

C语言集成生成系统 (CLIPS)

专家系统程序设计语言

(V4.0版)

岳文元 房桂荣 师钧 编译

中国科学院希望高级电脑技术公司

C语言集成生成系统 (CLIPS)

专家系统程序设计语言

岳文元 房桂荣 师 钧 编译

中国科学院希望高级电脑技术公司

一九九〇年九月

版权所有

翻印必究

■ 北京市新闻出版局

准印证号：891436

■ 订购单位：北京8721信箱资料部

■ 邮码：100080

■ 电话：2562329

■ 传真：01—2561057

■ 乘车：302 320 332 路车至

海淀黄庄下车

■ 办公地点：希望公司大楼101房间

序 言

美国国家航空和宇宙航行局—NASA人工智能部在向NAS用户推广以LISP语言为基础的专家系统中遇到了许多问题妨碍了NASA专家系统的使用。第一,使用效力低;第二,LISP工具及硬件成本高;第三,与其它语言集成能力差。为了解决这些问题,NASA人工智能部开发了一种专家系统工具,它是用C语言写的,而且完全与C语言兼容,称为C语言集成生成系统(“C” Language Integrated Production System)——CLIPS。CLIPS将提供较高的可移植性、较低的成本及与外界系统较容易集成的特性。

CLIPS最基本的表达方法是一种建立在Rete算法基础上的正向链规则语言。它在很大范围的计算机系统上都能提供相当好的性能,还包括一些查错工具。目前CLIPS已广泛地使用在NASA的一些部门、许多学校及工业界。

本书共包括三大部份:第一部份CLIPS用户指南;第二部份CLIPS基本程序设计指南;第三部份CLIPS高级程序设计指南。详细描述了CLIPS语言的使用方法及程序设计技巧。北京信息工程学院计算中心持有CLIPS 4.0版本的软件。

本书编译工作中得到了康越同志及楚毓苏同学的帮助,谨此表示谢意。

由于水平有限,错误之处,恳请读者批评指正。

编译者

1989. 4

愿得到CLIPS 4.0版本软件
盘片的读者请同北京信息工程
学院计算中心联系,(或直接与希
望公司联系,电话2562329)

地址:北京德胜门外苇子坑

电话:449942

449906

邮编:100101

目 录

第一部分: CLIPS 用户指南

第一章 事件	(1)
1.1 序言	(1)
1.2 程序的进入和结束	(1)
1.3 建立事件表	(1)
1.4 事件列表	(2)
1.5 基本元素	(2)
1.6 删除空格	(3)
1.7 删除事件	(3)
1.8 清除事件	(4)
1.9 习题	(4)
第二章 规则	(6)
2.1 构造良好的规则	(6)
2.2 规则点燃条件 1	(7)
2.3 规则点燃条件 2	(7)
2.4 规则点燃条件 3	(8)
2.5 显示规则	(8)
2.6 装入规则	(9)
2.7 注释	(9)
2.8 书写程序	(10)
2.9 习题	(10)
第三章 编程	(12)
3.1 停止和前进	(12)
3.2 实际情况	(12)
3.3 步行	(13)
3.4 定义事件语句	(13)
3.5 有选择的删除	(14)
3.6 优先级	(14)
3.7 顺序	(15)
3.8 Watch命令	(17)
3.9 习题	(17)
第四章 变量	(19)
4.1 建立变量	(19)
4.2 变量赋值	(19)
4.3 Printout 变量	(20)

4.4	删除变量	(20)
4.5	条件元素与变量对应关系	(21)
4.6	assert与retract的使用	(21)
4.7	通配符的使用	(22)
4.8	多域通配符的使用	(23)
4.9	习题	(24)
第五章	计算	(26)
5.1	软件包	(26)
5.2	数字的精度	(26)
5.3	基本运算	(28)
5.4	扩展操作数	(29)
5.5	混合运算	(30)
5.6	赋值并打印	(31)
5.7	平方根函数	(31)
5.8	数学函数	(32)
5.9	习题	(34)
第六章	控制	(36)
6.1	读函数	(36)
6.2	读入插入	(36)
6.3	用Test函数控制循环	(38)
6.4	用Test检查逻辑关系	(40)
6.5	程序效率	(41)
6.6	习题	(43)
第七章	逻辑操作	(44)
7.1	或 (OR) 操作	(44)
7.2	限制 1	(49)
7.3	限制 2	(49)
7.4	限制 3	(49)
7.5	习题	(50)
第八章	强功能的句型	(53)
8.1	谓词函数	(53)
8.2	真假	(55)
8.3	检查数字的函数	(56)
8.4	改进CLIPS程序设计	(56)
8.5	习题	(58)
第九章	逆向与正向程序设计	(59)
9.1	正向与逆向连接	(59)
9.2	正向程序设计	(59)
9.3	逆向程序设计	(63)

9.4	注释	(68)
9.5	诊断功能	(70)
9.6	习题	(72)
第十章	扩充句型	(73)
10.1	IF THEN ELSE句型	(73)
10.2	While 句型	(74)
10.3	Not—mean—While句型	(75)
10.4	排序规则 (sort)	(77)
10.5	排序函数中的标识符	(78)
10.6	While钳套	(80)
10.7	否定	(81)
10.8	习题	(82)
第十一章	文件传送	(85)
11.1	打开文件	(85)
11.2	关闭文件	(86)
11.3	EXIT的使用	(87)
11.4	读一个文件数据	(87)
11.5	读文件数据	(87)
11.6	向文件写入数据	(88)
11.7	标准输入输出设备	(89)
第十二章	CLIPS的扩展	(90)
12.1	随机数问题	(90)
12.2	连接	(92)
12.3	测试	(93)

第二部分 CLIPS 基本程序设计指南

第一章	引言	(94)
第二章	CLIPS 概述	(94)
2.1	规则 (Rules)	(94)
2.2	事件 (Facts)	(94)
2.3	基本执行过程	(95)
2.4	参考手册句法	(95)
2.5	定义结构	(96)
2.6	注释CLIPS规则	(97)
2.7	与其它语言的集成	(97)
第三章	LHS 句法——条件	(98)
3.1	文字句型	(98)
3.2	通配符——单域及多域	(98)

3.3 变量——单域及多域	(99)
3.4 限定域	(101)
3.5 用测试限制变量	(104)
3.6 限制句型	(106)
3.7 句型连接	(109)
3.8 说明规则特性	(109)
第四章 RHS 句法——动作	(110)
4.1 基本动作	(110)
4.2 多域函数	(113)
4.3 CLIPS I/O 系统	(114)
4.4 算术函数	(119)
4.5 其它函数	(121)
第五章 CLIPS 的使用	(125)
5.1 环境命令	(125)
5.2 系统状态命令	(125)
5.3 查错命令	(126)
5.4 存储器管理命令	(126)
5.5 其它命令	(126)
附录A 由CLIPS提供的定义函数	(127)
附录B CLIPS的安装	(128)
附录C 3.0版本与4.0版本的差别	(129)
附录D 术语	(130)
附录E 维护信息	(131)
附录F CLIPS错误信息	(131)

第三部分 CLIPS 高级程序设计指南

第一章 引言	(134)
第二章 CLIPS 与外部函数集成	(134)
2.1 用户定义外部函数的说明	(134)
2.2 变量从 CLIPS 传送到外部函数	(135)
2.3 数据从外部函数传送到CLIPS	(136)
2.4 高级接口函数	(136)
第三章 CLIPS 集成	(140)
3.1 环境函数	(140)
3.2 系统状态函数	(141)
3.3 查错函数	(142)
3.4 其它函数	(142)
第四章 CLIPS 与除C语言外的其他语言组合	(143)

4.1	引言.....	(143)
4.2	嵌入CLIPS——利用一个外部主程序	(144)
4.3	将事件加入到CLIPS中	(146)
4.4	从CLIPS调用一个子程序	(146)
4.5	从CLIPS传送参数到外部函数	(147)
4.6	编译和连接.....	(150)
第五章	I/O 子程序系统	(152)
5.1	引言.....	(152)
5.2	逻辑名.....	(152)
5.3	子程序.....	(153)
5.4	子程序优先级.....	(154)
5.5	内部I/O函数	(154)
5.6	子程序处理函数.....	(156)
第六章	性能分析	(157)
6.1	结构顺序.....	(157)
6.2	事件与事件表比较.....	(158)
第七章	存贮管理	(159)
7.1	CLIPS 如何使用存储器.....	(159)
7.2	标准存储器函数.....	(160)
第八章	编辑CLIPS.....	(161)
附录A	语言集成清单.....	(162)
附录B	I/O子程序例子	(173)

第一部分 CLIPS 用户指南

第一章 事件

这一章将学习专家系统的基本概念，并学习CLIPS中有关事件的插入和删除。

1.1 序 言

CLIPS是一种为专家系统书写应用而设计的一种计算机语言，它的基本元素是：

1. 事件表：记录所有的数据
2. 工作存储器：包括所有的规则和结果
3. 推理系统：控制所有执行过程

用CLIPS书写的程序由规则和事件组成。推理系统决定哪一条规则应当执行。在后面几章中我们将详细讨论CLIPS的各个部分。

现在，看一下CLIPS中的事件。事件是非常重要的。因为没有事件，CLIPS程序是不能执行的。这正是CLIPS与其它诸如Pascal, Ada, BASIC, FORTRAN和C等过程设计语言不同的一个方面。在过程设计语言中，程序的执行不需要事件。这就是说，在这些语言中，程序的描述已足以使得程序得以执行。然而，在CLIPS中，需事件来导致规则执行。因此规则起着十分重要的作用。

1.2 程序的进入和结束

CLIPS程序开始，需键入你所使用系统的适当的运行命令，你应当能看到如下的CLIPS的提示符：

```
CLIPS>
```

在这个状态下，可以开始打入CLIPS的命令。

通常的退出CLIPS的方式是使用'exit'命令。只需打入(exit)并打回车键来回答CLIPS提示符。

1.3 建立事件表

同其它的程序语言一样，CLIPS能识别某些关键字，例如，如果你要在事件表中加入数据，可使用插入命令。

作为插入的一个例子，只要在CLIPS提示符之后接着打入下述命令：

```
CLIPS> (assert (duck))
```

在每一行程序之后，总是打入回车键来标志CLIPS一行结束。上述的命令是将事件“duck”插入存放事件的寄存器中。

注意，插入命令和事件duck都需要加上括号。如同许多其它的专家系统语言一样，CLIPS有类似LISP的句法，即要求所有的语句加上括号。虽然CLIPS不是用

LISP写成的，但是LISP的风格对CLIPS的发展有一定影响。这也即是退出命令需加括号的原因。

1.4 事件列表

假设你要看一下事件表中的内容，只需要使用事件命令。在CLIPS提示符之后打入(facts)，CLIPS就显示出事件存储器中的一张事件表。切记在命令语句中要加入括号，否则CLIPS不予接受，(facts)命令的结果应该是

```
f-1 (duck)
```

f-1是CLIPS插入的事件标识符，所有插入事件表中的事件都被赋予一个唯一的事件标识符，用字母f开头，后面接一个称为事件索引的整数。

如果你试图再将duck插入事件表会发生什么情况呢？让我们试试看，键入一个新的duck，然后发一个(facts)命令，你将看到先前的“f-1 (duck)”，这表明CLIPS将不接受事件的重复键入。

当然，你可以插入其它不同的事件，例如，打入事件(quack)，然后发一个(facts)命令。你将看到

```
f-1 (duck)
```

```
f-2 (quack)
```

如同你看到的，事件(quack)已在事件表中。重要的一点是要认识到事件表中的事件标识符号并不是严格连序的，在后面的几节中你将看到，事件可以加入，也可以删除。由于事件从事件表中删除，被删除的事件的标识符将在事件表中消失。所以，事件标识符随着程序的执行并不总是连序的。

1.5 基本元素

一个事件，例如(duck)或(quack)，是由一个单元元素组成的。在类似LISP的语言中，元素的含义是不能够再分解的最小的单位。在诸如Pascal, Ada, C, BASIC和其它语言中，象(duck)这样的事件可被分解为单个字。然而，在CLIPS中，因为它是一个元素，它不能进一步分解。

一个符号元素是一个用字母开头的，后跟字母、数字或破折号的符号串。例如，这个句子中的每个单词都是元素。另一种元素是字符串，字符串元素必须以双引号开始，并以双引号结束。引号是元素的一部分。在双引号之间可以有零个或多个字。第三种元素是数值元素，它表示一个浮点数。在CLIPS中所有的数值都被看作浮点数。一个字符串可以是一个符号元素或一个字符串元素。

一个事件由用左、右圆括号括起来的一个或多个元素组成。多个元素之间必须用一个或多个空格分隔开。例如，输入下述的命令：

```
(assert(The duck said "Quack."))
```

并发一个(facts)命令。你可以看到一组元素(The duck said "Quack")。由于(The duck said "Quack")只赋予一个标识符，所以被认为是一个事件。

如果你在元素之间留多了空格将会怎样呢？插入

```
( duck quack quack )
```

然后检查事件表。你将看到

```
(duck quack quack)
```

这是因为CLIPS已经将元素间多余的空格消去了。

还应注意到, CLIPS的事件的元素中可有大写和小写。这就是说, The的T, Quack的Q是大写。由于CLIPS能区分大小写, 因此大、小写是能识别的。插入事件(duck)和(Duck), 然后发一个(facts)命令。你将看到, 由于CLIPS识别大、小写, 因此允许插入(duck)和(Duck)。

1.6 删除空格

由于空格用来分隔多个元素, 因而多个空格不能包含在事件中。例如, 插入事件(duck), (duck), (duck)和(duck), 因为CLIPS不把空格作为元素的一部份, 所以认为这些事件都与(duck)等价, 因此你将只看到(duck)。

如果你要在事件中包含空格, 你必须使用双引号。例如, 插入

```
( "duck" )
```

```
( "duck " )
```

```
( " duck" )
```

```
( " duck " )
```

然后用(facts)命令查看事件表, 你将看到这些事件都被插入到事件表中。

如果你要在事件中包含双引号本身将怎么做? 首先, 你可能想到用双引号括起引号。让我们试一下看。插入事件

```
( " "duck" " )
```

然后查看事件表。你将看到一个事件

```
( " " duck " " )
```

这个事件与要插入的类似, 实际上完全不同。虽然事件中的确出现双引号, 但它已不在duck周围。它暗示CLIPS在duck前后插入空格。应注意到, 被插入事件duck周围无空格。因为CLIPS认为你插入三个元素

```
" "
```

```
duck
```

```
" "
```

所以CLIPS在duck周围加入空格, 并用空格分隔三个元素。

在事件中加入双引号的正确方法是用\。例如, 插入事件

```
( "\ " duck "\ " )
```

然后查看事件表。于是你将看到正确的事件

```
( " "duck" " )
```

CLIPS没有加入空格。

1.7 删除事件

知道如何将事件插入事件表中, 现在学习如何删除事件。从事件表中去掉一个事件叫删除, 用retract命令来完成。要删除一个事件, 必须确切地给出事件的索引作为删除的依据。例如, 有事件表

```
f-1 ( duck )
```

```
f-3 ( quack ).
```

```
f-4 ( The duck said "Quack" )
```

要删除 (quack), 打入命令

```
(retract 3)
```

然后用(facts)命令检查事件表。你会看到 (quack) 被删掉了。要删除 (duck), 打入

```
(retract 1)
```

要删除 (The duck said "Quack"), 打入

```
(retract 4)
```

注意, 要删除事件, 必须确切给出事件索引, 并不是给出事件在表中的位置。你可能会犯这样的错误, 即用命令 (retract 1) 来删除事件表中的第一个事件。只有当第一个事件的索引是 1 的时候, 这个命令才会正确执行。

如果你试图删除一个不存在的事件将会如何? 第二次打入 (retract 4) 命令时, CLIPS 将给出出错信息:

```
fact 4 does not exist
```

这个道理就是, 你没有付出任何东西, 也不能得到任何东西。

1.8 清除事件

如果你要删除特定的事件, 用 (retract) 命令是很方便的。然而, 你要删除事件表中所有的事件就要重复多次。为了避免多次在每一条命令中打入事件索引, 更有效的方法是使用清除命令。只要打入

```
(clear)
```

所有在事件表中的事件将都被删除。

(clear) 使 CLIPS 复原到初始状态, 如同刚开始时一样。为了解这一点, 插入 (duck), 然后查看事件表。注意, (duck) 的事件标识符为 f-1, 因为 (clear) 命令重建事件标识符。(clear) 命令实际上不仅删除事件, 例如, 它还删除所有的规则, 这将在后面了解到。随着对 CLIPS 的深入了解, 你将认识到 (clear) 命令的更多的作用。

1.9 习 题

下列事件是有效还是无效的? 试打入它们来检查你的结论是否正确。如果是有效事件, 标出每个事件所包含元素的类型。注意, 对于有的例子, CLIPS 将暂停直到你给出必要的字符。例如引号或者括号。如果你不能给出正确的符号, 你就必须用 Control 命令来中止 CLIPS 的执行并重新启动 CLIPS。

```
(t)
```

```
(this)
```

```
(this is a test)
```

```
(but what is th!!!)
```

```
(**!!!)
```

```
(and so he ate 19 hamburgers)
```

```
(1 2 3 4.5 -.0012)
```

```
(A1)
```

```
(start-fact)
```

```
(the---end)
```

(^ &)
(duck")
(" ")
(John loves Mary)
(" ")
(1000 Main street)
(1000 Main st.)
("duck)
()
(100 -25.5)
(1e10 10000)
(())
(
) (
(duck (quack))

第二章 规 则

前一章已经学习了事件。现在将学习专家系统的规则如何利用事件使程序得以执行。

2.1 构造良好的规则

为完成一定的任务，一个专家系统必须有规则以及事件。学习了事件如何插入和删除之后，现在看看规则是如何工作的。

一个规则类似于 Ada, Pascal, FORTRAN, BASIC或C语言中的IF THEN 语句。一个IF THEN规则可用如下自然语言和计算机语言的混合来表达：

IF 某些条件成立

THEN 执行下述的动作

上述语句的另一种术语是伪码，字面的意思是“错误的代码”。更专用的伪码语句是程序员设计和说明规则的良好工具。

在Ada, Pascal, FORTRAN, BASIC和C等计算机语言中，这些语言能识别关键字IF THEN。然而，由于CLIPS有着类似LISP的句法。CLIPS 没有IF THEN关键字。尽管如此，如果你将类似IF THEN的词记在脑子里，那么将规则由自然语言译成CLIPS并不难。随着你对CLIPS的运用不断加深，你将会感到写CLIPS规则变得容易了。

规则输入系统可以直接从键盘输入，也可从原先建立的CLIPS 文件装入系统。可以用任何字处理器输入规则，只要在源程序中不包含格式化字符就可以。而这种格式化字符CLIPS是不接受的，所以在用字处理器输入规则时，一定要使字处理器处于 non-document模式。

首先，了解CLIPS中如何直接键入规则。这个规则的伪指令是

IF there is a duck THEN make a quack

下面是用CLIPS 句法表示的规则。唯一增加的特征是用双引号括起的项，它是关于规则的任选描述。为了打入CLIPS规则，只需在CLIPS提示符下打入，然后按回车键。

(defrule duck-quack "The First Rule" (duck) ⇒ (assert(quack)))

如果你如上面那样正确的打入规则，你可以看到CLIPS提示符再一次出现。否则，你将看到错误信息。如果看到错误信息，很可能是关键字拼写错或漏掉了括号。注意，一个语句中的左右括号总是应该相匹配。

下面是同样一个规则，在规则下边用数字表示规则各部分的说明。

```
(defrule duck-quack "The First Rule" (duck) ⇒ (assert(quack)))  
|<2>|< 3 >|< 4 >|<5>|<6>|< 7 >|  
|< 1 >|
```

一个规则包括下述几个部分：

1. 整个规则必须用括号括起来。
2. 规则必须以关键字defrule开始。

3. `defrule`之后必须是规则名。名字可以是任何有效的符号元素。如果你打入一个规则名和一个已有的规则名相同,那么新规则将代替旧规则,在这个规则中,规则名是`duck-quack`。

4. 名字之后是用双引号括起的任选的注释。注释通常用来描述规则的用途或程序员希望标注的其它信息。

5. 在注释之后是一个或多个条件元素或结构。每个条件元素包含一个或多个域。在`duck-quack`规则中,条件元素是`( duck )`且有一个域`duck`。一个有两个域的条件元素的例子是`( duck quack )`。

CLIPS设法根据事件表中的事件来满足规则中的条件元素。如果一个规则中的所有条件元素都和事件相匹配,那么规则将被激活,并被放入记事册中。这个记事册是一个被激活规则的集合。在该记事册中可能没有或多个规则。

6. 规则中跟在条件元素之后的符号“ $\Rightarrow$ ”叫箭头。箭头是一个表示IF THEN规则中THEN部分的符号。

7. 规则的最后部分是当规则被激活后将要执行的一系列动作。规则点燃意味着CLIPS从记事册中选择某一规则去执行。当记事册中有多个规则时,CLIPS将点燃具有最高优先级的规则。在上述的例子中,执行的动作是插入事件`( quack )`。

箭头左边的部分叫“左部”(LHS),箭头右边的部分叫“右部”(RHS)。

2.2 规则点燃条件 1

一个规则在记事册中的所以规则中有最高优先级时被点燃。如果在记事册中只有一个规则,它将被点燃。在规则`duck-quack`中,它的条件元素被事件`( duck )`满足,因此,规则`duck-quack`被点燃。

要使程序运行,只需打入运行命令。即在CLIPS提示符下打入`( RUN )`并按回车键。

没有执行任何操作。

此时你可能奇怪,不过CLIPS一点也没弄错。CLIPS没有点燃规则的原因与CLIPS的设计方法有关。CLIPS的设计是这样的,即CLIPS只能识别在规则之后输入的事件。在前面,在`( clear )`之后首先打入的事件`( duck )`。然后输入的`duck-quack`规则。由于`( duck )`在规则`duck-quack`之前输入,规则不能识别事件,因此规则没有放入作为点燃规则的记事册中。由于`duck-quack`没有放入记事册中,所以它不能被点燃。

可以用`agenda`命令检查记事册中的内容。在CLIPS提示符下,打入`( agenda )`并按回车键。在此情况下,由于记事册中没有任何内容,所以不打印任何东西,CLIPS回到提示符。

2.3 规则点燃条件 2

现在走入正轨将此规则点燃。为了点燃规则,我们必须在规则之后插入事件`( duck )`。粗略地想,似乎只要插入`( duck )`。但是,如果试一下再按`( run )`,规则还是没有被点燃。这也许有点另人扫兴。然而,做以前要冷静考虑。

由于这里已经有了一个`( duck )`在事件表中,打入新的`( duck )`不起作用。因为已经有了一个`( duck )`,CLIPS将不接受这个新的`( duck )`。至此,解决办法是什么呢?

方法是删除前一个 (duck)，然后再插入新的 duck。做完这一切之后，再发一个 (agenda) 命令，你将看到：

```
0 duck—quack f—1.
```

0 表示这个规则在记事册中的优先级，有关优先级将在下一章学习。“0”之后跟着规则的名字以及它与相匹配的事件标识符。在这种情况下，只有一个事件标识符 f—1。

此时，打入 (run) 命令，你将看到规则最终被点燃。如果你检查事件表，你可看到最后有一个 quack。

2.4 规则点燃条件 3

此时可能出现一个有趣的问题。如果你再次运行会怎样？这里有一个规则，并有一个满足规则条件的事件，所以规则应再次被点燃。打入 (run) 命令，你将看到没发生任何变化。如果你查看记事册，你将看到由于在记事册中无任何内容，因此规则不点燃。

规则不再点燃是由于 CLIPS 的设计方法所造成的。CLIPS 被设计成具有神经细胞特征的。在神经细胞传输一个神经脉冲之后（点燃），在一段时间内不再点燃，这种现象叫做 refraction，这在专家系统中是非常重要的。

没有 refraction，专家系统将陷入无限循环，这就是说，规则一旦点燃，就不断地一次又一次在同样的事件上点燃。实际上，使事物点燃的能量最终将消失。例如， duck 最终将游走或飞到南方去过冬。然而，在计算机世界，一但一个事件进入事件表，它将一直存在直到被删除或关掉电源。

怎样将规则再次点燃？只需将事件删除再插入。这就好象是旧 duck 飞走了，新的 duck 来使规则再一次被点燃。原则上，CLIPS 能记住使规则被激活的事件标识符，并且不在用同样的事件标识符来激活规则。当然，一个规则可用同样的事件标识符多次放入记事册中。然而，被放入记事册的规则与被点燃的不同。只有那些被点燃的规则可被 refraction，并不是在记事册中的规则。

2.5 显示规则

当在 CLIPS 执行过程中，通常要看看规则。有一个显示规则的命令叫 pprule，表示完美地打印出规则。要看一个规则，指定规则的名字作为自变量。例如，在 CLIPS 提示符下打入下述内容并按回车键：

```
(pprule duck—quack)
```

你将看到：

```
(defrule duck—quack "The First Rule"
  (duck)
  =>
  (assert(quack)))
```

为便于阅读，CLIPS 将规则的不同部分写在不同的行。如果有足够的空间，你可将规则打入在一行中，你也可以把规则打在多个行中。从现在起，为了阅读方便，我们将规则显示在多个行中。

使用字处理机可方便地打出程序，因为所有的编辑命令有效。箭头前的部分仍然被