国际电信联盟

CCITT

第七次全体会议

CCITT 高 级 语 言

CHILL 概　　述

人民邮电出版社

INTRODUCTION
TO
CHILL
THE CCITT HIGH LEVEL LANGUAGE

Kristen Rekdal

ITU, GENEVA 1980

内 容 提 要

本书是CCITT高级语言（CHILL）的非形式化介绍，用比较通俗的语言说明了CHILL语言的描述方法和程序结构。其重点放在如何编写易读并可靠的程序上，并试图给出结构的典型用法。在写法上力图使读者能尽快掌握具体编写简单工作程序的方法和要求。为供参考，每节包含一个用形式文体写的语法摘要。书末附有汉英词汇索引。本书通俗易懂，例子较多，是一本较好的CHILL语言指导性读物。

本书可供从事程控电话交换系统设计的工程技术人员，计算机软件的工程技术人员，以及高等院校有关专业的师生学习参考。

CHILL概 述

〔挪威〕K·雷克达尔 著

吕 常 义 译

人民邮电出版社出版

北京东长安街27号

顺义兴华印刷厂印刷

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　　1985年6月第一版

印张：5　　页数：40　　1985年6月北京第一次印刷

字数：114千字　　印数：1—6,500 册

统一书号：15045·总3062-有5423

定价：1.25元

目　录

1. 引　言

本书是CHILL（CCITT高级语言）的非形式介绍。CHILL语言是在CCITT建议Z.200中定义的。

1.1. 用于SPC程序设计的高级语言

随着SPC（存储程序控制）电话交换机使用的不断增加，程序设计正在成为电话技术的一个重要组成部分。SPC的软件系统已变得非常庞大而复杂。这样，使用合适的程序设计工具就显得非常重要了。

CCITT高级程序设计语言CHILL就是这样一种工具。它被设计成适合编制SPC电话交换机用的各种程序。这种应用包括了范围广泛的计算机程序设计。这样，可以认为，该语言对于其它一个应用来说一般也是够用的。

1.2. 有关说明

本书是CCITT高级语言用法的基本介绍。假定读者已具有一定程序设计的实际经验，并具有懂得另一种高级语言的有利条件。本书重点在于语言本身，而不在于编译程序，或其它有关的程序开发工具的使用。本书也不是一般计算机程序设计的介绍。

本书重点放在如何编写易读的和可靠的程序上，并试图给出结构的典型用法，而不是所有可能的用法。为了全面掌握CHILL语言的正确用法，读者应参考该语言定义和有关该语言特定实现的有关资料。

本书在写法上力图使读者能尽快掌握具体编写简单工作程序的方法和要求。首先说明该语言一些简单、基本的特点，更多的特点和细节在以后说明。

本书的立意是从前到后按顺序阅读。它不是一本参考手册。然而为了便于前后引用，在书末给出了一个索引。

由于CHILL语言的特点，本书可以完全按顺序编写而不会有过多的重复。因而，在本书中将会发现偶然使用了未经说明的内容。这种情况仅在没有必要详细了解CHILL语言某些特点的地方才会出现。如果你想了解更多的东西，请利用书后的索引去查找有关这些特点的更详细说明。

CHILL是一种丰富的语言。甚至可以有几种方法达到相同的目的，在类似这样的介绍中，哪些特点必须介绍，哪些细节可以跳过，是有选择余地的。显然，这种选择反映了作者的爱好和意图。可以设想，本书中的这些部分，尤其是例子，可以被反映其它爱好和意图的例子代替。尤其是在某些情况下，CHILL允许用多种方法编写具有相同含义的结构。为易于理解，本书只采用了一种文体。为了完整起见，在个别章节给出了等价的文体。此外，未涉及的有关CHILL语言的其它一些方面的问题将在有关章节简要地叙述。为了充分理解这

些问题，请读者参考建议Z.200。

1.3. 本语言的描述方法

可以认为，语言描述由两部分组成。第一部分是语言的**语法**描述，即在该语言中哪些字符序列形成合法结构。第二部分是语义描述，即这些结构的含义。

本书中，语法和语义都是采用非形式语言和一些例子来描述的。多数例子在左边有一个行号。这只是为易于引用有关说明。行号不是CHILL程序文本的成分。为供参考，每节包含一个语法摘要，它是作为一组语法规则用更简洁更形式化的形式书写的。所用的这些规则类似于建议Z.200的那些规则。

语法类别是用括在尖括号中的一个或多个汉语词表示的，如〈行模式〉。在有关的汉语散文文本中，使用相同的表示法而不加尖括号。CHILL源文本中出现的字符和字符序列是按其出现顺序书写的，如MODULE。

为了表示一个语法元素紧接另一个语法元素（连接），它们按从左到右的顺序书写。

例：

〈赋值语句〉::=〈单元〉:=〈表达式〉;

这个规则的意思是，赋值语句的构成是，一个单元后跟赋值号“:=”，再跟一个表达式，并用分号“;”结束。

语法元素可用花括号进行分组。一个组重复零次或多次，用花括号后跟一个星号表示。这样，{A}*代表包括空在内的A的任意序列。重复一次或多次用“+”表示。替换用“竖线”表示。任选的语法元素放在方括号中。

例：

〈名字〉::=
〈字母〉{〈字母〉|〈数字〉|_}*

这个规则的意思是，名字是由一个字母，后面可能跟随字母、数字或下线符号的任意序列组成。

当连接和替换在同一规则中出现时，连接要在替换之前完成。

2. 语言基础

CHILL 程序逻辑上由三部分组成。即要执行的**动作描述**，由这些动作所操作的**对象描述**，以及**程序结构**描述。

2.1.表达式和赋值语句

表达式用来计算一个值。它是由用**运算符**连结起来的一系列**操作数**组成的。运算符是**单目的**或**双目**的。双目运算符带两个操作数，一个在左边，一个在右边。单目运算符只带右边一个操作数。表达式从左到右求值，但要考虑到运算符的优先级和圆括号。

例：

```
A+B*3   等价于  A+(B*3)
-A*B+C  等价于  ((-A)*B)+C
```

由表达式给出或产生的值可以通过赋值语句存放在单元里。赋值号“：=”右边的值存放在赋值号左边名字代表的单元中。

例：

```
1. I:=1;
2. I:=-2;
3. I:=(I+2)*J;
```

第一行读作：“把1赋给单元I”，或者“I变为1”。

多重赋值是通过在赋值号左边给出一个用逗号隔开的单元表表示的。

例：

```
4. I,J,K:=0;   /*I,J,和K每个都取值为0*/
```

赋值语句用分号结束。

2.1.1.摘要

赋值语句由两部分组成，赋值号左边是一个或多个单元，赋值号右边是一个表达式。

表达式由一个或多个用双目运算符连结起来的操作数组成。操作数是值、单元、单目运算的结果、内部子程序调用或括在圆括号里的表达式。内部子程序调用由子程序名跟以圆括号中的表达式表组成。

利用语法记法，上述情况可表示为：

```
〈赋值语句〉::=
    〈单元〉{,〈单元〉}*:=〈表达式〉;
〈表达式〉::=
    〈操作数〉{〈双目运算符〉〈操作数〉}*
〈操作数〉::=
    〈值〉
```

〈单元〉
| 〈单目运算符〉〈操作数〉
| 〈内部子程序调用〉
| (〈表达式〉)

〈值〉::=
〈字面值〉

〈单元〉::=
〈名字〉

〈内部子程序调用〉::=
〈子程序名〉(〈表达式〉{,〈表达式〉}*)

注意，值和单元的概念，以后充分说明。

2.2.数据对象和模式

数据对象是程序动作所操作的实体。**数据对象**是**值**或**单元**。可把单元想象成能存放值的抽象容器。单元不应与物理存储单位混淆起来。

数据对象有一个加到其上的**模式**。模式是一组数据对象所共有的性质。对象模式定义了对象值的集合（如果对象是一个单元，还可以假定对象值的存取方式），以及在这些值上的有效操作。在本语言中可使用一些标准模式。几种标准模式表示是 INT（整数）， BOOL（布尔）和CHAR（字符）其它模式以后介绍。

单元总有一个加在其上的唯一模式。然而对值来说，附加一个唯一模式并非总是可能的。在这种情况下，值有一个附加的**类别**（class）。类别是由能够包含该值的那些单元的模式构造出来的一组模式。

在程序中，数据对象是作为**表记**（denota-tion）表示的。一个**表记**是，例如基本**字面值**采用的一个**名字**或特定记法。单元的**表记**称为**单元表记**，或简称**单元**。新单元是由**单元说明**引进的，该说明给新单元的值分配一个单元，并给其附加一个名字和模式。只有单元能被赋值。值是由**值表记**（或简称为值）表示的，例如，**字面值**。字面值表示一个特定的值。一般说来，对某类的每个值都有一个字面值。

例：

```
5.DCL 1 INT;
6.I:=1+7;
```

第 5 行是一个单元说明语句。I 是一个单元， INT 是一种模式。在第 6 行中，7 是一个字面值。

2.2.1.离散模式

离散模式是附加到具有有限个有序的孤立值对象上的模式，以下是几种标准的离散模式

INT —这个对象是实现所定义范围的整数。

（例如，对十六位计算机来说，它是-32__768到32__767。）

最简单形式的整数字面值由一个或多个十进制数字组成。

下线符仅用于将数字分组，以增加可读性。

这里至少要有一个有意义的数字。它可以任选地前置以D′。

例：

```
0
D'21
1_000_000
```

二进制，八进制，十六进制的整数字面值也可提供。

例：

```
B'101_111
O'7034
H'FA03
```

BOOL —布尔或逻辑模式。字面值是：

```
FALSE
TRUE
```

CHAR —CCITT 5 号字母表的一个字符的模式。一个字符字面值是通过把这个字符括在撇号中表示的：

```
'A'
'B'
' '
'7'
```

撇号字符字面值书写为''''。第一个和最后一个撇号把中间两个撇号括起来表示一个撇号字符。

无图形表示的字符，可以写成十六进制数字对。

例：

```
C'0A'
```

2.2.2.单元说明

单元说明分配一个可存放值的单元，并对它附加一个名字和模式。在其最简单形式中，说明由字DCL后跟一个或多个用逗号隔开的**新单元名字**，再跟以模式，最后以分号结束。

名字必须以字母开头，后面可跟一个字母和/或数字和/或下线字符序列。在名字中，下线符是个有意义的字符。因此，名字OFFHOOK与名字OFF_HOOK是不同的。

为了提高可读性，并有利于文件编制，名字应该是该对象的意义和描述。

例：

```
7.  DCL  NO_OF_LINES   INT;
8.  DCL  OFF_HOOK  BUSY  BOOL,
9.       ALPHA  BETA      CHAR;
```

在第 7 行中，说明NO_OF_LINES为整数模式单元的名字。在第 8 行中，说明OFF_HOOK和BUSY为布尔模式单元的名字，而在第 9 行中，说明ALPHA和BETA为字符模式的名字。

2.2.3.赋初值

在说明单元时，可以赋给出它们初值。

例：

```
10. DCL  NO_OF_LINES  INT:=10_000;
```

这与下列语句具有相同作用：

```
11. DCL  NO_OF_LINES  INT;
12.          NO_OF_LINES =10_000;
```

相同模式的一系列单元可以赋相同的初值：

```
13. DCL X,Y,Z BOOL:=FALSE;
```

这与下列语句具有相同的作用：

```
14. DCL X,Y,Z BOOL;
15.       X,Y,Z:=FALSE;
```

只要可能就要在说明内赋初值，而不要等到最后再赋初值。这使程序易读并少出错，因为保证单元有确定的初值。

2.2.4.只读单元

程序员有时知道，只给单元赋值一次，其后不得修改。在程序中，这可以通过在模式名称前置以字READ来表示。

```
16.DCL  NO_OF_LINES READ INT:=10_000;
```

现在说明的NO_OF_LINES是一个只读整数单元。把它赋初值10_000，以后对它的任何赋值都将产生一个错误信息。只读对象总是要赋初值的。

2.2.5.摘要

```
〈名字〉::=
〈字母〉{〈字母〉|数字_}●
〈离散模式〉::=
   [READ]{INT|BOOL|CHAR}
〈整数字面值〉::=
    |〈十进制整数字面值〉
    〈二进制整数字面值〉
    |〈八进制整数字面值〉
    〈十六进制整数字面值〉
〈十进制整数字面值〉::=
    [D']{〈数字〉|_}+
〈二进制整数字面值〉::=
    B'{0|1|_}+
〈八进制整数字面值〉::=
    O'{0|1|2|3|4|5|6|7}+
〈十六进制整数字面值〉::=
    H'{〈十六进制数字〉|_}+
〈十六进制数字〉::=
    〈数字〉|A|B|C|D|E|F
〈数字〉::=
    0|1|2|3|4|5|6|7|8|9
〈布尔字面值〉::=
    FALSE|TRUE
〈字符字面值〉::=
    '〈CCITT字母5号字符〉'
〈字母〉::=
  A|B|C|D|E|F|G|H|I|J|K|L|M|N|O|P|Q|R|S|T|U|V|W|X|Y|Z
```

```
〈说明语句〉::=
  DCL〈说明〉{,〈说明〉}*;
〈说明〉::=
  〈名字表〉〈模式〉[〈赋初值〉]
〈赋初值〉::=
  :=〈表达式〉
```

2.3.语句和程序

一个简单的CHILL程序的组成是，字MODULE后跟一个语句序列，再跟以END；。

模块是CHILL语言中几种括号结构之一。字MODULE构成开括号，而字END构成闭括号。所有括号结构都可以附加一个名字，名字后跟一个冒号，冒号直接加在开括号之前。有时为了引用这个结构（例如，出口语句），采用这个名字是必要的。为了使开括号和闭括号易于配对，可以在对应的闭括号之后重复这个名字。尤其是在带有嵌套结构的情况，当开括号和闭括号相距较远时，最好采用这种名字。这种作法可以大大提高程序的可读性，并有利于文件编制。语句用“；”结束。

语句分类如下：

——定义语句

——说明语句

——动作语句

定义语句和说明语句给程序引进新的对象。到现在为止，我们仅遇到了建立新单元的说明语句。定义语句不建立单元。定义语句将在以后论述。

关于动作语句，至今我们已讨论了赋值语句。其它动作语句的例子是条件语句和循环语句，将在稍后论述。

定义语句和说明语句必须放在动作语句之前。除此之外，关于这些语句，没有特殊的顺序限制。

下面是一个简单CHILL程序的例子：

```
1.  COMPUTE_ABSOLUTE_VALUE:
2.  MODULE
3.      DCL I INT:=1,
4.              J INT:=I*2;
5.      I:=J*3;
6.      J:=I/4;
7.      I:=ABS(J-I);
8.  END COMPUTE_ABSOLUTE_VALUE;
```

第3行，说明一个整数单元I，并赋给初值1。第4行，继续进行整数说明，说明整数单元J，并赋给初值I*2。I的当前值是1，所以I*2是2。

在第5行，J的值是2，所以J*3的值是6，把它赋给I。在第6行，运算I/4现在产生值1（整除截取商的整数部分）把它赋给J。

第7行，I是6，J是1，J-I是-5。ABS是一个函数或内部子程序。ABS取其变元的绝对值。于是，ABS(J-I)是5，把它赋给I。

2.3.1.关于空格、换行和注解的注释

CHILL程序的基本语法实体称为**词法单位**。词法单位分为三组：

—名字，例如I,X,CALL_COUNT,END

—基本整数字面值，字符字面值和串字面值，例如125,'A','ABC'

—专用符号,例如,";","+",":=","/*"

词法单位以在本单位内不允许出现的第一个字符结束。对多数词法单位来说，空格字符作为结束符是方便的，因为在任何词法单位内都不能出现空格字符。这条规则的例外情况是，在字符字面值和字符串字面值内，空格是该字面值的一部分。

为避免混乱，有时在各词法单位之间至少要求一个空格。例如，名字和名字之间，以及名字和基本字面值之间就是这种情况，例如，MODULEDCL被看作一个名字，而不同于MODULE DCL，后者是模块的开始，它带有一个说明语句。

在各词法单位之间可以出现任意多的空格。换行和注解象（一个或多个）空格一样具有分隔作用。它们从不出现在词法单位内。注解是包含在"/*"和"*/"之内、可以包含字符和换行的任意序列。

除上述规则之外，CHILL程序可以按十分自由的文体书写，例如，一个语句可以延伸几行。

文章中使用空格、换行和注解对程序的可读性是非常重要的。

例：

```
MODULE AA(I-J) (-I):=-(I+AA (I) (J)); END; /*THIS IS VERY COMPACT*/
MODULE
  AA(I-J)              /*THIS IS THE*/
    (-I):=-(I          /*SAME  CODE*/
+AA(I)(J));            /*VERY EXPANDED*/
       END;
MODULE                 /*BUT THE IMPORTANT*/
  AA(I-J)(-I):=-(I+AA(I)(J));
                       /*THING IS TO MAKE*/
END;                   /*IT READABLE,LIKE*/
                       /*THIS FOR INSTANCE*/
```

2.3.2.输入和输出

在CHILL中，未定义标准的输入输出子程序。这种子程序可以用CHILL本身来写。然而，在本书中，为了编写例子，假定存在一组输入/输出子程序。

表2-1 输入/输出子程序

子程序名	参数类别	结果类别	动作
ININT	无	INT	读一个整数值
INBOOL	无	BOOL	读一个布尔值
INCHAR	无	CHAR	读一个字符值
OUTINT	INT	无	写一个整数值
OUTBOOL	BOOL	无	写一个布尔值
OUTCHAR	CHAR	无	写一个字符值

2.3.3. 摘要

```
〈程序〉::=
    [〈名字〉:]MODULE
    〈模块体〉
    END[〈名字〉];
〈模块体〉::=
        {〈数据语句〉}●
        {〈动作语句〉}●
〈数据语句〉::=
        〈定义语句〉
      |〈说明语句〉
```

2.4. 运算符和内部子程序

现在我们继续讨论表达式的表示，更加仔细地考察某些重要的运算符和内部子程序。除去操作数应写在内部子程序名之后的圆括号内以外，内部子程序类似于单目运算符。

下面给出几个使用运算符和内部子程序表达式的例子。为简洁起见，只给出表达式。在实际程序中，表达式将作为语句部分出现。

表 2-2 算术运算符

运算符	左操作数类别	右操作数类别	优先级	结果类别	动作
+	INT	INT	4	INT	整数加
−	INT	INT	4	INT	整数减
−		INT	6	INT	整数负
•	INT	INT	5	INT	整数乘
/	INT	INT	5	INT	整数除
REM	INT	INT	5	INT	整数除的余数
MOD	INT	INT	5	INT	模运算

整数除截取商的整数部分，不进行舍入。取余运算符 REM 定义为 X REM Y = X − (X/Y) • Y。X MOD Y 在区间 0 〈 =K 〈 Y 中给出值 K，因此，K = X − N • Y，这里 N 是一个整数，并且 Y > 0。

具有最高优先级值的运算符首先计算。

表 2-3 关系运算符

运算符	左操作数类别	右操作数类别	优先级	结果类别	动作
=	任意	同左	3	BOOL	相等测试
/=	任意	同左	3	BOOL	不等测试
>	离散	同左	3	BOOL	大于测试
>=	离散	同左	3	BOOL	大于或等于测试
<=	离散	同左	3	BOOL	小于或等于测试
<	离散	同左	3	BOOL	小于测试

例：

假定说明：

```
DCL I INT:=1,
    J INT:=2,
    C CHAR:='Z';
```

则下面是一些有效的表达式：

```
I=J                 /*I=2，结果：FALSF      */
C='A'               /*结果：TRUE            */
I+J>=J              /*结果：TRUE            */
I<(J-1)             /*(J-1)是1，结果：FALSE */
0<=(I/2)            /*(I/2)是0，结果：TRUE  */
(I-2)<0=(-J<0) /*(-1<0)=(-2<0)，这是TRUE=TRUE，结果：TRUE */
```

表 2-4 布尔运算符

运 算 符	左操作数类别	右操作数类别	优先级	结果类别	动　　作
AND	BOOL	BOOL	2	BOOL	若左、右操作数都为TRUE，则结果为TRUE
OR	BOOL	BOOL	1	BOOL	若左或右操作数为TRUE，或两个都为TRUE则结果为TRUE
XUR	BOOL	BOOL	1	BOOL	若左或右操作数为TRUE，而另一个为结果为FALSE，则结果为TRUE
NOT		BOOL	6	BOOL	若操作数为FALSE，则结果为TRUE，反之亦然

例：

假定说明：

```
DCL I INT:=1,
    J INT:=2,
    B BOOL:=FALSE;
```

则下面是一些有效的表达式：

```
I<0 OR B          /*FALSE OR FALSE，结果：FALSE      */
I<0 AND J>0       /*FALSE AND TRUE，结果：FALSE      */
I<0 OR  J>0       /*FALSE OR TRUE，结果，TRUE        */
I>0 XOR J>0       /*TRUE XOR TRUE，结果：FALSE       */
NOT(I<0)          /*结果：TRUE                       */
NOT(I<0) AND J>0/*TRUE AND TRUE,结果：TRUE           */
I<0 OR NOT(J>0)/*FALSE OR FALSE/结果：FALSE          */
```

注意，在“NOT(J>0)”中，我们使用了圆括号。括号内的表达式比括号外优先求值。如果我们写“NOT J>0”，则 NOT 的优先级比“>”高，应该先求值。在这种情况，NOT 的操作数应该是 J，而 J 的模式是整数。因为 NOT 运算符没有关于整数的运算，所以将给出错误信息。

经常用运算符修改单元的值，例如，I:=I+1；对这种情况，作为一种简单记法，CH-ILL 提供了**赋值运算符**。

例：

```
I+:=1;    /*等价于*/I:=I+1;
B AND:=I; /*等价于*/B:=B AND I;
```

下列这些双目运算符可以用作赋值运算符：

+，-，*，/，AND，OR，XOR。

表 2-5 内部子程序

子程序名	参数类别	结果类别	动作
ABS	INT	INT	取参数对象的绝对值
PRED	离散	同参数模式	PRED(X),取小于X的最大值
SUCC	离散	同参数模式	SUCC(X),取大于X的最小值
NUM	离散	INT	取变元的数值

例:

假定说明:

```
DCL I INT:=-1,
    C CHAR:='A';
```

则这些是有效的表达式:

```
ABS (I)    /*结果: 1*/
PRED (6)   /结果: 5*/
SUCC (C)   /*结果: 'B'*/
```

2.5. 组合数据对象

可以通过组合较简单的对象来构造**组合数据对象**。

2.5.1. 数组

构造组合数据对象的一种方法是采用**数组**。数组是相同模式对象的组合。每个对象称为**数组的元素**。例如:

```
DCL CALL_COUNT ARRAY(1:5) INT;
```

这个语句说明 CALL_COUNT 是一个数组单元的名字。该数组由 5 个整数元素组成,其下标是 1 至 5。当说明一个数组时,其大小必须已知。也就是,下标集通常由字面值或字面值表达式给出。所说的数组模式是指"ARRAY(1:5) INT"。

CALL_COUNT 的元素可以象任何其它整数单元一样使用。例如:

```
CALL_COUNT(I):=CALL_COUNT(J)+1;
```

CALL_COUNT(I)和 CALL_COUNT(J)是整数模式的数组元素。存取一个数组元素的操作采用所谓**求下标**的方法。在上例中,右边的求值顺序是,首先求 CALL_COUNT(J) 的下标,然后相加。左边求值是求 CALL_COUNT(I) 的下标。然后进行赋值。在 CHILL 中没有规定赋值的那一边首先求值。所以人们不可能按一种特定顺序确切知道正在计算哪一边。在这种情况,该下标可以是在正常值范围内得到一个整数的任何表达式。

数组操作数原则上象任何其它操作数一样使用。对数组操作数可以赋值,也可以对数组进行相等和不等比较。

数组值可以用**数组元组**(array tuples)表示。元组是包含在元组括号内的一个值表,每个数组元素都有一个值。元组括号是(:和:)。可以用方括号[和]代替。

例如,把一个数组元素赋给一个数组单元可以表示为:

```
CALL_COUNT:=(:3,4,0,1,2:);
```

赋值之后,CALL_COUNT(1)有值 3,CALL_COUNT(2)有值 4 等等。

在比较数组时，是逐个元素进行比较的。两个数组操作数长度必须相同。

例：

```
DCL ERROR BOOL:=CALL_COUNT=(:0,0,0,0,0:);
```

这里，ERROR 被说明为一个布尔对象，而且若 CALL_COUNT 的所有元素都为 0，则对它赋初值为 TRUE。否则赋初值为 FALSE。

象 ARRAY(1：5) INT 一样，数组模式完全是一种普通模式，并且凡在产生的结构是有意义的地方都可以使用。例如，我们可以构造数组的数组：

```
ARRAY(1:3) ARRAY(1:5) INT
```

这是三个“ARRAY(1：5)INT”元素的数组，它实际上是一个二维数组。

数组的数组又是一个完全普通的模式，例如，我们可以把它用在说明中：

```
DCL MATRIX ARRAY(1:3) ARRAY(1:5) INT;
```

这说明 MATRIX 是一个 3 组，每组 5 个整数的对象，总计有 15 个整数。

MATRIX(3) 是第 3 组的 5 个整数。MATRIX(3) 的模式是“ARRAY (1：5) INT”我们可以象存取 CALL_COUNT 的元素那样来存取 MATRIX(3)的元素：

```
MATRIX(3)(5)
```

这是 MATRIX 中第 3 组整数的第 5 个元素。这个元素的模式是整数。MATRIX 还可以按一般方法使用。

```
MATRIX(I+1)(I-J):=I+MATRIX(I)(J),
MATRIX(I):=(:3,4,5,1,2:);
MATRIX:=    (:(:5,4,3,4,5:),
             (:-10,-5,0,5,10:),
             (:-8,16,24,32,41:):);
```

通常，元组在左括号前必须有一个模式名。此模式在上下文中的位置是明确的，然而，象上例一样，这个模式可以省略。元组的元素象在本例子中那样不一定是字面值。

元组的另一种形式是带标号数组元组。当表示大的数组值和／或列出全部元素明显地不方便时，采用这种记法是方便的。

例：

```
NEWMODE M=ARRAY(1:10) INT;
DCL I, J INT;
DCL A M;
IF M(:(1):5,(2:4):I,(6:8):I+J,(ELSE):0:)=A
THEN
  ......
FI;
```

在上面的元组中，第 1 个元素有值 5，第2、3和4个元素有 I 的值，第 6、7 和 8 个元素有 I+J 的值，其它所有元素，即第 5，9 和 10 个元素有值 0。

可以取数组的一部分形成一个子数组或数组段（array slice）。

例：

```
CALL_COUNT(2:4)
CALL_COUNT(2 UP 3)
```

这是两个都包含 CALL_COUNT 的第 2，3 和 4 个元素的子数组。

内部子程序 UPPER（〈数组对象〉）取给定参数的最大下标。