FORTH语言基础及PC/FORTH系统

齐国光　曹谷崖　译编

清华大学出版社

内 容 简 介

本书分为二篇。第一篇全文译自 "Starting FORTH" 一书。该书是迄今为止最优秀的 FORTH 语言教科书。全书共分十二章，详尽深入地介绍了 FORTH 语言的结构原理和程序设计方法，深入浅出，通俗易懂。每章都附有习题，并统一给出习题答案。第二篇为编著，介绍了微型计算机系统实验室的 PC／FORTH 系统及其使用方法，为读者提供了学习 FORTH 语言的实践工具。

FORTH 语言是第四代计算机高级语言。它独具特点的结构形式，为用户提供了强有力的软件开发工具和环境，尤其适合于实时过程控制、机器人控制、图象处理、通讯，数据采集和分析，科学和医学仪器仪表等方面的应用。

本书可作为 FORTH 语言的基础教材，也可作为 FORTH 系统的用户指南。适合从事计算机工作的高等院校师生，科研人员，工程技术人员，FORTH 用户，以及初学者自学参考之用。

FORTH 语言基础及 PC/FORTH 系统

齐国光　曹谷崖　译编

☆

清华大学出版社出版

（北京　清华园）

北京奥海公司印刷厂排版

北京联华印刷厂印刷

新华书店北京发行所发行　各地新华书店经售

☆

开本：787×1092 1/16　印张：21.75　字数：541千字

1986年9月第 1 版　1986年9月第 1 次印刷

印数：00001～6500

书号：15235·237　　定价：4.45元

目　录

第一篇 FORTH 语言基础

序 言

STARTING FORTH 的出版是 FORTH 界值得祝贺、具有重要意义的事件。较之早先任何说明性手册，本书作出了更大的努力，加进了更多的技巧和内容。我尤其对于 FORTH 语言的日益流行感到特别高兴。

若干年前，我是把 FORTH 作为我和我为之编程的计算机之间的界面来开发的。传统的计算机语言不能提供给我所期望的功能、简易性或者灵活性。为了使该语言完全包含着有能力的程序设计员所需要的各种功能，我在 FORTH 中没有遵循一些常规的做法。这些功能中最重要的一点就是使 FORTH 具有凡是以后需要的任何功能都能被加入的能力。

开发初期，我把我已发展的概念组合成一个实体，在“第三代计算机” IBM1130 上运行，其结果看来是如此有效，因此我认为它是“第四代计算机语言”。我曾打算把它取名为 FOURTH。但是，由于 IBM1130 计算机只能接受 5 个字符的标识符，于是 FOURTH 便改为 FORTH。这就是 FORTH 这个名称的微妙出处。

直截了当地说，指导 FORTH 发展并继续指导它应用的基本原则就是保持它的简单性。简单的解决办法具有精巧性。采用这个原则是由于对实际问题深入理解的结果，同时也考虑到迫切需要恰当地简化。我之所以强调它的简单性是因为它与那种以为愈复杂功能愈强的常规观点相违背。简单性提供了可靠、准确、紧凑和速度。

STARTING FORTH 是由廖·布罗迪（Leo Brodie）编写的并附有大量插图。该作者是位才干非凡的人，他的洞察力和想象力是显而易见的。本书是研究 FORTH 语言的基本的和详细的说明书。它巧妙地引导初学者越过所有 FORTH 程序设计员了解 FORTH 所必须跨越的门槛。

尽管我是唯一不再需要学习 FORTH 的人，但我确知学习 FORTH 语言是件困难的事情。正如学习人类语言一样，FORTH 中许多词的用法必须熟记。对于初学者来说，廖的风趣的说明和巧妙的图示似乎使得这种学习变为容易和有趣的了。由于上述特点，象我这样已经熟知 FORTH 的人，快速阅读一遍本书也犹如故地重游，颇感愉快和新鲜。但是，我希望你不要以为本书易懂和风趣而忽视了它的价值。在此我要预先告诫：本书的内容是有份量的，通过它你能学到许多有关计算机，编译程序以及程序设计的知识。

FORTH 提供了人与他周围灵活的机器之间建立通讯的自然手段。这就要求 FORTH 遵守人类语言的特点，包括紧凑性，通用性和可扩充性。我不能设想还有一种比 FORTH 更好的语言可用于书写程序，表示算术式或理解计算机。当你阅读本书的时候，我期待你能赞成这种看法。

Charles H. Moore FORTH 的发明者

关 于 本 书

欢迎使用 STARING FORTH，这是一种令你振奋和极为有效的叫做 FORTH 的计算机语言。

如果您是一位希望了解更多有关计算机知识的初学者，FORTH 是你用来学习的有效途径。比起就我所知的任何其它计算机语言（见初学者导言），用 FORTH 来书写程序显得更为有趣。

如果您是一位想要研究 FORTH 的熟练的专业人员，本书正是你所需要的。FORTH 是一种非同一般的语言。因此，各种人，从新手到老手，学习 FORTH 语言最好是从基础开始。如果你已精通其它计算机语言，要把它们从你的记忆中丢开，只要记住你所了解的计算机即可（见专业员导言）。

由于许多人对 FORTH 不同的背景材料感兴趣，本书的安排将使你只需阅读你所必须了解的内容，而用脚注方式为各类读者加以说明。第七章的前半部分为初学者提供了计算机运算的有关知识。

本书讨论了如何用 FORTH 来编写简单的应用程序。它还包含了你用于编写高级的、单任务的应用程序所需要的全部标准 FORTH 词。这一套词极为有用，它包含了从简单的算术运算符到编译程序中各种控制词中的所有词。

本书没有讨论与汇编程序、多道程序、打印和磁盘管理实用程序以及目标编译程序有关的所有命令。这些命令只适用于某些 FORTH 版本，例如 POLYFORTH。

本书中我选择了将在具有终端和磁盘的 FORTH 系统上实际运行的例子。这并不意味着 FORTH 只局限于批处理或串处理任务，因为 FORTH 的实用性实际上是没有限制的。

下面是一些便于使用本书的特点：

所有命令都列表两次：第一次是在每节介绍该词时列出；第二次是在每章结束处摘要时列出。

每一章结束时有术语回顾和一组练习题。篇末列出了习题答案。

书中给出了一些揭示程序中的奥妙的或对读者有用的任选子程序的“提示”，但对它们的工作机理未加解释。

本人认为：FORTH 是一种不同一般的语言。它不遵循程序设计中的许多基本规则。起初我对 FORTH 语言也是特别怀疑的，但是当我尝试研制复杂的应用程序时，我开始看到了它的精巧和能力。当你读到它的某些奇异特点时，你自己不抱成见就会承认这些。在此，我要预先告诉你：几乎从来没有过研究 FORTH 的程序员会半途而废，再回头去研究其它语言。

愿你幸运，享受到研究 FORTH 的乐趣！

Leo Brodie
FORTH, Inc

导　言

一、初学者导言：什么是计算机语言？

初学者乍一听到“计算机语言”这一术语时，他们会感到惊奇，“计算机会讲哪种语言？它必定为人们所难于理解。你看

976#!@NX714&+

它根本什么都不象”。

实际上，要了解计算机语言并不困难。它的用途仅为在人与计算机之间的通讯上起适当的协调作用。

考察一个木偶。你只要简单地操纵控制木架，甚至不用触及拉线，木偶就会“行走”。那么，你可以说摇动控制木架就是木偶的语言——“行走”。木偶操纵者正是用这种木偶能够“理解”而操纵者又易于掌握的方法指挥木偶。

计算机正象木偶一样，人们也必须用一种特定的语言正确地告诉它干什么。因此，我们要指挥计算机就需要一种语言，这种语言要具有下述表面上看来似乎相互矛盾的两个特点：一方面，该语言对于计算机来说其意义必须十分精确，以便准确地传达计算机执行操作所必须了解的全部信息。另一方面，它又必须简单而且易于程序员使用。

自从计算机诞生以来已经开发出许多语言：FORTRAN 语言是这个领域中资格较老者；COBOL 是事务数据处理的标准语言；BASIC 是为初学者设计的语言，通过它可以学习象 FORTRAN 和 COBOL 这样更复杂的语言。本书所介绍的是一种与众不同的语言——FORTH 语言。在过去的几年中，FORTH 已逐渐稳步地流行开来，并在所有领域的程序员中得到推广。

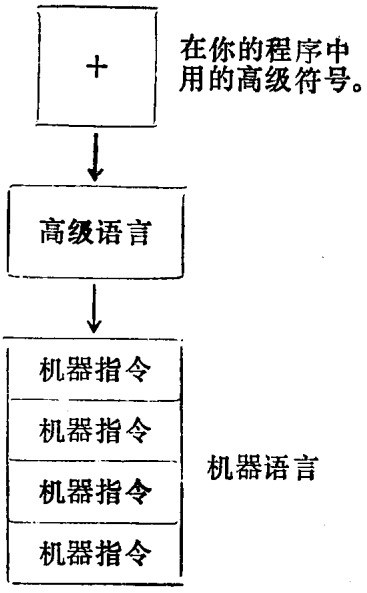
上面所述包括 FORTH 在内的所有语言都被称为“高级语言”。对于初学者而言，弄清楚高级语言与运用它的计算机之间的区别是很重要的。高级语言并不要求程序员注意到运用它的计算机的具体结构和型号。但每种结构和型号的计算机都具有它们各自的内部语言，或称“机器语言”。为了了解什么是机器语言，还让我们以木偶为例来加以说明。

假设木偶没有控制木架，操纵者必须由拉线来直接操纵木偶。每一条拉线恰好对应木偶身体的一个部位。这样，各条拉线运动的有机组合就可称为木偶的“机器语言”。

现在，若把所有拉线联结起来操纵，这种操纵就类似于高级语言。这时只要简单地旋转手腕，操纵者便能同时运动多条拉线。对于高级语言也是如此，在执行加法运算时，简单而熟悉的符号“+”要执行许多内部功能。

计算机是一种很灵巧的东西：它能被编程以把高级符号（例如“+”）翻译成自己的机器语言。然后继续执行机器指令。高级语言是一种把人们能够理解的词和符号翻译成实际结构和型号的计算机的机器语言的计算机程序。

FORTH 语言和其它高级语言之间的区别是什么呢？简言之，FORTH 是人与计算机之间各种因素的协调考虑。一种语言应当设计得方便于它的用户，同时又要兼顾到计算机的操作。FORTH 是诸语言中唯一达到这种要求的语言，因为它是唯一按这种要求来设计的。本书将说明 FORTH 是如何做到这一点的。



二、专业程序员导言：现实社会中的 FORTH

近年来颇受人们欣赏的 FORTH 语言已广泛流行开来，这在一些热心研究 FORTH 语言的人们和业余爱好者中间尤为明显。但这只是在 FORTH 的发展历史中激起的一层新的浪花。FORTH 已在严谨的科学研究和工业应用中使用了十多年。事实上，如果你使用的是小型或微型计算机，很可能使用 FORTH 比你现在正在使用的其它语言更能有效地完成你的任务。

可能你会问："既然 FORTH 如此有效，为什么我起初没有使用它"？回答是，象大多数人一样，你还不了解 FORTH 语言。

为了真正了解 FORTH，你应当阅读本书。如果可能的话，你还应当找一个 FORTH 系统亲自去试试它。本节将为那些仍在书店中浏览的人们回答两个问题："FORTH 是什么？""它的优点是什么？"

FORTH 是什么？

FORTH 可以说是：

——一种高级语言；——一种汇编语言；——一种操作系统；——一套开发工具；

——一种软件设计准则。

作为一种语言，FORTH 最初只具有一套功能极强的标准命令，然后提供了根据这套标准命令可以定义你自己的命令的机构。根据已有定义建立新定义的过程叫做 FORTH 的高级编码形式。有时，也可以使用 FORTH 的汇编程序以汇编助记符来直接定义各种词。所有命令都由同一解释程序来解释，由同一编译程序来编译，这就给予了该语言极强的适应性。

高级编码形式的代码类似于你的应用中的英语解释。因此有人把 FORTH 称为"元——应用语言"——一种能用于建立面向问题的语言的语言。

作为一种操作系统，FORTH 的功能与传统的操作系统相同，包括解释、编译、汇编、

虚拟存贮器处理、输入／输出、文本编辑等，但是由于设计方面的原因，FORTH 操作系统比起传统的操作系统要简单得多，运行之快，使用之方便，占内存之少都远远优于传统的操作系统。

FORTH 的优点是什么？

简单说来，FORTH 可以最大限度地发挥处理机的效率。例如：

FORTH 运行速度快。高级编码的 FORTH 其执行速度比其它高级语言都快，比等效的汇编语言程序只慢 20～75%。但对于那些时间是至关重要的程序，可以用汇编语言写，以使机器发挥出最高速度。不象传统的操作系统，FORTH 删除了过多的和不必要的运行过程错误检查。

FORTH 已编译好的代码是紧凑的。用 FORTH 编写的应用程序所需要的内存空间比用汇编语言编写的要少！用 FORTH 写出的整个操作系统及其标准词占用不到 8K 内存字节。支持目标程序可以少于 1K 字节。

FORTH 可以移植。工业中已知的几乎每一类小型和微型机都可以运用 FORTH。

FORTH 的程序开发时间是汇编语言开发时间的 1/10，是其它高级语言的 1/2。这是因为 FORTH 采用“结构性程序设计”，又是会话式和模块式语言。

下面是 FORTH 在现实社会中应用的几个例子：

过程控制——FORTH 曾用于控制自动电影摄像机，产生了影片“星球大战”中的特技效果。当时之所以选用 FORTH 是由于它能够使摄影师快速灵活地描述摄像机的运动。其它过程控制的应用例子还很多，从庞大的 U.S. 航空公司的行李传送装置到加利福尼亚食品罐头工厂的桃子分类器都使用 FORTH。

轻便的智能装置——用于门诊的心脏监听器，自动引燃检测器，测量各类谷物相对湿度的袖珍仪器和 Craig 语言翻译机都是用 FORTH 运行的。

医学方面的应用——在一台处理大量病历的 PDP-11 计算机上，FORTH 还同时管理一巨大的患者数据库：管理 32 个终端和一个光电输入机；对数据库进行统计分析以把患者身体类型、疾病情况、处理方案和诊断结果关联起来；分析血样并实时监听患者心脏博动情况。

数据采集和分析——一台用 FORTH 语言运行的PDP11/34计算机控制了整个观察系统，该系统包含一台非常精密的望远镜，圆盖，几个显示器，一个时钟，一台行式打印机和一台软盘驱动器。除此之外仍有时间来收集空间红外幅射的有关数据，以及分析这些数据并在图象监测器上显示其结果。这类应用经常使用用 FORTH 编写的快速傅里叶和沃尔什变换、数值积分和其它数学运算的子程序。

在此，我们必须承认，精通 FORTH 语言要比精通传统的高级语言困难些。后者基本上是相互类似的（即学会一种语言以后，学习另一种语言不会感到困难。）但是，一旦你精通了 FORTH 语言，它便会赋予你最大限度节省 CPU 时间和内存空间的能力，教会你一种科学的组织原则，运用这种原则，你能引人注目地减少目标程序开发时间。

还必须提到的是，FORTH 的所有组成部分，包括操作系统，编译程序，解释程序，文本编辑程序，虚拟存贮器，汇编程序和多道程序处理都遵循着同一约定，这就缩短了学习 FORTH 的时间。

如果你觉得看完本节后能令你振奋起来，请你往下开始学习 FORTH。

第一章 FORTH 基础

在这一章我们将向你介绍一些 FORTH 语言独有的特点。尔后，我们将教你使用 FORTH 终端。如果你没有 FORTH 终端，请不要担心，我们将把使用方法的每一步结果告诉你。

1.1 一种有生命力的语言

假设你是一位办公室管理人员并且刚刚雇用了一位热心的新助手。第一天你教给这位助手打印函件的正确格式（假定这位助手已掌握打字技术）。在当天结束时，你只需说“请打印这个”。

第二天你对他讲解档案制度。你花了一上午时间来解释如何归档，下午你就只需说“请把这个归档”。

到了周末你便可以用一种简记的形式同助手交谈了。“请发出这封信”，意思包含“打印它，让我签发，照相复制，附本归档，邮寄原件”。这样，你和你的助手就能更愉快更有效地自由执行你的商业事务。

良好的组织和有效的通讯要求你：

①对有用的任务加以定义并对它们命名；

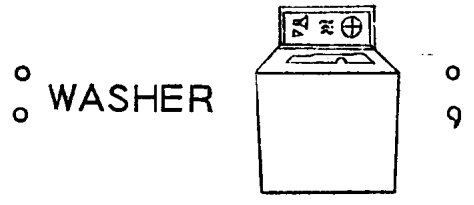
②把各个相关的任务组合成一些较大的任务，并给它们相应命名等等。

FORTH 正是按上述方法（除了你不必说“请”之外）让你来组织你自己的过程并和计算机建立联系。

作为一个例子，我们设想用 FORTH 编程来控制一台由微机控制的洗衣机。在该例中主要命令被命名为 WASHER。下面是用 FORTH 书写的 WASHER 的定义：

```
: WASHER   WASH SPIN RINSE SPIN ;
```

在 FORTH 中，冒号表示一个新定义的开始。冒号后的第一个词 WASHER 是新过程的名字。其余的词 WASH、SPIN、RINSE 和 SPIN 组成该新过程的定义。最后的分号表示定义结束。



组成 WASHER 定义的每一个词都应是在洗衣机控制程序中已被定义过的词。例如，察看定义 RINSE：

```
: RINSE   FILL AGITATE DRAIN ;
```

可以看到它是由词：FILL、AGITATE 和 DRAIN 组成的。而且这些词又应是本例中其它地方已定义过的词。例如可以查到 FILL 的定义：

```
: FILL   FAUCETS OPEN TILL-FULL FAUCETS CLOSE ;
```

在这个定义中，我们既涉及了对象（faucets，水龙头）又涉及了动作（open 和 close，开和关）。而且定义了词 TILL-FULL 去建立一“延迟循环”，它指示当水到达水平面时开关已被启动的时间，没有其它动作。

如果我们顺着这些定义察看下去，最终将会发现它们都是由一组非常有用的、组成全部 FORTH 系统的基本命令定义的。在 PolyFORTH 中大约包含 300 条基本命令。这些命令中的许多命令本身也是由如同本例中的“冒号定义”所定义的，另一部分由特定机器的机器语言直接定义。在 FORTH 中，一条已定义的命令称为一个“词”[1]。

用其它词定义一个词的能力称为“扩展性”。这种扩展性导致了一种非常简单、结构天然严谨、功能无限扩展的程序设计格式。无论你的程序是流水运行，还是获取科学计算、商业管理或游戏中的数据，你都可以根据特定需要建立你自己的“有生命力的语言“——各种词。

在本书中将包括最有用的标准 FORTH 命令。

1.2 人机会话

FORTH 有许多独有的特点，其中之一就是简单地呼叫一个词的名字便可以执行它。若在终端键盘上操作，只要键入词的名字再按 RETURN 键即可。自然也可以在其它词的定义中放入该词的名字，再由其它词来使用它。

由于只要键入一命令，机器便可执行该命令，所以 FORTH 被称为“会话式”语言。

首先我们给出一个你能亲自试验的例子，并显示出把一些简单的命令组合成一些功能极强的命令的步骤。在此我们将使用一些控制终端屏幕或打印机的简单 FORTH 词。现在让我们熟悉一下通过终端键盘与 FORTH “交谈”的技巧。

请坐在你实际的或假想的终端前面。我们将假定你已请求别人为你装备了使用 FORTH 的所有软硬件，或者你了解如何装备。

现在请按下键：

RETURN

这时计算机将回答： OK

RETURN 键是你用来使 FORTH 知道你的请求的信息。OK 是 FORTH 回答它已完成你的请求而且没有出错的信息。在上面的情况下，你要求 FORTH 不进行任何工作，所以它遵从你的请求不做任何事情，只是回答 OK。（OK 可能是大写字母，也可能是小写字母，这取决于你的终端）。

现在键入：

15 SPACES

如果你在键入过程中发生错误，可以按 “backspace” 键进行修改。方法是使光标回到发生错误的位置，重新键入正确的字符，然后继续。在一行键入正确后再按 RETURN 键。（一旦按下 RETURN 键后就无法再对本行进行修改了）。

注 1）有的FORTH 书刊称为“字”。为了不与存贮器的“字长”相混淆，我们译为“词”（word）。而且在本书中我们把内存中的一个字（16bit）称为一个“单元”（cell）。

在本书中，我们利用符号 [ ] 来表明你必须按 RETURN 键的地方。在计算机输出的下边划一道横线（尽管计算机不会这样做）以区别是由你键入的还是由计算机输出的字符串，例如：

15 SPACES [ ]
OK

当你按下 RETURN 键时，FORTH 随后打印出 15 个空格，再回打 OK，表示已处理了你的要求。

现在键入：

42 EMIT [ ]
* OK

短语 "42 EMIT" 告诉 FORTH 打印一个 "*" 号，因此 FORTH 打印出一个星号后再回答 OK。

我们可以在同一行中键入多个命令，例如：

15 SPACES 42 EMIT 42 EMIT [ ]
* * OK

这时，FORTH 打印出 15 个空格和两个星号。请注意：对于键入的词或数，可以用任意多个空格分开。但是要使 FORTH 能把它们当作词或者是数来识别，至少要用一个空格分开。

如果我们要多次使用短语：

42 EMIT

我们可以把它定义为一个词 "STAR"：

: STAR 42 EMIT ; [ ]
OK

其中 "STAR" 是名字，"42 EMIT" 是定义。请注意，必须把冒号和分号与邻近的词用空格隔开。另外，为了使 FORTH 定义易于为人们阅读，我们习惯于把定义的名字和内容用二个空格分开。

当你已经键入上述定义并按了 RETURN 键后，FORTH 回答 OK，这表明 FORTH 已认可该定义并将记住它。

现在键入：

STAR [ ]
* OK

FORTH 便执行你的定义而打印一个星号。

象 STAR 这样你自己定义的词和象 [EMIT] 这样系统中已定义的词之间并没有什么区别。但为了容易区分这两类词，在本书中我们将把系统定义的词用方框框上。

下面介绍另一个系统定义的词 [CR]，它在你的终端上执行换行。例如：

CR [ ]
OK

FORTH 则先执行回车换行，再打印 OK（在下一行）。

现在请试试：

CR STAR CR STAR CR STAR [ ]
*
*
* OK

请把 [CR] 放在定义中，如：

```
: MARGIN   CR 30 SPACES ;  [ ]  OK
```

现在我们就能键入：

```
MARGIN STAR MARGIN STAR MARGIN STAR [ ]
```

它将在离左边 30 个空格的垂直线上打印 3 个星号。

为了下面的需要我们把 MARGIN STAR 结合起来定义成一个词：

```
: BLIP   MARGIN STAR ;  [ ]  OK
```

还需要定义一个在水平方向打印多个星号的词（在后面我们将解释它是如何工作的）：

```
: STARS   0 DO STAR LOOP ;  [ ]  OK
```

现在我们可以键入：

```
5 STARS [ ]  * * * * * OK
```

或

```
15 STARS [ ]  * * * * * * * * * * * * * * * OK
```

或要求打印出任意多个星号。

我们还需要定义一个词，它先执行 MARGIN，然后再打印 5 个星号：

```
: BAR   MARGIN 5 STARS ;  [ ]  OK
```

现在我们可键入：

```
BAR BLIP BAR BLIP BLIP CR
```

便可得到一个由星号组成的字母“F”：

```
* * * * *
*
* * * * *
*
*
```

最后把它定义为一个词，我们称它为 F：

```
: F   BAR BLIP BAR BLIP BLIP CR ;  [ ]  OK
```

通过上述例子看出，简单的 FORTH 命令是组成复杂命令的基础。当列表显示 FORTH 程序时，可以看到它是由一系列功能逐步增强的定义组成，而不是由依次执行的指令序列组成。

为了给出一个真实的 FORTH 应用程序的例子，我们列出打印“F”字母的程序清单：

```
0  (  LARGE LETTER-F)
1  : STAR   42 EMIT ;
2  : STARS   0 DO STAR LOOP ;
3  : MARGIN   CR 30 SPACES ;
4  : BLIP   MARGIN STAR ;
5  : BAR   MARGIN 5 STARS ;
6  : F   BAR BLIP BAR BLIP BLIP CR ;
7
8
```

1.3 词 典

FORTH 的每个词和它的定义都被登记在 FORTH 词典中。机器启动时，词典中已包含了许多词，这时，你自己定义的词也可以登记进词典。

当你定义一个新词时，FORTH 便把你的定义翻释成词典形式写入词典条目。这个过程称为“编译”。例如，当你键入一行：

```
: STAR 42 EMIT ; [ENTER]
```

编译程序便把该新定义编译进词典。但编译程序不直接打印出星号。

一旦词被登记进词典，它是如何被执行的呢？让我们在终端直接键入下述行（不在定义内部）：

```
STAR 30 SPACES [ENTER]
```

这时将启动称为 INTERPRET 的词，也称为“文本解释程序”。该词进行如下操作：

①文本解释程序首先扫描输入流，寻找以空格分隔的字符串；

②当发现这样的字符串时，它便到词典中去查对该字符串；

③若在词典中找到这样一个词，它便把该词的定义指给一个称为 EXECUTE 的词，EXECUTE 便执行该词（这时便打印一个星号）。文本解释程序回答 OK；

④若在词典中找不到这样的字符串，文本解释程序便调用数字处理程序（叫做 NUMBER），若 NUMBER 发现它是一个数字则把它放入数字暂存区。

当你试图执行一个词典中尚不存在的词时会出现什么现象呢？例如键入：

```
XLERB [ENTER] XLERB?
```

这时文本解释程序在词典中找不到 XLERB，便试图把它传给 NUMBER。而它并非数字，NUMBER 拒绝接受。因此文本解释程序回答该串名字并跟随一问号“？”。

在某些 FORTH 版本中，包括 PolyFORTH，编译程序并不是把定义的全名复制进词典，只是复制前三个字符和全名的字符个数。例如，在PolyFORTH中文本解释程序不能区别 STAG 和 STAR，因为这两个词都是由 4 个字符组成且前三个字符又都是 S-T-A。尽管如此，许多专业程序员还是喜欢采用三字符规则，这样节省内存。某些程序员和许多业余爱好者都欣赏自由选择任意名字。标准 79-FORTH 可以把多到 31 个字符的名字存入词典。

: 也是一个词。现在解释一下文本解释程序对它的处理。如在：

```
: STAR 42 EMIT ; [ENTER]
```

中出现了 :，这时：

①文本解释程序在输入流中发现了冒号，则把它指给 EXECUTE，EXECUTE 则说“请开始编译”；

②编译程序便把该定义翻译成词典形式并写入词典；

③当编译程序遇到分号时则停止编译；

④返回到文本解释程序，给出信息 OK。