第24章　高级语言和低级语言

用机器码编程就像用牙签吃东西，刺的块很小且做起来很费力，吃一顿饭要花很长时间。同样，每个机器码字节只是完成可以想像得到的最小且最简单的计算工作——从内存装入一个数至处理器，与把它另一个数相加，再把结果存回到内存——所以，很难想像机器码如何完成一整项的工作。

至此，我们至少已从第22章开始处的原始模型阶段有了一些进步，从前我们一直用控制面板上的开关输入二进制数据到内存。在第22章里，介绍了如何写简单的程序用键盘输入并在视频显示器上检查机器码的十六进制字节码。这当然不错，但还不是改进的终点。

正如我们所知道的，机器码字节与某些短的助记符相关联，如MOV、ADD、CALL和HLT，因此，可以用一些模糊类似的英语来引用机器码。这些助记符通常与操作数写在一起，进一步表明机器码指令的功能。例如：8080机器码字节46h使得微处理器把寄存器对HL中的16位数寻址的内存单元所存放的内容传送到寄存器B。这可以很简明地写成：

```
MOV  B,[HL]
```

当然，用汇编语言编写程序比用机器码要容易得多，但微处理器并不能理解汇编语言。我们已经讲过如何在纸上编写汇编程序，只有当想在微处理器上运行汇编语言程序的时候，才会手工汇编程序，意思是把汇编语言语句转换成机器码字节并输入到内存。

如果能让计算机来做这项转换工作就更好了。如果你正在8080计算机上运行CP/M操作系统，则你已经具有了所需要的工具。下面介绍它是如何工作的。

首先，建立一个文本文件，包含有你用汇编语言编写的程序。可以用CP/M程序ED.COM来完成这项工作。该程序是一个文本编辑器，可用来创建、修改文本文件。现假设创建的文本文件名称为PROGRAM1.ASM。ASM文件类型表明该文件是汇编语言程序文件，该文件看起来就像下面这样：

```
        ORG  0100h
        LXI  DE, Text
        MVI  C,9
        CALL  5
        RET
Text:   DB  'Hello!$'
        END
```

文件中有几个语句以前没有见过。第一个是ORG（origin）语句，该语句不对应于任何一条8080指令，它表示下一条语句的地址从地址0100h处开始。前面讲过，这就是CP/M程序装入到内存的地址。

下一个语句是LXI（load extended immediate）指令，它装入一个16位数到寄存器对DE。本例中，16位数由标号Text给出。标号在程序的下部，DB（Data Byte）语句之前。DB也是以前未见到的，DB语句后面可以是几个逗号隔开的字节或（就像本例中）在单引号里的一些字符。

MVI（move immediate）语句把数9送到寄存器C。CALL 5 语句进行CP/M功能调用。功能9的意思是：显示一个字符串，起始地址由寄存器对 DE给出，遇到美元符号结束。（注意，文本以美元符号作为字符串的结束是很奇怪的，但 CP/M就采用这种方法。）最后的RET语句用来结束程序并把控制权返还给 CP/M（这实际上是结束CP/M程序的几种方法之一）。END语句表示汇编语言文件结束。

即然已经有了7行文本的文本文件，下一步是汇编该文件，即把它转换成机器码。以前是用手工来完成，自从运行CP/M后，可以用包含在CP/M中的名为ASM.COM的程序来完成。这个程序是CP/M汇编程序，在CP/M命令行运行ASM.COM，方法为：

```
ASM    PROGRAM1.ASM
```

ASM程序汇编PROGRAM1.ASM文件并创建新的文件，名为 PROGRAM1.COM，它含有与编写的汇编语言程序相对应的机器码（实际上，在这个过程中还有另外一步，但在这里并不重要）。现在，在CP/M命令行就可以运行PROGRAM1.COM，结果显示字符“ Hello! ”，然后结束。

PROGRAM1.COM文件包含有下面16个字节：

```
11 09  01  0E  09  CD  05  00  C9  48  65  6C  6C   6F  21  24
```

前面3个字节是LXI指令，紧接着2个字节是MVI指令，再后面3个字节是CALL指令，然后是RET指令，最后7个字节是“ Hello ”、感叹号和美元符号的ASCII码。

像ASM.COM这样的汇编程序所做的工作是：读入一个汇编语言程序（常称作源代码文件），产生一个包含有机器码的文件 —— 可执行文件。从大的方面来看，汇编程序是相当简单的程序，因为在汇编语言助记符与机器码之间存在一一对应的关系。汇编程序把每一个文本行分成助记符和参数，然后把这些单词和字符与一张表相对照，该表中存有所有可能的助记符和参数。通过这种对照就可以找到每个语句所对应的机器码指令。

注意汇编程序是如何得出 LXI指令必须把寄存器对 DE设置为地址0109h的。如果LXI指令本身在0100h处（CP/M把程序装入内存运行时的地址），则0109h是就Text字符串的开始地址。通常，使用汇编程序的程序员并不需要关心程序各部分的地址。

当然，第一个编写汇编程序的人必须手工汇编程序。在同一台计算机上编写新的（或改进）汇编程序的人可以用汇编语言编程然后用最初的汇编程序来汇编。一旦新的汇编程序经过了汇编，它也可用来汇编自身。

每当一个新的微处理器诞生，就需要新的汇编程序。新的汇编程序可以在已有的计算机上编写，利用原有的汇编程序来汇编。这种汇编称之为交叉汇编，即用在计算机A上的汇编程序来生成在计算机B上运行的代码。

尽管汇编程序消除了汇编语言编程缺少创造性这一问题（手工汇编部分），但汇编语言还存在两个主要问题，第一个（也许你已经猜测到了）是汇编语言程序冗长、乏味。因为你是在微处理器芯片级编程，所以必须要考虑每一个细节。

第二个问题是汇编语言不可移植。如果为 Initel 8080编写汇编语言程序，则不适用于在Motorola的6800上运行，必须用6800的汇编语言重新编程。也许，这不像编写最初的汇编语言程序那么困难，因为已经解决了主要的组织和算法问题，但是，仍然有许多工作要做。

上一章解释了现代微处理器芯片如何集成机器码指令来进行浮点运算，这当然已经很方便了，但还不是十分令人满意。一种选择是彻底放弃与处理器相关的实现每个基本算术操作

的机器码，取而代之的是用代数符号来表示许多数学运算。以下是一个例子：

```
A×Sin（2×PI＋B）/C
```

这里A、B和C是数字，PI ＝ 3.14159。

既然如此，何乐而不为呢？如果这样的一条语句是在一个文本文件里，则可以编写汇编语言程序读取文本文件并把代数表达式转换成机器代码。

如果只计算一次这样的代数表达式，则可以手算或用计算器计算。如果需要用不同的 A、B、C值来计算表达式的值，则可能需要考虑如何用计算机来计算。正因为如此，代数表达式不可能单独出现，应该考虑到表达式的上下文，用不同的值代入计算。

现在已开始创建所谓的高级程序设计语言。汇编语言称作低级语言，因为它与计算机硬件密切相关。尽管“高级”用来描述除汇编语言以外的任何程序设计语言，但这些语言中，一些语言还比另一些语言更要高级一些。如果你是一家公司的总裁，且坐在计算机前输入“计算全年的收益和支出，做出年度报表，打印两千份给所有的股东”，那么你确实正在用非常高级的语言工作。在现实生活中，程序设计语言并没有达到这样理想的境界。

人类语言是千百年来复杂的影响、随机变化和不断适应的结果，即使像世界语这样的人工语言也来源于现实语言。然而，高级计算机语言是审慎而周密的概念语言。发明程序设计语言面临的挑战是如何使语言具有吸引力，因为语言定义了人们如何向计算机发送指令。从20世纪50年代开始到1993年，估计已发明和实现了1000多种高级语言。

当然，这还并不足以简单地定义高级语言（它牵涉到语言所采用的语法），还必须有编译程序用来将高级语言语句转换成机器码。像汇编程序一样，编译程序需要一个字符接一个字符地读取源代码文件，并分解成短语、符号和数字，但编译程序比汇编程序更复杂。从某种意义上讲，汇编程序较简单，因为在汇编语言语句与机器码之间有一一对应的关系。编译程序通常需要把一条高级语言语句转换成许多机器码指令。编译程序不容易编写，许多书中描述了它们的设计与构造，所以本书不作介绍了。

高级语言有优点也有缺点。最主要的优点是高级语言比汇编语言容易学且容易编写。用高级语言编写的程序清晰、简明。高级语言通常是可移植的 —— 也就是说，它不像汇编语言那样依赖于特定的处理器。所以，程序设计员不需要知道程序将要运行其上的机器的内部结构。当然，如果需要让程序在不止一种处理器上运行，则需要相应的编译程序生成针对这些处理器的机器码。可执行文件仍然只适用于某一个处理器。

另一方面，差不多都是如此，一个好的汇编语言程序设计员可以写出比编译程序所能产生的更优化的代码。也就是说，用高级语言编写的程序所产生的可执行文件比用汇编语言编写功能相同的程序所产生的可执行文件要大，且执行速度较慢。（最近几年，随着微处理器的日趋复杂以及编译程序在优化代码方面的日趋成熟，这种差别已变得不很明显。）

还有，尽管高级语言使得处理器更容易使用，但并没有使它的功能更强大。而使用汇编语言可以最大限度地利用处理器的能力。因为高级语言需要转换成机器码，所以高级语言只会降低处理器的能力。如果一个高级语言是真正可移植的，则它不能使用某种处理器的独有特点。

例如，许多处理器都有移位指令。前面讲过，这些指令把累加器中的位向左或向右移动。但是，几乎没有高级语言有这样的操作。如果程序中要用到移位，则需要用乘 2或除2操作来处理（其实，现在许多编译程序都用处理器的移位指令来实现乘或除以 2的幂）。许多高级语

言同样也不包括按位的逻辑运算。

早先的家用计算机中，许多应用程序是用汇编语言编写的，而现在除非有特殊需要，汇编语言已经很少用到。由于已在处理器中添加了一些硬件来实现流水线技术——同时有多个指令码累进执行——汇编语言则变得更难以掌握。与此同时，编译程序却逐步走向成熟。现代计算机的大容量存储能力也在这种趋势——程序设计员不再满足于编制在小的内存和磁盘上运行的代码——中也扮演着重要角色。

尽管许多早期计算机的设计者都试图用代数符号来阐明他们的观点，但通常认为第一个真正成功的编译程序是由Grace Murray Hopper（1906－1992）于1952年在雷明顿为UNIVAC而设计的A-0。当Hopper博士1944年为Howard Aiken工作时，就已开始了计算机的早期研究工作。在她80多岁时，仍然活跃在计算机界，为DEC公司作一些公关工作。

今天仍然在使用的最古老的高级语言（尽管这些年中得到了广泛的修改）是FORTRAN。许多计算机语言的名字都是大写字母，因为它们是由许多单词的首字母构成的。FORTRAN是由FORmula前3个字母和TRANslation的前4个字母混合而成，是在20世纪50年代中期由IBM公司为704系列计算机开发的。多年来，FORTRAN一直被选作为科学和工程的计算语言，它广泛支持浮点运算甚至支持复数运算。

所有计算机语言都有它们的支持者和批评者，并且人们可能只热衷于他们所喜好的语言。尽量站在中立的立场上，我选择一种语言作为原型来解释那些差不多再没有人用的程序设计概念。这种语言的名字是ALGOL（即ALGOrithmic Language）。ALGOL也可用来探索高级程序设计语言的本质，因为从某种意义上来说它正如一粒种子，成为过去40年来许多流行的通用语言的直接祖先。甚至在今天，人们也用到“类ALGOL”的程序设计语言。

它的第一个版本是ALGOL58，由一个国际委员会在1957～1958年设计而的。两年后，即1960年进行了修改，修订版命名为ALGOL 60。最后的版本是ALGOL 68。本章用到的ALGOL版本在文档“Revised Report on the Algorithmic Language ALGOL 60”中有描述，该文档在1962年完成，1963年第1次印刷。

让我们来编写一些ALGOL代码。假设一个名为ALGOL.COM的编译程序运行在CP/M或MS-DOS下。第一个ALGOL程序是一个名为FIRST.ALG的文本文件，注意它的文件类型。

一个ALGOL程序必须由begin和end作为开始和结束，以下为显示一行文本的程序：

```
begin
    print('This is my fist ALGOL program!');
ende
```

可以用ALGOL编译程序来编译FIRST.ALG 程序，操作如下：

```
ALGOL  FIRST.ALG
```

ALGOL编译程序的响应可能是显示类似于下面的内容：

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

ALGOL对拼写的挑剔不亚于传统的英语教师。在输入程序时若拼错了单词end，编译程序则会告知程序有一个语法错误。当它碰到ende时，它希望那是它可以识别的关键字。

修改了错误以后，可以再运行ALGOL编译程序。有时，编译程序会直接生成一个可执行文件（名为FIRST.COM，或者是MS-DOS下的FIRST.EXE）；有时，还需要进行另一个步骤。无论怎样，你都可以从命令行运行FIRST程序：

```
FIRST
```

FIRST程序的响应是显示：

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

糟糕！还有一个拼写错误。这是一个编译程序不能发现的错误，因此，称为运行时错误（run-time error）——即只在运行程序时才出现的错误。

可以看出，在该ALGOL程序中，print语句在屏幕上显示一些内容，本例是一行文本（因此，这个ALGOL程序等效于本章前面CP/M下的汇编程序）。print语句实际上并不是ALGOL语言正式定义的一部分，这里只假设正在用的这个ALGOL编译程序包含有这样一个实用工具，有时称作内部函数。print语句——就像许多ALGOL语句（除begin和end外）一样——后面必须跟引号。print语句向里缩进不是必须的，只不过使得程序结构更清晰。

假设要编写一个程序计算两个数的乘法。每一个程序设计语言都有变量这个概念。在程序中，变量名可以为一个字母、一个短的字母序列，甚至为一个短词。实际上，变量对应于一个内存单元，但在程序中是通过名字来引用的，并不是通过内存地址。下面这个程序有3个变量，名为a、b和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语句是说明语句，用来表明程序中要说明的变量。本例中，变量a、b、c是实数或浮点数（ALGOL也支持关键字integer，用来说明整型变量）。通常，程序设计语言要求变量名以字母开头。只要第一个字符是字母，变量名可以包含数字，但不能包含空格及许多其他字符。通常编译程序要限制变量名的长度。本章的例子都采用一个字母作为变量名。

如果使用的ALGOL编译程序支持IEEE浮点数标准，则程序中的3个变量都需要4个字节的存储空间（对单精度数）或8个字节的存储空间（对双精度数）。

接下来的三个语句是赋值语句。在ALGOL中，赋值语句定义为冒号后紧跟等号。（在许多计算机语言中，赋值语句只需用等号。）赋值语句的左边是变量，右边是表达式。前两个赋值语句是给a和b赋给一个值，第三个赋值语句中变量c的值由变量a和b产生。

今天，在程序设计语言中，大家熟悉的×（乘号）通常不允许使用，因为它不属于ASCII码和EBCDIC的字符集。许多程序设计语言用星号（*）表示乘法。虽然ALGOL用斜杠（/）表示除法，但也包括一个除号（÷）表示整数除法，即表明被除数中有多少倍的除数。ALGOL中也定义了箭头（↑），这是另一个非ASCII码字符，用来表示乘方。

最后是用来显示的print语句。本例中即有文本又有变量，它们用逗号隔开。显示ASCII字符可能并不是print语句的主要工作，本例中，它的功能还包括把浮点数转换成ASCII码：

```
The product of  535.43  and  289.711  is  155152.08653
```

接着程序终止，返回到操作系统。

如果想乘另外两个数，则需要修改程序，改变数，重新编译，再运行。可以利用一个名为read的内置函数来避免这种频繁的重新编译工作：

```
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语句从键盘读入ASCII码字符并转换成浮点数。

高级语言中一个非常重要的结构是循环。循环使得同一段程序依据一个变量的多个不同的值来运行。假设有一段程序用来计算3、5、7和9的立方，就可以这样做：

```
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语句设置变量a的初值为3，然后执行do关键字以后的语句。如果要执行的语句不止一条（本例中正是如此），则这些语句必须包括在begin和end之间，这两个关键字定义了一个语句块。for语句接着把变量a设置成5、7和9，并执行这些相同的语句。

下面是for语句的另一种形式，它计算3～99间奇数的立方值：

```
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语句设置变量a 的初值为3，然后执行for语句后的语句块。然后a以step关键字后面的值2为步长增加，得到新值5，并用来执行代码块。变量a不断加2，当它超过99时，for循环结束。

程序设计语言通常都有非常严格的语法。例如，在 ALGOL 60中，关键字for后只能跟一种类型的东西，即变量名。而在英语里，单词for后可以跟许多不同的单词，如“ for example ”。虽然编译程序不是容易编写的简单程序，但它显然要比解释人类语言的程序要简单得多了。

大多数程序设计语言的另一个重要特性是包含条件语句。条件语句只是在某个条件为真时才允许执行另一条语句。下面是使用 ALGOL内部函数sqrt的一个例子，用来计算平方根。sqrt函数不能用来处理负数，所以程序中应避免出现这种情况：

```
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。如果用户输入的一个数小于 0，则执行第一个print语句。否则，该数大于等于0，则执行包含另一个print语句的语句块。

到目前为此，本章程序中的每个变量只能存放一个值。用一个变量来存放多个值也是很方便的，这就是数组。ALGOL程序中声明一个数组的方法如下所示：

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

本例中，表明要用该变量来存储 100个不同的浮点值，这些值称作数组元素。第一个为a[1]，第二个为a[2]，最后一个为a[100]。方括号中的数字称作数组下标。

下例程序计算从1～100的所有数的平方根，把结果存放在数组中并显示出来：

```
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
```

程序中也声明了一个整型变量，名为 i（因为它是integer的第一个字母，所以经常用来作为整型变量名）。在第一个for 循环中，数组的每一个元素赋值为它的下标的平方根；第二个for循环中，输出这些值。

除了实型和整型外，变量还可以声明为布尔型（为了纪念第 10章提到的乔治·布尔）。一个布尔变量只有两个可能的值，即true和false。本章的最后一个程序里将用到布尔数组（和到目前为止学到的几乎所有特性）。该程序实现称为“ Eratosthenes漏勺”的用来找到素数的著名算法。Eratosthenes（大约公元前276-196年）是亚历山大传说中的图书馆的管理员，他由于精确地计算出了地球的圆周长而名垂史册。

素数是指只能被1和它本身整除的自然数。第一个素数是 2（唯一的偶数素数），此外，素数还有3、5、7、11、13、17等等。

Eratosthenes方法是从以2开始的正的自然数列表开始。因为 2是素数，则要删除所有是 2

的倍数的数（即除2以外的所有偶数），这些数都不是素数。因为3是素数，则要删除所有是3的倍数的数。已经知道4不是素数，因为它已被删除了。下一个素数是5，则要删除所有是5的倍数的数。依此类推，那些余下的数就是素数。

下面的ALGOL程序用来确定2～10 000的所有素数，通过声明一个布尔数组来标识从2～10000的所有数来实现该算法：

```
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循环把数组所有元素的布尔值设置为true。这样，程序一开始假设所有的数都是素数。第二个for循环从1～100（为10 000的平方根）。如果数是素数，意味着a[i]为真，则另一个for循环用来把该数的倍数设置为false，这些数都不是素数。最后一个for循环输出所有的素数，即a[i]为真时对应的i值。

有时人们在争论程序设计到底是一门艺术还是一门科学。一方面需要在大学里学习有关计算机科学的课程，另一方面又要看著名的如Donald Knuth的《The Art of Computer Programmign》系列这样的书。物理学家Richard Feynman写道“更确切的说，计算机科学更像工程——都是用一些东西来实现另一些东西”。

如果让100个不同的人都来编写输出素数的程序，将会得到100个不同的方法。即使这些程序员都有“Eratosthens漏勺”这种思想，也不会正好以同样的方法实现。如果程序设计真的是一门科学，就不会有如此多的方法，而不正确的解决方法也是经常有的。偶尔程序设计问题会激起富有创造性和敏锐观察力的火花，而这就是“艺术”的成分。但是，程序设计更多的是设计和组装的过程，就像在架设一座大桥。

早期的许多程序设计员都是科学家和工程师，他们利用FORTRAN和ALGOL所要求的数学算法来阐述自己的问题。然而，纵观程序设计语言的历史可以发现，人们希望有能被更大范围的人们所使用的语言。

第一个为商务系统设计的成功语言是COBOL（common business oriented language），今天仍被广泛使用。由美国工业和国防部组成的委员会于1959年早期推出了COBOL，它受到了Grace Hopper的早期编译程序的影响。从某种意义上说，COBOL使得管理人员——可能并不具体设计编码——至少可以看懂程序代码并且能够检查代码是否按所预定的去工作（在现实生活中，这种情况很少发生）。

COBOL广泛支持记录和生成报表。记录是按照一致方式组织的信息的集合体，例如：保

险公司可能要维持包含有它所卖的所有险种的一个大文件，每一险种为一单独记录，包括客户姓名、出生日期和其他信息。早期的许多 COBOL程序设计成能处理存储在IBM穿孔卡片上的80列记录，为了尽可能少地占用卡片空间，日期中的年份通常用 2位编码而不是4位，这导致了随着2000年的到来而普遍出现的“千年虫”问题。

20世纪60年代中期，伴随着System/360项目的开发，IBM公司开发了名为PL/I的程序设计语言（I是罗马数字1，PL/I表示programming language number one）。PL/I试图把ALGOL的块结构、FORTRAN的科学和数学计算功能以及 COBOL的记录和报表能力结合起来。但是，它却远没有像FORTRAN和COBOL那样流行。

尽管FORTRAN、ALGOL 、COBOL和PL/I都有适用于家用计算机的版本，但是它们都不具备BASIC所具备的那种对小计算机的影响力。

BASIC（beginner ’s all-purpose symbolic instruction code）是Dartmouth数学系的John Kemeny和Thomas Kurtz在1964年为Dartmouth的分时系统开发的。 Dartmouth的许多学生并非主修数学或工程课程，所以他们不能在穿孔卡片和很难的程序设计语法上花费很多时间。Dartmouth的学生坐在终端前，只需在数字之后简单地敲入 BASIC语句，即可建立BASIC程序。数字表明程序中语句的顺序。没有数字在前的语句是对系统的命令，如 SAVE（存储BASIC程序到磁盘）、LIST（按顺序显示行）和 RUN（编译和执行程序）。第一批印刷的BASIC指令手册中的第一个BASIC程序为：

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

不同于ALGOL，BASIC不需要程序设计员来指定一个变量是按整数存储还是浮点数存储。不需要程序员操心，大多数数都是按浮点数存储。

许多后来的BASIC版本是解释程序而不是编译程序的。前面讲过，编译程序是读取一个源文件，并产生一个可执行文件；而解释程序读取源代码并在读的过程中直接执行而不生成可执行文件。解释程序比编译程序容易编写，但是，解释程序的执行时间却比编译程序的执行时间要慢。当比尔·盖茨（生于 1955年）和他的密友保罗·艾伦（生于 1953年）在1975年为Altair 8800编写BASIC解释程序并创立他们的公司 —— 微软公司的时候，BASIC才开始应用到家用计算机中。

Pascal程序设计语言继承了 ALGOL的许多结构，但也包括了 COBOL的记录处理程序。该语言由瑞士计算机科学教授 Niklaus Wirth（生于1934年）在20世纪60年代后期设计而成。Pascal在IBM PC程序设计员中很受欢迎，但却以一种特殊的形式 —— Turbo Pascal这种产品形式流行。该产品于 1983年由Borland公司推出，售价为 $49.95。Turbo Pascal（由丹麦学生 Anders Hejlsberg（生于1960年）编写）是Pascal的一个版本，提供了完整的集成化开发环境。文本编辑器和编译程序集成在一个程序里，促进了快速编程。集成化开发环境在大型机上很流行，但Turbo Pascal却首先在小机器上实现了。

Pascal对Ada也有很大影响。 Ada是为美国国防部开发使用的一种语言，是以 Augusta Ada Byron命名的。第18章中已提到过这个人，他是查尔斯·巴贝芝的解析机的见证人。

然后就有了C语言，一种受到万般宠爱的程序设计语言。它于 1969年～1973年产生，大部分是由贝尔电话实验室的 Dennis M.Ritchie 完成的。人们常常问为什么叫 C语言，简单的回答是它来自于一种早期的语言 B，B是BCPL（Basic CPL）的一种简单版本，而 BCPL又来自于

CPL（combined programming language）。

第22章曾提到过UNIX操作系统被设计成可移植的形式。那时许多操作系统都是用汇编语言针对特定处理器而编写的。1973年，UNIX采用C来编写（更确切地说是重写）。从那时起，操作系统和C语言的关系就开始紧密起来。

C是很简洁的语言，例如，ALGOL和Pascal中用begin和end来定义的块，在C语言中用{}来代替。下面是另一个例子，该例对程序设计员来说是很常见的，就是把一个常量与一个变量相加：

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

在C语言中，可以简写为：

```
i+=5;
```

如果只需要把变量加1（即增量），甚至可以这样来简写语句：

```
i++;
```

在16位或32位微处理器中，这样一条语句可以由一条机器码指令来实现。

前面曾提到，许多高级语言不包括移位操作和按位逻辑操作，而这些是许多处理器所支持的操作，C语言是个例外。另外，C语言的另一重要特点是支持指针，指针实质上是数字化的内存地址。由于C有许多操作类似于常见的处理器指令，因而有时候也把C语言归类于高级汇编语言。胜过于任何类ALGOL语言，C更接近于常用的处理器指令集。

然而，所有的类ALGOL语言——即指常用的程序设计语言，是在冯·诺依曼计算机体系结构基础上设计而成的。在设计计算机语言时，突破冯·诺依曼框架并不容易，而让人们来使用这种语言则更加困难。一个非冯·诺依曼的语言是LISP（LISt Processing），是由John McCarthy在20世纪50年代末设计而成的，可用在人工智能领域。另一个与众不同且与LISP完全不同的语言是APL（A Programming Language），是由Kenneth Iverson也在20世纪50年代末开发而成的。APL采用了一个奇怪的符号集用来一次在整个数字数组上执行操作。

虽然类ALGOL语言仍保持着主导地位，最近几年，出现了叫作面向对象的程序设计语言，使这类语言的地位得到加强。这些面向对象语言与图形化操作系统一起使用，图形化内容在下一章（即最后一章）将作介绍。