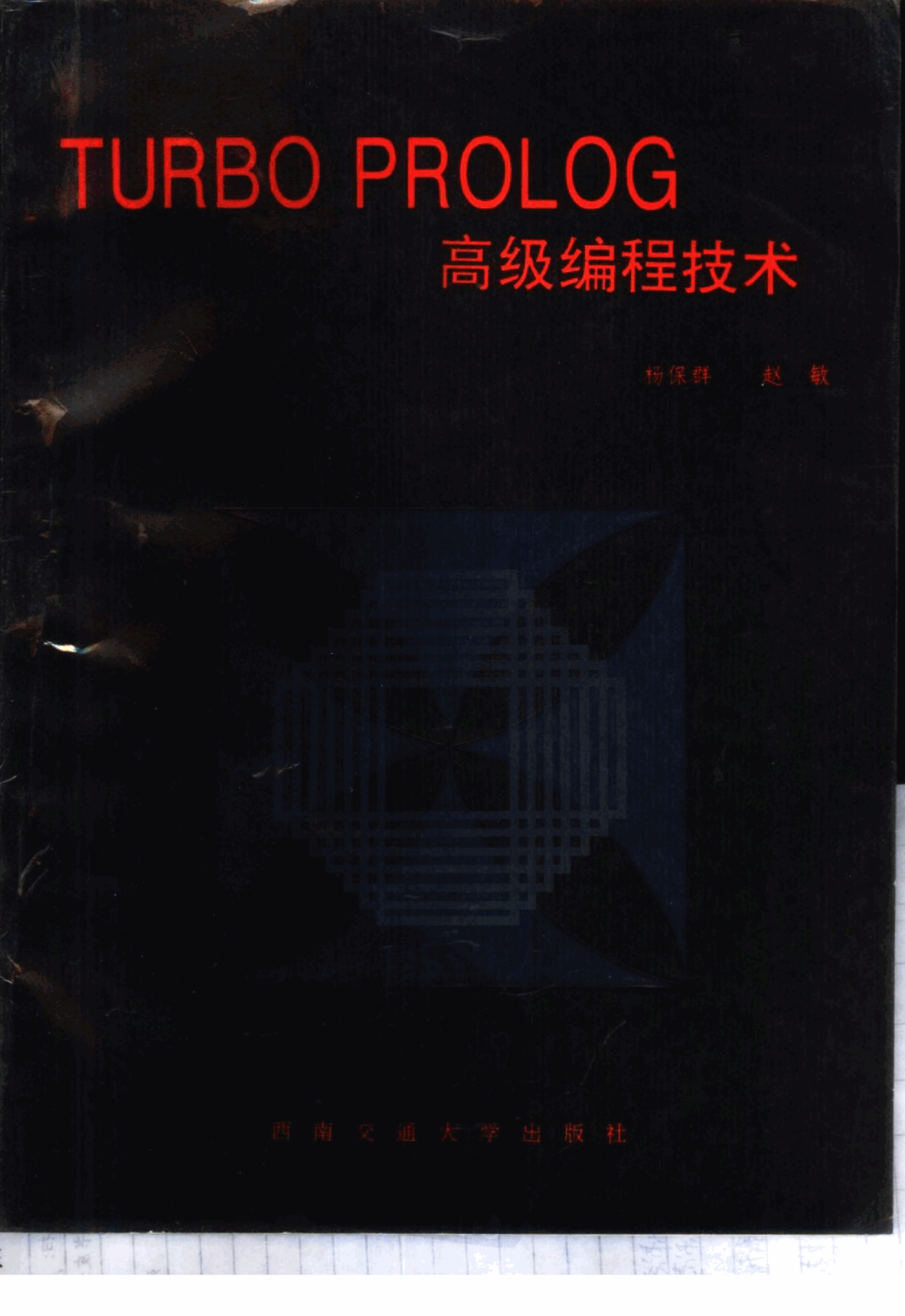
TURBO PROLOG

高级编程技术

杨保群　赵敏



西南交通大学出版社

前　言

Prolog是一种新颖的非过程型计算机程序设计语言——又称为陈述型语言。Prolog是"Programming in Logic"的简称，目前已越来越引起国内外从事计算机科学及人工智能人士的浓厚兴趣。它不仅可以解决过程型语言（如Basic、Fortran、Pascal …）难以解决的实际问题，而且也是人工智能理论研究的重要工具。

由于Prolog诞生较晚（到目前为止也只有十几年），而且在程序设计风格方面与传统的高级语言有着明显的差别，因此从使用角度来看，编程技术就不如传统的高级语言成熟，参考资料也不如传统的高级语言丰富，在实际编程工作中往往会发生这样或那样的问题。

我们和广大读者一样极希望有一本能进一步指导Prolog编程的书籍，为此，我们编译了这本《Turbo Prolog高级编程技术》，该书把编程中所遇到的一些主要问题都已基本包括进去。

本书共分五章。第一章主要解决模块化编程及Turbo Prolog工程诸问题，并详细举例说明了模块化编程的原则、方法和步骤以及动态数据库设计，而且还给出了建立Turbo Prolog工程的实例，使读者能一目了然地掌握这些编程的高级技术。第二章阐明Turbo Prolog编程中的另一个高级技术——Turbo Prolog和其它高级语言及DOS等系统的接口问题。以上二章可以说是本书的中心。有了这二章的知识，在实际编程中可能遇到的一些复杂技术问题就能迎刃而解了。第三章对研究自然语言处理的科学工作者来说是值得一读的。在这一章中举例说明了自然语言处理的原则、方法，并附有一些简短的程序，一般读者可以略去不读。第四章叙述了Turbo Prolog另一个突出的特点——逻辑推理。在这一章中列举了一些典型的智力游戏谜题及其程序，以说明Turbo Prolog是如何像人脑那样地来进行逻辑推理，同时可从这些程序中学到不少编程的技巧和推理的具体方法。在这一章中还介绍了用Turbo Prolog进行数学运算的问题，说明它还具有一般数学运算功能。第五章中提供了一些表管理、模拟调用和谓词传送的原则和方法，并附有一些实用谓词，在设计复杂的应用程序时可以直接调用它们。

在本书中我们虽运用Turbo Prolog结合实际课题开发了一些较大的程序，但毕竟时间不长，经验不够丰富，对原书的内容一定会有理解不透的地方，热情欢迎广大读者、同行们批评指正。

编　者

1991年8月于成都

目　录

第一章　模块化编程及 Turbo Prolog 工程

Turbo Prolog 的应用范围很广泛，而且采用的是模块化方法编程。本章的第一节将扼要地介绍一下模块化编程的定义、重要性和用它来编程的步骤。本章的其它部分是用模块化方法开发的一份小型应用程序。

1.1　模块化编程的概念

一、模块化编程的定义

模块化编程就是把一个较大的程序（在 Turbo Prolog 语言中称为“工程”）分解成许多部分，每一部分称为一个模块，然后对每一部分进行编程，最后将它们连接起来，构成一个完整的可执行程序。

二、模块化编程的优点

模块化编程有如下优点：

1. 编程人员可并行工作

当某一工程要求很急时，负责该工程的管理人员可以把此工程分解为若干模块，每个编程人员负责一块。这样就可使工作并行进行，加快速度，对完成任务是一种有效的方法。

2. 便于调试

由于所分成的模块是相对独立的，所以在调试总体程序的过程中发现问题时，很容易找出问题并解决之。因为程序模块化后，问题不外乎出现于两个方面：

(1)如果单个模块测试时是正确无误的，则问题往往发生在模块与模块的连接过程中。

(2)如果问题发生在某个模块之中，这时可将有问题的模块交给负责该模块的编程人员处理即可。

3. 易于管理

相对于通常的大规模程序来说，修改模块化程序中的问题要容易得多。

三、模块化编程的步骤

一般说来，模块化编程的步骤在不同的语言开发环境下，基本上是相同的。但 Turbo Prolog 有着已知事实和规则的特点，因此在模块化编程的步骤中有与其它语言不同的特殊步骤。

步骤 1　决定全局数据

连接到同一工程中的不同模块在执行过程中必须共享信息，并且彼此影响所有的数据，因此会有相同的数据文件和数据——变量、常数等等。我们把一个以上模块调用的数据称为全局数据或全局信息，把个别模块使用的数据称为局部数据或局部信息。

在设计模块化程序时，编程人员必须首先决定出哪些是全局信息，哪些是局部信息。在第二节中我们将研究如何来决定它们的准则。

步骤 2　设计模块

设计各个独立模块时，首先要考虑如何把大的工程分解成不同的模块，然后是设计模块的工作情况。详细情况将在第三节中叙述。

步骤 3　编写和测试模块

这一步与所应用的单个 Turbo Prolog 程序相同。将在第四节中讨论这个问题。

步骤 4　设计咨询的数据库

这是应用 Prolog 时专门要求的一步。要咨询的数据库存放在磁盘上，Turbo Prolog 应用它作为信息源以完成工程的任务。例如在 Turbo Prolog 的专家系统中，我们把职员情况存储在外界数据库中，人事管理人员可用它来作为推荐任免工作人员的信息。

自然这一步也可提前作好准备。数据库做起来并不困难，但是要花很多时间和精力去修改和补充，以满足工程的最终要求。第五节是有关这方面的内容。

步骤 5　准备模块连接

调试各个模块时是独立进行的，也就是说不考虑其它模块。但是，连到工程中去时往往要作些修改才行。例如，测试模块用的数据可能是臆造的局部数据，而在工程中这种数据应从磁盘数据库取得。这样在连接模块时就应对模块作些修改，以适应最终的数据来源。在 1.6 节中将专门讨论这一问题。

步骤 6　编译和连接模块

程序员在其它语言环境下对模块进行编译和连接时，还必须学会具有特殊语法和语句的编译或连接语言。而 Turbo Prolog 把连接程序和编译程序综合在一起，因此很容易进行连接和编译。但是，了解这一步骤的工作情况以及出错后怎样去修改仍很重要。具体情况可参阅第二章的 2.1 节。

四、Turbo Prolog 和模块化编程

对于 Turbo Prolog 来说，用模块化方法编程要比其它语言方便得多，因为它提供了优异的开发环境。为了能方便地进行模块化编程，Turbo Prolog 设计有二种专用的编程功能。

1. 模块表

在 Turbo Prolog 中设计了一种名叫“module list”(模块表)的特殊文件，它是文件菜单中的一项选择项，如图 1—1 所示。选取了此项后就会显示如图 1—2 所示的屏幕，在屏幕上的当前目录中列出了扩展名为 .PRJ(Project——工程)的所有文件。例如 GENEALOG.PRJ 文件就是名叫 GENEALOG 的工程。

图 1—3 就是工程 GENEALOG 的模块表的格式。图中列出了 GENEALOG 工程中所有的模块(在此省略了扩展名 .PRO)。模块之间必须用“+”号连起来。模块表最后的一个“+”号也不能省略。

2. 编译连接菜单

当确定了工程的模块表后，则可在主菜单中选取 Option 选择项，于是 Option 子菜单就显示在屏幕上，如图 1—4 所示。在此子菜单上选取 Project(all modules)选择项。最后在主菜单上选取 Compile 命令使 Turbo Prolog 编译该工程。至此，工程所具有的模块表就会调入到内存进行编译。如果编译无误，就会产生扩展名为 .EXE 的可执行文件。很清楚，程序中的模块以及外部文件的连接都是自动进行的。

注意：为了能正确地工作，工程中的模块表必须具有 .PRJ 的扩展名，而且驻留在 .OBJ 目

录中。其它模块必须在.PRO 目录中。必要时可以在编译前用主菜单中的 Setup 选择项改变这些目录。

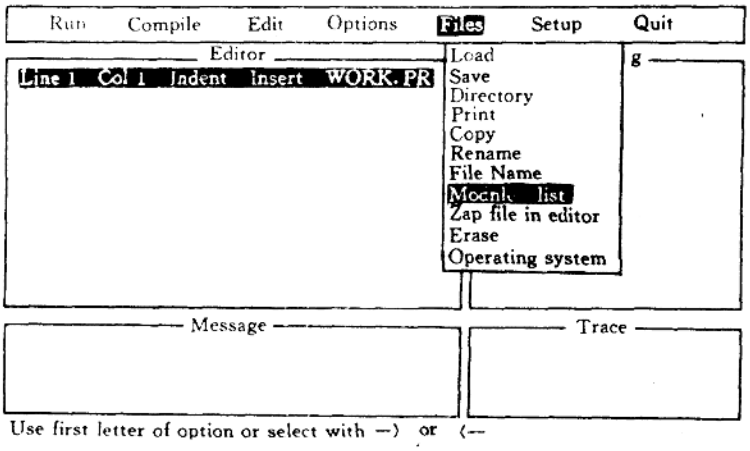


图 1—1 文件菜单中的模块表选择项

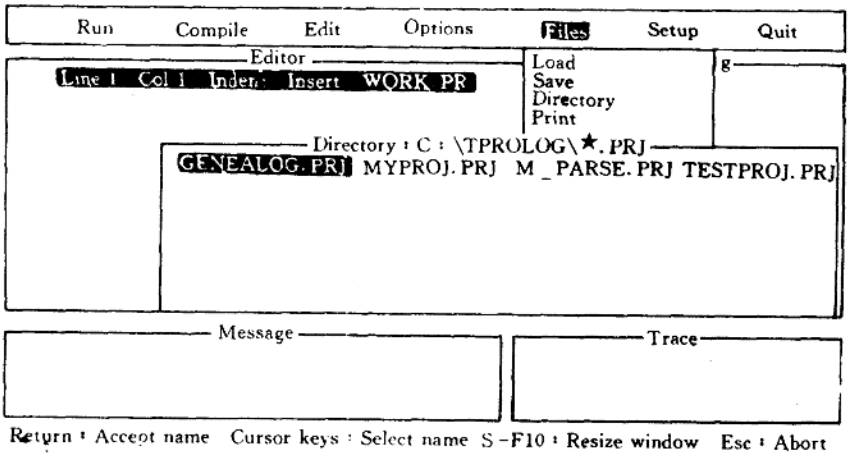


图 1—2 工程文件的目录

1.2 全局和局部领域及谓词

这一节主要介绍 Turbo Prolog 变量和其它编程语言变量的差别，同时还将讨论这种差别在 Turbo Prolog 模块化编程中产生的影响。最后还要阐述建立一项可执行的 Turbo Prolog 工程所必需的步骤。

一、Turbo Prolog 变量

在大多数计算机程序语言中，变量的作用是在整个作用域内保留信息。对于多数变量来说，系统确认的作用域是整个程序。也就是说当程序的每一部分需要时，可以存取或修改变量

的值。在最早的微机语言 BASIC 中，所有变量的作用域都是整个程序，别无其它可选择。对于 Pascal 那样的结构化微机语言，则存在着局部变量的情况。局部变量只能在程序的某一部分给以赋值使用，这部分程序通常称为过程，相当于 Turbo Prolog 的谓词。

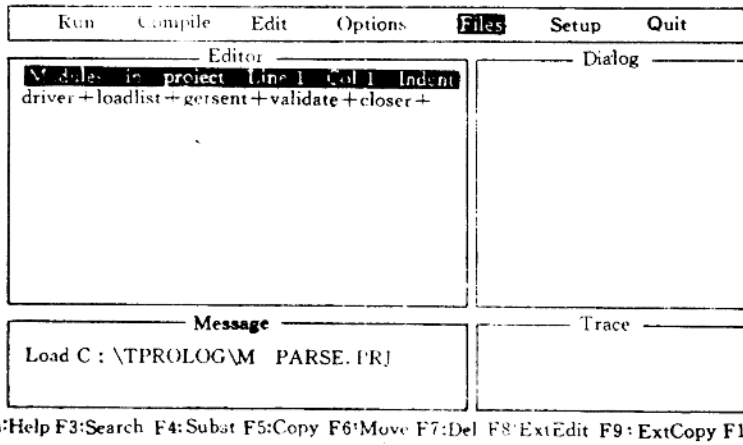


图 1—3 模块表的格式

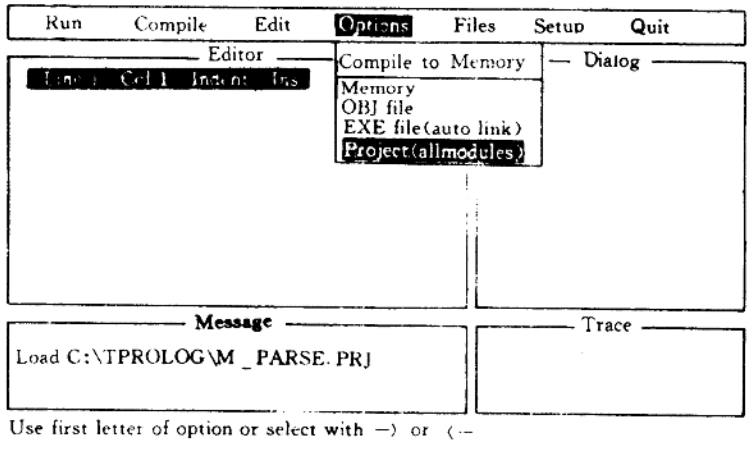


图 1—4 Options 子菜单

在编写程序时，局部变量是很重要的，它可以以同样的名字用在不同的地方而不会引起混乱。在 pascal(或其它语言)中，全局变量是系统默认的变量，而局部变量是需要定义的。

在 Turbo Prolog 中，变量处理的情况和其它语言是不尽相同的。为了能清楚地了解其情况，下面看几个例子。

Goal：X=9.

X=9

1 Solution

```
Goal : X=X+4.
Error 2201 Free variable in expression
```

从这里可以清楚地看出：得出结果后，X 就不保留其值 9 了。因此，当在第二条目标语句再次试用 X 求值时，就会出错：X 仍为没有给定其值的自由变量。

如果想把两个变量象一般语言那样相加起来，也会出错。例如：

```
Goal : X=9.
X=9
1 Solution
Goal : Y=4.
Y=4
1 Solution
Goal : A=X+Y.
Error 2201 Free variable in expression
```

在 Turbo Prolog 中，要把两个值通过变量加起来，只有将它们放在同一个谓词或目标语句中，如下例所示：

```
Goal : X=9,Y=4,A=X+Y.
X=9,Y=4,A=13
1 Solution
Goal :
```

总结上面几个例子情况可以看出：变量在 Turbo Prolog 中总是局部的，其范围是一条目标语句或一条谓词语句。

二、信息共享情况

变量局部化意味着程序中的某一部分不能直接存取同一程序另一部分的变量值。由于 Turbo Prolog 的变量都是局部变量，因此就产生了在程序段中如何共享信息的问题。这问题可从两方面考虑。

第一　Turbo Prolog 程序并没有需要共享信息的过程或程序段，因为 Turbo Prolog 是陈述式语言而不是过程式语言。

第二　需要在谓词间交换信息时是通过参数来进行的。

当某一谓词需要调用同一程序或模块中的另一个谓词时，它调用的是该谓词中的实例化或非实例化（变量是自由的，未曾赋以任何值）的自变量。下面将通过例子来阐明在谓词级共享信息的情况，这是理解用模块化编程技术编写 Turbo Prolog 程序的关键。

1. 谓词共享

下面是一份完整的 Turbo Prolog 程序，键入计算机并运行它：

```
domains
  person=symbol.
predicates
  parents(person,person,person).
  parent(person,person).
  ancestor_of(person,person).
```

```
clauses
  parent(A,B) :- parents(A,B,_).
  parent(A,B) :- parents(A,_,B).

  ancestor_of(X,Y) :- Parent(X,Y).
  ancestor_of(X,Y) :-
      parent(X,Z).
      ancestor_of(Z,Y).

  parents(ellen,ed,dianne).
  parents(ton,joe,vera).
  parents(anna,mel,corinne).
  parents(ed,don,eleanor).
  parents(dianne,brian,kathy).
  parents(ted,steve,susan).
  parents(steve,tom,anna).
```

这是一个典型的 Prolog 血统例子，ancestor 谓词的成功或失败决定于某两人之间是否是子女和父母的关系。所以在 ancestor 谓词中通过变量 X 和 Y 去调用 parent 谓词中的非实例化变量，以满足 ancestor 谓词的要求，而 parent 谓词中的变量名不需与 ancestor 谓词中的变量名相同。事实上，有经验的 Prolog 程序员往往使用不同的变量。

同样，parent 谓词通过 parents 谓词得到满足。parent 谓词中的两个变量名(自变量)与 parents 谓词中的变量名是相同的。

2. 变量共享

由于 Turbo Prolog 具有变量共享这个独特的性能，因此它比一般的语言更容易进行模块化编程。当 Prolog 程序运行时，如果有两个变量彼此匹配，则有一个变量实例化后，另一个也就实例化了。在我们所举的血统实例中，当 ancestor 谓词用变量 X 和 Y 调用 parent 谓词时，一般来说，其中一个变量已实例化了(并不是必须的)。parent 谓词有两个变量，所以调用它时 ancestor 谓词要给它两个值。这种变量共享情况如图 1—5 所示。

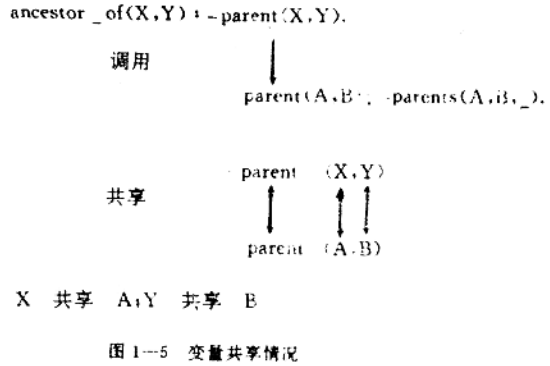


图 1—5 变量共享情况

不管在什么时候实例化，当 X 实例化后，A 也实例化了。如果在 ancestor 谓词调用 parent

谓词前X已实例化了，则当parent谓词要满足它自己的目标或子目标时，A也被实例化了。情况可用下面的例子说明。

应用前面的血统实例，如求目标：

```
Goal：ancestor _of(steve，Who).
```

则Parent谓词中的第一个变量实例化为steve，而第二个变量仍为自由变量，未被实例化。而后，parent谓词将这两个变量转递给parents谓词。也就是说：在Turbo Prolog程序中，不管该变量是否已实例化，变量可以在相应的谓词和相应的位置间转换（即可在程序的两部分间转换）。

三、全局说明

对于一份单独的Turbo Prolog程序，不需要作任何说明就可以在谓词一级实现共享。在每个谓词中应用不同的变量名字，程序可以运行；如用相同的名字，程序仍能运行。因为对Prolog来说，变量的名字是不重要的，重要的是变量在谓词中的位置。正是由于变量位置在谓词调用中的重要性，使得Turbo Prolog具有强大的模式匹配性能，可以很有效地处理多级谓词的交互作用。

还要解决的问题是：如何通过程序行共享信息？在模块化工程中信息如何从一程序转递给另一程序？从前面的算术例子中可以看到：不能通过目标语句来共享变量，也就是说，信息在目标语句中向前或向后转递。

为了使Turbo Prolog工程中的信息能在程序模块间经过谓词的调用转递，必须把这种谓词说明为全局谓词。又因为在Turbo Prolog中，谓词运行前还必须说明它的类型，所以在此也应把全局谓词的领域相应地说明为全局领域。

下面我们把血统的实例分解为两个程序段来说明问题。

1. 全局领域

如果我们把ancestor谓词段作为一个模块，程序的剩余部分作为第二个模块，分解后首先要了解什么信息在其间转递。

观察这两个程序段可以发现：只有通过parent谓词的信息需要前、后转递，因此parent是全局谓词。我们还可以用下面的方法更有效地找出全局谓词，即：把Turbo Prolog程序中两个程序段（或全部程序段）的所有谓词用表列出来，表中共有的谓词就是全局谓词。现在来看程序谓词段的parent谓词。在该谓词中有两个自变量，而且都是person。person并不是Turbo Prolog的标准领域，因此需要在程序的领域段内加以说明。在该程序的领域段内，person的数据类型被说明为symbol（符号）。

找出了全局谓词后，打开Turbo Prolog编辑窗口，建立一新文件。首先在领域段内把person说明为全局领域，如下所示：

```
global domains
    person=symbol.
```

由此可看出：全局领域的说明除了在domains前加一个关键字global外，其它与局部领域的说明是一样的。

2. 全局谓词

对person作了全局领域的说明后，Turbo Prolog工程中的所有程序段都能把person作为符号看待，但是还必须让所有的程序段都认得parent谓词，为此必须把parent谓词定义为

全局的谓词。在同一份文件中，只要在 person 的全局领域说明下增加下面两行即可：

```
global predicates
    parent (person, person) - (i, o), (o, i), (i, i), (o, o)
```

可以看出，说明全局谓词和说明全局领域是相似的，只有两个地方不一样：

(1) global 写在程序段的起始行。

(2) 定义的全局谓词后面有字母 i 和 o 的组合项。该组合项称为 Turbo Prolog 的流量模式。

最后，把全局领域说明和全局谓词说明存入 GLOBALS.PRO 文件中。

3. 流量模式

流量模式的作用是用来说明 Turbo Prolog 谓词中的参量是如何使用的。在单独程序中，由于没有信息要传到程序的外面去，所以就不需要作流量模式说明。但是在模块工程中，对于每一段程序或每一块模块来说，必须明确谓词中的参量在基本程序段或模块外是如何使用的以及是如何把信息转递给调用模块的。这就是流量模式的作用。

对所定义的全局谓词中的参量至少应有一级流量模式说明。如果谓词的自变量只有一个，而且只有一种模式可供使用，那么也要将这一种模式列出。

在流量模式说明中，i 表示输入参量，而 o 表示输出参量。如果某一参量在流量模式说明中的值是 i，则表示在转递时该位置上应是一种被实例化了的值；如果是 o，则在转递时不应是实例化了的值。可以参阅图 1—6 理解这种情况。

```
谓词名      (参量 1,参量 2,参量 3).
  test      (i,     i,      o).
能接受的模式值
  test (one,two,Variable).
不能接受的模式值
  test (X,two,three).
  test (X,Y,Z).
```

图 1—6 流量模式实例

在多重程序的 Turbo Prolog 工程中，大多数谓词具有多样的流量模式。在例子：

```
global predicates
    parent (person, person) -
  (i, o), (o, i), (i, i), (o, o).
```

中，共有四种流量模式，所以输入参量和输出参量的任何组合形式都是允许的。调用 parent 谓词时，可能与每种模式匹配的情况如图 1—7 所示。

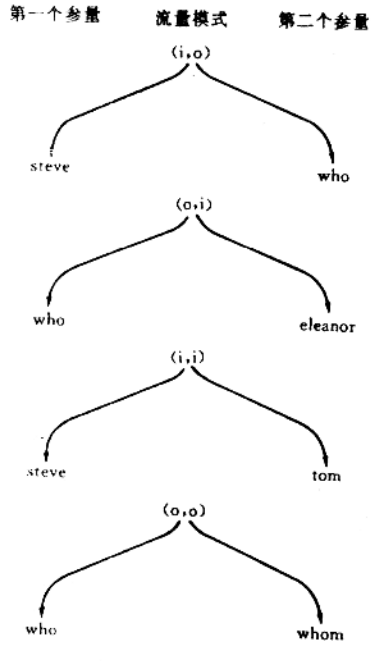


图 1—7 流量模式匹配实例

如果不想让二个参量为未知数（即未实例化）时调用 parent 谓词，只要把最后一种流量模式 (o, o) 取消掉就可以了。这样在运行时如果调用：

```
parent (Who, Whom).
```

就会出错。

谓词更复杂时，流量模式也更为复

杂，如何能保证程序正常运行，是一个需要很好考虑的问题。

如果我们能参阅 Turbo Prolog 用户手册中的所有内部谓词流量模式说明，就能知道它们工作时的要求，这对理解流量模式的概念会有很大帮助。

四、工程概念

进行全局领域和全局谓词说明是模块化编程的一个步骤。其它的情况分别介绍如下：

1. 工程和模块

Turbo Prolog 模块程序一般称作为工程，每一工程至少含有两个模块。图 1—8 为组成模块 Turbo Prolog 工程的成分图。

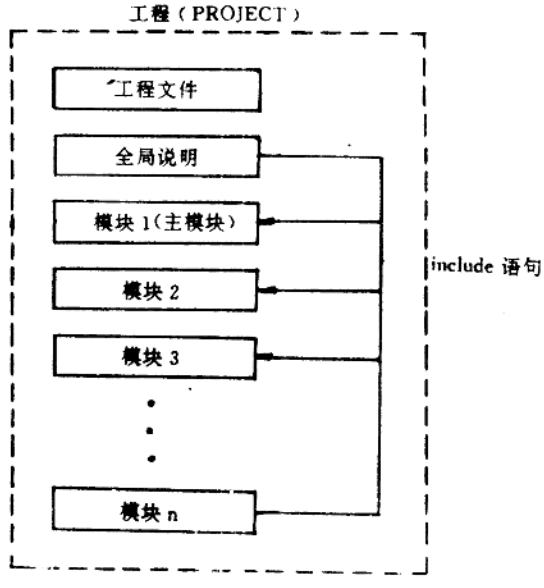


图 1—8 工程成分

第一个成分是工程文件。组成该工程的全部模块名列在工程文件中。我们可以按下面步骤把实例中的两个模块ANCESTOR. PRO和PARENT. PRO组成一个名为GENEALOG. PRT 的工程。

(1) 在 Files 的下拉菜单中选取 Module list 任选项。

(2) 回答 Turbo Prolog 询问的文件名——键入 GENEALOG（不须键入扩展名 .PRJ，程序会自动加上 .PRJ 扩展名）。

(3) 在编辑窗口上部显示一行提示行：Modules in Project. 如图 1—9 所示。

(4) 键入组成工程的模块名，模块名之间用“+”号相连接。要注意：在最后一块模块名后面也必须要有“+”号（如图 1—10 所示）。

(5) 列出了所有的模块后，按 F10 键将该文件存入计算机。

在此，我们要特别注意不能用 Esc 键进入 Files 菜单存储此文件，一定要用 F10 键。用了 Esc 键就会删掉模块表。

图 1—8 的全局说明是一份独立的文件。在此实例中是 GLOBALS. PRO。

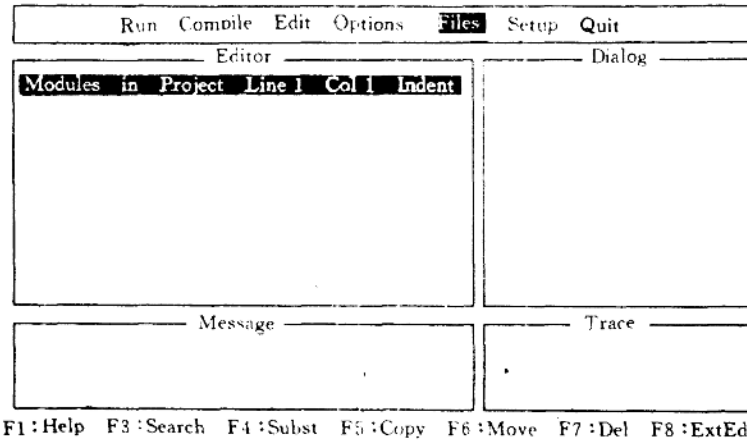


图 1—9 列模块表编辑窗口

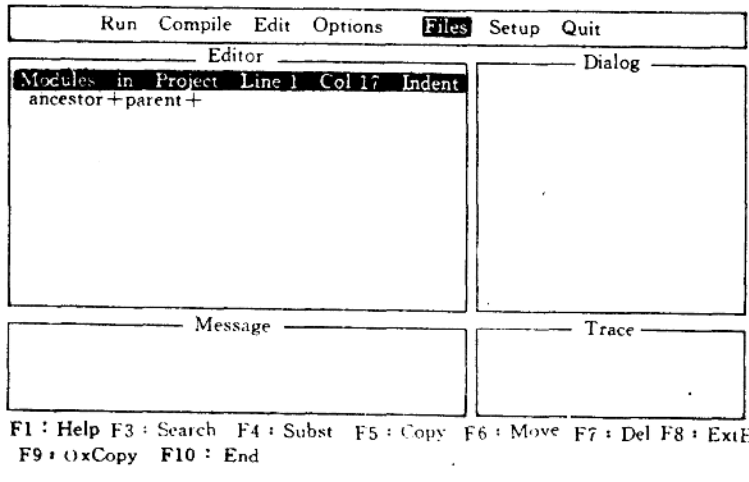


图 1—10 实例工程的模块表

在每一模块中具有一条 include 语句（将在下面介绍），以便标出含有全局说明的文件。

图 1—8 中的第一块模块称为 Main Module（主模块）。在 Turbo Prolog 工程中只能有一条目标语句。大多数情况下目标是一个谓词，而这个谓词就放在主模块中。因为与程序的交互作用只发生在由 Turbo Prolog 产生的执行程序中，而不是在 Turbo Prolog 中。程序是不会产生 goal：提示行的。目标谓词放在主模块中后，Turbo Prolog 工程经过编译运行时，就会向用户提问以决定“做什么”。但是只有单一目标（如对固定值进行某种计算或从其它语言的文件中获取数据，这种情况将在第二章中讨论）时，可能就不会有交互作用。

2. 模块的建立

正常情况下，我们是键入模块来建立 Turbo Prolog 工程的，但是为了容易理解模块与工

程的情况，我们在这里把前面的血统实例分解为两个模块。

用 Turbo Prolog 编辑程序把 ancestor _of 谓词从血统实例的程序中分离出来，编成一新程序存在 ANCESTOR . PRO 文件中，同时把它们从主程序中删去。将剩下的内容存入 PARENT. PRO 文件中。

调用编辑程序，把：

```
project "GENEALOG"
include "GLOBALS . PRO"
```

分别加在 ANCESTOR. PRO 和 PARENT. PRO 文件的开头。

注意：键入工程名"GENEALOG"时，不须键入扩展名 . PRJ；而键入定义的全局文件名"GLOBALS . PRO"时，必须要有扩展名 . PRO。

把领域段从两份文件中删去。从新文件 ANCESTOR. PRO 中删去领域说明中的 parent 和 parents 谓词并存起来。打开 PARENT. PRO 文件，删去 ancestor _of 和 parest 谓词并存起来。

打开 ANCESTOR. PRO 文件，在子句说明前插入下面的目标：

```
goal
    ancestor _of(steve,who).
    write(Who, "is an ancestor of steve."),nl.
```

此目标语句使血统程序变成了单一目标的程序，它将不断地寻找 Steve 的祖辈。

当我们把两个模块(分别为 ANCESTOR. PRO 文件和 PARENT. PRO 文件)和全局说明文件(即 GLOBALS . PRO)编辑好后，就可以进行编译并将它们连接成一个整体。其步骤是：

(1)在 Options 的下拉菜单中，选择 Project(all modules)任选项。

(2)按 Esc - C 进入编译。

如果编译进行得顺利，则在信息窗口的左下角能看到一系列的信息。编译和连接成功后，在磁盘上就会产生一个名为 GENEALOG. EXE 的新程序。Turbo Prolog 将询问你是否要运行此程序，如果想试运行一下，则可回答 y。但是现在不忙运行，所以回答 n。

此程序也可以在 DOS 系统下运行，只要在 DOS 提示符时键入 GENEALOG 即可，但是只能给出一个答案(在此例中可有六个答案)。至此，我们建立了第一份模块化的 Turbo Prolog 单程序。

小　结

在这一章中我们学习了：

(1)Turbo Prolog 变量以及与其它编程语言不同的地方。

(2)在处理过程中，信息是如何使用共享变量通过谓词在程序间转递的。

(3)Turbo Prolog 模块化编程的工程中所具有的主要成分。

(4)设计和建立一份工程所需要的步骤。

整章内容是通过一个小的实例程序加以阐述的。

1.3　模 块 设 计

对于模块化工程，仔细地设计各种模块的内容是非常重要的一步。设计模块时，必须考虑

它们的功能、大小以及独立(与系统其它部分无关)测试和诊断的可能性;还必须有一个主程序,以便各个模块测试完后能把它们组装在一起.

这一节将介绍如何把工程模块化以及设计主模块。有了主模块才能使系统中的其它模块工作。我们可以按照下面的顺序完成这个任务。

(1)分析工程的设计和过程,定出划分模块的原则和地点——位置;

(2)建立模块表,以便 Turbo Prolog 进行连接;

(3)写出主模块(主驱动)程序,使整个工程运行起来。

一、工程实例和工程目标

下面我们将从一个自然语言处理(NLP)实例来说明如何将一工程分解成若干模块,如何编制工程程序及模块程序,如何将它们连接起来。这个实例虽然不大,但其说明的方法却有普遍意义。

我们把此实例的程序称为 Micro－Parse。图 1—11 是其功能简图(这种简图不但可以说明实例的功能;而且还可用它来决定模块的划分)。

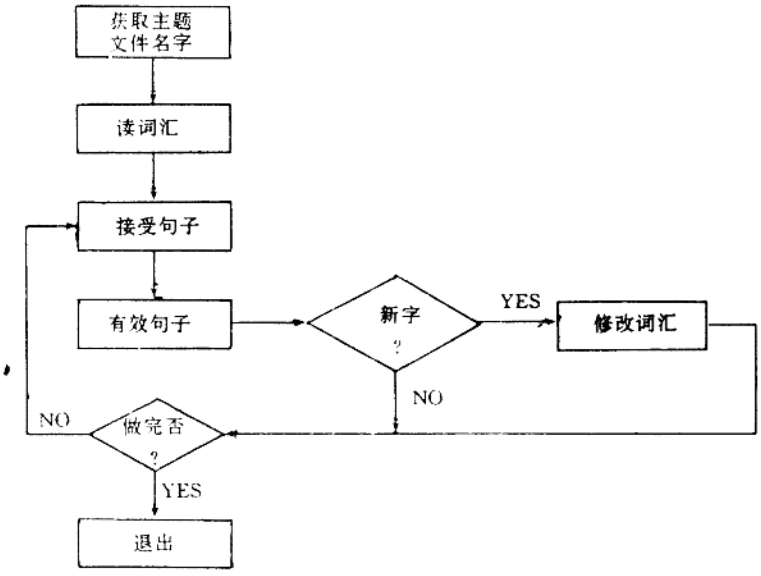


图 1—11　Micro－Parse 简图

Micro－Parse 的目标是让用户选择一个主语和输入与此主语有关的句子,而后由程序来进行分析(自然要把主语的一些字事先存放在磁盘文件中)。这是很一般的分析,包括辨认句型中的品词,询问品词以及维护此主题字的磁盘文件。

为了管理好程序,我们把程序能识别的句型(即有效句子)限制在一定范围内。要扩大此范围也不困难,但要有很多的信息量。

二、模块的划分

最终的程序有五块模块:

(1)主模块(或驱动模块)

(2)主题文件管理模块

(3)语句输入模块

(4)语句有效性检验模块

(5)退出模块(或结束模块)

图 1—12 是根据图 1—11 划分成模块的简图。现在我们来解释为什么要这样划分的理由。

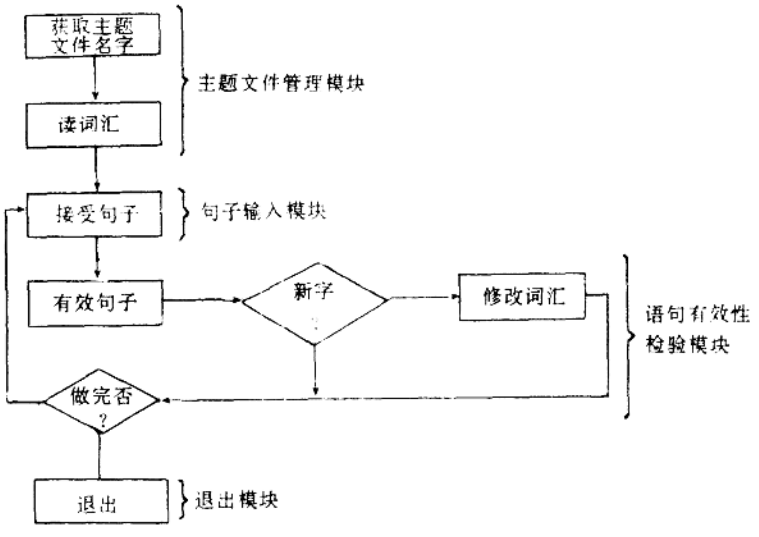


图 1—12 Micro - Parse 模块划分图

(1)头两步在这过程中仅仅完成从磁盘读入字文件并将此文件读入这一模块的功能,因此可把这两步作为一个功能单元而建立一块模块。该模块应能处理一些错误的信息:如不正确的磁盘文件、非法的文件名和不存在的磁盘驱动器。建立时要注意到模块测试的独立性,这是一个主要的特性。测试时我们可以假设一些小的样本文件,制造一些人为错误让模块去处理,这样就可确认模块能正确工作的环境。

(2)接受句子这一步的作用仅是接受从用户那儿输入的句子,并且确保它是符合一些起码条件的,因此可把这一步建立为一块独立的模块——语句输入模块。

(3)图 1—12 上的第三块模块是为检验语句的有效性而设立的,同时可以适当地往动态数据库中增加一些字。这是此工程或大程序的心脏。初看起来这块模块好像应是语句输入模块的一部分,那末为什么我们要把它设立为一块独立的模块呢?原因是:

第一　在程序内是用表形式的数据结构对语句进行处理的,而用户输入的是类英语语句。所以我们要用语句输入模块专门接受用户的输入。

第二　设置了有效性检验模块后,可以不管输入的情况而独立地对语句进行检验,判断它们是否合法。

第三　有效性检验模块可以在输入语句模块和退出模块之间起桥梁作用。它可以调用退出模块,反过来也可调用输入语句模块。如果把输入语句模块和有效性检验模块合在一起,则会使得交互作用不方便或含糊不清。

另外我们为什么要把有效性检验和修改词汇的两段程序作为一块模块而不分别设立两块模块呢?主要是对用户的输入进行有效性检验和判断用户输入的字是否是新词汇时需要大量

的交互作用。由于它们的功能是相同的，所以交互作用的情况也是一样的，因此将它们放在一起时只是在模块中增加一些字，与编写两块模块的程序相比是既小又简单。如果我们能分析一下这块模块的程序，就会更清楚地了解合并为一块模块的优越性了。

(4)退出模块的作用是很明显的，即退出程序的运行。

三、划分模块的准则

从上面的实例我们可以总结出设计 Turbo Prolog 工程时划分模块的一般准则。这不是必须严格遵守的原则，而是一种指导性的准则。

(1)将工程模块化并不存在着绝对正确和错误的方法。

(2)输入、输出或与其它模块有相同交互作用的模块可以合并成一块模块。

(3)从模块的大小来考虑，特别小的模块可以考虑合并到其它有关的模块中去，而特别大的模块可再划小一些。自然大小没有定量的概念，根据实际情况及经验来决定。

(4)准则的关键是模块的独立性。对模块进行测试和求值时，如果与其它模块完全无关，则这是一块好模块。如果测试时需要从别的模块输入信息，而这种信息又不易重复，那末把这两块模块合为一块就更恰当。

(5)需要和多块模块进行交互作用的模块(如退出模块)，哪怕很小也单独形成一块模块为宜。

我们要记住：只有在开发阶段模块化是有意义的。当程序编写、测试、诊断和编译连接后就变成一份工程了。因此模块化的目的是能方便地开发和调试程序。

四、模块表

模块划分后，接下来的工作就是设计工程本身，可以从建立模块表开始。这样做的理由有二：

其一　可以看出工程结构的全貌。因为在最后完成的工程中包含有这些模块。

其二　在后面的论述中要用到模块表时，便于了解其开发过程。

1. 模块表的结构

模块表由组成工程的各程序名组成。每一行列一程序名或把程序名都列在一行上。但是，每一程序名后必须有一"+"号。程序名本身不需有扩展名。

在此实例中，我们的模块表中共有五份文件名。文件名可以随便取，但最好能符合模块的含义，以便使人能据其名知其意。实例的模块表如下所示：

```
driver+   /*main driver routine*/
loadlist+   /*get user input for word list to be loaded and load it */
getsent +-/*get user's sentence to be validated */
validate+   /* validate and update vocabulary as appropriate */
closer+   /* user finished processing - close out if he wants to quit */
```

也可以把它们压缩写在同一行上：

```
driver+loadlist+getsent+validate+closer+
```

注意：在模块名和"+"号之间不要有空格。

2. 模块表的建立

可按下列步骤建立模块表：