（John Kemeny）和托马斯·克鲁兹（Thomas Kurtz）在 1964 年开发，该语言最初是为达特茅斯分时系统而设计的。达特茅斯大学的学生并非数学或工程专业，因此他们不应该为打孔卡片和复杂的程序语法花费太多精力，他们要做的只是端坐于计算机终端前，在数字后面输入一些 BASIC 语句来完成编程。BASIC 语句前的数字用来指明该语句在程序中的次序。前面没有数字的语句是系统命令，如 SAVE（将 BASIC 程序保存至磁盘），LIST（按顺序显示行）以及 RUN（编译并运行程序）。BASIC 手册的第一版中的第一个程序是这样的：

```
10 LET X = (7 + 8) / 3
20 PRINT X
30 END
```

与 ALGOL 语言不同，BASIC 不要求程序员指定变量的存储类型，究竟一个变量是保存为整型还是浮点型并不需要程序员担心，大部分数默认都是以浮点数格式存储的。

很多 BASIC 的后续版本都是解释型（interpreter）而不是编译型（compiler）。如前所述，编译器读取源文件并生成一个可执行文件；而解释器却采取边读边执行的方式，不会产生新的文件。解释器比编译器的原理简单一些，因此更容易编写，但其运行程序的速度要比后者要慢。BASIC 语言应用于家用计算机的时间较晚，1975 年，比尔·盖茨（Bill Gates，生于 1955 年）和其好友保罗·艾伦（Paul Allen，生于 1953 年）为 Altair 8800 编写了 BASIC 解释器，这一事件可以视为 BASIC 在此领域的开端，同一年他们创建了微软公司（Microsoft Corporation）。

Pascal 程序设计语言继承了 ALGOL 的大部分结构，同时还继承了 COBOL 的记录处

理功能，它由瑞士计算机科学教授尼尔莱斯·沃思（Niklaus Wirth，生于1934年）在20世纪60年代末开发完成。IBM PC的程序员对Pascal非常青睐，而备受欢迎Pascal版本却是大名鼎鼎的Turbo Pascal。1983年，宝兰公司（Borland International）发布了Turbo Pascal，当时的售价是49.95美元。Turbo Pascal由一名叫安德斯·海尔斯伯格（Anders Hejlsberg，生于1960年）的丹麦大学生开发，它提供了完整的集成化开发环境（integrated development environment）。程序的文本编辑器和编译器集成在一起，这样就方便了程序的调试和运行，大大加快了程序开发速度。集成化开发环境以前主要用于大型计算机，Turbo Pascal实现了在小型计算机上的突破。

Pascal对Ada的影响也非常大。Ada是为美国国防部开发应用的一种语言，它以奥古斯塔·艾达·拜伦（Augusta Ada Byron）命名。在第18章曾提到过，奥古斯塔·艾达·拜伦是查尔斯·巴贝芝的解析机发展历程的记录者。

接下来就是C，一种深受喜爱的程序设计语言。C语言主要是由贝尔电话实验室的丹尼斯·M·里奇（Dennis M.Ritchie）开发的，从1969年开始设计并于1973年开发完成。人们常常对为什么以C来命名该语言感兴趣，答案其实很简单，它是一种早期的程序设计语言B的后继者。B是BCPL（Basic CPL）语言的一种精简版本，而BCPL来源于CPL（Combined Programming Language）。

如第22章所述，UNIX操作系统在设计的过程中充分考虑到了可移植性。当时的许多操作系统都是基于某种处理器的，并且使用汇编语言编写，基本上没有可移植性可言。1973年，UNIX采用C语言编写（更准确地说，应该是重写）成功，从此以后UNIX操作系统和C语言就变得密不可分了。

C是一种风格非常简洁的语言。例如，ALGOL和Pascal使用关键字begin和end来界定程序块，而在C中这两个单词被一对大括号“{}”取代。下面给出一个例子，程序员常常会把一个常量和一个变量相加，比如：

```
i = i + 5;
```

在C程序中，你可以将上面的语句简写为：

```
i += 5;
```

如果只需要把变量加1（即增量），则该语句还可以精简成下面这样：

```
i++;
```

在16位或32位微处理器中，i++这种语句仅需要一条机器码指令就可以执行。

在本章的前面曾讲过，很多高级语言都不支持移位操作和按位布尔运算操作，而许多处理器其实支持这类操作，C语言打破了这种局限，它广泛地支持这类运算。除此之外，C语言的另一重要特征是对于指针（pointer）的支持，指针本质是数字化描述的内存地址。C语言中的很多操作与通用处理器的指令非常相似，因此C也被称为高级汇编语言（high-level assembly language）。与类ALGOL语言相比，C的操作集与通用处理器的指令集接近程度更高，或者说远胜过它们。

但是，所有的类ALGOL语言——即大多数常用程序设计语言——其设计模式都是基于冯·诺依曼计算机体系的。设计一种非冯·诺依曼体系的程序设计语言并非易事，而让人们接受并使用这种语言则更加困难。LISP（List Processing）是一种非冯·诺依曼体系程序设计语言，它主要应用于人工智能领域，由约翰·麦卡锡（John McCarthy）在20世纪50年代末期开发完成。APL（A Programming Language）是另一种全新的语言，与LISP完全不同，它同样完成于20世纪50年代末期，由肯尼斯·艾佛森（Kenneth Iverson）开发。APL的特殊之处在于，它使用一个特殊的符号集，利用其中的符号可以一次性对整个数组里的数字完成操作。

类ALGOL语言一直在程序语言领域占据着重要地位，而且近年来，此类语言在一些方面进行了改进，导致面向对象程序设计语言（object-oriented language）的产生。面向对象语言主要应用在图形化操作系统中，我们将会在下一章（也是最后一章）介绍这种操作系统。

25 图形化革命

对于《生活》（*life*）杂志的读者而言，1945 年 9 月 10 日这一天的杂志像以往一样，有很多习以为常的文章和照片：第二次世界大战结束的相关新闻；讲述舞蹈家瓦斯拉夫·尼金斯基（Vaslav Nijinsky）在维也纳生活的点点滴滴；主题为美国汽车工人的图片新闻。但同时，在这一期的杂志中还有些不寻常的内容：万尼瓦尔·布什（Vannevar Bush 1890–1974）发表了一篇关于未来科学大胆猜想的文章。万·布什（人们常这样称呼他）的发明与贡献对计算机历史产生了深远的影响——其中最著名的就是他设计开发具有划时代意义的模拟计算机——微分分析器（The Differential Analyzer）——1927～1931 年，当时万·布什在担任麻省理工学院（以下简称为 MIT）工程学教授期间发明了这个机器。这篇文章在杂志上发表的时候，也就是 1945 年，布什所担任的职位是科学研究及开发办公室（Office of Scientific Research and Development，OSRD）的主任，负责美国战时科研活动的协调工作，其中就包括了曼哈顿计划（Manhattan Project）。

万·布什将自己两个月前在《大西洋月刊》（*The Atlantic Monthly*）上发表的一篇文章，通过浓缩精简，重新发表在《生活》杂志上，并将这篇文章最终取名为《思维之际》（*As We May Think*），文中描述了一种未来的发明，这项发明可以帮助科学家和研究人员更轻松地处理日益增多的技术期刊及文章。布什提出可以利用微缩胶片作为解决方案，同时他构想出了一种叫做麦克斯储存器（Memex，又名记忆扩展器）的设备，它可以对书籍、

文章、录音和图片进行保存。麦克斯储存器还有一项重要的功能，那就是它可以让用户根据某个主题在所有的素材之间建立起关联，这些关联的基本来源就是我们人类的思维。他还大胆预言一种新的职业群体，他们的工作就是在繁杂的信息载体之间提炼并建立起可靠的关联。

在20世纪的那个年代，讲述辉煌未来的文章屡见不鲜，但《思维之际》这篇文章却异常耀眼。它所讲述的不是可以替代我们去做家务劳动的设备，也不是关于未来运输方式或智能机器人的故事，这个故事的主角是信息（Information），故事的主线是如何利用新技术成功的处理信息。

回顾历史，从第一台继电器计算器出现到现在为止，65 年过去了，计算机的体积越来越小，处理速度越来越快，价格也越来越便宜。这一趋势极大地改变了计算的原始属性。当计算机价格变得很便宜，可以实现人手一台；当计算机体积越小、处理速度越快，软件就能发挥更大的作用，而机器就可以承担越来越多的工作。

要充分利用日益增长的运算和处理能力，较好的一种方法就是不断改进计算机系统中的关键部位，最典型的就是用户界面（User Interface）——它可以看作人机交互的轴心。人与计算机是两种完全不同形式的“客观存在”，只可惜在人机交互的这个过程中，与其让计算机去适应人类的特性，远不如劝服人们进行调整以适应计算机的特性来得容易。

在计算机发展早期，交互式这个概念并没有它的实际意义。人们编程时更多使用的是开关和电缆，有一部分人使用的是打孔纸带或胶片。到了20世纪50到60年代（有些观点认为这一时间可以延续到70年代），计算机已经可以使用批处理（batch processing）进行编程：程序和数据被“分布”在打孔卡上，然后一次性录入到计算机内存。这些工作完成之后，再由程序对数据进行分析，得出结论，最后将结果打印在纸上。

最早的交互式计算机运用的是电传打字机。我们回忆一下前面讲过的达特茅斯（Dartmouth）时分操作系统（原型出现于20世纪60年代早期），这种系统支持多个电传打字机同时工作，而且互不影响。此类系统中，用户在打字机上输入一行，计算机会相应地输出一行或多行。通常，打字机和计算机之间的信息交流是由一串 ASCII 码（也有可能是其他字符集）来完成的，这些ASCII码大多由字符编码组成，当然还包括像回车、换行等一系列简单的控制字符编码。随着机器的运行，相应的事务也随着打印纸的旋转逐步推进。

阴极射线管（cathode-ray tube，CRT，这是20世纪70年代随处可见的设备）并不受这类限制。使用软件来协调整个屏幕显得更加灵活方便——这可以算得上是一种二维的信息平台。但是为了尽量保持操作系统显示输出的逻辑一致性，早期那些为小型计算机编写的软件都把CRT显示器看做“玻璃屏幕电传打字机”——所有内容都是一行一地显示的，当字符排到底端，屏幕被填满时，屏幕上的内容要整体向上翻滚。除了CP/M（微处理机操作系统）中的所有工具软件之外，大部分MS-DOS下的工具软件都采用这种方法——它们都仿照电传打字机的工作方式来使用视频显示器。使用电传打字机这种工作原理的操作系统有很多，或许UNIX才算是最典型的原型操作系统之一，它还一直保留着这种“传统工艺”。

不巧的是，ASCII码字符集不完全适用于阴极射线管的工作方式。在最原始的ASCII码设计中，编码1Bh被标识为Escape，它的主要作用是帮助字符集进行扩充。在1979年，美国国家标准协会（American National Standards Institute，ANSI）发布了一项题为“ASCII码使用的附加控制（*Additional Controls for Use with American National Standard Code for Information Interchange*）”的标准。该标准发布的初衷是为了“适应二维字符-图像设备输入/输出控制中迫在眉睫的相关需求，其中包括阴极射线管和打印机之间的交互终端……”

其实Escape的编码1Bh只占据一个字节，且它的含义是唯一的。Escape如果作为一串序列的前缀字符，那么这串字符序列的含义也随之改变。比如下面这串序列：

```
1Bh 5Bh 32h 4Ah
```

可以看出Escape编码随后紧跟的是字符“[”“2”“J”的ASCII码，现在这一串字符的含义为“清屏”然后移动光标至左上角。这种定义在电传打字机上是不可能出现的。下面这串序列：

```
1Bh 5Bh 35h 3Bh 32h 39h 48h
```

即Escape编码随后紧跟的是字符“[”、“5”、“;”、“2”、“9”、“H”，这串字符的作用是把光标移到第5行的第29列。

键盘和CRT一起对远程计算机传输来的ASCII码（可能还包括Escape字符序列）做出响应，这种设备我们称之为哑终端（dumb terminal）。哑终端相对于电传打字机速度要更快，从某种程度来讲也更灵活，但从速度的提高程度上来讲，并不足以引领用户界面

的革新。真正的革新出现在20世纪70年代小型计算机中——它类似于第21章我们构建的假想计算机，配备了“视频显示存储器”，并作为微处理器地址空间的组成部分。

第一个预示着家用计算机将与它的孪生兄弟——体积庞大、价格昂贵的大型机划分界限的标志性的事件是VisiCalc的使用。VisiCalc由丹·布莱克林（Dan Bricklin，生于1951年）和鲍勃·弗兰克斯顿（Bob Frankston，生于1949年）设计并编程实现，而这套系统于1979年引入苹果II型电脑（Apple II）中。VisiCalc通过屏幕将一个二维电子数据表呈现给用户。在VisiCalc出现之前，数据表就是一张划分好了行、列的纸，主要用于一系列计算。VisiCalc用视频显示器将纸质材料取而代之，通过这种方式，用户可以在数据表中随处游走，在相应位置输入数据、公式，并在修改后对结果进行重新计算，为用户提供了更多的自由。

令我们惊讶与无奈的是，VisiCalc这款应用程序无法在大型机上运行。因为像VisiCalc这类程序需要以较快的速度不断更新屏幕，所以，它们直接将数据写入Apple II视频显示器所配备的RAM中。该RAM是微处理器地址空间的一部分。大型时分计算机以及哑终端之间的接口速度过慢，以至于电子报表程序无法使用。

计算机对键盘的响应速度越快，对视频显示器的更新速度越快，则人机交互就越频繁。在IBM PC刚刚推出的10年里（即20世纪80年代），几乎搭配的所有软件都是直接将输出的数据写入视频显示存储器的。当时IBM建立了一套硬件标准，其他硬件制造商参照这些标准去生产，这样软件制造商就可以绕过操作系统直接操控硬件，统一化的硬件标准确保了程序的正确运行（同时也杜绝了不能运行的情况）。如果所有同构的PC都拥有异构的视频显示器硬件接口，这种做法无异于将软件厂商推到了火坑里，因为做软件的同时还要关注硬件设计细节是不现实的。

IBM早期PC配备的应用程序通常只有字符输出，很少有图形输出。使用文本输出大大加快了应用程序的运行速度。假设PC上配备一台第21章所描述的视频显示器，那么程序所要做的就是把字符相应的ASCII码写入内存，然后屏幕上就会显示出该字符。但是如果使用的是图形视频显示设备，那么相应的程序需要将8个或更多的字节写入到内存中，这样做的目的就是画出字符的外观并以图形的方式显示。

在计算机的发展史上，从字符显示到图形显示是一次伟大的变革，计算机在这次变革中迈出了重要的一步。然而，相对于显示文本和数字所采用的软硬件，图形化计算机

的软硬件发展十分缓慢。早在 1945 年，约翰·冯·诺伊曼（John von Neumann）就预见了一种类似示波器的显示器，它的最大特点是可以显示图像化信息。但直到 20 世纪 50 年代早期，MIT（当时得到了 IBM 资助）建立了林肯实验室，实验室的主要任务就是帮助美国空军开发一种适用于防空系统的计算机，这次项目使计算机的图形化成为现实。该项目被称为半自动地面防空系统，简称 SAGE（Semi-Automatic Ground Environment），项目的内容包括构建一个显示图形的屏幕，以此来帮助操作员分析海量数据。

早期的视频显示器，比如 SAGE 中使用的这一种显示器，与我们今天所使用的 PC 配套显示器不尽相同。我们日常所用的 PC 配套的纯平显示器属于光栅（raster）显示器。它的原理就像电视机，每一幅图像背后都是一行行的光栅线，这些光栅是电子枪（electron gun）发出光束迅速来回移动覆盖整个屏幕而形成的。我们可以把屏幕想象成一个巨大的矩形阵列，阵列的每个元素都是一个点，这些点称为像素（pixels）。在计算机内部，有一块专门供视频显示器使用的内存区域，屏幕上的每一个像素点由 1 个或多个比特表示。这些二进制数值不仅决定了像素点的亮度，还决定了它的颜色。

举例来讲，当今大多数计算机显示器的水平分辨率至少为 640 个像素值，垂直分辨率至少为 480 个像素，像素总和即两数之乘积：307,200。如果为每个像素赋予 1 比特内存空间，这时每个像素点只能有两种颜色，通常设置为黑、白两种颜色，比如可以设置让 0 代表黑色，1 代表白色。这种视频显示器需要占据 307,200 比特的内存，换算过来就是 38,400 字节。

由于要用到的颜色数目逐渐增加，为了表示这些颜色，每个像素所需要的比特越来越多，显示适配器需要配备的存储器容量也越来越大。比如我们想使像素点具备不同的灰度，那么可以提供一个字节的存储空间。在这种处理方式下，字节 00h 代表着黑色，FFh 代表着白色，两者之间的值代表着不同的灰度。

CRT 上的色彩空间由三个电子枪产生，每一个电子枪分别产生三原色中的一种，包括红色、绿色、蓝色（用放大镜来观察电视机或彩色计算机屏幕，你可以清楚地看到，每一幅图像都是利用许许多多不同的三原色组合显示出来的），红绿组合出黄色，红蓝组合出品红色，蓝绿组合是青色，三原色组合出白色。

在最简单的彩色显示适配器中，表示每个像素点需要 3 个比特。最直观的编码方式就是每一种原色对应编码中的 1 位。