第一章　Fortran 入门

本章将给初学 Fortran 的读者介绍 Fortran 的最基本概况。通过本章的阅读，读者将会编写一些最初级的 Fortran 源程序。在展开 Fortran 语言的内容介绍之前，先来介绍一点关于 Fortran 的发展历史。

第一节　Fortran 发展史

Fortran 是当今国际上极为流行的一种高级程序设计语言，它主要面向科学计算。Fortran 是英文词 Formula translation 的缩写，其含意为“公式翻译”。因而早期在我国曾把 Fortran 语言称为公式翻译语言。Fortran 语言比用英文表示的自然语言更接近数学语言，因此易为广大科学技术人员所接受。它好学、好记，便于推广使用。

第一个 Fortran 语言是在 1954 年提出来的，称为 Fortran Ⅰ，它于 1956 年在 IBM 704 计算机上实现。其后经过不断地发展，形成了很多不同的版本，但最为流行的是 1958 年出现的 Fortran Ⅱ，它对 Fortran Ⅰ 做了很多重要的扩充，如引进了子程序等概念。1958 至 1963 年期间，Fortran 在很多类型的计算机上得以实现。在此期间，又曾设计过 Fortran Ⅲ，但据说从未在任何计算机上实现。1962 年初出现了 Fortran Ⅳ，它对原 Fortran 语言做了某些改变，致使 Fortran Ⅱ 的源程序在 Fortran Ⅳ 的编译程序下不能全部直接运行，出现了语言不兼容问题。于是形成了当时最常用的 Fortran Ⅱ 和 Fortran Ⅳ 两种程序设计语言。

由于 Fortran 语言的种类很多，语义和语法的规定又不完全一致，但又大同小异，对于使用者很不方便，使用者热切希望有一

个能脱离实现机器、在各种机型上都能互换通用的 Fortran 语言，从而提出了 Fortran 语言标准化的问题。1962 年 5 月美国国家标准学会(American National Standards Institute)成立了有关工作组开展此项工作。1964 年提出了两个标准文本的草案，一为基本 Fortran(相当于 Fortran Ⅱ)，另一为 Fortran(相当于 Fortran Ⅳ)。1966 年 3 月正式公布了两个标准文本：一是美国国家标准 Fortran(ANSI X3.9 —1966)(相当于 Fortran Ⅳ)；一是美国国家标准基本 Fortran(ANSI X3.10 —1966)(相当于 Fortran Ⅱ)。美国国家标准基本 Fortran 是美国国家标准 Fortran 的一个子集，从而达到了语言的向上兼容。70 年代初我国计算机软件界参照这两个标准，在国产的 441B Ⅲ、DJS—6、DJS—8、DJS —100 等机型上编制了我国第一批 Fortran 编译程序，并于 1973 年起先后投入运行，它们采用的语言一般比 Fortran Ⅱ 大、比 Fortran Ⅳ 小。

由于国际上广泛使用了 Fortran 语言，1972 年国际标准化组织 ISO(International Standard Organization)公布了国际标准程序设计语言 Fortran (ISO R1539)。它描述了三种级别：基本级、中间级和完全级。基本级与 ANSI X3.10—1966 相近，完全级与 ANSI X3.9—1966 相近，而中间级则在两者之间。我国 Fortran 语言编译程序的研制在一定程度上受了 ISO 中间级的影响。

其后，根据语言的发展，美国国家标准学会于 1976 年对 1966 年公布的 ANSI X3.9—1966 进行了修订，增加了很多新的内容。1978 年 4 月美国国家标准学会正式公布了新的美国国家标准 ANSI X3.9—1978《程序设计语言 Fortran》，这就是我们通常称之为 Fortran 77 的 Fortran 语言标准，后经 ISO 正式采纳颁布为 ISO 1539—1980。我国国家标准局等同采用该标准，于 1982 年 5 月 12 日发布自 1983 年 5 月 1 日实施我国第一个《程序设计语言 Fortran》国家标准(GB 3057 —82)。ISO 的 Fortran 语言标准自此一直和美国国家标准相一致，工作组也是相同的，只是召集人有时稍有差别。

Fortran 77 公布之后，各国就开始研讨下一个 Fortran 标准

应是什么样的问题？几经周折，最后在1991年5月通过了研制期间称为Fortran 8x的Fortran 90，其美国编号为ANSI X3.198－1991，ISO编号为ISO/IEC 1539:1991。在此新标准中增加了许多新的功能，最重要的是增加了多字节字符集的数据类型及相应的内在函数，这一新数据类型的增加，给母语为非英语的国家在使用计算机方面提供了极大的支持。而且，此数据类型的方案正是采纳了我国计算机和信息处理标准化技术委员会程序设计语言分技术委员会Fortran工作组的提案的结果。

在1992年5月底ISO通过的ISO/IEC 10646－1.2通用多8位编码字符集(UCS－Universal Multiple－Octet Coded Character Set)第一部分之后，ISO的Fortran工作组又进一步在考虑研制一个适用于2000年、支持ISO/IEC 10646的下一个Fortran标准，并提出了一些有关思想和意图。以上这些思想脉络都说明了：Fortran程序设计语言虽然历史最悠久，但至今仍在日新月异地发展。本书将要介绍的Fortran语言就是最近才通过的Fortran 90，它的编译程序不久将会作为产品推出。

第二节 Fortran的语言元素

像自然语言一样，任何程序设计语言都是由一些最基本的语言元素组成的，Fortran语言也不例外。它的最基本语言元素包括：字符、常量、变量、语句、数据、表达式、过程等等。下面我们将一一给予定义和给出例子。

一、字　　符

Fortran语言中最基本的成分是字符，任何其它的Fortran元素都要由字符按一种规则组成。字符的集合称为Fortran语言的字符集。在Fortran Ⅳ中共有47个字符。它们是：

字母字符(26个)

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

Fortran 语言中使用的字母都是大写字母，程序中若出现小写字母时，作为相应的大写字母处理。

数字字符（10 个）

0 1 2 3 4 5 6 7 8 9

特殊字符（11 个）

+（加号）、-（减号）、*（星号）、/（斜线）、=（等号）、((左括号)、)（右括号）、,（逗号）、.（小数点）、'（撇号）、 （空格）

这些特殊字符的意义大多数是明显的，但 *、/、. 等在不同的位置时有不同的意义。空格字符的使用往往是为了使程序清晰。

在 Fortran 语言中，其源程序只允许出现这 47 个字符，但在字面常量（此名称仅在 Fortran Ⅳ 有效，在 Fortran 90 中另有新的含义）、A 编辑描述符、H 编辑描述符等中允许出现所在计算机能使用的任何字符。

二、常　　量

Fortran 程序中经常会出现一些常量，它们在程序执行的过程中是不变的。Fortran 语言规定有六种类型的常量，即：整型常量、实型常量、双精度型常量、复型常量、逻辑型常量、字面常量。

有的还允许使用双精度复型常量，但这一般是不允许的。前五种常量一般被称为算术型常量。

在 Fortran 语言中为提高处理效率把整型常量和实型常量区别开来了。如 24 为整型常量，24.0 为实型常量，而这在平常人们做算术运算中是没有区别的。

1. 整 型 常 量

用以表示一个正的、负的或为零的整数值。它由若干个 0～9 的数字组成，前面可以有正、负（+或-）符号，但不能包括小数点或其它符号。若前面不带符号时则视为正数。它所表示的整数范围由机型而定。例如，16 位计算机，它所能表示的整数范围是：$-2^{15}\sim2^{15}-1$ 之间，即 -32768～32767 之间，而 32 位计算机，它所

能表示的整数范围是：$-2^{31} \sim 2^{31}-1$ 之间，即 −2147483648～2147483647 之间。根据整数在计算机内的存放方式，整数是没有误差的。整型常量的例子是：

−103

+276

45

2. 实 型 常 量

实型常量是一个具有小数点的常量或为具有指数的常量。因此实型常量可以由三部分组成，即：整数部分、小数部分和指数部分。实型常量有两种表示方法，即：小数形式和指数形式。

(1)小数形式。实型常量的小数形式由一串 0～9 的数字和小数点构成，前面可具有正号或负号，若数字串之前不具有正、负号则认为是正的实型常量，但必须在数字串中带有小数点。具体形式可有三种：一个数字串后跟有一个小数点，如 10.；一个小数点后跟一数字串，如 .14；数字串中包含一个小数点，如 10.14；以上各情况中都可包含前缀正、负号。

(2)指数形式。实型常量小数形式后跟一实指数。实指数是由字母 E 后跟一个整型常量组成，此整型常量的位数最多为两位，且可带正、负号。但 E 之后不允许有小数点。现举例如下：

10^4 可表示成：1E+4、1E4、1.0E4、1.E+4，而 1.45 可表示成：145.E−2、14.5E−1、.145E1、.0145E+2、+0.145E+1 等。

如果写成 E34、.E34、0.14E2.3 等都是 Fortran 语言所不允许的。

实型常量的最大允许值及最大有效位数都是因所在机型而异的。

我们比较一下以下情况：

100　　Fortran 语言把它看作整型常量

100.　 Fortran 语言把它看作实型常量

1E+2　Fortran 语言把它看作实型常量

3. 双精度型常量

在科学计算中往往需要较高的精度，而一般计算机中的实型常量往往不能达到所需的精度，因此 Fortran 语言提供了双精度型常量。双精度型常量的有效数字比实型常量多出一倍左右，这样对大多数科学计算都能满足要求。Fortran 语言只要求双精度型常量比实型常量有较多的有效位数，并不要求其有效位数必须是实型常量的一倍。双精度型常量和实型常量具有相同的最大允许值，但这一点也并非硬性规定，只明确规定实型常量占内存一个存储单元，而双精度常量占内存的相邻两个存储单元。由于双精度运算一般都是用软件来实现，所以双精度型常量的运算速度比实型常量的运算速度要低得多。

双精度型常量是以指数形式表示的，它与实型常量的指数形式相类似，只是把指数符号从 E 变为 D。如果你在书写中将指数符号写成 E，那么不论你把有效数字位数写得多么长，Fortran 语言仍旧认为这是一个实型常量，它把多余的有效数字位数全部截断。

双精度型常量的例子是：

0.145D+1　　　0.14567987124097D−4

双精度型常量零表示为：0.0D0 或 0.D0

4. 复 型 常 量

复数是我们在科学计算中经常会碰到的一种数据类型，如二次方程的根就可能是复数。在电学中，交流电流、电压、阻抗等经常要使用复数表示。Fortran 语言提供了复型常量及相应的运算功能。

复型常量包括一个实部和一个虚部，在 Fortran 语言中用括在一个圆括号内的、用逗号分隔开的一对有序的实型常量表示，它们的前面可以带有正、负号，其中的第一个实型常量表示复数的实部，后者表示虚部。例如：

复数　4.3+2.4i　写为(4.3，2.4)

复数　0+40i　　写为(0，+4E1)（纯虚数）

在 Fortran 源程序的表达式内出现复数时，必须按复型常量的方式来书写，即必须写明括号及其数对之间的逗号。复型常量在

内存中占两个相邻的实型常量的存储单元，其中实部和虚部各占一个存储单元。

5. 双精度复型常量

在 Fortran 标准中并没有这个类型的常量，但有的 Fortran 编译程序提供了该非标准类型常量。它的含意是复型常量的虚部和实部都是双精度的实型常量。用括在一个括号中的两个双精度实型常量来表示，数对之间以逗号分开，第一个数表示实部，第二个数表示虚部。例如：

(0.D0, 0.145D+2)　　(1.44D+1, 0.D0)

(4.51D+1, 3.45D-2)　　(4.D-2, -3.4D3)

6. 逻辑型常量

我们知道，在程序中经常会需要一些“条件”，该“条件”是否成立，亦即是“真”或“假”，往往影响着进一步需要采取什么动作。这反映在 Fortran 中就是逻辑型常量，它只有两个值，即当“条件”成立时，我们就称其为“真”，表示为.TRUE.，当“条件”不成立时，我们就称其为“假”，表示为.FALSE.。注意，在 Fortran 语言中，逻辑型常量的两边必须各有一个“.”，即“真”为.TRUE.，“假”为.FALSE.，如果没有两侧的“.”，就会把它们作为一般的变量名来处理，而不是作为逻辑型常量来处理了。在以后的关系表达式和逻辑表达式中我们将进一步讨论逻辑型常量的问题。这里我们只需了解逻辑型常量的两个值，它们是.TRUE.和.FALSE.。逻辑型常量占内存的一个存储单元。注意，表示它们时必须在两侧各有一个“.”。

7. 字 面 常 量

在 Fortran 语言中，字符串可以做为常量处理，称为字面常量，其表现形式为：

$$nH\underbrace{x_1x_2\cdots x_n}_{n\text{个字符}}$$

其中的 $x_1x_2\cdots x_n$ 是 Fortran 中所允许的任何字符，在字面常量中除可包含 Fortran 字符集中的任何字符之外，还可以包含载

体计算机所允许出现的任何字符。在字面常量中,空格字符是有意义的,在 n 的计值中要把空格字符计算在内。例如:

4HGIRL　4H SIN

Fortran 语言规定每个字面常量中字符的最大个数不得超过载体计算机的每一存储单元所能存放的字符个数。如果字面常量中字符个数 n 小于载体计算机一个存储单元所能存放的最大字符个数 m,则这 n 个字符在该存储单元中靠左存放,后跟 m－n 个空格字符。

以上简述了 Fortran 语言各种类型的常量形式,这也就是整个 Fortran 语言中将要面对的各种类型的数据形式。

三、变　　量

在任何科学计算中都会出现一些执行过程里需要改变的量,这些量就是通常称之为变量的量。变量的值在程序执行过程中要通过赋值语句或其它方法予以赋值。在 Fortran 中,对每个变量都相应分配一个内存中的存储单元以存放数据。在 Fortran 程序中,每个变量都有一个名字称为变量名,该变量名只在本程序单元中有意义,即离开了该程序单元之后,此变量名就无意义了。同时,在同一程序单元内,变量名不能同名。

Fortran 语言对变量名的命名规则是:第一字符必须是字母字符、后面紧跟着字母或数字字符组成的 6 个以内的字符串。

以下是一些合法的变量名:

A, P1, M1AX, BROWN

以下是一些不合法的变量名(请读者应用变量名命名规则找出其不合法的原因):

3Y,　A.1,　C--D,　MS.LI,　αT,　B—ET,　BEI * JING

在选取变量名时,我们建议:

(1)采用相近似的、具有相应数学或物理意义的名字。

(2)尽量采用较简单、易记忆的名字。

(3)不要使用 Fortran 语言中的语句名、内部过程名、外部过

程名等做变量名。

在 Fortran 语言中，变量不但有名，而且有类型，它的类型基本上和常量的类型是相对应的，都有相应的类型说明语句来说明。

1. 整型变量和实型变量

Fortran 语言中对整型变量和实型变量有特殊的处理方法，因为它们是最常用的两种类型。

(1)隐式说明。Fortran 语言规定：凡变量名的第一个字母字符是 I、J、K、L、M、N 中任一个的变量，其变量类型均为整型，以其它字母字符为首的变量名均为实型。这就是 Fortran 语言的隐式说明原则，有时也称之为 I—N 规则。

如 L1、N2、IMIN、MAX 等都隐式说明了其所代表的变量是整型变量。而 A1、XVAR、E47B、YI 等都隐式说明了其所代表的变量是实型变量。因此，若没有其它说明，下列的赋值语句就是不恰当的。例如：

```
L1=4.2
YI=3
```

而若没有其它说明，下列的赋值语句就是合适的。例如：

```
MAX=478
XVAR=5.76
```

(2)显式说明。如果在写 Fortran 程序时想改变隐式说明对变量名的规定，就可使用类型说明语句来说明变量的类型。换言之，类型说明语句可以改变隐式说明中的变量类型。

说明是整型变量用 INTEGER，说明是实型变量用 REAL。其书写格式是：

```
INTEGER YI, XVAR, E47B
REAL L1,N2,IMIN
```

说明 YI、XVAR、E47B 为整型变量，而 L1、N2、IMIN 为实型变量。

一个类型说明语句可说明几个同一类型的变量，变量名之间用逗号分隔开来。

2. 双精度型变量

双精度型变量必须用双精度类型说明语句来说明。其书写格式是：

```
DOUBLE PRECISION X1,YI,LENG
```

以上类型说明语句说明 X1、YI、LENG 为双精度型变量。一个双精度类型说明语句可说明几个双精度型变量，变量名之间用逗号分隔开。

3. 复 型 变 量

复型变量必须用复型类型说明语句来说明。其书写格式是：

```
COMPLEX A1,BT,ORC
```

以上类型说明语句说明 A1、BT、ORC 为复型变量。一个复型类型说明语句可说明几个复型变量，变量名之间用逗号分隔开。

4. 双精度复型变量

双精度复型变量不是 Fortran 语言标准中的部分，但有的 Fortran 提供了此非标准类型。双精度复型变量必须用类型说明语句来说明。其书写格式是：

```
DOUBLE PRECISION COMPLEX B2,AX,IMA
```

以上类型说明语句说明 B2、AX、IMA 为双精度复型变量。一个双精度复型类型说明语句可说明几个双精度复型变量，变量名之间用逗号分隔开。

5. 逻辑型变量

逻辑型变量必须用类型说明语句来说明。其书写格式是：

```
LOGICAL C2,DD,MT
```

以上类型说明语句说明 C2、DD、MT 为逻辑型变量。一个逻辑型类型说明语句可说明几个逻辑型变量，变量名之间用逗号分隔开。

逻辑型变量只能赋以逻辑型的值，即它只能是.TRUE.或.FALSE.两个值中的一个。

以上是变量类型语句的一个基本说明，每个类型说明语句只能说明一种数据类型，但可同时说明具有该数据类型的几个变量。

类型说明语句放在程序单元的最前面,同时其起作用的范围只在本程序单元内,一离开此程序单元,变量的类型也同时失去其定义了。

四、数 据 类 型

在变量部分中我们已经介绍了6种有关的数据类型,其中双精度复型变量是非标准的。本段中我们只简单地把它们的分类回顾一下。在Fortran语言中可把数据类型分为三大类六种:

(1)算术型数据。它包括:整型数据、实型数据、双精度实型数据、复型数据和双精度复型数据(非标准的)

(2)逻辑型数据。作为复习,读者对以上每种数据类型请写出其类型语句,并按定义举出一、二个例子说明每种类型数据在Fortran语言中应如何书写。

(3)字面型数据。字面型数据只是为了输入、输出的需要而引入的,它没有类型语句来说明。其形式是:

$nHx_1x_2\cdots x_n$

即整型常量n后跟字符H再跟n个字符。此n个字符可以是字符集中的任何字符(包括空格字符在内,空格字符也参与计数),同时也可包括载体计算机所允许的各种字符。

字面型数据在输入、输出时使用,它一般只在数据初值语句或作为子程序的实元出现在调用语句中,这在以后有关的章节中介绍。

对于数据类型,我们应能熟练的写出它们在Fortran语言中的表示方式,前面介绍的是最常用的形式。

五、表 达 式

在科学计算中离不开各种运算公式,这些又和各种运算符及相关表达式有着密切关系。根据运算符的类别,Fortran语言中包含三类表达式:算术表达式、关系表达式和逻辑表达式。

1. 算术表达式

在科学计算中经常需要按公式计算某个变量的值，如计算二次方程的两个根 $x1$ 和 $x2$ 为：

$$x1=(-b+(b^2-4ac)^{1/2})/2a$$

$$x2=(-b-(b^2-4ac)^{1/2})/2a$$

在 Fortran 语言中(SQRT 为 Fortran 内在函数(在 Fortran 90 之前叫内部函数，现在统称内在函数)求平方根)将写为：

```
X1=(-B+SQRT(B*B-4.0*A*C))/(2.0*A)
X2=(-B-SQRT(B*B-4.0*A*C))/(2.0*A)
```

其中："="为赋值号，赋值号右端的式子称为算术表达式，此语句称为算术赋值语句。

算术表达式是用来计算数值的。组成算术表达式的基本元素是常量、变量、数组元素、函数引用、运算符和括号。

在 Fortran 语言中提供了五种算术运算符：+(加号)、-(减号)、*(乘号)、/(除号)和 * *(乘方)。

在表达式计算中应注意：

(1)两个算术运算符不能紧相邻。例如：A * -B 就是错的，应写为 A * (-B)。

(2)算术运算符的优先顺序是：先做指数运算，再做乘除，最后为加减。换句话说 * * 优先于 * 、/，而 * 、/又优先于+、-。

(3)在同一级运算中，按自左至右的顺序进行。例如，A/B * C 表示(A/B) * C，而不是 A/(B * C)。

(4)若有多层括号，从最内层括号开始计算。

(5)在 Fortran 表达式中不能省略乘号。

(6)Fortran 表达式不允许实型量、整型量进行混合运算。

(7)如连做几次乘幂运算，需用括号决定运算顺序。例如，A^{BC} 不能写为 A * * B * * C，应为 A * * (B * * C)。

(8)在使用两个整型量相除时要特别小心，因为其结果是忽略余数的整值。例如，2/3=0，3/2=1，(-5)/3=-1。

以上注意事项中很多与我们习惯稍有不同之处，因此，初学者应加以注意。运算之间哪些类型可以相互匹配，参见表 1—1 和表

1—2。

表1—1

A + - * / B		B的类型			
		整型	实型	双精度型	复型
A的类型	整型	整型			
	实型		实型	双精度型	复型
	双精度型		双精度型	双精度型	
	复型		复型		复型

表1—2

A ** B		B的类型			
		整型	实型	双精度型	复型
A的类型	整型	整型			
	实型	实型	实型	双精度型	
	双精度型	双精度型	双精度型	双精度型	
	复型	复型			

此外，我们应注意算术赋值语句和表达式的区别，X＝A＋B这是一个算术赋值语句，而A＋B则是表达式，因此表达式不应包括赋值号。

表1—3给出一些例子说明如何正确书写Fortran的表达式。

表1—3

数学表达式	Fortran表达式	
	错误写法	正确写法
$3a+b$	3A＋B	3.0＊A＋B
$2a+(-b)$	2A＋－B	2.0＊A＋(－B)
$\frac{a}{b}\div 2c$	A/B/2C	(A/B)/(2.0＊C)
$4a$	4＊A	4.0＊A

续上表

数学表达式	Fortran 表达式	
	错 误 写 法	正 确 写 法
a^{3+a}	A * * 3+A	A * * (3.0+A)
$cos3x$	COS3X	COS(3.0 * X)
$a[b+c(d+e)-f]$	A[B+C(D+E)-F]	A * (B+C * (D+E)-F)
$2a\div 3(b+\frac{e}{4c+d})$	2A/3B+E/4C+D	2. * A/(3. * B+E/(4. * C+D))

2. 关系表达式

在科学计算中往往要根据一些条件决定下一步应做什么工作，这些情况是根据前一步的计算结果得到的。在 Fortran 语言中用条件语句实现这一点。现举例说明(有关语句的含义在以后的章节中叙述)：

```
C   EXAMPLE
    I=1
    J=1
2   I=I*J
    J=J+1
    IF (J.GT.5) GOTO 3
    GOTO 2
3   ⋮
```

这里的 J.GT.5 就是关系表达式。Fortran 语言运用关系表达式变出许许多多复杂的判断条件以利于程序的运转。

在 Fortran 语言中提供了六种关系运算符，参见表 1—4。

表 1—4

关系运算符	数学含义	英文原文
.GT.	大于	Greater Than
.GE.	大于等于	Greater than or Equal to
.LT.	小于	Less Than
.LE.	小于等于	Less then or Equal to
.EQ.	等于	EQual to
.NE.	不等于	Not Equal to

关系运算符的前、后各有一个圆点，这是 Fortran 语言的规定，一定要记住此符号，否则就会产生其它意义。若你想表示 A 小于 B，则必须写成 A.LT.B；若你写成 ALTB，Fortran 语言就会把此理解成一个变量名。

关系表达式就是两个算术表达式中间用一个关系运算符连接起来而构成的表达式。以下是关系表达式的例子：

```
4.0**3+5..GT.80.
B*B-4.*A*C.LT.0.0
```

Fortran 语言规定关系表达式两边的算术表达式应具有相同的类型，如整型表达式必须和整型表达式相比较，不允许整型表达式和实型表达式相比较。但具体到某台计算机上时，有时会放松这个规定。

关系表达式的计算顺序是：先计算关系运算符两边的算术表达式（按算术表达式的计算规则），然后再对两个计值结果进行比较（按关系运算符的规定）。

关系表达式的结果是两个逻辑量.TRUE.或.FALSE.中的一个。在表达式中不允许两个关系运算符相邻接出现，因为这是毫无意义的。由于在计算机中表示实型数据时都是近似的，因此与理论上的实型数之间都有误差。所以在关系表达式两边都是实型时，建议尽量不用.EQ.或.NE.，而是根据具体的情况给出一个允许的误差范围。如想比较两个实数 A 和 B，不要用 A.EQ.B，而考虑给出一个范围，如在 10^{-4} 之内时就认为是相等，那么可用：

```
ABS(A-B).LT.1.0E-4
```

此处 ABS 是 Fortran 语言的内在函数求绝对值。

关系表达式往往在逻辑 IF 语句中使用，如我们在例子中看到的那样。

3. 逻辑表达式

在实际问题中经常出现一些逻辑量的比较，而不单是算术表达式之间的比较。比如我们有时要讨论 A>B 和 C<D 两个条件是否要同时成立，或只要 A>B 成立，或只要 C<D 成立。这些就

是逻辑表达式可以起作用的地方。

例如我们要考虑$-1 \leqslant A \leqslant +1$，即问什么时候A处在〔$-1$，$+1$〕这个闭区间之内。可用下列两个逻辑表达式中的任一个来表示：

.NOT.((A.LT.(−1.0)).OR.(A.GT.(+1.0)))&

(A.GE.(−1.0)).AND.(A.LE.(+1.0))

Fortran语言提供了以下三种逻辑运算符：.AND.（与）、.OR.（或）和.NOT.（非）。

具体意义如下（设A、B为逻辑量）：

A.AND.B

表示：如A为“真”和B为“真”同时成立，则A.AND.B为“真”。

A.OR.B

表示：如A或B中只要一个为“真”，则A.OR.B为“真”。

.NOT.A

表示：如A为“真”，则.NOT.A为“假”，如A为“假”，则.NOT.A为“真”。

由此，我们可以列出计算逻辑真值的“真值表”，参见表1—5。

表1—5

A	B	A.AND.B	A.OR.B	.NOT.A	.NOT.B
T	T	T	T	F	F
T	F	F	T	F	T
F	T	F	T	T	F
F	F	F	F	T	T

表中T代表.TRUE.，F代表.FALSE.。

逻辑运算符和关系运算符一样，在其两端必须各有一个圆点，否则即失去逻辑运算符的意义。在书写时我们需要注意这一点，如在写.NOT..TRUE.时若写成.NOT.TRUE.就不对了。

一个逻辑表达式中可以同时含有逻辑运算符、关系运算符和算术运算符三者。因此Fortran语言中规定了它们的运算顺序为：先计算算术表达式，按算术运算符的优先顺序计值；其次计算关系表达式的逻辑值；最后计算逻辑表达式的逻辑值。逻辑运算符的优

先顺序是：先.NOT.，次.AND.，最后为.OR.。

综上所述，各类运算符的优先顺序如表1—6所示。

表1—6

类别	运算符	优先级
括号	( )	1
算术运算符	* *	2
	* /	3
	+ −	4
关系运算符	.GT.;.GE.;.LT.;.LE.;.EQ.;.NE.	5
逻辑运算符	.NOT.	6
	.AND.	7
	.OR.	8

下面举一例来看一看如何执行以上规则。

已知：A=3.5，B=4.0，C=4.5，求：A+B.EQ.C.AND.B+C.GT.A−B。

依照优先顺序执行过程如下：

```
A+B.EQ.C.AND.B+C.GT.A - B
①7.5          ①8.5    ① .5
   ②F              ②T
          ③F
```

其中T代表.TRUE.，F代表.FALSE.，可见多么复杂的逻辑式我们也可循序渐进地分层处理。

逻辑表达式主要用于各种条件语句中。

六、语句、注解、行和标号

和自然语言一样，语言都是由语句构成的。Fortran语言也是由各种类型的Fortran语句构成的。主要有可执行语句和非执行语句两大类。

可执行语句用于计算、转移、输入输出等。非执行语句主要用于提供某些信息，如说明语句等。

Fortran语句是语法项目的一个序列。除赋值语句和语句函

数外，每个语句都是以关键词开始的。例如，READ 语句。

Fortran 语句可以写在一行内，若一行内写不下，也可写在几行内，但一行内不允许写几个语句。此外，行是 72 个字符的序列。

Fortran 源程序中的行可分为以下几种：

(1)始行。一个语句可以写成若干行，其中的第一行称为始行。始行的第 6 列上必须为空格或 0。始行不是注解行。其第 1 至 5 列可含有语句标号或全为空格。

(2)续行。一个语句若在一行内写不完可以写几行，除第一行称为始行外，其余后继行都称为续行。续行的第 6 列写除空格符和 0 以外的其它字符，如数字、字母等字符。Fortran Ⅱ 和 Ⅳ 规定一个语句最多可以有 9 个续行，Fortran 77 规定可以有 19 个续行。续行的第 1 至 5 列只含空格字符。

(3)注解行。若在某行的第 1 列上写了字母“C”或星号，该行称为注解行，注解可以写在第 2 至 72 列之间。若注解在一行内写不完可以写在几行内，但每一行的第 1 列都必须写字母“C”或星号。也就是说，注解不能有续行的概念。注解行不能插在一个语句的始行和其续行之间。注解行中的字符可以是载体计算机能够表示的任何字符。若一行全部是空格字符也称之为注解行。注解行不影响程序的执行，只为程序提供说明加强可读性。

(4)结束行：在 Fortran Ⅱ 中规定 END 不是语句，即在第 7 列之后写 END 表示一程序单元的结束。其第 1 至 6 列为空格。但 Fortran Ⅳ 之后规定了 END 为语句，不叫结束行。

Fortran 源程序中有的语句是要由其它语句来引用的，这些要被引用的语句就要给予语句标号。因此，语句标号提供了引用语句的可能。它的形式是在语句始行的第 1 至 5 列的任何地方写一个 1 到 5 个数字的序列(其中至少有一个数字不是零)称为语句标号的定义。在一个程序单元内不得定义相同的语句标号，语句标号只在本程序单元内有意义。在区分语句标号时，空格和前导零都是无意义的。

Fortran 源程序在书写时，对列都有严格的规定，尤其是第 1

至 6 列。这称为固定格式的书写方法。这与早期的输入设备卡片读入机有密切关系，因为每张卡片有 72 列，正好表示 Fortran 中的一行。这种书写方法给使用者带来很大的不便。因此在 Fortran 90 中冲破了这一点，引进了 Fortran 源程序的自由格式书写方法，这必将给使用者带来很大的方便。

关于一个程序如何写语句、注解等举一简单例子来说明：

```
C   THIS IS AN EXAMPLE        ……此为注解行
      DO 3 I=1,4            ……此为语句始行
        J=I*I
3       WRITE(5) I,J         ……此为带语句标号的语句
      END
```

由此可见，书写 Fortran 源程序时必须很注意其格式，否则会导致编译错误。这点对初学者尤为重要。

七、过程和过程引用

在 Fortran 语言中过程包括函数和子程序。有四种过程：内在函数、语句函数、外部函数和子程序。

其中，内在函数、语句函数和外部函数统称为函数，而外部函数和子程序又称为外部过程。

因为在科学计算和数据处理中有很多同样类型的工作会反复进行，如求绝对值、开平方、求解线代数方程组等。这些工作往往需要用一系列 Fortran 语句来实现。为提高效率并保证正确性，经常编制一些标准函数、标准子程序为使用者提供这种便利。

在定义过程和过程引用中，首先我们要定义虚元和实元两个重要的概念。虚拟变元简称为虚元。

过程的虚元是符号名。当过程的虚元是符号名时，它标识变量、数组或过程，当每次引用该过程时，它与实元结合。

语句函数、函数辅程序和子程序辅程序利用虚元来指出实元的类型，并指出每个变元是单个值、数组值或过程。虚元分类为变量、数组或虚拟过程，而语句函数的虚元只能是变量。

对子程序或函数的一次特定的引用而言，实元说明了与虚元结合的实体是什么。每个实元的类型必须与它相结合的虚元的类型一致。

在执行过程引用时，在对应的虚元和实元之间建立结合关系。例如，将第一个虚元和第一个实元相结合，第二个虚元和第二个实元相结合，等等。仅当实元的类型和对应的虚元类型相同时，才能实现合法的结合。若实元是一个表达式，则结合之前先求值；若虚元当前未和实元结合，则它无定义。

1．内在函数及其引用

内在函数是由处理系统提供的。也就是说，Fortran 编译系统必须提供这些函数才可称为一个完整的、符合标准的编译系统。每个内在函数有特定的名字、专门的意义、特定的变元名及变元顺序、专门的函数类型等，参见表 1—7。

表 1—7

内在函数	定　　义	变元个数	属名	特　定　名	类　型	
					变　元	函　数
类型转换	转换为整数 int(a)	1		INT	实　型	整　型
				IFIX	实　型	整　型
	转换为实数	1		REAL	整　型	实　型
				FLOAT	整　型	实　型
	转换为整数	1		ICHAR	字符型	整　型
截　断	int(a)	1		AINT	实　型	实　型
最近同型取整	int(a+.5) 当 $a\geq 0$ int(a−.5) 当 $a<0$	1		ANINT	实　型	实　型
最近取整	int(a+.5) 当 $a\geq 0$ int(a−.5) 当 $a<0$	1		NINT	实　型	整　型
绝对值	$\lvert a\rvert$	1		IABS	整　型	整　型
	$\sqrt{\lvert ar^2+ai^2\rvert}$			ABS	实　型	实　型

续上表

内在函数	定义	变元个数	属名	特定名	类型	
					变元	函数
求余	$a_1-\text{int}(a_1/a_2)*a_2$	2		MOD	整型	整型
				AMOD	实型	实型
符号传送	$\|a_1\|$ 当 $a_2\geqslant 0$ $-\|a_1\|$ 当 $a_2<0$	2		ISIGN	整型	整型
				SIGN	实型	实型
正差	a_1-a_2 当 $a_1>a_2$ 0 当 $a_1\leqslant a_2$	2		IDIM	整型	整型
				DIM	实型	实型
选最大值	$\max(a_1,a_2,\cdots)$	$\geqslant 2$		MAX0	整型	整型
				AMAX1	实型	实型
				AMAX0	整型	实型
				MAX1	实型	整型
选最小值	$\min(a_1,a_2,\cdots)$	$\geqslant 2$		MIN0	整型	整型
				AMIN1	实型	实型
				AMIN0	整型	实型
				MIN1	实型	整型
平方根	$(a)^{1/2}$	1		SQRT	实型	实型
指数	e**a	1		EXP	实型	实型
自然对数	log(a)	1		ALOG	实型	实型
常用对数	$\log_{10}(a)$	1		ALOG10	实型	实型
正弦	sin(a)	1		SIN	实型	实型
余弦	cos(a)	1		COS	实型	实型
正切	tan(a)	1		TAN	实型	实型
反正弦	arcsin(a)	1		ASIN	实型	实型
反余弦	arccos(a)	1		ACOS	实型	实型
反正切	arctan(a)	1		ATAN	实型	实型
	$\arctan(a_1/a_2)$	2		ATAN2	实型	实型
双曲正弦	sinh(a)	1		SINH	实型	实型
双曲余弦	cosh(a)	1		COSH	实型	实型
双曲正切	tanh(a)	1		TANH	实型	实型
按字序大于等于	$a_1\geqslant a_2$	2		LGE	字符型	逻辑型

续上表

内在函数	定义	变元个数	属名	特定名	类型	
					变元	函数
按字序大于	$a_1>a_2$	2		LGT	字符型	逻辑型
按字序小于等于	$a_1\leqslant a_2$	2		LLE	字符型	逻辑型
按字序小于	$a_1<a_2$	2		LLT	字符型	逻辑型

内在函数的引用就如把所要引用的函数作为表达式的初等量那样来引用。例如：

Y=SIN(0.5)*COS(1.5)+2.

这里引用内在函数只要写出内在函数的正确的名和用圆括号括起实元表，但实元的类型、个数、顺序不得有错，必须如表 1－7 所规定的那样。引用结果就是得到一个相应的函数值。实元表中的各个实元可以是一个具体值，也可是以是一个表达式。例如：

Y=TAN(1.2+A)*SIN(3.5*B)/2.0

2. 语句函数及其引用

语句函数是一种由单个语句函数语句指明的过程，语句函数语句在形式上类似于算术赋值语句或逻辑赋值语句。其一般形式是：

fun ([d[,d]…])=e

其中：fun 为语句函数的符号名；d　为语句函数虚元，可出现在表达式 e 中；e　为表达式。

语句函数语句只能出现在引用它的那个程序单元的所有说明语句之后，而在第一个可执行语句之前。语句函数语句属于 Fortran 语言中的非执行语句，但又不是说明语句，它不是正常执行序列中的一部分。语句函数只在它所定义的程序单元内才能被引用，在其它的程序单元内不允许被引用。引用语句函数是作为表达式的一个初等量来引用的。引用按以下步骤进行：

(1)计算实元的表达式的值。

(2)实元与相应虚元结合。

(3)求右端表达式 e 的值。

(4)必要时要做相应的类型转换。

因此在引用语句函数时,实元的个数、顺序、类型应和语句函数中的要求相一致。由于语句函数的虚元名只用于表示语句函数中虚元的顺序、个数和类型等,在引用中最终要由实元所代替。所以其虚元名在本程序单元内其它地方也可用做变量名。例如,语句函数定义及其引用:

```
    ⋮
FUN (X,Y,Z) =2.0*X+Y*Z
    ⋮
FAX=A+B**3-FUN(3.,7.2,0.4)
    ⋮
```

3. 外部函数及其引用

外部函数是在引用该外部函数的程序单元以外的地方指明的函数,不是由处理系统的编译程序指明必备的。一般可有两种情况:一是在各种科学计算中频繁出现的,因此由科学工作者编制出很多常用的、标准的外部函数(如求一个常用函数的定积分值等);一是在本程序单元内经常频繁出现的求某个函数值,可能比较繁琐而复杂,此时也往往用外部函数的方式来实现。

外部函数是过程的一种,在该程序单元内它的第一个语句一定是 FUNCTION 语句。其一般形式是:

〔typ〕FUNCTION fun (〔d〔,d〕…〕)

其中:typ 为 INTEGER、REAL 或 LOGICAL,指明函数值的类型;fun 为本外部函数的符号名,叫外部函数名;d 为虚元,可以是变量名、数组名或虚过程名等。

外部函数这种程序单元内必至少有一个语句对外部函数名赋值,且至少有一个 RETURN 语句可返回到引用该外部函数的地方去。

外部函数的引用步骤是:

(1)计算实元的表达式的值。

(2)实元与相应的虚元相结合。

(3)执行外部函数指明的整个程序动作。

在引用中,要注意类型的一致性,在 Fortran Ⅱ 和 Fortran Ⅳ 中,注意外部函数不能是字符型的。例如,外部函数的定义及其引用:

```
FUNCTION FOT(X)
  FOT=X**2+SQRT(1.+2.*X**2)
  RETURN
END
```

在主程序中,如在某处引用此外部函数:

```
    ⋮
  Y=2.4*FOT(3.6)-1.5+2.6*FOT(4.2)**2
    ⋮
```

在外部函数程序单元内不能出现 SUBROUTINE、PROGRAM 等语句。外部函数名是全局名,因此不能和其它全局名相同。

4. 子程序及其引用

子程序是在引用该子程序的程序单元以外的地方指明的。子程序是过程。

子程序是以 SUBROUTINE 语句作为其第一个语句的程序单元。其一般形式为:

SUBROUTINE sub[([d[,d]…])]

其中:sub 为 SUBROUTINE 语句所在的子程序的符号名。sub 是一子程序名;d 为虚元,可以是变量名、数组名或虚过程名等。

由上述一般形式我们可以知道虚元 d 是可以没有的,此时的形式就可以是:

SUBROUTINE sub

或 SUBROUTINE sub ( )

子程序的引用由 CALL 语句实现,其一般形式是:

CALL sub [([d_1[,d_2]…])] (d_i 是相应的实元)

或　CALL sub ( )

或　CALL sub

但绝不可为 CALL SUBROUTINE sub 〔(d_1〔,d_2〕…〕)〕,这是一般初学者最易犯的错误。

CALL 语句的执行步骤是:

(1)计算实元表达式的值。

(2)实元与相应虚元结合。

(3)执行被引用子程序指明的各项动作。

在子程序引用中的实元必须在顺序、个数、类型上与所引用的子程序的虚元表中的相应虚元相一致。

SUBROUTINE 语句必须是子程序单元中的第一个语句。在子程序中不得包含 FUNCTION 或 PROGRAM 语句。子程序的符号名是一个全局名,不得与别的任何其它全局名相同,并且也不能和它自己程序单元内的任何局部名相同。例如,子程序的定义及其引用:

```
SUBROUTINE FOT1(F,X)
  F=X**2+SQRT(1.+2.*X**2)
  RETURN
END
```

可在其它程序单元中的某处引用此子程序:

```
     ⋮
CALL FOT1(F1,3.6)
CALL FOT1(F2,4.2)
Y=2.4*F1-1.5+2.6*F2**2
     ⋮
```

我们可以把这里的引用情况和上面对外部函数的引用情况比较一下,可以仔细地考虑什么时候使用外部函数,什么情况使用子程序。当然,这里举的例子是非常简单的,我们在具体工作中绝不会使用如此简单的外部函数和子程序,因为这种简单情况完全可用语句函数语句处理。

过程是 Fortran 语言中频繁使用的一种程序单元,是一个强

有力的语言成份。它们可以把繁杂的科学计算变得结构简单并且易于调试。

八、源　程　序

用 Fortran 语言编写的计算机程序就称为 Fortran 源程序。这种程序在进入计算机之后，必须通过相应的 Fortran 编译程序把它翻译成相应载体计算机的机器语言的目标程序，这时计算机才能执行该目标程序去完成程序作者的意图。

在本节中我们再次提醒读者，从 Fortran Ⅱ 到 Fortran Ⅳ 到 Fortran 77 都应严格地按照固定格式书写 Fortran 源程序，即：一行只能写一个 Fortran 语句；一行包含 72 列；第 1 至 5 列可包含语句标号；第 6 列用来表示续行；第 7 至 72 列书写语句内容；一个语句不得包含多于 660 个字符；第 1 列若为字母 C 或星号为注解行。

这个源程序的固定格式给程序员带来了极大的不便，在 Fortran 90 中才提供了自由格式。

Fortran 源程序是按程序单元来执行的。程序单元由语句和任选注解行的序列组成。一个程序单元或是一个主程序，或是一个辅程序。

辅程序是以 FUNCTION 语句或 SUBROUTINE 语句作为第一个语句的程序单元。第一个语句是 FUNCTION 语句的辅程序，称为函数辅程序。第一个语句是 SUBROUTINE 语句的辅程序，称为子程序辅程序。函数辅程序和子程序辅程序统称为过程辅程序。

在程序单元内允许的语句是：

(1)FORMAT 语句可以出现在任何地方。

(2)所有的说明语句必须放在所有 DATA 语句、语句函数语句和可执行语句之前。

(3)所有语句函数语句必须放在所有可执行语句之前。

(4)DATA 语句必须出现在说明语句之后，语句函数语句和

可执行语句之前。

(5)在程序单元的说明语句内，IMPLICIT 语句必须放在所有其它说明语句之前。

以上有关源程序的一些具体规定都是我们在编写 Fortran 源程序时要认真遵守的，在以后的章节中，还要作详细的介绍，并逐渐展开介绍一些尚未介绍的内容。

本节中我们介绍了 Fortran 最基本的各语言成份和有关规定，并给出了一些最简单的例子。

第三节　最基本的 Fortran 语句

Fortran 语言中的语句分为可执行语句和非执行语句两大类。可执行语句指明动作，并在可执行程序中形成执行序列。非执行语句指明数据的特性、排列和初值，以及指明包含的编辑信息、语句函数和程序单元的类别。非执行语句不是执行序列的一部分。非执行语句可以带语句标号，但这种语句标号不能用于控制执行序列。

Fortran 语言的可执行语句包括：

(1)算术、逻辑、字符赋值语句和语句标号赋值(ASSIGN)语句。

(2)无条件 GOTO、赋值 GOTO 和计算 GOTO 语句。

(3)算术 IF 和逻辑 IF 语句。

(4)块 IF、ELSE IF、ELSE 和 END IF 语句。

(5)CONTINUE 语句。

(6)STOP 和 PAUSE 语句。

(7)DO 语句。

(8)READ 和 WRITE 语句。

(9)REWIND、BACKSPACE、ENDFILE 和 OPEN 语句。

(10)CALL 和 RETURN 语句。

(11)END 语句。

Fortran 的非执行语句包括：

(1)PROGRAM、FUNCTION 和 SUBROUTINE 语句。

(2)DIMENSION、COMMON、EQUIVALENCE、IMPLICIT、EXTERNAL、INTRINSIC 和 SAVE 语句。

(3)INTEGER、REAL、COMPLEX 和 LOGICAL 类型语句。

(4)DATA 语句。

(5)FORMAT 语句。

(6)语句函数语句。

在本节中只介绍最基本的语句，以便我们可以编写一些简单但是完整的 Fortran 源程序。

一、说 明 语 句

说明语句是非执行语句，它们是：

(1)DIMENSION 语句。

(2)EQUIVALENCE 语句。

(3)COMMON 语句。

(4)INTEGER、REAL、COMPLEX 和 LOGICAL 类型语句。

(5)IMPLICIT 语句。

(6)EXTERNAL 语句。

(7)INTRINSIC 语句。

(8)SAVE 语句。

但在这里只介绍各种类型语句、IMPLICIT 语句和 DIMENSION 语句。其它说明语句将在有关的部分予以介绍。

1. 类 型 语 句

类型语句用来确认隐含的类型或否定隐含的类型，在类型语句中还可以指明有关的维信息。各种变量、数组、外部函数或语句函数的符号名出现在类型语句内，给程序单元指明它们的数据类型。

注意，在一个程序单元内，一个名只能一次显式地指明其类型。类型语句中的名不能包含主程序及子程序的符号名。

类型语句的一般形式是：

typ V[,V]…

其中：typ 为 INTEGER、REAL、COMPLEX、DOUBLE PRECISION 或 LOGICAL 之一，在 Fortran 77 中还可以是 CHARACTER；V 为变量名、数组名、数组说明符、函数名或虚拟过程名（虚拟过程名在 EXTERNAL 语句中再讨论）。

以下是 Fortran 77 中类型语句的几个简单的例子。例如：

REAL IJ1M, I(15,20), NET

在隐式说明中，以 I－N 这 6 个字母为首的名都表示为整型量。这里表示的三个量其名都是 I－N 6 个字母中的一个为首的名，但经过这个语句说明后，IJ1M 和 NET 变成实型变量，而 I(15,20)是个数组（其第一维的大小为 15，第二维的大小为 20）且是实型的。例如：

INTEGER AT(10,10,20), HIT, OTT(10,20)

在隐式说明中，以 A、H、O 为首的名表示为实型量，但这里说明 AT(10,10,20)是一整型数组（第一维的大小为 10，第二维的大小为 10，第三维的大小为 20），HIT 是整型变量，OTT(10,20)是整型数组（第一维的大小为 10，第二维的大小为 20）。

由以上两例可见，变量名和数组名之间的分隔符都是逗号，切不可用错。下面我们举几个例子：

LOGICAL SWITCH, CODE, LIGHT

这个语句说明 SWITCH、CODE、LIGHT 这三个变量都是逻辑变量（即其值或为. TRUE. 或为. FALSE. ）。

DOUBLE PRECISION BIG,FAT

这个语句说明 BIG、FAT 这两个变量都是双精度变量。

COMPLEX COMPA, COMPB

这个语句说明 COMPA、COMPB 这两个变量都是复型变量。

在书写类型语句时，首先要注意你所使用的计算机的 Fortran 编译程序允许有哪些算术类型，其次在编写类型语句时注意正确的书写方式，即类型和变量名表之间须有一个以上空格，而各变量

名之间用逗号分开，然后一个变量名之后不再添加任何符号，最后要检查一下是否在程序单元中使用的全部变量名、数组名、数组说明符、函数名或虚拟过程名都已声明过其类型而且仅声明过一次。

如果变量中所含内容全部是字符，就需要引进字符变量这种类型。其类型语句的形式是：

CHARACTER [* len[,]]nam[,nam]…

其中：nam 为下列形式之一：

v[* len]　　　　v 是变量名

a[(d)][* len]　　　a 是数组名，a(d)是数组说明符

len 是字符变量或字符数组元素的长度(即字符个数)，称为长度说明。len 必须是一个无符号非零整常数。

对于 len 所说明的长度的管辖范围是：

(1)直接跟在词 CHARACTER 之后的长度 len 是该语句内没有自己的长度说明的每一实体的长度说明。

(2)直接跟在一个实体后面的长度说明仅是该实体的长度说明。

(3)对于数组，指明的长度是其每一个数组元素的长度。

(4)若对一个实体没有指明长度，则它的长度为 1。

现举例以说明如下：

CHARACTER A * 2,NAME * 10,STREET * 40

以上为三个字符型变量。其中 A 的长度为 2，NAME 的长度为 10，STREET 的长度为 40 个字符。

CHARACTER * 10 A,B * 20,C

以上三个字符型变量。其中 A 和 C 的长度分别为 10，B 的长度为 20 个字符。

CHARACTER A,B,C * 40

以上三个字符型变量。其中 A 和 B 的长度分别为默认值 1，C 的长度为 40 个字符。

CHARACTER A(10,20) * 10

以上的数组 A(10,20)为字符型数组，其中每一个数组元素的

长度都是 10 个字符。

显式的类型语句一般都写在非执行语句中说明语句的最前面，以达到一目了然的目的。在编写程序时最好还要留出一些位置来随时补充后续的变量类型说明。

2. IMPLICIT 语句

从以上有关类型语句的讨论中，我们可以想到对于大型的 Fortran 程序的编制，显式类型语句是非常不方便的。编制人必须把每个变量都列出来，必须弄清楚没有漏掉任何一个需加以说明的变量和数组等。因而在 Fortran 语言中为部分解决此困难，引进了 IMPLICIT 语句。

IMPLICIT 语句的作用就是用于改变预先隐含的整型或实型或确认预先隐含的整型或实型，或给出一有关类型的命名范围。

IMPLICIT 语句的作用范围仅在它所在的程序单元。IMPLICIT 语句的一般形式是：

```
IMPLICIT typ (a[,a]…)[,typ(a[,a]…)]…
```

其中：typ 为 INTEGER、REAL、LOGICAL、DOUBLE PRECISION、COMPLEX 或 CHARACTER [* len]的其中之一；a 是单个字母或按字母次序的字母的范围。

一个范围就用该范围的首字母和末字母，其间用"—"(横线)隔开来表示。因此，写字母范围 a1—a2 和写整个从 a1 到 a2 的全部字母表的作用是相同的。len 是字符实体的长度，是无符号的非零整常数。若没有指明 len，则其长度为 1。

IMPLICIT 语句指明变量、数组、外部函数和语句函数的类型，这些变量名、数组名、外部函数名和语句函数名是以该语句说明中作为单个字母或作为包括在字母范围内出现的任一字母开头的。

IMPLICIT 语句不改变任何内在函数的类型。

对于特定的变量名、数组名、外部函数名和语句函数名，可以用类型语句中出现的那些名来改变或确认 IMPLICIT 语句的类型说明。FUNCTION 语句中的显式类型说明也能取代 IMPLIC-

IT 语句对该函数辅程序名的类型规定。当一个特定名出现在 CHARACTER 语句中时，其长度也被取代了。

在一个程序单元的说明语句内，IMPLICIT 语句必须处在所有其它说明语句的前面。一个程序单元内可以包含若干个 IMPLICIT 语句。在一个程序单元的所有 IMPLICIT 语句中，相同的字母不得作为单个字母或被包括在一个字母范围内出现一次以上。

IMPLICIT 语句举例如下：

```
IMPLICIT LOGICAL (L), INTEGER(A—C), REAL(Y)
```

这个语句说明以 L 为首的为逻辑型量，以 Y 为首的为实型量，以 A、B、C 为首的是整型量。凡这里没有说明的或取其显式说明或用其预先隐式说明的规定。

若在一程序单元内有以下两个语句：

```
IMPLICIT REAL(A—O,X—Z)
INTEGER ATI, ZEBRA
```

以上 IMPLICIT 语句说明从 A 到 O、X、Y、Z 为首的是实型量，而第二个语句又说明 ATI、ZEBRA 为整型量，这时按 IMPLICIT 语句似乎它们应是实型量，但显式类型语句的结果取代了 IMPLICIT 语句的说明，因此这两个量应为整型量。

所以，对一个量是什么类型的量，需要全面考察所有的类型语句才能得到准确的信息。

3. DIMENSION 语句

在介绍 DIMENSION 语句之前，先要介绍数组的概念。在科学计算中，离不开向量或矩阵运算，那么在 Fortran 语言中如何表示呢？向量或矩阵在 Fortran 中被表示成所谓数组。

数组是数据的非空序列。数组元素是该数据序列中的一项。数组名是数组的符号名。数组元素名是由下标限定的数组名。数组元素名标识序列中的一个元素。下标值指明标识哪一个数组元素。改变数组元素名的下标值可以标识不同的数组元素。数组名只在它被说明的程序单元内有效。

在一个程序单元内，数组说明符指明了标识数组的符号名，并且指明了该数组的某些特性。在一个程序单元内，一个数组名只允许有一个数组说明符。数组说明符的形式是：

a(d[,d]…)

其中：a 为数组的符号名；d 为维说明符。

数组的维数是数组说明符中维说明符的个数。最小维数是 1，最大维数是 3(Fortran 77 中规定最大维数是 7)。

维说明符的形式是：

d

其中：d 为整常数或整变量名，称为维上界。维下界是 1。

例如，DIGIT(14)表示一个一维的，其维上界为 14 的数组。数组名是 DIGIT，它共有 14 个数组元素。

若一个数组说明符是 A(2,3)，它表示 A 是一个二维数组，第一维上界为 2，第二维上界为 3，它共有 6 个数组元素。它们在存储单元中的存放顺序如下：A(1,1)，A(2,1)，A(1,2)，A(2,2)，A(1,3)，A(2,3)。

在这里要特别注意它们的存放顺序，先固定其它下标值，顺序排第一个下标，再增加第二个下标值，顺序排第一个下标…，数组元素就是这样排列的。对三维数组依次类推。在 Fortran 语言中数组元素的排列方式和通常的习惯不同，采用按列顺序排列的方式，因此我们需要特别予以注意。

按数组元素排序所确定的数组元素的存储序列构成数组的存储序列。数组中存储单元的个数是 X * Z，其中 X 是数组中元素的个数，Z 是每一个数组元素占有的存储单元的个数。

数组元素名的形式是：

a(s[,s]…)

其中：a 为数组名；(s[,s]…)是下标；s 为下标表达式。

下标表达式的个数必须等于该数组名的数组说明符中的维数。

下标的形式是：

(s[,s]…)

其中:s 为下标表达式。

下标包括下标表达式的表以及外面的括号。下标表达式在 Fortran Ⅳ 中只能是 C * V±K 的形式。下标表达式不允许包含数组元素引用和函数引用。

在程序单元内,每个下标表达式的值必须大于或等于 1,且不许超过该程序单元内其数组所说明的相应维上界。

下标指明各个下标的值。下标值决定了该数组元素名标识哪一个数组元素。在程序单元内,下标值依赖于下标中下标表达式的值,并依赖于此程序单元中该数组说明符指明的数组维。若下标值为 r,则标识数组的第 r 个元素。见表 1—8。

表 1—8

n	维说明符	下　标	下　标　值
1	(d_1)	(s_1)	s_1
2	(d_1,d_2)	(s_1,s_2)	$1+(s_1-1)+(s_2-1)*d_1$
3	(d_1,d_2,d_3)	(s_1,s_2,s_3)	$1+(s_1-1)+(s_2-1)*d_1$ $+(s_3-1)*d_1*d_2$

表中:n 为维数,$1\leqslant n\leqslant 3$;d_i 为第 i 维的上界值,也就是第 i 维的大小;s_i 为第 i 个下标表达式的整数值。

下标为(1)、(1,1)或(1,1,1)者,其下标值均为 1,且标识数组的第一个元素。(d_1)、(d_1,d_2)、(d_1, d_2, d_3)的下标则标识数组的最后一个元素,它的下标值等于数组中元素的个数。

下标值和下标表达式值不一定相同,例如:

DIMENSION A(10),B(10,10)

A(2)标识 A 的第二个数组元素,下标是(2),下标值为 2,其下标表达式的值为 2 是相等的。而 B(1,2)则标识 B 的第 11 个数组元素,下标是(1,2),下标值为 11,而下标表达式是 1 和 2,其值分别为 1 和 2。

Fortran 规定下标表达式仅能为下列形式之一:

I

V

V+I

V−I

I*V

I*V+J

I*V−J

其中:I、J 为整常量;V 为整变量。

因此,以下数组元素的下标是错误的:

A(I(2)), A(−K), A(I−2.0), A(2+I),A(I*3), A(+3*I), A(3+2*I),请读者自己找出其错误的原因。

为了指明数组的符号名和维说明,就要使用 DIMENSION 语句。其一般形式是:

DIMENSION a(d)[,a(d)]…

其中:每个 a(d)是一个数组说明符。

出现在 DIMENSION 语句中的每个符号名 a,说明 a 是该程序单元的一个数组。为使用户简化书写 Fortran 程序,允许数组说明符也可以出现在 COMMON 语句和类型语句中。但在一个程序单元中,作为数组名的一个符号名,以数组说明符的形式只允许出现一次。

DIMENSION 语句的例子如下:

DIMENSION A(10,20,30), B(70,35), C(10)

以上语句说明了三个数组:A 是一个三维数组,其数组元素个数是 10*20*30=6000 个;B 是一个二维数组,其数组元素个数是 70*35=2450 个;C 是一个一维数组,其数组元素个数是 10 个。由此可见,一个 DIMENSION 语句可以说明多个数组,它们之间用逗号分开即可。

以上介绍的是几个最常用的基本说明语句,用这几个说明语句就可来编制一些常用程序了。

二、赋 值 语 句

在任何科学计算中都不可避免的要进行计算,并要求保留在某处,这就需要赋值语句。显然,赋值语句是可执行语句。完成赋

值语句的执行导致定义一个实体。

Fortran 语言提供了四种赋值语句，即：

(1)算术赋值语句。

(2)逻辑赋值语句。

(3)语句标号赋值(ASSING)语句。

(4)字符赋值语句。

在上节表达式中对算术表达式和逻辑表达式讨论得比较多。在此处只简单地严格叙述一下，并给以例子。而语句标号赋值语句则在以后有关赋值 GOTO 语句时再介绍。

1. 算术赋值语句

算术赋值语句是最常用的赋值语句，其一般形式是：

V=e

其中：V 为整型或实型的变量名或数组元素名；e 为算术表达式。

执行算术赋值语句首先按运算规则求表达式 e 的值，并按表 1—9 的规则将 e 转换成 V 的类型，并将此结果赋给 V，使 V 有定义。

表 1—9

V 的类型	e 的类型	被赋的值
整型	整型	e
实型	实型	e
整型	实型	INT(e)
实型	整型	REAL(e)

表 1—9 中的 INT(e)和 REAL(e)是 Fortran 的内在函数，分别将 e 转换成整型或实型。

若已知各变量的值，则下式为算术赋值语句的一个简单例子：

```
PAY=(HOURS*RATE)—TEXT*(HOUR*RATE)
```

注意，HOURS 和 HOUR 是两个不同的变量名。

2. 逻辑赋值语句

逻辑赋值语句也是常用的一种赋值语句，其一般形式是：

V=e

其中：V 为逻辑变量名或逻辑数组元素名；e 为逻辑表达式。

逻辑赋值语句的执行是先求逻辑表达式 e 的值，然后将 e 的值赋给 V，使 V 有定义。而 e 必须有真值或假值。

若已知下列表达式中 A 和 B 的逻辑值，则以下的逻辑赋值语句就有确定的值：

```
C=A.AND.B
```

上式中的 A、B 和 C 都必须在程序的说明语句部分说明为逻辑型变量(用 LOGICAL 语句)，若 A 为.TRUE.，B 为.FALSE.，则赋予 C 的值为.FALSE.。

3. 字符赋值语句

字符赋值语句将一个字符串赋给一个变量。字符赋值语句的形式是：

```
V=e
```

其中：V 为字符变量名或字符数组元素名；e 为字符表达式。

执行字符赋值语句首先求表达式 e 的值，然后将 e 的值赋给 V，使 V 有定义。V 和 e 可以有不同的长度。若 V 的长度大于 e 的长度，则用空格字符向右延长 e，直至 e 和 V 的长度相等，然后赋值。若 V 的长度小于 e 的长度，则从右边去掉 e 的若干个字符，直到 e 和 V 的长度相等为止，然后再赋值。

字符赋值语句的例子如下：

```
CHARACTER LONG*10, SHORT*5
SHORT='ABCDE'
LONG=SHORT
```

以上语句执行的结果是 LONG 的值为 ABCDE，SHORT 的值为 ABCDE。

有关字符表达式的运算规则将在以后有关部分阐述。

三、控 制 语 句

控制语句是程序设计语言中必不可少的一种语句，它可按编制者的意图控制程序的执行顺序，相当于在程序中设立很多“开关”。Fortran 语言中提供了以下 16 种控制语句：

(1)无条件 GOTO 语句。
(2)计算 GOTO 语句。
(3)赋值 GOTO 语句。
(4)算术 IF 语句。
(5)逻辑 IF 语句。
(6)块 IF 语句。
(7)ELSE IF 语句。
(8)ELSE 语句。
(9)ENDIF 语句。
(10)DO 语句。
(11)CONTINUE 语句。
(12)STOP 语句。
(13)PAUSE 语句。
(14)END 语句。
(15)CALL 语句。
(16)RETURN 语句。

其中:块 IF 语句、ELSE IF 语句、ELSE 语句、END IF 语句、END 语句均为提出 Fortran 77 时才增加的。

1. 无条件 GOTO 语句

在结构程序设计中,认为转移语句是破坏程序结构的。因此,在结构化程序设计中主张尽量少用或不用转移语句。

Fortran 语言中提供了多种形式的转移语句。这里首先介绍无条件 GOTO 语句,其一般形式是:

```
GOTO S
```

其中:S 为与该无条件 GOTO 语句在同一个程序单元中的可执行语句的语句标号。

执行无条件 GOTO 语句的结果就是把控制转移去执行语句标号为 S 的语句。例如:

```
GOTO 107
```

则当程序执行到这个语句时,就立即转去本程序单元中语句标号

为 107 的语句。语句标号 107 的语句位置可在此转移语句的之前或之后。

2. 计算 GOTO 语句

计算 GOTO 语句提供了根据条件不同而选择不同的转移方向的方法。其一般形式是：

GOTO (S[,S]…)[,]i

其中：i 为整变量名；S 是和本计算 GOTO 语句在同一程序单元中的可执行语句的语句标号。

注意，同一个语句标号可以在同一个计算 GOTO 语句中出现多次(这意味着在多种情况下都需转向同一个可执行语句)。

执行计算 GOTO 语句引起控制转移，下一步将要执行语句标号表中第 i 个标号标识的语句，规定 $1\leqslant i\leqslant n$，其中 n 是语句标号表中语句标号的个数。若 $i<1$ 或 $i>n$，则顺序执行下去，即不产生控制转移。

例如：计算 n!，n=1,2,…,5，这 5 个 n! 的值。

由于我们还没学过更多的 Fortran 语句，我们现在就用计算 GOTO 语句来实现它。

```
      I=0
5     I=I+1
      GOTO (10,20,30,40,50), I
10    J1=1
      GOTO 5
20    J2=1 * 2
      GOTO 5
30    J3=1 * 2 * 3
      GOTO 5
40    J4=1 * 2 * 3 * 4
      GOTO 5
50    J5=1 * 2 * 3 * 4 * 5
      WRITE…(WRITE 语句将结果 J1，J2，J3，J4 和 J5 印出，从略)
```

当然，为计算 n! 的 5 个值，我们绝无必要编如此一个程序。但

如果每次需要计算的是一无规律而复杂的式子，我们就需根据条件来判断转向哪个语句去进行计算了。

3. 赋值 GOTO 语句

赋值 GOTO 语句和计算 GOTO 语句形式上稍有差异，需要仔细区分 i 是在语句标号表之前或之后，作用也很类似，但主要的区别是赋值 GOTO 语句永远和语句标号赋值（ASSIGN）语句一起连用，它们必须是同时出现的。因此先给出语句标号赋值（ASSIGN）语句的一般形式是：

ASSIGN S TO i

其中：S 为语句标号；i 为整变量名。

执行 ASSIGN 语句后就是把语句标号 S 赋给整变量 i。此语句标号必须是可执行语句的标号或 FORMAT 语句的标号。

用语句标号值定义的变量不允许作为非语句标号量来引用，另一方面，在赋值 GOTO 语句中引用的语句标号必须用语句标号赋值（ASSIGN）语句来定义。

赋值 GOTO 语句的一般形式是：

GOTO i[[,](S[,S]…)]

其中：i 为整变量名；S 为与赋值 GOTO 语句在同一程序单元中的可执行语句的语句标号。

注意，同一个语句标号可以在同一个赋值 GOTO 语句中多次出现。

执行赋值 GOTO 语句时，变量 i 必须是用同一程序单元中出现的可执行语句的语句标号值定义的。此变量只可通过与该语句在同一程序单元内的 ASSIGN 语句用语句标号值定义。执行赋值 GOTO 语句将使控制转移，转移到该语句标号 i 标识的语句去。

若赋值 GOTO 语句中存在括号括着的表，则赋给 i 的语句标号必须是表内的一个语句标号。

下面举例比较计算 GOTO 语句和赋值 GOTO 语句的不同用法（看看如何使它们转移到同一语句标号去）。采用

ASSIGN 101 TO K

```
    …
GOTO K，(10,20,101,204)

    …
```
与采用计算 GOTO 语句
```
K=3
  …
GOTO (10,20,101,204),K
  …
```
具有相同的效果。

以上两种 GOTO 语句达到同一目的，但要注意 ASSIGN 语句中的 K 决不可作为一般变量去引用或参与运算。

4. 算术 IF 语句

算术 IF 语句是根据算术表达式的结果来决定程序执行的转移。其一般形式是：
```
IF (e) S1,S2,S3
```
其中：e 为整型或实型表达式。

S1、S2、S3 中的每一个都是与本算术 IF 语句在同一程序单元中的可执行语句的语句标号。同一个语句标号可在同一个算术 IF 语句中出现多次。

执行算术 IF 语句首先求表达式 e 的值，然后判 e 的值，若 e 小于零则转去执行 S1，若 e 等于零则转去执行 S2，若 e 大于零则转去执行 S3。

例如，可用算术 IF 语句编制一个程序段实现下列分段定义的函数：

$$Y=\begin{cases} -X^2 & X<0 \\ 0. & X=0 \\ X^2 & X>0 \end{cases}$$

可用
```
      …
      IF (X) 5,10,15
5     Y=-X**2
```

```
      GOTO 20
10    Y=0.
      GOTO 20
15    Y=X**2
20    ...
```

实现。

由此可见，利用表达式的值可决定程序转移的不同方向，这是一个很常用的控制语句。

5. 逻辑 IF 语句

逻辑 IF 语句是根据所含逻辑表达式的值决定程序转移的方向，我们可以想到，由于逻辑表达式的值只能有真值和假值两种情况，因此转移也就只有两种可能的方向。

逻辑 IF 语句的一般形式是：

```
IF (e) st
```

其中：e 为逻辑表达式；st 为一个可执行语句，但不能是 DO 语句、块 IF 语句、ELSE IF 语句、ELSE 语句、END IF 语句、END 语句或另一个逻辑 IF 语句。

执行逻辑 IF 语句先求逻辑表达式 e 的值，若 e 的值为真，则执行语句 st，若 e 的值为假，则不执行语句 st，而继续顺序执行。

例如，用逻辑 IF 语句编制的一个程序段：

```
      ...
IF (X.GE.0.0) Y=X**2
IF (X.LT.0.0) Y=-X**2
      ...
```

将实现分段定义的函数：

$$Y=\begin{cases}-X^2 & X<0\\ X^2 & X\geqslant 0\end{cases}$$

实际上这个算术式与上面的相同，但逻辑 IF 语句只有两个转移方向，因此只能表示两种可能性的选择。

6. DO 语句

任何科学计算中循环都是不可缺的一种表达方法，在 Fortran

语言中自然也要引进有关循环的语句。Fortran 中用 DO 语句来指明一个循环，一般称为一个 DO 循环。DO 语句的一般形式是：

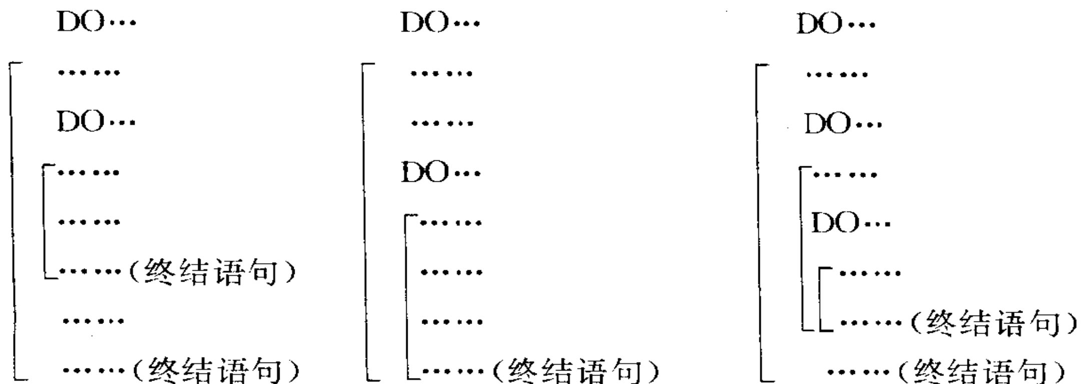
DO S〔,〕i=e1,e2,〔,e3〕

其中:S 为一个可执行语句的语句标号，由它标识的语句称为此 DO 循环的终结语句(它与此 DO 语句必须在同一程序单元中，且按语句的编写顺序必须在此 DO 语句的后面);i 为一整变量名，称为 DO 变量;e1、e2、e3 为整常数或整变量名，它们分别建立 DO 循环中的初值参数、终值参数和增值参数。

DO 循环的终结语句不能是无条件 GOTO、赋值 GOTO、算术 IF、块 IF、ELSE IF、ELSE、END IF、RETURN、STOP、END 或 DO 语句。若 DO 循环的终结语句是逻辑 IF 语句，则它可以包含除以下语句外的任何可执行语句，即 DO、块 IF、ELSE IF、ELSE、END IF、END 或另一个逻辑 IF 语句。

DO 循环的范围也叫 DO 的循环体，它由指明此 DO 循环的 DO 语句后紧跟的语句开始，直到并包括此 DO 循环的终结语句在内的所有可执行语句构成。

若一个 DO 语句出现在某个 DO 循环的范围中，则由该 DO 语句指明的 DO 循环的范围必须整个被包含在外面的 DO 循环范围中。多个 DO 循环可以有同一个终结语句。这种循环包含循环的情形被称为嵌套的 DO 循环。嵌套的 DO 循环可具有以下的形式：



若 DO 语句出现在 IF 块、ELSE IF 块或 ELSE 块中，则 DO 循环的范围必须分别地整个被包含在 IF 块、ELSE IF 块或 ELSE

块中。

若块 IF 语句出现在一个 DO 循环的范围中，则相应的 END IF 语句也必须出现在该 DO 循环范围中。

执行 DO 语句将依次完成下面的步骤：

(1)计算 e1、e2 和 e3 分别建立初值参数 m1、终值参数 m2 和增值参数 m3。若不出现 e3，则认为 m3 的值为 1，m3 的值不能是零。

(2)DO 变量用初值参数 m1 的值定义。

(3)建立重复次数，且重复次数为下列表达式的值：MAX0((m2－m1＋m3)/m3)，0)，当 m1＞m2 且 m3＞0 或 m1＜m2 且 m3＜0 时，重复次数总是零。

完成以上步骤之后，再进行循环的控制处理。循环控制处理决定是否需要执行 DO 循环体。其步骤为：

开始执行 DO 循环体的第一个语句直到此循环体的终结语句。然后再进行下述增值处理，然后再作如下的判断：

若重复次数是零，则执行紧跟该终结语句后的下一个可执行语句，否则再去执行循环体。

增值处理的步骤如下：

(1)选择刚执行过的 DO 语句的 DO 变量、重复次数和增值参数。

(2)DO 变量的值加上增值参数 m3 的值。

(3)重复次数减 1。

(4)继续转去执行重复次数减少后的 DO 循环的循环控制处理。

我们看一下以下的例子：

```
N=0
DO  100  I=1,10
J=I
DO  100  K=1,5
L=K
```

```
100   N=N+1
101   CONTINUE
```

以上这些语句执行之后，在执行 CONTINUE 语句时，I=11，J=10，K=6，L=5 和 N=50。再对比以下的例子：

```
      N=0
      DO 200 I=1,10
      J=I
      DO 200 K=5,1
      L=K
200   N=N+1
201   CONTINUE
```

这些语句执行后，在执行 CONTINUE 语句时，I=11，J=10，K=5，L=5 和 N=10。这表明内层循环只执行了一次。

但在 Fortran 77 中定义有所不同，它首先检查重复次数，若不是零则正常执行，若是零则不执行循环体，所以一开始若循环重复次数为零，则循环体一次也不执行。而 Fortran Ⅳ 则循环体至少执行一次。

于是在 Fortran 77 中，第二个例子在执行 CONTINUE 语句时，I=11，J=10，K=5，N=0，L 未定义。

这一点差别很重要请读者充分注意。

应用中，循环语句往往和数组连用，即在循环时对数组中的数组元素进行循环地处理。

7. CONTINUE 语句

CONTINUE 语句的作用相当于一条空指令。其一般形式是：

CONTINUE

CONTINUE 语句执行之后不产生任何作用。那么，什么样的情况下我们才需要用此语句呢？在某些情况下，DO 循环用其它语句作终结语句不合适时，则用 CONTINUE 语句作为终结语句，以便使 DO 循环可做增值处理。另外，也可以不管 DO 循环有无终结语句，都统一格式以 CONTINUE 语句作为终结语句。

我们看一下以下例子：

```
      Y=1.0
      DO  10  I=1,100
      Y=Y+5.
20    ...

10    IF(55.-Y) 20,20,10
```

这显然是一个错误的程序段，因为 DO 循环的终结语句不能是算术 IF 语句，这时就可用一个 CONTINUE 语句来解决此问题：

```
      Y=1.0
      DO  10  I=1,100
      Y=Y+5.
20    ...

      IF (55.-Y) 20,20,10
10    CONTINUE
```

类似地，在逻辑 IF 语句中也有许多语句是不允许出现的，此时也可用 CONTINUE 语句来解决问题。

8. STOP 语句

STOP 语句的一般形式是：

STOP [n]

其中：n 是一个不多于 5 位的数字串，或是一个字符常数。

执行 STOP 语句将结束可执行程序。在程序结束时，数字串或字符常数是可见的，以便用户可鉴别是在什么地方停止执行的。

每一个主程序结束时都有一个 STOP 语句。

STOP 3321

9. PAUSE 语句

PAUSE 语句是过去机器速度不高、独享整个系统资源而遗

留下来的产物，在 Fortran 90 中把它列在 Fortran 的过时的功能内，亦即在 Fortran 90 之后的 Fortran 语言中将把它淘汰。

PAUSE 语句的一般形式是：

PAUSE [n]

其中：n 是一个不多于 5 位的数字串，或是一个字符常数。

PAUSE 语句的执行将使可执行程序暂停执行。而执行必须是可恢复的。在执行暂停时，数字串或字符常数是可见的。执行的恢复不是在本程序的控制下完成的，一般需外来的干预。若执行恢复，就像执行 CONTINUE 语句一样，继续正常的执行序列。例如：

PAUSE 4321

10. END 语句

每个程序单元的最后一行必须是 END 语句，END 语句指出程序单元的语句和注解行序列的结束。

END 语句的形式是：

END

END 语句只能写在始行的 7 至 72 列上。END 语句不得有续行，它必须写在语句始行上。在一个程序单元内，字符序列 END 不能作为其它语句的始行。在 Fortran Ⅳ 中称为 END 行且没有下面所述的功能，而在 Fortran 77 中称为 END 语句且有下面所述的功能。

若在一个函数辅程序或子程序辅程序内执行 END 语句，它的作用和 RETURN 语句相同。若在主程序内执行 END 语句，它结束可执行程序的执行。

四、简单输入/输出语句

在任何实用程序的编制中都离不开输入/输出，而甚至在某些程序中，输入/输出语句可能占到一个很大的比例。最基本的输入/输出语句可分为带格式语句或不带格式语句两种。格式语句又可以有多种编辑描述符。这里只介绍最简单、最常用的输入/输出语句。

1. FORMAT 语句

格式语句和格式输入/输出语句共同使用，为输入/输出格式提供一种信息，这种信息指出了在内部表示和文件中的记录或记录序列的字符串之间的一些编辑法则。

FORMAT 语句是非执行语句，它的作用是确定数据依照 Fortran 语言中规定的哪一种格式进行输入/输出（它必须由输入/输出语句所引用）因此该语句一定带有语句标号，而其全部编辑信息都要用括号括起来。其一般形式是：

FORMAT ([flist])

其中：flist 是一个编辑描述符的表，提供各种编辑信息。

关于编辑描述符，其种类是很多的，而且用于输入语句和用于输出语句时含义是不同的。

在输出语句中常用的编辑描述符为：

Iw　　（整型输出）

Fw.d　（小数型输出）

Ew.d　（指数型输出）

其中：w 为数值所占宽度，即字段宽度，意味着整个数值应占多少个字符位置。d 为小数部分所占的宽度，即不包括小数点在内，小数点后取几位。w 和 d 均为整型常数。

以下举例说明在各种编辑描述符情况下，计算机内数值和相应输出结果的对照：

机内数值	编辑描述符	输出结果	注解
12345	I6	12345	右对齐，左补空格
12345	I5	12345	
+12345	I6	+12345	
12345	I3	* * *	出错，w 位数太少
123.45	F6.2	123.45	
-123.45	F8.3	-123.450	
123.45	F6.1	123.5	多余小数四舍五入
123.45	F5.2	* * * * *	出错，w 位数太少
123.45	E12.5	0.12345E 03	

123.45	E12.4	0.1235E 03	多余部分四舍五入
0.0012345	E12.4	0.1235E−02	
123.45	E5.2	* * * * *	出错，w 位数太少

由上可见，当输出的数值不足 w 位时，字符向右对齐，左边高位补上空格字符。如果数值所需位数多于 w，则给出错误信息，但此出错信息的格式可由各自的编译程序定义。

实型数据的输出格式可用 F 型格式，也可用 E 型格式。另外，在实用中为了输出美观、间距一致，E 型或 F 型输出的 w 宽度都留得比需要稍大一些，这样可在一行中分别打印 5 列、6 列等数据。

用 F 型输出比较直观，可令小数点位数一致，各行小数点对齐，但对打印太大或太小的数有时不能正确印出。用 E 型输出时可保留所需的有效位数，不会丢失有效数位，但直观性较 F 型输出差。在选 w 时要有余量，不能用几位就设定几位，并要留出正负号和小数点的位置。F 型输出时为 Fw.d，d 为小数部分所占位数。由于小数点占一位，因此留给整数部分只有 $w-(d+1)$ 位，且正负号还需要占有一位。若整数部分有 k 位，则 $w \geq k+d+2$，这样的 w 才是保险的。

E 型输出时为 Ew.d，指数部分占了 4 位(即 E、正负号、两位表示指数值)，在有效数字部分有一个零、一位小数点以及一位正负号，故应选取 $w \geq d+7$，这样的 w 才是保险的。

在 I 型、E 型、F 型编辑描述符中都允许带有重复因子 r。也就是说，如果 FORMAT 语句中需要连续几个同样的编辑描述符，且其 w 与 d 都相同，则只需写一个，前面只需加上其重复因子 r。例如：

rIw 相当于 r 个 Iw：$\underbrace{\text{Iw,Iw,}\cdots\text{,Iw}}_{r\text{个}}$

rFw.d 相当于 r 个 Fw.d：$\underbrace{\text{Fw.d,Fw.d,}\cdots\text{,Fw.d}}_{r\text{个}}$

rEw.d 相当于 r 个 Ew.d：$\underbrace{\text{Ew.d,Ew.d,}\cdots\text{,Ew.d}}_{r\text{个}}$

在输出上还有 X 型和 H 型编辑描述符也是较常用的。

X 型编辑描述符是表示空格的，nX 即表示连续输出 n 个空格。常常用 X 型编辑描述符在打印中设置必要的空格，如在打印标题或表格中常用 X 型编辑描述符。H 型编辑描述符是用来表示字符型字符的，如打印标题即需要 H 型编辑描述符。它的形式是 $nHh_1h_2\cdots h_n$。即需要输出 n 个字符，在 H 前写 n，H 后把需输出的 n 个字符内容全部列出，如需输出夹在字符中的空格时，空格也占一个字符的位置。所表示的字符可以是载体计算机所能表示的任何字符。

我们看一下 H 型编辑描述符的例子：

18HTHIS IS AN EXAMPLE

从上例可以看到空格也要占一位置。

在输入语句中常用的编辑描述符是：

Iw　　(整型输入)

Fw.d　　(实型输入)

其中：w 为数值所占宽度。d 为小数部分所占的宽度。

在输入时，E 型编辑描述符和 F 型编辑描述符的作用相同，故不再说明。

下面举例说明外部数值经过编辑描述符作用之后输入到计算机内是什么值：

外部数值	编辑描述符	机内数值	注解
1234	I5	1234	
1234	I5	12340	
12345	I4	1234	w 小，只取 4 位
−1234	I5	−1234	
−8,120	I6	−8	用逗号将逗号后位数去掉
1234	F7.3	123.400	取 3 位为小数部分
−123.4	F7.4	−123.4	自带小数的数，d 不起作用
123456789	F7.3	1234.567	w 不够大，只取 7 位

12.34E+02	F10.3(或 E10.3)	1234	原数据带小数点并带指数部分，则乘以指数得机内数值
1234E－02	F10.2(或 E10.2)	0.1234	原数据无小数点，先按 d，取 d 位小数再乘以指数得机内数值

和用于输出语句的编辑描述符一样，它们也同样可以有重复因子 r，如：

rIw

rFw.d

rEw.d

当在输入语句中采用 X 型编辑描述符时，nX 表示跳过输入记录中相应的 n 个字符，因此在使用中需要极为注意。例如，有以下 FORMAT 语句。

```
FORMAT (I3,4X,IR,5X,I5,6X,I6)
```

而外部记录的形式如下：

```
123,45,6789,012,34567,8901,234567
|3||4|| 4||5 || 5 || 6 || 6 |
I3  4X I4  5X  I5    6X  I6
```

因此，送入机内的记录为 123、6789、34567、234567，而 45、012、8901 及其前后的逗号未送入机内。由此可见，在输入时用 X 型编辑描述符要很小心。

打印输出时，在控制打印纸的换行、换页方面有一些特殊规定如表 1－10 所示：在行式打印机上输出时，记录的第一个字符不打印，作为控制打印机的走纸控制符。

表 1－10

记录的第一个字符	作　　用
空格	在下一行打印当前记录(下行打印)
0	空推一行再打印当前记录(隔行打印)
1	换页后打印当前记录(换页印)
+	纸不动，当前记录和上一记录打印在同一行(重印)

为使记录的第一个符号是以上四种走纸符，可在格式说明中括入 1H（或 1X）、1H0、1H1、1H＋等含一个字符的 H 型编辑描述符，也可把走纸符合并在其它编辑描述符中。

由上可知，在格式语句左括号之后的第一个 X 型编辑描述符“1X”并不输出一个空格，而是做为打印机的走纸控制符，如上表所述。因此在写格式说明时要注意第一个编辑描述符的设置，避免设置其它字符而引起打印机控制走纸失误。

在格式说明时，编辑描述符之间用逗号或斜线分开，它们被称之为字段分隔符。

用“，”表示其前后两个编辑描述符所对应的字符属于同一记录。对于输出，则表示应打印在同一行上。对于输入，则表示属同一记录。用“/”表示结束当前所处理的记录，并开始处理一个新记录。对于输出，则表示打印当前一行的同时开始组织新的一行的打印。对于输入，则表示已处理完上一记录并读入下一记录。若连续用 n 个“/”，表示需处理 n－1 个空记录。对于输出则打印 n－1 个空行，对于输入则跳过 n－1 个记录。

介绍了一些基本编辑描述符之后，现在我们来给出 FORMAT 语句的一个例子：

```
20  FORMAT (1X,I4,3X,F7.3,3X,I4)
```

FORMAT 语句一定要有语句标号，20 为其语句标号。若此语句由输出语句所引用，则 1X 为走纸控制，相应第 1 个数据按 I4（整型）输出，空 3 个字符位置，第 2 个数据按 F7.3（实型）输出，空 3 个字符位置，第 3 个数据按 I4（整型）输出。若还有数据要输出，仍用此语句，从最右的一个左括号开始，周而复始。若此语句由输入语句所引用，则相应记录空第 1 个字符，按 I4 输入 4 个字符（整型），再空 3 个字符，按 F7.3 输入 7 个字符（实型），再空 3 个字符，按 I4 输入 4 个字符（整型），若还有记录，按以上规定从最右的一个左括号开始，周而复始。在此格式语句中就是从头进行格式控制。

FORMAT 语句以及编辑描述符的种类比较复杂，用起来也

很灵活，这里介绍的是最基本的。

2. READ语句

任何由计算机处理的信息，首先我们都要通过输入的手段把所需的原始数据输入计算机，因此就要和外部设备发生关系，需要考虑将此原始数据放在哪个外部设备上，以及数据应按什么格式输入等问题。

READ语句是数据传输的输入语句，它要说明被传输数据来自何处、送至何处、执行顺序、按什么格式进行编辑加工等。

READ语句的一般形式是：

READ(u,f)l

其中：u输入设备的设备号或通道号；f为与此读语句同一程序单元内的某格式语句的语句标号；l为输入变量名表。

执行READ语句之后，输入变量名表中的变量被赋值，从输入设备u上读入记录，按输入变量名表中变量出现的顺序和f给出的相应格式，对输入字段进行加工并转换成所需类型，最后才依次赋给表中相应的各个变量。

下面给出READ语句的一个例子：

```
      READ (4,10) I,X,J
  10  FORMAT (I5,F14.5,I4)
```

执行此READ语句将从输入设备4上按照格式语句10的编辑描述符读入3个数据，5位整数赋给I，14位实数赋给X，4位整数赋给J。若I、J都要求赋给5位整数，而我们准备数据时也按此要求准备时，可用以下READ语句：

```
      READ (4,20) I,J,X
  20  FORMAT (2I5,F14.5)
```

3. WRITE语句

任何由计算机处理的信息，最终我们都要通过输出的手段把处理结果输出，这样必然要与外部设备发生连系，并且考虑处理结果应从哪个输出设备上输出，以及数据或文字应按什么格式输出等问题。

WRITE 语句是数据传输的输出语句，它要说明被传输的数据送至何处、从何处取数、执行顺序、按什么格式进行编辑加工等。

WRITE 语句的一般形式是：

```
WRITE (u,f) l
```

其中:u 为输出设备的设备号或通道号;f 为与此写语句处在同一程序单元内的格式语句的语句标号;l 为输出变量名表。

执行 WRITE 语句之后,输出变量名表中的各项值将被输出。按它们在表中出现的顺序和语句标号给出的相应格式,在输出设备 u 上输出结果,并要对输出字段进行编辑加工。

以下给出 WRITE 语句的一个例子：

```
      WRITE (5,30) X,Y,Z
 30   FORMAT (1X,F12.5,F14.5,F12.5)
```

我们知道 1X 是打印机走纸控制字符,X、Y、Z 各按 F12.5、F14.5、F12.5 的实型格式输出。如果 X、Y、Z 都按同一实型格式输出,则可把格式语句简化,例如：

```
      WRITE (5,40) X,Y,Z
 40   FORMAT (1X,3F14.5)
```

当然,我们也可在输出打印时表明哪些是 X 值、哪些是 Y 值、哪些是 Z 值。例如：

```
      WRITE (5,50) X,Y,Z
 50   FORMAT (1X,2HX=,F14.5,2HY=,F14.5,2HZ=,F14.5)
```

这时将在输出 X 值之前，先输出 X=，在输出 Y 值之前，先输出 Y=，在输出 Z 值之前，先输出 Z=，这样输出的形式就显得很直观而明了。

在很多 Fortran 编译程序中,都设定输入设备号为 4,输出设备号为 5。不过在你使用某台计算机时,请注意使用说明书上有关的说明。

输入/输出部分是 Fortran 语言中变化很多的部分,它可给用户提供各种情况下使用不同的格式的可能性,但对最基础部分我

们暂时先介绍以上这些。用以上的介绍，我们已可以开始编制较完整的 Fortran 源程序了。

五、过程和过程引用

在本章第二节中介绍了有关过程的定义及过程引用的方法，这里不再赘述。但在这里我们要简要介绍和过程有关的两个说明语句，即 EXTERNAL 语句及 INTRINSIC 语句，并讨论 ENTRY 语句，以利进一步深入学习过程和过程引用。

1. EXTERNAL 语句

在 Fortran 语言中，在一个可执行的程序单元里，如果外部过程名在 CALL 语句或函数引用中做为实元使用时，则在这个程序单元里，该外部过程名必须通过外部语句说明。

因此，EXTERNAL 语句用于指出一个名字，它标识一个外部过程或虚拟过程，并且允许使用这样一个名作为实元。

EXTERNAL 语句的一般形式是：

```
EXTERNAL proc[,proc]…
```

其中：每一个 proc 都是外部过程名或虚拟过程名。

一个名字在 EXTERNAL 语句中出现就说明该名字是一个外部过程名或虚拟过程名。若在一个程序单元内，外部过程名作为实元使用，则它必须在该程序单元内的 EXTERNAL 语句中出现。注意，语句函数名不得出现在 EXTERNAL 语句中。若在一个程序单元内，一个内在函数名出现在 EXTERNAL 语句中，则该名字就变成了某个外部过程名，而在该程序单元内，同名的内在函数就不能被引用了。

一个名字在一个程序单元的所有 EXTERNAL 语句中只允许出现一次。

我们从下面的例子中可看到 EXTERNAL 语句的作用（如在主程序中包含）：

```
    …
  EXTERNAL SIN,COS
```

```
      …
CALL FAX (A,SIN,B)
      …
CALL FAX (A1,COS,B1)
      …
```

相应地至少应有三个外部辅程序：FAX、SIN 和 COS，因为 SIN 和 COS 虽为标准的内在函数，但它们的名已出现在 EXTERNAL 语句中，因而在这个主程序内它们就不是标准的内在函数了，需要对它们重新定义，而 FAX 显然是一个具有三个虚元的子程序，其中一个虚元是过程。例如：

```
SUBROUTINE FAX (X,F,Y)
EXTERNAL F
Y=F(X)
RETURN
END
FUNCTION SIN (X)
      …
RETURN
END
FUNCTION COS (X)
      …
RETURN
END
```

当然，EXTERNAL 语句中的外部过程既可是以 SUBROUTINE 语句为首的子程序，也可是以 FUNCTION 语句为首的函数。

2. INTRINSIC 语句

INTRINSIC 语句的作用和 EXTERNAL 语句有某些类似之处。它主要用于指出某个名字是一个内在函数的特定名，并允许作为一个实元使用。

INTRINSIC 语句的一般形式是：

```
INTRINSIC fun[,fun]…
```

其中：每个 fun 都是内在函数名。

一个名字出现在 INTRINSIC 语句中就说明该名字是内在函数名。若一个内在函数的特定名在一个程序单元内作为实元，则它必须出现在该程序单元的一个 INTRINSIC 语句内。值得注意，用于类型转换（INT、IFIX、IDINT、FLOAT、SNGL、REAL、ICHAR）、字序关系（LGE、LGT、LLE、LLT）和选取最大值或最小值（MAX0、AMAX1、AMAX0、MAX1、MIN0、AMIN1、AMIN0、MIN1）的内在函数名不得作为实元使用。

只允许一个名字在一个程序单元的所有 INTRINSIC 语句中出现一次。注意，在一个程序单元内，一个名字不允许既出现在 EXTERNAL 语句中又出现在 INTRINSIC 语句中。

下面举例说明内在函数如何作为实元使用。若在一个程序单元内有：

```
      ...
  INTRINSIC SIN,COS
      ...
  CALL FAY (A1,SIN,B1)
      ...
  CALL FAY (A2,COS,B2)
      ...
```

SIN 和 COS 在调用子程序 FAY 时作为实元出现，它们又在 INTRINSIC 语句中说明。因此，我们知道作为实元使用的 SIN 和 COS 是系统提供的内在函数的特定名。所以，这里只需提供子程序 FAY 的定义即可。例如：

```
  SUBROUTINE FAY (X,F,Y)
  EXTERNAL F
  Y=F(X)
  RETURN
  END
```

由以上两个例子的比较，我们就可发现同一个名不允许既出现在 EXTERNAL 语句中又出现在 INTRINSIC 语句中，这样将

出现不可克服的矛盾。

3. RETURN 语句

前面已多次使用 RETURN 语句，这里我们将给出它的有关说明。RETURN 语句使控制返回到正在引用它所在的程序单元去。它只可出现在函数辅程序或子程序辅程序中。

RETURN 语句的形式是：

RETURN

执行 RETURN 语句，就结束了函数辅程序或子程序辅程序的引用，并返回到引用它们的程序单元去。辅程序可包含一个以上的 RETURN 语句，但辅程序并不一定要包含 RETURN 语句。执行函数辅程序或子程序辅程序内的 END 语句与执行辅程序内的 RETURN 语句作用相同。

4. ENTRY 语句

ENTRY 语句为函数辅程序或子程序辅程序提供多个入口点。这样，我们在引用辅程序时可以只引用其一部分而非全部。

ENTRY 语句允许过程引用从该 ENTRY 语句所在的函数辅程序或子程序辅程序中的某个特定的可执行语句开始。它可出现在函数辅程序内的 FUNCTION 语句之后的任何地方，也可出现在子程序辅程序内的 SUBROUTINE 语句之后的任何地方，但不能出现在程序的块 IF 语句和它的相应 END IF 语句之间或 DO 语句与它的 DO 循环终结语句之间。

辅程序可有一个或多个 ENTRY 语句。ENTRY 语句是非执行语句。

ENTRY 语句的一般形式是：

ENTRY en[([d[,d]…])]

其中：en 为函数或子程序辅程序内入口的名字，称为入口名（若 ENTRY 语句出现在子程序辅程序中，则 en 是外部子程序名。若 ENTRY 语句出现在函数辅程序中，则 en 是外部函数名）；d 为变量名、数组名或虚拟过程名，也可是星号；d 为 ENTRY 语句的虚元，而且星号只允许出现在子程序辅程序内的 ENTRY 语句中。

对于子程序辅程序的入口名可没有虚元，也可有虚元，引用方法是 CALL en( )或 CALL en 这两种形式。

函数辅程序内的 ENTRY 语句中的入口名标识了可执行程序内的一个外部函数，并可作为外部函数引用。子程序辅程序内的 ENTRY 语句中的入口名标识了可执行程序中的一个子程序，并可作为子程序引用。

当入口名 en 被用于过程引用时，该过程的执行是从跟在以 en 为入口名的 ENTRY 语句后的第一个可执行语句开始的。

除包含入口名的 ENTRY 语句所在的程序单元内不能引用该入口名外，在可执行程序的任何程序单元内，均可引用该入口名。

现举例说明 ENTRY 语句的使用：

```
SUBROUTINE ROOT (X1,X2,A,B,C,Y)
D=SQRT(B * * 2-4.0 * A * C)
X1=(-B+D)/(2.0 * A)
X2=(-B-D)/(2.0 * A)
ENTRY SUMT (A,B,C,Y)
Y=A+B+C
RETURN
END
```

从上例可以看出，如果我们只想求 A、B、C 的和，则程序就可从 SUMT 开始。如果又想求和，又想求二次方程的两个根，则程序就从子程序的开头开始。当然，这是一个极简单的例子，在辅程序的多入口点这个问题中，我们可以把复杂的问题按需要分解得非常具体。

本节简单地介绍了 Fortran 语言中使用的最基本的语句，使用这些语句，我们已经可以着手编制一些较简单但是完整的 Fortran 程序了，而且也许甚至可以做一些大型的科学计算项目了。

第四节 Fortran 程序

以上几节介绍了组成 Fortran 语言的各种语言元素以及 Fortran 语言中的最基本的语句。我们最终的目的还是要把它们有机地结合在一起构成一个真正能在计算机上运行的 Fortran 源程序。本节将讨论 Fortran 源程序是怎样构成的，以及源程序之间的关系怎样等问题。

一、主　程　序

Fortran 语言所编写的程序是块结构的，即它们由一个或数个相对独立的程序段组成，这种结构在高级程序设计语言发展过程中显示了它的优越性，由于其相对独立的几段程序都可以分开调试再连接组装，所以对大型程序的编制调试带来了极大的便利。

在 Fortran 语言中，这种相对独立的程序段称为一个程序单元。其定义为：程序单元由语句和任选注解行的序列组成。一个程序单元可以是一个主程序、也可以是一个辅程序。

主程序是没有 FUNCTION 语句或 SUBROUTINE 语句作为其第一个语句的程序单元，它可以有一个 PROGRAM 语句作为其第一个语句。在很多编译程序中主程序可以没有 PROGRAM 语句作为其第一个语句。

在一个可执行程序中必须有而且仅有一个主程序。一个可执行程序的执行是从执行主程序的第一个可执行语句开始的。

PROGRAM 语句的形式是：

```
PROGRAM pgm
```

其中：pgm 为出现 PROGRAM 语句时的主程序名。

PROGRAM 语句不一定在一个可执行程序中出现，若出现的话，则必须是主程序中的第一个语句。pgm 是可执行程序的全局名，而且在该可执行程序中，禁止它和外部过程名和公用块名相同。pgm 也不能和主程序内的任何局部名相同。PROGRAM 语句

只能在主程序中作为第一个语句出现，除 FUNCTION、SUBROUTINE 或 RETURN 语句外，主程序可以包括任何其它语句。主程序不能被辅程序引用，也不能被它自己引用。

Fortran 程序可以只由一个主程序组成，或者可以由一个主程序和若干个辅程序组成，即一个 Fortran 程序必须含有一个主程序单元。Fortran 程序的结构可为：

```
PROGRAM ABC
…                主程序
END
SUBROUTINE A1
…                第一个辅程序
END
…
…                第 n 个辅程序
END
```

二、辅　程　序

辅程序是以 FUNCTION 语句或 SUBROUTINE 语句作为第一个语句的程序单元。严格地说，第一个语句是FUNCTION 语句的辅程序称为函数辅程序，第一个语句是 SUBROUTINE 语句的辅程序称为子程序辅程序。函数辅程序和子程序辅程序统称过程辅程序。

在前面的介绍中我们知道虚拟过程是标识一个过程的虚元。那么什么是虚元呢？什么是虚元和实元的结合呢？在例子中我们曾看到过一些，但它们严格的定义是什么呢？在 Fortran 中我们离不开实元和虚元的问题。

1. 虚　　元

在子程序辅程序和函数辅程序中都可看到它们在辅程序名之后带有一个由括号括起来的虚元名表。

语句函数、函数辅程序和子程序辅程序利用虚元指出实元的类型，指出每个变元是单个值、数组值或过程。其中语句函数虚元

仅可以是变量。

虚元可分为变量、数组或虚拟过程几类。与虚元名有相同类别、类型以及实元名可以出现的所有地方，除了某些明确指出禁止出现虚元名的地方外，相应的虚元名均可出现。

下面给出虚元的一个例子：

SUBROUTINE ABC (X,Y,SUB)

以上的 X、Y、SUB 就是虚元，通常在该子程序中会进一步用语句说明它们的类型和性质。

2. 实　元

在引用子程序或函数时，虚元表中的各个虚元就必须由特定的实际数值、变量和过程等所代替。为了便于表述引进了实元的概念。

对子程序或函数的一次特定的引用而言，实元说明了与虚元结合的实体是什么。在包含有该引用的程序单元中的语句函数名不能作实元，并且仅当与实元相结合的虚元是一个变量，且它在被引用的外部过程执行期间不被定义时，实元才可以是常数、函数、包含运算符的表达式。以及括在括号中的表达式。

除了当实元是子程序名外，每个实元的类型必须与它相结合的虚元的类型一致。

下面给出实元的一个例子如：

CALL ABC (E+F,E*F,COS)

以上的 E+F、E*F 都是实元，分别与上述的虚变量 X 和 Y 相结合，而上述虚元 SUB 对应的实元是这里的一个函数过程 COS。

3. 虚元和实元的结合

在实元的例子中可以看到，在引用过程中，就有用实元代替虚元的过程，这样才使得引用真正有意义，这就是虚元和实元结合使用的缘故。

执行函数引用或子程序引用时，就在对应的虚元和实元之间建立了结合关系。第一个虚元和第一个实元相结合，第二个虚元和

第二个实元相结合，等等。在执行一个函数或子程序的引用时，被引用的过程名的虚元表中出现的虚元，它在函数辅程序或子程序辅程序中的所有出现都要和实元相结合。仅当实元的类型和对应的虚元的类型相同时，才实现合法的结合。子程序名没有类型，它一定和虚拟过程名相结合。若实元是一个表达式，则在进行变元结合之前求该表达式的值；若实元是外部过程名，则该过程必须在对其引用的执行期间是可用的；若虚元当前并未和实元结合，则它是无定义的。

变元的结合可能通过多级过程引用进行。仅当在所有中间各级都存在合法结合关系时，最后一级才存在合法的结合关系。当执行到程序单元中的 RETURN 语句或 END 语句时，该程序单元中的变元结合关系就终止了。要特别注意，在辅程序的一次引用和该辅程序的下一次引用之间并不保持变元的结合关系。

由于主程序不能被其它程序所引用，因此显而易见地主程序必然不会在程序名之后跟有虚元表。

三、程　　序

在介绍了主程序和辅程序之后，我们对于 Fortran 程序的块结构有了一个初步的概念，但程序单元中语句的排列应是什么样的?这一点在很多种程序设计语言中都有比较严格的规定，尤其是早期的程序设计语言更是如此。然而，今天的 Fortran90 已不太拘泥于这种固定格式了。

这里我们要介绍一个程序单元内语句的顺序应如何排列，以及一个编制程序时经常会用到的特殊的非执行语句——DATA 语句。

1. 程序单元中的语句顺序

PROGRAM 语句仅能作为主程序的第一个语句出现。辅程序的第一个语句不是 FUNCTION 语句就是 SUBROUTINE 语句。

从表 1－11 中可以比较简洁地看到程序单元中允许的语句及其顺序。

表 1—11

<table>
<tr><td rowspan="6">注解行</td><td colspan="2">PROGRAM、FUNCTION 或 SUBROUTINE 语句</td></tr>
<tr><td rowspan="5">FORMAT 语句</td><td>IMPLICIT 语句</td></tr>
<tr><td>其它说明语句</td></tr>
<tr><td>DATA 语句</td></tr>
<tr><td>语句函数语句</td></tr>
<tr><td>可执行语句</td></tr>
<tr><td colspan="3">END 语句</td></tr>
</table>

从表 1—11 中可以看出，一个程序单元中对语句和注解行所要求的顺序。垂直线画出了各种语句所能分布的范围，如 FORMAT 语句可以出现在任何地方，可以出现在语句函数语句和可执行语句的前后。水平线画出各种语句应以什么顺序分布在程序中，如语句函数语句不能在可执行语句之后，而 END 语句虽然也是可执行语句，但它只能作为程序单元的最后一个语句出现等等。

程序的正常执行序列就是按可执行语句在程序单元中出现的顺序执行可执行语句。执行可执行程序从主程序的第一个可执行语句开始执行。当辅程序指明的外部过程被引用时，从指明过程的 FUNCTION 或 SUBROUTINE 语句之后的第一个可执行语句开始执行。

2. DATA 语句

大多数为科学计算编制的程序中都要对变量或数组元素赋初值。可以通过 READ 语句、赋值语句以及 DATA 语句等实现此目的。比较起来，DATA 语句有简单易用的优点，当源程序每次重新启动时都能取到未经改变的初值。

DATA 语句适用于给变量、数组和数组元素提供初值。DATA 语句是非执行语句。若在程序单元内有 DATA 语句，它可出现在说明语句以及任何语句函数语句或可执行语句之间。

当可执行程序开始执行时，所有初始有定义的实体是有定义的，所有没有被初始定义的或没有与初始有定义的实体相结合的实体是无定义的。

DATA 语句的一般形式是：

DATA nlist/clist/[[,]nlist/clist/]…

其中：nlist 为变量名、数组名和数组元素名构成的表；clist 的形式如下：clist 为下述形式的 a[,a]…形式的表。

a[,a]…

其中：a 为 c 或 r * c。c 是常数；r 是非零无符号整常数；r * c 形式等价于常数 c 连续出现 r 次。

由每个表 nlist 指定的项目的个数必须与相应的表 clist 的项目个数相同。由表 nlist 指定的项目和由表 clist 指定的常数一一对应，即表 nlist 的第一个项目和表 clist 的第一个常数相对应。以此类推。根据这种对应，建立 nlist 中各实体的初值并且将实体初始定义。若在表中有不带下标的数组名，则该数组的每一个元素必须有一个常数。数组元素的排序由数组元素下标值确定。

表 nlist 中的实体类型和相对应的表 clist 常数的类型要一致。

DATA 语句的最简单的例子如下：

DATA K/3/,A/2.2/,B/3.45/,C/-1.24/

上一语句也可写为：

DATA K,A,B,C/3,2.2,3.45,-1.24/

其含义为把四个初值赋给 K、A、B、C 四个量，其顺序和类型必须一致，数值之后的"/"对初学者往往会忘记，请多多注意。

如果 A 已说明为一个 2 * 2 的数组，我们要对整个数组赋初值，其 DATA 语句可写为：

DATA A(1,1),A(2,1),A(1,2),A(2,2)/1.0,1.5,2.0,2.5/

亦可为：

DATA A/1.0,1.5,2.0,2.5/

当然，若要对数组 A 不按其存放顺序赋值，则 DATA 语句写成对数组元素一一赋值更明确一些。

四、综 合 举 例

到目前为止，我们已具备编写一个完整的、真正可以运行的

Fortran 程序的能力了。

当我们需要编写一个 Fortran 程序时，首先要分析问题、确定程序分块、画出流程图和程序结构图等。例如，现要求编写一个求二次方程 $AX^2+BX+C=0$ 根的通用的程序。求根和打印结果各编写一个子程序。求根子程序名为 QUAD，打印子程序名为 OUTPUT。

对打印的要求是：对每个要求根的二次方程在一行内印出方程的系数和求出的根。当方程的根是实根时，虚部不印出；当方程的根是纯虚根时，实部不印出。每印一行之前先空一行。

显然，二次方程的系数 A、B、C 和根的实部 X1R、X2R，虚部 X1I、X2I 都必须是这两个子程序的虚元。在主程序中读入方程的系数，通过调用语句引用这两个子程序。要求每印出一个二次方程的根后又再读入一组系数进行计算，直至读入的 A＝0 时计算才停止进行。并且把 10 个方程的结果分为一组，每组印一表头，组与组之间空 5 行。

主程序的流程图如图 1.1 所示。

根据以上流程图可编写出主程序：

```
C THIS IS THE MAIN PROGRAM
      PROGRAM ABC
      INTEGER COUNT
190   WRITE(5,12)
12    FORMAT(4X,1HA,9X,1HB,9X,1HC,8X,3HX1R,7X,&
         3HX1I,7X,3HX2R,7X,3HX2I)
      COUNT=0
170   READ(4,18)A,B,C
 18   FORMAT(3F7.4)
      IF (A.EQ.0.0) STOP
      CALL QUAD(A,B,C,X1R,X1I,X2R,X2I)
      CALL OUTPUT(A,B,C,X1R,X1I,X2R,X2I)
      COUNT=COUNT+1
      IF (COUNT.LT.10) GOTO 170
```

```
      WRITE (4,22)
22    FORMAT(//////)
      GOTO 190
      END
```

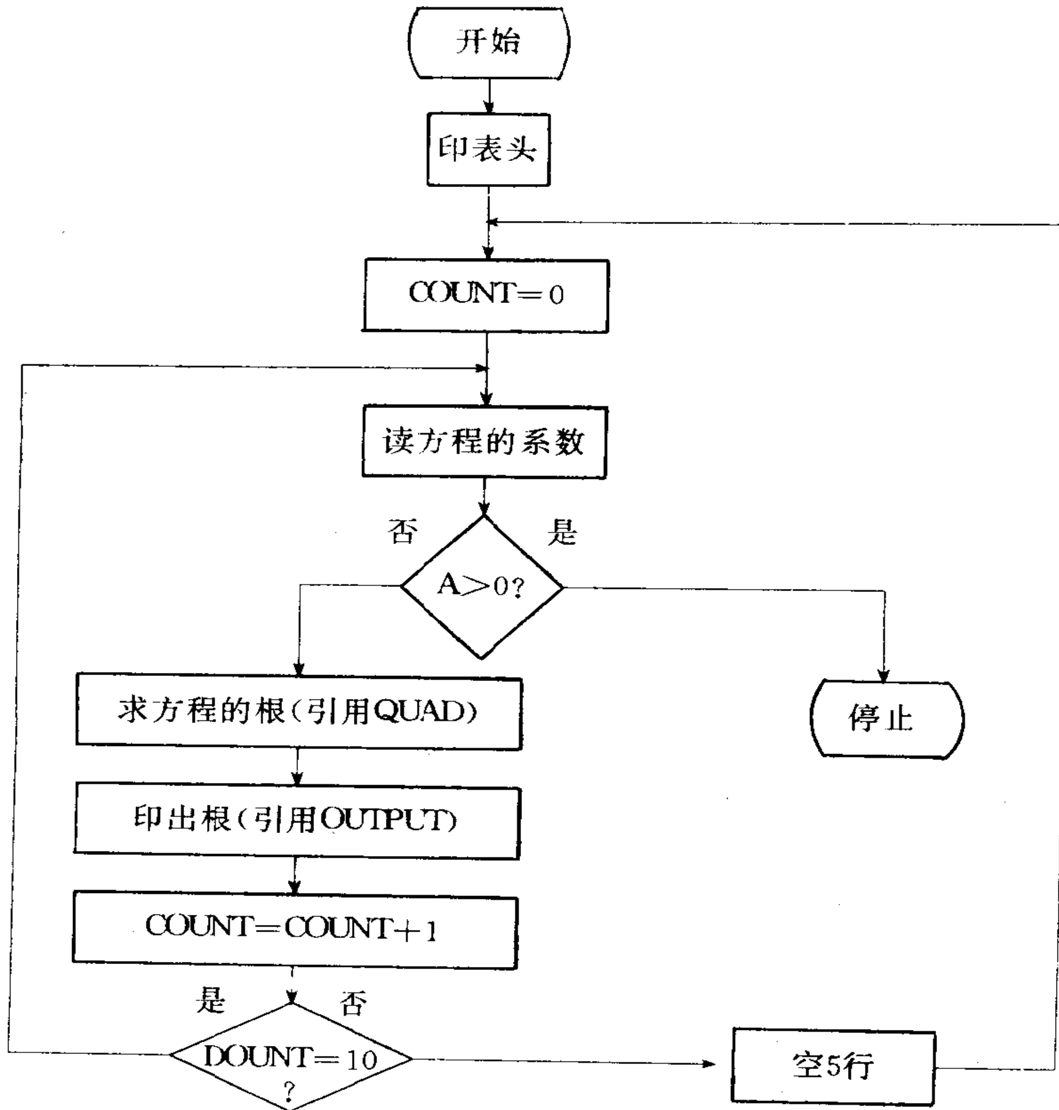


图 1.1

打印输出子程序 OUTPUT 的流程图(见图 1.2)和程序是:

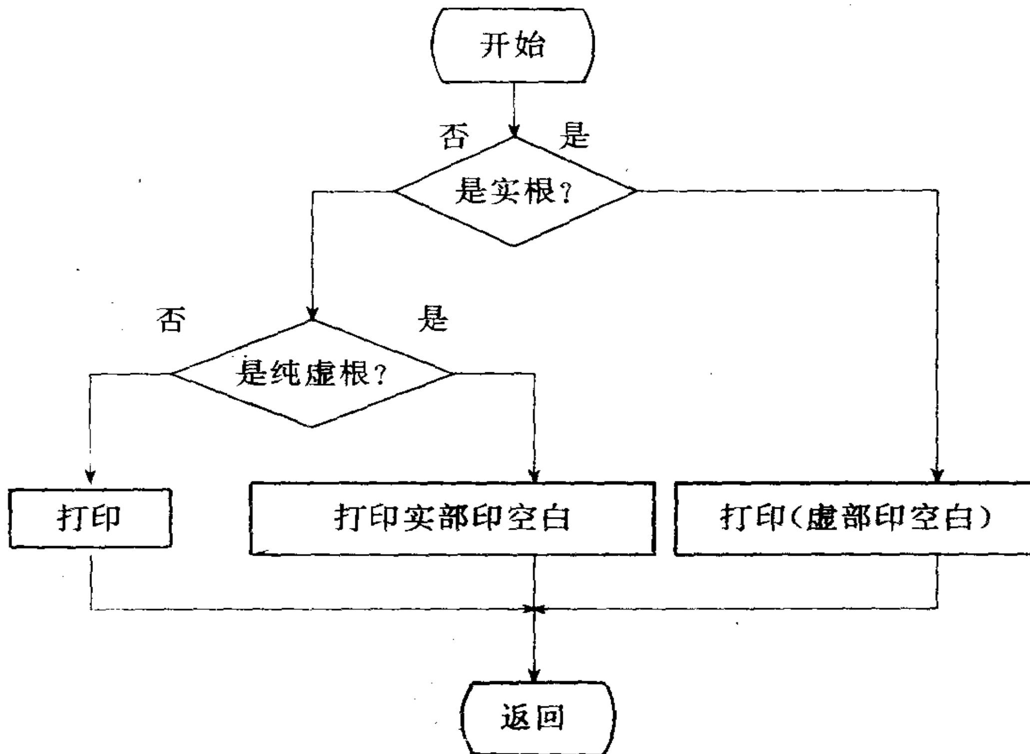


图 1.2

```
C THIS IS THE SUBROUTINE FOR PRINTING
C THE RESULTS
      SUBROUTINE OUTPUT (A,B,C,X1R,X1I,X2R,X2I)
      IF (X1I.NE.0.0) GOTO 90
      WRITE (5,180) A,B,C,X1R,X2R
180   FORMAT (/1X,4(F7.3,3X),10X,F7.3)
      RETURN
90    IF (X1R.NE.0.0) GOTO 270
      WRITE (5,360) A,B,C,X1I,X2I
360   FORMAT (/1X,3(F7.3,3X),2(10X,F7.3,3X))
      RETURN
270   WRITE (5,450) A,B,C,X1R,X1I,X2R,X2I
450   FORMAT (/1X,7(F7.3,3X))
```

```
      RETURN
      END
```

求二次方程根子程序 QUAD 的流程图(见图1.3)和程序是:

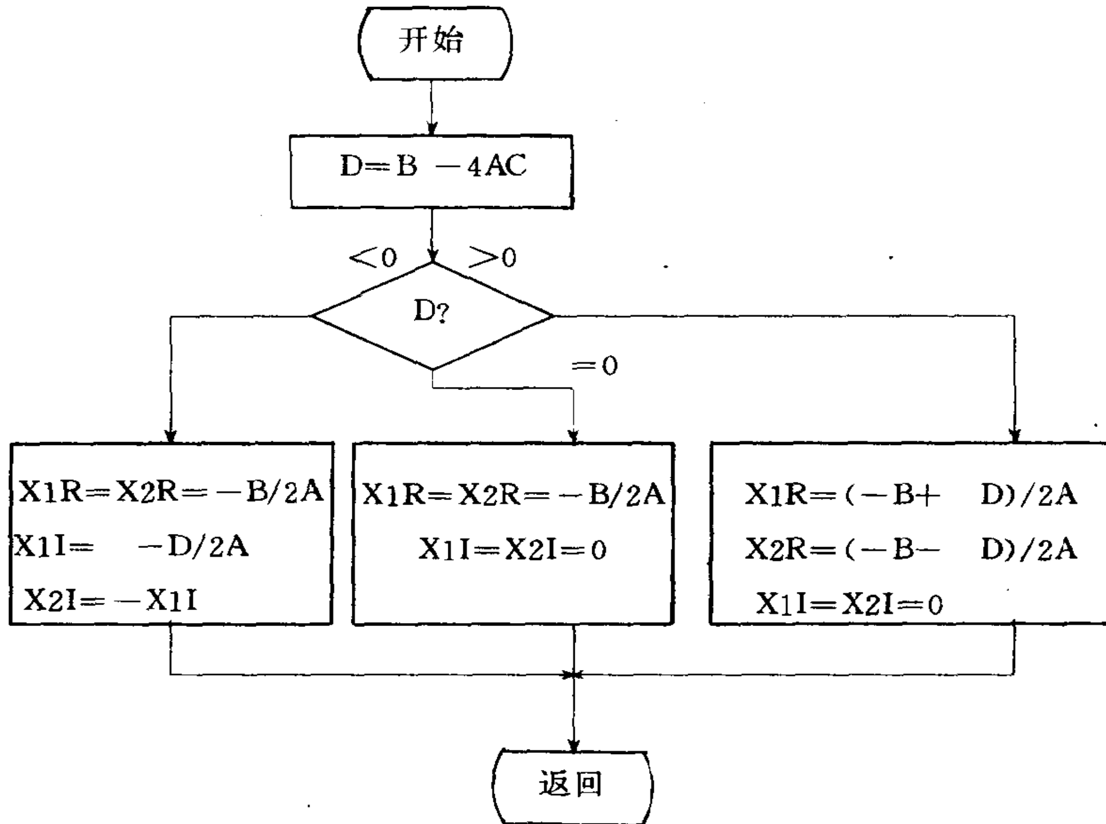


图1.3

```
C THIS IS THE SUBROUTINE FOR FINDING
C THE ROOTS
      SUBROUTINE QUAD (A,B,C,X1R,X1I,X2R,X2I)
      D=B*B-4.0*A*C
      IF (D) 111,222,333
111   X1R=-B/(2.0*A)
      X2R=X1R
      X1I=SQRT(-D)/(2.0*A)
      X2I=-X1I
```

```
      RETURN
222   X1R=-B/(2.0*A)
      X2R=X1R
      X1I=0.0
      X2I=0.0
      RETURN
333   SQRTD=SQRT(D)
      X1R=(-B+SQRTD)/(2.0*A)
      X2R=(-B-SQRTD)/(2.0*A)
      X1I=0.0
      X2I=0.0
      RETURN
      END
```

以上用一个主程序和两个子程序完成了我们的全部需求。我们看到在一个问题提出之后，首先要有清晰的问题分析，其次是实现你的意图的明确流程图，按照这些你就可以“不动脑筋”按部就班地写出一个 Fortran 源程序了。

小结

本章扼要地介绍了 Fortran 的发展史、Fortran 语言的基本元素、最基本的语句以及程序结构。如果读者学习过其它程序设计语言，可以将它们做些比较，以加深理解。如果未接触过其它程序设计语言，可以按照本章的叙述，把你手边的任务编写一些 Fortran 程序来加深理解本章内容。

在结束本章时我们必须提醒读者：任何时候都要特别注意程序设计语言各个成份的语义、语法以及输入输出的格式等。

习题

1. 下列名在 Fortran 源程序中是否正确？为什么？

3KING KING－3 KING(3) KING3 KING.3

2. 下列 Fortran 实常量是否正确？为什么？

12 12.5 －125 －10^2 0.05－E12 E03 4E＋2.4 12,375.3
－12,300.04E－02 125.

3. 已知 X＝120.7，Y＝－43.7，I＝12，若有以下语句：

```
WRITE (5,20) I,X,Y
```

20为格式语句标号。各为以下形式，请列出相应打印结果中 I、X、Y 等的形式：

```
20 FORMAT (1X,I3,F6.1,F7.2)
20 FORMAT (1X,I3,2F6.1)
20 FORMAT (1X,3HI= ,I3,3HX= ,F7.2,3HY= ,F7.2)
```

4. 将下列数学表达式表示为正确的 Fortran 表达式。

$$\frac{X-2}{Y+3} \qquad (X+Y)^3 \qquad \frac{X \cdot Y}{(X-Y)^2} \qquad \frac{1}{x} \cdot \frac{\sin x}{1-y}$$

$$\left(\frac{x+\pi y}{3z}\right)^{21} \quad a+bY+c\sin x+d\cos^2 x \quad a+by+c\sin x+d\cos x^2$$

5. 用引用外部函数和引用子程序两种不同的方法，编写一个完整的求矩阵 A(M,N)和 B(N,M)相乘的 Fortran 程序，其中 M 和 N 分别为4和3、7和9、12和14三组值（本题要求画出正确的流程图并带有合理的输入、输出格式）。

第二章　基本术语和概念

在第一章中我们已经介绍了 Fortran 的一些入门知识，读者通过前面的学习应该已经可以编写一些完整的 Fortran 程序了。本章将在介绍了前述 Fortran Ⅱ 和 Fortran Ⅳ 的内容的基础上，进一步讨论有关 Fortran 77及部分 Fortran 90的基本语义和语法。

第一节　字　符　集

Fortran 77的字符集由26个字母(字母是英文26个字符之一),0～9共10个数字和13个特殊字符组成(除包括 Fortran Ⅳ 中的11个特殊字符外,还包括冒号(:)和货币符(¤或￥),参见第一章第二节)。

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

数字是下列10个字符之一:

0 1 2 3 4 5 6 7 8 9

当需要数值解释时,数字串解释为十进制数。

字母数字字符是一个字母或是一个数字。

特殊字符是表2—1所列13个字符之一。

表2—1

字　符		=	+	—	*	/	(	)	,	.	¤或￥	'	:
字符名	空格	等号	加号	减号	星号	斜线	左括号	右括号	逗号	小数点	货币符号	撇号	冒号

该字符集比 Fortran Ⅳ 使用的字符集多了两个特殊字符,即货币符号(¤或￥)及冒号(:),在各国家标准中,货币符号是随国家而异的。

以上列出的字母次序对字母指明了排序序列,即 A 先于 B, B 先于 C, ……。以上列出的数字次序对数字指明了排序序列,即0

先于1，1先于2，……。在排序序列中，数字和字母不许混合排序，即所有数字被认为排在A前面或Z的后面。空格字符先于字母A，也先于数字0。特殊字符的次序没有表示排序序列的意义。

而Fortran 90中的字符集则是由字母、数字、下划线和特殊字符组成的。

下划线为_

在名字中下划线可以作为有效字符使用。

Fortran 90中有21个特殊字符，见表2－2。

表2－2

字符	字符名	字符	字符名
	空格	:	冒号
=	等号	!	叹号
+	加号	"	引号
−	减号	%	百分号
*	星号	&	英语的 and
/	斜线	;	分号
(	左括号	<	小于
)	右括号	>	大于
,	逗号	?	问号
.	小数点或句号	¤或¥	货币符号
'	撇号		

特殊字符中的货币符号因国家而异。

在Fortran 90中还允许有该处理系统中可以表示的附加字符，但只可以出现在字符常量、字符串编辑描述符、注解和输入/输出记录中。

默认字符类型必须支持包含Fortran字符集的一个字符集。其它字符集可以由操作系统依照非默认字符类型来支持。不规定非默认字符类型中可利用的字符。

我们可以看到，Fortran 90的字符集比Fortran 77的字符集有了更大的扩充并有更大的灵活性。从字符集上可以看出Fortran 90具有一个新面貌，它增加了很多新内容。

第二节 基 本 术 语

本节将要进一步介绍一些常用的基本术语，并引进一些更深的语义。

一、数 据

Fortran 77中的数据类型基本上如第一章中所述。它强化了字符型数据及对数据的精确定义。在 Fortran 90中有关数据部分中引进了一些新的概念，这里将对此进行介绍。

(1)数据。在 Fortran 90中定义了两类数据类型，即内在类型和导出类型。这里我们只给出它们的定义。

内在类型是和各种运算一起隐式定义的一种类型，总是可访问的。内在类型包括 INTEGER、REAL、COMPLEX、CHARACTER 和 LOGICAL。内在类型的属性由类型参数表示。类型参数是 KIND 和 LEN。

KIND(种别类型参数)指明整数类型的十进指数范围，实数类型和复数类型的十进制精度和指数范围，字符类型和逻辑类型的表示方法。LEN(长度类型参数)对字符类型规定了字符的个数。例如：

```
REAL (KIND=3) :: ABC, X, LONG
CHARACTER (LEN=40, KIND=HANZI) :: NAME
```

导出类型是非隐式定义的类型，要求用一个类型定义以便声明其成份是某种内在类型还是其它导出类型。导出类型的标量对象称为结构。例如：

```
TYPE ADDRESS _ TYPE
INTEGER :: POST _ CODE
CHARACTER(LEN=14, KIND=HANZI) :: STATE
CHARACTER(LEN=30, KIND=HANZI) :: CITY, STREET
INTEGER :: NUMBER
END TYPE ADDRESS _ TYPE
```

内在类型和导出类型以后都还要仔细介绍，在此仅是简单提及一下。

(2)数据值。每个内在类型都和一组值相连，即该类型可以采取的数据。导出类型的数据值是根据它的成分的数据值确定的。

(3)数据实体。数据实体是一个数据对象、表达式求值的结果、或函数引用的执行结果(即函数值)。数据实体有数据类型(内在的或导出的)，并且有或可以有数据值。每个数据实体有一个秩，它或是一个标量或是一个数组。

(4)数据对象(简称对象)。数据对象是一个常量、变量、或常量的子对象。有名数据对象的类型可以显式或隐式地规定。

(5)子对象。子对象是某些有名对象的一部分，可由其它部分引用和独立地定义。包括数组的部分(数组元素和数组片段)、字符串的部分(子串)和结构的部分(成分)。子对象只能用子对象指定符来引用。变量的子对象是变量。

(6)由一个名引用的对象。它指有名标量(标量对象)和有名数组(数组对象)。

(7)由子对象指定符引用的子对象。它指数组元素(标量子对象)、数组片段(数组子对象)、结构成分(标量或数组子对象)和子串(标量子对象)。

在 Fortran 90中对数组引入了很多新的名词及相关定义。

(8)数组。数组是一组标量数据，全部数据都有相同的类型和类型参数，其各个元素按矩形模式排列。

(9)数组元素。数组元素是数组中的一个单个元素，并且是标量。

(10)数组片段。数组片段是数组元素的一个子集，而且它自己又是一个数组。

(11)维展。数组最多可以有7维，在各维各有一个维展(即该维的元素的个数)。

(12)秩和大小。数组的秩是维的个数，数组的大小是数组中元素的总个数，它等于各个维展的乘积。数组的大小可以是零。

(13)形状。数组的形状由其秩及每维的维展来表示，也可以表示为秩为1的数组，其元素是它的各个维展。

若两个数组有相同的形状，则称这两个数组是可符合的。标量和任一数组是可符合的。对标量对象定义的任何内在运算都可应用于与其可符合的对象。

秩为1的数组可由标量或其它数组来构造，而且它的形状可以重新改造为任何可允许的数组形状，只要改造前后大小一样即可。

下例为类型相同但形状不同的几个实体：

```
INTEGER A, B
INTEGER, DIMENSION (10) :: C, D
INTEGER, DIMENSION (5,6) :: E, F
```

字符型数据是一字符串。该字符串可以由处理系统所能表示的任何字符组成。在字符型数据中，空格字符是有效的且有意义。字符型数据的长度是该字符串中字符的个数。字符串中的每一个字符在存储序列中都占有一个字符存储单元。

在 Fortran 77中字符常量的形式是一个撇号后跟非空字符串，再后跟一撇号。该字符串可由处理系统所能表示的任何字符组成。而起分界作用的撇号不算作该常量所表示的数据的一部分。在数据串内的撇号，用两个相邻的其间不插入空格字符的撇号表示。在字符常量中，嵌入在分界撇号之间的空格字符是有意义的。

字符常量的长度是分界撇号之间的字符个数，当然，每一对相邻的撇号只算作一个字符。分界撇号不计算在内。字符常量的长度必须大于零。

字符常量的例子是：

'I DO NOT KNOW'长度为13。

'I DON'' T KNOW'长度为12，注意插入的撇号要重复，但只算一个长度。

表示常量的字符串形式既指明了它的值也指明了它的数据类型。

在 Fortran 77中数组说明符的形式是：

a(d[, d]…)

其中：a 是数组的符号名；d 是维说明符。

数组的维数是数组说明符中维说明符的个数，在 Fortran 90中也称数组的秩，最小维数是1，最大维数是7。

而维说明符的形式是：

$[d_1:]d_2\cdots$

其中，d_1是维下界；d_2是维上界。

维上界和维下界都是算术表达式，称为维界表达式，其中所有常量、常量符号名和变量都应是整型。任何维界的值可以为正、负或零。但维上界的值必须大于或等于维下界的值。若仅指明维上界，则维下界的值为1，只在这种情况下才和 Fortran Ⅳ 中维说明符的表示是一致的。

维的大小是值：

d_2-d_1+1

其中：d_1是维下界的值；d_2是维上界的值。

若维下界的值为1，则维的大小是 d2。

表2－3指明了任一个下标的下标值。

表2－3

n	维说明符	下　标	下标值
1	$(j_1:k_1)$	(S_1)	$1+(S_1-j_1)$
2	$(j_1:k_1,j_2:k_2)$	(S_1,S_2)	$1+(S_1-j_1)+(S_2-j_2)*d_1$
3	$(j_1:k_1,j_2:k_2,j_3:k_3)$	(S_1,S_2,S_3)	$1+(S_1-j_1)+(S_2-j_2)*d_1+(S_3-j_3)*d_2*d_1$
…	…	…	…
n	$(j_1:k_1,\cdots,j_n:k_n)$	$(S_1,\cdots,S_n)$	$1+(S_1-j_1)+(S_2-j_2)*d_1$ $+(S_3-j_3)*d_2*d_1+\cdots$ $+(S_n-j_n)*d_{n-1}*d_{n-2}*\cdots*d_1$

在表2－3中：n 是维数，$1\leqslant n\leqslant 7$；j_i 是第 i 维的下界值；k_i 是第

i 维的上界值；若只指明上界，则 $j_i=1$；S_i 是第 i 个下标表达式的整值；$d_i=k_i-j_i-1$是第 i 维的大小。若 $j_i=1$，则 $d_i=k_i$。例如：

DIMENSION A(－6:7)

这说明了 A 是一个一维数组它有14个元素，从 A(－6)到 A(7)。

由此看到，维上、下界给用户带来很多方便之处，可不拘泥于维下界只能是1了。

二、符号名、指定符、语句标号

符号名、语句标号的定义和规定在 Fortran 77中是和 Fortran Ⅳ 中相一致的。但在 Fortran 90中规定了在符号名中下划线可以做为有效字符使用。符号名的最大长度是31个字符。符号名的例子为：

ABC1
NAME_LENGTH （单下划线）
S_P_A_C_E_M_A_N （有两个连续下划线）
TRANSFER_ （尾部下划线）

语句标号的定义和使用则几种 Fortran 都是一致的，只在 Fortran 90中更加明确地规定了：若语句有标号，则语句必须包含一个非空格字符。即不可对空语句设置语句标号。并规定了在语句标号间进行区别时，前导零不起作用。

例如：10和010表示同一个语句标号。

在 Fortran 90中引进了源程序书写的自由格式，在自由格式的语句中规定最多可有39个继续行，这个数字比以前标准中的规定有很大的提高。

在 Fortran 90中用名字来代替以前 Fortran 的术语符号名，名字沿用符号名的定义。可用名字来标识一个程序成分，如标识程序单元、有名变量、有名常量、虚元或导出类型等。同时还引入了指定符的概念，它用于指定子对象。

子对象指定符是一个名，其后跟随一个或多个下列内容：成分

选择符、数组片段选择符、数组元素选择符以及子串选择符。

在下面的例子中看到导出类型可以有成分，而成分又是数组：

```
TYPE TRIP
  REAL, DIMENSION(3) :: VECTOR
END TYPE TRIP
```

另一方面，数组元素又可具有导出类型，例如：

```
TYPE (TRIP), DIMENSION(10) :: T
```

而 T(1:7)和 T(1)%VECTOR 就是一些子对象指定符，它由对象的名字后跟有关的选择符组成。

三、关　键　词

在 Fortran 90中使用的关键词有两种：

(1)语句关键词。它是语句语法的一部分。例如：IF、READ、UNIT、KIND 等都是语句关键词。而它们并不是保留词，即具有相同拼法的词做为名字使用是允许的。

(2)变元关键词。变元关键词指虚元名。在内在过程中规定了位置变元和变元关键词，例如 KIND、STRING、STRING _ A、BACK、MASK、DIM 等都是内在过程中的变元关键词。

四、说　　明

说明涉及各种程序实体的属性的说明，比如包括规定有名数据对象的数据类型或规定有名数组对象的形状等。

五、定　　义

在可执行程序执行期间的任何给定时刻，每个变量、数组元素或子串的定义状态或是有定义的，或是无定义的。

有定义的实体有一个值。有定义的实体的值，在它变为无定义或用不同的值再定义它之前是不会改变的。

若变量、数组元素或子串是无定义的，则它没有可预定的值。

一个先前有定义的变量或数组元素可以变为无定义的。一个

有定义的变量或数组元素，除了在明确禁止的地方以外，允许以后再定义它们。

若字符变量、字符数组元素或字符子串中每一个长度为1的子串是有定义的，则它们是有定义的。若一个字符串是有定义的，则该字符串的每一个子串也是有定义的，并且若这个字符串的任一子串是无定义的，则该字符串也是无定义的。定义任何子串都不会导致任何其它字符串或子串变为无定义的。

若一个实体在 DATA 语句中赋了值，则它是初始定义的。在开始执行可执行程序时，初始定义的实体处在有定义状态。在开始执行可执行程序时，不是初始定义的或不是与初始定义的实体相结合的所有变量和数组元素，都是无定义的。

在执行对一个实体的引用时，它必须是有定义的。

六、引　　用

在可执行程序执行期间，当程序的某些地方需要使用变量、数组元素或子串的值时，该变量名、数组元素名或子串名在语句中相应的地方出现，此种出现就称为变量引用、数组元素引用或子串引用。当执行对一个实体的引用时，它的当前值应是可用的。注意：定义一个实体的动作不认为是对那个实体的引用。

在可执行程序执行期间，当程序的某处需要执行由某个过程指定的动作时，该过程名在语句中某处出现，此种出现就称为过程引用。当执行过程引用时，该过程必须是可供使用的。

模块引用是指在 USE 语句中模块名的出现。

除非必须要由引用完成对实元的说明外，在实元表中，数据对象名、数据子对象指定符或过程名的出现不构成对该数据对象、数据子对象或过程的一个引用。

七、结　　合

若在同一程序单元内，用不同的名字标识同一数据，或在同一可执行程序的不同程序单元内，用相同的名字或不同的名字标识

同一数据，则称在这些实体间存在着某种结合。

结合可以是名结合、指针结合、或存储结合。名结合可以是变元结合、宿主结合、或使用结合。

存储结合导致不同的实体使用相同的存储单元。

一旦存在某种结合之后，就允许实体使用同样作用域单位中不同的名或使用不同作用域单位中同样的名或不同的名来标识它。

上述这些基本术语是在 Fortran 中经常要使用的，在此先集中介绍一下，在以后的使用中将会不断地提及它们，从而使这些概念愈来愈明确。

第三节　关于“执行”的概念

程序执行就是执行程序规定的计算动作的序列。Fortran 语句分类为可执行语句和不可执行语句。在程序单元中可以出现哪种语句，次序怎样等都有明确的规定。而且有时某些可执行语句只允许出现在某些可执行构造中，编程序时必须特别小心。本节将对这些内容进行介绍。

一、可执行语句

可执行语句是实现或控制一个或多个计算动作的一条指令。因此，程序单元的可执行语句确定了程序单元的计算行为。

Fortran 77比 Fortran Ⅳ的可执行语句增加了：

PRINT 语句

CLOSE 和 INQUIRE 语句

而 Fortran 90的可执行语句的功能增加主要在于提供了对结构化程序设计的支持。而不符合结构化思想的一些特性和内容都被看作是在下一个 Fortran 标准中可以去掉的部分。

Fortran 90 中增加了几种可执行的块构造：

IF 构造

CASE 构造

DO 构造

每种构造中有其相应的新的语句，这部分内容将在第七章的控制语句中介绍。

二、不可执行语句

不可执行语句指明数据的特性、排列和初值；指明编辑信息；指明语句函数；指明程序单元的类别以及辅程序中的入口点。不可执行语句不是执行序列的一部分。不可执行语句可以带标号，但这种语句标号不许用于控制执行序列。Fortran 77中增加的不可执行语句有：

ENTRY 语句

BLOCK DATA 语句

PARAMETER 语句

以上这些语句将在有关章节中予以介绍。

在 Fortran 90中的不可执行语句可用来配置计算动作发生的程序环境。在块数据单元内的全部语句必须是不可执行的。模块只可在模块中的辅程序内包含可执行语句。

Fortran 90中进一步增加的不可执行语句有：

TYPE 语句

POINTER 语句

TARGET 语句

ALLOCATABLE 语句

PUBLIC 语句

PRIVATE 语句

INTENT 语句

OPTIONAL 语句

NAMELIST 语句　　等

这些语句将随着介绍内容的深入在有关章节中介绍。

三、语句执行顺序

首先讨论 Fortran 77中的各种语句应处的位置及其执行顺

序。

PROGRAM 语句仅能作为主程序的第一个语句出现。辅程序的第一个语句不是 FUNCTION、SUBROUTINE 语句，就是 BLOCK DATA 语句。

程序单元内允许的语句是：

(1)FORMAT 语句可以出现在任何地方。

(2)所有的说明语句必须写在所有 DATA 语句、语句函数语句和可执行语句之前。

(3)所有语句函数语句必须写在所有可执行语句之前。

(4)DATA 语句必须出现在说明语句之后的任何地方。

(5)除了在块 IF 语句及其相对应的 END IF 语句之间，或 DO 语句及 DO 循环的终结语句之间。ENTRY 语句可出现在任何地方。

在程序单元的说明语句内，IMPLICIT 语句必须写在除了 PARAMETER 语句外的所有其它说明语句之前。说明常数符号名类型的任何说明语句必须写在定义常数的特定符号名的 PARAMETER 语句之前；PARAMETER 语句必须写在包含被 PARAMETER 语句定义的常数符号名的所有其它语句之前。

表2—4

<table>
<tr><td rowspan="5">注解行</td><td colspan="3">PROGRAM、FUNCTION、SUBROUTINE 或 BLOCK DATA 语句</td></tr>
<tr><td rowspan="4">FORMAT
和
ENTRY
语句</td><td rowspan="2">PARAMETER
语句</td><td>IMPLICIT 语句</td></tr>
<tr><td>其它说明语句</td></tr>
<tr><td rowspan="2">DATA
语句</td><td>语句函数语句</td></tr>
<tr><td>可执行语句</td></tr>
<tr><td colspan="4">END 语句</td></tr>
</table>

程序单元的最后一行必须是 END 语句。

语句和注解行所要求的次序见表2—4。

表2—4垂直线画出了各种语句所能分布的范围。水平线画出了各种语句允许的出现顺序。

正常执行序列就是按可执行语句在程序单元中出现的次序执

行可执行语句。当由辅程序指明的外部过程被引用时，从指明过程的 FUNCTION、SUBROUTINE 或 ENTRY 语句之后的第一个可执行语句开始执行。

在 Fortran 90中语句排序要求有所不同，见表2－5所示：

表2－5

<table>
<tr><td colspan="3">PROGRAM、FUNCTION、SUBROUTINE、MODULE 或 BLOCK DATA 语句</td></tr>
<tr><td colspan="3">USE 语句</td></tr>
<tr><td rowspan="4">FORMAT
和
ENTRY
语句</td><td colspan="2">IMPLICIT NONE</td></tr>
<tr><td>PARAMETER 语句</td><td>IMPLICIT 语句</td></tr>
<tr><td>PARAMETER
和
DATA 语句</td><td>导出类型定义
接口块
类型声明语句
语句函数语句和
说明语句</td></tr>
<tr><td>DATA 语句</td><td>可执行构造</td></tr>
<tr><td colspan="3">CONTAINS 语句</td></tr>
<tr><td colspan="3">内部辅程序或模块辅程序</td></tr>
<tr><td colspan="3">END 语句</td></tr>
</table>

表2－5指出了语句在程序单元和辅程序中的排序规则。一般地，不可执行语句在可执行语句之前，虽然 FORMAT、DATA 及 ENTRY 语句也可出现在可执行语句之间。

表2－6指出在各种作用域单位内允许的语句。

一个可执行程序的执行是从主程序的第一个可执行构造开始的。主程序或辅程序的执行包含了其作用域单位内的各种可执行构造的执行。当引用一个过程时，从所引用的入口点之后出现的第一个可执行构造处开始执行。有以下的例外情况，执行的效果是各种可执行构造按序执行，直到执行 STOP、RETURN 或 END 语句时为止。例外情况是：

(1)转移语句的执行可改变执行序列。对于执行序列，这些语

表2—6

作用域单位的种类	主程序	模块	块数据	外部辅程序	模块辅程序	内部辅程序	接口体
USE 语句	是	是	是	是	是	是	是
ENTRY 语句	否	否	否	是	是	否	否
FORMAT 语句	是	否	否	是	是	是	否
杂项声明(注)	是	是	是	是	是	是	是
DATA 语句	是	是	是	是	是	是	否
导出类型定义	是	是	是	是	是	是	是
接口块	是	是	否	是	是	是	是
语句函数	是	否	否	是	是	是	否
可执行语句	是	否	否	是	是	是	否
CONTAINS	是	是	否	是	是	否	否

注：杂项声明是指 PARAMETER 语句、IMPLICIT 语句、类型声明语句和说明语句。但表中所指的模块作用域单位没有把模块可能包含的模块辅程序在内。

句显式地规定了一个新的启动位置。

(2)IF 构造、CASE 构造和 DO 构造都包含一个内部语句结构，这些构造的执行包含了隐式(即自动的)的内部转移。

(3)交错返回和 END=、ERR=及 EOR=说明符都可以在转移中得到。

内部辅程序可以在主程序或辅程序的 END 语句之前。

由此可知，在 Fortran 源程序中语句是按什么顺序去执行的了。

第四节 程序单元

程序单元是 Fortran 程序的基本成分。程序单元可以是主程序、外部辅程序、模块或块数据程序单元。一个辅程序可以是函数辅程序或子程序辅程序。模块中包含可由其它的程序单元访问的各种 实体。块数据程序单元用来对有名公用块中的数据对象规定初始值。外部辅程序是不包含在主程序、模块或另一辅程序内的一

个辅程序。内部辅程序是包含在主程序或外部辅程序内的一种辅程序。模块辅程序是包含在模块中但却又不是内部辅程序的一种辅程序。

一个程序单元由一组非覆盖的作用域单位组成。作用域单位包括：

(1)导出类型定义。

(2)过程接口体，不包括其中的导出类型定义和包含的过程接口体部分。

(3)一个程序单元或辅程序，不包括其中的导出类型定义、过程接口体以及包含在它内部的辅程序。

一、可执行程序

可执行程序由一个且仅有一个主程序单元和任意个(包含零个)其它种类的程序单元组成。这组程序单元的出现次序可以任意组合。

二、主 程 序

主程序是不以 FUNCTION、SUBROUTINE 或 BLOCK DATA 语句作为第一个语句的程序单元。它可以用 PROGRAM 语句作为它的第一个语句。

在一个可执行程序中必须恰好有一个主程序。其构造为：

PROGRAM 程序名

说明部分

执行部分

END (或 END PROGRAM 程序名)

程序名对可执行程序是全局的，它和任何其它程序单元名、外部过程名、可执行程序中的公用数据块名或主程序中的局部变量名都不能相同。

主程序的说明部分不能包括 OPTIONAL 语句、INTENT 语句、PUBLIC 语句、PRIVATE 语句以及与上述语句等价的语句，

SAVE 语句在主程序中没有意义。

主程序的执行部分的语句序列说明了程序执行过程中主程序的操作。一个可执行程序的执行是从主程序的第一个可执行构造开始执行的。

主程序是不可递归的，即可执行程序中的任何程序单元，包括主程序本身，都不能调用主程序。

一个可执行程序执行到主程序的 END 语句时结束，或在可执行程序中遇到 STOP 语句时结束。

在主程序中定义的所有内部辅程序必须跟在 CONTAINS 语句之后。主程序是它的内部过程的宿主。

三、过　　程

我们可根据过程的引用形式和定义方式对过程进行分类。

根据过程的定义可以把它分成函数和子程序。一个函数的引用方式或者作为基本表达式显式地出现在表达式中，或者以定义的运算符的形式在表达式中出现。对一个子程序的引用是通过 CALL 语句或定义的赋值语句引用的。如果一个内在过程是按元素引用的，则称之为元素的内在过程。

一个过程或是一个内在过程、或是一个外部过程、一个模块过程、一个虚拟过程、或语句函数。

如果一个过程是由系统作为其不可分割的一部分而提供的，那么该过程就称为内在过程。

外部过程是外部辅程序定义或由非 Fortran 程序编制的过程。

内部过程是由内部辅程序定义的。内部过程可以出现在主程序中、外部辅程序中或模块辅程序中。内部过程不能出现在其它内部过程中。与外部过程不同之处在于：内部过程不是全局实体，内部过程中不能包括 ENTRY 语句，内部过程与虚拟过程不能进行变元结合，内部过程通过宿主结合可以对宿主实体进行引用和操作。

模块过程是模块辅程序定义的过程。

如果一个辅程序包含一个或多个 ENTRY 语句，那么每一个 ENTRY 语句都定义了一个过程，当然，FUNCTION 或 SUBROUTINE 语句也定义一个过程。

被说明为一个过程的虚拟变元或出现在一个过程调用中的虚拟变元称为一个虚拟过程。

由单独一条语句定义的函数称为语句函数。

在第一章中我们已介绍了对过程的引用，在此我们只是更精确地对过程进行了定义而已。

四、模　　块

在实际应用中往往需要在各程序单元间传递大批的数据，有时又要求把有些数据放在全局数据区来让所有的程序单元共享。公用块部分地解决了这些需求，但对接口信息以及更多的相关信息如何提供，这些都可由程序单元模块来提供更方便地使用，它可以看作是块数据程序单元的一种扩充和替换。模块可以被任何程序单元存取，因而也就使模块的内容可由任何程序单元所使用。这样，模块对定义全局数据区域、过程包等提供了一个有力的工具。模块提供了共享常量、变量、类型定义和过程的一种途径。它也可提供类似程序库的功能。

模块是一个不直接执行的程序单元，它包含数据说明以及由其它程序单元可通过 USE 语句来利用的过程。模块名与任何程序单元名、外部过程名、可执行程序中的公用块名或模块中的局部变量名都不能相同。使用 USE 语句进行模块引用。不论是直接或间接，一个模块不能引用它自己。

以下我们举例说明模块是怎样定义以及如何引用的：

```
 MODULE LOST_CARD_MODULE
 INTEGER, DIMENSION (:), ALLOCATABLE :: LOST_CARD
 INTEGER::NUMBER_OF_LOST_CARDS
END MODULE LOST_CARD_MODULE
```

引用此模块的方法如下：

```
SUBROUTINE SEARCH (CARD _ NUMBER, FOUND)
  USE LOST _ CARD _ MODULE
      …
  DO I=1, NUMBER _ OF _ LOST _ CARDS
    IF (CARD _ NUMBER= =LOST _ CARD(I)) THEN
      …
```

在模块中只能出现说明语句部分及模块辅程序部分，并且在说明部分中不能出现函数定义语句、ENTRY 语句和 FORMAT 语句。

小　　结

本章扼要地介绍了 Fortran 90的一些基本术语及概念，在定义上更加严格了，可以看作是学习 Fortran 90的一个前期准备部分，也相应给出了 Fortran 77和 Fortran 90的一些异同点，更详细的阐述将在以后的章节中陆续展开。

习　　题

1. Fortran 77的字符集和 Fortran 90的字符集有何差异？
2. 导出类型的定义是什么？试举一例。
3. 种别类型参数的表示方法是什么？试举一例。
4. 解释下列名词，并举例：
   数组片段，维展，秩，形状
5. Fortran 90中的名字有什么规定？
6. Fortran 90中如何用模块来解决全局数据区的问题？如何利用一个模块？

第三章 数据类型

Fortran 语言处理的对象是各种数据，它提供了一种不依赖于具体物理表示的抽象方法来将数据进行分类，这种数据抽象的方法就是数据类型的概念。例如数学上将数分为整数、有理数、无理数、实数、复数等。Fortran 语言中的数据被分为许多类型，例如：表示数值的有整型、实型、复型；表示非数值的有逻辑型和字符型。这些类型由于都是由系统提供的，故统称为内在类型。本章我们将详细地来介绍这些内在数据类型，在第十三章中将进一步介绍导出类型。

一般来讲，数据类型具有：

(1)名字。

(2)允许值的集合。

(3)表记允许值(常量)的方法。

(4)操作这些值的运算的集合。

数据类型的名字一般用作类型说明符，这将在第二、三节中介绍。在第一节中先介绍后三项内容。

第一节 数据类型的概念

一、数据类型的值集

对于每种数据类型，存在一个允许值的集合。这个集合的元素的个数有的是完全确定的。例如，逻辑型仅有表示真和假的两个值(即两个元素)，有的是以依赖于处理系统的方法来确定的，如整型、实型。对于一个处理系统来讲，表示整型数据的方法也可能不是唯一的。例如，一般来讲在 -2^{31} 到 $+2^{31}$ 之间可能是一种表示方

法，在-2^{63}到$+2^{63}$之间又是另一种表示方法。为此，Fortran 90引入了种别(KIND)的概念，种别作为一个类型参数来处理，这样不同的种别可对应于不同的表示方法，而表示方法由处理系统自行确定。这样 Fortran 的标准化程度就得到了进一步提高，从而提高了程序的可移植性。

二、常　　量

常量有两种：一种称为字面常量；一种称为有名常量(将在第四章中介绍)。在一个可执行程序中，所有具有相同形式的字面常量具有相同的值。在数值类型中均包括有零值，这个零值既不认为是正的，也不认为是负的，带有正负号的零值与不带正负号的零值都认为是相同的，且都认为是零值。下面我们分别来介绍各种类型的字面常量。

1. 整型

整型值的集合是数学上整数的一个子集。一个处理系统需提供一种或几种表示整数值方法，每种方法定义一个整型数据值的集合。每种方法用称为种别类型参数的类型参数值来区分。整型字面常量的表示形式为一个数字串，并在数字串后有一个可任选的下划线后再跟种别类型参数。整型数据为可选的正负号后跟整型字面常量。例如：

```
+373
-1024
35792
0
32_2
1992110235764803_8
32_SHORT
```

等都是合法的整型数据，这里要指出几点：

(1)上述整数按十进制数解释。

(2)数字串后的下划线和种别类型参数是可选项。若省略则称

为是默认整型，这时种别类型参数值是内在询问函数 KIND(0)的结果。其他种别类型参数值指出各种表示方法，上述例中32_2的2就是一个种别类型参数值，与其相应的表示方法与8所对应的表示方法不同。而例32_ SHORT 中 SHORT 应在前面说明是有名常量，且它具有常量值，这个值必须是非负的，而且与一种表示方法相对应。

由于用户的需要，Fortran 90中在整型中还定义了二进制、八进制和十六进制的字面常量，但它只能用于 DATA 语句中。

二进制常量的形式是以字母 B 开头，后跟用一对撇号或引号括起来的数字串，而且每个数字不是0就是1。其数值按二进制解释。

八进制常量的形式是以字母 O 开头，后跟用一对撇号或引号括起来的数字串，而且每个数字是0到7之间的一个数字。其数值按八进制解释。

十六进制常量的形式是以字母 Z 开头，后跟用一对撇号或引号括起来的数字或字母的串，而且每个数字是数字0到9或字母 A 到 F 之一。其数值按十六进制解释。

2. 实型

实型值的集合是数学上实数的近似表示的一个子集。一个处理系统要提供两种或更多种的近似表示方法，每种方法定义一个实型数据值的集合。每种方法用称为种别类型参数的类型参数值来区分。实型数据为可选的正负号后跟实型字面常量，实型的字面常量具有这样的形式：

有效数[指数字母　指数][_种别参数]

而有效数是可带有小数点的数字串或不带小数点的数字串，指数字母可为 E 或 D；指数是可带正负号的数字串；种别参数是整的有名常量或不带符号的整型字面常量。例如：

−12.78

+3.73E2

.966

3E4

4877_2

-13.6E-4_4

10.93E7_QUAD

.96D2

等都是合法的实型数据，这里要指出几点：

(1)上述实数按十进制科学记数法解释，即有效数乘以10的指数幂。而且书写的有效数字个数可以多于处理系统用来近似表示该常量值的位数。

(2)若有效数为不带小数点的数字串，即整型字面常量时，必须出现指数字母及指数，否则就变为整型数据了。

(3)下划线和种别参数是可选项，若省略，则称为是默认实型，这时种别类型参数值是内在询问函数 KIND(0.0)的结果。其他种别类型参数值指出各种表示法，上述例中48.77_2的2就是一个种别类型参数值，其相应的表示方法与后为4相应的表示方法应该不同。而例10.93E7_QUAD 中的 QUAD 应在前面说明是有名常量，且它具有非负的常量值，它必须与一种表示方法相对应。若种别参数和指数字母都出现时，指数字母只能是 E。

(4)若指数字母是 D，则表示双精度实常量。在 Fortran 90中引进了 KIND 的概念，显然表示实数的种别要比原先语言更为广泛。双精度是作为实型的一种特别形式出现的，所以此时禁止再说明种别参数，而规定其种别类型参数值是内在询问函数 KIND(0.0D0)的结果，而且规定其表示方法的十进制精度要大于默认实型方法的精度。

3. 复型

复型值的集合是数学上复数的近似表示的一个子集。复型的值是实值的有序对。第一个实值称为实部，第二个实值称为虚部。

每种用于表示整型数据或实型数据的近似方法都可用于复型数据的实部和虚部的表示。

对于复型数据的种别类型参数是用于说明实部和虚部这两个

实型数据的近似表示方法的，若省略种别类型参数，则称此复型数据为默认复型，亦即实部和虚部均为默认实型数时，此复型数据是默认复型的。

其表示形式是用一对园括号括起来的，中间用逗号分隔的实部和虚部。而实部和虚部都可以是有正负号的整字面常量或有正负号的实字面常量。

由上可见一个复型字面常量可能有整一整，整一实，实一整或实一实四种结构，那么怎么来决定其种别类型参数值呢？其办法是：

（1）将整型数据的部分转换成实型数据，若两部分都是整型的，则转换成默认实型，因而认为是默认复型的；若一部分是整型的，另一部分是实型的，则将整型的部分转换成另一部分的实型的近似方法，种别类型参数就是实型部分的那种种别类型参数。

（2）若两部分都是实型数据且种别类型参数相同，则复型的种别类型参数就是实型数据的那种种别类型参数。特别当都是默认实型时，就得到默认复型。如果两部分的种别类型参数不同，则具有精度高的那部分的种别类型参数就是复型的种别类型参数，且另一部分还需转换为精度高的那种近似表示。例如，

(1, 3)

(2.0,-2.0)

(0,3.2E5)

(3.5_4,+3.7E3_8)

等都是合法的复型字面常量。而且前三个例子都是默认复型的表示，而在最后一例中，如果假定种别类型参数值4比8所表示的实数精度低，则该复型数据具有种别类型参数值8。

4. 逻辑型

逻辑值仅有两个，即表示真(.TRUE.)和假(.FALSE.)的两个值后跟可选的下划线和种别类型参数。一个处理系统要提供一种或几种表示方法。若省略可选的下划线和种别类型参数，则称为默认逻辑型，此时种别类型参数值是内在询问函数 KIND(.FAL-

SE.)的结果。其他种别类型参数值指出各种不同的表示方法。例如：

.TRUE.

.FALSE._2

.TRUE._LONG

都是合法的逻辑字面常量，当然 LONG 应该是已说明的有名常量，且具有非负的整常量值，它必须与一种表示方法相对应。

5. 字符型

字符型值的集合是字符串的一个集合。一字符串是一个字符序列，从左到右编号为1，2，3，……直到该串包含的字符个数。这个字符个数称为该字符串的长度。长度也是一种类型参数，其值大于等于零。称长度等于零的字符串为空串，不同长度的字符串都是字符型的。

处理系统需提供一种或几种定义字符型数据值的集合的表示方法。每种方法用不同的种别类型参数来区分。

字符字面常量用可选的种别类型参数和下划线后跟用撇号或引号为界的可表示字符的字符串来表示。前一个撇号或引号称为前导界限符，后一个称为尾界限符，它们统称为界限符。如果前导界限符前不出现种别类型参数和下划线，则称为默认字符型。

对于可表示字符和关于字符长度的计算有几点要注意：

(1)界限符不计在字符长度之中，也不是字符字面常量值的一部分。

(2)当字符字面常量值中需要包含撇号时，当用引号作界限符时，则用一个撇号来表示该字符，当用撇号作界限符时，则用两个连续的撇号来表示，且其间不夹有空格字符，此时两个连续的撇号计作一个字符。对字符字面值中的引号也类似地处理。

(3)两个连续的同样的界限符中间不夹空格，用来表示空串，即长度为零的字符串。

对于可表示字符的规定是与源程序形式有关的，在固定形式中，它依赖于处理系统字符集，它是除一部分(甚至全部)控制字

符外的所有字符;在自由形式中,它依赖于处理系统的图形字符。例如:

"DON'T"

'DON'' T'

'DON' 'T'

4_'DON' 'T'

HANZI _'缺了她就什么也干不成了'

等都是合法的字符字面常量,前三个例子的值是 DON'T,第四个例子的值也是 DON'T 但表示方法与前三个例子有所不同。最后一个例子中有名常量 HANZI 在先前已被定义为正整常量,其值表示该处理系统具有的一种汉字表示方法。

三、运　　算

1. 算术运算

对于整型、实型和复型数据,Fortran 90语言定义了内在的7种算术运算。

2种一元运算是:求反运算,其运算符为一;求同运算,其运算符为+。运算结果的数据类型和种别类型参数就是操作数的数据类型和种别类型参数。

5种二元运算是:加法运算,其运算符为+;减法运算,其运算符为一;乘法运算,其运算符为 *;除法运算,其运算符为/;乘幂运算,其运算符为 * *。运算结果的数据类型在前面已有介绍,不再重复。这里再谈一下种别类型参数。

(1)若两个操作数是相同类型和相同种别类型参数,则结果的种别类型参数就是操作数的种别类型参数。

(2)若两个操作数都是整型但种别类型参数不同,则运算结果的种别类型参数是取十进制幂范围大的那个种别类型参数;若范围一样大,则由处理系统决定取那一个。

(3)当一个操作数是整型,另一操作数是实型或复型,则运算结果的种别类型参数就是那个实型或复型的种别类型参数。

(4)若两个操作数是不同种别类型参数的实型或复型数据，则运算结果的种别类型参数是取十进制精度高的那个种别类型参数；若精度一样，则由处理系统决定取那一个。

运算结果的值就是相应算术运算的结果，加法为两个操作数之和；减法为两个操作数之差；乘法为两个操作数之积；除法为两个操作数之商；乘幂为以第一个操作数为底，第二个操作数为指数的乘幂值。这里特别要指出的是两个整型量相除时称为整除，其结果商就是算术上的商去掉小数部分后的整值。例如：25/2的值为12；(－25)/2的值为－12；2/3的值为0；(－1)/2的值也为0。

2. 并置运算

对于相同种别类型参数的字符型数据定义了并置运算，其运算符是//，并置运算的结果仍是该种别类型参数的字符型数据。运算结果的值为第一操作数的字符值，后面(右边)紧接着第二个操作数的字符值。例如：

'ABC' //'DE'的结果为'ABCDE'

3. 关系运算

对于整型、实型、复型和字符型数据，Fortran 90语言内在定义了六种关系运算：大于，其运算符为.GT.或＞；大于等于，其运算符为.GE.或＞＝；小于，其运算符为.LT.或＜；小于等于，其运算符为.LE.或＜＝；等于，其运算符为.EQ.或＝＝；不等于，其运算符为.NE.或/＝。对于 X_1OPX_2 这种比较运算，实际上是 (X_1-X_2) OP 0，所以 X_1，X_2的类型及运算 X_1-X_2的类型可以见前面算术运算部分的讨论。除此之外，还要说明一点，就是当 X_1-X_2是复型结果时，由于复数没有大小的概念，所以它没有前四种关系运算，而只有后两种关系运算，即等于或不等于。

对于字符数据来说，要求两个操作数具有相同的种别类型参数。其关系运算的执行可以理解为执行下列几个步骤：

(1)字符串中字符长度变为一致，即字符长度短者，右边以空格字符填充，直到长度相同为止。

(2)然后对两个操作数按字符位置从第一个字符开始逐个进

行比较，直到满足关系要求为止。

(3)字符的比较是按字符在字符集中理序序列的位置的先后来进行的，若字符1在字符2之前，则认为满足小于关系和不等于关系，若在之后，则认为满足大于关系或不等于关系；而位置相同，则认为满足等于关系。

由此可见所有空串都是相等的。等于关系和不等于关系运算结果与理序序列无关，而其它四种关系的运算结果是依赖于理序序列的。而 ASCII 理序序列是 各处理系统一致的，所以一般而言可移植性是能得到保证的。

此外，如果参与运算的默认字符数据值全是字母或全是数字，按语言的规定其顺序是严格确定的，否则也依赖于处理系统。所以在使用时要注意这点。对于非默认的字符型数据更是依赖于处理系统。

运算结果是默认的逻辑型数据，当关系式成立时为.TRUE.，否则为.FALSE.。

4. 逻辑运算

由前可知数值型数据和字符型数据进行关系运算后可得默认逻辑型数据，此外还可以定义逻辑型数据，对于逻辑型数据可以施行逻辑运算。逻辑运算也有一元运算，这就是非运算，运算符为.NOT.，运算结果定义如下：

.NOT.操作数值	.TRUE.	.FALSE.
结果值	.FALSE.	.TRUE.

二元运算有与运算，其运算符为.AND.；或运算，其运算符为.OR.；逻辑等值运算，其运算符为.EQV.；逻辑不等值运算，其运算符为.NEQV.；各运算的结果定义如下：

A. AND. B (A across, B down)	. TRUE.	. FALSE.
. TRUE.	. TRUE.	. FALSE.
. FALSE.	. FALSE.	. FALSE.

A. OR. B (A across, B down)	. TRUE.	. FALSE.
. TRUE.	. TRUE.	. TRUE.
. FALSE.	. TURE.	. FALSE.

A. EQV. B (A across, B down)	. TRUE.	. FALSE.
. TRUE.	. TRUE.	. FALSE.
. FALSE.	. FALSE.	. TRUE.

A. NEQV. B (A across, B down)	. TRUE.	. FALSE.
. TRUE.	. FALSE.	. TRUE.
. FALSE.	. TRUE.	. FALSE.

运算结果的种别类型参数是这样确定的，当两个操作数的种别类型参数相同时，则结果的种别类型参数与操作数的相同；如果两个操作数的种别类型参数不同，则结果的种别类型参数将依赖于处理系统。

第二节 数值类型

对于非字面常量的数值数据都有一个名字，对于名字要说明其类型，具体说明的方法将在下一章中叙述。Fortran 语言预定义了三种数值类型，下面将分别给予介绍。

一、整　　型

取整型值的数据对象被定义为整型数据对象。其种别类型参数由内在询问函数 KIND 送回，其最小范围由内在函数 SELECTED _ INT _ KIND 送回，其十进制幂的范围是由内在函数 RANGE 送回。若没有说明种别类型参数，则默认是 KIND(0)，且数据对象是默认整型的。

整型数据在特定情况下可以用二进制、八进制或十六进制字面常量赋以初值。

说明整型数据的关键词为 INTEGER。

二、实　　型

对于取实型值的数据对象定义为实型数据对象。对于其种别类型参数由内在询问函数 KIND 送回的，其最小十进制范围由内在函数 SELECTED _ REAL _ KIND 送回的，近似方法的十进制精度由内在函数 PRECISION 送回的，十进制幂的范围由内在函数 RANGE 送回的。

说明实型数据的关键词是 REAL。

这里要注意在 Fortran 90中，双精度型认为是特殊种别的实型数据，其种别类型参数值可由 KIND(0. 0D0)求得，而种别类型参数值由 KIND(0. 0)求得的实型数据称为是默认实型。

为了与以前的语言相容，保留了说明默认双精度实型的关键词 DOUBLE PRECISION，但也可不用。

三、复　　型

取复型值的数据对象被定义为复型数据对象。当复型数据的实部和虚部都是默认实型时，此复型数据是默认复型的。

说明复型数据的关键词是 COMPLEX。

第三节　非数值类型

非数值类型是提供给非数值处理的，包括字符型和逻辑型。

一、字　符　型

取字符值的数据对象被定义为字符型数据对象，其种别类型参数由内在询问函数 KIND 送回。若没有说明种别类型参数，则默认的种别类型参数值由 KIND('A')求得，且数据对象是默认字符型的。

说明字符型的关键词是 CHARACTER。

二、逻　辑　型

取逻辑值的数据对象被定义为逻辑型数据对象，其种别类型参数由内在询问函数 KIND 送回。若没有说明种别类型参数，则默认的种别类型参数值由 KIND(.FALSE.)求得，且数据对象是默认逻辑型的。

说明逻辑型的关键词是 LOGICAL。

第四节　数　　组

当数据对象是相同类型和相同种别类型参数的有序的一系列标量时，我们可以把它声明为一个数组。声明的具体形式在下一章给出。

构成数组的各个元素被称数组元素，一个数组元素用数组名

后跟指定符来表示，此时指定符的形式用括号中的下标表达式来表示。有关数组的一些详细介绍将在第五章中介绍。

在这里我们要特别强调Fortran 90中常量与变量、标量与数组这些概念与以往 Fortran 的不同点。

常量有两种，一种称为字面常量，这就是以往所说的常量；另一种称为有名常量，这就是以往所说的常数符号名，新的名称更突出了该数据是常量的特点，但与字面常量的最大差别是它有一个名字。不管那一种常量，它们的值在可执行程序执行期间是不变的。

变量是指它们所具有的值在可执行程序执行期间可能是变化的，即使它们用 DATA 语句给以初始化也不例外。但 Fortran 90 的变量包括了以往的变量和数组。

标量是一数据实体，该实体是用某数据类型的单个值表示的，例如数组元素等。

数组是标量的有序集合，它在 Fortran 90中有许多新的扩充，这将在以后的章节中详细介绍。

小　　结

本章简要介绍了数据类型，其特点是有名字、允许值的集合、表记允许值的方法和操作这些值的运算集合。特别对 Fortran 90 引进了种别类型参数的概念，由于这个概念的引进，使语言表达的数据类型和种别更加丰富，特别是为处理各种非英语字符集奠定了基础，这一点要提请读者特别注意。

习　　题

1. 写出各种整型、实型、双精度实型、复型、逻辑型和字符型字面常量五个例子。

2. 两个整型量相除时，其数学上计算结果与语言的规定有什

么异同点？

3.对于字符型数据进行关系运算时要注意些什么？为什么？

4.你认为引进种别类型参数有什么好处？

5.为了提高程序的可移植性，你认为有关本章的内容中应特别注意些什么方面？

第四章 数据对象的类型说明

前一章已经介绍，数据对象无论是标量变量、数组或函数等都需要有一个名字，对于每个名字要说明该数据对象的类型和可能有的种别类型参数。本章将进一步来介绍这些类型和种别类型参数是如何说明的。对于一个数据对象来讲它有许多属性，类型是它的属性之一，其它的属性及其说明方法将在别的章节中介绍。

第一节 首字母隐含类型法则

Fortran 语言有一传统的约定，当数据对象没有用其它方法说明其类型时，其名字的第一个字母确定其类型。当第一个字母是 I、J、K、L、M 和 N 这六个字母之一时，该名字是整型的，当是其它字母时是实型的。这称为默认的首字母隐含类型法则。

在 Fortran 90中这条法则可用 IMPLICIT 语句来加以确认或否认。

如果在一程序单元中有说明语句：

```
IMPLICIT NONE
```

则说明在该程序单元中不存在首字母隐含类型法则，名字的类型一律采用类型语句来加以说明。

IMPLICIT 语句的另一种形式：

IMPLICIT 隐式说明表

每个隐式说明是类型说明符后跟用括号括着的字母说明表，其中可以是单个字母，也可用两个字母之间以减号连接起来表示一范围。例如：

```
IMPLICIT INTEGER(I-N), REAL(A-H,O-Z)
IMPLICIT INTEGER(I,J), LOGICAL (L)
```

IMPLICIT COMPLEX (Z), CHARACTER (C)

REAL (A－B,D－H)

都是合法的语句。第一个语句的意义实际上就是默认的首字母隐含类型法则。第二个和第三个语句的意义是说首字母为 I、J 的名字是整型的，首字母 L 表示逻辑型的，Z 表示复型的，C 表示字符型的，A、B、D、E、F、G、H 表示实型的。

在书写这语句时要注意三点：

1. 字母说明中出现两个字母之间用减号连起来的项，表示是从前一个字母开始直到后一个字母为止的所有字母，所以在字母的顺序表中前一个字母必须在后一个字母之前。前例中 A－B 与 A、B 是完全一样的，而 D－H 表示 D、E、F、G 和 H 五个字母，所以

IMPLICIT INTEGER (J－I)

是错误的书写。

2. 在一个程序单元的所有 IMPLICIT 语句中，每字母只能说明一种类型。所以

IMPLICIT INTEGER (I－M), LOGICAL (L)

这也是错误的，因为首字母 L 既说明是整型的又说明是逻辑型的。

3. 在一个程序单元中出现了后一种形式的 IMPLICIT 语句，对于没有被说明的字母，其类型仍遵照默认首字母隐含类型法则。

实际上一个程序单元中可以把首字母隐含类型法则看成一映象表，每个字母至多对应一种类型，如果说明 IMPLICIT NONE 则说明映象表是空的。否则按说明建立首字母与类型的映象，对于没有被说明的字母按默认首字母隐含类型法则建立映象关系。

第二节　类型说明符

类型说明符的关键词大家都已熟悉，它们是 INTEGER 整型、REAL 实型、COMPLEX 复型、CHARACTER 字符型及

LOGICAL 逻辑型。此外还有一种特殊种别的实型DOUBLE PRECISION 双精度实型。在 Fortran 90中还增加了可选的种别类型参数说明，如果没有种别说明就是默认的类型，如果有种别说明那就应该有相应的表示数据的方法与之对应。

一、整 型 说 明

整型说明符的形式是：

INTEGER 〔kind－selector〕

而 kind－selector 的形式是：

(〔KIND＝〕 siie)

siie 是整型标量初始化表达式，可以简单地理解为整型字面常量，或整型有名常量（下同）。其值必须为非负，每个值与存在于处理系统的一种数据的表示方法相对应。如果省略 kind－selector 则认为其种别类型参数值为 KIND(0)，即默认整型。例如：

```
INTEGER EXPENDITURE, B
INTEGER (2) X, Y
INTEGER (KIND=3) QUAD
```

都是正确的，说明 EXPENDITURE 和 B 是默认整型的，X 和 Y 是整型的且种别类型参数值为2，QUAD 是整型的且种别类型参数值为3。

二、实 型 说 明

实型说明符的形式是：

REAL 〔kind－selector〕

关于 kind－selector 的形式及意义同前，如果省略则认为其种别类型参数值为 KIND(0. 0)，即默认实型。

为了与 Fortran 77向上兼容，实型说明中还包含双精度实型说明符，它的形式是：

DOUBLE PRECISION

注意它的后面不允许出现 kind－selcetor，而且约定其种别类

型参数值为 KIND(0.0D0)。例如：

```
REAL M12, INT, AREA
REAL (2) DOU _PRE, X
DOUBLE PRECISION DX, DY
REAL (KIND=3) PI
```

都是正确的，并说明名字 M12，INT，AREA 为默认实型的，DOU _PRE 和 X 是实型的且种别类型参数值为2，PI 是实型的且种别类型参数值为3，DX 和 DY 是双精度实型的。

三、复 型 说 明

复型说明符的形式是：

```
COMPLEX [kind-selector]
```

kind－selector 用以说明构成复型数据的实部和虚部两个实值的种别类型参数，若省略则认为其种别类型参数值为 KIND(0.0)，即默认复型。例如：

```
COMPLEX Z1, Z2
COMPLEX (2) COM _VAL
COMPLEX (KIND(0.0)) Z3
```

都是正确的，并说明 Z1，Z2和 Z3都是默认复型的，COM _VAL 是复型的，且种别类型参数值为2。

四、逻 辑 型 说明

逻辑型说明符的形式是：

```
LOGICAL [kind-selector]
```

kind－selector 的意义同前，当省略 kind－selector 时，则认为其种别类型参数为 KIND(.FALSE.)，即默认逻辑型。例如：

```
LOGICAL L1, L2
LOGICAL (2) LOG _VAL
```

说明名字 L1，L2是默认逻辑型的，而 LOG _VAL 是逻辑型的，且种别类型参数值为2。

五、字符型说明

字符型说明符的形式是：

CHARACTER 〔char－selector〕

char－selector 的形式就比较复杂，但总的来说它可能包含两部分的内容，一个是说明字符的长度，一个是说明字符的种别。当两个都说明时不会引起误解，可写为：

(〔LEN＝〕tpv，〔KIND＝〕siie)

这里 tpv 为说明表达式，它和 siie 都可简单地理解为整型字面常量或整型有名常量。

当不是按这顺序书写时，则必须写为：

(KIND＝siie，LEN＝tpv)

如果是默认字符类型时，即种别类型参数值为 KIND('A')时，写法还可进一步简化为：

(〔LEN＝〕tpv)

或　*(tpv)

或　*siic

其中 siic 为非负的整型标量字面常量。例如：

```
CHARACTER (50) NAME, ABSTRACT
CHARACTER (LEN=50) NAME, ABSTRACT
CHARACTER *(50) NAME, ABSTRACT
CHARACTER *50 NAME, ABSTRACT
CHARACTER (20,2) TITLE, ADDRESS
```

这说明名字 NAME，ABSTRACT 为50个字符的默认字符类型，而 TITLE 和 ADDRESS 是20个字符长度的字符型，其种别类型参数值是2。

最后对字符长度的说明还要注意一点，那就是允许在每名字后说明其长度。例如：

```
CHARACTER (20) TITLE, ADDRESS, TEXT*100
```

则说明 TITLE，ADDRESS 是长度为20的默认字符型的，而 TEXT 是长度为100的默认字符型的。也就是说名字后的长度可

以否定类型说明符中的长度说明对该名字的作用。如果在名字后没说明长度，而在类型说明符中也没说明长度，则默认为其长度为1。

六、一个程序单元中名字的类型

在一个程序单元中名字的类型是这样确定的：

(1)首先看它是否在类型说明符之后出现，如果是，则按类型说明符说明的类型确定。

(2)如果没有在类型说明符之后出现，则根据首字母隐含类型法则来确定。

例如，在一个程序单元中有这样一些语句：

```
IMPLICIT INTEGER (X), INTEGER (KIND=2) (Y)
LOGICAL (L)
IMPLICIT CHARACTER (LEN=30) (C,H), COMPLEX(Z)
INTEGER   ZERO, LABLE, HANZI, AREA, Y_VAL
REAL        X_VAL, CDOS, IBM
CHARACTER (20) LOC_VAL, CH * 40
CHARACTER (5, 2) HT
    ......
```

则名字 ZERO，LABLE，HANZI，AREA，I，J，X 和 Y_VAL 均为默认整型；X_VAL，CDOS，IBM，A 均为默认实型；L 为默认逻辑型；YY 为整型且种别类型参数值为2；LOC_VAL 为默认字符型且其长度为20；CH 为默认字符型且其长度为40；CHAR，HANG 为默认字符型且其长度为30；HT 为字符型，其长度为5，其种别类型参数为2。

第三节 属　　性

Fortran 语言处理的数据对象都具有一些性质，除了前面叙述较多的类型(包括种别类型参数)外，还有秩和形状等，这些性质确定了数据的特征和对象的使用方式，它们都被称为数据对象的属

性。属性的说明可以有两种方法，一种是在类型声明语句中加以说明，一种是在属性说明语句中加以说明，本节介绍前一种方法，下节介绍后一种方法，本章先介绍一些基本的属性，有些属性将在以后的章中加以介绍。

一、PARAMETER 属性

在前面经常提及有名常量，其说明的方法就是名字具有 PARAMETER 属性，说明的形式是：

ts, PARAMETER ::实体说明表

其中 ts 为类型说明符，就是前一节所介绍的各种形式，每个实体说明的形式是：

名字＝ 初始化表达式

或

名字(as)＝ 初始化表达式

在未介绍表达式之前，先把表达式理解成字面常量、有名常量或简单的二元运算式。as 是数组说明，详见下段。例如：

```
REAL, PARAMETER :: ONE=1.0, PI=3.1416
INTEGER, PARAMETER :: A(3)=(/1,2,3/)
```

前一个例子说明 ONE, PI 是有名常量，且为默认实型，它们分别有值1.0和3.1416；后一个例子说明 A 是有名常量，是默认整型，它有三个元素分别有值1,2和3。

这里要注意几点：

(1)具有 PARAMETER 属性的数据对象是有名常量，相当于给字面常量取了个名字，在一个程序单元中其值是不能改变的，也就是说 ONE, PI, A 在该程序单元中不能作为变量那样给其赋值。在原先语言中称此为常量符号名。

(2)如果初始化表达式中出现有名常量的话，它必须在前面已经定义。例如：

```
REAL, PARAMETER :: ONE=1.0
REAL, PARAMETER :: PI=2.1416+ONE
```

或

```
REAL, PARAMETER :: ONE=1.0,PI=2.1416+ONE
```

都是正确的。而

REAL, PARAMETER :: PI=2.1416+ONE, ONE=1.0

是错误的。

(3)有名常量不得出现在格式说明中。

属性通常用于给字面常量取名字,这在许多情况下还是很有用的。例如,前面介绍中说到种别类型参数值,汉字的种别值语言没作统一规定,有的处理系统可能规定为2,有的可能是7……,为了减少不同系统上移植的方便,可以使用这样的写法:

```
INTEGER, PARAMETER :: HANZI=2
CHARACTER (LEN=50,KIND=HANZI) TEXT
```

当这程序运行在其它处理系统上时,只需将一个2改为7即可。此外,有名常量在程序中的使用也可提高程序的可读性。

二、DIMENSION 属性

DIMENSION 属性说明实体是数组,其形式是:

ts, DIMENSION (as) :: 实体说明表

每个实体说明可以是个名字,也可以是名字后跟括号括着的as,而 as 在这里我们先介绍一种形式:它是一界对表,每个界对是:

[lb:] ub

这里 lb 称为下界,ub 称为上界,它们都可以是整常量或整型标量变量或更复杂的说明表达式。下界或上界的值是整值,它确定该维的界,以后使用时该维下标表达式的值需要在该界对限定的范围之内,即不小于下界,不大于上界。

在这里对 as 要注意几点:

(1)说明时 lb 省略的话,默认值为1。

(2)说明时一般上界不小于下界,若上界小于下界,则认为该维的范围是空的,此时称该数组的大小为零,也称为零大小数组。

(3)当 DIMENSION 后有(as)而在实体说明中也有(as)时,则该实体以其自带的那个(as)作为说明,而 DIMENSION 后的(as)对它无效;而不带(as)的实体,均以 DIMENSION 后的(as)作

为其说明。

我们称某维说明的上界减下界加1为该维的维展，称各维展的乘积为该数组的大小，表中界对的数为其秩（原 Fortran 77等称为维数）。

我们把一个一维数组，其元素是某数组的各维维展，称为该数组的形状。

（4）当 as 中现变量名字时，该数组必须是虚元。

上述各种明确指明形状的数组称之为显形数组，还有其他类的数组将在以后进行介绍。例如：

```
SUBROUTINE EX (N,A,B,C)
REAL, DIMENSION (N,2:N) :: A, B(N,10),D(10), C
```

说明实体 A 和 C 是大小为 N＊(N－1)的数组，秩为2，形状为(N，N－1)，且为虚元；B 是大小为 N＊10的数组，秩为2，形状为(N，10)，且为虚元；D 为10个元素的数组，大小为10，秩为1，形状为(10)。

三、INTENT 属性

INTENT 属性用于说明虚元的使用意向，它的说明形式是：

```
ts, INTENT (is) :: 实体说明表
```

其中每个 is 是 IN，OUT 或 INOUT 之一。实体说明表中每个实体均是虚元。

INTENT (IN)属性说明这个(些)虚元在过程的执行期间禁止对它重定义或变为无定义。也就是说该虚元是输入参数，对该虚元的定义状态不能改变，其值也不能改变。

INTENT (OUT)属性说明在该过程中引用该虚元之前它必须先已定义，所以与这样的虚元结合的实元必须是可定义的。而在引用此过程的时刻，这样的虚元是无定义的。也就是说该虚元是输出参数，在过程执行过程的某时刻要对其进行定义(赋值)，定义后才能被引用或作为过程的一个输出值。

INTENT (INOUT)属性说明该虚元既用于从被引用的作用

域单位接收数据，又用于向被引用的作用域单位送回数据，所以与这样的虚元结合的实元必须是可定义的。也就是说该虚元既是输入参数，又是输出参数。如果省略这种属性，则将受到与之结合的实元的限制。例如：

```
SUBROUTINE MATMUT (A,B,C,M,N,L)
REAL, INTENT(IN) :: A(M,N), B(N,L),
REAL, INTENT(OUT) :: C(M,L)
```

四、EXTERNAL 属性

EXTERNAL 属性用来说明对象名是一外部函数或虚拟函数，且允许该名字作实元用。这种说明的形式是：

ts, EXTERNAL :: 实体说明表

每实体说明是外部函数名或虚拟函数名。

Fortran 语言允许用户自定义函数，而且对函数名是没有什么限制的，特别是当用户定义了一外部函数，而该函数名与内在函数名重名时，而又要将该函数名作实元时，处理系统进行编译时怎么来区分该实元是用户定义的外部函数名，还是系统提供的内在函数名呢？答案是用此属性说明的就是外部函数名。例如，用户程序是：

```
FUNCTION SIN (X)
    ……
  SIN=……
END FUNCTION SIN

FUNCTION FUN (X,Y)
  REAL, EXTERNAL :: X
  FUN=X(Y)
    …
END FUNCTION FUN

SUBROUTINE SUB1 (A,B)
```

```
  REAL, EXTERNAL :: SIN
    ...
  A=FUN (SIN, B)
    ...
END SUBROUTINE SUB1
```

在子程序辅程序 SUB1中，调用函数 FUN，而函数 FUN 定义一虚元 X 是虚拟函数，所以调用时需要一实元为函数名，程序中前一个包括 EXTERNAL 属性的语句说明 X 是虚拟函数名，而后一个说明 SIN 是外部函数名，而且该外部函数名是有定义的。

五、INTRINSIC 属性

INTRINSIC 属性用来说明一些对象名是内在函数的特定名或类属名，当是特定名时，允许该名字用作实元用。

这种说明的形式是：

ts, INTRINSIC :: 实体说明表

每个实体说明必须是内在函数的特定名或类属名，其名字见第十一章。

这属性说明与 EXTERNAL 属性是相对应的，用于区分一个函数名究竟是外部函数名、虚拟函数名、还是内在函数名。例如，若在前一例子中还有一子程序辅程序是：

```
SUBROUTINE SUB2 (A, B)
  REAL, INTRINSIC :: SIN
    ...
  A=SIN(B)
    ...
END SUBROUTINE SUB2
```

虽然这程序中自定义了外部函数 SIN，但在 SUB2中所调用的函数 SIN，仍是调用的内在函数 SIN，而不是自定义的外部函数。其区别在于在 SUB2中说明了 SIN 的 INTRINSIC 属性。

六、SAVE 属性

我们知道在 Fortran 语言中名字的值和它的一些状态是局部

于一个程序单元的，除非这些名字是虚元、函数结果、公用块元素等，当执行 RETURN 语句或 END 语句后，其值和一些状态是不再保留的，当我们想要保留其值和一些状态时可采用把它说明成具有 SAVE 属性的办法。

当名字具有 SAVE 属性时，在包含这个声明的作用域单位中执行了 RETURN 语句和 END 语句后，它们的结合状态、分配状态、定义状态和值是被保留的。我们称这样的对象为保留对象。

显然在主程序中可以出现 SAVE 属性，但实际上不起作用。

这种说明的形式是：

```
ts, SAVE :: 实体说明表
```

例如：

```
REAL, SAVE :: X, A(10), IC
```

都是正确的。它说明 X, A(10) 和 IC 都是保留对象。

七、OPTIONAL 属性

OPTIONAL 属性用于说明虚元，说明该过程在被引用时，该虚元不一定需与实元相结合。

其形式是：

```
ts, OPTIONAL :: 虚元名表
```

例如：

```
SUBROUTINE SUB3 (A,B,C,M,N,L,D)
  REAL, INTENT(IN) :: A(M,N), B(N,L)
  REAL, INTENT(OUT) :: C(M,L)
  REAL, OPTIONAL :: D
    ...
END SUBROUTINE SUB3

SUBROUTINE SUB4
  REAL,DIMENSION::X(10,20),Y(20,10),Z(10,10)
    ...
  CALL SUB3(X,Y,Z,10,20,10)
```

```
    ...
    END SUBROUTINE SUB4
```

SUB3的第四个语句说明虚元 D 在调用时可以没有实元与之对应，执行调用时可以没有与其结合的实元。SUB4的 CALL 语句是一例，其用途将在后面的章节中介绍。

第四节　基本属性说明语句

在前一节介绍了数据的属性，和在类型语句中如何声明数据对象具有那些属性，本节再介绍一下如何用一些基本属性说明语句来说明这些属性。

一、PARAMETER 语句

PARAMETER 语句提供定义有名常量的方法。PARAMETER 语句定义的有名常量与在类型语句中说明 PARAMETER 属性的有名常量具有相同的性质和限制。

PARAMETER 语句的形式是：

PARAMETER (有名常量定义表)

每个有名常量定义项的形式是：

有名常量 = 初始化表达式

在此可把初始化表达式简单地认为是常量、及内在运算符组成的表达式。但是特别要注意的是对其中名字的类型要先加以说明，例如：

```
PARAMETER (LENGTH=3.2)
REAL LENGTH
```

在执行 PARAMETER 语句时，按默认首字母隐含类型法则，LENGTH 为整型而不是实型，这例子是错误的。但若把语句的次序颠倒一下，即：

```
REAL LENGTH
PARAMETER (LENGTH=3.2)
```

就是正确的，它说明有名常量 LENGTH 为实型的，且其值为3.

2，具有 PARAMETER 属性。

其实写成：

```
REAL, PARAMETER :: LENGTH=3.2
```

也是正确的。所以我们可以认为类型语句中属性说明符的出现提供了简化编程序者书写的一个方法。

和前面的例子一样

```
PARAMETER (PI=2.1416+ONE, ONE=1.0)
```

是错误的，而

```
PARAMETER (ONE=1.0, PI=2,1416+ONE)
```

则是正确的。

二、DIMENSION 语句

DIMENSION 语句说明数据对象具有 DIMENSION 属性。

DIMENSION 语句的形式是：

DIMENSION〔::〕数组名(as)，数组名(as)，…

其中每个 as 是前面已介绍过的数组说明。例如：

```
REAL IC
DIMENSION A(10), B(10), IC(10), D(1:10,-3:6)
```

等都是正确的，其实类型语句也给出了简化书写的方法，一方面是可将类型与 DIMENSION 属性在一个语句中书写出来，另一方面是将相同的(数组说明)可以简化，例如上句可以简写为：

```
REAL, DIMENSION (10) :: A, B, IC, D(1:10,-3:6)
```

要注意的是在 DIMENSION 语句中只能写成每个名字后自跟数组说明而不能简化。例如：

```
DIMENSION (10) :: A, B, IC, D(1:10,-3:6)
```

是错误的语句。

三、INTENT 语句

INTENT 语句用来说虚元名字的使用意向，在该语句中说明的名字必须是虚元，并且有 INTENT 属性。

INTENT 语句的形式是：

INTENT(is)〔::〕虚元名表

其中 is 是前述的 IN、OUT 或 INOUT 之一。例如:

```
SUBROUTINE MATMUT (A, B, C, M, N, L)
INTENT (IN):: A(M,N), B(N,L)
INTENT (OUT):: C(M,L)
```

四、SAVE 语句

SAVE 语句说明有名的对象具有 SAVE 属性。

SAVE 语句的形式是:

SAVE 〔〔::〕保留实体表〕

其中保留实体表的项可以是对象名或/公用块名/。注意,这里不允许出现公用对象的实体名,但可出现公用块名,一旦在某一非主程序中这样出现,必须在包含该公用块名说明的所有程序单元中均作同样的说明。

此外,当在一个程序单元中出现省略保留实体表的形式时,要求该程序单元中不再包含任何 SAVE 语句,或类型语句中不再包含任何 SAVE 属性。它的作用相当于说明了所有对象全部为保留的对象。

五、OPTIONAL 语句

OPTIONAL 语句说明虚元名字具有 OPTIONAL 属性。

OPTIONAL 语句的形式是:

OPTIONAL〔::〕虚元名表

例如:

```
SUBROUTINE EX (A, B, C)
OPTIONAL :: A
```

六、DATA 语句

DATA 语句用来对变量提供初值。

既然 DATA 语句是对变量提供初值,只允许对一变量提供一次初值而不能是多次。

由于提供初值，对变量来讲就有类型问题，若在 DATA 语句之前，名字没有用类型语句声明其类型，则将按首字母隐含类型法则来确定，而在 DATA 语句后再用类型语句声明其类型时，不能与前者相矛盾，否则为非法。

由于得到初值的可能是数组元素，所以这样的数组名字应在先前被说明具有 DIMENSION 属性。

DATA 语句的形式是：

DATA〔::〕变量表/初值表/〔,〕变量表/初值表/…

DATA 语句中变量及提供的初值之间的形式是：

变量表/ 初值表/

变量表可以将一系列变量按表的形式列出。例如：X，Y，Z，A(1)，A(2)，A(3)，…，A(10)对于后者可以有简单的形式表示即：

(A(I)，I=1，10，1)

来替代 A(1)到 A(10)，当然也可以有更复杂的，例如：

(((B(I,J,K)，I=1,2)，J=1,2)，K=1,2)

表示 B(1,1,1)，B(2,1,1)，B(1,2,1)，B(2,2,1)，B(1,1,2)，B(2,1,2)，B(1,2,2)，B(2,2,2)这八个元素，其意义类似于 DO 循环，称为隐 DO 表。

当然如果数组 A,B 分别是十个、八个元素也可分别写成数组名 A,B。

对于初值表可以列出一系列常量，这些常量可以是字面常量也可以是有名常量，甚至结构构造符。

例如，3.5，4.7，18.9，1.0，1.0，…，1.0对于后面这十个1.0可以简写为10＊1.0。

下面是一个 DATA 语句的例子：

```
DIMENSION A(10), B(2,2,2)
DATA  X, Y, Z, (A(I),I=1,10,1),B/3.5, 4.7, 18.9 10*1.0,&
     11.0,12.0, …, 18.0/
```

此外，要特别注意名字的类型与初值的类型之间的关系，若

名字是逻辑型或字符型的，则初值必须是相同类型的。当名字是数值的，初值也应该是数值型的，当具体类型不同时，按赋值规则进行转换。当名字是整型时，还允许二进制、八进制或十六进制的字面常量作初值。例如：

```
CHARACTER (LEN=10) NAME
INTEGER, DIMENSION (-5:5) ALPHA
REAL, DIMENSION (10,10) :: KEY
DATA NAME / 'BLACK COW'/, II / B'01010011'/
DATA ALPHA / 11 * 0/, ((KEY(I,J), J=1,I), I=1,10)&
    /55 * 0.0/
DATA ((KEY (I,J), J=I+1, 10) I=1,9)/ 45 * 1.0/
```

小　　结

本章介绍了数据对象确定类型的规则，IMPLICIT 语句和类型语句的形式，以及类型语句中的 PARAMETER 属性，DIMENSION 属性，INTENT 属性，EXTERNAL 属性，INTRINSIC 属性，SAVE 属性和 OPTIONAL 属性。此外还介绍了 PARAMETER 语句，DIMENSION 语句，INTENT 语句，SAVE 语句，OPTIONAL 语句和 DATA 语句。

通过这些内容的介绍使读者了解到如何来对一名字确定其类型、类型参数及其它的一些属性，以至于如何给变量以初始定义等。要编写一严格和高质量的程序，这些都是必须掌握的。

习　　题

1. 首字母隐含类型法则是什么？如何使 Fortran 语言能实现“强类型”(即对每个名字都必须说明才能使用)？

2. 试利用类型语句中的各种属性和属性语句来分别编写两份程序，说明其异同点。

第五章　数据对象的使用和存储结合

一个数据对象的名字出现在某程序的上下文中，当使用其值时称为引用，当使其变为具有值时称为定义。

第一节　标量对象的使用

标量是一种数据实体，它用一种数据类型的单个值来表示，它不是数组，秩为0。

如果标量已有定义，则其值是表征其数据类型的值的集合中的一个元素。

一、常　　量

常量有两类，它们是字面常量和有名常量。

字面常量是标量，这种标量由字面的构成指明其类型、类型参数和值。

有名常量是一个与具有 PARAMETER 属性的名字相结合的常量。

对于常量来讲其值可以被引用但不能被重定义。所以有名常量虽然出现的形式是一名字，但它是常量的标记（名字），与变量是不同的。

二、变　　量

变量顾名思义是可变化的量，在程序中以名字的形式出现，而这个名字不具有 PARAMETER 属性，它是在程序中出现最多的数据对象。

变量具有类型和类型参数，这约束了其取值的形式及范围。当

程序中确定其值时称为定义，而使用其值时称为引用，定义了的变量可以重定义或变成无定义。例如，在程序单元中使用 DATA 语句可使其具有初值，但在执行过程中可以再对其赋值而重定义，如果不具有 SAVE 属性的话，当该程序单元执行 RETURN 语句或 END 语句时它就变为无定义的。

注意：在 Fartran 90中变量的概念与以往的有所不同，它包含了两种变量，一种是标量，一种是数组。具有 DIMENSION 属性的名字称为变量但它是一数组。

三、子　　串

子串是字符串中的一邻接部分，而原来的字符串就称为父串。子串用在父串的名字后跟用括号括着的始点位置与终点位置来表示，例如：

```
A(1) (1:5)
IX(10:12)
'ABCDEF' (N:N)
```

都是子串，其中 A(1)是字符数组元素，A(1)(1:5) 表示由 A(1)中第1个到第5个字符构成的字符串；IX 是字符变量，IX(10:12) 表示由 IX 的第10个到第12个字符构成的字符串；而最后一例子是字符字面常量的一子串，但由字面常量中的那几个字符组成要由 N 的值确定。注意表示中的冒号分隔符是不可少的，始点可以省略，此时默认值为1，终点也可以省略，此时默认值为父串的字符长度值。始点值一般不大于终点值 而且两者均不小于1，不大于父串的字符长度值。所以在第三个例子中$1 \leqslant N \leqslant 6$。当始点值超过了终点值时，认为该子串是空串，例如 A(1) (5:4)就是空串。

当父串是变量时，子串也是变量，前两个例子就是这种情况。

四、标　　量

标量是一种数据实体，它能用一种数据类型的单个值来表示，它不是数组。所以标量与数组的区别在于它们的秩不同，标量的秩

为0，而数组的秩为1至7。根据子串的定义我们知道标量可以是变量的一部分，也可以是常量的一部分。当变量中要求是标量时，一般指明它是标量变量。所以在子串中特别指出当父串是变量时，子串也是变量，而例子 'ABCDEF'(N:N)是标量，但不是变量。

第二节　数组及有关对象的使用

在前面的章节中我们已介绍了一些数组的基本性质，在此节中我们将对数组作一完整的介绍。

一、数　组

数组是具有相同类型和种别类型参数的标量数据的一有序集合，其中的元素是按矩形的形式排列的。

一完整数组是一有名数组。完整数组变量是一变量名，而该变量是一数组，它的名字后不跟下标表。

当完整数组变量出现在表达式或其它可以出现的地方，表示该数组的全部元素。

完整数组也可是一完整数组有名常量，这就是说该数组名可用前述 PARAMETER 属性来说明或用相应的 PARAMETER 语句来说明。例如：

```
INTEGER, DIMENSION (5), PARAMETER :: ORDER=&
      (/1,2,3,4,5/)
```

说明 ORDER 是完整数组有名常量，该数组秩为1，大小为5，形状为(5)，而且是常量。

下面我们按数组在 DIMENSION 属性中说明的不同形式来分别介绍几类数组。

1. 显形数组与自动数组

如果数组说明的每维界均为表达式而且维上界不省缺时，则称之显形数组。其最常见的形式是上下界均为字面常量或有名常量。

当表达式中出现非常量时，则该显形数组或为虚元或为自动数组。例如：

```
SUBROUTINE MTRMUL (A,B,C,M,N,L)
DIMENSION A(M,N), B(N,L), C(M,L),D(M)
```

其中 A,B,C 是虚元。

自动数组是在过程辅程序中声明的显形数组，它不是虚元但其说明的界中含有非常量。如上例中 D 不是虚元，但界的说明表达式中含有变量 M，故是一个自动数组。

这里要注意的是当显形数组的界说明表达式中含有非常量时，这非常量的值是在过程入口处确定的，从而这数组的界和形状也就随之确定。在过程执行期间，这非常量的值可能变化，可能变为无定义，但这均不再影响数组的界和形状。

2. 假取形数组

由前可知，数组每一维的维展是上界与下界之差加1，而数组的形状是一个一维数组，其中各个元素依次是每维的维展。例如：

```
DIMENSION A(-1:3,10)
```

数组 A 的各维展分别是5与10，故其形状为(5,10)。

当下界说明符省缺时，默认值为1。但当上界说明符不出现时，此数组就无法从说明直接得到其维展和形状，在 Fortran 90中允许出现这种情况，此时称这种数组为假取形数组，同时语言规定假取形数组只能是虚元，其维的维展取与其结合实元数组相应维的维展，因此可以说该数组的形状是取与它结合的实元数组的形状。根据这个规定不难算出假取形数组每维的上界。若假取形数组的某一维下界说明符的值为 d(当省略时默认值为1)，而与之相结合的实元相应维的维展是 s，则在虚元与实元相结合后，该维的上界是 d+s-1。例如：

```
SUBROUTINE EXP1 (A,M,N)
  DIMENSION A (M:, M+N:)
    ……
END SUBROUTINE EXP1
PROGRAM ASSHAPE
```

```
    DIMENSION D(10,20)
      ......
    CALL EXP1 (D,-1,+3)
      ......
  END  PROGRAM ASSHAPE
```

执行CALL语句时，在过程入口处进行变元结合，此时得到第1维的下界为－1，维展为10，则其上界为8；第2维的下界为2，维展为20，则其上界为21。即此时假取形数组 A 秩为2，形状为(10,20)。

语言规定假取形数组说明时，其每一维的上界说明表达式均不出现，不是有的维可出现，有的维不出现，这是要注意的地方。

3. 迟形数组

迟形数组有两种：一种是数组指针；一种是可分配数组。数组指针在介绍指针时再介绍，在此我们介绍可分配数组。

从数组说明的形式来看，其上、下界说明表达式均不出现，也就是说仅有冒号。为了与假取形数组省略下界说明表达式相区别，语言规定必须说明 ALLOCATABLE 属性。这种属性可在类型语句中说明，也可在 ALLOCATABLE 语句中说明。例如：

```
  REAL, ALLOCATABLE, DIMENSION (:) :: E
  ALLOCATABLE :: A(:),B(:,:)
```

均是正确的说明，它们分别说明 E 和 A 是秩为1的两个迟形数组，B 是秩为2的一个迟形数组。

从上例可见从说明来说仅说明了数组的秩，而没有说明其形状。其界和形状将在执行 ALLOCATE 语句对数组分配空间时确定。

4. 假取大小数组

假取大小数组在以前的标准中称为假定大小数组，假取大小数组是一虚元，其大小取自与其结合的实元。实元数组与虚拟数组的秩和维展可以不同，虚拟数组仅假取实元数组的大小。

假取大小数组的说明与显形数组说明的差别是其最后一维的上界说明表达式是 *，其余形式均一样。例如：

```
  SUBROUTINE EXP2(A,B)
```

```
DIMENSION A(2:11,5:*), B(*)
```

都是正确的说明。例中说明 B 是一秩为1的僭取大小数组，A 是一个秩为2的僭取大小数组。

僭取大小数组的大小是这样确定的：

(1)若实元是除默认字符型外的任何类型的数组，则虚拟数组的大小是实元数组的大小。

(2)若实元是除默认字符型外的任何类型的数组元素，该数组元素的下标次序值是 r，相应实元数组的大小是 x，则虚拟数组的大小是 x－r＋1。

(3)若实元是默认字符型的数组，数组元素或数组元素子串，数组具有 C 个字符存储单元，从第 t 字符存储单元开始，则虚拟数组的大小是：

```
MAX(INT((C-t+1)/e),0)
```

其中 e 是虚拟字符数组的一个元素的长度。例如：

```
DIMENSION X(10,50), Y(500)
```

与上例中 A(2:11,5:*)相结合的实元是数组 X，则 A 的大小为500；若是数组元素 X(1,2)与它结合，则 A 的大小为490。注意 Y 也可与 A 结合，则 A 的大小也是500；若 Y(100)与之结合，则 A 的大小就是400。

语言中没有明确规定 KIND 不为 KIND('A')的字符型时，僭取大小数组的大小的确定法则。由于语言对此种字符型数据每个字符所占存储单元数未作规定，所以依赖于实现，但是上述3种确定方法原则上仍是可以用的。

由于僭取大小数组最后一维没有形状，所以除了在某些特殊情况下(不要求其形状时)一般不作完整数组引用。由于除了其最后一维的上界说明表达式为 * 外，其余的形式与显形数组的界说明表达式相一致，所以对显形数组的一些限制和规定在此也适用，就不多说了。

二、数 组 元 素

数组元素是一标量，关于数组元素的一些有关术语 Fortran

90与以往的是不同的，这点希望特别加以注意。数组元素是数组名后用括在括号的下标表来表示，每个下标表达式称为下标(原先将括在括号内的下标表达式表统称下标)，数组元素按先后次序形成一序列，称为数组元素次序。一个数组元素在这个序列中的位置用该元素的下标表的下标次序值(原称下标值)来确定。下标次序值的计算公式见表5－1。

表5－1

秩	下标界	下标表	下标次序值
1	$j_1:k_1$	S_1	$1+(S_1-j_1)$
2	$j_1:k_1, j_2:k_2$	S_1, S_2	$1+(S_1-j_1)+(S_2-j_2)\times d_1$
3	$j_1:k_1, j_2:k_2,$	S_1, S_2, S_3	$1+(S_1-j_1)+(S_2-j_2)\times d_1$
	$j_3:k_3$		$+(S_3-j_3)\times d_2\times d_1$
…			
7	$j_1:k_1, \cdots, j_7:k_7$	$S_1, \cdots, S_7$	$1+(S_1-j_1)+(S_2-j_2)\times d_1$
			$+(S_3-j_3)\times d_2\times d_1$
			$+\cdots$
			$+(S_7-j_7)\times d_6\times d_5\times\cdots\times d_1$

式中 $d_i=\max(k_i-j_i+1, 0)$是第 i 维的大小。若数组的大小是非零的，即所有 di 均不为0，则对所有 $i=1,2,\cdots,7$有 $j_i\leqslant S_i\leqslant k_i$。

三、数 组 片 段

数组片段是数组的一部分，它提供了按某种规则在数组中选择一系列元素构成一个数组的方法。有两种表示形式，一种是下标三元组，一种是向量下标。

1．下标三元组

下标三元组的形式是：

〔下标表达式1〕:〔下标表达式2〕〔:下标表达式3〕

当下标表达式1省略时，等价于数组说明中该维的下界，下标表达式2省略时，等价于该维的上界，下标表达式3是下标值的增

量，称为跨距。其值禁止为0，如果省略则认为是1。

由上述可知在假定大小的数组中，最后一维的上界是 * ，所以相应的下标表达式2一定不可省略。

数组 A 由上述下标三元组指明的数组片段被表示为：

A([下标表达式1]:[下标表达式2][:下标表达式3])

当跨距是正的时，用三元组说明的下标构成了一个有规则的下标序列，从下标表达式1的值开始，按跨距作增量直至一个不大于下标表达式2的值的最大整值。由下标三元组指明的数组片段就是由这个下标序列指明的数组元素构成的。若下标表达式1的值大于下标表达式2的值，则该下标序列是空的，相应的数组片段的大小为零。

例如，若数组说明为 A(20)，则数组片段 A(3:16:3)由数组元素 A(3)，A(6)，A(9)，A(12)，A(15)五个元素构成。

注意，数组片段中所选的数组元素的各个下标值都必须在原数组说明的上下界之间，但是下标表达式1或2的值可能越界，例如对数组声明为 A(20)，数组片段 A(3:21:4)，该片段由数组元素 A(3)，A(7)，A(11)，A(15)，A(19)构成，下标表达式2的值为21超过说明的20是可以的。

当跨距是负的时，同样从下标表达式1的值开始，按跨距作增量向小的方向直到一个不小于下标表达式2的最小值为止。当下标表达式2的值大于下标表达式1的值时，该序列是空的，即该数组片段的大小为零。

例如，A(19:0:—4)这个数组片段是由数组元素 A(19)，A(15)，A(11)，A(7)，A(3)构成的。同样这里的0超出了说明中默认的下界1是允许的。

再如，假定数组声明为 B(10,6,4)，则数组片段 B(6:11:2,5,2:3)是形状为(3,2)的数组：

B(6,5,2)，　B(6,5,3)

B(8,5,2)，　B(8,5,3)

B(10,5,2)，　B(10,5,3)

2. 向量下标

向量下标同样也可用来指明一个下标序列，这个序列由向量的每个元素依次的整值来决定，假定 U 是一个一维数组，其值是：

U=(/1,13,5,7/)

数组 A 声明为 A(20)，则数组片段 A(U)表示由数组元素 A(1)，A(13)，A(5)，A(7)构成的数组。

由于在向量下标的情况下，数组片段的构成元素取决于向量的各分量的值，而这些值之中允许有若干个是相同的，例如：

V=(/1,13,5,7,5/)

V 中有两个分量的值为5，则数组片段 A(V)由五个数组元素构成，它们分别是 A(1)，A(13)，A(5)，A(7)，A(5)。这时称数组片段是多对一的。一个多对一的数组片段是具有二个或多个相同值的元素的向量下标指明的数组片段。在这种情况下，该数组片段不得出现在赋值语句赋值号的左部，也不得作为 READ 语句的输入项。向量下标也可以定义多维数组片段，在此我们看一个例子。假定 Z 是形状为(10,5)的二维数组，U 和 V 的值分别是：

U=(/3,7,5/)

V=(/3,1,1,5/)

则 Z(U,V)由元素

Z(3,3)，Z(3,1)，Z(3,1)，Z(3,5)

Z(7,3)，Z(7,1)，Z(7,1)，Z(7,5)

Z(5,3)，Z(5,1)，Z(5,1)，Z(5,5)

所组成。

第三节 动态结合

上一节介绍的可分配数组，其存储空间的分配是可以动态控制的，当执行 ALLOCATE 语句时才对其分配空间，此后数组是已分配的。当执行 DEALLOCATE 语句时回收对其分配的存储空间，此时数组变成去分配的，或称未分配的。下面分别介绍这两个语句。

一、ALLOCATE 语句

ALLOCATE 语句的形式是：

ALLOCATE(可分配数组名(界表达式表)〔,STAT=V〕)

其中界表达式表为一系列用冒号分隔的下界表达式和上界表达式的对，当然，在此也允许各种省略的形式。这些表达式应由整型的标量所构成，而且在该语句执行时其值都应已是确定的。执行此语句时，按计算出的界、形状和大小来分配存储空间，并将此可分配数组的状态变为已分配的。对已分配的可分配数组才可对其定义值。

STAT=V 是可选的，其中 V 是整型标量变量。当成功执行此语句时，V 以0来定义，当执行此语句有错误情况发生，V 得一正整数值，其具体数值及意义由处理系统决定。

例如对一个已分配的可分配数组再执行 ALLOCATE 语句就是一种错误。

最后还要指出在语句中省略了 STAT=V 时，一旦执行此语句时发生了错误，可执行程序就终止执行。

语句的例子是：

ALLOCATE(A(M), X(-5:N, 10))

二、DEALLOCATE 语句

DEALLOCATE 语句的形式是：

DEALLOCATE(可分配数组名〔,STAT=V〕)

执行此语句后将收回对可分配数组分配的存储空间，并且此数组变为去分配的。

其中 STAT=V 的意义与前虽相同。例如对未分配的可分配数组执行该语句就是一种错误，结果将使分配状态变成不定义的。

语句的例子是：

DEALLOCATE (A,X)

这里要指出在执行 RETURN 或 END 语句时，若该可分配数

组已分配，且具有 SAVE 属性，则其分配状态及其定义状态将得到保留，否则其分配状态和定义状态都变成未定义的。对已分配空间的可分配数组若值未定义，则禁止对它的引用。若可分配数组的分配状态未定义，也禁止对其引用、定义、分配和去分配。所以一般来说用户要特别注意 DEALLOCATE 语句的使用。

下面给出应用这二个语句的一个综合的例子：

```
      SUBROUTINE MATMUT (A,B,C,M,N,L)
        DIMENSION A(M,N), B(N,L),C(M,L)
        REAL, DIMENSION(:), ALLOCATABLE :: D,E
        INTEGER V
        ALLOCATE (D(N), E(L), STAT=V)
        DO 30 I=1, M
         DO 20 J=1, L
          D(1:N)=A(I,1:N) * B(1:N,J)
          E(J)=0.0
          DO 10 K=1, N
           E(J)=E(J)+D(K)
10        CONTINUE
20       CONTINUE
         C(I,1:L)=E(1:L)
30      CONTINUE
        DEALLOCATE (D,E, STAT=V)
      END
```

第四节　存 储 结 合

由计算机和 Fortran 语言的发展历史可知 Fortran 语言是历史上最早的高级程序设计语言之一，当时一方面由于计算机的存储器资源可贵，另一方面 Fortran 语言成分遗留了相当一部分手编机器语言编程序的风格的影响，所以在 Fortran 语言中保留了使用者对存储分配的一些控制手段，这主要体现在存储结合的部

分。这种手段随着计算技术的发展，存储器件的价格下降，特别是使用不当容易造成较难发现隐患的缺点，建议尽量少用这方面的功能。但考虑到语言的向上兼容性，和人们可能阅读或用到以前的程序，所以在此还简单介绍一下有关的内容。

一、EQUIVALENCE 语句

该语句用来说明在一个作用域单位中两个或多个对象共享存储单元，这导致共享存储单元的对象的存储结合。

1. 语句的形式

语句的形式是：

EQUIVALENCE　等价集表

每个等价集形式是：

(等价对象表)

其中等价对象可是变量名，数组元素指定符或子串。

这里强调几点：

(1)等价是两个或多个对象的等价，所以等价对象表中至少出现两个等价对象。

(2)等价语句表示存储单元的共享，当它们的类型或类型参数不同时，语句并不包含类型和属性等的转换。特别是当变量名分别为标量和数组时，既不意味着数组具有标量的性质，也不意味着标量具有数组的性质。

(3)等价对象必须是已确定分配存储单元的对象，对于虚元，可分配数组，有名常量等在说明时不分配存储单元或另外专门分配存储单元的不能作为等价对象出现。

(4)同样的道理，当等价对象是数组元素指定符或子串时，其下标表达式或子串范围都应该是整的初始化表达式。也就是说，在处理该语句时下标值或子串范围值都必须是能确定的。

(5)由于等价将引起存储结合，在分配存储单元时，语言将默认的算术型和逻辑型当作一类，而默认的字符型当作另一类，不再定义两类之间的关系。所以在等价集中所有等价对象都必须属于

同一类的。

(6)为了便于可移植性，语言还规定对于非默认的内在类型，将每一种类型的每一种种别各分为一类，在等价集中所有等价对象也必须同属一类。

(7)当等价对象表中出现不用下标限定的数组名时，这与标识该数组第一个元素的数组元素指定符起着相同的作用。

2. 等价结合

在一个等价集中的所有大小不为零的等价对象存储序列具有相同的第一个存储单元，若出现大小为零的序列，所有大小为零的序列也是存储结合的，并与大小不为零的序列具有相同的第一个存储单元。这就是说一个等价集中的数据对象的存储单元是结合的，这被称为数据对象之间的等价结合。注意这可能导致与其它数据的对象的存储结合。

这种结合对于默认字符型的数据也一样成立，值得注意，语言并不要求等价对象长度相同。例如：

```
CHARACTER (LEN=4) :: X, Y
CHARACTER (LEN=3) :: C(2)
EQUIVALENCE (X, C), (Y, C(2))
```

这样存储结合的关系见表5－2。

表5－2

<table>
<tr><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td></tr>
<tr><td colspan="3">C(1)</td><td colspan="3">C(2)</td><td colspan="3"></td></tr>
<tr><td colspan="4">X</td><td colspan="5"></td></tr>
<tr><td colspan="3"></td><td colspan="4">Y</td><td colspan="2"></td></tr>
</table>

上表可知，上述 EQUIVALENCE 语句导致 X 与 Y 也有部分存储是结合的。

3. 限制

(1) EQUIVALENCE 语句禁止将同一存储单元在一个存储序列中说明多次。例如：

```
REAL A(2), B
```

```
EQUIVALENCE (B, A),(A(2), B)
```

后一个语句是不允许的。因为它将数组元素 A(1)、A(2)说明成相同的存储单元。

(2) EQUIVALENCE 语句禁止将邻接的存储单元说明为不邻接的。例如：

```
CHARACTER (LEN=4) :: X(3)
CHARACTER (LEN=3) :: Y(2)
EQUIVALENCE (X(2),Y), (X(3),Y(2))
```

```
1   2   3   4   5   6   7   8   9   10   11   12   13
|---X(1)--|----X(2)----|---X(3)------|
           |--Y(1)--|    |--Y(2)--|
```

原来数组元素 Y(1)与 Y(2)的存储单元应该邻接的，现在变成非邻接了，这也是不允许的。

二、COMMON 语句

COMMON 语句说明称为公用块的物理存储块，在可执行程序的任何用 COMMON 语句说明了该存储块的作用域单位中都是可访问的。于是，该语句提供了基于存储结合的全局数据功能。

1. COMMON 语句的形式

COMMON 语句的形式是：

COMMON 公用表

其中每个公用表的形式是：

/公用块名/公用块对象表

每个公用对象可以是标量变量名，数组元素指定符，数组名等。

语言规定：

(1)一个给定的对象，在所有公用块对象表中只能出现一次。

(2)公用块对象是已确定分配存储单元的对象，对于虚元、可分配数组、有名常量等在说明时不分配存储单元或另外分配专门存储单元的不能作为公用块对象出现。

(3)同样的道理，公用块对象是数组元素指定符时，其下标表达式应是整的初始化表达式，即处理 COMMON 语句时下标表达式的值应是已确定的。

(4)当公用块名省略时称为空白公用块，当空白公用块出现在 COMMON 语句的第一个公用表中时，两个斜线也可省略。这样说明的公用块对象表中所有对象都被声明属于空白公用块。当公用块名不省略时，称此公用块为有名公用块，此时相应的公用块对象表被声明属于该有名公用块。

在一个作用域单位中对于所有具有相同公用块名(包括所有空白公用块)的公用块对象表，按出现的先后次序自动合并成一张公用块对象表。例如：

```
COMMON I,J,K,/BLA/A,B,D(10),//L,X(-1:5)
COMMON /BLA/C(5,5),//M,Y(100)
```

是正确的语句，它和

```
COMMON I,J,K,L,X(-1:5),M,Y(100),/BLA/A,B,D(10),&
          C(5,5)
```

具有一样的作用。

2. 公用块存储序列

对于每个公用块，公用块存储序列构成如下：

(1)公用块的存储序列是由在公用块对象表中所有数据对象的存储序列所包含的存储单元的序列所构成的。其次序与对象表出现的次序相同。

(2)上述构成的存储序列还包括用与其等价结合而构成的任何存储序列的所有存储单元的扩展。这种扩展只能在最后一个存储单元之后增加存储单元。与公用块中一个实体相结合的数据对象被认为是属于该公用块。

(3)公用块存储序列的大小就是公用块的大小。

上例中空白公用块的大小为112个数值存储单元，BLA 为37个数值存储单元。

3. 公用结合

在一个可执行程序中，具有相同名字（包括空白公用块）的所有大小不为零的公用块的公用块存储序列有相同的第一个存储单元，所有大小为零的公用块的公用块存储序列彼此也是结合的，并也具有与大小不为零的公用块相同的第一个存储单元。这导致不同作用域单位中的对象的结合称为公用结合。

和等价结合一样，语言将默认的算术型和逻辑型归为一类，默认的字符型另作一类，一个公用块的所有公用块对象都必须属于同一类的。对非默认的内在类型，语言将每种类型的每一种种别各分为一类，要求一个公用块的所有公用块对象也必须属于同一类。

这里特别要注意在相同公用块名中存储序列是按公用块对象出现的顺序排列的，也就是所谓"按位公用"。例如：

```
SUBROUTINE  S1(A,B)
COMMON I,J,K /BLA/ X(10),Y,Z(3,3)
  ...
SUBROUTINE S2(X,L)
COMMON /BLA/Y, A(10),Z(3,3),//M,K,J,P(5)
  ...
```

两个子程序中S1的I与S2的M；S1的J与S2的K，S1的K与S2的J分别是存储结合的，它们都属于空白公用块。S1的X(1)与S2的Y，S1的X(2)与S2的A(1)，……，S1的X(10)与S2的A(9)，S1的Y与S2的A(10)，S1的Z(1,1)与S2的Z(1,1)，……，S1的Z(3,3)与S2的Z(3,3)分别是存储结合的，它们都属于BLA公用块。特别注意S1的J,K分别与S2的K,J相结合；这是由它们在公用块对象表中出现的次序所决定的。

4. 有名公用块与空白公用块的差别

从形式上看空白公用块与有名公用块有许多一样的性质，但有时是有差别的，不能把空白公用块简单地看作名为空白的有名公用块。其差别主要有：

(1)在执行RETURN或END语句后，除非有名公用块名在SAVE语句中声明，否则有名公用块中所有数据对象变为无定义

的。而空白公用块中的对象则定义状态不变。

(2)所有同名的有名公用块在可执行程序中它所出现的所有作用域单位中都必须是同样大小的，而空白公用块可以有不同的大小。例中BLA 在S1和S2都为20个数值存储单元，而空白公用块在S1中大小为3而在S2中大小为8，都是符合标准的。

(3)在有名公用块中的数据对象可以在块数据程序单元中用DATA 语句来初始化定义，但空白公用块中数据对象禁止初始化定义。

5. 公用与等价的限制

(1)EQUIVALENCE 语句不得导致两个不同的公用块的存储序列相结合。

(2)等价结合不得导致公用块存储序列在COMMON 语句中对该公用块说明的第一个对象的第一个存储单元前增加存储单元来扩展。例如：

```
COMMON /X/ A, /Y/ B(3)
REAL  C(2)
EQUIVALENCE (A, C(1)), (B(2), C(2))
```

是不允许的，因为等价导致有名公用块X和Y的存储序列的结合。

```
COMMON /X/A
REAL  C(2)
EQUIVALENCE (A,C(2))
```

也是不允许的，因为有名公用块X的第一个存储单元为A，这样等价后将导致在A的前面扩展存储单元以存放C(1)，这是不符合标准的。

小　结

本章介绍了标量对象，数组对象的定义和使用。特别对数组详细介绍了显形数组、自动数组、假取形数组、迟形数组、假取大小数

组。介绍了数组的一种子对象数组元素，它是标量；另一种子对象数组片段是数组。此外还介绍了数组元素在数组中的排列次序以及构成数组片段的两种形式——下标三元组和向量下标。这对以后特别是数组的运算是重要的基础，也为充分利用具有向量部件的计算机提供了说明数据对象的基础。

本章还介绍了动态结合和存储结合的部分概念，以及与此有关的语句。

本章的内容有相当多的部分是Fortran 90所独有的，希望能得到读者的充分注意。

习　　题

1．试述标量与数组、常量与变量这些概念的差别和联系。

2．什么叫显形数组、自动数组、假取形数组、迟形数组、假取大小数组？试举例说明。

3．动态结合有什么用处？

4．为什么Fortran 90不定义非默认类型的存储结合？

5．为什么COMMON语句，EQUIVALENCE语句已成为将被淘汰的成分？

第六章　表达式和赋值语句(一)

数值运算、字符运算、关系运算、逻辑运算、标量运算和数组运算都是科学计算中最基本的运算，运算的结果或用于条件判断，或作赋值以保存运算得到的结果。本章将主要叙述各种运算的规则、运算的优先顺序、计值规则和赋值规则等。

第一节　内在运算和相应的标量表达式

在第三章介绍数据类型时曾指出，对于算术型的标量可以进行算术运算和关系运算，对于字符型的标量可以进行字符运算和关系运算，对逻辑型的标量可以进行逻辑运算。在第一章中曾对这些运算的运算符作了较详细的介绍。本节仅就 Fortran 77或 Fortran 90的特点再补充说明一些内容。要特别指出 Fortran 90允许用户自己定义所需的运算，这将在第十五章介绍，本节仅介绍系统内在定义的各种运算。

一、数值运算和算术表达式

在 Fortran 90中有七个数值运算符，称为内在运算符。它们的符号及名称如下：

一元运算符：＋ ——等同运算，－ ——反号运算；

二元运算符：＋ ——加法运算，－ ——减法运算，* ——乘法运算，/ ——除法运算，* * ——乘幂运算。这些运算和运算符的意义前面已经介绍过了。

在此要指出，Fortran 77或 Fortran 90并不要求二元运算的两个操作数都具有相同的类型，Fortran 90也不要求相同的类型参数。这一点已在第三章中作了叙述，为了清楚起见再列表于表6

—1中：

表6—1

A (+ − * / * *) B		B 的 类 型		
		整 型	实 型	复 型
A的类型	整 型	整 型	实 型	复 型
	实 型	实 型	实 型	复 型
	复 型	复 型	复 型	复 型

对于一元运算来讲，只需把 A 看作是与 B 同类型的0值即可。允许类型混合的主要目的是为了便于用户书写程序，例如第一章中曾指出计算公式4a 在程序中写为4 * A 是错误的，因为4是整型而 A 是实型，现在 Fortran 90这是可以的，它与4.0 * A 起一样的作用。当然从实际的运行效率来讲可能是不同的，它与编译技术有关，有的可能要增加类型转换的操作，所以在可能的情况下应养成书写类型一致的良好的习惯。

在 Fortran Ⅳ 中还有一条规则，那就是相同优先级的算术运算是左结合的，即从左到右计算。例如，A * * B * * C＝(A * * B) * * C。而现在在 Fortran 90 中乘幂运算规定是右结合的，即从右到左计算。例如，A * * B * * C＝A * * (B * * C)。

一个算术表达式的类型和类型参数是与计值规则一起确定的。表达式计值是先从内层括号开始，一层一层往外计算，在同一层括号内先乘幂运算，后乘除运算，再加减运算。在同一类运算中从左到右进行运算，但乘幂运算是从右到左进行的。每个运算结果的类型在上表中已给出，至于种别类型参数已在第三章中介绍过了。当表达式值计算出来时，该结果的类型和种别类型参数就是表达式的类型和种别类型参数。

二、字符运算和字符表达式

内在字符运算符只有一种，那就是//——并置运算符。运算的两个操作数都必须是字符型的，其结果也是字符型的，其值是将两

个字符串连接起来。例如,'ABC'//'DE'的结果是'ABCDE',运算结果的长度类型参数值是两个操作数的长度类型参数值之和。在Fortran 90中还规定两个操作数都应有相同的种别类型参数。

三、关系运算和关系表达式

在Fortran 90中有六种内在关系运算符，每种运算符都有两种表示形式:大于运算符是.GT.或>,大于等于(不小于)运算符是.GE.或>=,小于运算符是.LT.或<,小于等于(不大于)运算算符是.LE.或<=，等于运算符是.EQ.或==，不等于运算符是.NE.或/=。

这些运算都是二元运算,它们的两个操作数或都是算术表达式,或都是字符表达式,这在第三章中已介绍过。要重复提醒注意的几点是:

(1)Fortran Ⅳ要求两个算术表达式类型一致,而现在没有这个要求,假定关系运算符用.RE.来表示,则A.RE.B可以认为就是(A-B).RE.0。

(2)复型量没有大小的概念,所以只能用等于或不等于这两种运算符。

(3)字符型最好也只用等于和不等于两种运算。对于其它关系运算就不一定能保证可移植性,因为大于、小于等的比较取决于字符集的排序,语言规定数字可能先于字母,也可以相反,所以数字1同字母A是否使小于关系成立,是依赖于处理系统的。对于其它字符集来讲更是如此。特别要指出两个字符型表达式进行关系运算时,其种别类型参数必须相同。

(4)关系运算的结果是逻辑值.TRUE.或.FALSE.。当关系式成立时为.TRUE.,否则为.FALSE.。Fortran 90还特别指出这是默认逻辑型值。

四、逻辑运算和逻辑表达式

Fortran 90 中有5种内在逻辑运算符:

一元运算符：.NOT. —— 非运算。

二元运算符：.AND. —— 与运算，.OR. —— 或运算，.EQV. —— 等值运算，.NEQV. —— 不等值运算。

所有操作数均需是逻辑表达式，运算结果见表6－2。

表6－2

A	B	.NOT.B	A.AND.B	A.OR.B	A.EQV.B	A.NEQV.B
T	T	F	T	T	T	F
T	F	T	F	T	F	T
F	T	F	F	T	F	T
F	F	T	F	F	T	F

注：T 表示.TRUE.，F 表示.FALSE.。

第二节　常量标量表达式

常量标量表达式是标量表达式的一种特殊情况，参加运算的操作数或为常量，或为常量子对象，且其中每个下标，子串的始点和终点均是常量表达式，或为每个变元都是结果为标量的常量表达式的内在函数引用，而且常量表达式中的运算符均为内在运算符。例如：

```
4
7.3+4.6*3.576
'AB'
'AB'//'XYZ' (2:2)
3.5**2.0
.TRUE.
(4.5,5.7)+(3.6,7.93)
ABS (9.0)
```

都是合法的常量标量表达式的例子。

第三节　表达式的一般形式

本节将介绍表达式的一般形式。在前面介绍了标量表达式及其特殊情况常量标量表达式。在后面还要介绍数组表达式，不管是标量表达式还是数组表达式其主要区别在于操作数，若操作数全为标量，则被称为标量表达式，否则称为数组表达式。表达式是对数据的引用并进行计算，其结果是一个标量或一个数组。

表达式可由操作数、运算符和括号所组成。操作数的最简单形式是常量和变量。运算符分为内在的或定义的两种，内在运算符在前一节中已介绍，定义的运算符的定义方法将在第十五章中给予介绍。表达式本身又可作为操作数而构成更复杂的表达式。

表达式的定义可以分为几类：初等量、一级表达式、二级表达式、三级表达式、四级表达式和五级表达式。在后面将会清楚地看到这种分类是与运算符的优先级相联系的。

一、初　等　量

初等量是表达式的最简单的形式。初等量可以是常量、常量的子对象、变量、数组构造符、结构构造符（在第十三章中介绍）、函数引用或用括号括着的表达式。例如：

```
3.0
'ABCD' (I:I)
A
(/3.1, 3.2, 3.5/)
F(X,Y)
(S+G * W)
```

都是合法的初等量。

二、一级表达式

一元运算符后跟一个初等量就构成一个一级表达式。

三、二级表达式

二级表达式定义为：

〔〔二级表达式〕加运算符〕加操作数

加运算符为运算符＋或－

加操作数定义为：

〔加操作数 乘运算符〕乘操作数

乘运算符为运算符＊或/

乘操作数定义为：

一级表达式〔幂运算符乘操作数〕

幂运算符为＊＊

例如：

A

C＊＊F

H＊Q

＋3

J－I

－A＋H＊Q－C＊＊F

都是正确的二级表达式。

四、三级表达式

三级表达式是由二级表达式任选地包含并置运算符所组成，定义为：

〔三级表达式//〕二级表达式

例如：

A

X//Y

X//Y//'I AM A BOY'

都是正确的三级表达式。

五、四级表达式

四级表达式是由三级表达式任选地包含关系运算符所组成，

定义为：

〔三级表达式 关系运算符〕三级表达式

六种关系运算符已在前一节列举。例如：

A

M.EQ.N

I<J

X.NE.Y

(X+Y).EQ.Z

都是正确的四级表达式。

六、五级表达式

五级表达式定义为：

〔五级表达式 等值运算符〕等值操作数

等值运算符为.EQV.和.NEQV.

等值操作数定义为：

〔等值操作数.OR.〕或操作数

或操作数定义为：

〔或操作数.AND.〕与操作数

与操作数定义为：

〔.NOT.〕四级表达式

例如：

A

.NOT. X

E.AND.F

M.OR.N

S.EQV.T

W.NEQV.R

E.AND.F .EQV. .NOT.X

都是正确的五级表达式。

七、表 达 式

表达式定义为在五级表达式前任选地出现定义的二元运算

符。例如：

A

X//Y .EQ. Z(1:5)

等均是正确的表达式。

八、运算的优先顺序

前面的定义看起来很复杂，其实质是定义了运算符的优先顺序。这种优先顺序可见表6—3。

表6—3

运算分类	运　　算　　符	优　先　级
扩　充	定义的一元运算符	最　高
数　值	* *	
数　值	* 或/	
数　值	一元+或—	
数　值	二元+或—	
字　符	//	
关　系	.EQ.、.NE.、.LT.、.LE.、.GT.、.GE.、= =、/=、<、<=、>、>=	⋮
逻　辑	.NOT.	
逻　辑	.AND.	
逻　辑	.OR.	
逻　辑	.EQV. 或. NEQV.	
扩　充	定义的二元运算符	最低

掌握运算符的优先顺序很重要，因为在是否使用括号，括号的完整性检查时需要用到有关的性质。例如计算负的 X 平方则写成—X * * 2即可，因为 * * 比一元运算符—的优先级高，所以上述等价于—(X * * 2)。如果计算$(-X)^2$时，必须写成(—X) * * 2，而写成—X * * 2将得到错误的结果。

此外从表达式的定义中，我们还知道了同优先级的运算是从

左到右计算或从右到左计算，例如：A＊＊B＊＊C是什么意思，从定义中可知是A＊＊(B＊＊C)而不是(A＊＊B)＊＊C。所以计算a^{bc}时可表达为A＊＊B＊＊C，而计算$(a^b)^c$时要用(A＊＊B)＊＊C。A/B＊C表示计算ac/b，而A/(B＊C)表示计算a/bc。注意：只有幂运算是从右到左计算的，而其它运算均从左到右计算。

九、计 值 规 则

知道了运算符的优先顺序就不难确定其计值规则，可以大致归结为：

(1)先计算内层括号，再逐层往外计算。

(2)在同一层括号内，按运算符的优先顺序确定计算的先后顺序，优先级高者先计算。

(3)在计算连续的相同优先级的运算符时，除了幂运算是从右到左的顺序计算外，其余运算均从左到右的顺序计算。

(4)在作每个运算时，对运算符后一个或运算符前后的两个操作数进行运算，并按第一节介绍的办法确定运算后结果的类型，类型参数和形状。关于形状在标量表达式中总为零大小的一维数组。

这样，只要遵循这些计值规则，我们就可正确地逐步求得表达式的值、类型、类型参数和形状。

第四节　初始化表达式和说明表达式

在前面的一些语句中经常提到初始化表达式和说明表达式，但由于当时还未完整地介绍表达式，所以当时简单地将其理解为常量、标量变量或内在的二元运算，这实质上是表达式的简单情况。现分别予以介绍初始化表达式和说明表达式。

一、初始化表达式

初始化表达式是一个常量表达式，其中幂运算只允许有整数幂，而每个初等量允许是下列之一：

(1)常量或常量的子对象，其中每个下标、片段下标、子串的始点和终点都是初始化表达式。

(2)数组构造符，涉及到表达式的也只允许初始化表达式或可能有的隐 DO 变量。

(3)结构构造符，其中每个成分是初始化表达式。

(4)元素内在函数的整型或字符型引用时，其中每个变元是整型或字符型的初始化表达式。

(5)变换内在函数只引用 REPEAT、RESHAPE、SELECTED_INT_KIND、SELECTED_REAL_KIND、TRANSFER 或 TRIM，而且变元是初始化表达式。

(6)引用数组查询函数(除 ALLOCATED 外)，位查询函数 BIT_SIZE，字符查询函数 LEN，种别查询函数 KIND，数值查询函数(详见第十一章)时，其中每个变元是初始化表达式。

(7)括在括号中的初始化表达式。

初始化表达式的例子是：

```
5
'AB'//'CDE'//'FG'
KIND('A')
```

而下面的例子不是初始化表达式：

```
ABS(3.0)
4.0**2.0
```

二、说明表达式

说明表达式是一种特殊的受限表达式，在此先介绍受限表达式。

受限表达式的每个运算都是内在的，每个初等量是下列之一：

(1)常量或常量的子对象。

(2)虚元变量或其子对象，但该虚元不得具有 OPTIONAL 属性或 INTENT(OUT)属性。

(3)公用块中的变量或其子对象。

(4)通过使用结合或宿主结合可访问的变量或其子对象。

(5)数组构造符,涉及到表达式的也只允许受限表达式或可能有的隐 DO 变量。

(6)结构构造符,其中每个成分是受限表达式。

(7)元素内在函数的整型或字符型引用时,其中每个变元是整型或字符型的受限表达式。

(8)变换内在函数只引用 REPEAT、RESHAPE、SELECTED_INT_KIND、SELECTED_REAL_KIND、TRANSFER 或 TRIM,而且变元是受限表达式。

(9)引用数组查询函数(除 ALLOCATED 外),位查询函数 BIT_SIZE,字符查询函数 LEN,种别查询函数 KIND,数值查询函数(详见第十一章)时,其中每个变元是受限表达式。

(10)括在括号中的受限表达式。

而说明表达式是一个整型标量的受限表达式。

当 M 是整型虚元,C 是字符型虚元时,M+LEN(C)就是说明表达式的例子。

第五节　标量赋值语句和赋值规则

执行标量赋值语句,将导致一个标量成为定义的或再定义的。其一般形式为:

V=e

其中 V 为一个标量,e 为一个标量表达式,=为赋值号,这里要特别提醒读者=并不是等号。

注意,标量与标量表达式的类型,两者或同为数值型、或同为字符型、或同为逻辑型。

(1)当 V 是整型,e 是整型、实型或复型时,先按表达式的计值规则得到 e 的值取整后再赋给 V。当种别类型参数也不相同时,按 V 的种别类型参数为准进行转换。所以其结果就象执行了类属内

在函数。

INT (e,KIND=KIND(V))

后才赋给 V,内在函数的定义详见第十一章。

(2)当 V 是实型,e 是整型、实型、复型时,先按表达式的计值规则得到 e 的值后再实型化后再赋给 V。当种别类型参数不同时,按 V 的种别类型参数为准进行转换。所以其结果就象执行了类属内在函数。

REAL (e,KIND=KIND(V))

后才赋给 V。

(3)当 V 是复型,e 是整型、实型、复型时,先按表达式的计值规则得到 e 的值后再复型化后再赋给 V。当种别类型参数不同时,按 V 的种别类型参数为准进行转换。所以其结果就象执行了类属内在函数。

CMPLX (e,KIND=KIND(V))

后才赋给 V。

(4)当 V 是逻辑型,e 是逻辑型时,先按表达式的计值规则得到 e 的值,当种别类型参数不同时,按 V 的种别类型参数为准进行转换,并将最终结果赋给 V。

(5)当 V 为字符型,e 为字符型时,注意此时要求其种别类型参数必须相同。先按表达式的计值规则计算出 e 的值,当 e 的长度大于 V 的长度时,按 V 的长度从右面截断 e 的值,然后赋给 V。当 e 的长度小于 V 的长度时,在 e 的值的右面用空格加以扩充,直到与 V 长度一致为止,再赋给 V。

注意对于非默认字符型,填充字符是依赖于处理系统的。例如:

V 的类型	赋值语句	V 的值
整　型	I=3+7	10
整　型	J=3.2+7.7	10
整　型	K=(3.5,4.5)+(7.4,5.0)	10
实　型	A=3+7	10.0
实　型	B=3.2+7.7	10.9

实　型	C=(3.5,4.5)+(7.4,5.0)	10.9
复　型	Z=3+7	(10.0,0.0)
复　型	Y=3.2+7.7	(10.9,0.0)
复　型	X=(3.5,4.5)+(7.4,5.0)	(10.9,9.5)
逻辑型	L=.TRUE..OR.L1	.TRUE.
字符型	C4='AB'//'CD'//'EF'	'ABCD'(C4长度为4)
字符型	C6='AB'//'CD'//'EF'	'ABCDEF'(C6长度为6)
字符型	C8='AB'//'CD'//'EF'	'ABCDEF　'(C8长度为8)

第六节　数组表达式和数组赋值语句

当一元运算的操作数为数组，二元运算的两个操作数均为数组，或一个为标量一个为数组时，这样构成的表达式称为数组表达式，也就是说在第三节定义的表达式中变量有一个是数组时，该表达式就是一个数组表达式。例如：

```
A(1:5)+B(2:6)
3.4*C(1:15)+D(5:19)
E(1:99:2)+F(2:100:2,10)/G(3:101:2,5,5)
H(5:20:5)//P(1:4)
```

其中 A,B,C,D,E,F,G 均是数值型数组，H,P 是字符型数组。以上各例都是正确的数组表达式，在这里要强调几点：

(1)当两个操作数都是数组时，要求它们的形状是相符合的，即相同的。上例中 A 和 B 的形状为(5)，C 和 D 的形状为(15)，E，F 和 G 的形状均为(50)，H 和 P 的形状均为(4)。

(2)当二元运算的两个操作数中一为数组、一为标量时，则将此标量认为是形状同另一操作数数组的一个数组，其每个元素均为该标量的值。

(3)数组运算是将对应元素进行由运算符指明的运算，其结果仍为数组，其元素的值就是操作数相应元素运算的结果。例如，A*B 并不意味着数学上的矩阵相乘，这点在使用时也要特别小心。

数组表达式的计值规则、类型和类型参数的确定与标量表达

式对应的规则相同。其形状就是操作数的形状，所以要求各操作数的形状是相符合的。

数组赋值语句的形式是：

V=e

其中 V 为数组，e 为数组表达式，而 = 为赋值号。数组赋值语句可以认为是同类型标量赋值语句的扩展，所以数组表达式的计值及赋值规则与标量表达式相应部分是一样的。最后要强调的是：V 和 e 的形状必须是相符合的。

V 不允许是多对一数组。

当 V 是大小为零的数组，或是长度为0的字符型时，则没有值赋给 V。

当 e 是标量时，则把 e 处理成一个与 V 同形状的数组，数组的每个元素均等于 e 的标量值。

例如：

```
A(1:10)=3.0+7.5
```

是正确的数组赋值语句，A(1)到 A(10)每个元素的值均被赋为10.5。又如：

```
AZ(1:50)=E(1:99:2)+F(2:100:2,10)/G(3:101:2,5,5)
Q(10:40:10)=H(5:20:5)//P(1:4)
W=Y+Z
```

其中 W,Y,Z 都是数组，这些也都是正确的数组赋值语句。

小　　结

本章较为详细地介绍了各种表达式和赋值语句，特别强调了运算符间的优先级顺序，以及同级运算符间的计算顺序等，这对计算值是很重要的。此外还介绍了运算结果的类型、类型参数与操作数的类型、类型参数的关系。介绍了计值规则和赋值规则。这些构成了有关运算的最基本内容，也可以说是 Fortran 90的重要内容。

习　　题

1. 试述各种内在运算符及其优先级。

2. 表达式的计值规则是什么?

3. 赋值语句的赋值规则是什么?

4. 试对第二章的例题使用数组表达式及数组赋值语句进行改编。

第七章　控 制 语 句

为了使读者对 Fortran 90有个全面的认识，本章将介绍 Fortran 90中的语句标号、GOTO 语句、计算 GOTO 语句、赋值 GOTO 语句、算术 IF 语句、CONTINUE 语句、STOP 语句和 PAUSE 语句这些常见的控制语句，其中很多内容是与前面对 Fortran 77的介绍是一致的。重点将放在对 Fortran 90来说更重要的块结构、IF 构造、CASE 构造和 DO 构造等的介绍上。

第一节　分 支 语 句

分支用于改变通常的依次执行的顺序。分支导致控制从作用域中一个语句转到该域中一个带标号的分支目标语句。分支目标语句是动作语句，IF－THEN 语句，END－IF 语句，SELECT－CASE 语句，DO 语句，END－DO 语句，DO 终结活动语句，DO 终结共享语句，或 WHERE 构造语句。

一、语 句 标 号

语句标号提供了引用单个语句的手段。语句标号可由至多五个数字组成，且其中至少有一个是非0数字。如果一个语句有标号，则该语句必须包含非空格字符。标号中的前导0不起区分作用。

二、GOTO 语句

GOTO 语句的形式是：

GOTO 标号

其中标号必须是出现在与 GOTO 语句同一作用域单位的某分支目标语句的语句标号。

GOTO 语句的执行引起控制转移，接着执行的是该标号所标识的分支目标语句。

三、计算 GOTO 语句

计算 GOTO 语句的形式是：

GOTO（标号表）〔,〕sie

其中每个标号都必须是出现在与该语句同一作用域单位的分支目标语句的语句标号，相同的标号可以在表中多次出现。sie 是一整标量表达式。

计算 GOTO 语句的执行，首先计算整标量表达式 sie 的值，设值为 i，又设标号表中出现标号的个数为 n，若 $1 \leqslant i \leqslant n$，则控制转移到标号表中第 i 个标号所标识的分支目标语句；若 $i < 1$ 或 $i > n$，则继续顺序执行，就象执行一个 CONTINUE 语句一样。

四、ASSIGN 语句和赋值 GOTO 语句

ASSIGN 语句的形式是：

ASSIGN 标号 TO V

其中标号必须是出现在与该语句同一作用域单位的分支目标语句的语句标号，V 是默认整型标量变量名。

该语句的执行，导致变量 V 被语句中的标号值定义。注意它绝不是用整值定义，这两者是有重要区别的。这个 V 所得到的定义值只能用于赋值 GOTO 语句或 I/O 语句中的格式说明符。例如：

```
ASSIGN 100 TO I
```

执行该语句后，整变量 I 中有标号值100。

赋值 GOTO 语句的形式是：

GOTO V〔〔,〕（标号表）〕

其中表中的每个标号都必须是出现在与该语句同一作用域单位中的分支目标语句的语句标号，相同的标号可以在表中多次出现，V 必须是默认整型标量变量名。

该语句执行时，V 必须已被 ASSIGN 语句赋以标号值，它的执行将引起控制转移到 V 的语句标号所标识的分支目标语句继续执行。如果语句中出现标号表，则 V 中的标号值必须是标号表中的语句标号之一。

五、算术 IF 语句

算术 IF 语句的形式是：

IF(sme) 标号1 ，标号2 ，标号3

其中每个标号都必须是出现在与该语句同一作用域单位中的分支目标语句的语句标号，同一标号可以出现多次，sme 是标量算术表达式，该表达式不能是复型的。

算术 IF 语句的执行，首先计算标量算术表达式的值，然后按值是小于、等于或大于零，将控制分别转移到标号1 、标号2 或标号3 所标识的分支目标语句继续执行。

第二节　包含块的控制构造

在 Fortran 90中明确提出了块的控制构造，以及把块作为单元来看待的可执行构造序列的概念。它可用于 IF、CASE 和 DO 的构造中，当然 DO 还可有非块的构造，本节将分别加以介绍。

一、块及其规则

块是作为单元看待的一个可执行构造的序列，这个序列可以不包含任何可执行语句，这样的块的执行对控制序列没有影响。

块可以用于 IF、CASE 和 DO 构造中，对于这三种构造都可以命名，如果一个构造是有名的，那么该名字必须是该构造的第一个语句的第一个词法记号及该构造的最末一个词法记号，在固定格式的源程序中，这个名字必须在第6 列之后。

语句是它的最内部构造，例如：

```
IF (A>0.0) THEN
    B=SQRT (A)
    C=LOG (A)
END IF
```

中第二、三两个语句构成一个块，这个块是属于 IF 构造的最内部构造。

对于块有这样一些规则必须遵循：

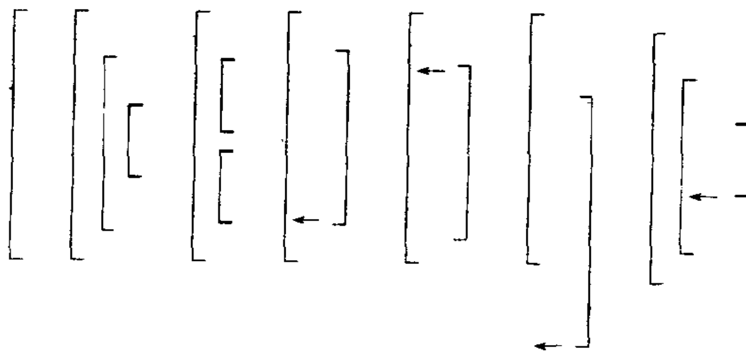
(1)如果块包含一个可执行构造，那么它必须完整地被包含在该块中。

(2)块中可以包含子程序和函数的引用，此外禁止从块的外部控制转移到块的内部。在块中进行控制转移是可以的，从块内控制转移到块外是可以的。

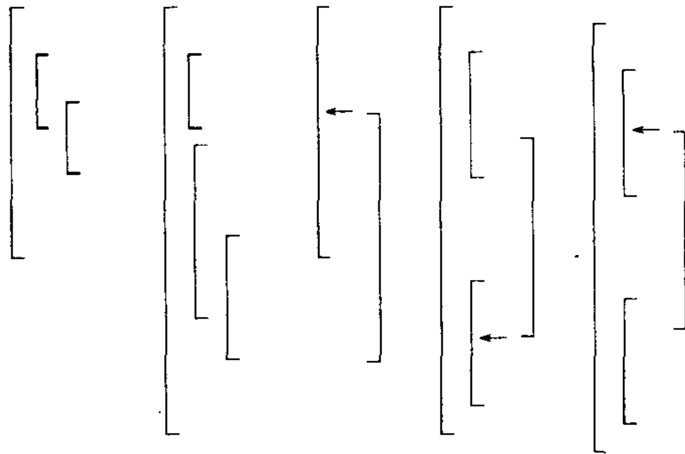
(3)块的执行是从块中的第一个可执行构造的执行开始的，直到在序列中最后一个可执行构造的执行完为止，除非有块内的控制转移到块外。在终止时发生的动作取决于特定的构造和在构造中的块。

在这里我们要特别提醒，从块的定义中可以看到块是可以嵌套的，这些规则不仅对于外层的块是适用的，而且对于每一层的块，特别是最内层的块也是适用的。如果我们用[来表示一个块，用"↓"来表示控制转移的话，那么上述规则可以图示如下：

允许的情况的例子是：



不允许的情况的例子是：



这些规则在以后的各种构造中将会不断地用到。

二、IF 构造

IF 构造执行时至多选择其所组成的块之一，而 IF 语句仅控制单个语句的执行。

IF 构造的形式是：

〔IF 构造名:〕IF (sle) THEN
块1
〔ELSE IF (sle) THEN〔IF 构造名〕
块2〕…
〔ELSE〔IF 构造名〕
块3〕
END IF〔IF 构造名〕

其中sle是逻辑型的标量表达式，IF构造名为一个名字，块1、块2、块3均是块。方括号中的内容表示是可选的，以下同此理解。

注意：如果 IF 构造的 IF—THEN 语句前用 IF 构造名来标识，则相应的 END IF 语句之后必须指明相同的 IF 构造名。如果 IF 构造的 IF—THEN 语句没用 IF 构造名来标识，则相应的 END IF 语句不得指明 IF 构造名。如果 ELSE IF 语句或 ELSE 语句用 IF 构造名来标识，则相应的 IF—THEN 语句也必须用指明相同的 IF 构造名。

IF 构造的执行是：

(1)先计算 IF－THEN 语句中逻辑型标量表达式 sle 的值，若能判明其值为.TRUE.，则执行块1，然后结束该构造的执行。

块1 或者以后续的 ELSE IF 语句或 ELSE 语句为界，或者以该构造的 END IF 语句为界，其最后一个语句的下一语句必须是上述三者之一。

(2)若判明其值为.FALSE.，而且下一个 ELSE IF 语句也出现，则继续求 ELSE IF 语句的逻辑型标量表达式的值，若能判明其值为.TRUE.，则执行紧连着的块2，然后结束该构造的执行。

块2或者以后续的下一个 ELSE IF 语句或 ELSE 语句为界，或者以该构造的 END IF 语句为界，其最后一个语句的下一语句必须是上述三者之一。

(3)若判明其值为.FALSE.，而且下一个 ELSE IF 语句也出现，则重复(2)的过程。

(4)若 IF－THEN 语句和后续的 ELSE IF 语句中的逻辑型标量表达式的值均为.FALSE.，且 ELSE 语句出现，则执行块3，然后结束该构造的执行。若 ELSE 语句不出现，则结束该构造的执行。

块3 以该构造的 END IF 语句为界，即其最后一个语句(或无任何语句)的下一个语句必须是该构造的 END IF 语句。

一个 IF 构造的执行可以简述为，按该 IF 构造中出现的次序，依次计算逻辑型标量表达式的值。若有一个值为.TRUE.，则执行紧接着的块，该块以一个语句为最后语句，其下一个语句必须是 ELSE IF 语句、ELSE 语句或该构造的 END IF 语句三者之一。然后结束该构造的执行。否则执行后续的 ELSE 语句后的块，若没有则结束该构造的执行。

下面我们来看几个例子：

```
IF (CHAR .EQ. 'RESET') THEN
    I=0; J=0; K=0
END IF
```

这是一个最简单的例子，没有 ELSE IF 语句和 ELSE 语句，也不使用 IF 构造名，块1由三个赋值语句所组成。

```
PROOF_DONE: IF (PROP) THEN
                  WRITE (3,'("QED")')
                  STOP
                ELSE
                  PROP=NEXT PROP
                END IF PROOF_DONE
```

这个例子使用了 IF 构造名 PROOF_DONE，块1由两个语句所组成，块3是一个语句，没有 ELSE IF 语句。

```
FIRST: IF (A>0) THEN
             B=C/A
             SECOND: IF (B>0) THEN
                        D=1.0
             END IF SECOND
         ELSE IF (C>0) THEN
             B=A/C
             D=-1.0
         ELSE
             B=ABS (MAX(A,C))
             D=0
         END IF FIRST
```

这个例子略为复杂一些，块1中包含了一个赋值语句和另一个 IF 构造，块2和块3均有两个赋值语句构成。这个程序实际上是用于计算下列公式：

$$b=\begin{cases} c/a & \text{当 } a>0 \\ a/c & \text{当 } a\leqslant 0,\ c>0 \\ |\max(a,c)| & \text{当 } a\leqslant 0,\ c\leqslant 0 \end{cases}$$

$$d=\begin{cases} 1.0 & \text{当 } a>0,\ b>0 \\ \text{不变} & \text{当 } a>0,\ b\leqslant 0 \\ -1.0 & \text{当 } a\leqslant 0,\ c>0 \\ 0 & \text{当 } a\leqslant 0,\ c\leqslant 0 \end{cases}$$

在此我们要特别指出几点：

(1)由于块中可以再包含 IF 构造，所以一定要求 IF－THEN 语句和 END IF 语句之间的配对，这就象表达式中要求括号配对一样。在这种情况下，对于 IF 构造使用 IF 构造名来标识不失为一种好方法。书写时注意缩格对齐也是一个好习惯。同时注意我们在描述一个 IF 构造时，强调了该构造的 END IF 语句，也就是为了强调这种意思。

(2)ELSE 语句是在所有的逻辑型标量表达式的值为.FALSE.时的公用部分，对于 ELSE IF 语句没有单独的 ELSE 语句与其配合，所以在书写程序时，要使用一些程序技巧。例如在必须这样做时可以写为：

```
ELSE IF (.TRUE.) THEN
          NEXT: IF (sle) THEN
                    …
                ELSE
                    …
                END IF NEXT
```

关于这点我们再举一例子：

计算

$$y=\begin{cases} x^3 & \text{当 } x>1 \\ x^2+z & \text{当 } -1\leqslant x\leqslant 1,\ z\geqslant 0 \\ -x^2-z & \text{当 } -1\leqslant x\leqslant 1,\ z<0 \\ -x^3-z^2 & \text{当 } x\leqslant -1,\ z\geqslant 0 \\ -x^3+z^3 & \text{当 } x\leqslant -1,\ z>0 \end{cases}$$

写成程序

```
IF (X>1) THEN
    Y=X * * 3
ELSE IF (X<=1. AND. X>=-1. AND. Z>=0) THEN
    Y=X * * 2+Z
  ELSE
    Y=-X * * 2-Z
ELSE IF (Z>=0) THEN
```

```
      Y=-X**3-Z**2
   ELSE
      Y=-X**3+Z**3
   END IF
```

就是错误的。可将第5个语句(ELSE 语句)写成：

```
   ELSE IF (X<=1.AND.X>=-1.AND.Z<0) THEN
```

或将第3到第6个语句换成：

```
   ELSE IF (X<=1.AND.X>=-1) THEN
     NEXT: IF (Z>=0) THEN
              Y=X**2+Z
           ELSE
              Y=-X**2-Z
           END IF NEXT
```

(3)当依次计算逻辑型标量表达式时,当有一个值为.TRUE.时,后面可能有的 ELSE IF 语句中标量表达式就不需再计算，这说明并不是每个都是必须计算的。

(4)ELSE IF 语句和 ELSE 语句不可作为分支目标语句。从 IF 构造中转移到 END IF 语句是允许的。而且从构造外转移来 END IF 语句也是允许的,但这是一种将被淘汰的用法,因为这不是一种好的结构化程序设计的习惯,建议不用。END IF 语句的执行对执行序列没有影响。

最后还要介绍一个 IF 语句,这就是以往 Fortran 语言称作为逻辑 IF 语句的。逻辑 IF 语句的形式是：

```
   IF (sle)  动作语句
```

其中 sle 是逻辑型标量表达式,动作语句不可以是另一 IF 语句,END PROGRAM 语句,END FUNCTION 语句或 END SUBROUTINE 语句。

该语句的执行引起计算逻辑型标量表达式的值,当其为.TRUE.时执行动作语句。当其为.FALSE.时，动作语句不执行而被跳过,就象执行了一个 CONTINUE 语句一样。例如：

```
   IF (IHAR .EQ. 'RESET')  I=0
```

```
J=3
……
```

这里要注意，无论动作语句是否被执行，IF 语句执行后总是执行下一语句。所以上例中不管 IHAR 的结果是什么，J=3这个语句总是要执行的，这是它与 IF—THEN 语句加上 ELSE 语句的 IF 构造的重要区别。而且在此动作语句仅允许一个语句，当要执行多个语句时最好用 IF 构造。例如：

```
IF (B* *2-4*A*C>0) X=-B+SQRT (B*B-4*A*C)
X=-B+SQRT (4*A*C-B*B)
  ……
```

这达不到控制负数不进行开方的目的，应改为：

```
    IF (B* *2-4*A*C>0) GOTO 20
    X=-B+SQRT (4*A*C-B*B)
    GOTO 30
20  X=-B+SQRT (B*B-4*A*C)
30  ……
```

显然这里用了较多的不必要的 GOTO 语句，而导致程序结构化不好，推荐写成如下程序：

```
IF (B* *2-4*A*C>0) THEN
    X=-B+SQRT(B*B-4*A*C)
ELSE
    X=-B+SQRT(4*A*C-B*B)
END IF
```

三、CASE 构造

CASE 构造在多个组成块中至多选择一个组成块执行。在某些多种条件的选择情况下使用 IF 构造显得既不直观，程序又冗长，然而采用 CASE 构造就要好得多。

CASE 构造的形式是：

```
〔CASE 构造名:〕 SELECT CASE (case 表达式)
            〔CASE case 选择符〔CASE 构造名〕
```

块〕…

END SELECT〔CASE 构造名〕

其中 case 表达式是整型、字符型或逻辑型的标量表达式，而 case 选择符是 case 值范围表或 DEFAULT，case 值范围有下列四种形式：

case 值

case 值：

：case 值

case 值：case 值

而每个 case 值是整型、字符型或逻辑型的标量初始化表达式。

注意：

(1)如果 CASE 构造的 SELECT CASE 语句用 CASE 构造名来标识，则相应的 END SELECT 语句也必须指明相同的 CASE 构造名。如果 CASE 构造的 SELECT CASE 语句没有用 CASE 构造名来标识，则相应的 END SELECT 语句也不得指明 CASE 构造名。如果 CASE 语句用 CASE 构造名来标识，则相应的 SELECT CASE 语句也必须用相同的 CASE 构造名来指明。

(2)对于一个给定的 CASE 构造，DEFAULT 选择符至多只能出现一个。

(3)对于一个给定的 CASE 构造，每个 case 值必须与 case 表达式的类型相同。对于字符型来说允许长度不同，但种别类型参数必须相同。

(4)若 case 表达式是逻辑型的，则在 case 值范围中禁止使用冒号。

(5)对于一个给定的 CASE 构造，case 值范围必须没有重叠，也就是说禁止一个 case 表达式的值落在多个 case 值范围之内的情况发生。

case 构造的执行是：先计算 case 表达式的值，其结果称为 case 索引；假定 case 索引值为 C，对 case 选择符中的 case 值范围表进行匹配，若下述之一成立，即称为匹配发生了。

（1）如果 case 值范围只包含单个值 V，没有冒号，对于逻辑型来说若表达式 C. EQV. V 为. TRUE. ，对于整型和字符型来说若表达式 C. EQ. V 为. TRUE. 则匹配发生。

（2）如果 case 值范围是形如 l：的，若表达式 l. LE. C 为. TRUE. 则匹配发生。

（3）如果 case 值范围是形如：h 的，若表达式 C. LE. h 为. TRUE. 则匹配发生。

（4）如果 case 值范围是形如 l：h 的，若表达式 l. LE. C . AND. C. LE. h 为. TRUE. 则匹配发生。

（5）如果没有任何 case 选择符相匹配而有 DEFAULT 选择符出现，则认为匹配发生。

当匹配发生时，就执行含有被匹配的选择符的 CASE 语句后的块，CASE 构造的执行随块执行的结束而结束。

如果没有任何 case 选择符相匹配，而且 DEFAULT 选择符也不出现，则认为没有匹配发生，此时 CASE 构造的执行也就结束了。

由此可见一个 CASE 构造至多只能使一个块被执行。此外我们要注意 CASE 语句不能作为分支目标语句。仅仅从 CASE 构造内控制转移到 END SELECT 语句是允许的。

CASE 构造的几个例子：

〔例1〕写出求整数的符号函数。

我们知道一般符号函数的定义是：

$$\mathrm{Sign}(x)=\begin{cases}1 & x>0\\ 0 & x=0\\ -1 & x<0\end{cases}$$

而现在我们要的是整数的符号函数，所以可以把定义改写为：

$$\mathrm{Sign}(i)=\begin{cases}1 & i\geqslant 1\\ 0 & i=0\\ -1 & i\leqslant -1\end{cases}$$

这样我们就可用 CASE 结构来描述如下：

```
INTEGER FUNCTION SIGN (I)
```

```
SELECT CASE (I)
  CASE (1:)
    SIGN=1
    CASE (0)
      SIGN=0
    CASE(:-1)
      SIGN=-1
    END SELECT
  END FUNCTION SIGN
```

〔例2〕检查一个语句中括号配对的结构。

```
CHARACTER (1320) :: LINE
     ...
LEVEL=0
DO I=1, 1320
  CHECK _ PARENS: SELECT CASE (LINE(I:I))
    CASE ('(')
        LEVEL=LEVEL+1
    CASE (')')
        LEVEL=LEVEL-1
        IF (LEVEL. LT. 0) THEN
            PRINT *, 'UNEXPECTED RIGHT PARENTHESIS'
            EXIT
        END IF
    CASE DEFAULT    ! 忽略其它字符
    END SELECT CHECK _ PARENS
END DO
IF (LEVEL .GT. 0) THEN
      PRINT *, 'MISSING RIGHT PARENTHESIS'
END IF
```

〔例3〕IF 构造与 CASE 构造有时可得到相同结果,CASE 构造的编制也不是唯一的。

IF 构造:

```
IF (SILLY .EQ. 1) THEN
    CALL THIS
ELSE
    CALL THAT
END IF
```

CASE 构造1：

```
SELECT CASE (SILLY .EQ. 1)
    CASE (.TRUE.)
        CALL THIS
    CASE (.FALSE.)
        CALL THAT
END SELECT
```

CASE 构造2：

```
SELECT CASE (SILLY)
    CASE DEFAULT
        CALL THAT
    CASE (1)
        CALL THIS
END SELECT
```

〔例4〕

```
SELECT CASE (N)
    CASE (1, 3:5, 8)
        CALL SUB
    CASE DEFAULT
        CALL OTHER
END SELECT
```

最后我们要指出几点：

(1)CASE 构造中包含的块同样可以是空的，也可以包含其他的构造或另一个 CASE 构造，特别当又包含构造时，要注意块的规则。

(2)CASE 构造中的算术标量表达式只能是整型的，所以在其他算术类型的情况下有的用变换的办法，有的用逻辑表达式的方

法来实现。有时也许还是用 IF 构造好，这要具体分析区别对待。但是象其它语言的枚举类型及控制转移一样，可能用 CASE 构造要好一些，这点请读者自行比较。

(3)DEFAULT 选择符与其它语言的 CASE 语句中的 OTHER 有类似的地方，但它不一定要出现在最后，可以出现在任何地方。

四、块 DO 构造

DO 构造用来说明可执行构造序列的重复执行，这种重复地执行可执行构造序列称为循环。EXIT 和 CYCLE 语句可用于改变循环的执行。

块 DO 构造的形式是：

〔DO 构造名:〕DO〔标号〕〔循环控制〕

块

DO 终结

其中有标号出现时称有标号 DO 语句，标号不出现时称无标号 DO 语句。

循环控制的形式是：

〔,〕SV=sne1, sne2〔, sne3〕

或

〔,〕WHILE (sle)

其中 SV 是一个标量变量，称为 DO 变量；sne 是数值型(即算术型)标量表达式；sle 是一个逻辑型标量表达式。

DO 终结的形式是：

END DO〔DO 构造名〕或

CONTINUE

注意：

(1)若块 DO 构造的 DO 语句有一个 DO 构造名标识，则相应的 DO 终结必须是指明相同的 DO 构造名的 END DO 语句。若块 DO 构造的 DO 语句没 DO 构造名标识，则相应的 DO 终结不得指

明 DO 构造名。

(2)若 DO 语句是无标号 DO 语句，则相应的 DO 终结必须是 END DO 语句。

(3)若 DO 语句是有标号 DO 语句，则相应的 DO 终结必须用与 DO 语句中标号相同的标号来标识。

(4)DO 变量必须是一个整型标量变量，虽然 Fortran 90标准允许是默认实型和双精度实型的变量，但是这将是淘汰的内容，请尽可能用整型而不用其它类型。

(5)循环控制中 sne 必须是整型标量表达式，虽然 Fortran 90 标准允许是默认实型和双精度实型的标量表达式，但这也将是淘汰的内容，请尽可能用整型而不用其它类型。

块 DO 构造的范围必须满足块规则，严禁从块的外部控制转移到内部，只允许从 DO 构造范围中分支到该构造的 DO 终结。

DO 构造在最初是不活跃的，当 DO 语句执行时才变为活跃的，直到构造被终止时才又变为不活跃的。当 DO 构造从活跃变为不活跃时，若有 DO 变量，则保留其最终的值。

DO 构造的执行有三个阶段。

1. 循环初始化

DO 语句执行时，DO 构造变成活跃的，若其循环控制是

```
[,] SV = sne1, sne2 [,sne3]
```

时，则：

(1) 求 sne1，sne2，sne3的值，必要时按 SV 的类型和种别类型参数进行转换，得到相应的初始参数 m1、终止参数 m2和增量 m3，若 sne3省略，则 m3为默认整型值1。注意 m3值不得为零。

(2) 根据 m1的值定义 DO 变量的值。

(3) 设置重复计数，其值是：

```
MAX (INT((m2-m1+m3)/m3),0)
```

注意：当 $m1>m2$ 且 $m3>0$ 或 $m1<m2$ 且 $m3<0$ 时重复计数值为零。

如果省略了循环控制，则不计算重复计数。其后果就好像设置

了一个很大的正整数，大到它不可能减到零那么大。因此，循环将无休止地进行。

如果循环控制是：

〔,〕 WHILE (sle)

其后果就好象省略循环控制，但在块的第一个可执行语句之前增加一个语句：

IF (.NOT. sle) EXIT

2. 执行循环

(1) 若有重复计数就测试重复计数，若为零则循环终止，DO构造变为不活跃的。

若循环控制是〔,〕 WHILE (sle)形式，则对 sle 求值，若值为.FALSE. 则循环终止，DO 构造变为不活跃的。

(2) 如果重复计数非零，则循环的范围被执行。

(3) 如果有重复计数就减1，有 DO 变量，则加上 m3。注意 DO 变量除在此外不能重新定义或变为无定义。然后返至(1)继续执行。

3. CYCLE 语句

CYCLE 语句的形式是：

CYCLE 〔DO 构造名〕

注意：若 DO 构造名出现，则该语句必须在该 DO 构造的范围之内，否则它至少必须在一个 DO 构造的范围之内，这个范围就是它所在的最内层 DO 构造。CYCLE 语句的执行，截断了前述的循环步骤(2)，立即转移到它所属的 DO 构造去执行循环步骤(3)中。控制转移到 DO 终结的后果与执行属于该构造的 CYCLE 语句是一样的。

4. EXIT 语句

EXIT 语句的形式是：

EXIT 〔DO 构造名〕

注意：若 EXIT 语句引用 DO 构造名，则该语句必须在该 DO 构造的范围内，否则它属于其出现的最内层 DO 构造。该语句的执

行导致循环终止。

5. 循环终止

当发生下列情况之一时，循环终止，DO 构造变为不活跃的。

(1) 执行循环时，当测试到重复计数为零或 sle 的值为 .FALSE.。

(2) 属于 DO 构造的某 EXIT 语句的执行。

(3) 在 DO 构造范围内但属于外部 DO 构造的 EXIT 语句或 CYCLE 语句的执行。

(4) 从 DO 构造范围的语句控制转移到既不是 DO 终结也不是同一 DO 构造范围内的语句，即从内部转移到 DO 构造的外部去时发生的情形。

(5) DO 构造范围内 RETURN 语句的执行。

(6) 任何地方 STOP 语句的执行；或任何其它原因导致程序的终止。

当 DO 构造变为不活跃时，如果有相应的 DO 变量，则保留最后确定的值。

我们来看几个例子：

〔例1〕计算两个数组的张量乘积。

```
DO I=1,M
  DO J=1,N
    C(I,J)=SUM(A(I,J,:) * B(:,I,J))
  END DO
END DO
```

〔例2〕反复读入一个记录，对一般的记录则去进行处理，直到读入文件结束记录或 I/O 的错误条件发生。若读入一负值，则跳过该记录。

```
READ (IUN, '(1X,G14.7)', IOSTAT=IOS) X
DO WHILE (IOS .EQ. 0)
  IF (X.GE.0) THEN
    CALL SUBA(X)
    CALL SUBB(X)
```

```
      ...
    CALL SUBZ(X)
  END IF
  READ (IUN, '(1X,G14.7)', IOSTAT=IOS) X
END DO
```

〔例3〕将上例中不用循环控制而用 CYCLE 语句和 EXIT 语句进行等效的改写。

```
DO
  READ (INU, '(1X,G14.7)', IOSTAT=IOS) X
  IF (IOS .NE. 0) EXIT
  IF (X<0) CYCLE
  CALL SUBA(X)
  CALL SUBB(X)
    ...
  CALL SUBZ(X)
END DO
```

〔例4〕DO 构造的块内包含另一 DO 构造，外层有标号的 DO 语句，而内层采用无标号的 DO 语句。

```
      N=0
      DO 50 I=1,10      !有标号 DO 语句
        J=I
        DO K=1,5        !无标号 DO 语句
          L=K
            N=N+1       !该语句共执行50次
        END DO          !无标号 DO
50      CONTINUE
```

该 DO 构造循环终止时，I=11,J=10,K=6,L=5,N=50。

〔例5〕外层用无标号 DO 语句，内层用有标号 DO 语句的两重嵌套 DO 构造。

```
N=0
DO I=1,10
  J=I
```

```
      DO 60, K=5,1        !此循环永不能执行
        L=K
        N=N+1
60    CONTINUE
    END DO
```

该 DO 构造循环终止时，I=11,J=10,K=5,N=0，而 L 没被这些语句定义。

五、非块 DO 构造

非块 DO 构造是将被淘汰的内容，建议读者不要再使用这些内容，但考虑到读者可能要阅读以前的 Fortran 程序，在此仍作一简单介绍。读者不难看出，一般非块 DO 构造都可用块 DO 构造来替代，而块 DO 构造具有结构化程度更高的程序风格。

非块 DO 构造的形式是：

动作终结 DO 构造

或

外层共享 DO 构造

动作终结 DO 构造的形式是：

```
DO 标号〔循环控制〕
    DO 体
标号 DO 终结动作语句
```

其中 DO 终结动作语句是动作语句，但不能是 CONTINUE 语句，GOTO 语句、RETURN 语句、STOP 语句、EXIT 语句、CYCLE 语句、END FUNCTION 语句、END SUBROUTINE 语句、END PROGRAM 语句、算术 IF 语句或赋值 GOTO 语句。DO 终结动作语句必须有标号，且该标号必须与 DO 语句中标号相同。

外层共享 DO 构造的形式是：

```
DO 标号〔循环控制〕
    DO 体
    共享终结 DO 构造
```

共享终结 DO 构造的形式是：

外层共享 DO 构造

或

内层共享 DO 构造

内层共享 DO 构造的形式是：

DO 标号〔循环控制〕

DO 体

标号 DO 终结共享语句

其中 DO 终结共享语句是动作语句，但不能是 GOTO 语句、RETURN 语句、STOP 语句、EXIT 语句、CYCLE 语句、END FUNCTION 语句、END SUBROUTINE 语句、END PROGRAM 语句、算术 IF 语句或赋值 GOTO 语句。DO 终结共享语句必须有标号，且该标号必须与共享终结 DO 构造的所有 DO 语句的标号相同。从上可见，外层共享 DO 构造是一种嵌套结构。

上述 DO 体后的 DO 终结动作语句、DO 终结共享语句或共享终结 DO 构造称为该构造的 DO 终结。

在作用域单位中，DO 语句引用相同标号的所有 DO 构造都是非块 DO 构造，它们共享由该标号标识的语句。

非块 DO 构造的范围由 DO 体和随后的 DO 终结组成，该范围需满足块的规则，禁止从范围外控制转移到 DO 体或 DO 终结，仅允许从相应的内层共享 DO 构造范围中分支到 DO 终结共享语句。

关于非块 DO 构造的活跃或不活跃的描述与块 DO 构造相同。

非块 DO 构造的执行基本上与前述相同，但在执行循环步骤(1)中发生循环终止时，若所有共享的 DO 终结共享语句的 DO 构造都不活跃时，这些 DO 构造的执行就完成了，否则若还有某些共享此 DO 终结共享语句的 DO 构造还是活跃的，则对 DO 语句的最后被执行的活跃构造继续执行循环步骤(3)。

对于 CYCLE 语句的执行，在非块 DO 构造中 DO 终结动作

语句或 DO 终结共享语句不予执行。控制转移到 DO 终结动作语句或 DO 终结共享语句，从而引起语句或构造自身被执行。除非有进一步的控制转移，就执行 DO 构造当前执行的循环步骤(3)。

下面是非块 DO 构造的例子：

〔例1〕与块 DO 构造的[例3]相类似的例子。

```
   DO 70
     READ (IUN, '(1X,G14.7)', IOSTAT=IOS) X
     IF (IOS .NE. 0) EXIT
     IF (X<0.) GOTO 70
     CALL SUBA(X)
     CALL SUBB(X)
       …
     CALL SUBY(X)
     CYCLE
70 CALL SUBNEG(X)
```

显然这不是块 DO 构造，因为其 DO 终结的形式既不是 END DO 也不是 CONTINUE。

〔例2〕与块 DO 构造的[例4]相等效的例子。

```
    N=0
    DO 100 I=1,10
      J=I
      DO 100 K=1,5
         L=K
100  N=N+1
```

该非块 DO 构造执行终止后，I=11，J=10，K=6，L=5，N=50。

〔例3〕与块 DO 构造的[例5]相等效的例子：

```
    N=0
    DO 200 I=1,10
       J=I
       DO 200 K=5,1
          L=K
```

```
200    N=N+1
```

当该非块 DO 构造执行终止后,I=11,J=10,K=5,N=0,但 L 没有被这些语句所定义。

在此还要对终结语句要多说几句,以往的 Fortran 中,在相应 DO 终结动作语句地方也可用 CONTINUE 语句,但 Fortran 90标准中明确规定不能用。由于 DO 体中还可包含 DO 构造,所以 DO 构造也会形成嵌套。但是,如果 DO 终结都是用不同的 CONTINUE 语句,则它就是由块 DO 构造形成的嵌套;如果有几层共享一个 CONTINUE 语句,则它就是由非块 DO 构造形成的嵌套,这点务必请使用者注意。这就是说,每层 DO 构造用各自 CONTINUE 语句作为其 DO 终结是一个好习惯,我们建议读者采用。

第三节　其它控制语句

一、CONTINUE 语句

CONTINUE 语句的形式是:

```
CONTINUE
```

该语句的执行对执行序列没有影响,将继续顺序执行下一语句。该语句主要引进作为 DO 终结用。特别是多重嵌套时,为使结构更清晰,使用它是值得推荐的。在以往的 Fortran 程序中作为 DO 终结共享语句用得较多。但这种共享在 Fortran 90中并不推荐。

二、STOP 语句

STOP 语句的形式是:

STOP〔可见代码〕

其中可见代码或为1至5个数字,或为默认字符型标量字符常量。

该语句的执行导致可执行程序的执行终止。在终止时,如果有可见代码,则对它可按依赖于处理系统的方式使用,或者显示,

或者打印。1至5个数字的情况下前导0是无效的。

三、PAUSE 语句

PAUSE 语句的形式是：

PAUSE 〔可见代码〕

其中可见代码的解释与上相同。

该语句的执行导致可执行程序的执行被挂起。执行必须是可继续的。在执行它时，如有可见代码，则同上处理。执行的恢复不在程序的控制下进行。如果执行恢复了，则继续顺序执行，就象执行了一个 CONTINUE 语句一样。

小　　结

本章重点介绍了 IF、CASE 和 DO 的构造，特别是 CASE 构造是 Fortran 90增加的，这种块构造对于程序设计的结构化有很大的好处，读者应多下功夫。为了完整起见，又重复介绍了语句标号、GOTO 语句、计算 GOTO 语句、ASSIGN 语句和赋值 GOTO 语句、算术 IF 语句、CONTINUE 语句、STOP 语句、PAUSE 语句。在此要提醒读者 ASSIGN 语句、赋值 GOTO 语句、算术 IF 语句、PAUSE 语句都是将要淘汰的成分，Fortran 90 将不推荐使用它们，特别是 PAUSE 语句可以说是在单道作业条件下的产物，希望读者注意用其它语句来替代它们。

习　　题

1. 试从单分支、双分支、三分支和多分支的各种转移结构来总结本章的内容。

2. 试从执行序列的角度来看为什么有的语句不能作为分支目标语句。

3. 已知三条边长为 a、b 和 c，计算三角形的面积 S。试用不同

的构造来编制计算S的程序。

4. 试用块DO构造和非块DO构造来编制同一目的的程序，并加以比较。

5. 为什么有那么多将被淘汰的成分？对每一种成分说明其要被淘汰的原因。

第八章　输入/输出语句

数据的输入及输出是程序的基本功能之一。在前面几章中，其实都已经使用了 READ 语句及 WRITE 语句来输入数据或输出计算结果。然而，Fortran 语言具有很强的输入/输出功能。例如：它既能输入/输出格式数据，也能输入/输出无格式(内部表示的二进制的)数据；输入/输出数据的格式既可以是由用户程序控制的显式格式，也可以是由处理系统确定的默认格式(如表控格式，名表格式)等。输入/输出语句还提供了很多对文件进行控制和操作的功能。

本章主要介绍数据形式、文件的基本概念以及各种输入/输出功能。

第一节　一般介绍

本节将介绍输入/输出语句的分类，以使读者对 Fortran 的输入及输出功能有大概的了解；将介绍数据的形式、字段、记录及文件等基本概念，这些概念对正确地理解 Fortran 的输入/输出是非常重要的；本节还将介绍输入/输出语句中的数据传送语句的形式，以便读者对这些主要的输入/输出语句的基本形式有初步了解。

一、输入/输出语句分类

Fortran 中的输入/输出语句可以分为两类：数据传送语句和辅助输入/输出语句。

1. 数据传送语句

数据传送语句用来实现程序与部件(指文件及各种输入/输出

设备)之间数据的传送,即数据的输入(读)或输出(写)。它们包括:

READ 语句(数据传送输入语句)

WRITE 语句(数据传送输出语句)

PRINT 语句(数据传送输出语句)

此外,还有一个 FORMAT 语句,它可以用来为上述数据传送语句提供一个指定数据格式的格式说明(参见第三节)。

2. 辅助输入/输出语句

辅助输入/输出语句包括两个文件连接语句,用来实现部件与文件之间的连接或取消这种连接,它们是:

OPEN 语句(打开文件语句)

CLOSE 语句(关闭文件语句)。

此外,还包括一个文件查询语句,用来查询部件与文件之间的连接的属性等,它是:

INQUIRE 语句(查询文件语句)

还包括三个文件定位语句,用来改变文件当前指针所指向的文件位置,它们是:

BACKSPACE 语句(回退语句)

ENDFILE 语句(写出文件结束记录的语句)

REWIND 语句(文件反绕语句)

二、基本概念

Fortran 的输入/输出语句最初是为了在行式打印机上输出具有完善格式的可视结果,或者读入这类格式数据而设计的。事实上,这些格式数据也可以存储在诸如磁盘文件这样的外部文件中。因而,Fortran 的输入/输出最终被抽象为是对文件进行的。

为此,本章常将行式打印机上的纸或者监视器上的显示屏,甚至于终端上的键盘(键入的数据是可视的)等这样的媒体称之为(外部)显现媒体,而将磁盘或磁带等这样的媒体称之为(外部)存储媒体。

本节重点介绍输入/输出中与数据、字段、记录及文件有关的

一些概念。

1. 格式数据及无格式数据

在 Fortran 语言的变量中，数据通常是以二进制的形式表示的。例如，整数 17 可以用 16 位二进制形式表示为 0000 0000 0001 0001 这种形式的数据。在计算机中，这是数据的内部表示形式，它非常适宜于实现计算机内部的运算，但不便于人的阅读。按照 Fortran 语言输入/输出中的术语，这种数据被称作无格式数据。

在计算机中，同一个数据还可以用便于人阅读的形式表示。例如，上面的整数可直观地表示为十进制的 17 或＋17 等。这是该数据的外部表示形式。驻存在外部存储媒体上的文件中，这种数据具有字符串的形式，且能直接被印刷输出或在显示屏上显示，也很容易由人工准备数据以便输入。按照 Fortran 语言输入/输出中的术语，这种可视的数据被称作格式数据。

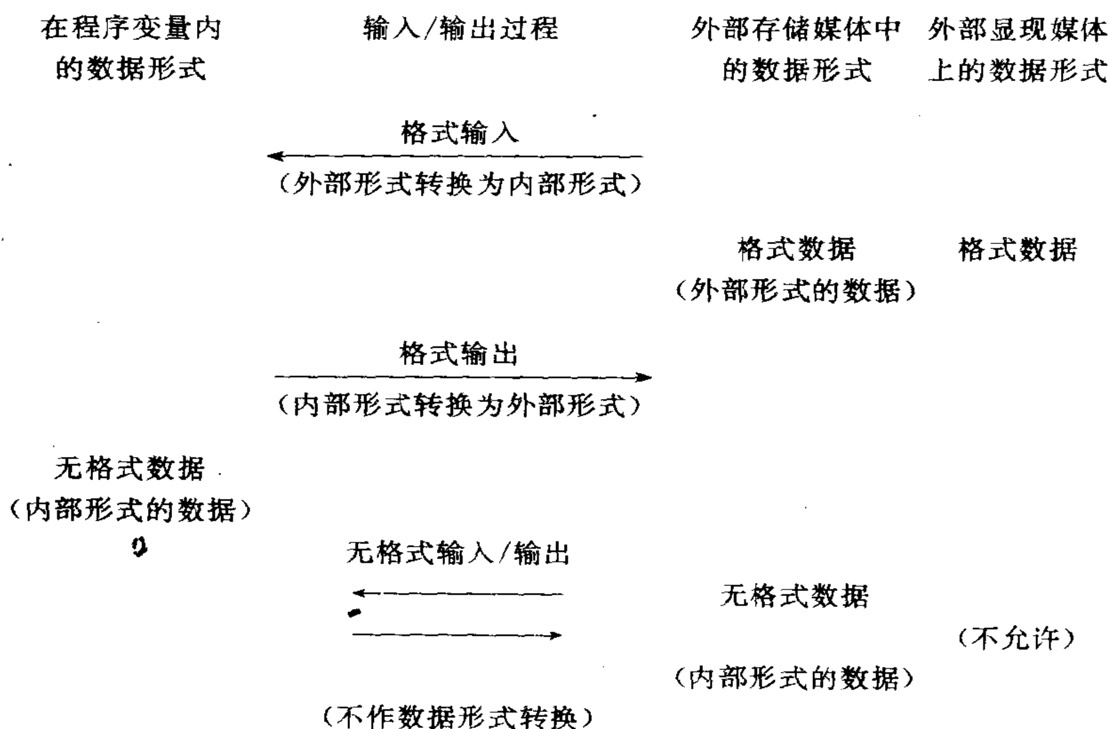


图 8.1 输入/输出过程中的数据形式转换

Fortran 语言提供了无格式输入/输出能力，主要用来以数据

的内部表示形式“存储”程序的计算结果；Fortran 还提供了格式输入/输出能力，主要用来向程序提供原始数据（格式数据输入），或者输出可视的计算结果（格式数据输出）。

无格式输入/输出，其间不涉及对数据形式的二进制到十进制转换，执行速度较快，而格式数据输入/输出，其间需要进行数据形式的转换，执行速度较慢。它们之间的执行区别如图 8.1 所示。

2. 字段

按照 Fortran 的传统概念，在格式数据的形式中，一个数据在输入/输出记录（存储媒体）中所占字符位置，或者在外部显现媒体上所占的字符位置，称作字段。在 Fortran 90 中，这一定义对 Fortran 字符宽度的字符仍然适用。下面所示是一个格式输出语句及其产生的实型数据的字段：

```
PRINT "(1X, F6.1)", 45.2
   □   □   □   4   5   .   2
   ↑   1   2   3   4   5   6
控制走纸 ↑                   ↑
用的空格 └───────────────────┘
                 |
           字段(宽度=6)
```

其中，由编辑描述符 1X 产生的第一个空格，用作行式打印机的走纸控制，在外部显现媒体上不被印出（详见第三节）。

注意：在本章中为了精确地表示字段中的空格字符将以符号□表示。而在外部显现媒体上，它们不具有可视的图形而仅为空白。

每个字段由若干个字符组成。构成一个字段的字符个数，称为该字段的宽度。

按照 Fortran 的传统，每个 Fortran 字符在外部显现媒体上都占有相同的宽度。Fortran 字符集是一个等宽字符集。

在计算机中引入汉字这类民族字符后，它们与 Fortran 字符是不等宽的。其宽度比通常被规定为 2：1，这时，包括汉字信息的格式输入/输出将变得较为复杂。

不少民族（如蒙文、阿拉伯文等）的字符是变宽字符。包括这类

字符的格式输入/输出更为复杂。对这类数据输入/输出的解释，是依赖于处理系统的。

对于无格式数据，不存在字段的概念。

3. 格式记录与无格式记录

一个记录由一组数据组成。

按照 Fortran 传统的概念，记录是程序与文件交换信息的最小单位。无论是输出格式记录或是无格式记录，都是首先将数据组织成一个记录，暂存于一个缓冲区中，然后再成记录地传送到文件上。类似的，读入数据是先从文件中将一个记录读入缓冲区，然后再将记录中的数据分别传送到 Fortran 程序的变量中。

然而，Fortran 90 中新扩充的非推进式的顺序格式输入/输出提供了对部份记录进行格式输入与输出的能力，以便实现诸如会话式输入/输出能力(详见第五节)。

Fortran 中有两类记录：格式记录及无格式记录。

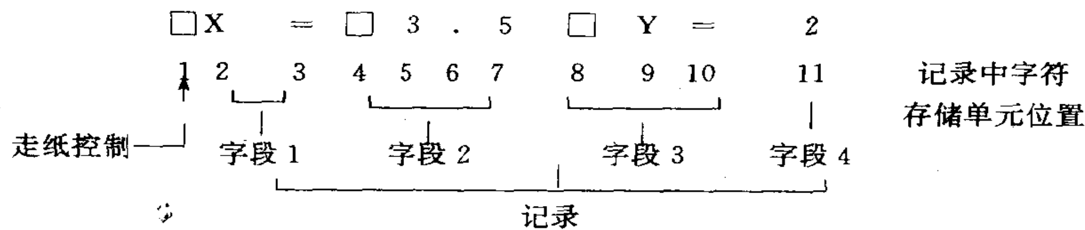
(1)格式记录。格式记录是只包含格式数据的记录。一个格式记录由若干个字段组成。

在显现媒体上，一个格式记录就是(例如印刷纸上的)一行。

下面所示是一个格式输出语句及其在存储媒体中(或输入/输出缓冲区中)产生的格式记录的例子：

```
PRINT '(1X,"X=",F4.1,"□Y=",I1)',3.5,2
```

输出记录形式为：



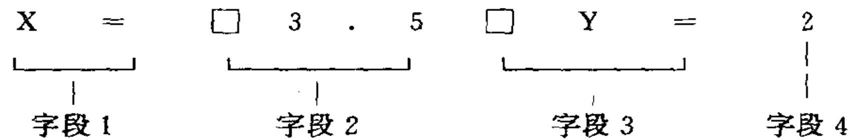
格式记录的长度，原则上依赖于该记录中字符的个数。但这也可能是依赖于处理系统的。

格式记录可以用人工键入数据的方法建立，用于数据输入；也可以由 Fortran 程序中的输出语句建立，用于输出人可阅读的数

据，或者用作其后的进一步处理。

格式数据很少依赖于处理系统，主要依赖于编码字符集（如ASCII码或EBCDIC码），这种数据比较便于用作不同处理系统之间的数据交换。

上述在存储媒体中的记录，在外部显现媒体上显现的形式为：



注意，存储媒体中格式记录的第一个字符（空格）未被印出。

（2）无格式记录。无格式记录是只包含无格式数据的记录。一个无格式记录由若干个无格式数据组成，并存储在外部存储媒体上的文件中，记录中的数据可以是任何一种类型的。

下面所示是一个无格式输出语句及其在存储媒体中产生的无格式记录的例子：

WRITE (UNIT=7, REC=N) +17, −0.00045, 'ABCD'

1	00　11[注]	整型数据(17的内部表示)
2	BE　1D	实型数据(−0.000450的内部表示)
3	7D　BF	
	41　42	字符型数据("ABCD"的内部表示)
n	43　44	

其中n为记录长度。

注：内部值均以十六进制表示且是依赖于处理系统的。

无格式记录的长度，依赖于该记录中数据的个数，而无格式记录长度的单位则依赖于处理系统。

在一个处理系统中，由某个输入/输出语句中的输入/输出表所建立的无格式记录的长度，可以通过执行INQUIRE语句（参见第四节）得到。

无格式记录比较适合于Fortran程序用来存储中间计算结果或者存储其后要作进一步处理的计算结果。

无格式数据的形式依赖于处理系统，它不宜用于不同类型的计算机之间的数据交换。

此外，还有一种记录称为文件结束记录，在顺序访问文件时，它被用来指示文件的结束，详见下面对文件概念的介绍。

4. 文件

文件(File)是若干记录的一个序列。

有两类文件，外部文件及内部文件。内部文件是 Fortran 中一个特别的概念。

(1) 文件到部件的连接。在文件系统中(操作系统中)，用户必须通过文件名来引用文件。然而，在 Fortran 语言中，输入/输出语句是通过其形式为正整数的部件号来访问某个文件的。因而，在程序通过执行某个输入/输出语句而读写某个文件之前，必须将被访问的文件与该输入/输出语句中所使用的部件"连接"起来。即，建立"文件到部件的连接"，使得该输入/输出语句能通过对该部件的访问来访问文件。图 8.2 是建立这种连接的示意图。

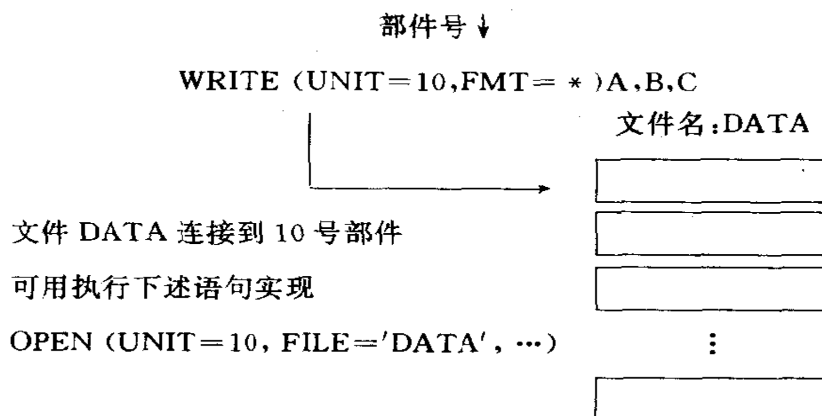


图 8.2 文件到部件的连接

图中，首先要执行一个 OPEN 语句，将名为 DATA 的文件连接到 10 号部件上。其后，当执行部件号为 10 的输入/输出语句，都将向文件 DATA 输入或输出数据。除 OPEN 语句外，将文件连接到部件也可由操作系统完成。

在将文件连接到部件时，还可以将许多属性赋给这一"连接"，例如，是为直接访问而连接的或者是为顺序访问而连接的等。因而

这类属性是属于某次“连接”的。详见下面的讨论。

将文件连接到部件，也可以说是打开该文件，或者说打开该部件。

(2) 外部文件。外部文件是驻存在对一个 Fortran 程序而言是外部媒体上的文件。这类媒体可以是磁盘、磁带等存储媒体，也可以是印刷纸、显示屏这样的显现媒体。而内部文件则是把一个 Fortran 程序内部的默认字符型变量，用作文件的存储媒体，即以内存用作文件的存储媒体。在此介绍外部文件。

一个文件由若干个记录组成，可参见图 8.2 及图 8.3。

一个外部文件中的记录，或者全部是格式记录，用于格式数据的输入/输出；或者全部是无格式记录，用于无格式数据的输入/输出。

由文件系统提供的外部文件，通常既可以用于格式数据的输入/输出，也可用于无格式数据的输入/输出。一个特定的外部文件实际是用于格式或无格式数据的输入/输出，是在将文件连接到部件时，由 OPEN 语句中的 FMT＝说明符决定的，详见第四节中的 OPEN 语句。

驻存在外部显现媒体上的文件只能用于格式数据的输入/输出。

(3) 外部文件的访问方法。访问外部文件的方法有两种，即顺序访问及直接访问。访问一个特定文件的方法则是在将文件连接到部件时，由 OPEN 语句中的 ACCESS＝说明符决定的。详见 OPEN 语句。

a. 直接访问。所谓直接访问，就是通过外部文件中各记录的记录号去访问某个特定的记录，如图 8.3 所示。

此时，文件有下述特性：

①文件中的每一个记录都有一个记录号。记录号是在输出语句写出该记录时确定的。一个记录的记录号一旦建立，就不能改变。且该记录也不能被删除，只能对具有某记录号的记录重写。直接访问时，文件中记录的次序被规定为这些记录的记录号的次序。

这一规定使得以直接访问方法写入的记录，可以在其后用顺序访问方式重新打开该文件时，用顺序访问方法读出那些记录。

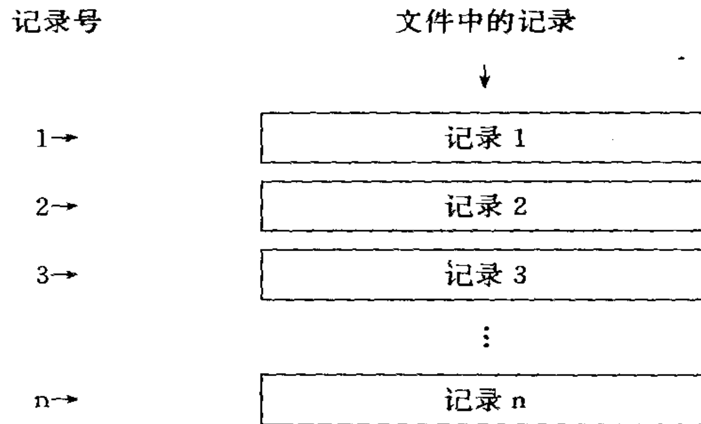


图 8.3　文件的直接访问

②在文件与部件的当前的连接中，则只能用直接访问输入/输出（参见第四节）来写出或读入这些记录。

③一个文件中的所有记录都具有相同的记录长度。

④记录号不必连续。例如，一个文件中可以有 3 号记录而没有 2 号记录。

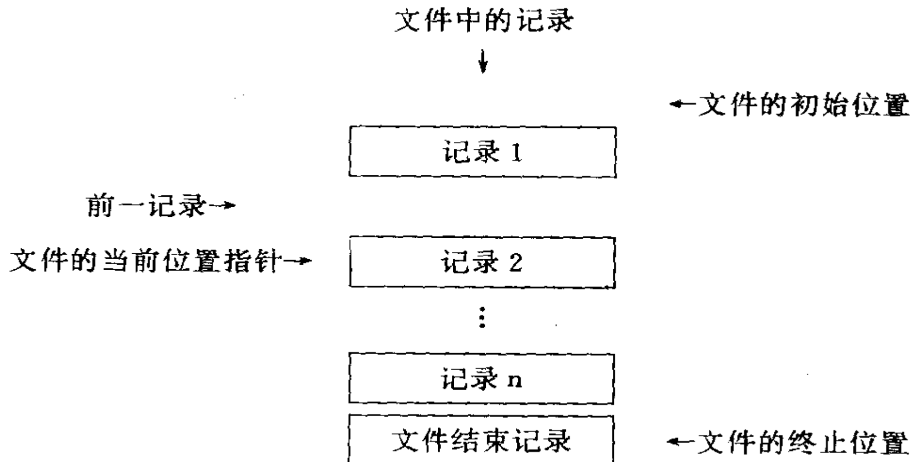


图 8.4 文件的顺序访问

b. 顺序访问。所谓顺序访问，就是按文件中记录的次序访问其中的记录。顺序访问有两种情况，其一是文件中的记录，是在先

前的连接中，由直接访问方法建立的，如图 8.3 所示。其二是文件中的记录，是由顺序访问方法建立的，如图 8.4 所示。

图中，由顺序访问方法建立的文件，文件内有一个假想的"文件当前位置指针"，它可以位于"文件的初始位置"，或者位于两个记录之间，或者位于"文件的终止位置"。当输入/输出语句读写文件时，是对文件当前位置指针下面一个记录开始进行的。在执行一个格式输入/输出语句期间，或者在执行非推进式的格式输入/输出之后，文件的当前位置指针则指向当前正在处理的格式记录的内部。详见第四节。

在顺序访问时，文件有下述特性：

①若文件中的记录是由顺序输出所建立的，则在其后对此文件的访问中，只能按这一已建立的次序进行访问。

②若文件中的记录是以直接访问的方法建立的(此时，每个记录均有记录号)，则在其后对此文件的顺序访问中，访问各记录的次序，就是该文件中记录号的次序。

③文件中的最后一个记录可以是一个文件结束记录。文件结束记录用以指示此记录之后不再存在记录。

c. 文件结束记录

文件结束记录是一种特殊的记录。在顺序访问时，它用来指示一个文件的结束，如图 8.4 中所示。因而，一个文件最多只能有一个文件结束记录。

文件结束记录中没有值，也没有记录长度的概念。

文件结束记录可以通过执行一个 ENDFILE 语句而显式地产生，它也常在下列条件下被隐式地写入到文件中：当最近的一次引用该文件的语句是数据传送输出语句(其间没有对该文件执行文件定位语句)且：

①对该文件执行了 REWIND 语句或 BACKSPACE 语句。

②该文件(通过相关部件)被关闭；或者

③非错误性的程序执行终止。

在执行数据传输输入语句时，若遇到文件结束记录，READ

语句将通过 END=控制说明符被告知这一情况(参见第四节)。

当文件连接用于直接访问时,不存在文件结束的概念。

(4) 内部文件

Fortran 中还有一个独特的概念,即内部文件。

内部文件是只驻存在内存中的文件,而且是以一个默认字符型变量用作内部文件的标识。

若该变量是一个标量变量,则该内部文件只包含一个记录,且记录的长度就是该标量变量的长度,例如:

```
CHARACTER (LEN=50):: A, B
  ...
WRITE (UNIT=A, FMT="(3I5)") I, J, K
```

字符型标量变量 A 可通过控制说明符 UNIT=A 被指定用作内部文件,其记录长度为 50。

若该变量是字符型数组变量,则该变量被当作数组元素的一个序列对待,且每个数组元素是内部文件中的一个记录,各记录均具有相同的长度,例如:

```
CHARACTER (LEN=40), DIMENSION(10)::B
  ...
WRITE (UNIT=B, FMT="(3F10.2)") (X(I), I=1,30)
```

字符型数组 B 可通过控制说明符 UNIT=B 被指定用作内部文件,每个文件包括 10 个记录,每个记录可放 40 个字符。

引入内部文件概念的目的,是企图借助格式输入/输出语句提供数据形式的转换能力,使得只依靠内存就能实现无格式数据与格式数据之间的转换。例如:

```
CHARACTER (LEN=7) :: CHAR
A=1.23456E2
WRITE (UNIT=CHAR, FMT="(F7.2)")A
  ...
```

执行上述语句后,在 A 中是无格式(二进制)数据“1.23456E2”,而在 CHAR 中是格式(十进制)数据“ 123.45”。此程序中,字符型标量变量名 CHAR 被写语句当作内部文件名使用。

对内部文件的输入/输出，详见第五节。

(5) 文件概念的小结

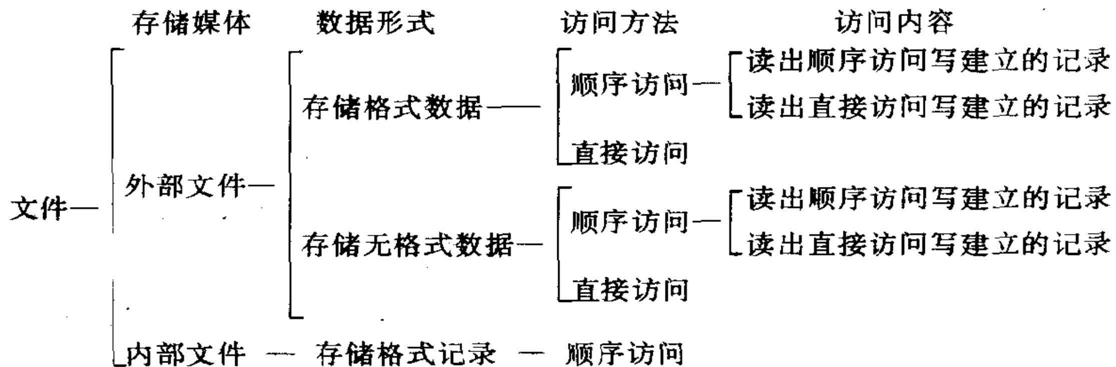
至此，介绍了与 Fortran 文件有关的主要概念：文件是若干记录的集合；按照所驻存的媒体不同，文件分为外部文件及内部文件；按记录中数据形式不同，文件可以是全部由格式记录组成，或者是全部由无格式记录组成。一个格式记录由字段的一个序列组成，而一个字段又由字符的一个序列组成；一个无格式记录由值的一个序列组成，并且其记录长度依赖于处理系统的度量单位。

按访问方法的不同，文件可以用顺序访问方法访问，也可以用直接访问方法访问。在直接访问时，每个记录都有一个记录号。

以直接访问方法建立的文件，在其后的连接中，既可以连接用于直接访问，也可以连接用于顺序访问。但是，用顺序访问方法建立的文件，其后只能连接用作顺序访问。

对内部文件只能进行顺序格式输入/输出。

上述有关文件的使用分类，可以直观地总结如下：



三、数据传送语句的一般介绍

数据传送语句分为两类：数据传送输入语句，包括 READ 语句；数据传送输出语句，包括 WRITE 语句及 PRINT 语句。此外，还有一个 FORMAT 语句，需要时，它可以为数据传送输入/输出语句提供格式说明。

数据传送语句共有 4 种形式，它们是：

READ (输入控制说明符表)〔输入项表〕

READ 格式〔,输入项表〕

WRITE (输出控制说明符表)〔输出项表〕

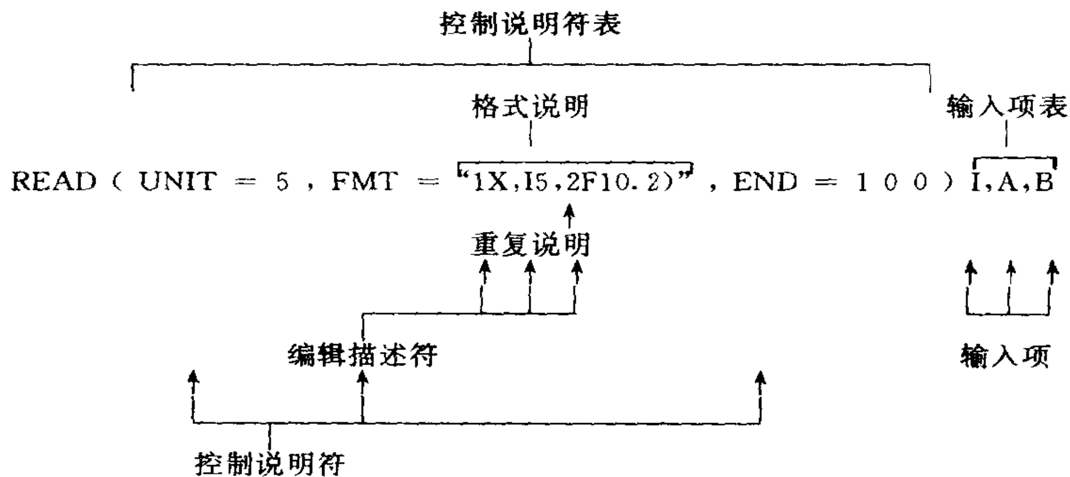
PRINT 格式〔,输出项表〕

其中,第二及第四种形式是简缩的输入/输出语句,其中不包括输入/输出控制说明符表。

FORMAT 语句的形式是:

标号　　FORMAT　　格式说明

下面以 READ 语句为例,解释语句的形式。



一个数据传送语句可以由三个主要部份组成,即:

(1)输入/输出项表(输入项表或输出项表)。输入项表用来指定要输入值的一些变量,如上述例中的 I、A、B 等;输出项表用来指定要输出的值,它可以是一些变量或表达式等。在第二节中将对输入/输出项表作专门介绍。

(2)格式说明。格式说明用来显式地指定要输入或输出数据的格式。本例中,格式说明是以字符型字面常量的形式(即,括在撇号中的串)给出的。格式说明还可以由 FORMAT 语句给出。例如,上例可改写为下述两个语句:

```
      READ (UNIT=5, FMT=900, END=100) I, A, B
900   FORMAT (1X, I5, 2F10.2)
```

并具有相同的执行效果。执行此语句时所使用的格式说明,是通过

FMT＝900 控制说明符指定的标号 900，由相应的 FORMAT 语句提供的。

在数据传送语句中直接写出格式说明，可使程序显得紧凑。但当一个程序单元中的多个数据传送语句都要使用同一个格式说明时，或者一个语句要使用的格式说明很庞杂时，则更宜于由 FORMAT 语句单独提供格式说明。

在第三节中将对格式说明作专门介绍。

(3)输入/输出控制说明符表。输入/输出控制说明符表用来指定与该语句执行有关的一些控制说明符，从而提供不同的输入/输出功能。例如：部件说明符 UNIT＝5 指定从部件 5 读入数据，格式说明符 FMT＝"(1X，I5，2F10.2)"指定要读入数据的格式说明是(1X，I5，2F10.2)，而当不出现 FMT＝说明符时，表示指定了无格式输入/输出。

在不少控制说明符之间，还有较为复杂的相互搭配及制约关系。因此，本书将以读者便于掌握使用为目的，将控制说明符分散在具有不同功能的输入/输出语句中分别加以介绍。控制说明符的详细介绍见第三节和第四节。

许多程序(尤其是简单程序)并不需要复杂的输入/输出控制(例如，遇到文件结束时转入某个语句执行等)，这时，可以使用前述简缩形式的数据输入/输出语句。例如，下面的语句：

```
      READ "(I5, F10.2)", I, A, B 或者
      READ 900, I, A, B
  900 FORMAT (I5, F10.2)
```

以及

```
      PRINT "(I5, F10.2)", J, X, Y 或者
      PRINT 900, J, X, Y
  900 FORMAT (I5, F10.2)
```

都只包括输入/输出项表及格式说明。而不用给出输入/输出控制说明符表。在这种语句形式中，语句中不能指定输入/输出部件号，此时的输入/输出是对系统默认的部件进行的。例如，终端上的键

盘及显示屏，或者行式打印机，或者某个磁盘文件等。

四、小结

本节主要向读者介绍输入/输出中最基本的概念，即与文件有关的一些概念；也介绍了输入/输出语句的主要概念，即输入/输出语句的分类以及对数据传送（输入/输出）语句的总貌等。

在各种高级程序设计语言中，Fortran 语言是输入/输出功能较强且较实用的一种语言。

在 Fortran 90 中，没有增加新的输入/输出语句，但多数语句的功能均有所增强。

在后面的三节中，将对数据传送语句作详细介绍。

第二节　表控格式输入/输出及输入/输出项表

表控格式输入/输出是一种不需要由输入/输出语句显式地指定每一个外部数据形式的格式输入/输出。外部数据的形式是分别由 Fortran 标准及处理系统协同决定的。因而，对使用者而言，它是一种默认格式的输入/输出。其优点是语句中不用给出描述数据形式的格式说明，从而学习容易，使用简单。其不足之处是用户几乎不具有控制数据形式及记录格式的能力。

一、表控格式输入/输出一般介绍

在数据传送输入/输出语句的形式中，通常用来显式地指定“格式”的地方，若给出了星号 *，即表示指定了表控格式。下面是 4 个典型的指定了表控格式输入/输出的语句的例子：

```
READ (UNIT=5,FMT= * ) I,A,B
READ * ,I,A,B
WRITE (6, * ) J,X,Y
PRINT * ,J,X,Y
```

上述语句中的 4 个星号 * 都是用来指定以表控格式进行输入或输

出。

在下一个表控格式输入语句的例子中：

READ *,I,A,B,C,D,E

且外部记录为：

325,□42.6,□76,□33.5E2,2*0.0

则此输入语句将从左至右顺序读入外部记录中由逗号分隔的六个数据。其中，第一个数据为整型，后五个数据为实型，其中 2*0.0 表示两个 0.0。并分别赋给 READ 语句中的六个变量。

再看一个输出的例子：

X=2.0

PRINT *,"Square—root of x=",SQRT(X)

在系统上执行后，将产生下述结果：

Square—root of x=□□□□□□1.41421356

其中，实型输出值的形式是依赖于处理系统的。详见下面对输出数据形式的描述。

二、表控格式输入

表控格式输入用来输入表控格式的数据。其语句的两种形式分别为：

READ (〔UNIT=〕输入部件 &
　　,〔FMT=〕 * &
　　〔,IOSTAT=默认整型标量变量〕 &
　　〔,ERR=标号〕 &
　　〔,END=标号〕 &
　　)〔输入项表〕

或者

READ *,〔输入项表〕

其中，输入项表将在后面详细介绍。两个必须的控制说明符解释如下：

〔UNIT=〕输入部件必须是一个非负整型表达式，它指示一个输入部件，它指向一个顺序访问的文件。此部件也可以用一个星

号 * 表示，它指出一个由处理系统决定的(默认的)外部文件，且此部件与未显式地指定部件的 READ 语句及 PRINT 语句的使用的部件相同，例如：

```
READ (UNIT= * ,FMT= * ) I
```

与

```
READ * ,A
```

指定同一个默认的输入部件。而

```
WRITE (UNIT= * ,FMT="(1X,I5)") I
```

与

```
PRINT "(1X,F10.2)", A
```

则均指定一个默认的输出部件，但输入部件和输出部件不一定是同一个部件。

若省略任选的字符串 UNIT=，则部件说明符必须是控制说明符表中的第一项。

〔FMT=〕* 用以指定以表控格式输入。若省略任选的字符串 FMT=，则字符串 UNIT=也必须省略，且格式说明符必须是控制说明符表中的第二项。例如：

```
READ (5, * ) I,A,B
```

表示从 5 号设备以表控格式输入。

其它的任选的控制说明符 IOSTAT=、ERR=及 END=等请参见第五节。

表控格式输入数据形式解释如下：

(1)外部记录由一些值(数据)及值分隔符(如逗号，斜线等)组成。

a. 值。值具有下述四种形式：

值的形式	注释
c	c 可以是整型、实型、复型、逻辑型及字符型等各种类型的字面常量，如 439，3.25 等。允许带种别参数如 35 _ 2，43.6 _ 5 等。但种别参数不能是有名常量。
r * c	表示 r 个相同的值 c。r 是非零整常量，且不具有种别参数。如 2 * 0.0 表示 2 个相继的值 0.0。

NULL(无值)	输入记录中给出的 NULL 值。NULL 值意指不向与之相应的输入项提供值。在值分隔符之间未给出值(例如,在形如“,,”的两个值分隔符间)时,就给出了一个 NULL 值。
r *	表示 r 个相同的 NULL 值。如 10 * 表示 10 个 NULL 值。

在上述各种值的形式的内部,除在字符常量中以外,均不允许有空格字符出现,因为空格已被用作值分隔符。例如,10 * □2.0 被认为是 10 个 NULL 值,再后随一个值 2.0 而不是 10 个 2.0。

b. 值分隔符。值分隔符有三个,如下所示:

值分隔符	作　用	注　释
,(逗号)	分隔值	其前后可以有若干个空格。
/(斜线)	输入值终止点	其前后可以有若干个空格,其后的值不再被输入。
□(空格)	分隔值(* 注)	多个相继的空格当作一个值分隔符看待。它必须位于两个非空值(形如 c、r * c 及 r * 的值)之间或在最后一个非空值之后。且记录结束也具有如同空格一样的作用。

* 注　字符字面常量中的空格不是值分隔符。

(2)输入项的类型与外部数据的形式之间的对应关系如下:

输入项类型	外部数据的形式	注　释
整型变量	适于 I 编辑输入的数据	参见第三节 I 编辑。
实型变量	适于 F 编辑输入的数据	参见第三节 F 编辑。
复型变量	(实部,虚部)	实部及虚部数据的形式均同实型数据。
逻辑型变量	适于 L 编辑输入的数据	参见第三节 L 编辑。
字符型变量	见下面(3)中的描述	种别类型参数必须相同。

对上述输入数据形式的补充解释如下:

①复型数据实部及虚部的前后均可出现空格。记录结束可以出现在实部与虚部之间。

②在逻辑值 T 或 F 之后的任选字符中不得包含逗号、斜线或空格等值分隔符。

③一个字符型常量可以跨记录,但不能在两个相继的撇号'或引号"之间断开。跨记录时,记录结束不在字符常量中形成一个空

格或其它如“新行”之类的字符。只要能避免歧义性，即，若某个默认字符型常量中不包含可用作值分隔符的字符，也不跨记录，而且字符常量中前导的字符也不构成如同重复值的 r＊这样的形式，那么，字符常量的界限符'或"就可以省略不用。例如：

'ABC□DEF'

被认为是一个字符常量，而

ABC□DEF

被认为是两个字符常量，且以值分隔符□分隔。

若省略了界限符'或"，则字符常量内出现的撇号'及引号"均不用重写两次。

假设 len 是下一个字符型输入项的长度，w 是字符常量的字段宽度，则：

若 len<＝w，则读入最左 len 个字符。

若 len> w，则将 w 个字符读入到输入项的左边，右边填充空格。

(3)对 NULL 值的补充说明。NULL 值不改变对应输入项的定义状态。

在下述情况下将给出 NULL 值：

① 在两个值分隔符之间没有其它字符时。

② 在执行一个表控格式输入语句时，读入的第一个记录中的第一个值分隔符之前没有字符时。

③ 由 r＊形式给出。

此外，一个 NULL 值可以用来表示整个 NULL 值的复型数据。但它不能表示复型数据的实部或虚部的某一部份。例如，不允许形为(－2.5，)的复型数据。输入时，当遇到值分隔符/时，立即终止该输入语句的执行。对于输入项表中所余表项的处置，如同读了 NULL 值一样。在隐 DO 中(参见本节的四)也是如此。

表控格式输入的实例如下：

```
INTEGER I; REAL X(8)
CHARACTER (11) P; CHARACTER (LEN=4,KIND=3)::C
```

```
COMPLEX Z; LOGICAL G
  ...
READ *,I,X,P,C,Z,G
```

要输入的数据被安排在两个记录中：

```
1234,1234,,2*2.3,4*
ISN'T_BOB'S,3_'汉字',(1.23,4.65),.TEXAS
```

上述程序执行的结果为：

变量	值	注　释
I	1234	
X(1)	1234.0	类型相容
X(2)	未改变	遇到 NULL 值
X(3)	2.3	
X(4)	2.3	
x(5)至X(8)	未改变	遇到 NULL 值
P	ISN'T—BOB'S	
C	3_'汉字'	
Z	(1.23,4.65)	
G	.TRUE.	

三、表控格式输出

表控格式输出用来输出表控格式的数据。其语句的形式为：

```
WRITE ([UNIT=]输出部件 &
,[FMT=] * &
[,IOSTAT=默认整型标量变量]&
[,ERR=标号]&
)[输出项表]
```

或者

```
PRINT * [,输出项表]
```

其中，控制说明符中关于输出部件的解释同表控格式输入中输入部件的描述。上述语句形式中星号 * 指定了表控格式。

表控格式输出产生的值，其形式与表控格式输入的值相类似，主要规则如下：

(1)输出记录由值及值分隔符组成：通常以空格或逗号作值分隔符；对于连续几个无界限符'或"的字符型常量，其值由一个或多个空格分隔；或者由前后均可任选地具有一个或多个空格的逗号分隔。

(2)当一个记录(一行)被认为已写满时，处理系统可以在输出项表中的任何一项前开始一个新记录。记录结束可以出现在复型常量中逗号的两侧及字符型常量的内部。

(3)记录中连续 r 个相同的常量 c，处理系统可以用 r * c 的形式输出。

(4)输出项中各种类型值的输出形式如下：

输出项类型	输出数据的形式	注　释
整型	具有如同 I 编辑输出的效果	参见第三节
实型	详见下面(1)中描述	
复型	(实部，虚部)	实部及虚部的数据形式均与实型值相同
逻辑型	.TRUE. 为 T，.FALSE. 为 F	
字符型	详见下面②中描述	

对上述输出数据形式的补充解释如下：

① 根据实型数据值的范围不同，处理系统将选择一种合理的输出形式，使之分别具有如同 Fw.d 或 1PEw.dEe 编辑描述符(即小数点前有一位数字，详见第三节)输出的效果。且其中的输出字段宽度 w，小数部分的位数 d 以及指数部分的位数 e 都是依赖于处理系统的。

例如，某个处理系统可以按如下规则处理：

若 $10^{-1}<$|输出值|$<10^{6}$，则以 0PF16.8 的形式输出该值。如，下面的语句：

```
PRINT * ,"X=",-123.456
```

将输出：

```
X=□□□-123.45600000
```

而若|输出值|<=0.1

或

|输出值|>=10^6

则以 1PE16.8E2 的形式输出该值。如,下面的语句:

```
PRINT *,"Y=",23456789.123,"□□Z=",0.0000456
```

将输出:

```
Y=□□2.34567891E+07□□Z=□□4.56000000E-05
```

其它处理系统可用另外的 w,d 及 e 参数输出这些值。

② 字符型值的输出。字符型值的输出形式,受 OPEN 语句打开相关文件时所使用的 DELIM=(界限符)说明符中给出值(参见第四节)的控制:

若不给出此说明符,或者此说明符给出的值为 DELIM="NONE",或者是向内部文件输出,则输出形式为:

· 输出的字符常量不被限定符'或"括起来。

· 相邻的字符常量之间不由值分隔符分隔。例如:

```
CHARACTER (LEN=15) :: NAME
   ...
NAME='Mr.□Greenwood'
PRINT *, 'The□name□is□',NAME
```

将印出:

```
The□name□is□Mr.□Greenwood□□
```

等等。

· 字符常量内的'或"字符不被重复两次输出。

· 当在一个字符常量内要开始一个新记录时,在新记录的开始处,处理系统自动插入一个空格,用以正确地控制印刷装置的走行控制(参见第三节)。例如:

```
PRINT *, 'PAI=',+3.141592653
```

输出结果可以是:

```
PAI=□□□□□□3.14159265
```

若在相关 OPEN 语句中给出了 DELIM=说明符,且若给出的值分别为 DELIM='APOSTROPHE'(即撇号')或者 DELIM='QUOTE'(即引号"),则:

· 输出的字符常量将被相应的界限符'或者"括起来,而且,字

符常量自身可能还有一个前导的种别参数及一个下划线,例如:

```
CHARACTER (LEN=2, KIND=3) CHAR
  ...
CHAR=3 _'汉字'
PRINT * , CHAR
```

其输出结果在指定了 DELIM='APOSTROPHE'时是:

```
3 _'汉字'
```

而在指定了 DELIM='QUOTE'时是:

```
3 _"汉字"
```

- 字符常量的前后各带一个值分隔符。
- 字符常量内相应的撇号'或者引号"将被重复两次输出。

(5)输出时不会产生 NULL 值,也不会作为值分隔符而输出斜线/。

注意,除了带界限符的字符常量被分隔在两个记录上这一情况以外,系统在每个记录的开始处,也都自动填充一个空格字符,以便在印刷时用作走纸控制(参见第三节)。

四、输入/输出项表

本节已介绍了表控格式输入/输出中数据及记录的形式,这里将详细介绍语句中的输入项表及输出项表。

输入项表和输出项表统称输入/输出项表,它是输入/输出项的一个表,项间以逗号分开,其形式如下:

〔输入/输出项〔,输入/输出项〕...〕

输入/输出项表在输入/输出语句中用来指明要输入或输出的一些对象。无论在表控格式输入/输出中,以及在后面几节要介绍的其它格式的输入/输出中,其形式都是一样的。

在输入语句中,一个输入项可以是:

(1)变量。包括标量变量、数组变量、结构、子串及指针变量等;

(2)输入/输出隐 DO。

在输出语句中,一个输出项可以是:

(1)表达式。

(2)输入/输出隐 DO。

下面是带有各种输入/输出项表的一些例句：

```
REAL, DIMENSION (50) ::A
CHARACTER (LEN=20) CHAR
  ...
READ *,I                                        ! 表控格式,输入标量
READ "(10F8.2)",A                               ! 显式格式,输入数组
READ *,A(1),A(2)                                ! 输入数组元素
PRINT "(1X,5F8.2)",A(1:5)                       ! 输出数组片段
PRINT "(1X,A)",CHAR                             ! 输出字符型变量
PRINT "(1X,2F10.3)",I*A(1),SQRT(I*A(1)) ! 输出表达式
WRITE (UNIT=5,FMT="(1X,5F10.2)"), &
    (A(I),SIN(A(I)),I=1,5,1)                    ! 隐 DO
```

对各类输入/输出项解释如下：

(1)当输入/输出项为标量变量(包括数组元素)时,则对该标量变量进行输入或输出。例如,下面两个语句：

```
READ "(I5)", I
PRINT "(I10)", I
```

将从处理系统默认的输入设备读入 I,并在处理系统默认的输出设备上输出 I 的值。注意,在输出时,输出变量必须是有定义的。

(2)若输入/输出项是一个数组(或数组片段),则它与按数组中元素的顺序列出其全部元素效果相同,例如：

```
REAL, DIMENSION (100) ::A
  ...
PRINT "(1X,10F10.2)", A
```

将输出 A(1)至 A(100)的 100 个值。但请注意,数组中的任何一个元素均不能影响此输入项中任何表达式的值。而且,在一个输入项中,任何一个元素最多只能出现一次。例如,下例中：

```
INTEGER, DIMENSION (4) :: I,J
  ...
J=(/1,2,3,3/)
```

```
READ "(4I5)", I(I(1):I(2))
READ "(4I5)", I(J)
```

两个读语句中都出现了错误。

(3)若输入/输出项是导出类型的对象，且若为格式输入/输出，则如同按导出类型定义中相同的次序指定该对象的所有成分。例如，对于下面的定义：

```
TYPE PERSON
  CHARACTER (LEN=10) :: NAME
  REAL :: AGE
  INTEGER :: PHONE_NUMBER
  LOGICAL :: MARRIAGE_STATUS
END TYPE PERSON
TYPE (PERSON) :: HE
```

下面的语句：

```
READ "(A,F4.1,I7,L1)", HE
```

与语句：

```
READ "(A,F4.1,I7,L1)", HE%NAME, HE%AGE,&
   HE%PHONE_NUMBER, HE%MARRIAGE_STATUS
```

具有相同的输入效果。

但若是无格式输入/输出(参见第五节)，则一个导出类型对象的输出值，被认为是一种依赖于处理系统的单个值，例如，对上面相同的类型定义，下面的语句：

```
WRITE (UNIT=7,REC=N) HE
```

与语句

```
WRITE (UNIT=7,REC=N) HE%NAME, HE%AGE, &
   HE%PHONE_NUMBER, HE%MARRIAGE_STATUS
```

可以具有不同的输出效果。

(4)若输入/输出项为隐 DO，其形式为：

(输入/输出项〔,输入/输出项〕...,DO 变量=标量数值表达式，&
　　标量数值表达式〔,标量数值表达式〕)

其中，输入/输出项本身还可以是隐 DO。即：允许隐 DO 的嵌

套；等号右边的标量数值表达式分别用作控制循环的初值，终值及增量；DO 变量及标量数值表达式可以是整型、默认实型或双精度型。但是使用后两种类型均为过时的功能。

隐 DO 的执行，与 DO 构造一样（参见第七章）。例如，下面的语句：

```
INTEGER, PARAMETER :: M=5, N=10
REAL,DIMENSION (M,N) :: A
  ...
PRINT *, ((A(I,J),I=1,M),J=1,N)
```

将在默认的设备上若干行内印出 50 个数据。

(5)若输入项是指针，则它必须当前与一个可定义的目标相结合，数据将从文件传送到该被结合的目标；若输出项是指针，则它必须与当前一个目标相结合，数据将从该被结合的目标传送到文件中，例如：

```
REAL,DIMENSION(:),POINT::P
REAL,DIMENSION(100),TARGET::A
  ...
P=>A
WRITE(UNIT=6,FMT="(1X,10F8.2)")P
```

此程序的执行，将在部件 6 上输出数组 A 中的 100 个数据，每行 10 个数据.

五、小　　结

表控格式输入/输出是使用最简便的格式输入/输出。但它不能为输入/输出提供丰富的数据形式，主要用于那些不需要严格的数据格式的地方。比如，用于程序的调试或少量数据的输入/输出等。

本节还详细描述了输入/输出项表，其作用是指定要输入或输出的数据对象，是数据传送输入/输出语句中的基本组成部分。

在第三节中将介绍显式的格式输入/输出，它为程序提供了描述各种数据的形式的手段，从而能以丰富的数据格式进行输入或

输出。

第三节　顺序格式输入/输出

上节介绍的表控格式输入/输出虽使用简单，但用户几乎不具有控制外部数据形式的能力。它是一种隐式的格式输入/输出。

本节介绍的格式输入/输出，语句中总是要指定一个格式说明，使得在输入/输出语句中能描述每一个要输入或输出数据的外部形式，从而使得用户程序可以具有很强的控制外部数据形式及记录格式的能力。也称显式格式输入/输出。

格式输入/输出可以是顺序访问方式的，也可以是直接访问方式的。本节给出了顺序格式输入/输出的语句形式，顺序格式输入/输出是 Fortran 语言的传统长处之一。　本节的重点是介绍描述外部数据形式的格式说明及各种编辑描述符，以及介绍输入/输出项与格式说明的相互作用。

一、顺序格式输入/输出的语句形式

顺序格式输入/输出的语句形式为：

```
READ(〔UNIT=〕输入部件 &
,〔FMT=〕格式 &
〔,IOSTAT=默认整型标量变量〕&
〔,ERR=标号〕&
〔,END=标号〕&
〔,ADVANCE="YES"〕&
)〔输入项表〕
```

或者

```
READ 格式〔,输入项表〕
```

以及

```
WRITE(〔UNIT=〕输出部件 &
,〔FMT=〕格式 &
〔,IOSTAT=默认整型标量变量〕&
```

```
〔,ERR=标号〕&
〔,ADVANCE="YES"〕&
)〔输出项表〕
```

或者

```
PRINT 格式〔,输出项表〕
```

其中,输入/输出项表已在第二节介绍。现将两个必须的控制说明符解释如下:

〔UNIT=〕输入/输出部件

参见第二节的描述。

〔FMT=〕格式

在显式的格式输入/输出中,FMT=格式说明符中的格式处及简缩形式输入/输出语句中的格式处必须给出一个格式说明。对于顺序格式输入/输出,有两种给出格式说明的方法:

(1)以默认字符型表达式直接给出格式说明。当上述语句中的格式处是默认字符型表达式时,它用来直接给出格式说明。下面是这类语句的例子:

```
CHARACTER(LEN=8),PARAMETER::FMT100=&
    "(3F10.5)",FMT="3F10.5"
  ...
READ(UNIT=5,FMT="(3F10.5)")X,Y
READ FMT100,A,B,C
WRITE(6,"("//FMT//")")(X(I),I=1,30)
```

其中,第一个 READ 语句以默认字符型字面常量指定格式说明;第二个 READ 语句以默认字符型有名常量指定格式说明;而第三个 WRITE 语句则以默认字符型表达式指定格式说明,它们用到的格式说明均为(3F10.5)。

(2)由 FORMAT 语句给出格式说明。当上述语句中能出现格式的地方出现 FORMAT 语句的标号时,则由该 FORMAT 语句给出格式说明。例如:

```
READ(UNIT=5,FMT=800)A,B,C
READ 800,A,B,C
```

```
      WRITE(UNIT=6,FMT=800)(X(I),I=1,30)
      PRINT 800,A,B,C
800   FORMAT(3F10.5)
```

这四个语句均以格式说明(3F10.5)控制输入/输出的格式。

其它任选的说明符 IOSTAT=、ERR=、END=及 ADVANCE=等请参见第四节。

二、格式说明及 FORMAT 语句

在格式输入/输出语句中,格式说明用来由程序显式地指定外部数据的形式及记录格式。格式说明可以在输入/输出语句中直接给出;也可以通过 FORMAT 语句给出。后一方式是 Fortran 的传统方式。

1. 格式说明

格式说明的一般形式为:

([格式项[,格式项]...])

格式项可以是如下之一:

(1)[r]数据编辑描述符(如 I,F 等)

(2)控制编辑描述符(如 nX,/,S,SP 等)

(3)字符串编辑描述符(如"String")

(4)[r]([格式项[,格式项]...])

其中:r 是重复说明,表示其后的格式项将重复使用 r 次,r 是具有正值的整字面常量。下面是格式说明的一些例子:

```
( )
(1X,I5,F10.2,S,G10.5)
(1X,I10,/,"DATES:□",2(2I3,I2))
```

其中,I5、F10.2、I10、I3、I2 等为数据编辑描述符;1X 和/等为控制编辑描述符;"DATES:□"是字符串编辑描述符。

2. FORMAT 语句

FORMAT 语句的形式为:

标号　　FORMAT　　格式说明

格式语句的例子是：

```
100 FORMAT( )
200 FORMAT(I5,F10.2)
300 FORMAT(1X,I10/"DATES:□",2(2I3,I2))
```

前面已经给出了足够多的格式输入/输出语句与格式语句配合使用的例子。

FORMAT 语句是一个特殊的语句，它是不可执行语句。它只是在执行相关的格式输入/输出语句时，为每个数据的转换提供编辑信息。因而，不管它出现在一个程序单元的什么地方，都可以把他看成是相关输入/输出语句的一个有关部份。

三、格式记录印刷输出时的纵向走纸控制

Fortran 语言的格式输出语句，原本是为在行式打印机上印刷输出数据而设计的，规定在打印机上印刷输出一个格式记录(一行)时，该记录中的第一个字符不是被印刷内容的一部份，而是在该记录被印出之前，用来控制纵向进纸的一个控制字符。而该记录中余下的字符，则在走纸之后，再从当前行的左部边界开始印刷。

控制纵向走纸字符的控制功能如下：

格式记录中的第一个字符	在印刷一行之前的纵向走纸控制
空格	印刷之前走纸一行(正常行间距)
0	印刷之前走纸两行(空一行)
1	印刷之前走纸至下页第一行处(换页)
+	印刷之前不走行(用于重影印刷)
其它字符	处理系统可能定义的其它功能

上述重影印刷功能“+”号能产生许多特殊印刷效果，印出发音符号是其典型的应用之一。例如，若默认字符集中包含有汉语拼音的四声符号，则下述语句：

```
PRINT"(1X,'bei jing'/'+','□ˇ □□□¯□□')"
```

将在行式打印机上印出：

bĕi jīng

进一步的注释：

(1)纵向走纸控制字符必须是默认字符。例如：

PRINT"('□ABC')"

PRINT "(1X,2 _'ABC')"

都是允许的，而

PRINT"(2 _'□ABC')"

则是不允许的。

(2)为了保证纵向走纸控制字符正确地产生，应该显式地安排记录中的第一个字符。例如，在一个记录的开始处，利用'□'，T2或者1X等编辑描述符显式地产生一个空格□，用以控制纵向走纸。否则，可能产生下例中的差错：

PRINT"(I5)",12345

此语句的执行，输出记录中的第一个字符1被解释为"印刷之前走纸至下一页第一行处"。于是在下一页开始的一行印出：

2345

较合适的写法应是：

PRINT"(1X,I5)",12345

(3)上述描述可以看出，向外部(磁盘、磁带等)存储文件写出(WRITING)数据与向行式打印机印刷(PRINTING)数据，概念上有微小差异。但是，在Fortran语言中的WRITE语句及PRINT语句，只是语句形式不同，WRITE语句并不隐含一定要向存储文件写出，PRINT语句也不隐含一定要在打印机上印出，它们执行的输出过程完全相同。

(4)有某些输出装置(如终端显示装置等)不一定按本节的描述解释记录中的第一个字符；另一方面，有些外部存储文件也按本节的描述解释记录中的第一个字符。

四、编辑描述符

上面已经介绍了格式说明的形式，这里详细介绍用来组成格

式说明的各种编辑描述符。

表 8—1

分类		编辑描述符的形式	功能	注释
数据编辑描述符	整型编辑	Iw〔.m〕 Bw〔.m〕 Ow〔·m〕 Zw〔.m〕	十进制整型数据编辑 二进制数据编辑 八进制数据编辑 十六进制数据编辑	
	实型及复型编辑	Fw.d Fw.d〔Ee〕 Dw.d ENw.d〔Ee〕 ESw.d〔Ee〕	浮点形式数据编辑 指数形式数据编辑 指数形式数据编辑 工程形式数据编辑 科学形式数据编辑	
		复型编辑可以使用上述各种编辑描述符		
	逻辑型编辑	Lw	逻辑型数据编辑	
	字符型编辑	A〔w〕	字符型数据编辑	
	通用编辑	Gw.d〔Ee〕	可用于上述除 B,O,Z 以外的各种类型数据的编辑	
控制编辑描述符	位置编辑	Tn TLn TRn nX	跳至记录内第 n 个位置 左跳 n 个位置 右跳 n 个位置 右跳 n 个位置	
	斜线编辑	〔r〕/	结束上一记录开始下一记录(产生〔r〕个新行)	
	冒号编辑	:	阻止多余的字符串编辑描述符的输出	
	正负号编辑	S SP SS	恢复由处理系统决定是否输出任选的正号 强制输出任选的正号 阻止输出任选的正号	
	P 编辑	kP	设置比例因子,影响后对实型编辑的解释	设置一种状态
	空格解释编辑	BN BZ	输入记录中空格被解释为 NULL 值 输入记录中空格被解释为零值	设置一种状态 设置一种状态
字符串编辑描述符	字符常量编辑	"[字符]…" '[字符]…'	直接输出字符串	
	H 编辑	cH 字符[字符]…	直接输出字符串	

注释:表中的 w 是字段宽度,m 为至少输出的位数(高位补

零),d 是小数部份位数或输出位数(对 G 编辑输出),e 是指数部份位数,n 是记录中文件位置(字符位置),r 是重复次数,k 是比例因子,c 是字符个数。它们均必须为默认整型字面常量,且仅 k 可以有负值。

Fortran 为输入/输出提供的丰富的格式并不是每一个程序都要用到的,因此建议初学者先只浏览本节,然后再有选择地作深入了解。

1. 一般说明

编辑描述符用来控制数据的编辑,以及用来控制记录的格式。表 8—1 分类列出了全部编辑描述符以及他们的形式及功能。其中的大写字母是该编辑描述符的本身,小写字母的含意请见表后的注释。

2. 整型编辑

整型编辑描述符用于整型数据的格式输入及输出,二进制数据、八进制数据及十六进制数据的输入/输出,也归属于整型数据的输入/输出。这些编辑描述符是:

Iw〔.m〕 I(integer)编辑描述符

Bw〔.m〕 B(binary)编辑描述符

Ow〔.m〕 O(octet)编辑描述符

Zw〔.m〕 Z(hexadecimal)编辑描述符

此外,通用(Generalized)编辑描述符:

Gw.d〔Ee〕

也能用于十进制整型数据的格式输入/输出。

上述描述符中,w 是记录中字段的宽度。m 是至少输出的位数。在输入时,m 没有作用。

I 编辑的数据以数字 0 至 9 表示;B 编辑的数据以数字 0 及 1 表示;O 编辑的数据以数字 0 至 7 表示;Z 编辑的数据以数字 0 至 9 及 A 至 F(或 a 至 f)表示。

G 编辑的作用等同于 I 编辑,且 d 及 e 部分不起作用。

(1)I 编辑输入。对于 Iw〔.m〕编辑描述符,其输入字段由 w 个

字符组成。字段中表示整数本身的字符串，除了对空格字符□有专门的解释外，必须具有整字面常量形式，且可以任选的带有正号。示例如下：

编辑描述符	输入字段中数据形式	输入后的值	注释
I5	□□□24	24	
I5.3	□□+24	24	输入时，m 没有作用。
I5	□□-24	-24	
I5	□□13□	13	非前导□对输入值无影响（当 BN 有效时）。
I5	□□13□	130	非前导的□被解释为 0（当 BZ 有效时）。
I5	□-□13	-13	数字以前的前导□忽略不计。
I5	325_2	325	种别为 2 的整常数。
G10.2	□□□□□□□□24	24	d=2 无作用。

例如：

```
INTEGER(KIND=2)::K
  …
READ(UNIT=5,FMT="(I4,I5.3,I5)")I,J,K
```

若输入记录为：

□□24□□□24325_2

则输入结果分别为：I=24、J=24 及 K=325（种别为 2 的整型）。

注意，在不同的处理系统上，上例中种别为 2 的整型由多少个二进制位（比如 32 位）表示，可以是不一样的。

进一步注释：

a. 输入字段中，正号可以省略。

b. 输入时，第一个数字之前的前导空格，总是被忽略不计；除了以 BZ 编辑描述符专门指明了输入字段中的第一个数字之后的非前导空格应解释为数 0 这一情况（参见 OPEN 语句中的 BLANK=说明符以及下面要介绍的 BN 及 BZ 编辑描述符）以外，输入字段中的非前导空格均被忽略不计。

c. 全部由空格字符所组成的字段，被认为具有整型值。

d. 字段中只允许出现空格、正负号、数字及下划线这几类字符。

e. 输入数据的种别参数，必须是字面常量的形式。例如：

3652 _NKIND

是不允许的。

(2)B、O、Z 编辑输入。对于 Bw〔.m〕，Ow〔.m〕及 Zw〔.m〕编辑描述符，输入字段中的字符串，必须分别由二进制数字、八进制数字及十六进制数字所组成。例如：

编辑描述符	输入字段中数据形式	输入后的值	注释
B10	0001110101	117	
O4	□165	117	
Z3	□75	117	
Z3.2	□75	117	输入时，m 不起作用。

例如：

READ(UNIT=5,FMT="(B10,O4,Z3)")I,J,K

当输入记录为：

0001110101□165□75

时 I,J 及 K 中的值均为 117。

(3)I 编辑输出。以 Iw〔.m〕编辑描述符输出的数据，字段宽度为 w，且至少输出 m 位数字。其组成为：开始是若干个前导的空格；后随以一个任选的正负号；再后随以该数据的无符号的值。示例如下：

编辑描述符	内部值	输出字段串的形式	注释
I5	25	□□□25	
I5.4	25	□0025	m=4，填充两个前导的 0。
I5	−25	□□−25	
I5.3	−25	□−025	m=3，填充一个前导的 0。
I5	12345	12345	
I5	0	□□□□0	默认的 m=1，外部字段中输出一个 0。

I5.0	0	□□□□□	m=0,且输出值=0,外部字段中不输出0。
G10.2E4	25	□□□□□□□□25	d=2,e=4,无作用。

例如:

PRINT"(1X,I5,I5.4,I5)",+25,+25,+0

将输出:

□□□25□025□□□□0

进一步注释:

a. 若省略m,则至少输出一位数字。

b. 至少输出位数m不得大于字段宽度w。

c. 如果输出数据的位数不足m位,则在输出数据前填充适当个数的0,补充到共m位。

d. 只有m为0且输出值亦为0时,输出字段才全为空格。

e. 是否输出数据中任选的正号,是依赖于处理系统的,但下面要介绍的SP及SS编辑描述符,为程序提供了控制是否输出此任选正号的手段。

(4)B、O、Z编辑输出。Bw〔.m〕、Ow〔.m〕及Zw〔.m〕编辑描述符分别导致在输出字段中产生二进制数据、八进制数据及十六进制数据。示例如下:

编辑描述符	内部值	输出字段中的形式	注释
B10	117	□□□1110101	
B10.4	117	□□□1110101	输出数字多于4位,m无作用。
O6	117	□□□165	
O6.4	117	□□0165	m=4,填充一个前导0。
Z4	117	□□75	
Z4.4	117	0075	w=m=4,填充前导0。

例如:

PRINT"(1X,B10.4,O6.4,Z4.4)",+117,+117,117

输出结果将为:

□□□1110101□□01650075

3. 实型编辑

实型编辑描述符用于实型数据的格式输入及输出。允许的实型数据的形式包括浮点数形式(F 编辑)、指数形式(E 编辑)、工程数据形式(EN 编辑)及科学数据形式(ES 编辑),下面将分别介绍。

(1)F 编辑。F 编辑描述符用于实型数据(或浮点数 floating point)的输入/输出,其形式为:

Fw.d

其中,w 是外部字段的宽度,d 是小数点后的位数。对于输出,F 编辑比较适合用于输出其量值在 0.01 至 1000 范围内的数据。

a. F 编辑输入。以 Fw.d 编辑描述符输入的字段,由 w 个字符组成。其基本形式是:一个任选的正负号;后随由若干个数字组成的数字串,其中包括被解释为 0 的那些空格(参见 BN 及 BZ 编辑),此数字串中还可任选的包含一个小数点。若有此小数点,则 d 不起作用;若无此小数点,则小数点的位置被假定在此数字串右起第 d 位数字之前,且必要时认为有前导的 0。

除上述基本形式外,还可后随下述形式之一的指数部分:

· 带正负号的整常数;

· 以 E 开始,后随 0 个或多个空格,再后随带任选正负号的整字面常量;

· 以 D 开始,后随 0 个或多个空格,再后随带任选正负号的整字面常量。上述 E 形式的指数和 D 形式的指数具有相同的效果。输入数据示例如下:

编辑描述符	输入字段中数据的形式	输入的内部值	注　释
F6.3	123456	123.456[注]	小数点位置在数字串右起第三位数字之前。
F6.3	123.45	123.45[注]	d(=3)不起作用。
F6.3	+2□4□□	20.400[注]	数字串中的空格被解释为 0(当 BZ 有效时)。
F6.3	□-12.3	-12.3[注]	d 不起作用。

F6.3	□□□□□1	0.001[注]	前导的空格被解释为前导的 0。
F6.3	1234−2	0.01234	$1.234*10^{-2}$
F6.3	12.3E3	12300.	$12.3*10^{3}$,d 不起作用。
F6.2	123D−2	0.0123	$1.23*10^{-2}$

注:这里是没有设置比例因子时的输入值,否则此内部值还可受当前起作用的比例因子 k 的控制。

进一步注释:

在一个格式说明中,可以通过 kP 编辑描述符设置比例因子的值 k,当外部字段中有指数部分时,比例因子不起作用。而当没有指数部分时,输入值受当前起作用的比例因子 k 的影响。且输入之值为输入字段中基本形式值乘以 10^{-k}。详见下面对 kP 编辑的介绍。

b.F 编辑输出。以 Fw.d 编辑描述符输出的数据在输出字段上占 w 个字符:以若干个前导的空格(若该字段足够宽)开始;后随一个负号(若内部值为负)或一个任选的正号;再后随一个表示该内部值的量值的数字串,且该数字串是经舍入的,串中包括一个小数点、小数部分有 d 位数字,以及输出值还要受到当前起作用的比例因子 k 的影响。示例如下:

编辑描述符	内部值	输出字段中的形式[注]	注　释
F7.2	−123.454	−123.45	舍去。
F9.4	123.454	□123.4540	
F6.1	−123.454	−123.5	舍入。
F5.0	−123.454	−123.	d=0,无小数部分。
F9.2	−123.454	□□−123.45	w 足够大,补前导的空格。
F7.2	123.454	□123.45	正号不印出。
F6.2	0.454	□□0.45	该数的量值小于 1,小数点前印一个 0。
F6.2	123.45	******	w 位数不足,全部填充*。

注:这里是没有设置比例因子时的输出。否则,输出值还受当前起作用的比例因子 k 的控制。

进一步注释：

①d 不得大于 w。

②d 可以为 0，此时也印出小数点，而在小数点后不输出数字。

③除了当输出的数值小于 1 时，小数点的左边可以有一个 0（依赖于处理系统）以外，其它情况均无前导 0。

④是否输出任选的正号，是依赖于处理系统的（参见下面介绍的正负号编辑描述符 S，SP 及 SS）。

(2)E 编辑及 D 编辑。E 编辑描述符及 D 编辑描述符用于带有指数（Exponent）部分的实型数据的输入及输出，其形式为：

Ew.d[Ee]

Dw.d

其中，w 是字段宽度，d 是小数部分位数，而 e 则用来指定指数部分的位数。

与 F 编辑比较，就输出而言，Ew.d 及 Dw.d 编辑描述符比较适合应用于输出其量值小于 0.01 或者大于 1000 这样的数据。而 Ew.d Ee 则适合于输出其量值极大或极小的数据。

a. E 及 D 编辑输入。E 及 D 编辑描述符用于输入时，其效果与 F 编辑描述符完全相同。

b. E 编辑及 D 编辑输出。当标量因子为零（参见 kP 编辑描述符）时，以 Ew.d、Ew.dEe 及 Dw.d 编辑描述符输出的字段中数据的一般形式如下：

输出字段一般形式	内部值的指数范围	使用的编辑描述符
□…□[±][0]. $x_1x_2\cdots x_d$ E±z_1z_2	|指数|<=99	Ew.d 或 Dw.d
□…□[±][0]. $x_1x_2\cdots x_d$ ±0z_1z_2	|指数|<=99	Ew.d 或 Dw.d
□…□[±][0]. $x_1x_2\cdots x_d$ D±z_1z_2	|指数|<=99	Dw.d
或为：		
□…□[±][0]. $x_1x_2\cdots x_d$ ±$z_1z_2z_3$	99<|指数|<=999	Ew.d 或 Dw.d
或为：		
□…□[±][0]. $x_1x_2\cdots x_d$ E±$z_1z_2\cdots z_e$	|指数|<=10^{e-1}	Ew.dEe

其中，$x_1x_2\cdots x_d$ 是数据在舍入后各位的数字，且每个 z_i 都是一

位数字。数据中任选的正号以及小数点前面的零是否产生，是依赖于处理系统的。而指数部分的正负号则一定要产生。即使指数为零，正号也产生。输出形式中指示指数部分的 E 或 D 的使用常是依赖于处理系统的。示例如下：

编辑描述符	内部值	输出字段中的形式注	注释
E12.5	1234567.8	□□.12346E+07	某些处理系统，小数点前产生一个0。
E12.5	−12.34	□−.12340E+02	
E12.5	−0.000475	□−.47500E−03	
E12.5	1.23456E12	□□.12346E+13	
E12.5	1.234E−150	□□.12340−149	99<|指数|<=999
E12.4E4	1.234E1234	□.1234E+1235	|指数|>999
D12.5	1234567.8	□□.12346D+07	某些系统产生与 E12.5 相同的结果。

注：这里是没有设置比例因子时的输出。否则，输出形式还受当前起作用的比例因子 k 的控制。例如：

PRINT″(1X,2E12.5)″,1.23456E12,1.234E−150

产生的输出结果为：

□□.12346E+13□□.12340−149

而另一个系统可能产生如下结果：

□0.12346+013□0.12340−149

进一步注释：

①对于输出数据的形式中那些依赖于处理系统的功能（例如，小数点前是否印出一个零），语言本身没有提供控制手段。

②D 编辑原是为双精度实型数据的输入/输出编辑而设计的。

(3)EN 编辑。EN 编辑描述符用于以工程 (engineering) 表数法的形式输入/输出实型数据。其形式是：

ENw.d〔Ee〕

其中，w 是字段宽度，d 是小数部分的位数，e 是指数部分的宽度。

a. EN 编辑输入。EN 编辑输入，其效果与 F 编辑完全相同。

b. EN 编辑输出。由 EN 编辑描述符输出的实型数据的组成为：指数部分的值必须能被3除尽；且舍入后，其小数点左边整数部分的绝对值大于等于1且小于等于999(输出值为零时除外)。其外部字段中数据的一般形式如下表：

输出字段一般形式	内部值的指数范围	使用的编辑描述符
□…□〔±〕$y_1y_2y_3.x_1x_2\cdots x_dE\pm z_1z_2$	|指数|<=99	ENw.d
□…□〔±〕$y_1y_2y_3.x_1x_2\cdots x_d\pm 0z_1z_2$	|指数|<=99	ENw.d
□…□〔±〕$y_1y_2y_3.x_1x_2\cdots x_d\pm z_1z_2z_3$	99<|指数|<=999	ENw.d
□…□〔±〕$y_1y_2y_3.x_1x_2\cdots x_d\pm z_1z_2\cdots z_e$	|指数|<=10^{e-1}	ENw.d〔Ee〕

其中，$y_1y_2y_3$是输出值的最高有效位的第1个至第3个十进制数字位。$x_1x_2\cdots x_d$ 是该值的其后 d 位。每个 z_i 都是指数部分的一位数字。

下面是 EN 输出的一些示例(其时使用了 SS 编辑描述符，即不输出数据中任选的正号而代之以输出空格)：

编辑描述符	内部值	输出字段中的形式*注	注　释
EN12.3	6.421	□□□6.421E+00	
EN12.3	-.5	-500.000E-03	
EN12.3	.00217	□□□2.170E-03	
EN12.3	0.0	□□□0.000E+00	值为零。
EN12.3	4721.3	□□□4.721E+03	
EN12.3	4721.3	□□□4.721+003	在另一处理系统中，可能的输出。
EN12.3	0.32456E125	□□32.456+123	|指数|>99
EN12.3E4	0.32456E1234	□□3.246+1233	|指数|>999，必须使用 Ew.dEe 形式。

进一步注释：

①对于 EN 编辑输出，比例因子 k 不起作用。

②当指数部分的绝对值大于999时，必须使用 ENw.dEe 形式的编辑描述符。

(4)ES 编辑。ES 编辑描述符用于以科学(Scientific)表数法的

形式输入/输出实型数据。其形式是：

ESw.d〔Ee〕

其中，w 是字段宽度，d 是小数部分的位数，e 是指数部分的位数。

a. ES 编辑输入。ES 编辑输入，其效果与 F 编辑完全相同。

b. ES 编辑输出。由 ES 编辑描述符输出的实型数据的组成是：经过舍入后的数值的最高有效位，位于小数点的左边。其外部字段中数据的一般形式如下：

输出字段一般形式	内部值的指数范围	使用的编辑描述符
□…□〔±〕$y.x_1x_2\cdots x_dE\pm z_1z_2$	|指数|<=99	ESw.d
□…□〔±〕$y.x_1x_2\cdots x_d\pm 0z_1z_2$	|指数|<=99	ESw.d
□…□〔±〕$y.x_1x_2\cdots x_d\pm z_1z_2z_3$	99<|指数|<=999	ESw.d
□…□〔±〕$y.x_1x_2\cdots x_d\pm z_1z_2\cdots z_e$	|指数|<=10^{e-1}	ESw.dEe

其中，y 是一位最高有效位，$x_1\cdots x_d$ 是小数部分，每个 z_i 都是指数部分的一位数字。

下面是 ES 输出的一些示例(其时，使用了 SS 编辑描述符，即不输出数据中任选的正号，而代之以输出空格)：

编辑描述符	内部值	输出字段中的形式注	注　释
ES12.3	6.421	□□□6.421E+00	
ES12.3	−0.5	□□−5.000E−01	
ES12.3	0.00217	□□□2.170E−03	
ES12.3	0.0	□□□0.000E+00	值为零。
ES12.3	4721.3	□□□4.721E+03	
ES12.3	0.32456E123	□□□3.246+122	|指数|>99
ES12.3E4	0.32456E1234	□3.426E+1233	|指数|>999必须用 ESw.dEe 形式。

进一步注释：

对于 ES 编辑输出，比例因子 k 不起作用。

4. 复型编辑

一个复型数据由一对实型数据组成，构成该复型数据的实部

与虚部。编辑一个复型数据需要两个实型编辑描述符(即 F、E、EN、ES、G 或 D 等),其第一个指定实部的编辑,其第二个指定虚部的编辑。这两个编辑描述符可以不同。控制编辑描述符(如 Tn、/、:、kP 等等)及字符串编辑描述符(如'ABC'等)可以出现在这两个编辑描述符之间。例如:

```
COMPLEX CM(2)
READ (5,"(4E7.2)")(CM(I),I=1,2)
PRINT"(1X,F7.2,A,F7.2,A,A,F7.2,A,F7.2)",&
    REAL(CM(1)),"+",AIMAG((CM(1)),"I";&
    "real PART=",REAL(CM(2)),"imaginary PART=",&
    AIMAG(CM(2))
…
```

且若输入记录为:

□□55511□□□2146□□□□100□□□□621

则赋给 CM(1)及 CM(2)的值分别为555.11+21.46i 及1.0+6.21i。且输出结果应是:

□555.11□+□□□21.46□I□□real□PART=□□□1.00

□imaginary□PART=□□□6.21

5. 通用编辑

通用的 G 编辑描述符可以用于包括整型、实型、复型、逻辑型及字符型等任何类型数据的输入及输出。其形式为:

Gw.d[Ee]

其中,w 是外部字段的宽度,d 是小数部分的位数,而 e 是指数部分的位数。

当 G 编辑描述符用于整型、逻辑型或字符型数据的输入/输出时,d 及 e 没有作用。在这三种应用情况中,对 G 编辑描述符的解释,分别在相应各节中作介绍。

当 G 编辑描述符用于实型数据的输入/输出时,其含义解释如下:

(1)G 编辑的实型数据输入。G 编辑描述符用于实型数据输入,其效果与 F 编辑描述符完全相同。

(2) G 编辑的实型数据输出。当 G 编辑描述符用于输出时，d 不再如其它所有的编辑描述符那样是表示数据外部形式中小数部分的位数，而是表示外部数据中有效数字的位数。

首先，假设输出的数据经 d 位的精度舍入后，其量值为 N。此时，若输出的量值 N 满足：

$0.1 <= N < 10^d$

则相当于以适当的 F 编辑输出该数据，该数据由 d 位数字组成，且其后跟随以4个（当使用 Gw.d 编辑描述符时）或 e+2个（当使用 Gw.dEe 编辑描述符时）空格。以此形式输出时，比例因子不起作用。示例如下：

编辑描述符	内部值	输出字段中的形式	注　释
G13.5	22.38631	□□□22.386□□□□	按适当的 F 编辑数据形式输出。
G13.5	−4367.24	□□−4367.2□□□□	按适当的 F 编辑数据形式输出。
G13.5	0.0	□□□0.0000□□□□	输出值为零时，小数点后占 d−1位。
G13.5	−0.123896	□−0.12390□□□□	输出值小于1时，小数点前有一个零。

注意：当内部值为零时，小数点后占 d−1位；当$0.1 <= N < 1.0$时，小数点前有一个零。否则，若输出的量值 N 满足：

$0 < N < 0.1$或者 $N > 10^d$

则按 d 位有效数字输出该数，且后随指数部分。就如同分别按 kPEw.d 编辑输出（当使用 Gw.d 编辑描述符时），或按 kPEw.dEe 输出（当使用 Gw.dEe 编辑描述符时）一样。其中，k 是当前起作用的比例因子（参见 P 编辑描述符）。示例如下：

编辑描述符	内部值	输出字段中的形式	注　释
G13.5	−0.0354	□−0.35400E−01	0<N<0.1按 E13.5
G13.5	3254.364	□□0.32544E+04	
0P,G15.5E4	−0.0003456	□−0.34560E−0003	
2P,G15.5E4	−0.0003456	□□−34.560E−0005	当比例因子 k=2时。

进一步注释：

对于G编辑输出，必须满足：

w>=d+7　　(对Gw.d)

或者

w>=d+e+5　　(对Gw.dEe)。

6.P编辑及比例因子

在输入/输出过程中，P编辑描述符用来把比例因子的当前值设置为k，而此比例因子的当前值，能影响其后以F、E、EN、ES、D及G等数值编辑描述符对实型数据进行的输入/输出编辑。比如，能按一定的比例修正输入数据的值，或者影响输出数据的外部形式等。其形式如下：

kP

其中，比例因子k为默认的整型字面常量。

比例因子对输入/输出的影响，与所使用的编辑描述符、数据的传送方向以及数据的外部形式等有关。

(1)比例因子对输入的影响

对于F、E、EN、ES、D及G编辑描述符，若外部输入字段的数据中不存在指数部分，则已输入的内部数据的值将等于外部表示的数据值除以10^k；若外部输入字段中有指数部分，则比例因子不起作用。示例如下：

编辑描述符	输入字段中数据的形式	输入后的值	注释
2P,F4.2	−678	−0.0678	输入值6.78除以10^2。
−3P,G8.4	□1234.56	1234560.	输入值1234.56除以10^{-3}，对于输入，G与F效果一样。
2P,F8.2	□12.3E−3	$12.3*10^{-3}$	比例因子不起作用。
0P,F8.3	□□□45678	45.678	将比例因子恢复为零。

例如：

```
READ"(4P,F6.2,F8.4,0P,F6.2)",X,Y,Z
```

若外部记录为：

□□2345□□123456□□2345

则输入结果分别为：

X＝0.002345、Y＝0.00123456及Z＝23.45

比例因子适宜用于量值比较大及比较小的数据的输入。

(2)比例因子对输出的影响，与所使用的编辑描述符有关。

a. 对F编辑的影响。若用F编辑描述符输出，则数据的外部表示将等于内部数据乘以10^k。示例如下：

编辑描述符	内部值	输出字段中的形式	注　释
3P,F7.2	0.0452	□□45.20	输出值乘以 10^3。
−5P,F7.2	123400.	□□□1.23	输出值乘以 10^{-5}。
0P,F7.4	5.876331	□5.8763	将比例因子恢复为零。

b. 对E、D编辑的影响。对于用E、D编辑描述符输出，在数据的输出形式中(其中含有一个指数部分)，有效数部分乘以10^k，而指数部分减去k。即，比例因子只改变输出数据的形式，主要是调整小数点的位置，而不改变输出数据的值本身。示例如下：

编辑描述符	内部值	输出字段中的形式	注　释
3P,E11.2	53.2735	□532.74E−01	
−2P,E11.4	282.1124	□0.0028E+05	
0P,E11.3	0.005354	□□0.535E−02	将比例因子恢复为零。

c. 对G编辑的影响。若该数据的值在允许使用F编辑输出的范围内，则比例因子暂时对此编辑描述符不起作用；若该数据的输出值范围需要使用E编辑，则比例因子的作用与使用E编辑时一样。即，在使用G编辑时，比例因子总是不改变输出数据的值本身，只改变输出数据的形式。

d. 对EN及ES编辑的影响。比例因子对EN及ES编辑输出不起作用。

注释：在执行每一个输入/输出语句的开始时，比例因子的值为零；格式控制的翻回本身不会影响比例因子；比例因子与其后的F、E、D、EN、ES及G等编辑描述符之间的逗号可以省略不写，例如：

3P,3F10.2

可写为：

3P3F10.2

但建议不要这样使用，因为这样容易被误解为此比例因子只对该编辑描述符起作用。

7.逻辑型编辑(L 编辑)

逻辑型编辑描述符用于逻辑型数据的输入及输出，其形式是：

Lw

此外，通用编辑描述符

Gw.d〔Ee〕

也能用于逻辑型数据的输入/输出。上述描述符中，w 是字段宽度。d 及 e 不起作用。

(1)L 及 G 编辑用于逻辑型值的输入。输入字段为 w，其输入数据的组成为：开始是任选的若干个空格；后随任选的一个小数点；再后随一个用来表示.TRUE.的 T 或者后随一个用来表示.FALSE.的 F；最后是任选的若干个任意的字符。当处理系统能表示大小写字母时，小写字母与大写字母有相同的效果。示例如下：

编辑描述符	输入字段中数据的形式	输入的内部值	注释
L5	□TRUE	.TRUE.	
L5	FALSE	.FALSE.	
L5	□□□.T	.TRUE.	
L6	.TRUE.	.TRUE.	
L1	T	.TRUE.	
L1	F	.FALSE.	与大写字母同效。
G5.2	□□□□F	.FALSE.	相应的输入项为逻辑型。

(2)L 及 G 编辑用于逻辑型值的输出。输出字段组成为：w－1 个前导的空格；依据相应输出项的值是.TRUE.或.FALSE.而分别后随一个字母 T 或 F。示例如下：

编辑描述符	内部值	输出字段中的形式	注释
L3	.TRUE.	□□T	
L3	.FALSE.	□□F	
L1	.TRUE.	T	

G6.2	.FALSE.	□□□□□F	相应的输出项为逻辑型。

例如：

```
LOGICAL  L1,L2
L1=.TRUE.;L2=.FALSE.
PRINT"(1X,2L5)",L1,L2
```

其产生的输出结果为：

□□□□T□□□□F

8. 重复说明

在前面介绍的数据编辑描述符I、B、O、Z、F、E、EN、ES、G、L、A及D等前面都可以前缀一个重复说明。如：

r Iw[.m]

r Bw[.m]

等。其中，重复说明r用来指明该编辑描述符被重复使用的次数。r必须是默认整型字面常量。例如，下述格式说明：

(3F12.3)

具有与下述格式说明

(F12.3,F12.3,F12.3)

完全相同的效果。

类似的，重复说明还能用在括号中的一组编辑描述符的前面。例如，下面的格式说明：

(1x,2(I5,F8.2))可以展开为等价的：

(1x,I5,F8.2,I5,F8.2)

9. BN编辑及BZ编辑

当以前面介绍的I、B、O、Z、F、E、EN、ES、D及G等数据编辑描述符用作数值输入时，除了前导的空格以外，对于其它位置的非前导空格，既可以忽略不计，也可以解释为零。BN(即Blank as Null)及BZ(即Blank as Zero)编辑描述符就是用来指定对这类非前导空格的解释的。解释如下：

编辑描述符	作用
BN	其后的数值输入字段中的非前导空格忽略不计。

BZ　　其后的数值输入字段中的非前导空格被解释为零。

例如：

READ(UNIT=5,FMT="(BZ;I5,BN,I5)")I,J

若外部记录为：

□□2□4□□2□4

则语句的执行，导致 I=204，而 J=24。

进一步的注释如下：

(1)在一个文件以 OPEN 语句打开时，就指定了对其记录中空格的解释。即，BLANK="ZERO"说明符指定将字段中非前导空格解释为零。而 BLANK="NULL"说明符指定将字段中非前导空格忽略不计。

(2)对于内部文件，就处理成如同该文件是以 BLANK="NULL"说明符打开的。

(3)在一个格式输入语句的执行开始时，空格将按上述默认的解释进行处理，直至遇到格式说明中的 BN 或 BZ 编辑描述符为止。

(4)非前导空格忽略不计的效果，如同将该字段中的非前导空格抽出，余下的字符右移至字段边界，再将抽出的空格用作前导空格。

(5)当一个字段中只有空格时，输入值为零。

10. 正负号(S,SP 及 SS)编辑

在以前介绍的 I、F、E、D、EN 及 ES(还包括 G)等数据编辑描述符输出数值型数据时，数据中任选的正号"+"是否输出，通常是由处理系统决定的。而这里介绍的正负号编辑描述符则为输出语句提供了控制是否输出这些任选正号的能力，解释如下：

编辑描述符	作　用
SP	其后输出的数值型数据中，任选的正号"+"必须产生。
SS	其后输出的数值型数据中，任选的正号"+"不得产生，代之以输出空格。

S　　其后输出的数值型数据中，是否产生任选的正号，恢复到由处理系统决定。

注意，在一个输出语句执行的开始时，是否输出数值字段中的任选的正号，由处理系统决定。例如：

```
PRINT"(1X,SP,F10.2,SS,F10.2)",1.46,234.12
```

产生的输出结果为：

□□□□□+1.46□□□□234.12

11. 字符编辑

字符编辑描述符用于字符型数据的输入及输出，其形式是：

A〔w〕

此外，通用编辑描述符：

Gw.d〔Ee〕

也能用于字符型数据的输入/输出。其作用与 Aw 编辑描述符相同，而 d 及 e 部分不起作用。

字符编辑描述符的外部字段由 w 个字符组成，当不指定字段宽度 w 时，则相应的输入/输出项的字符长度（假定为 len）将被用作字段宽度。

在输入/输出中，每个 Fortran 字符在外部显现媒体上具有相同的显现宽度。即，一个字符在显现媒体上占有一个字符位置，这类字符是一种等宽字符。不少民族字符（例如作为俄文本源的西

表8－2

	Fortran 字符宽度字符	非 Fortran 字符宽度字符
等宽字符	Fortran 字符、ASCII 字符、西里尔文等	汉字（不包括其中的半角字）*
非等宽字符		蒙文、阿拉伯文、汉字（包括其中的半角字）*，UCS** 等

＊指现行国家标准《信息交换用汉字编码字符集基本集 GB　2312－80》；

＊＊UCS－《国际标准 ISO/IEC 10646－1通用字符集》，是包括多种文字的编码字符集的新国际标准。

里尔文字符等）也都是这样。但是，另有许多民族的字符，其显现宽度与 Fortran 字符不同。例如，汉字、蒙文、阿拉伯文（维吾尔文）

等。这些字符是非 Fortran 字符宽度的字符。它们可能是等宽字符，如汉字。也可能是非等宽字符，如蒙文、维吾尔文。

各种字符按其显现宽度分类列于表8－2中。

下面分两种情况介绍：

(1)字符编辑用于 Fortran 字符宽度字符。这类字符的特点是：每个字符在外部显现媒体上占有一个字符位置。因而，一个字段的字段宽度与它在外部显现媒体上所占的字符位置的个数是一致的。

a. A〔w〕及 Gw. d〔Ee〕编辑描述符用于输入。输入时，若字段宽度 w 大于相应输入项的字符长度 len，则读入输入字段中最右边 len 个字符；若 w 小于 len，则将输入字段读入到输入项的左边，且右边填充 len－w 个空格。示例如下：

编辑描述符	输入项字符长度 len	输入字段	输入后的值	注释
A4	4	ABCD	ABCD	
A4	3	ABCD	BCD	读入字段中最右边三个字符。
A4	7	ABCD	ABCD□□□	输入项右边填充三个空格。
G10.2	4	ABCDEFGHIJ	GHIJ	与 A10的效果相同。
A	4	ABCD	ABCD	将输入项字符长度用作 w。

例如：

```
CHARACTER(LEN=4)::CH1,CH2,CH3,CH4
  ...
READ(UNIT=5,FMT="(A3,A,A4,A5)")CH1,CH2,CH3,CH4
```

若输入记录为：

```
ABCDEFGHIJKLMNOPQRST
```

则，字符型变量 CH1中为 ABC□、CH2中为 DEFG、CH3中为 HIJK、CH4为 MNOP。

允许非默认种别字符的输入及输出。但是，在一个 A 或 G 编辑描述的控制之下进行输入或输出的字符，与相对应的数据必须具有相同的种别类型参数。例如，假设西里尔文字符集由处理系统的一个非默认字符类型支持，其种别参数被规定为5，则下面的程序段：

```
CHARACTER(LEN=30,KIND=5)::CRLLC
CHARACTER(LEN=9)LANG
  ...
READ"(A9,A)",LANG,CRLLC
```

在读入下面的记录：

CYRILLIC:БЕЗ НЕЕ НИУТО НЕ ВОЗ МОЖН

时，在 LANG 中的值为：

CYRILLIC:

而在 CRLLC 中则为种别为5的字符串：

БЕЗ НЕЕ НИУТО НЕ ВОЗ МОЖН □□□□□

b. A〔w〕及 Gw. d〔Ee〕编辑描述符用于输出。输出时，若字段宽度 w 大于字符长度 len，则在输出字段的左部填充 w－len 个空格；若 w<len，则输出项中最左边的 len 个字符将被输出。示例如下：

编辑描述符	输出项的字符长度 len	内部值	输出字段中的形式	注　释
A4	4	ABCD	ABCD	
A4	3	ABC	□ABC	左部填充一个空格。
A4	7	ABCDEFG	ABCD	输出最左边的四个字符。
G10.2	4	ABCD	□□□□□□ABCD	与 A10效果一样。
A	4	ABCD	ABCD	将输出表项字符长度用作 w。

例如：

```
CHRACTER (LEN=16)ITEM_NAME
REAL PRICE
  ...
PRINT"(1X,A16,F10.2)",ITEM_NAME,PRICE
```

PRINT 语句在某次执行中输出的结果可能是：

TELEVISION□3052□□□□4230.00

对于这类数据，很容易产生表格式的输出结果，因为第一个字段输出的字符型数据在外部显现媒体上占有确定的字符位置数(此例中为16)。

(2)字符编辑用于非 Fortran 字符宽度的字符。在输入/输出

中，记录的长度、输入/输出字段的宽度(w)都是以字符个数为单位度量的，而不管这是什么种别的字符。于是对于非 Fortran 字符宽度的字符，若要输入或输出的 w 个字符，它们在外部显现媒体上常常不是占有 w 个字符位置。

例如，长度为4、包含值"国际标准"的非默认字符型变量，在A4编辑描述符的控制下，其字段宽度为4，但它在显现媒体上占有8个字符位置。可见，字段宽度是面向字符的概念，而字符位置则是面向显现媒体的概念。

一个汉字占有两个字符位置是由中国国家标准《信息交换用汉字编码字符集 基本集 GB2312—80》规定的。而通常，字形的相对宽度应由国家的字形标准规定。

蒙文是非等宽字符的竖写拼形文字。其字符有几种宽(厚)度。一个蒙文字则由若干个字符连写而成，并被横向放置，以与 Fortran 字符书写习惯一致。它们不可能如拉丁字母那样被简化为等宽字符，如将 I 与 m 处理为等宽。这样，一个蒙文字符串在外部显现媒体上所占的字符位置数，不仅决定于该字符串的字段宽度(字符长度)，而且取决于该字符串是由那些字符组成的。这给带有这类文字的数据的制表输入/输出带来了很大的困难。

a. A〔w〕及 GW. d〔Ee〕编辑描述符用于输入。有关规则与用于 Fortran 字符的规则相同。下面是读入汉字的例子：

```
CHARACTER(KIND=3,LEN=4)HANZI
  ...
READ"(A4)",HANZI
```

若外部记录为：

```
汉字标准
```

则字符型变量 HANZI 中的字符串为"汉字标准"。应注意的是，4个汉字在外部显现媒体上占8个字符位置。

b. A〔w〕及 GW. d〔Ee〕编辑描述符用于输出。有关规则与用于 Fortran 字符的规则相同，下面是输出汉字的例子：

```
RESULT=-72.435
```

```
...
PRINT"(1X,A3,F10.2)",3_"结果＝",RESULT
```

其输出结果为：

```
结果＝□□□□－72.44
```

注意，由于我国现行汉字标准字符集中有一个全角字的等号（与汉字一样宽）及一个半角字的等号（与 Fortran 字符一样宽）。这里的等号应为全角字的等号，否则，若想输出具有表格形式的结果，其问题常会如同非等宽文字一样复杂。

进一步注释：

①从 Fortran 90标准讲，支持所有民族文字的输入/输出。但是，是否支持某种具体的民族文字，取决于处理系统（包括输入/输出设备）。

②Fortran 90语言标准支持蒙文这类非等宽（厚）文字及 UCS 这类非等宽字符集字符的输入/输出。但是，若想输出对齐的表格，标准目前尚未提供有效简便的手段。

③A 编辑及 G 编辑是从记录中读入或向记录写出非默认字符型数据的唯一手段。

④请特别注意，非默认种别字符的显现宽度不一定总是与 Fortran 字符的显现宽度不同。默认种别字符的宽度也不一定总是与 Fortran 字符的显现宽度相同。例如，当 UCS 通用字符集被用作处理系统的字符集时，将出现这种情况。

12. 字符串编辑

字符串编辑是将要输出的字符串本身嵌入到格式说明中，用于直接在输出字段上产生此字符串。与 A 编辑的不同之处是，在输出项表中，它没有对应的输出项。字符串编辑描述符又分为两类，他们是：

（1）字符常量编辑描述符：

'字符串'　　（串中有撇号需重复两次）

"字符串"　　（串中的引号需重复两次）

（2）cH 编辑描述符：

cH 字符串

其中,c 是字符 H 之后构成此字符串的字符个数。上述字符串都必须是默认字符型的。字段的宽度是该字符串中字符的个数。

字符串编辑描述符不能用于输入。表8－3为字符串编辑输出的示例。

表8－3

编辑描述符	输出项	输出字段的形式	注　释
'ABCD'	不需要	ABCD	
"DON'T"	不需要	DON'T	
'DON''T'	不需要	DON'T	
5HABCDE	不需要	ABCDE	需要仔细数字符个数

例如:

PRINT"(1X,'This□is□a□FORMAT□statement')"

或

```
        PRINT 100
100     FORMAT(1X,26HThis□is□a□FORMAT□statement)
```

这两个语句均印出英语句子:

This□is□a□FORMAT□statement

外部字段宽度为 w＝26。

注释:

(1)字符串中字母的大小写是有意义的,空格也是有意义的。

(2)不允许由非默认字符去构成字符串编辑描述符。

(3)H 编辑描述符可读性差,又易出错,属过时的功能。

13. 位置编辑

位置编辑描述符用来在一个格式记录内移动文件的位置,以控制下一个要输入或输出的字符的位置。位置编辑通常用于方便表格数据的输入或制表输出。

对 T 及 TL 位置编辑描述符的解释,还受"左制表限"的影响。

在执行本节介绍的顺序格式输入/输出语句的开始时,文件总是定位于当前记录的开始处,即此记录的第1个字符处,此时,左制表限为1。

但是，在对某个文件执行 Fortran 90扩充的非推进式格式输入/输出的语句(参见第五节)后，且在紧接着执行的输入/输出语句的执行开始时，文件通常是定位于当前记录内的某个位置处，此时，此位置便成为左制表限。

下面给出了位置编辑描述符及其解释：

编辑描述符	功　能
Tn	跳至记录中相对于左制表限的第 n 个字符位置处。
TLn	从当前位置向左跳 n 个字符位置。但不会位于左制表限的左边。
TRn	从当前位置向右跳 n 个字符位置。
nX	同 TRn。

其中，n 是记录内字符位置或字符位置数。它必须是默认正整型字面常量。

(1)位置编辑描述符用于输入。若位置编辑描述符导致当前位置向右跳，其效果是不处理被跳过的字符，若导致当前位置向左跳，其效果是重复处理已处理过的字符。并且可以使用不同的编辑描述符处理这同一批字符。例如：

```
READ"(T3,I4,TL4,I1,TR1,I2)",L,M,N
```

若输入的记录是：

```
□ □ 4321
1 2 3456    ←  记录内字符位置。
```

则，L、M、N 的值分别是4321、4及21。其中，输入项 L 从第3个位置起读入4位数字(4321)，M 从第三个位置起读入1位数字(4)，而 K 则从第五个位置起读入两位数字(21)。

(2)位置编辑描述符用于输出。若位置编辑描述符导致当前位置向右跳，其效果是对被跳过的位置不作处理。而且，从未处理过的位置，被预设置为空格。若导致当前位置向左跳，其效果是重写有关字段，且覆盖原已写入的内容。例如：

若 X＝24.557，Y＝－3291.454

则：

```
PRINT"(1X,F9.2,TR4,F9.2)",X,Y
```

将产生如下结果：

□□□□24.56□□□□□-3291.45

而下面的例句在印制表头时非常有用：

PRINT"('1'///1X,T10,'QUANTITY',T22,'AMOUNT'/1X,&

T10,'SOLD',T22,'RECEIVED')

其执行将在一次的开头印出下面的栏目名：

□□□□□□□□□□QUANTITY□□□□AMOUNT

SOLD RECEIVED

进一步注释：

a.由位置编辑描述符进行定位本身，只是将记录中一个假想的"指针"从当前位置移动到指定的位置，而并不输出任何字符。指针的移动也不导致记录变长或变短。

b.当 TLn 导致可能使文件定位于左制表限的左边时，文件将定位在左制表限处。

c.在数据传送期间，若导致文件定位于下一记录，则左制表限被设定为1。

d.若由位置编辑描述符所跳过的字符是非默认种别的字符(严格说是非 Fortran 字符宽度字符，见字符编辑)，则使用该编辑的后果是依赖于处理系统的。

14.斜线编辑

在格式说明中，斜线编辑描述符用来终止当前正在输入或输出的记录，并且开始下一个记录的输入或输出。其形式为：

〔r〕/

其中，r 表示有 r 个连续的斜线。例如"3/"与"///"具有相同的效果，且三个连续的斜线之间有两个记录被空跳过去。

在格式说明中，斜线两边的逗号可以省略不写。例如，格式说明(1X,I3,/,I5)可以写为(1X,I3/I5)。但当斜线编辑描述符以 r/ 形式出现时，其前面的逗号不允许省略。

(1)斜线编辑描述符用于输入。输入时，若文件是连接为顺序

访问的，则文件位置跳过当前记录剩余的部分，并定位于下一记录的开始处。例如：

READ"(I5,I3/I5,I3,I2)",I,J,K,L,M

读入下面两个记录：

□□□99□□□10

□□100□□□11

将导致 I=99,J=0,K=100,L=0以及 M=11。第一个记录中的10被跳过。

若文件是连接为直接访问的，则文件位置跳过当前记录剩余的部分；记录号加1；并使具有此新记录号的记录成为当前记录。

(2)斜线编辑描述符用于输出。若文件是连接为顺序访问的，则首先终止向当前记录的输出；将当前记录传送到外部输出媒体；并使文件定位于下一记录的开始处。例如：

PRINT"(1X,F7.2/1X,2F7.2)",23.4125,33.54,1.7241

将产生下述两个记录：

□□□23.41

□□□33.54□□□1.72

下面的语句：

PRINT"(///)"

将在外部文件上产生四个空记录。

若文件是连接为直接访问的，则首先终止向当前记录的输出；将当前记录已产生的结果传送到外部输出媒体；然后记录号加1；并使具有此新记录号的记录成为当前记录。

15. 冒号编辑

在格式说明中，冒号编辑描述符用来有条件地终止整个格式控制。即：当格式控制遇到冒号编辑描述符时，若在输入/输出项表中已经不再存在需要输入/输出的项，则格式控制立即终止；若还存在需要输入/输出的项，则此冒号编辑描述符不起作用。例如：

PRINT"(1X,3(F5.1,:,',')",(1.1*I,I=1,N)

当 N=3时，输出结果如下：

□□1.1,□□2.2,□□3.3

此例中，冒号用来阻止第三个输出数据之后的逗号的输出。

阻止格式说明中的字符串编辑描述符在输出字段中产生编程序者不希望的字符串，是冒号编辑的主要用途。

冒号编辑前后的逗号可以省略。

五、输入/输出项表与格式说明的相互作用

在格式输入/输出中，输入/输出项表指定了要输入/输出的数据对象，而格式说明则指定每个数据对象在外部记录中的格式。在格式控制开始以后，总是要以格式说明为主导，从左至右地将格式说明中的“下一个如 I,B,O,Z,F,E,EN,ES,G,L,A 及 D 等数据编辑描述符”与输入/输出表中的“下一个有效的（有要输入/输出的数据的）输入/输出项”相配对，并由它们共同决定下一次的内部数据与其在格式记录中的外部形式之间的输入转换或输出转换（编辑）。当遇到格式说明中除数据编辑描述符以外的控制编辑描述符及字符串编辑描述符时，则都将被独立地进行解释，不需要输入/输出项与之配对，下面以它们的配对为依据分三种情况进行介绍。

1. 数据编辑描述符的数量与输入/输出项的数量相同

实现上述配对，必须先进行展开：对输入/输出表，要展开其中的每一项，如同在程序中写出了数组或数组片段中的每一个元素，或者展开了结构中的每一个成分，或者展开了隐 DO 一样；对于格式说明，则要对重复说明进行展开，例如：

```
PRINT"(1X,I2,2(I3,TR2,I4),F10.2/1X,'END-OF OUTPUT')",&
      I,(N(I),I=1,4),FLOATING
```

此语句中格式说明及输出表项分别展开之后，从左至右进行配对的对应关系如下：

```
(1X,         I2,  I3,TR2,    I4, I3,TR2,    I4, F10.2              )
 ↑           ↑    ↑          ↑   ↑          ↑   ↑                  ↑
开始格式说   ↓    ↓          ↓   ↓          ↓   ↓              终止格式说
明的左括号   I    N(1)       N(2)N(3)       N(4)FLOATING       明的右括号
```

这种从左至右的配对过程，一直进行到遇到终止格式说明的

右括号。且没有出现更多的输入/输出项时为止。

若输入/输出项表中至少有一个有效项，则格式说明中必须存在至少一个数据编辑描述符。

在输出时，形式为( )的格式说明(此时不允许有输入/输出项)，其效果是该语句的输出记录中没有字符；而在输入时，其效果是跳过输入文件中的一个记录。

格式说明中的每一数据编辑描述符，总是相应于一个输入/输出表中的有效项。但是，一个复型的输入/输出项，需要两个相同类型的数据编辑描述符与之相应。

2. 数据编辑描述符的数量多于输入/输出项的数量

只要格式控制在格式说明中遇到一个数据编辑描述符，它都要判定在输入/输出表中是否存在相应的有效项。若没有相应的有效项，则格式控制将立即终止，例如：

```
      PRINT 90,123
   90 FORMAT(1X,I4,1X,'IS THE ANSWER'/1X,&
        'NEW LINE',F10.2,'EXTRA CHARACTERS')
```

所产生的输出为：

```
 123 IS THE ANSWER
 NEW LINE
```

格式控制在 F10.2处被立即终止。若不希望输出第二个记录中的内容 NEW LINE 及该记录本身，则应在斜线编辑描述符/的前面加上一个冒号编辑描述符:。

3. 数据编辑描述符的数量少于输入/输出项

当输入/输出项中数据的数量等于或多于格式说明中展开后的编辑描述符的数量时，格式控制都会遇到完整格式说明中的最右侧右括号。

在格式控制遇到这样的右括号时，则输入/输出表中已没有尚待编辑的项，则格式控制将终止。如1. 中所述。

然而，若还有需要编辑的输入/输出项，则格式控制按下述规则进行：

这个终止格式说明的右括号本身将被当作如同一个斜线编辑描述符一样对文件进行定位，即结束上一记录并开始一个新记录的输入/输出。且：

(1)若格式说明中没有嵌套的括号，则格式控制返回到开始格式说明的左括号处。例如，执行下面的语句：

PRINT"(1X,2I4)",(I,I=1,5)

格式说明返回位置

将如同执行语句：

PRINT"(1X,2I4/1X,2I4/1X,I4)",(I,I=1,5)

并产生如下的输出记录：

□□□1□□□2

□□□3□□□4

□□□5

(2)若格式说明中有嵌套的括号，则不管嵌套有多少层，格式控制均返回到由格式说明最右括号的前一右括号所决定的格式项的开始处，且包括该格式项的重复说明。

例如：

```
      PRINT 900,1.0,2,3.0,4,5.0,6,7.0,8,9.0
900   FORMAT(1X,4(F10.2,I5))
```

格式说明返回位置

所产生的输出结果，将与使用下述格式语句所产生的输出结果相同。

```
900   FORMAT(1X,F10.2,I5,F10.2,I5,F10.2,I5,&
          F10.2,I5/1X,F10.2)
```

下面是一个更复杂的格式说明格式控制返回的示例：

(1X,I3/3(I5,F6.1)/1X,　4(2F10.3,3(I3,F7.2)),3I2)

格式控制返回位置

六、小结

本节介绍的是输入/输出中最主要的功能——顺序格式输入/输出。重点介绍了指定外部数据格式的格式说明，以及逐个介绍了众多的编辑描述符。所介绍的输入/输出项与格式说明的相互作用也是很重要的内容。

大多数编辑描述符均来自 Fortran 77。新扩充的包括用来实现二、八和十六进制整型数据输入/输出的 B,O,Z 编辑描述符，以及用来实现工程及科学形式数据输入/输出的 EN 及 ES 编辑描述符。

由于在 Fortran 90的数据类型中引入了种别的概念，某些外部数据的形式中(如字符型数据等)可以带有种别参数。

cH 编辑描述符在使用中较易出错，且其功能可由字符串编辑描述符所替代，因而被归类为过时的功能。

Fortran 90还提供了其它一些输入/输出能力，这些将在第四节介绍。

第四节　辅助输入/输出语句

Fortran 的输入/输出是对文件进行的，本节介绍对文件进行操作的辅助输入/输出语句。

一、与文件有关概念的补充

除第一节中已介绍的一些概念外，这里补充几个重要概念。

1. 文件的存在

按照计算机的概念，存在于磁盘或磁带上的文件就是存在的。但对 Fortran 而言，文件的“存在”被狭义地使用。它是相对某一个可执行程序的执行而言的，程序的执行只能访问那些对它来说已经“存在”的文件。

这样，在某个时刻，一个(或一些)文件是对一个可执行程序而

言是存在的文件，而另有许多文件则或者由于口令保护的原因，或者由于作业控制语言中未作适当说明等原因而对该可执行程序而言是不存在的。

注意，一个存在的文件可以不包含记录。例如，新创建而尚未写入内容的空文件。

于是，创建(create)一个文件只是意指使得原来不“存在”的文件变为“存在”，而删除(delete)一个文件只是意指终止该文件的“存在”。因而，这与通常的创建文件与删除文件的概念有一定的差异。

2. 文件的定位

对于每一个被处理的顺序访问的文件，都存在一个假想的指针，指向文件中的某一个位置，此位置称做文件(当前)位置。对文件的许多操作，都是相对此文件位置进行的，如图8.4所示。

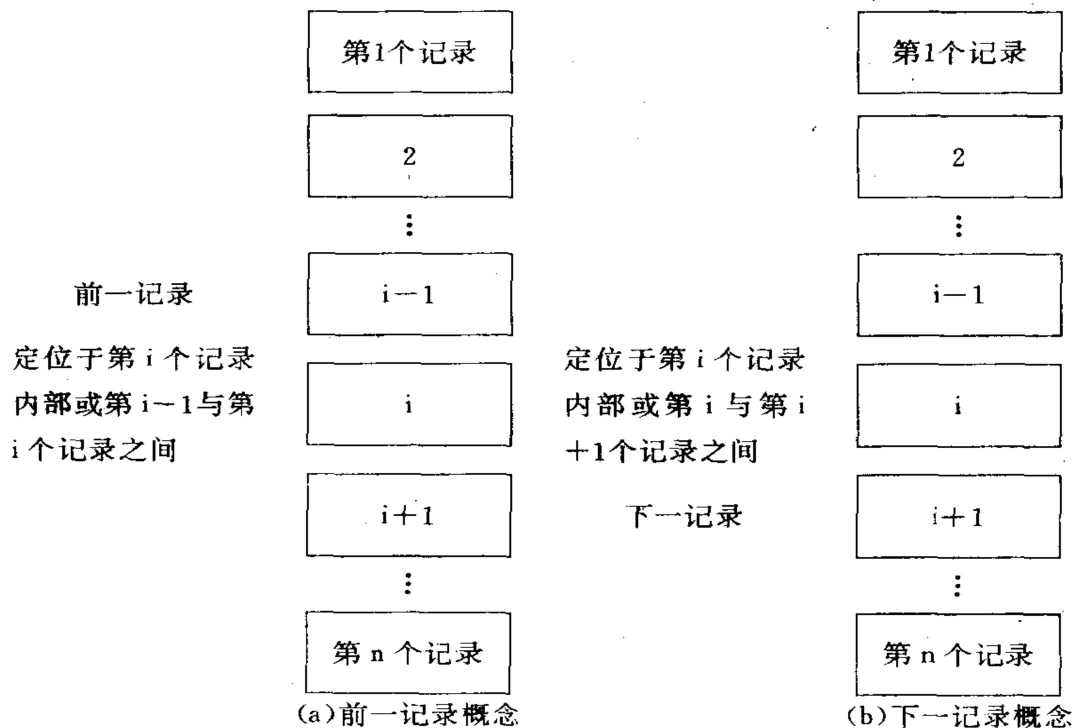


图8.4　顺序访问时前一记录与下一记录的概念

(1)文件的初始位置与结束位置。文件的初始位置位于第1个记录的紧前面，而文件的结束位置位于最后一个记录的紧后面。若

文件中没有记录，则文件的初始位置与结束位置位于同一个位置。

(2)文件的当前记录。若文件定位于某一个记录的内部，则该记录是文件的当前记录，若文件不是定位于一个记录的内部，而是定位于两个记录之间，则没有当前记录。

(3)前一记录与下一记录。若文件定位于图8.4(a)中的第i个记录内或定位于第i－1个记录与第i个记录之间，则第i－1个记录是前一记录。若文件已经定位于其初始位置，或者定位于第1个记录内，则没有前一记录。

若文件定位于图8.4(b)第i个记录内，或定位于第i个记录与第i＋1个记录之间，则第i＋1个记录是下一记录。若文件已经定位于其结束位置，或者定位于最后一个记录内，则没有下一记录。

执行READ、WRITE、PRINT语句以及本节要讨论的辅助输入/输出语句，常导致文件当前位置指针的改变。

3. 连接的属性

在数据传送输入/输出语句之前要在外部文件与部件之间建立连接(见第一节)，同时赋于这一连接某些属性。

比如，一个文件可能既可以存储格式数据也可以存储无格式数据，而当该文件为格式输入/输出而连接到某部件时，则该连接才具有格式输入/输出这一属性；同样，一个文件可能既可以用顺序访问的方法访问也可以用直接访问方法访问，而当该文件为顺序访问而连接时，则该连接才具有顺序访问的属性。因而，这类属性是属于每次连接的，而不是属于文件的。

本节将介绍的OPEN语句用来建立外部文件与部件之间的连接，并赋于这一连接一组属性。而INQUIRE语句可以用来查询这种连接的属性。

二、OPEN 语句

OPEN语句用来将一个外部文件连接到某个部件，即建立连接，以使数据传送输入/输出语句能通过该部件实现对外部文件的访问。

在文件连接到部件的期间，还要通过连接说明符向这一连接赋于某些连接属性，或者改变这一连接的某些属性。例如下面的语句：

```
OPEN (UNIT=10,FILE="PLOT _ DATA",&
      ACCESS="DIRECT",RECL=N,ERR=99)
```

的执行，将打开文件 PLOT _ DATA，并将它连接到部件号10。该文件是为直接访问(ACCESS="DIRECT")而连接的。按下面 FORM=说明符中介绍的默认规则，此文件只用于无格式输入/输出。记录长度为 N。在执行此语句时若出现错误，控制将转向标号为99的语句处执行。执行此语句后，输入/输出语句对部件10的访问，就是对文件 PLOT _ DATA 的访问。

OPEN 语句的一般形式是：

OPEN(连接说明符表)

其中，可以使用的连接说明符如下：

〔UNIT=〕外部文件部件

IOSTAT=默认整型标量变量

ERR=标号

FILE=文件名表达式

STATUS=默认字符型标量表达式

ACCESS=默认字符型标量表达式

FORM=默认字符型标量表达式

RECL=整型标量表达式

BLANK=默认字符型标量表达式

POSITION=默认字符型标量表达式

ACTION=默认字符型标量表达式

DELIM=默认字符型标量表达式

PAD=默认字符型标量表达式

各个连接说明符之间还有一些相互制约关系，现逐个解释如下：

〔UNIT=〕外部文件部件

它应是一个默认整型表达式，其值为正，用来指定被连接的部件。若省略字符串 UNIT=，则部件必须是第一项。

FILE＝文件名表达式

文件名表达式是一个默认字符型标量表达式，其值必须是一个文件的名字，且忽略尾部的空格。此文件将被连接到指定的部件。

若省略文件名，而且该部件又未连接到文件，则此时 STATUS＝连接说明符必须指定为 SCRATCH，即打开一个由处理系统决定的文件，且它是一个临时文件。例如，下述语句：

```
OPEN (UNIT=8,STATUS="SCRATCH",&
        ACCESS="DIRECT",RECL=100)
```

将使部件8连接到一个临时文件。且按默认规则可用于直接访问的无格式输入/输出。

STATUS＝默认字符型标量表达式

此说明符用来设置所连接文件的状态。表达式的求值结果必须是：OLD、NEW、SCRATCH、REPLACE 或 UNKNOWN。

若指定了 OLD（打开一个旧文件），则该文件必须已经存在。若指定了 NEW（打开一个新文件），则该文件必须尚不存在，但由于本语句的执行将创建该文件，且该文件的状态也将变为 OLD。

若指定了 REPLACE（替换一个文件），且若该文件尚不存在，则创建该文件，但若该文件存在，则删除该文件，且以相同的文件名创建一个新文件。在上述两种情况下，文件状态均改变为 OLD。

若指定了 SCRATCH（临时文件），则创建一个临时文件。该文件只存在到对相应部件执行 CLOSE 语句时为止，或者存在到本程序执行终止时为止。临时文件必须是无名文件。

若指定了 UNKNOWN，则该文件的状态是依赖于处理系统的，当省略此说明符时，则默认的值为 UNKNOWN。

ACCESS＝默认字符型标量表达式

此表达式的值必须是 SEQUENTIAL 或 DIRECT，分别用来指定此次连接的访问方法是顺序访问还是直接访问。

若相关文件已经存在，则访问方法必须是该文件允许的方法。若该文件不存在，则创建一个文件，且此文件应允许此说明符所指

定的访问方法。若省略此说明符，则默认值为 SEQUENTIAL。

FORM＝默认字符型标量表达式

此表达式的值必须是 FORMATTED 或 UNFORMATTED，分别用来指定此文件是为格式或无格式输入/输出而连接的。

若省略此说明符，且若此文件是为顺序访问而连接的，则默认值为 FORMATTED。若此文件是为直接访问而连接的，则默认值为 UNFORMATTED。

RECL＝整型标量表达式

此表达式的值必须是正整数。

若此文件是为直接访问而连接的，则它指定文件中每个记录的长度，且此说明符必须出现；若文件是为顺序访问而连接的。则它指定记录的最大长度。

若文件是为格式输入/输出而连接的，则记录长度以可存储的默认字符的数量来度量；若文件是为无格式输入/输出而连接的，则记录长度依赖于处理系统的度量单位。且下面介绍的 INQUIRE 语句，可以用来测定无格式输入/输出语句（见第五节）中输出项表的长度，并依据此长度打开相应文件。

BLANK＝默认字符型标量表达式

此表达式的值必须是 NULL 或 ZERO。此说明符只用于为格式输入/输出而连接的文件，以便在格式输入时用来控制对外部记录中非前导空格的解释（见第三节 BN 及 BZ 编辑）。

若指定了 NULL，则在数值型编辑（见第三节）输入字段中的非前导空格均被忽略不计；若指定了 ZERO，则在数值型编辑输入字段中的非前导空格均被解释为零。若省略此说明符，默认值为 NULL。

POSITION＝默认字符型标量表达式

此表达式的值必须是 ASIS、REWIND 或 APPEND，用来指定在文件连接到部件后文件（当前）位置指针的位置。因而，此说明符只能用于带有 ACCESS＝"SEQUENTIAL"说明符的这类 OPEN 语句中。

REWIND 使文件定位于其初始位置(见图8.4),APPEND 使文件或者定位于其结束位置,或者定位于文件结束记录(若有的话)紧前面的位置。

若文件存在且已连接,则 ASIS 不改变文件位置;若文件存在但未连接,则 ASIS 不指定文件位置。若省略此说明符,默认值为 ASIS。

ACTION=默认字符型标量表达式

此表达式的值必须是 READ、WRITE 或 READWRITE,用来指定其后对所连接的文件可进行的输入/输出操作。

若指定 READ,则禁止对此文件执行 WRITE、PRINT 及 ENDFILE 语句;若指定 WRITE,则禁止对此文件执行 READ 语句。若指定 READWRITE,则对此文件执行任何数据传送语句都是允许的。若省略此说明符,默认值为 READWRITE。

DELIM=默认字符型标量表达式

此表达式的值必须是 APOSTROPHE、QUOTE 或 NONE。在表控格式(见第二节)或名表格式(见第五节)的输出时,此说明可用来指定输出字段中字符型常量的界限符。

若指定 APOSTROPHE,则撇号将用作字符型字面常量的界限符,且常量内部的撇号均被重复两次输出;若指定 QUOTE,则引号将用作字符型字面常量的界限符,且常量内部的引号均被重复两次输出。对于非默认字符型字面常量,在前导的界限符之前应有一个种别参数,且后随一个下划线,如2_'汉字'。

若指定 NONE,则输出的常量不带界限符。且常量内的撇号或引号也不被重复两次。若省略此说明符,则默认值为 NONE。

PAD=默认字符型标量表达式

此表达式的值必须是 YES 或 NO,用来指定输入记录中空格的填充。

若指定 YES,则当输入项表及格式说明需要的数据多于相关记录中包含的数据时,以空格填充该格式输入记录;若指定 NO,则输入项表及格式说明所要求的数据不得多于记录中所含有的数

据。若省略此说明符其默认值为 YES。

此外,IOSTAT=及 ERR=说明符请见第五节。

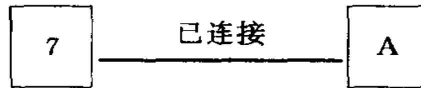
下面是进一步的注释:

(1)OPEN 语句可以出现在一个可执行程序的任何程序单元中,一旦建立部件与文件的连接,便可通过此部件访问该文件。

(2)在连接说明符中出现的字符型表达式的值,其尾部的空格均忽略不计。除在 FILE=说明符中以外,小写字母均被转换成大写字母。

(3)OPEN 语句的原始设计目的,是将一个文件连到一个部件。若部件与文件已经连接,则执行 OPEN 语句会另有含意,如下面所介绍的。首先假设7及8是部件号,而 A 及 B 为文件名。

允许将已连接的部件连接到另一个文件。即若文件已连接到部件7,如下所示:

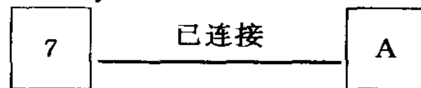


则对另一文件 B 执行下面的语句:

```
OPEN(UNIT=7,FILE="B",…)
```

其效果是:首先撤消这一连接,如同执行了一个不带 STATUS=说明符的 CLOSE 语句,然后再执行此 OPEN 语句,将文件 B 连接到部件7。

允许改变连接的某些属性,即若文件已连接到部件7,如下所示:



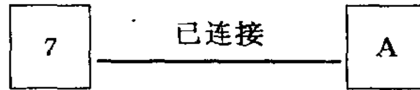
则对相同的部件和文件执行下面的语句:

```
OPEN(UNIT=7,FILE="A",…)
```

将修改此次连接中的某几个属性,即可由 BLANK=,DELIM=及 PAD=等三个说明符指定相应的新值,并用于其后的输入/输出中。此 OPEN 语句中还可出现具有不同参数的 ERR=及 IOSTAT=说明符。执行这样的 OPEN 语句不影响该文件的位置,且

不得再改变其它的连接属性。

不允许将已连接的文件连接到另一个部件上，如下所示：



则对另一部件8执行下面的语句：

OPEN(UNIT=8,FILE="A",…)

是不允许的。

三、CLOSE 语句

与 OPEN 语句的功能相对应，CLOSE 语句的执行，将终止一个外部文件至某个部件的连接。例如，执行下面的语句：

CLOSE(UNIT=10,STATUS='KEEP',ERR=100)

将终止连接到部件10的文件（例如 OPEN 语句例中的 PLOT_DATA 文件）与部件10的连接。并且，在终止这一连接后，相关文件仍然存在(STATUS='KEEP')。

CLOSE 语句的形式为：

CLOSE(关闭说明符表)

其中，可使用的关闭说明符如下：

[UNIT=]外部文件部件

IOSTAT=默认整型标量变量

ERR=标号

STATUS=默认字符型标量变量

其中的关闭说明符逐个解释如下：

[UNIT=]外部文件部件

除此说明符是用来指定要关闭的部件（终止该部件至某文件的连接）以外，其它均与 OPEN 语句中的解释相同。

STATUS=默认字符型标量变量

此表达式的值必须是 KEEP 或 DELETE。用以确定对连接到此部件的文件的后置处置。

若指定了 KEEP，则相关文件在被关闭后仍然存在，因此，不

得对状态为 SCRATCH 的临时文件指定 KEEP；若指定了 DELETE，则相关文件在被关闭后不再存在。

若省略此说明符，则对于状态不是 SCRATCH 的文件（参见 OPEN 语句）其默认值为 KEEP，而对于状态是 SCRATCH 的文件其默认值为 DELETE。

此外，IOSTAT＝及 ERR＝说明符请参见第五节。

下面是进一步的注释：

（1）在一个部件由于执行 CLOSE 语句而终止与文件的连接后，它还可以在同一个可执行程序内重新连接到同一个文件（若仍存在）或者另一个文件。

（2）在一个有名文件由于执行 CLOSE 语句而终止连接之后，只要该文件依然存在，它还可以在同一个可执行程序内重新被连接到同一个部件或者另一个部件。

（3）在一个可执行程序执行终止时，除由于出错条件而终止外，全部已连接的部件都将被关闭，其效果就如同对每一个这样的部件执行了一次省略了 STATUS＝说明符的 CLOSE 语句一样。

四、INQUIRE 语句

INQUIRE 语句可以用来查询某个有名文件的属性，或者查询文件与部件的连接的属性。这些被查询的属性包括：文件的存在、连接、访问方法、记录形式（格式记录或无格式记录）等。这种查询可以在文件与部件连接之前、期间、或断开连接之后进行。

有三种查询方式：按文件查询（通过 FILE＝说明符查询）、按部件查询（通过 UNIT＝说明符查询）以及按输出项表查询（通过 IOLENGTH＝说明符查询）。

按某个输出语句的输出项表查询，是为了得到该输出项表的（依赖于处理系统的）长度，以便在其后以此长度作为记录长度打开有关文件。

下面是几个例句：

```
INQUIRE (FILE="PLOT_DATA",OPENED=OP,&
```

```
              ACCESS=ACC)!按文件查询
    INQUIRE (UNIT=10,OPENED=LOG1,&
              NAMED=LOG2,FORM=CHAR_VAR,&
              IOSTAT=IOS)!按部件查询
    INQUIRE(IOLENGTH=IOL)A(1:N),X,Y,Z!按输出项表查询
```

其中,第一个语句查询 PLOT_DATA 文件的一些属性;第二个语句查询部件10与外部文件之间的连接的属性;第三个语句查询输出项表的长度。

1. 按文件及按部件查询

按文件及按部件查询的语句形式是:

INQUIRE(查询说明符表)

其中,查询说明符可以包括:

[UNIT=]外部文件部件

FLIE=文件名表达式

IOSTAT=默认整型标量变量

ERR=标号

EXIST=默认逻辑型标量变量

OPENED=默认逻辑型标量变量

NUMBER=默认整型标量变量

NAMED=默认逻辑型标量变量

NAME=默认字符型标量变量

ACCESS=默认字符型标量变量

SEQUENTIAL=默认字符型标量变量

DIRECT=默认字符型标量变量

FORM=默认字符型标量变量

FORMATTED=默认字符型标量变量

UNFORMATTED=默认字符型标量变量

RECL=默认整型标量变量

NEXTREC=默认整型标量变量

BLANK=默认字符型标量变量

POSITION=默认字符型标量变量

ACTION=默认字符型标量变量

READ＝默认字符型标量变量

WRITE＝默认字符型标量变量

READWRITE＝默认字符型标量变量

DELIM＝默认字符型标量变量

PAD＝默认字符型标量变量

若按部件查询，则查询说明符表中必须包含UNIT＝说明符。若按文件查询，则查询说明符表中必须包含FILE＝说明符。

下面详细介绍各项说明符，除专门指明的以外，它们均可出现在按部件查询或按文件查询的INQUIRE语句中。

〔UNIT＝〕外部文件部件

它应是一个默认整型表达式，其值为正，用来指定被查询的部件。它只用于按部件查询的INQUIRE语句。若省略任选的字符串UNIT＝，则此说明符必须是表中的第一项。

FILE＝文件名表达式

此表达式的值指定一个被查询的文件的名字，此文件可以是存在的或者不存在的。

EXIST＝默认逻辑型标量变量

若按文件查询，则此变量中返回的值为.TRUE.或.FALSE.分别指示该文件是否存在；若按部件查询，则此变量中返回的值为.TRUE.或.FALSE.分别指示该部件是否存在。

OPENED＝默认逻辑型标量变量

若按文件查询，则此变量中返回的值为.TRUE.或.FALSE.分别指示指定的文件是否连接到某个部件；若按部件查询，则此变量中返回的值为.TRUE.或.FALSE.分别指示该部件是否连接到某个文件。

NUMBER＝默认整型标量变量

此变量返回连接到被查询文件的部件号。若没有部件连接到此文件，则返回值为－1。

NAMED＝默认逻辑型标量变量

若相关文件有名，则此变量中被返回.TRUE.，否则为.FAL-

SE.。

NAME＝默认字符型标量变量

若相关文件有名，则此变量中被返回该文件的名字，否则是无定义的。

注意，当此说明符在一个按文件查询的语句中，返回的文件名不必与 FILE＝说明符中给出的文件名完全相同。例如，返回值中可能加入了一个用户标识，或者加入了处理系统要求文件名具有的其它信息。但返回的文件名应能在其后执行的 OPEN 语句中使用，且涉及同一个文件。

ACCESS＝默认字符型标量变量

若相关文件是为顺序访问而连接的，则变量中的返回值为 SEQUENTIAL；若是为直接访问而连接的，则返回值为 DIRECT；若不存在连接，则返回值 UNDEFINED。

SEQUENTIAL＝默认字符型标量变量

若顺序访问是被查询文件所允许的访问方法，则变量中返回值 YES；否则返回 NO；若处理系统不能判定是否允许顺序访问，则返回 UNKNOWN。

DIRECT＝默认字符型标量变量

若直接访问是被查询文件的允许的访问方法，则变量中返回值 YES；否则返回 NO；若处理系统不能判定是否允许直接访问，则返回 UNKNOWN。

FORM＝默认字符型标量变量

若文件连接用于格式输入/输出，则此变量中返回值 FORMATTED；若文件连接用于无格式输入/输出，则返回值 UNFORMATTED；若不存在连接，则返回值 UNDEFINED。

FORMATTED＝默认字符型标量变量

若格式记录是被查询文件所允许的记录形式，则此变量中返回值 YES；否则，返回值 NO；若处理系统不能判定是否允许格式记录，则返回值 UNKNOWN。

UNFORMATTED＝默认字符型标量变量

若无格式记录是被查询文件所允许的记录形式，则向此变量中返回值 YES；否则，返回值 NO；若处理系统不能判定是否允许无格式记录，则返回值 UNKNOWN。

RECL＝默认整型标量变量

在此整型变量中，将返回为直接访问而连接的文件的记录长度；或者返回为顺序访问而连接的文件的最大记录长度。若文件是为格式输入/输出而连接的，则长度是字符的个数；若文件是为无格式输入/输出而连接的，则长度依赖于处理系统的度量单位。若不存在连接，则该变量变为无定义的。

NEXTREC＝默认整型标量变量

对于一个为直接访问而连接的文件，此变量中返回的值，是上次读或写的记录的记录号＋1。若文件连接之后尚无记录被读或写过，则返回的值为1。若该文件不是为直接访问而连接的，或者由于先前操作的出错条件而使得文件位置是不确定的，则该变量变为无定义的。

BLANK＝默认字符型标量变量

对于为格式输入/输出而连接的文件，若 NULL 控制有效(见 OPEN 语句中的 BLANK＝说明符)，则此变量中返回值 NULL。若 ZERO 控制有效，则返回值 ZERO。若文件不是为格式输入/输出而连接的，或者不存在连接，则返回值 UNDEFINED。

POSITION＝默认字符型标量变量

若文件以定位于其初始位置的方式连接，则此变量中返回值 REWIND。若以定位于其结束位置或文件结束之前的方式连接，则返回值 APPEND。若文件连接时不改变文件位置，则返回值 ASIS。若不存在连接，或者文件是为直接访问而连接的，则返回值 UNDEFINED。

若文件在连接之后又被重新定位，则返回值是依赖于处理系统的。并且，此时只有文件确实处于其初始位置时，返回值才能为 REWIND；只有文件确实处于其结束位置或者处于文件结束记录紧前面时，返回值才能为 APPEND。

ACTION＝默认字符型标量变量

若该文件分别是连接用于输入的，输出的或输入/输出的，则此变量中返回的值分别是 READ、WRITE 或 READWRITE。若文件尚未连接，则返回值 UNDEFINED。

READ＝默认字符型标量变量

若 READ 操作对该文件分别是允许的、不允许的、或者是未确定的，则此变量中返回的值分别是 YES、NO 或者 UNKNOWN。

WRITE＝默认字符型标量变量

若 WRITE 操作对该文件分别是允许的、不允许的、或者是未确定的，则此变量中返回的值分别是 YES、NO 或者 UNKNOWN。

READWRITE＝默认字符型标量变量

若 READWRITE 操作对该文件分别是允许的、不允许的、或者是未确定的，则此字符变量中返回的值分别是 YES、NO 或 UNKNOWN。

DELIM＝默认字符型标量变量

此说明符用来查询由表控格式及名表格式所输出的数据中所使用的字符型字面常量的界限符。若撇号被用作界限符，则返回值 APOSTROPHE。若引号被用作界限符，则返回值 QUOTE。若不使用界限符，则返回值 NONE。若不存在连接，或者此次连接不是用于格式输入/输出的，则返回值为 UNDEFINED。

PAD＝默认字符型标量变量

若相应文件连接时所使用的 PAD＝说明符的值为 NO，则返回值为 NO，否则，返回值均为 YES。

此外，IOSTAT＝及 ERR＝说明符请见第五节。

表8－4是一个查询说明符总表。表中给出了执行按文件或按部件查询后各个查询说明符变量中所返回的值。其中，英文大写为返回值自身。

表8－4

说明符	按文件名查询		按部件查询	
	已连接	未连接	已连接	未连接
EXIST＝	.TRUE.（若文件存在） .FALSE.（若文件不存在）		.TRUE.（若部件存在） .FALSE.（若部件不存在）	
OPENED＝	.TRUE.	.FALSE.	.TRUE.	.FALSE.
NUMBER＝	部件号	－1	部件号	－1
NAMED＝	.TRUE.		.TRUE.（若文件有名） .FALSE.（若文件无名）	.FALSE.
NAME＝	文件名		文件名（若为有名文件） 无定义（否则）	无定义
ACCESS＝	SEQUENTIAL 或 DIRECT	UNDEFINED	SEQUENTIAL 或 DIRECT	UNDEFINED
SEQUENTIAL＝	YES,NO 或 UNKNOWN			UNKNOWN
DIRECT＝	YES,NO 或 UNKNOWN			UNKNOWN
FORM＝	FORMATTED 或 UNFORMATTED	UNDEFINED	FORMATTED 或 UNFORMATTED	UNDEFINED
RECL＝	记录长度（若为直接访问） 最大记录长度（若为顺序访问）	无定义	记录长度（若为直接访问） 最大记录长度（若为顺序访问）	无定义
NEXTREC＝	下一记录号（若为直接访问） 无定义（若为顺序访问）	无定义	下一记录号（若为直接访问） 无定义（若为顺序访问）	无定义
BLANK＝	NULL、ZERO 或 UNDEFINED	UNDEFINED	NULL、ZERO 或 UNDEFINED	UNDEFINED

续上表

说明符	按文件名查询		按部件查询	
	已连接	未连接	已连接	未连接
POSITION=	REWIND APPEND ASIS 或 UNDEFINED	UNDEFINED	REWIND APPEND ASIS 或 UNDEFINED	UNDEFINED
ACTION=	READ、WRITE 或 READWRITE	UNDEFINED	READ、WRITE 或 READWRITE	UNDEFINED
READ=	YES 或 NO			UNKNOWN
WRITE=	YES 或 NO			UNKNOWN
READWRITE=	YES 或 NO			UNKNOWN
DELIM=	APOSTROPHE QUOTE、NONE 或 UNDEFINED	UNDEFINED	APOSTROPHE QUOTE、NONE 或 UNDEFINED	UNDEFINED
PAD=	YES 或 NO	YES	YES 或 NO	YES

2. 按输出项表查询

对于无格式记录，其记录长度依赖于处理系统的度量单位。因而，当输出无格式数据时，通常需要知道在无格式输出时输出语句中输出项表的长度（见第五节）。按输出项表查询就是用来测定输出项表的长度，而此长度可以在其后执行的 OPEN 语句中用作 RECL＝说明符的参数，以提高程序的可移植性。

按输出项表查询的语句形式是：

INQUIRE(IOLENGTH＝默认整型标量变量)输出项表

其中，IOLENGTH＝说明符中的整型变量用来接受输出项表的长度值。

下面是一个按输出项表查询的例子：

```
PARAMETER(NUNIT=2,LEN=100)
REAL A(LEN); REAL (KIND=2) B(LEN), C(2*LEN)
INQUIRE (IOLENGTH=LENGTH) A, B, C
```

```
OPEN (UNIT=NUNIT, FILE="COMP_DATA",&
      FORM="UNFORMATTED", ACCESS="DIRECT", &
      RECL=LENGTH)
N=1
…
WRITE (UNIT=NUNIT, REC=N) A, B, C
…
```

其中,INQUIRE 语句查询输出项表 A,B,C 的长度,OPEN 语句按所返回的长度值打开一个无格式访问文件,而 WRITE 语句则以相同的输出项表向该文件写入无格式数据。

五、文件定位语句

在图8.4中介绍了有关文件位置的概念。每当执行一次顺序访问数据传送输入/输出语句,都将改变文件位置。

这里介绍的三个文件定位语句,用来辅助实现文件定位方面的操作。其中,BACKSPACE 语句用来回退一个记录;REWIND 语句用来"反绕"文件至文件起始位置;而 ENDFILE 语句用来在文件中产生一个文件结束记录。它们的形式为:

BACKSPACE 外部文件部件

BACKSPACE (位置说明符表)

或者

REWIND 外部文件部件

REWIND (位置说明符表)

或者

ENDFILE 外部文件部件

ENDFILE (位置说明符表)

其中,可用的位置说明符为:

〔UNIT=〕外部文件部件

IOSTAT=默认整型标量变量

ERR=标号

其中,〔UNIT=〕外部文件部件说明符是必须的,其它说明符

是任选的。而且，连接到此外部文件部件的文件必须是为顺序访问而连接的。

位置说明符解释如下：

〔UNIT=〕外部文件部件

指定连接到外部文件的部件。

此外，IOSTAT=及ERR=说明符请见第五节。

1. BACKSPACE 语句

在程序中可能遇到这样的情况：刚写入的记录需要重写，或者重读刚写入的记录以确认其内容的正确性等，BACKSPACE 语句用来实现这一操作。

执行 BACKSPACE 语句的效果是：若文件中存在当前记录，则定位于当前记录之前；若没有当前记录，则定位于前一记录之前。若既无当前记录也无前一记录，则文件位置不改变。若前一记录是文件结束记录（见 ENDFILE 语句），则定位于文件结束记录之前。例如：

```
   …
READ (UNIT=10) A,B,C,I,J            !读入一个无格式记录
   …                                !处理
BACKSPACE 10                        !回退一个记录
READ (UNIT=10) X,Y,Z,L,M            !重读同一记录
   …                                !作不同处理
```

2. REWIND 语句

REWIND 语句用来使一个连接到指定部件的外部文件定位于其初始位置。其目的是在对文件进行了某些输入/输出操作之后能重新定位于该位置，以便进行重写或重读。REWIND 一词源自早期的磁带文件，这一操作总是导致磁带反绕。

若文件已位于其初始位置，此语句的执行没有效果。

REWIND 语句的使用如下例所示：

```
   …
REWIND (8, ERR=90)          !反绕8号部件，使其处于初始位置
   …
```

```
READ (8,'(3F10.5)') X,Y,Z  !第一轮使用数据
  …
REWIND (8,ERR=90)          !再次使文件处于其初始位置
  …
READ (8,'(3F10.5)') A,B,C  !第二轮使用此数据
  …
```

3. ENDFILE 语句

ENDFILE 语句用来在指定的文件中写出一个文件结束记录(EOF),以标识文件的结束。此时,文件结束记录是该文件的最后一个记录,且文件定位于此记录之后。若进行数据传送之前,文件已定位于文件结束之后。则首先应通过执行 BACKSPACE 或 REWIND 语句使文件定位于文件结束之前或定位于文件初始位置。

若此文件也能连接用于直接访问,则只有在文件结束之前的那些记录被认为是已写出内容的记录,并能在此次直接访问中被读入。

若在随后的程序中存在着读到文件结束记录的可能性,则在相应的输入语句中应包含一个 END=控制说明符(见第五节),否则,读到此记录时,程序的执行将被终止。例如:

```
      …
      WRITE (10) A,B,C,D              !建立文件
      …
      ENDFILE 10                      !写入文件结束记录
      …
      REWIND 10                       !文件定位于起始位置,再
                                      !次处理此文件
      …
      READ (10, END=90) W,X,Y,Z       !读入数据,若遇到
      …                               !文件结束记录,控制转向
                                      !标号90处执行
90    …
```

第五节　其它输入/输出能力及控制说明符

在第二节及第三节中分别介绍了表控格式输入/输出及顺序格式输入/输出，并分别详细介绍了构成数据传送输入/输出语句的两个重要组成部分：输入/输出项表及格式说明。

本节的重点是介绍输入/输出语句的第三个重要组成部分——控制说明符表，它能控制实现不同的输入/输出能力。

由于控制说明符很多，而某些控制说明符之间又有较复杂的制约关系，因而本节沿用了前两节所使用的介绍体例：以编程序者关心的输入/输出功能为主导，给出它们的语句形式，再分别介绍各个有关的控制说明符。

一、非推进式的顺序格式输入/输出

非推进式的输入/输出是相对推进式的输入/输出而言的。

第三节介绍的顺序格式输入/输出就是推进式的输入/输出。所谓推进式，是在每执行完一次格式输入/输出语句之后，文件被定位于刚读入或写出的记录的紧后面，且在下一记录的前面。

请注意，本节中“推进式”的概念仅此而已，它不隐含着在行式打印机上应实际“进纸”。输出时，行式打印机的进纸总是由输出记录中的第一个字符控制的，如第三节的介绍。

推进式的输入/输出不足之处是，一次语句的执行只能成记录地处理数据，没有处理部分记录的能力。非推进式的输入/输出就是为处理部分记录而设计的，也常称作流式文件的输入/输出。

所谓非推进式，是当此格式输入/输出语句执行结束时，文件定位于刚读入或写出的字符之后，即，假想的文件指针位于当前记录的内部，而不是跳到此记录与下一记录之间。当前记录中剩余的字符，可以由其后执行的输入/输出语句来处理。

非推进式的输入/输出只适用于顺序格式输入/输出，且语句中应具有显式的格式说明（即既非表控格式亦非名表格式），并且

不适用于内部文件的输入/输出。

其语句形式如下：

```
READ (〔UNIT=〕输入部件 &
        ,〔FMT=〕格式 &
        ,ADVANCE=默认字符型标量表达式 &
        〔,SIZE=默认整型标量变量〕&
        〔,EOR=标号〕&
        〔,IOSTAT=默认整型标量变量〕&
        〔,ERR=标号〕&
        〔,END=标号〕&
        )〔输入项表〕
```

以及

```
WRITE (〔UNIT=〕输出部件 &
        ,〔FMT=〕格式 &
        ,ADVANCE=默认字符型标量表达式 &
        〔,IOSTAT=默认整型标量变量〕&
        〔,ERR=标号〕&
        )〔输出项表〕
```

从语句形式上讲，非推进式的与推进式的顺序格式输入/输出的主要区别是，在其 ADVANCE=控制说明符中的默认字符型标量表达式的值必须为 NO。

下面详细介绍各控制说明符：

〔UNIT=〕输入或输出部件的描述同第二节的介绍。

〔FMT=〕格式的解释见第三节中的介绍。注意，非推进式输入/输出时，不得将格式指定为星号 *（即表控格式）。

ADVANCE=默认字符型标量表达式，其中，对于非推进式的输入/输出，此表达式的值必须为 NO。若其值为 YES，或省略此控制说明符，均为推进式的格式输入/输出（即第三节所述的情形）。

SIZE=默认整型标量变量，此变量将被赋给在执行此输入语句时所读入的字符个数。

EOR＝标号，在读入字符过程中，若遇到记录结束(EOR)，则控制转向所指定的标号。此标号必须与本语句处于同一个作用域单位中。

IOSTAT＝默认整型标量变量，在执行本语句后，此变量中返回一个值，指示出此语句的执行状态：正值指示出现了出错条件；负值指示出现了文件结束或记录结束条件，区别这两种条件的值是依赖于处理系统的；零值指示未出现上述条件。

ERR＝标号，若此语句的执行出现出错条件，则控制转向此标号处继续执行。此标号必须与本语句位于同一个作用域单位中。

END＝标号，执行顺序访问的 READ 语句时，若导致读入文件结束记录(EOF)，则控制转向此标号处执行。此标号必须与本语句位于同一个作用域单位中。

下面是一个在终端上对话的程序例子：

```
CHARACTER * 1 CHAR
  ...
WRITE ( * ,"(1X,'Please enter Y or N : ')", ADVANCE='NO')
READ ( * ,"(A1)") CHAR
  ...
```

执行此程序，首先在终端上印出：

```
Please enter Y or N :
```

然后在同一行上等待键入一个字母，并将它读入 CHAR 中。

但也请注意，这一程序的含意常是依赖于处理系统的。首先，由星号指定的默认设备不一定是终端。其次，默认的输入设备与输出设备不一定被认为是同一个设备，比如，显示屏与键盘被认为是两个设备而不是一个设备。

二、无格式输入/输出

无格式输入/输出，是以计算机中的二进制内部形式进行数据的输入及输出。在输入/输出过程中，不进行数据形式的转换。

可以用直接访问方法或顺序访问方法进行无格式输入/输出。

无格式输入/输出的语句形式中，一定不能包含用来指定外部数据格式的FMT＝控制说明符。其语句形式为：

```
READ ([UNIT=]输入部件 &
        [,REC=记录号]&
        [,IOSTAT=默认整型标量变量]&
        [,ERR=标号]&
        [,END=标号]&
        )[输入项表]
```

以及

```
WRITE ([UNIT=]输出部件 &
        [,REC=记录号]&
        [,IOSTAT=默认整型标量变量]&
        [,ERR=标号]&
        )[输出项表]
```

这里，READ语句的一次执行，将从外部文件中读入一个无格式记录，并将其中的数据分别赋给输入项表的各个项；WRITE语句的一次执行，将把输出项表中的数据组织成一个无格式记录，并写到外部文件中。

注意，在某个处理系统上，由一个输出语句中的输出项表所建立的无格式记录的长度是由该处理系统决定的，而此记录长度可以通过执行对输出项表查询的INQUIRE语句(见第四节)得到。

语句中的控制说明符解释如下：

[UNIT＝]输入或输出部件，连接到此部件的文件，应是一个外部存储文件，而不是如打印机这样的外部显现装置；此文件必须是为无格式访问而连接的；就访问方法而言，它可以是直接访问或者顺序访问。

REC＝记录号，其中的记录号应是一个默认整型标量表达式。其值指定要读或写的记录。若出现此控制说明符，则指定进行直接访问的无格式输入/输出；若不出现此控制说明符，则指定进行顺序访问的无格式输入/输出。

IOSTAT＝、ERR＝及END＝等控制说明符请见本节一、中

的解释。

注意，只有在顺序访问的READ语句中(即不出现REC=说明符时)才允许出现判断文件结束的END=说明符。

下面是无格式读写语句的一些例句：

```
READ (UNIT=5, ERR=99, END=100) A,B, (C(I),I=1,40)
READ (9, REC=14, IOSTAT=IEND) X,Y
READ (5) Y
WRITE (9, IOSTAT=IS, ERR=99) A,B,C,D
WRITE (ERR=99, UNIT=7) X
WRITE (9, REC=RECORD_NUMBER) X
```

下面是一个应用无格式输入/输出的程序例子。该程序处理零件的库存管理问题，每个零件的数据包括零件号及库存量。

```
TYPE PART
  INTEGER :: ID_NUMBER, QTY_IN_STOCK
END TYPE PART
TYPE (PART), DIMENSION(10000) :: PART_LIST
INTEGER :: NUMBER_OF_PARTS
```

其中，PART_LIST是存放各种零件的数据的数组，而NUMBER_OF_PARTS是零件种类数。在一次数据处理完成之后，可将零件种类数及各种零件的数据以无格式数据的形式存储在外部文件中，该文件命名为PART_FILE，且在下面的OPEN语句中依据默认规则被连接用于顺序访问。程序如下：

```
OPEN (UNIT=8, FILE="PART_FILE", &
      POSITION="REWIND", FORM="UNFORMATTED,&
      ACTION="WRITE")
        ...
WRITE (UNIT=8), NUMBER_OF_PARTS, &
        PART_LIST (1:NUMBER_OF_PARTS)
```

当在另一种场合需要对上面已保存的数据作进一步的处理时，可用下面的程序，将数据读回程序相应的变元中：

```
OPEN (UNIT=8,FILE="PART_FILE",&
      POSITION="REWIND",FORM="UNFORMATTED",&
```

```
        ACTION="READ",STATUS="OLD")
        ...
    READ (UNIT=8), NUMBER_OF_PARTS, &
        PART_LIST (1:NUMBER_OF_PARTS)
```

三、直接访问的输入/输出

与顺序访问不同，直接访问不是按记录在文件中的次序去顺序地访问各记录，而是按文件中记录的记录号去随机地访问各记录。被访问的文件，可以是为格式输入/输出而连接的，也可以是为无格式输入/输出而连接的。其语句形式为：

```
    READ ([UNIT=]输入部件 &
          [,[FMT=]格式]&
          ,REC=记录号 &
          [,IOSTAT=默认整型标量变量]&
          [,ERR=标号]&
          )[输入项表]
```

以及

```
    WRITE ([UNIT=]输出部件 &
           [,[FMT=]格式]&
           ,REC=记录号 &
           [,IOSTAT=默认整型标量变量]&
           [,ERR=标号]&
           )输出项表
```

对于直接访问的无格式输入/输出，语句的每一次执行将读入或写出一个无格式记录；对于直接访问的格式输入/输出，语句的每一次执行，可以读入或写出一个或多个格式记录，详见第三节对斜线编辑描述符的描述。

上述输入/输出语句中的控制说明符解释如下：

[UNIT=]输入或输出部件，要求连接到此部件的文件，必须是为直接访问而连接的。

[FMT=]格式，若出现此说明符，则指定为格式输入/输出。

但不能指定星号 *（表控格式）。若省略此控制说明符，则指定为无格式输入/输出。

REC＝记录号，此控制说明符指定要读或写的记录的记录号。对于直接访问，此控制说明符是必须的。

IOSTAT＝及 ERR＝说明符见本节一、中的解释。

下面是直接访问的输入/输出的例句：

```
DIMENSION B(10),C(20),D(40)
    ...
READ (UNIT=7,FMT=FMT_F,REC=10) A !直接访问格式输入
READ (UNIT=8,REC=20) B,C,D(10:20) !直接访问无格式输入
WRITE (8,"(2F15.3)",REC=N+1,IOSTAT=IO_S,&
      ERR=90) X,Y
```

四、对内部文件的输入/输出

前面介绍的输入/输出，都是对驻存在外部存储媒体上或驻存在外部显现媒体上的文件进行输入或输出的。

对内部文件的输入/输出，则是把程序的字符型变量当作内部文件使用，以实现对“内存”这样的一种存储媒体进行输入或输出。

对内部文件进行输入/输出的实质是利用格式输入/输出语句所提供的数据形式转换能力，在内存变量之间，实现数据的内部二进制表示形式与其外部格式表示形式之间的转换。

对内部文件的输入/输出，总是顺序格式的输入/输出。其语句形式为：

```
READ ([UNIT=]输入部件 &
      ,[FMT=]格式 &
      [,IOSTAT=默认整型标量表达式]&
      [,ERR=标号]&
      [,END=标号]&
      )[输入项表]
```

以及

```
WRITE ([UNIT=]输出部件 &
```

```
,〔FMT=〕格式 &
〔,IOSTAT=默认整型标量变量〕&
〔,ERR=标号〕&
)〔输出项表〕
```

表明上述能力的一个简单程序例子如下：

```
CHARACTER (LEN=8) :: CHART
  ...
N=+12345
WRITE (UNIT=CHART, FMT="(I8)") N
```

此程序把整型变量N中以内部表示的整型数据转换成字符型变量CHART中以字符串表示的整数如下：

□□□12345

各控制说明符解释如下：

〔UNIT=〕默认字符型变量，此默认字符型变量被用作内部文件。若此变量是标量变量，则该内部文件由一个记录组成，且记录长度是该标量变量的长度；若此变量是一个数组或数组片段，则其中的每一个数组元素是该文件的一个记录，且此文件中的每一个记录具有相同的长度，即数组元素的长度。注意，此变量不允许是带向量下标的数组片段。

若写入到一个记录中的字符数目小于该记录的长度，则该记录余下的部分将填充以空格；写入字符的数目不得超过该记录的长度。

内部文件中的记录还可以用其它的手段使其成为有定义的，例如，可以用字符赋值语句对它们赋值。

在数据传送之前，内部文件总是被定位在其第一个记录的开始位置处。

FMT=说明符详见第三节，而IOSTAT=、ERR=及END=等说明符见本节一、中的描述。

注意，允许对内部文件进行表控格式的输入或输出。而且，此时的字符常量不被撇号'及引号"括起来；但是，不允许对内部文件进行名表格式的输入/输出。

内部文件适合于下面几种情况：例一是，所输入数据的格式用数据中的某个指示字段给出。例如，要读入一个由30个数字组成的记录，而该记录可能是30个1位的整数。或是15个两位的整数。或者是10个3位的整数。为此，例如还可在记录中设置第31个数字，且当其值为1、2或3时，分别指示上述记录的三种类型。

此时，可先将被读入的数据以字符型数据的形式读入一个用作内部文件的缓冲区 BUFFER 中，并将该记录的类型读入变量 TYPE 中。然后，再按记录的类型，正确地将数据从内部文件 BUFFER 中读入到数组变量 IVALUE 中。程序如下：

```
INTEGER :: IVALUE (30), TYPE
CHARACTER (LEN=30) :: BUFFER
CHARACTER (LEN=6) :: FRMT(3)=(/"(30I1)",&
                      "(20I2)","(10I3)"/)
  ...
READ "(A30,I1)", BUFFER, TYPE
READ (UNIT=BUFFER, FMT=FRMT(TYPE))&
     (IVALUE(I),I=1,30/TYPE)
```

例二是，当格式数据与内部形式的无格式数据混合存在，而又需要给出一致的格式数据时，可以利用对内部文件的输入/输出实现这一数据形式的转换。程序如下：

```
REAL CASH
CHARACTER (LEN=50) :: LINE, NAME * 12
NAME='A.□Smith□□□□'
CASH=329. 15
   ...
WRITE (UNIT=LINE,FMT="(3A,F8. 2,A)")&
   "Takings□for□Mr.□",NAME,"□are□",CASH, &
   "□dollars□"
```

此程序将字符型及实型数据组织成一个字符串(格式数据)，并输出到其名为 LINE 的内部文件中，该字符串是：

Takings□for□Mr.□A.□Smith□□□□are□329. 15□dollars

用作内部文件的字符型变量 LINE 中的值，其后可作进一步的处理。

注意，非默认字符型变量不能被用作内部文件。

五、名表格式输入/输出

名表格式(namelist formatting)输入/输出与表控格式类似，也是一种默认格式的输入/输出，它们都不需要在语句中以格式说明的形式显式地指定格式。它的不同之处在于，要输入/输出的对象应通过 NAMELIST 语句定义在一个名表组中，而且名表组名则应出现在 NML＝控制说明符中。

下面的输入程序的例子能帮助对名表格式输入/输出的理解：

```
INTEGER I; REAL X(8); CHARACTER(11) P
COMPLEX Z; LOGICAL G
NAMELIST/TODAY/G,I,P,Z,X
  ...
READ (UNIT= * ,NML=TODAY)
```

READ 语句中的名表组名 TODAY 指定了要输入 G,I,P,Z 及 X 等变量，而输入数据记录可以是：

```
&TODAY I=123, X(1)=12345, X(3:4)=2 * 3.0, I=5,
P="ISN'T _ BOB'S", Z=(123,0)/
```

记录中包括名表组名 TODAY，后随以各变量的名字及它们的值。程序执行后，各变量中的值为：

变量	值	注　　释
I	5	
X(1)	12345.0	
X(2)	未改变	未给出输入数据。
X(3)	3.0	
X(4)	3.0	
X(5)－X(8)	未改变	未给出输入数据。
P	ISN'T _ BOB'S	
Z	(123.0,0.0)	

G　　　　　　　未改变　　　　　　　未给出输入数据。

名表格式输入/输出的语句形式如下：

```
READ ([UNIT=]输入部件&
        ,[NML=]名表组名&
        [,IOSTAT=默认整型标量变量]&
        [,ERR=标号]&
        [,END=标号]&
        )
```

或者

```
WRITE ([UNIT=]输出部件&
        ,[NML=]名表组名&
        [,IOSTAT=默认整型标量变量]&
        [,ERR=标号]&
        )
```

其中的控制说明符解释如下：

[UNIT=]输入部件或输出部件如第二节的解释。

[NML=]名表组名应在 NAMELIST 语句中声明。通过此名表组名，指定了此语句中要输入或输出的对象。若省略任选的字符串 NML=，则它必须是控制说明符表中的第二项，而第一项必须是省略了 UNIT=的部件说明符。

IOSTAT=、ERR=及 END=等控制说明符同本节一、中的解释。

1. 关于名表格式输入数据

(1) 名表格式输入数据的形式与表控格式一样，外部记录中值本身的形式是：

c(常量)	各种类型的字面常量
r*c	r 个值 c
NULL(无值)	NULL 值
r*	r 个 NULL 值

数据的形式与表控格式中的一致(见第二节)。但字符型字面常量必须由界限符'或" 括起来。

（2）名表格式输入记录的形式是由输入记录的若干个空格开始；其后是字符 & 且紧随相应的名表组名；然后是若干个空格；再后是零个或多个由值分隔符分开的“名字—值子序列”所组成的序列，而此“名字—值子序列”的形式是：一个对象名或子对象指定符后随一个等号再后随一个或多个值及值分隔符；输入记录最后是一个斜线/，它用来终止数据的输入过程。请参见前面为名表组名 TODAY 准备的输入记录。

（3）在输入记录中，每个数据的名字必须与相应输入语句的名表组中某个名表组对象的名字相同，但顺序不必相同。

2. 名表格式输出数据的形式

当做名表格式输出时，所产生的输出记录的形式，除了实型、字符型及逻辑型常量本身的形式以外，与名表输入所要求的形式是一样的。例如：

```
INTEGER NUMBER, LIST(10)
NAMELIST/OUT/NUMBER,LIST
  ...
WRITE (UNIT=6,NML=OUT)
  ...
```

的输出结果可以是：

```
&OUT NUMBER=1, LIST=10,7*0,17,20/
```

在此假设变量 NUMBER 和 LIST 中在程序执行过程中已经赋了相应的值。下面做几点注释如下：

（1）若处理系统能表示大、小写字母，而输出记录中的名字将只使用大写字母。

（2）依赖于处理系统，输出的值可能由若干个空格分隔，也可能由前后有若干个空格的逗号分隔。

（3）一个记录将满时，由处理系统控制在新的记录上继续输出。只有复常量及字符型常量可能被分隔在两个记录上输出。

（4）整型、实型、逻辑型、复型及字符型常量的输出编辑规则，与表控格式输出相同，见第二节。

六、数据传送输入/输出语句的执行步骤

至此，可以总结一下数据传送输入/输出语句的执行步骤如下：

(1)确定数据传送的方向。即，此语句或者是从文件读入数据，或者是向文件写出数据。

对于一个不存在的文件(第四节)执行 WRITE 或 PRINT 语句将创建该文件。

(2)验明部件。若语句中有部件说明符，则判别是外部文件或者是内部文件。

由简缩的 READ 语句(没有显式地指定部件)所使用的部件与由 READ 语句中的星号 * 所标识的部件，都是依赖于处理系统的部件。

由 PRINT 语句(没有显式地指定部件)所使用的部件与由 WRITE 语句中的星号 * 所标识的部件，都是依赖于处理系统的部件。

在执行这类语句之前，该语句所使用的部件必须是已连接到文件的。或者通过执行 OPEN 语句连接，或者由系统预连接的。比如，许多系统都将4号部件预连接为输入部件而将5号部件预连接为输出部件。

(3)建立格式。或者建立表控格式(若指定了〔FMT=〕*)，或者建立名表格式(若指定了〔NML=〕名表组名)，或者建立由〔FMT=〕格式或〔FMT=〕标号指定的格式，或者建立无格式(若不指定格式)的数据传输。

(4)在数据传输之前对文件位置进行定位。

(5)在文件的记录与输入/输出表或名表所指定的实体之间传送数据，数据传送的次序，请见第三节中五、及本节的五、。

数据传送过程是一项一项独立完成的，例如语句：

```
READ (N) N,X(N)
```

中的第一个 N 中的值指明输入设备号，第二个 N 中的值将变为从

文件中读入的值，而当解释到第三个 N 时，其值为刚读入的值，且被用作下标。

(6)确定是否出现过出错条件、文件结束条件、或者记录结束条件。

(7)在数据传输之后对文件位置进行定位。

(8)使得 IOSTAT＝及 SIZE＝等说明符中指定的变量变为有定义的。

(9)终止数据传送语句的执行。

七、小　　结

本节依据程序设计中经常遇到的使用方法，补充介绍了 Fortran 90提供的六种输入/输出能力，即非推进式的顺序格式输入/输出、无格式输入/输出、直接访问的输入/输出、内部文件的输入/输出、表控格式输入/输出及名表格式输入/输出。并且给出了它们相应的语句形式，直观地展示了每一个语句形式中要使用的控制说明符及其相互搭配，以便于读者学习、使用。

Fortran 90新增加的能力主要是非推进式的顺序格式输入/输出及名表格式输入/输出。

小　　结

本章全面介绍了 Fortran 90中的输入/输出功能。它可分为两个部分。

第一，是数据传送输入/输出。这里，Fortran 提供了各种输入/输出能力，包括格式输入/输出，无格式输入/输出；推进式的输入/输出，非推进式的输入/输出(流式文件输入/输出)；顺序访问的输入/输出，直接访问的输入/输出；对外部文件的输入/输出，对内部文件的输入/输出等以及它们的适当组合。而在格式输入/输出中，又提供了极丰富的外部数据的形式。当应用中需要有很强的数据形式及记录格式控制能力时，读者就不会感到 Fortran 输入/输出

中内容的烦杂，而能体会到它是十分适用于工程、科学等广泛领域应用的一个实用性很强的语言，远非其它程序设计语言所能比拟的。

第二，是对文件的操作能力，这类能力在 Fortran 程序之内的输入/输出与 Fortran 程序之外的文件之间建立了一种可移植的界面。使 Fortran 程序能在不同的操作系统上运行。

Fortran 90在输入/输出中没有增加新的语句，但扩大了原有各个语句的能力。如非推进式的输入/输出等。同时，由于数据类型的扩充，也导致了输入/输出能力的扩充。

对于新扩充的非默认字符型数据的输入/输出，Fortran 90把许多功能都规定为是依赖于处理系统的，这一方面是因为尚缺少成功的系统实现的经验，另一方面也是由于许多非默认字符中都存在实际设备的显现宽度不同于 Fortran 字符的显现宽度的问题，这使具体实现变得比较复杂。

习　题

1. 解释下述基本概念：

(1)分别说明格式数据与无格式数据，格式记录与无格式记录以及存储格式记录与存储无格式记录的文件之间的区别。

(2)分别说明顺序访问(输入/输出)与直接访问、格式输入/输出与无格式输入/输出的区别。

(3)说明格式输入/输出、表控格式输入/输出及名表格式输入/输出之间的差别。

(4)对 Fortran 程序而言，顺序访问及直接访问等属性，是属于某个文件的，还是属于文件与部件的某次“连接”的，为什么？

2. 写出下面 WRITE 语句的输出结果(假设该处理系统在表控格式输出的以 I16编辑输出整型数据，以 F16.7或1PE16.8E2编辑输出实型数据)：

I＝1543；A＝＋253.7546；B＝0.054276

WRITE(*,*)I,A,B

3. 为下面的 READ 语句准备输入数据记录：

```
LOGICAL L
CHARACTER(LEN=4)CHAT1,CHAT2
READ(*,*)I,J,X,Y,L,CHAT1,CHAT2
```

要求执行 READ 语句后，I=425，J=−1363，X=−354.277，Y=−354.927，L=.FALSE.，CHAT1="ABCD"及 CHAT2="3*45"。

4. 对于整型数据33，分别以 I6.3，B6.3，O6.3及 Z6.3编辑后，写出其输出结果。

5. 写出下列表中各项的输入值：

输入字段中的数据	编辑描述符	输入值	注释
854	I3		
□83□□	I5		假设 BN 有效。
□83□□	I5		假设 BN 有效。
−5426	F5.2		
□45.3	F5.2		
+7435E4	F7.2		
□0.3645	E6.2		
−3.7164E−5	E10.2		
□6925E12	E8.2		
35426	EN5.2		
−3.7164E−5	ES10.2		

6. 写出下面各小题中在相应编辑描述符的控制下各数据的输出格式：

(1)I5：　0，12，−738，90574，−4357

(2)F7.2：　0.0，3.05，−42.374523，−3124.554

(3)E10.2：　0.0，−3.2475E2，352679.4

(4)EN14.3： 0.0,7.2546,3542.743,－67.35,353.642

(5)ES10.4： 0.0,0.00354,－7632.4507

7.对于实型数－0.0005324及＋17245.638,分别用 F14.4,E14.4,G14.4,EN14.4及 ES14.4编辑输出,写出其输出结果。

8.执行下面的程序：

```
M=12; X=384.65; Y=-295.4392
WRITE(5,200)M,X,Y
```

且若标号为200的格式语句分别为：

```
200  FORMAT(1X,I3,2F8.2)
200  FORMAT("□M=",I4,"□□□X=",F8.2,"□□□Y=",
          F8.2)
200  FORMAT(T2,"M=",I4, 3x,"X=",F8.2,TR3,"Y=",F8.2)
```

写出三种情况下的输出结果。

9.写出下面两个 WRITE 语句的输出结果：

```
INTEGER ::I=2546; REAL ::A=-42.7462
COMPLEX ::C=(-7.254,-12.9265)
LOGICAL ::L=.TRUE.
CHARACTER (LEN=4) ::CHAR="CHAR"
WRITE(*,"(1X,4(A,G10.2E2))") "I=",&
    I4,TR4,"A=",A,TR4,"L=",L,TR4,&
    "CHAR="CHAR
WRITE(*,"(1X,G10.2E2,'+',G10.2E2,'I')")C
```

10.编一个程序,它能将读入的一个整型数据转换为二进制数据,并打印输出。整数值的范围应不小于8位数字(提示,使用 SELECTED_INT_KIND 函数选择适当的种别)。

(1)要求利用内部文件实现。

(2)要求利用非推进式的顺序格式输出实现。

(3)要求将输入的整数分别以二进制、八进制及十六进制形式打印输出(提示:使用 B,O 及 Z 编辑描述符输出)。

11.说明下面的输出语句所输出数据的次序：

```
REAL A(10),B(10,20)
```

```
WRITE(UNIT=5,FMT="(1X,10F10.2)")A,B
WRITE(UNIT=5,FMT="(1X,10F10.2)")(A(I),I=1,10),&
  ((B(I,J),J=1,20),I=1,10)
```

12. 假设 X 是含有50个元素其秩为1的数组，为了读入前 N(N≤50)个元素的值，必须先读入输入记录中的值 N，然后以隐 DO 读入 X(1)，X(2)，…，X(N)。若以下述语句：

```
READ (UNIT=4,FMT=100) N,(X(I),I=1,N)
```

读入数据，且相应的 FORMAT 语句中的格式说明分别为：

```
(I2,3F4.2)
(I2/3F4.2)
(I2/(F4.2))
(I2,(/3F4.2))
(I2/(3F4.2))
```

说明哪些格式说明不正确?哪些不合理?为什么?对于其中正确的格式说明应如何安排输入数据?

13. 用 INQUIRE 语句查询下列有关文件及部件的情况：

(1)查询部件5是否与文件连接?所连接的文件的名字是什么?此次连接是用于格式访问还是无格式访问?是用于顺序访问还是直接访问?

(2)查询文件 MY_DATA 是否存在?是否已连接到部件?连接到哪个部件?是为顺序访问还是为直接访问而连接的?是为格式访问还是为无格式访问而连接的?是为读(输入)还是写(输出)或为读/写而连接的?若是为格式输入(或读)而连接的，输入字段中的非前导空格是被当作零处理还是被认为无效?

14. 对于下面的数组：

```
INTEGER (KIND=2),DIMENSION(20:50)::I,J
REAL,DIMENSION(100:100)::A,B
```

写一段程序：

(1)将数组 I,J,A 及 B 的内容作为一个无格式记录写入到为无格式直接访问而连接的文件中(该文件连到部件20)，要写入的记录号在变量 N 中。

(2)打开一个文件，其名为 MY _ DATA，用于无格式直接访问，连接到部件20。并且要求该文件中每个记录的长度足够大，以便容纳前面定义的 I，J，A 及 B 四个数组（提示：利用按输出项表查询语句以求得记录的长度）。

(3)用途同(2)但要求以一个临时文件实现。

第九章　程序单元

早期的 Fortran，例如 Fortran Ⅱ 和 Fortran Ⅳ，是一种纯粹块型结构的程序设计语言。一个 Fortran 程序由一个个相对独立的程序模块组成。其中有一个程序模块称为主程序模块，另一些程序模块称为辅程序模块，每个模块可调用其它模块。除主程序模块之外，也可被其它模块调用。这种程序模块就称为程序单元(program unit)，俗称程序段。从 Fortran77 开始，Fortran 语言逐步向过程嵌套型结构转变，即允许一个过程嵌套地包含一些内部过程。但为了保持原 Fortran 块结构的风格，仍保留了程序单元这种形式，而且 Fortran 90 又比 Fortran 77 多了一种称为模块的程序单元。于是，Fortran 90 中共有四种程序单元，即主程序(main program)，外部辅程序(external subprogram)，块数据程序单元(blockdata program unit) 和模块(module)。并且扩充为每个程序单元可调用其它程序单元，除主程序单元之外，也可被其它程序单元甚至自身调用。其中外部辅程序又可分为外部函数辅程序和外部子程序辅程序两种。有时，又将外部辅程序和用其它语言编写的可供 Fortran 程序调用的过程一起称为外部过程(external procedure)。我们将在第十章中进一步介绍各种过程。本章将仅对主程序、外部辅程序和块数据等三种程序单元进行讨论，而把非 Fortran 77 的成分模块程序单元留待第三篇去介绍。

第一节　主　程　序

主程序是整个 Fortran 程序的一个主控程序部分。主程序在一个 Fortran 程序中必须有一个，而且只能有一个。一个程序从主程序单元开始起动，然后它执行各种可执行语句，或不时地从中

调用其内部辅程序、外部辅程序和其它外部过程执行，并用各种控制语句控制着整个程序的执行流程。主程序被定义为其第一个语句不包含 SUBROUTINE，FUNCTION，MODULE，或 BLOCK DATA 语句的一种程序单元。其一般形式为：

〔PROGRAM 语句〕
〔说明部分〕
〔执行部分〕
〔内部辅程序部分〕
主程序的 END 语句

其中可省缺的 PROGRAM 语句具有形式：

PROGRAM 主程序名

主程序的 END 语句具有形式：

END〔PROGRAM〔主程序名〕〕

一个主程序单元如果有名，它用来标识该主程序，它必须是整个程序的一个全局名。它不能与程序中的其它程序单元的名、外部过程名、或公共块名相同，也不能与该主程序中的任何局部名相同。如果在主程序的 END 语句中包含主程序名，则它必须与开头的 PROGRAM 语句中的主程序名相同。而且只有当选用了 PROGRAM 语句以后，其 END 语句中才允许出现主程序名。

下面就是两个主程序单元的例子：

```
PROGRAM ANALYSE
  REAL A, B, C(10,10)           ! 说明部分，说明 A,B 和数组
                                ! C(10,10)为实型
  CALL FIND                     ! 执行部分，调用内部过程 FIND
  CONTAINS
    SUBROUTINE FIND             ! 内部过程开始
    ……
    END SUBROUTINE FIND         ! 内部过程结束
  END PROGRAM ANALYSE
```

要将三个实数按升序进行排序的程序，其基本步骤分三步：第一、读入数据，第二、排序，第三、印出结果。排序程序如下：

```
PROGRAM SORT _ 3
  IMPLICIT NONE
  REAL :: N1, N2, N3
    ! 读数据
  READ *, N1, N2, N3
  PRINT *,"INPUT DATA N1:", N1
  PRINT *," N2:", N2
  PRINT *," N3:", N3
    ! 排序
  IF(N1>N2) THEN
    TEMP=N1; N1=N2; N2=TEMP
  END IF
  IF(N1>N3) THEN
    TEMP=N1; N1=N3; N3=TEMP
  END IF
  IF(N2>N3) THEN
    TEMP=N2; N2=N3; N3=TEMP
  END IF
    ! 打印数据
  PRINT *, "THE NUMBERS, IN ACENDING ORDER, ARE"
  PRINT *, N1, N2, N3
END PROGRAM SORT _ 3
```

下面我们分别来讲述主程序单元中的各个部分。

一、说 明 部 分

主程序的说明部分由各种说明和 定义(详见第四章和第十三章)组成，用以说明其中各种实体的结构、类型、维数、大小和其它属性等。要求在主程序中将要用到的各种对象都要在说明部分中说明。未被说明的对象在以后被使用时，编译程序将发现错误。说明部分在主程序中是任选的，当没有对象需要说明时可以省缺。例如，下面的语句序列就可作为一个说明部分：

```
INTEGER,PARAMETER ::SHORT=3
```

```
REAL A(10)
LOGICAL, DIMENSION(5,5) :: MASK1, MASK2
COMPLEX :: CUBE_ROOT=(-0.5, 0.88)
INTEGER (SHORT) K
REAL (KIND=2) B
CHARACTER (LEN=10, KIND=2) C
REAL, ALLOCATABLE, DIMENSION(:) :: E
REAL, POINTER :: D(:, :)
```

值得注意，在主程序的说明部分中不允许包括任何 OPTIONAL 语句、INTENT 语句、PUBLIC 语句、PRIVATE 语句以及与它们等价的属性说明。主程序中的 SAVE 语句没有任何效果。

二、可执行部分

主程序的可执行部分是由各种可执行语句组成的序列，用来表示主程序要完成的加工和处理动作。一般来说，一个主程序可能先要输入一些数据或加工对象，然后对它们进行必要的运算和加工。包括调用各种内部的或外部的子程序或函数以完成必要的处理功能。最后，输出所需的结果。

一个程序的执行是从其主程序的可执行部分的第一个可执行构造或语句开始的。只要不遇到控制转移就一直继续顺序执行，直到执行主程序的 END 语句或任意可执行程序单元中的 STOP 语句为止。这时，一个程序的执行终止。在主程序中还可采用各种控制语句来控制和改变程序的执行流程。这些已在有关控制语句的章节进行了讨论，在此不再赘述。

值得注意，主程序一定不能是递归的。即任何程序单元包括主程序自身的执行部分都不能直接或间接地调用主程序。例如：

```
PROGRAM MAIN
  ……
  CALL SUB
END PROGRAM MAIN
```

```
SUBROUTINE SUB

  ……
  CALL MAIN
END SUBROUTINE SUB
```

就是一个错误的程序，因为主程序 MAIN 间接地调用了它自身。

三、主程序的内部辅程序

Fortran 90 的主程序中允许包含一些内部辅程序，供其可执行部分来多次调用执行。一般，把一些在主程序内部需要多次进行的处理功能独立出来编成内部辅程序，即内部过程。内部过程可以带参数，于是在可执行部分必要时就只需提供不同的参数值调用它们即可。内部过程的引入提供了在一个程序单元内共享一些公共代码的可能，从而简化了程序的编制。

内部辅程序可分为内在函数过程和内部子程序过程两种。它们的定义和调用方式与外部辅程序的类似，故都将在第二节中详细介绍。

主程序中的内部过程是局部于主程序的，即在主程序外的其它程序单元中是不允许也不可能调用它们的。在主程序中如果有内部辅程序，那么，正如上面的例子所示，它们必须跟在 CONTAINS 语句之后。这时，主程序称为其内部过程的"宿主(host)"。

第二节　外部辅程序

在 Fortran 中，把用 Fortran 语言编写的可供程序中多次调用的过程称为辅程序(subprogram)。辅程序有内部辅程序和外部辅程序两类。其中外部辅程序是 Fortran 的一种程序单元。它又可分为外部函数辅程序和外部子程序辅程序两种分别用来定义一个(或若干个)函数或子程序。由于它们是在主程序之外定义的，故又称外部函数或外部子程序，以便区别于在程序单元内部定义的

内在函数和内部子程序，而 Fortran 90 的内在函数或内部子程序是用内部辅程序来定义的。下面我们分别来叙述函数辅程序和子程序辅程序。

由于外部辅程序和内部辅程序具有统一的形式表示，区别仅在于内部辅程序具有更多的限制而已。下面我们将统一地来介绍辅程序。首先介绍函数辅程序。

一、函数辅程序

函数辅程序是以一个 FUNCTION 语句开头，以 END 语句结尾的一种程序结构。函数辅程序的一般形式为：

FUNCTION 语句
〔说明部分〕
〔执行部分〕
〔内部辅程序部分〕
函数辅程序的 END 语句

其中 FUNCTION 语句，也称函数语句，具有形式：

〔前缀〕FUNCTION 函数名（〔虚元名表〕)〔 RESULT（结果名）〕

这里，虚元名表是用逗号隔开的一些虚元名或星号（ * ）构成的序列，其中虚元是虚变元(dummy argument，曾称哑元)的简称，在有些程序设计语言中被称为形式参数。函数辅程序的虚元在语义上就是函数自变量的推广，可用来表示函数过程中需要根据具体情况变化的各种参数。星号虚元用来指明过程返回的位置，故相应的实元应是一些语句的标号。上述前缀可有下列两种形式：

类型说明〔 RECURSIVE 〕

或

RECURSIVE〔类型说明〕

例如：

FUNCTION 函数名( )
FUNCTION 函数名(X)
REAL FUNCTION 函数名(X,Y,Z)
RECURSIVE FUNCTION 函数名(X)

```
INTEGER RECURSIVE FUNCTION 函数名(X,Y) RESULT &
    (结果名)
```

等都是合法的FUNCTION语句。

函数辅程序的END语句具有形式：

```
END [FUNCTION [函数名]]
```

例如：

```
END
END FUNCTION
END FUNCTION 函数名
```

都是合法的函数辅程序的END语句。

注意，内部函数或模块函数的END语句中必须出现关键词FUNCTION。而在外部函数辅程序的END语句中则不一定需要出现这个关键词。

一个函数辅程序必须有名，它用来标识该函数辅程序，它也就是该函数辅程序定义的函数的名。如果未指明RESULT，该函数名也就是函数的结果变量，否则结果名作为它的结果变量。

一般，外部函数名必须是整个程序的一个全局名。它一定不能与程序中的其它程序单元的名、外部过程名、或公共块名相同，也不能与出现在该函数辅程序的作用域单位中说明语句里的名相同。但是，内部函数辅程序的函数名一般则是局部于其宿主的。所以，在不同的辅程序或主程序中的内部函数辅程序允许同名，同名的内部辅程序分别代表不同的函数过程。如果指明RESULT，则函数名必须不同于结果名。如果在函数辅程序的END语句中包含函数名，则它必须与相应的FUNCTION语句中的函数名相同。

如果函数辅程序直接或间接地引用自身或由该辅程序中的ENTRY语句定义的函数，则FUNCTION语句中必须出现关键词RECURSIVE。类似地，如果由该辅程序中的ENTRY语句定义的函数直接或间接地引用自身或由该辅程序中的其它ENTRY语句定义的函数、或由该FUNCTION语句定义的函数时，

FUNCTION 语句中的关键词 RECURSIVE 也必须出现。这表明由这个函数辅程序定义的是一个递归函数。

当 RECURSIVE 和 RESULT 同时指明时，所定义的函数的接口必须在该函数辅程序中明显地给出(见第十五章)。

1. 说明部分

函数辅程序的说明部分与主程序中的说明部分类似也由各种说明和定义(详见第四章和第十三章)组成，用以说明其中各种实体的结构类型、维数、大小和其它属性等。要求在辅程序中将要用到的各种实体应在说明部分中预先说明。在以后使用未被说明的实体时，编译程序将发现它作为一个错误。

说明部分是任选的。当没有对象需要说明时说明部分可为空。例如，下面的语句序列就可是函数辅程序的一个说明部分：

```
INTEGER,PARAMETER::SHORT=3
REAL A(10)
LOGICAL, DIMENSION(5,5) :: MASK1, MASK2
COMPLEX :: CUBE_ROOT=(-0.5, 0.88)
INTEGER (SHORT) K
REAL (KIND=2) B
CHARACTER (LEN=10, KIND=2) C
REAL, ALLOCATABLE, DIMENSION(:) :: E
REAL, POINTER :: D(:, :)
```

一个函数辅程序定义的函数的结果的类型或类型参数或者由 FUNCTION 语句的前缀指明，或者在该函数辅程序的说明部分中由结果变量名的类型指明。但是不能在两处同时指明。如果两处都没有指明，则按隐含类型规则确定结果的类型。如果函数结果是数组或指针，它必须在说明部分中对相应的结果变量指明。

在说明部分中还必须指明虚元的各种属性。包括用 INTENT 属性指明虚元是输入量(IN)还是输出量(OUT)，或既是输入量又是输出量(INOUT)，用OPTIONAL属性指明虚元在函数调用时是否一定要提供对应的实元，以及用EXTERNAL或INTRINSIC 属性指明虚元是否外部或内在函数且可作为其执行

部分某过程的实元等等。

2. 可执行部分

函数辅程序的可执行部分由一个可执行语句的序列组成。它把函数的自变量加工处理成函数的结果，即函数值。在函数辅程序被引用时，首先要进行虚元和实元的引用结合，然后从其可执行部分的第一个可执行语句开始执行。一当执行到 RETURN 语句或 END 语句时，就把结果变量的当前值作为函数值送出，然后返回引用处或某指定的位置。

如果在 FUNCTION 语句中指明 RESULT，则函数的结果变量就是跟在 RESULT 之后的结果名，否则结果变量就是函数名。函数值就被放在结果变量中。如果指明 RESULT，则函数名在执行部分的出现表示对该函数的递归调用，这时，该函数必须是递归的，在前缀中必须出现 RECURSIVE 。否则如果未指明 RESULT，函数名在可执行部分的出现表示对结果变量的引用。

结果变量在函数执行完成时的值就是由该函数送回的函数值。当函数的结果已被声明为一个指针时，则由函数送回的值的形状由结果变量的值在函数的执行完成时的形状决定，并且在函数的可执行部分中必须或者使该结果变量指针与一个目标结合，或者使该指针的结合状态变成不结合的。如果结果变量不是一个指针，则它的值必须由函数来定义。

可执行部分中的可执行语句包括各种赋值语句、控制语句、输入/输出语句、以及过程调用语句等。关于如何使用各种可执行语句来编制能完成各种功能的可执行部分已分别在有关控制语句、赋值语句和输入/输出语句等章节介绍，这里仅来介绍几个与函数辅程序的编写特别有关的语句，即 ENTRY，RETURN 和 CONTAINS 语句。

(1) ENTRY 语句。可以采用在外部辅程序的执行部分某些位置插入 ENTRY 语句的办法来定义多个外加的函数。ENTRY 语句具有如下形式：

ENTRY 入口名〔(〔虚元表〕)〔RESULT (结果名)〕〕

其中的入口名就是由该 ENTRY 语句所定义的函数名。如果未指明 RESULT，该函数名也就是函数的结果变量，否则结果名作为它的结果变量。

这时，函数引用从相应的 ENTRY 语句开始，首先进行虚元和实元的引用结合，然后执行紧跟其后的可执行语句，同样，当执行到 RETURN 语句或 END 语句时，就把结果变量的当前值作为函数值送出，然后返回引用处。

值得注意，RESULT 只能出现在函数辅程序的 ENTRY 语句中，在子程序辅程序的 ENTRY 语句中是不允许出现的。

如果在 ENTRY 语句中指明了 RESULT，则其入口名一定不能出现在其所在函数辅程序作用域单位的任何说明语句中，并且不能与相应的入口名相同。

ENTRY 语句只能出现在外部辅程序和模块辅程序中，而且一定不能出现在一个内部辅程序或控制结构中。在一个包含 ENTRY 语句的辅程序中，入口名一定不能是该辅程序的 FUNCTION 语句的虚元或该辅程序中其它 ENTRY 语句的虚元，并且不能出现在 EXTERNAL 和 INTRINSIC 语句中。

ENTRY 语句的虚元除非也是语句函数的虚元，否则不能出现在语句函数的表达式中。

关键词 RECURSIVE 不能用在 ENTRY 语句中，当必要时可标在其所在的辅程序的 FUNCTION 语句或 SUBROUTINE 语句中。

(2) RETURN 语句。RETURN 语句用来终结函数辅程序或子程序辅程序的一次执行。其一般形式为：

RETURN〔整型标量表达式〕

RETURN 语句必须包含在函数或子程序辅程序的作用域单位中。其中的整型标量表达式用来控制返回的位置。当其值 N 在 1 到其虚元表中星号(*)的个数之间时，则在返回时返回到本次引用的实元表中对应第 N 个星号(*)的返回位置处。当 N 超出范围时，与没有整型标量表达式的 RETURN 语句一样，都返回引

用处继续执行。

值得注意，辅程序的 END 语句的执行效果与不带标量表达式的 RETURN 语句的效果是一样的。

(3) CONTAINS 语句。CONTAINS 语句把主程序、辅程序、和模块的执行体与它们可能包含的内部辅程序或模块辅程序分开。它是不可执行的。其形式为：

```
CONTAINS
```

只有当在程序单元中有内部辅程序或模块辅程序时，才有 CONTAINS 语句，故可认为 CONTAINS 语句是内部辅程序部分或模块辅程序部分的开始。

3. 内部辅程序部分

在外部函数辅程序中可以包含内部函数辅程序或内部子程序辅程序供该函数辅程序调用。它们以 CONTAINS 语句与该函数辅程序的可执行部分分开。

内部过程是一个由内部辅程序定义的过程。内部过程可以出现在主程序、外部辅程序和模块辅程序中。但是内部过程一定不能出现在其它的内部过程中。即不允许在内部过程中再嵌套内部过程。因此，Fortran 90 仍不是一种过程嵌套式的语言，而基本上仍还是块结构的。不过值得注意，一个内部过程除了它的名不是全局名、其中不能包含 ENTRY 语句、内部过程名不能作为与虚过程结合的实元、内部过程不能再包含其它内部过程以及内部过程可以通过宿主结合存取其宿主的各种实体之外，它的语法和语义与外部过程是相同的。例如：

```
FUNCTION F(X) RESULT (F_RESULT)
  IMPLICIT NONE
  REAL :: F_RESULT, X
  F_RESULT =(1+1/X) * * X
END FUNCTION F
```

是一个计算函数

```
F(X)=(1+1/X) * * X
```

的函数辅程序。它可被下列主程序来调用：

```
PROGRAM FUNCTION _ VALUES
  IMPLICIT NONE
  REAL :: F, X
  INTEGER :: I
  DO I=0, 10
    X=10.0* *I
    PRINT *, X, F(X)
  END DO
END PROGRAM FUNCTION _ VALUES
```

下面是一个递归函数的例子：

```
RECURSIVE FUNCTION CUMM _ SUM (ARRAY) RESULT &
    (C _ SUM)
  REAL ARRAY (:), C _ SUM (SIZE (ARRAY))
  INTENT (IN) ARRAY
  INTEGER N
  N = SIZE (ARRAY)
  IF (N .LE. 1) THEN
    C _ SUM = ARRAY
  ELSE
    N = N/2
    C _ SUM (:N) = CUMM _ SUM (ARRAY (:N))
    C _ SUM (N+1:) = C _ SUM (N) + CUMM _ SUM &
        (ARRAY (N+1:))
  END IF
END FUNCTION CUMM _ SUM
```

不难看出，在其可执行部分中多处调用了函数 CUMM _ SUM 本身。

二、子程序辅程序

子程序辅程序是以一个 SUBROUTINE 语句开头，以 END 语句结尾的一种程序结构。子程序辅程序的一般形式为：

SUBROUTINE 语句

〔说明部分〕

〔执行部分〕

〔内部辅程序部分〕

子程序辅程序的 END 语句

其中，SUBROUTINE 语句，也称子程序语句，具有形式：

〔 RECURSIVE 〕 SUBROUTINE 子程序名〔(〔虚元名表〕)〕

这里，虚元名表是由虚元名或星号(*)用逗号分开的一个序列。星号虚元用来指明过程返回的位置，故相应的实元应是一些语句的标号。例如：

```
SUBROUTINE SUB1
SUBROUTINE SUB2 ( A, B, C )
SUBROUTINE SUB3 ( )
RECURSIVE SUBROUTINE S4 ( A, B )
RECURSIVE SUBROUTINE R( A, B, * , * )
```

等都是合法的 SUBROUTINE 语句。

子程序辅程序的 END 语句具有形式：

END 〔 SUBROUTINE 〔子程序名〕 〕

例如：

```
END
END SUBROUTINE
END SUBROUTINE SUB_NAME
```

等都是合法的子程序辅程序的 END 语句。注意：在内部子程序或模块子程序的 END 语句中必须有关键词 SUBROUTINE。而在外部函数辅程序的 END 语句中则不一定需要出现这个关键词。

一个子程序辅程序必须有名，它用来标识该子程序辅程序，它也就是该子程序辅程序定义的子程序的名。一般，外部子程序名必须是整个 Fortran 程序的一个全局名。它不能与该程序中的其它程序单元的名、外部过程名、或公共块名相同，也不能与出现在该子程序辅程序的作用域单位中说明语句里的名相同。但是，内部子程序辅程序的子程序名一般则是局部于其宿主的。所以，在

不同的外部辅程序或主程序中的内部子程序辅程序允许同名，同名的内部子程序辅程序分别代表不同的子程序。如果在子程序辅程序的 END 语句中包含子程序名，则它必须与开头的 SUBROUTINE 语句中的子程序名相同。

如果一个子程序辅程序直接或间接地引用自身或由该辅程序中的 ENTRY 语句定义的子程序，则子程序语句中必须出现关键词 RECURSIVE。类似地，如果由该外部辅程序中的 ENTRY 语句定义的子程序直接或间接地引用自身或由该外部辅程序中的其它 ENTRY 语句定义的子程序、或由该 SUBROUTINE 语句定义的子程序时，子程序语句中的关键词 RECURSIVE 也必须出现。当上述情形出现时，所定义的子程序称为递归子程序。

如果子程序语句中指明 RECURSIVE，则所定义的子程序的接口必须在该子程序辅程序中明显地给出(见第十五章)。

1. 说明部分

子程序辅程序的说明部分与主程序中的说明部分类似也由各种说明和定义(见第四章和第十三章)组成，用以说明其中各种实体的结构、类型、维数、大小、和其它属性等。要求在辅程序中将要用到的各种实体都要在说明部分中预先说明。未被说明的实体在以后被使用时，编译程序将发现它作为一个错误。说明部分是可任选的。当没有实体需要说明时说明部分可为空。例如，下面的语句序列就可是子程序辅程序的一个说明部分：

```
INTEGER,PARAMETER::SHORT=3
INTEGER A(10)
REAL, DIMENSION(10,5) :: K1, K2
COMPLEX :: CUBE_ROOT=(-0.5, 0.88)
INTEGER (SHORT) SI
CHARACTER (LEN=10, KIND=2) C
REAL, ALLOCATABLE, DIMENSION(:) :: E
INTEGER, POINTER :: PT(:, :)
```

形式上，子程序辅程序的说明部分与函数辅程序的没有区别。当然，其中到底需要说明些什么实体，要根据具体需要而定。

2. 执行部分

子程序辅程序的执行部分由一个可执行语句的序列组成。它描述由其定义的子程序所需的加工和处理。一般它将对输入参数进行各种处理和加工，并输出一个或多个结果。它也可采用在其执行部分某些位置插入 ENTRY 语句的办法来定义多个外加的子程序。子程序辅程序被 CALL 语句引用时，首先进行虚元和实元的引用结合，然后从其执行部分的第一个可执行语句开始执行。一当执行到 RETURN 语句或 END 语句时，就返回引用处或由某星号（*）虚元指明的位置。其 END 语句的执行与不带标量表达式的 RETURN 语句的执行效果一样。其实，子程序辅程序的执行部分与函数辅程序的执行部分的编写没有原则的区别。它与内部辅程序之间也用 CONTAINS 语句分开。

关于如何使用各种可执行语句来编制能完成各种功能的执行部分已分别在有关控制语句、赋值语句和输入输出语句等章节介绍，这里仅需提一下几个与子程序辅程序的编写特别有关的语句，即 ENTRY，RETURN 和 CONTAINS 语句。

（1）ENTRY 语句。它可以采用在外部辅程序的执行部分某些位置插入 ENTRY 语句的办法来定义多个外加的子程序。

子程序辅程序中的 ENTRY 语句具有如下形式：

ENTRY 入口名〔（〔虚元名表〕）〕

其中，入口名就是由该 ENTRY 语句所定义的子程序名。

这时，子程序的引用从相应的 ENTRY 语句开始，首先进行虚元和实元的引用结合，然后执行紧跟其后的可执行语句，同样，当执行到 RETURN 语句或 END 语句时，就返回引用它的 CALL 语句之后或某指定的位置继续执行。

ENTRY 语句只能出现在外部辅程序和模块辅程序中，而且一定不能出现在一个内部辅程序或控制结构中。在一个包含 ENTRY 语句的辅程序中，入口名一定不能是该辅程序的 SUBROUTINE 语句的虚元或该辅程序中其它 ENTRY 语句的虚元，并且不能出现在 EXTERNAL 和 INTRINSIC 语句中。

ENTRY 语句的虚元除非也是语句函数的虚元，否则不能出现在语句函数的表达式中。

关键词 RECURSIVE 不能用在 ENTRY 语句中，当必要时可标在其所在的外部子程序辅程序的 SUBROUTINE 语句中。

（2）RETURN 语句和 CONTAINS 语句。在子程序辅程序中，RETURN 语句和 CONTAINS 语句的语义与函数辅程序中的语法和语义完全相同，读者可参阅前面的叙述。

3. 内部辅程序部分

在外部子程序辅程序中可以包含内部函数辅程序或内部子程序辅程序供该子程序辅程序调用。它们以 CONTAINS 语句与该子程序辅程序的执行部分分开。

内部过程是一个由内部辅程序定义的过程。内部过程可以出现在主程序、外部辅程序和模块辅程序中。但是内部过程一定不能出现在其它的内部过程中。即不允许在内部过程中再嵌套内部过程。因此，Fortran 90 仍不是一种过程嵌套式的语言，而基本上仍还是块结构的。不过值得注意，一个内部过程除了它的名不是全局名、其中不能包含 ENTRY 语句、内部过程名不能是与虚过程结合的实元、内部过程不能再包含其它内部过程以及内部过程可以通过宿主结合存取其宿主的各种实体之外，它的语法和语义与外部辅程序过程是相同的。例如，下面就是一个合法的外部子程序辅程序的例子：

```
SUBROUTINE OUTER                         ! 子程序 OUTER 的开头
  REAL, POINTER :: A (:)                 ! 说明部分开始
  ……                                     ! 说明部分结束
  ALLCOCATE (A (1:N))                    ! 执行部分开始
  CALL INNER (A)
    ……                                   ! 执行部分结束
CONTAINS
  SUBROUTINE INNER (B)                   ! 内部子程序 INNER 开始
    REAL :: B (:)                        ! INNER 说明部分开始
    ……
```

```
    REAL :: C (:)                       ! INNER 说明部分结束
    CALL SET ( B(1), 1.0 )              ! INNER 执行部分开始
    ''''''
    ……                                  ! INNER 执行部分结束
  END SUBROUTINE INNER
  SUBROUTINE SET (C, D)                 ! 内部子程序 SET 开始
    REAL, INTENT (OUT) :: C             ! SET 说明部分开始
    REAL, INTENT (IN) :: D              ! SET 说明部分结束
    C = D                               ! SET 执行部分
  END SUBROUTINE SET
END SUBROUTINE OUTER                    ! 子程序 OUTER 的结尾
```

采用子程序辅程序可将第一节中的 PROGRAM SORT _ 3 改写成：

```
PROGRAM SORT _ 3
  IMPLICIT NONE
  REAL :: N1, N2, N3
  CALL READ _ THE _ NUMBERS
  CALL SORT _ THE _ NUMBERS
  CALL PRINT _ THE _ NUMBERS
  STOP
CONTAINS
  SUBROUTINE READ _ THE _ NUMBERS ! 读数
    READ *, N1, N2, N3
    PRINT *,"INPUT DATA N1:", N1
    PRINT *,"                N2:", N2
    PRINT *,"                N3:", N3
  END SUBROUTINE READ _ THE _ NUMBERS
  SUBROUTINE SORT _ THE _ NUMBERS ! 排序
    IF(N1>N2) THEN
        CALL SWAP (N1, N2)
      END IF
      IF(N1>N3) THEN
        CALL SWAP (N1, N3)
```

```
        END IF
        IF(N2>N3) THEN
            CALL SWAP (N2, N3)
        END IF
      END SUBROUTINE SORT_THE_NUMBERS
      SUBROUTINE PRINT_THE_NUMBERS ! 打印数据
        PRINT *,"THE NUMBERS,IN ACENDING ORDER,ARE"
        PRINT *,N1, N2, N3
      END SUBROUTINE PRINT_THE_NUMBERS
    END PROGRAM SORT_3
    SUBROUTINE SWAP (A, B)
      IMPLICIT NONE
      REAL :: A, B, TEMP
      TEMP = A
      A = B
      B = TEMP
    END SUBROUTINE SWAP
```

从上述例子我们可以看到三个内部辅程序和一个外部辅程序，以及与主程序之间的关系。

第三节　辅程序中虚元的限制

从上面已经看到，不论是函数辅程序还是子程序辅程序都可带虚元(在其它程序设计语句中称为形式参数)，在不同的场合可用不同的实体来替换它们，从而使辅程序的功能更加灵活。但是，值得注意，对于与一个辅程序的虚元相结合的实体有如下的限制：

(1) 影响该实体的值或可用性的任何动作只能通过与之结合的虚元进行。例如上例中，在INNER子程序的执行中不允许作诸如下列的赋值：

```
A (1) = 1.0
```

因为这将改变A的值而不通过与之结合的虚元B。可是，正如上

述，执行语句：

```
CALL SET (A (1), 1.0)
```

是允许的。

(2) 如果一个实体的任意的一部分是由虚元定义的，则在过程执行的任何时刻，不论在定义之前还是定义之后，它只能通过与其结合的虚元来引用。例如：

```
PROGRAM MAIN
  REAL :: W, X, Y, Z
  ……
  CALL INIT (X)
END PROGRAM MAIN
SUBROUTINE INIT (V)
  REAL :: W, X, Y, Z
  ……
  READ (*, *) V
  ……
END SUBROUTINE INIT
```

中，在 INIT 被执行的过程中，任何时刻都不能直接地引用变量 X。因为这时 X 是通过 V 定义的，若要引用必须通过 V 间接地引用。

一个虚元在辅程序的一次执行中被称为是已到位的 (prsent)，如果它已与一个实元结合，而且该实元或者是调用过程中的一个已到位的虚元，或者不是调用过程的一个虚元；否则称为是未到位的。在过程执行时，非任选的(即没有 OPTIONAL 属性的)虚元在一次执行中必须是已到位的，而且只有任选的虚元才允许是未到位的。但一个未到位的任选的虚元必须满足下列限制：

(1) 如果它是一个虚拟数据对象，则它一定不能被引用或定义。

(2) 如果它是一个虚过程，则它一定不能被调用。

(3) 除了可作为内在函数 PRESENT 的自变量之外，它一定

不能作为与非任选虚元对应的实元。

（4）它的子对象一定不能作为与任选虚元对应的实元。

一个未到位的任选虚元在一次执行中可作为另一过程的一个任选虚元对应的实元，但该虚元仍不被认为是已与一个实元结合的，因而也是未到位的。

第四节　块数据程序单元

块数据程序单元是一种为有名公用块中的数据对象提供初值的一种程序单元。一般这种给数据对象赋初值的过程称为数据的初始定义或初始化。块数据程序单元的一般形式为：

BLOCK DATA 语句

〔说明部分〕

块数据程序单元的 END 语句

其中，BLOCK DATA 语句具有形式：

BLOCK DATA〔块数据名〕

又称块数据语句。块数据程序单元的 END 语句具有形式：

END〔BLOCK DATA〔块数据名〕〕

一个块数据程序单元如果有名，它用来标识该块数据。如果在块数据的 END 语句中包含块数据名，则它必须与开头的块数据语句中的块数据名相同。而且只有当块数据语句中具有块数据名时，其 END 语句中才允许出现这个相同的块数据名。

在块数据程序单元的说明部分，只可包含 USE 语句、类型语句、IMPLICIT 语句、PARAMETER 语句、导出类型的定义、以及下列说明语句：COOMMON、DATA、DIMENSION、EQUIVALENCE、INTRINSIC、POINTER、SAVE 和 TARGET。而且在其中的类型语句中一定不能包含 ALLOCATABLE、EXTERNAL、INTENT、OPTIONAL、PRIVATE 和 PUBLIC 等属性指明符。

在一个块数据程序单元中，可以有多块有名公用块的数据对象被初始定义，或初始化。但是，在一个有名公用块中只要有一

个数据对象被初始定义之后，所有占有该公用块存储序列的存储单元的数据对象都已被说明过了，尽管并不要求它们全部应被初始定义。而且应注意，只有有名公用块中的非指针对象可被初始定义。

所有与一个公用块的某对象相结合的对象都被认为是该公用块中的对象。

在一个可执行程序中，相同的有名公用块一定不能同时在多个块数据程序单元中被指明。并且，在一个可执行程序中，一定不能有多个无名公用块。下面是一个块数据程序单元的例子：

```
 BLOCK DATA WORK
  COMMON / WRKCOM / A, B, C (10,10)
  DATA A /1.0/, B /2.0/ C /100 * 0.0/
END BLOCK DATA WORK
```

而块数据程序单元：

```
BLOCK DATA EXMP
  IMPLICIT NONE
  COMMON / X / A, B, C
  REAL A, B
  DATA A /1.0/
END BLOCK DATA EXMP
```

的公用块中有的对象没被说明，所以是错误的。

小　　结

在本章中，我们介绍了三种基本的程序单元，即主程序、辅程序（包括外部函数辅程序和外部子程序辅程序）、和块数据程序单元。介绍了它们的语法结构、语义、各种限制条件以及注事项等，并给出了一些必要的例子以说明所讲的内容。

主程序是整个 Fortran 程序的一个主控程序部分。主程序在一个程序中必须有一个，而且只能有一个。一个可执行程序从主程序单元开始起动，它可不时地从中调用其它外部的或内部的辅

程序和外部过程执行，或进行各种输入输出，或进行各种加工处理，并用各种控制语句控制整个程序的执行流程。

在 Fortran 中，把用 Fortran 程序设计语言编写的可供程序中多次调用的过程称为辅程序(subprogram)。辅程序有内部辅程序和外部辅程序两类。其中外部辅程序是 Fortran 的一种程序单元。它又可分为外部函数辅程序和外部子程序辅程序两种，分别用来定义一个(或若干个)函数或子程序。由于它们是在主程序之外定义的，故又称外部函数或外部子程序，以便区别于在程序单元内部由内部辅程序定义的内部函数和内部子程序。由于辅程序的引入，在 Fortran 中提供了一种共享或重用部分程序代码的手段，也使得程序的模块性更好。因此，辅程序(包括外部辅程序和内部辅程序)对简化 Fortran 的编程序具有重要意义。

内部过程是一个由内部辅程序定义的过程。内部过程可以出现在主程序、外部辅程序和模块辅程序中。但是内部过程一定不能出现在其它的内部过程中。因此，Fortran 90 仍不是一种过程嵌套式的语言，而基本上仍还是块结构的。不过值得注意，一个内部过程除了它的名不是全局名、其中不能包含 ENTRY 语句、内部过程名不能作为与虚过程结合的实元、内部过程不能再包含其它内部过程以及内部过程可以通过宿主结合存取其宿主的各种实体之外，它的语法和语义与外部过程是相同的。所以我们对此作了统一介绍。

块数据程序单元是一种为有名公用块中的数据对象提供初值的一种程序单元。这对有些应用是必须的。

上面已述的三种程序单元是在 Fortran 77 中已经有的，是用 Fortran 语言编程序的主要语言成分。Fortran 90 新增加的模块程序单元将在第三篇中介绍。

习　题

1. 设 A 是一个长度为 n、B 是一个长度为 m 的两个字符串，n

>m，编写一个逻辑型的函数辅程序，判定 B 是否包含在 A 中。

2. 编写一个子程序辅程序，用以计算一个圆形、一个正方形和一个正三角形的面积之和。要求 A、B 和 C 分别表示圆的半径、正方形和正三角形的边长以及面积 S 作为过程的虚元，并且三种图形的面积分别用三个内部辅程序计算。

3. 先给三个一维的数组 A、B 和 C 赋初值，分别表示圆、正方形和正三角形的半径和边长，然后用 2. 中编写的子程序辅程序计算一组面积放入数组 S。

4. 分别编写矩阵加法、矩阵减法和矩阵乘法的子程序辅程序。编一个从外部读入 N×N 的矩阵 A、B 和 C 的值，并计算 (A+B) * C－B 的 Fortran 程序。

第十章 过　　程

过程是在程序的执行中可被直接调用的封装在一起的一个进行计算或加工的指令或语句序列，它是任何一种过程型程序设计语言的重要组成部分，Fortran 语言也不例外。上一章已对用 Fortran 编写的辅程序过程的语法和语义进行了讨论，本章将对 Fortran 中的各种过程的分类、定义方法和引用方式等作进一步的介绍。

第一节　过程的分类

在 Fortran 90 中，过程可按其定义方法或引用方式来分类。下面分别来叙述。

一、按引用方式分类

一个过程的定义指明了它或者是一个函数或者是一个子程序。它们的引用方式随之也各不相同，对一个函数的引用或者作为一个初等量明显地出现在表达式中，或者隐含地作为表达式中一个由程序员定义的运算符；对一个子程序的引用或者是用 CALL 语句的调用，或者作为一个由程序员定义的赋值语句。此外，一个可按元素(elementally)引用的内在过程(见第十一章)被分类为元素的内在过程。在一个元素的内在函数的引用中，一个或多个实元是数组并且所有这些数组具有相同的形状，则这种函数引用的结果与数组变元具有相同的形状，而且结果中的各个元素就是用其标量实元以及实元数组的对应元素计算该函数来求得。

二、按定义方式分类

按一个过程的定义方式可将一个过程分类为内在过程、外部过程、模块过程、内部过程、虚过程、以及语句函数等六种。

1. 内在过程

内在过程是作为处理系统的一个固有部分而提供的一种过程。程序员无需自己定义即可引用它们。我们将在第十一章中详细介绍。

2. 外部过程

外部过程是一个由外部辅程序定义的或用非 Fortran 语言编写的过程。外部辅程序又分为外部函数辅程序和外部子程序辅程序两种。它们是 Fortran 语言的两种程序单元。外部辅程序的定义形式、语义解释和限制等已在第九章中介绍。

值得注意，如果一个外部辅程序包含了一个或多个 ENTRY 语句，则除了由 SUBROUTINE 语句或 FUNCTION 语句定义的外部过程之外，每个 ENTRY 语句都各定义一个外部过程。

3. 内部过程

内部过程是一个由内部辅程序定义的过程。内部过程可以出现在主程序、外部辅程序和模块辅程序中。但是内部过程一定不能出现在其它的内部过程中。即内部过程不能再嵌套内部过程。不过，一个内部过程除了它的名不是全局名、其中不能包含 ENTRY 语句、内部过程名不能是与虚过程结合的自变量以及内部过程可以通过宿主结合访问其宿主的各种实体之外，它的语法和语义与外部过程是相同的。所以，在上一节中我们并没有另外再详细介绍内部辅程序的结构和含义，而是把它与外部辅程序放在一起介绍的。

4. 模块过程

模块过程是一个由模块辅程序定义的过程。我们将在第三篇中详细介绍。

5. 虚过程

一个指明为过程的虚元或作为过程名出现在过程引用中的虚元称为一个虚过程。例如，我们要编一个近似求定积分

$$\int_a^b f(x)dx$$

的程序。若采用将积分区间分成 n 段的梯型公式来计算：

$$Tn=h*[f(a)/2+f(a+h)+...+f(b-h)+f(b)/2]$$

则可用下列程序：

```
FUNCTION INTEGRAL(F, A, B, N) RESULT(INT_RES)
  IMPLICIT NONE
  REAL::INT_RES
  INTERFACE
    FUNCTION F(X)
    REAL::F, X ! F 是默认的结果变量
  END INTERFACE
  REAL, INTENT(IN)::A, B
  INTEGER, INTENT(IN)::N
  REAL::X, H, SUM
  INTEGER::I
  H=(B-A)/N
  SUM=(F(A)+F(B))/2.0
  DO I=1, N-1
    SUM=SUM+F(A+I*H)
  END DO
    INT_RES=H*SUM
  END FUNCTION INTEGRAL
```

在上述 INTEGRAL 子程序中，虚元 F 被一个接口块(见第十五章)说明为一个过程，因此，就是一个虚过程。在调用该过程时，必须对该虚元提供一个另外定义的函数过程或某些内在函数过程作为实元与之结合。

6. 语句函数

语句函数是用单个语句定义的函数。其一般形式为：

函数名([虚元名表])=标量表达式

其中的标量表达式只可用常量、对标量变量或数组元素的引用、对函数或虚函数过程的引用和内在运算符组成。

如果在标量表达式中出现对一个语句函数的引用，则这个语句函数必须是在该作用域单位中的前面部分已被定义过的，而且不能引用它本身。

标量表达式中出现的有名常量必须是在该作用域单位中的前面部分已被声明过的，或者是在使用结合或宿主结合后可访问的。同样，若标量表达式中出现数组元素，则其数组名必须在该作用域单位中的前面部分已经声明过的，或者是在通过使用结合或宿主结合后可访问的。

在一个标量表达式中，如果一个标量变量、数组元素、函数引用、或虚函数引用的类型是由隐含类型规则确定的，则它们在后续的类型语句中的出现必须确认这种隐含类型和隐含类型参数值。

语句函数的函数名和各虚元名都必须显式地或隐含地被说明为一些标量数据对象。一个虚元名只能在虚元名表中出现一次。

标量表达式中的各个标量变量引用可以是对其虚变元的引用，也可是对该语句函数所在作用域单位中的一个局部变量的引用。

如果一个语句函数名与其宿主中可访问的一个实体的名相同，则该语句函数的定义必须出现在相应实体名的一个类型语句的类型声明之后。

语句函数的虚元的作用域是该语句函数语句。各虚元的类型和类型参数与包含该语句函数的作用域单位中的同名实体的类型和类型参数相同。

语句函数一定不能作为过程实在变元。

一个语句函数引用的值是在用实元的值替换相应的虚元之后计值其标量表达式而得到的，必要时可能还需要将所得结果转换成该函数声明的类型和类型属性。

在标量表达式中的函数引用一定不能导致语句函数的虚元的

重定义或不定义。

第二节　过程的特性

一个过程的特性包括将过程分为函数或子程序的分类特性和它的虚元的特性，如果它是一个函数，那么还包括其结果的特性。

一、虚元的特性

一个虚元可以是一个虚拟数据对象、一个虚过程、一个作为选择返回指示符的星号。当一个虚元不是星号时，它可有 OPTIONAL 属性，表示在对该过程引用时不需要有一个实元与该虚元结合。下面我们分别叙述三类虚元的特性：

1. 虚拟数据对象的特性

一个虚拟数据对象的特性包括它的类型、类型参数（如果有的话）、形状、输入或输出意向（INTENT）、它是否任选的（OPTIONAL）、是否一个指针（POINTER）或目标（TARGET）。此外，如果一个对象的类型参数或一个数组的界是一个依赖于另一个对象的值或属性的表达式，则这种对其它实体的精确依赖关系也是该虚对象的一个特性。如果一个形状、大小、或字符的长度是需在引用时僭取的（assumed），这也是一个特性。

2. 虚过程的特性

一个虚过程的特性包括其接口（见第十五章）是否明显给出的、它作为一个过程的特性（如果其接口是显式给出的）、以及它是否任选的等三项。

3. 星号虚元的特性

星号虚元没有特性。

二、函数结果的特性

函数结果的特性包括它的类型、类型参数（如果有的话）、秩、以及它是否一个指针等特性。此外，如果函数结果是一个非指针

数组，它的形状也是一个特性。当一个类型参数或一个数组的界不是一个常量表达式时，则体现在该表达式中的对其它实体的精确依赖关系也是一个特性。并且，如果一个字符数据对象的长度是在引用时僭取的，那么这也是一个特性。

第三节　过程的定义

在 Fortran 中，过程可用多种方式定义，包括由系统内部定义的内在过程、由辅程序定义的过程、由语句函数定义的函数过程以及用其它语言编写的过程等。

1. 内在过程

内在过程是作为 Fortran 处理系统的一个固有部分在内部预先定义的，可在 Fortran 程序中直接引用，而无需用户另行定义。一个与标准相符的 Fortran 处理系统必须包含第十一章中描述的内在过程，但可以包含其它外加的内在过程。然而，一个符合标准的程序不能使用这些外加的内在过程。

Fortran 90 的内在过程很多(共有 113 个)，我们将在第十一章中专门描述。

2. 由辅程序定义的过程

用一个辅程序可以定义一个(外部辅程序可定义多个)函数或子程序，它们或者是可在整个程序中被调用的外部过程，或者是仅局限于在一个程序单元中被调用的内部过程。外部辅程序过程用一个以 FUNCTION 语句或 SUBROUTINE 语句开头的程序单元来定义，也可用程序单元中的一个 ENTRY 语句定义。这些我们已在第九章中作过介绍。

一个辅程序过程被调用时，首先要建立一个该过程的实例，然后再执行之。各个实例各有一个独立的执行序列、一个独立的虚元集合以及一些局部的非保留的数据对象。过程的执行从该过程的 FUNCTION 语句、SUBROUTINE 语句或 ENTRY 语句之后的第一个可执行构造开始，如果没有任何可执行构造，则从 END

语句开始。

如果一个包含在某辅程序中的内部过程或语句函数直接在该辅程序的一个实例中被调用，或在可见该实例的一些实体的内部过程或语句函数中被调用，那么对该内部过程或语句函数建立的实例也可访问其宿主辅程序的实例中的实体。

辅程序的所有其它实体可由辅程序的所有实例共享。例如，一个实例中保留的数据对象的值或许是由前一实例中定义的，也许是由一个 DATA 语句或类型说明语句给的初值。

在过程定义中，虚拟数据对象的 INTENT 属性用来限制它们在过程体中的使用方式。一个有属性 INTENT(IN)的虚拟数据对象一定不能在过程体中被辅程序定义或再定义。一个有属性 INTENT(OUT)的虚拟数据对象在辅程序中的最初状态是不定义的，然后可在辅程序的执行中被定义和再定义。一个有属性 INTENT(INOUT)的虚拟数据对象既可被引用也可被定义。一个没有 INTENT 属性的虚拟数据对象受制于与之结合的实元数据实体的属性，即由其相应的实元决定。也就是说，如果实元是已定义的，它在辅程序中就可被引用；如果实元是可定义的，它在辅程序中也可被定义。

关于辅程序的结构我们已在第九章中介绍，其它语句在辅程序中的使用已在介绍各语句的章节中分别叙述，在此不再赘述。

3. 递归过程

通常，一个辅程序不可直接或间接通过一个调用序列调用其本身。但是，如果在函数语句或子程序语句之前冠以 RECURSIVE，则它就可直接或间接地调用自己。例如，可把计算阶乘的过程编成一个递归函数：

```
RECURSIVE FUNCTION FACTORIAL(N) RESULT(RES)
  INTEGER RES, N
  IF(N.EQ.1) THEN
    RES = 1
  ELSE
```

```
    RES = N * FACTORIAL(N-1)
  END IF
END FUNCTION FACTORIAL
```

递归过程每调用一次就建立一组局部数据对象，放在一个先进后出的"栈"的顶部；每当返回时，就从栈顶上去掉一组数据对象。这样就可有条不紊地处理在没有完全退出递归过程之前又需要进入它自身所带来的困难。

递归过程是简洁地构造一个程序的一种强有力的工具，如果没有这种设施可能会把程序编得很复杂。下面我们再来介绍著名的汉诺塔的例子。有三根柱子，其中 1 号柱上堆放有一组从大到小的圆盘(即所谓汉诺塔)，今要将圆盘搬到 3 号柱上去，可以利用 2 号柱，但是一次只能搬一个盘子而且永远不能出现有大盘压在小盘上的情形发生。这样一个较复杂的问题可用下列递归的辅程序来实现：

```
RECURSIVE SUBROUTINE HANOI (NUMBER_OF_DISKS, &
                       STARTING_POST, GOAL_POST)
  IMPLICIT NONE
  INTEGER :: NUMBER_OF_DISKS
  INTEGER :: STARTING_POST, GOAL_POST, &
        FREE_POST
  INTEGER, PARAMETER :: ALL_POSTS = 6
  FREE_POST := ALL_POSTS - STARTING_POST - &
      GOAL_POST
  IF (NUMBER_OF_DISKS > 1) &
    CALL HANOI (NUMBER_OF_DISKS - 1, &
              STARTING_POST, FREE_POST)
  PRINT *, "MOVE DISK", NUMBER_OF_DISKS, &
        "FROM POST", STARTING_POST, &
        "TO POST", GOAL_POST
  IF (NUMBER_OF_DISKS > 1) &
      CALL HANOI (NUMBER_OF_DISKS - 1, &
                FREE_POST, GOAL_POST)
```

```
END SUBROUTINE HANOI
```

如果用CALL HANOI (4，1，3)调用，即将由四个盘组成的汉诺塔从 1 号柱搬到 3 号柱，则将输出如下指令：

将盘 1 从 1 号柱搬到 2 号柱
将盘 2 从 1 号柱搬到 3 号柱
将盘 1 从 2 号柱搬到 3 号柱
将盘 3 从 1 号柱搬到 2 号柱
将盘 1 从 3 号柱搬到 1 号柱
将盘 2 从 3 号柱搬到 2 号柱
将盘 1 从 1 号柱搬到 2 号柱
将盘 4 从 1 号柱搬到 3 号柱
将盘 1 从 2 号柱搬到 3 号柱
将盘 2 从 2 号柱搬到 1 号柱
将盘 1 从 3 号柱搬到 1 号柱
将盘 3 从 2 号柱搬到 3 号柱
将盘 1 从 1 号柱搬到 2 号柱
将盘 2 从 1 号柱搬到 3 号柱
将盘 1 从 2 号柱搬到 3 号柱

下面我们再来讨论与编写辅程序过程有关的几个语句和属性。

4. EXTERNAL 语句

EXTERNAL 语句用来说明一些名具有 EXTERNAL(外部)属性，即由其说明的名或者表示一个外部过程、或者一个虚过程、或者一个块数据程序单元。EXTERNAL 语句的形式为：

EXTERNAL 外部名表

其中每个外部名必须是一个外部过程名、或者一个虚过程名、或者一个块数据程序单元名。

一个虚元名若出现在 EXTERNAL 中，则说明该虚元是一个虚过程。如果一个外部过程名或虚过程名要被用作实元，则它必须出现在一个 EXTERNAL 语句中，或被该作用域单位中的一个类型语句指明具有 EXTERNAL 属性，或在该作用域单位中被一

个接口块声明为一个过程。例如，在区间[A,B]上求函数 FUNC 的极小的函数辅程序中，函数 FUNC 可作为一个虚元，并用 EXTERNAL 语句来指明它：

```
REAL FUNCTION MINIMUM(A, B, FUNC)
  EXTERNAL FUNC
  REAL, INTENT(IN) :: A, B
  REAL F, X
       ......
  F = FUNC(X) ! 调用用户定义的函数 FUNC
       ......
END FUNCTION MINIMUM
```

这还可用给出一个显式的接口块来代替 EXTERNAL 语句，把上述程序变成：

```
REAL FUNCTION MINIMUM(A, B, FUNC)
  REAL, INTENT(IN) :: A, B
  INTERFACE
    REAL FUNCTION FUNC(X)
      REAL, INTENT(IN) :: X
    END FUNCTION FUNC
  END INTERFACE
  REAL F, X
       ......
  F = FUNC(X) ! 调用用户定义的函数 FUNC
       ......
END FUNCTION MINIMUM
```

或用一个说明语句来说明 FUNC 具有 EXTERNAL 属性，把程序改成：

```
REAL FUNCTION MINIMUM(A, B, FUNC)
  REAL, EXTERNAL :: FUNC
  REAL, INTENT(IN) :: A, B
  REAL F, X
     ......
```

```
  F = FUNC(X) ! 调用用户定义的函数 FUNC
    ……
END FUNCTION MINIMUM
```

上述函数辅程序可被下列主程序来调用，并与一个具体计算函数 FUN 的辅程序一起构成一个完整的 Fortran 程序：

```
PROGRAM MAIN
  REAL F
  EXTERNAL FUN
    ……
  F=MINIMUM(1.0, 2.0, FUN)   ! 用户定义的函数 FUN 作实元
    ……
  END PROGRAM
  REAL FUNCTION FUN(X)
    REAL, INTENT(IN) :: X
      ……                  ! 计算 FUN(X)的值
END FUNCTION FUN
```

其实还可采用下列两种程序形式来实现同样的目的：

```
PROGRAM MAIN
  REAL F
  INTERFACE
    REAL FUNCTION FUN(X)
      REAL, INTENT(IN) :: X
    END FUNCTION FUN
  END INTERFACE
      ……
  F=MINIMUM(1.0, 2.0, FUN)   ! 用户定义的函数 FUN 作实元
      ……
  END PROGRAM
  REAL FUNCTION FUN(X)
    REAL, INTENT(IN) :: X
    ……                  ! 计算 FUN(X)的值
  END FUNCTION FUN
```

或

```
PROGRAM MAIN
  REAL F
  REAL, EXTERNAL :: FUN
    ……
  F=MINIMUM(1.0, 2.0, FUN)   ! 用户定义的函数 FUN 作实元
    ……
END PROGRAM

REAL FUNCTION FUN(X)
  REAL, INTENT(IN) :: X
  ……                  ! 计算 FUN(X)的值
END FUNCTION FUN
```

如果一个内在过程名被列入 EXTERNAL 语句中，则该名将被用作一个外部辅程序的名，而同名的内在过程在该作用域单位内就不能再用了。而且还要指出，在一个作用域单位中，一个名只能在所有 EXTERNAL 语句中出现一次。

5. INTRINSIC 语句

INTRINSIC 语句用来说明一些名具有 INTRINSIC(内在)属性，即由其说明的名表示一个内在过程。INTRINSIC 语句的形式为：

INTRINSIC 内在过程名表

其内在过程名表中的名必须是内在过程的名，既可是内在过程的特定名，也可是内在过程的类属名。一个在 INTRINSIC 语句出现的内在过程的特定名(注意不能是类属名)可在其作用域单位中作为一个过程的实元。而且，如果一个内在过程名的特定名要被用作实元，则它必须出现在一个 INTRINSIC 语句中，或被该作用域单位中的一个类型声明语句指明具有 INTRINSIC 属性。而且还要指出，在一个作用域单位中，一个内在过程名只能在所有 INTRINSIC 语句中出现一次，并且不能同时出现在 INTRINSIC 语句和 EXTERNAL 语句中。例如：

```
PROGRAM MAIN
```

```
REAL F
INTRINSIC ALOG
  ……
F=MINIMUM(0.0, 1.0, ALOG) ! 内在对数函数 ALOG 作实元
  ……
END PROGRAM
```

注意，对数函数名必须用特定名 ALOG，而不能用类属名 LOG。其实也可用下列程序实现：

```
PROGRAM MAIN
REAL F
REAL, INTRINSIC :: ALOG
  ……
F=MINIMUM(0.0, 1.0, ALOG) ! 内在对数函数 ALOG 作实元
  ……
END PROGRAM
```

6. 程序的停止执行

STOP 语句用在主程序或辅程序中用来停止程序的执行。虽然一般总是让程序停在主程序中，但也不排除在辅程序中使用。为了区分一个程序中的几个 STOP 语句，可在 STOP 语句中附带标签。例如：

```
STOP 1
STOP 2
```

或

```
STOP 'Incomlete data, program terminated'
```

此外，主程序单元中的 END 语句也有停止程序执行的作用。

7. PRIVATE 和 PUBLIC 属性

在过程定义时，可用 PRIVATE 属性来防止一些数据对象被其它程序单元调用，而只供自己私用。例如，在模块中可把一些对象说明成 PRIVATE，从而把它的具体实现隐蔽起来，使外面不可见。于是，内部实现的改变将不会影响别的过程对它的使用。下面就是一个私用的类型定义的例子：

```
TYPE MATRIX
  PRIVATE
  INTEGER :: M, N ! M×N 的矩阵
  REAL, DIMENSION (:, :), ALLOCATABLE :: A
END TYPE MATRIX
```

PUBLIC 属性正好相反，它说明对象是公用的。这两种属性既可在类型语句中指明，也可用 PRIVATE 语句和 PUBLIC 语句来指明。在没出现 PRIVATE 语句的程序单元中默认是 PUBLIC 的。这种默认状态可用 PRIVATE 语句或 PUBLIC 语句来重置。

8. 程序单元中语句的次序

程序单元中语句的次序可用表 10—1 来指明。

表 10—1

<table>
<tr><td colspan="3">PROGRAM, FUNCTION, SUBROUTINE 或 MODULE 语句</td></tr>
<tr><td colspan="3">USE 语句</td></tr>
<tr><td rowspan="4">FORMAT
语句</td><td colspan="2">IMPLICIT NONE 语句</td></tr>
<tr><td>PARAMETER 语句</td><td>IMPLICIT 语句</td></tr>
<tr><td>PARAMETER 语句
和
DATA 语句</td><td>导出类型语句
接口块
类型声明语句
和　说明语句</td></tr>
<tr><td>DATA 语句</td><td>可执行语句或构造</td></tr>
<tr><td colspan="3">CONTAINS 语句</td></tr>
<tr><td colspan="3">内部辅程序
或
模块辅程序</td></tr>
<tr><td colspan="3">END 语句</td></tr>
</table>

第四节　过程的引用

过程引用又称过程调用。建立过程的目的就是为了使得一些

需要多次重复执行的功能独立地编写成一些程序段，即所谓过程，以便多次地调用它们。一般，过程不论是函数还是子程序都可以带参数，在 Fortran 中把参数称为变元(实元和虚元)。所以，在调用一个过程时首先要用当时的实元替换相应的虚元，即所谓引用结合，然后才能转入过程执行。

一、过程的引用形式

过程的引用形依赖于过程的接口(见第十五章)，而并不依赖于过程定义的方法。过程引用分为函数引用和子程序引用两种形式。函数引用的形式为：

函数名（〔实元说明表〕）

子程序的引用一般采用 CALL 语句进行，CALL 语句的形式为：

CALL 子程序名〔(〔实元说明表〕)〕

此外，函数还可作为一个用户定义的运算符引用，子程序还可作为一个用户定义的赋值来引用。

二、实元说明表

过程引用中的实元说明表用来说明调用该过程时的各实元。它是一个用逗号隔开的实元说明的序列。每个实元说明具有如下形式：

〔关键词 ＝〕实元

其中，关键词就是相应虚元的名，实元可以是一个变量、一个表达式、一个过程名，或者一个选择返回说明，即一个：

* 标号

在函数引用的实元说明表中一定不能包含选择返回说明。

如果过程接口在作用域单位中是隐含的，则任选项"关键词＝"一定不能出现在实元说明中。任选项"关键词＝"仅当在实元说明表中在它前面的实元说明中都已省略了任选项"关键词＝"时才能被省略掉。

每个关键词必须是过程的显式接口中的一个虚元的名。

如果实元是一个过程名，则它一定不能是一个内部过程或语句函数的名，而且也不能是一个内在过程的类属名(见第十一章)。

一个选择返回说明的标号必须是该CALL语句的作用域单位中的一个分支目标语句的一个语句标号。

三、变 元 结 合

在过程引用时，实元说明表指明了所提供的实元与过程的虚元之间的对应关系，或称结合关系。在没有指明任选项"关键词="时，一个实元与一个在虚元表中与之具有相同位置的虚元相结合，即在过程引用时用一个实元替换其相应的虚元。详细说就是，第一个实元与虚元表中的第一个虚元结合，第二个实元与虚元表中的第二个虚元结合，如此等等。如果实元说明中有一个关键词出现，则应将该实元与和该关键词同名的虚元结合。一个非任选的虚元必须恰好有一个实元与之结合。每个任选的虚元至多只能有一个实变元与之结合，即允许没有任何实元与它结合。例如，子程序过程：

```
SUBROUTINE SOLVE (FUNCT, SOLUTION, METHOD,&
                  STRATEGY, PRINT)
  INTERFACE
    FUNCTION FUNCT (X)
      REAL FUNCT, X
  END INTERFACE
  REAL SOLUTION
  INTEGER, OPTIONAL :: METHOD, STRATEGY, PRINT
    ......
END SUBROUTINE SOLVE
```

倘若在一个程序单元中上述程序的接口是显式给出的，则可用CALL语句：

```
CALL SOLVE (FUN, SOL, PRINT=6, METHOD=3)
```

或

CALL SOLVE (FUN, SOL)

来调用。由此可见，在调用时与任选虚元 METHOD，STRATEGY 和 PRINT 对应的实元都可省缺。

一当确定了实元与虚元的上述对应关系之后，在调用一个过程时必须首先进行具体的变元结合工作。具有不同特性的变元有不同的结合方式，下面将分别来叙述：

1. 虚拟数据对象的变元结合

当过程的一个虚元是一个虚拟数据对象时，与虚拟数据对象结合的实元必是一个同类型的数据对象或一个同类型的表达式。实元的种别类型参数值必须与虚元的一致。一个非默认字符类型的实元的长度类型参数的值也必须与相应虚元的一致。此外，如果虚元是一个默认字符类型的僭取形数组，则实元长度类型参数的值也必须与相应虚元的一致。

一个标量虚元只能与一个标量实元相结合。

如果一个标量虚元是一个默认字符类型的，则其长度 len 必须小于等于相应实元的长度，并约定虚元总是与相应实元的最左面的 len 个字符结合。如果一个数组虚元是默认字符类型的，则关于长度的限制是对整个数组而言的，而非对各数组元素而言的。即虚元数组中某个数组元素的长度可以同与之结合的实元数组的数组元素、实元数组元素、或实元数组元素子串的长度不同，但是虚元数组的总长度一定不能超过实元数组的总长度。

除非调用元素的过程(见第十一章)，否则一个数组值的实元的诸元素，或一个序列结合中的一个序列的诸元素将同按数组元素次序(见第五章)具有相同位置的虚元数组元素结合。

2. 指针虚元的变元结合

如果虚元是一个指针，则实元必须也是指针，而且其类型、类型参数和秩必须与虚元的一致。

在过程调用时，虚元指针接受实元的指针结合状态。如果实元指针当前是与一个目标结合的，那么虚元就变成是与同一个目标结合的。但是结合状态可在过程执行中改变。在过程执行完成

时，如果虚元指针是与一个具有 TARGET 属性的过程的虚元结合的，或者是与一个不定义的目标结合的，则该虚元指针的结合状态变成不结合的；随之，实元指针的结合状态变成虚元指针的结合状态。

如果实元具有 TARGET 属性，则在过程调用时与该实元结合的任何指针将不变成与相应的虚元结合的，但仍保持与该实元的结合。如果虚元具有 TARGET 属性，则当过程执行完成时任何与之结合的指针变成不定义的。

如果实元是一个标量，除非该实元是一个非假取形数组或指针数组的元素，或者这种元素的一个子串，则相应的虚元也必须是标量。如果过程是用类属名、用户定义的运算符、或用户定义的赋值来引用的，则实元与相应虚元的秩必须一致。

如果虚元具有 INTENT(OUT)或 INTENT(INOUT)，则其实元必须是可定义的。如果虚元具有 INTENT(IN)，则相应的实元在变元结合被建立时变成不定义的。

如果实元是一个具有向量下标的数组片段，则虚元是不可定义的，并且一定没有 INTENT(OUT)或 INTENT(INOUT)。

3. 假取形数组虚元的变元结合

如果虚元是一个假取形数组，则其实元必须不是一个假取大小数组，也不能是一个标量(包括数组元素指定符和数组元素子串指定符)。

4. 过程的变元结合

如果虚元是一个虚过程，则与之结合的实元必须是外部过程、模块过程、另一虚过程或内在过程的一个特定名。而且，只有那些在第十一章中所列的且未标圆点(.)的内在过程的特定名才允许作为实元。如果一个特定名也是类属名，那么只有特定内在过程名才可与该虚元结合。

如果虚过程的接口(见第十五章)是显式给出的，则与之结合的实元过程的特性必须与该虚过程的特性相同。

如果虚过程的接口(见第十五章)是隐式给出的，并且虚过程

名是显式地被指明类型的或者在过程体中该虚过程作为一个函数引用，则该虚过程一定不能作为子程序来引用，并且相应的实元必须是一个函数或虚过程。

如果虚过程的接口（见第十五章）是隐式给出的，并且在过程体中该虚过程作为一个子程序引用，则相应的实元必须是一个子程序或虚过程。

与一个虚过程结合的实元必须用 EXTERNAL 语句或 INTRINSIC 语句说明，或者用说明语句说明其属性为 EXTERNAL 或 INTRINSIC 的，或者给出一个接口来显式地说明（见关于 EXTERNAL 语句和 INTRINSIC 语句的介绍）。

5. 选择返回指示符的变元结合

如果虚元是一个星号（*），则与之结合的实元必须是一个选择返回指示符：

* 标号

其中的标号必须是包含该过程引用的作用域单位中的一个执行结构的标号。

6. 序列结合

如果实元是一个数组表达式、数组元素指定符、或数组元素子串指定符，那么它就表示一个元素序列。如果实元是一个数组表达式，则元素序列由该数组的元素按数组元素的顺序构成。如果实元是一个数组元素指定符，则元素序列由该数组元素以及按数组元素的顺序跟在其后的元素构成。

如果实元是默认字符类型的并且是一个数组表达式、数组元素、或数组元素子串指定符，则元素序列由该实元的第一个存储单元开始直到该数组末尾的所有字符存储单元构成。一个数组元素子串指定符的字符存储单元被看成一些数组元素，它们由一组具有其虚元的字符长度的相继的字符存储单元构成。注意，在此元素序列中的某些元素可能由原数组的不同元素的存储单元组成。

如果虚元是一个显形数组或僭取大小数组，则与该虚元结合

的序列是一个表示元素序列且对应于一个数组值数据对象的实元。该实元的秩和形状不需要与相应的虚元一致，但虚元中元素序列的个数一定不能超过相应实元的元素序列中的元素个数。如果虚元是假定大小的，虚元中元素的个数必须正好与实元的元素序列的元素个数相同。

四、函数引用

一个函数过程是在表达式的计值过程中通过函数引用或用户定义的运算符来调用的。当它被调用时，首先对所有实元表达式进行计值，然后进行变元结合，然后再执行该函数过程的体。一当函数执行完成时，返回函数结果的值供调用它的表达式中使用，这时函数结果的特性由该函数的接口决定。

正如前述，一个函数的引用具有形式：

函数名(〔实元说明表〕)

所以

```
FUN0(A,B,C)
FUN1( )
FUN2(Y=B,X=A,Z=C)
```

等都是合法的函数引用的例子。作为用户定义的运算符的函数引用方式将在第十五章中介绍。

五、元素内在函数的引用

如果在一个内在函数的引用中，一个或多个实元是数组并且所有这些数组具有相同的形状，则这种引用是一种元素的引用。如此计值的结果与数组变元具有相同的形状，而且结果中的各个元素是用其标量实元和实元数组的对应元素计算该函数的值而求得的。例如，若 X 和 Y 是形状为(m,n)的数组，则：

MAX (X, 0.0, Y)

是一个形状为(m, n)的数组，其元素有值：

MAX (X(i,j), 0.0, Y(i,j)), i=1,2,……,m, j=1,2,……,n

六、子程序引用

一个子程序过程是通过执行一个 CALL 语句或用户定义的赋值(见第十五章)来调用的。当它被调用时，首先对所有实元表达式进行计值，然后进行变元结合，然后再执行该子程序过程的体。一当由该子程序指明的动作执行完成时，相应的 CALL 语句或用户定义的赋值的执行也就完成了。如果一个 CALL 语句在其实元中包含一个或多个选择返回指明符，则根据子程序指明的动作可将控制转移到返回指明符指明的语句之一去。

正如前述，一个子程序的 CALL 调用具有形式：

CALL 子程序名〔(〔实元说明表〕)〕

所以

```
CALL SUB1
CALL SUB2 ( )
CALL SUB3 (A, B, C)
CALL SUB4 (Y=B, Z=C, X=A)
CALL SUB5 (A, Z=C, Y=B)
```

等都是合法 CALL 语句的例子。作为用户定义的赋值的引用方式将在第十五章中介绍。

七、元素内在子程序的引用

如果在一个内在子程序的引用中，所有对应于 INTENT(OUT)和 INTENT(INOUT)的虚元的所有实元都是具有相同形状的数组，并且余下的其它实元也与它们形状相同，则这种引用是一种元素的引用。而且与 INTENT(OUT)和 INTENT(INOUT)的虚元对应的那些数组元素的值由分别地将该子程序作用于各实元的相应数组元素而求得。

小　　结

在本章中，我们对 Fortran 的过程的有关问题作了详细介绍。

首先根据引用方式和定义方式对过程作了分类。接着讨论了过程的特性，包括其虚元的特性和函数结果的特性。在第十章一般讨论的基础上，进一步讨论了各种过程定义中的一些细节问题，包括内在过程的定义、辅程序过程的定义、递归过程的定义、EXTERNAL 语句、INTRINSIC 语句、PRIVATE 属性、PUBLIC 属性、程序的停止执行以及过程中语句次序的总结等。此外，本章还详细地讨论了过程的引用有关的问题，包括各种过程的引用的形式、各种变元结合的细节和限制、以及过程被引用时的执行顺序等。过程是 Fortran 语言的一种很重要的语言成分，在 Fortran 编程序中起着重要作用。关于内在过程的定义和细节将在第十一章中详细介绍。Fortran 90 的新成份模块过程的定义和细节将在第十五章中详细介绍。

习　题

1. 按引用方式分类，过程可分成哪几类？过程的特性包括哪些内容，请详细叙述，并举例说明。

2. 试编一个以一个字符串作为自变量的函数过程，它回送字符串自变量中按字符顺序最早出现的一个字符作为函数值。例如，Fortran 的函数值为 A。

3. 试编一个用简单消去法求解线性方程组的子程序过程。

4. 先编制矩阵的加法和乘法作为两个内部辅程序，然后编一个主程序计算矩阵表达式：

$$A+B\times(C+D\times E)$$

5. 叙述 PUBLIC 和 PRIVATE 属性在编制过程时的实际用途。

第十一章　内 在 过 程

内在过程是在系统内部预先定义的过程，程序员无需自己定义就可在其程序中调用。有四类内在过程：查询函数、元素函数、变换函数和子程序。在内在子程序中只有一个是元素子程序。本章将详细介绍各种内在过程，包括介绍它们的被调用的形式、功能描述、分类、变元和结果的类型、类型参数与形状等。

第一节　内 在 函 数

在 Fortran 90 中有 113 个内在过程，其中大部分是内在函数，完成种类繁多的各种计算、查询和转换等功能。

一、内在函数的分类

内在函数分为查询函数、元素函数和变换函数三种：

(1)查询函数 查询函数的结果只依赖于它的主要变元的性质而与其值无关。其实，该变元的值甚至可以是未定义的。

(2)元素函数 元素函数是一种对标量定义的函数，但它可作用于数组变元，其结果是分别对数组中的元素作用该函数而得到的数组。

(3)变换函数 除上述两种函数之外的内在函数都是变换函数。几乎所有变换函数都有一个或多个数组值变元和一个数组值结果。

二、内在函数的类属名和特定名

不少内在函数有一个类属名和一个特定名。多数情况下，具有类属名的内在函数的变元可有多种类型，而结果的类型与变元

的类型相同，随着变元类型的变化而变化。但是，具有特定名的内在函数的变元一般只能有一种专门的类型。

值得注意，如果一个内在函数被用作一个过程的实元，那么必须用它的特定名，并且在被调用过程中只能用标量变元来引用它。如果一个内在函数没有特定名，那么它就一定不能作为过程的实元。

三、按功能分类的内在函数

1. 数值函数、数学函数、字符函数、种别函数、逻辑函数和位运算函数

（1）数值的函数。数值的内在函数包括作类型转换的 INT、REAL、DBLE 和 CMPLX，作各种简单数值运算的元素内在函数 AIMAG、CONJG、AINT、ANINT、NINT、ABS、MOD、SIGN、DIM、DPROD、MODULO、FLOOR、CEILING、MAX 和 MIN。它们的具体功能将在以下统一叙述。

（2）数学函数。数学函数指计算各种初等数学的函数 SQRT、EXP、LOG、LOG10、SIN、COS、TAN、ASIN、ACOS、ATAN、ATAN2、SINH、COSH 和 TANH，它们都是元素内在函数。

（3）字符函数。字符函数指作各种字符运算的内在函数 ICHAR、CHAR、LGE、LGT、LLE、LLT、IACHAR、ACHAR、INDEX、VERIFY、ADJUSTL、ADJUSTR、SCAN 和 LEN_TRIM，它们都是元素内在函数，它们的具体功能将在以下统一叙述。变换函数 REPEAT 返回将一个字符串变元重复拼接后得到的结果，变换函数 TRIM 返回将变元的尾部空格去掉后的字符串，它们也是字符的函数。

（4）字符查询函数。字符查询函数 LEN 返回一个字符实体的长度。值得注意，这个函数的变元的值并不需要有定义，此时，如果函数的值可用其它方法确定，该函数的变元的计值是不必要的。

（5）种别函数。查询函数 KIND 返回整型、实型、复型、逻辑型或字符型实体的种别类型参数值。值得注意，这个函数的变元的值

并不需要有定义。变换函数 SELECTED _ REAL _ KIND 返回实型种别类型参数值，它至少具有其变元所指明的十进制精度和指数的范围。变换函数 SELECTED _ INT _ KIND 返回整型种别类型参数值,它至少具有其变元所指明的指数范围。

(6) 逻辑函数。逻辑函数 LOGICAL 实现逻辑型对象和其它不同种别类型参数值之间的变换，它是一个元素函数。

(7) 位运算函数。位运算过程由十个函数和一个子程序组成，位运算子程序将在下面介绍，在此先简单介绍位函数。十个位运算函数中包括四个位上的逻辑运算函数 IOR、IAND、NOT 和 IEOR，两个移位运算函数 ISHFT 和 ISHFTC，三个单位处理函数 BTEST、IBSET 和 IBCLR，以及一个引用位子域的函数 IBITS。

对这些位函数而言，一个"位(bit)"被定义为位于一个非负整型标量对象 j 的第 k 个位置上的一个二进制位 w_k，这里的非负整数 j 由下列模型定义：

$$j = w_0 + w_1 * 2 + \cdots + w_k * 2^k + \cdots + w_s * 2^s \quad (11.\ \mathrm{I})$$

其中 w_k 或为 0 或为 1。例如，定义一个 32 位的整数时，s＝31 。在这个模型中，整型对象由一个从右到左编号为 0 到 s 的位串组成。注意，这种模型只对用作位运算过程的变元和结果的场合有效，在所有其它场合仍应采用在下一段中将要定义的整数模型。

2. 数值操作函数

数值操作函数是一种借助数的表示模型和数在处理系统上的行为来描述的函数。在表示模型中带有参数，以便确定对处理系统最合适的模型，使得可执行程序能在其上有效地运行。为此必须在具体介绍数值操作函数之前介绍整数和实数的表示模型。

(1) 整数和实数的表示模型。一个 r 进制的整数 i 的表示模型为：

$$i = s * (w_1 + w_2 * r + \ldots + w_q * r^{q-1}) \quad (11.\ \mathrm{II})$$

其中，r 是一个大于 1 的整数，q 是一个正整数，w_k (k＝1，2，. . . ，q) 是一些小于 r 的非负整数，s 或为－1 或为＋1 。

一个实数 x 的表示模型为：

$$x = 0$$

或

$$x = s * b^{e} * (f_1 * b^{-1} + f_2 * b^{-2} + \cdots + f_p * b^{-p}) \qquad (11.Ⅲ)$$

其中 b 和 p 是大于 1 的整数；fk(k=1,2,...,p) 是一些小于 b 的非负整数，且 f_1 不为 0；s 或为 0 或为 1；e 是在最大整数 e_{max} 和最小整数 e_{min} 之间的一个整数，包括 e_{max} 和 e_{min} 在内。当 x=0 时，它的指数 e 和数字 f_k 都被定义为 0。

可见，在整数模型中，参数 r 和 q 确定了模型所表示的整数的集合。在实数模型中，参数 b、p、e_{max} 和 e_{min} 确定了模型所表示的实数的集合。这些参数在处理系统实现各种整型或实型数据时必须是已经定好了的。一当确定了这些参数之后，模型所表示的整数或实数的集合也就确定了。例如，本节的函数中所用的模型是：

$$i = s * (w_1 + w_2 * 2 + \cdots + w_q * 2^{30}) \qquad (11.Ⅳ)$$

和

$$x = 0$$

或

$$x = s * 2^{e} * (1/2 + f_2 * 2^{-2} + \cdots + f_{24} * 2^{-24}) \qquad (11.Ⅴ)$$

其中 $-126 \leqslant e \leqslant 127$

本节所述的数值操作和查询函数用来提供与这些参数有关的值和其它与它们有关的常量。下面分别来介绍。

(2) 数值查询函数。数值查询函数 RADIX、DIGITS、MINEXPONENT、MAXEXPONENT、PRECISION、RANGE、HUGE、TINY 和 EPSILON 用来返回与变元的类型和种别类型参数相联的有关模型参数的标量值。这些函数的变元值不需要是定义的，指针变元可以是不结合的，而且数组变元不需是已分配单元的。

(3) 浮点操作函数。浮点操作函数 EXPONENT、SCALE、NEAREST、FRACTION、SET_EXPONENT、SPACING 和 RRSPACING 返回与变元的实际值相联的与模型的参数值有关

的值。它们都是元素函数。

3. 数组内在函数

数组内在函数执行下列作用于数组上的运算：向量和矩阵乘法、数值或逻辑的降秩计算、数组结构查询、数组构造、数组操作和几何位置计算。在介绍这些数组内在函数之前，先来对其变元的性质作一些说明。

(1) 关于变元的一些说明。因为数组变换函数把数组变元都作为整个的数组来进行操作的，所以，相应实元的形状必须有定义。也就是说，实元必须是一个数组片段、一个僭取形数组、一个显形数组、已与一个目标结合的一个指针、一个已分配单元的可分配数组或者一个数组值表达式。它必须不是一个僭取大小数组。

有些内在查询函数也接受不需要有确定形状的数组变元，而可把僭取大小数组用作它们的变元，这些函数包括 LBOUND 以及对 SIZE 和 UBOUND 的某些引用。

有些数组内在函数有一个任选的 MASK 变元，它们被函数用来选择一个或多个变元的一些元素以供运算。没被选上的元素在函数被调用时是无需定义的。

MASK 只影响函数的值，而在调用该函数之前，并不影响对数组表达式变元的计值。MASK 变元必须是逻辑型的。

(2) 向量和矩阵乘法函数。矩阵乘法函数 MATMUL 作用在两个矩阵，或一个矩阵和一个向量上，返回相应的矩阵一矩阵、矩阵一向量或向量一矩阵乘积。MATMUL 的变元可以是数值的(整的、实的或复的)或逻辑的数组。

点乘函数 DOT _ PRODUCT 作用在两个向量上，返回它们的数量积。其中的向量变元如同 MATMUL 的一样，必须是同类型的，或者数值的或者逻辑的。对逻辑向量而言，DOT _ PRODUCT 返回布尔数量积。

(3) 数组归约函数。数组归约函数 SUM、PRODUCT、MAXVAL、MINVAL、COUNT、ANY 和 ALL 在数组上执行逻辑的、数值的和计数等运算。它们可作用在整个数组上而给出数量结果，

或者它们也可作用在一个给定的维上而产生一个秩减少了 1 的结果。采用一个与给定数组一致的逻辑 MASK，可以把计算限制在数组的任意一个元素子集(例如所有正元素)上。

(4) 数组查询函数。数组查询函数包括 ALLOCATED、SIZE、SHAPE、LBOUND 以及 UBOUND。函数 ALLOCATED 用来查询一个数组变元当前是否已被分配单元。如果一个数组变元当前已被分配单元，则该函数返回真值，否则返回假值。函数 SIZE、SHAPE、LBOUND 和 UBOUND 分别返回数组变元的大小、形状、数组沿各维下标的上界和下界。这时要求变元的大小、形状和上下界必须都是有定义的，但这些函数的数组变元的值并不一定需要有定义。

(5) 数组构造函数。数组构造函数包括 MERGE、SPREAD、PACK 和 UNPACK，它们被用来从已有数组的元素构造出新的数组。函数 MERGE 能将两个形状一致的数组根据一个逻辑 MASK 选择两者的元素而组合成另一个数组。函数 SPREAD 能从一个实元的几份拷贝构造出一个数组，其实它是如同由多个页面构成一本书那样以增加一个额外的维的办法来作到这个的。函数 PACK 和 UNPACK 分别用来将一个数组某些位置上的元素收集到一个一维数组和将一个一维数组的元素散开到另一数组的某些位置上，而这些位置是用一个逻辑 MASK 来指明的。

(6) 数组重构形函数。数组重构形函数 RESHAPE 用来生成一个与变元具有相同的元素，而具有不同形状的新数组。

(7) 数组操作函数。数组操作函数包括 TRANSPOSE、EOSHIFT 和 CSHIFT 三种。TRANSPOSE 作矩阵，即二维数组的转置操作。移位函数 EOSHIFT 和 CSHIFT 保留数组变元的形状不变而将其元素位置平行地移动到某数组的一个指定的维上。CSHIFT 进行循环移位，即从一端移出的元素又能重新出现在另一端。EOSHIFT 进行去端移位(end－off)，这时指定边界上的元素将被移成空位。

(8) 数组定位函数。数组定位函数有 MAXLOC 和 MINLOC

两个，它们分别返回变元数组的元素中具有最大值和最小值的下标位置。采用一个与给定变元数组具有相同形状的逻辑 MASK，可将这种定位工作限制在该数组的任意子集上进行。

4. 指针结合状态查询函数

函数 ASSOCIATED 测试一个指针当前是否已与任意一个目标结合，或已与一个特定的目标结合，或已与另一指针结合了一个相同的目标。

5. 变元存在性查询函数

内在函数 PRESENT 用来查询具有 OPTIONAL 属性的虚元是否已经结合了一个实元。

第二节　类属内在函数

上节我们已对内在过程，包括内在函数和内在子程序作了一般介绍，也谈到了一些内在函数可有特定名和类属名，用类属名引用的函数其变元可有多种类型。本节我们将列出 Fortran 90 中的全部类属内在函数。在以下的所有内在过程中，变元处列的是相应的关键词，即在实元用关键词形式引用时所用的关键词。例如，对 CMPLX 的一个引用既可用形式 CMPLX(A,B,M)，也可用形式 CMPLX(Y=B,KIND=M,X=A)，于是我们在下面把它列为 CMPLX(X,Y,KIND)。

许多变元关键词的名字具有指示其用途的含义，例如：

KIND	描述结果的 KIND
STRING, STRING_A	表示一个任意的字符串
BACK	指示对一个字符串的扫描是从右到左(BACKWARD)的
MASK	表示对该变元可作用以一个屏蔽码(MASK)
DIM	表示一个变元数组的一个选定的维

等等。

一、变元存在性查询函数

PRESENT(X)	查询变元的存在性

二、数值函数

ABS(A)	求绝对值
AIMAG(Z)	求复数的虚部
AINT(A,KIND)	截断为整数
KIND 任选	
ANINT(A,KIND)	求最近的整数
KIND 任选	
CEILING(A)	求大于等于变元值的最小整数
CMPLX(X,Y,KIND)	转换为复型
Y,KIND 任选	
CONJG(Z)	复数的共轭
DBLE(A)	转换为双精度实型
DIM(X,Y)	求正差(positive difference)
DPROD(X,Y)	求双精度实型乘积
FLOOR(A)	求小于等于变元值的最大整数
INT(A,KIND)	转换为整型
KIND 任选	
MAX(A1,A2,A3,...)	求极大
A3,... 任选	
MIN(A1,A2,A3,...)	求极小
A3,... 任选	
MOD(A,P)	求余数
MODULO(A,P)	取模
NINT(A,KIND)	求最近的整数
KIND 任选	
REAL(A,KIND)	转换为实型
KIND 任选	
SIGN(A,B)	转移符号

三、数学函数

ACOS(X)	反余弦
ASIN(X)	反正弦
ATAN(X)	反正切
ATAN2(Y,X)	反正切
cos(X)	余弦
COSH(X)	双曲余弦
EXP(X)	指数
LOG(X)	自然对数
LOG10(X)	公用对数(基底为 10)
SIN(X)	正弦
SINH(X)	双曲正弦
SQRT(X)	开方
TAN(X)	正切
TANH(X)	双曲正切

四、字符函数

ACHAR(I)	取 ASCII 理序序列中给定位置处的字符
ADJUSTL(STRING)	左对齐
ADJUSTR(STRING)	右对齐
CHAR(I,KIND) KIND 任选	取处理系统理序序列中给定位置处的字符
IACHAR(C)	求一个字符在 ASCII 理序序列中的位置
ICHAR(C)	求一个字符在处理系统理序序列中的位置
INDEX(STRING,SUBSTRING,BACK) BACK 任选	求一个子串的开始位置
LEN_TRIM(STRING)	求一个字符串在不计尾部空格字符时的长度
LGE(STRING_A,STRING_B)	字符次序上的大于等于

LGT(STRING _ A, STRING _ B)	字符次序上的大于
LLE(STRING _ A, STRING _ B)	字符次序上的小于等于
LLT(STRING _ A, STRING _ B)	字符次序上的小于
REPEAT(STRING, NCOPIES)	重复拼接一个字符串若干次
SCAN(STRING, SET,BACK)	在一个字符串中扫描集合中的一个字符
TRIM(STRING)	取掉字符串的尾部空格字符
VERIFY(STRING, SET,BACK) BACK 任选	在一个字符串中校验字符集合

五、字符查询函数

LEN(STRING)	求一个字符实体的长度

六、KIND 函数

KIND(X)	求变元的种别类型参数值
SELECTED _ INT _ KIND(R)	求整型种别类型参数值，给出范围
SELECTED _ REAL _ KIND(P,R) P，R 任选	求实型种别类型参数值，给出精度和范围

七、逻 辑 函 数

LOGICAL(L,KIND) KIND 任选	在具有不同种别类型参数的逻辑类型对象之间转换

八、数值查询函数

DIGITS(X)	求该表示模型中有效数字的数目

EPSILON(X)	回送一个与1比较起来几乎可忽略不计的数
HUGE(X)	求表示模型中最大的数
MAXEXPONENT(X)	求表示模型中的最大指数
MINEXPONENT(X)	求表示模型中的最小指数
PRECISION(X)	查询十进制精度
RADIX(X)	查询模型的基数
RANGE(X)	查询十进制指数的范围
TINY(X)	求表示模型中的最小正数

九、位查询函数

BIT_SIZE(I)	求表示模型中位的数目

十、位操作函数

BTEST(I,POS)	位测试
IAND(I,J)	逻辑与
IBCLR(I,POS)	清除一位
IBITS(I,POS,LEN)	抽出若干位
IBSET(I,POS)	置位
IEOR(I,J)	排除或
IOR(I,J)	或
ISHFT(I,SHIFT)	逻辑移位
ISHFTC(I,SHIFT,SIZE) SIZE 任选	循环移位
NOT(I)	逻辑补

十一、类型转移函数

TRANSFER(SOURCE, MOLD,SIZE) SIZE 任选	把第一变元类型处理作具有第二变元的类型

十二、浮点操作函数

EXPONENT(X)	求一个模型数的指数部分

FRACTION(X)　　求一个模型数的分数部分

NEAREST(X,S)　　在给定方向上找到最近的不同的处理系统数

RRSPACING(X)　　求靠近给定数的模型数的相对空位的倒数

SCALE(X,I)　　乘一个实数以基数的整数次幂

SET _ EXPONENT(X,I)置一个数的指数部分

SPACING(X)　　求靠近给定数的模型数的绝对空位

十三、向量和矩阵相乘函数

DOT _ PRODUCT(VECTOR _ A,VECTOR _ B)　　两个一维数组的点积(或称内积)

MATMUL(MATRIX _ A, MATRIX _ B)　　矩阵相乘

十四、数组归约函数

ALL(MASK,DIM)　　如果所有值为真则真
DIM 任选

ANY(MASK,DIM)　　如果有任一值为真则真
DIM 任选

COUNT(MASK,DIM)　　求数组中真元素的个数
DIM 任选

MAXVAL(ARRAY,DIM,MASK)　求数组中元素的最大值
DIM, MASK 任选

MINVAL(ARRAY,DIM,MASK)　求数组中元素的最小值
DIM, MASK 任选

PRODUCT(ARRAY,DIM,MASK)　求数组元素的积
DIM, MASK 任选

SUM(ARRAY,DIM,MASK)　求数组元素的和
DIM, MASK 任选

十五、数组查询函数

ALLOCATED(ARRAY)　　查询数组分配状态

LBOUND(ARRAY,DIM)　　查询数组各维的下界

DIM 任选

SHAPE(SOURCE) 查询一个数组或标量的形状

SIZE(ARRAY,DIM) 查询数组元素的总数

DIM 任选

UBOUND(ARRAY,DIM) 查询数组各维的上界

DIM 任选

十六、数组构造函数

MERGE(TSOURCE, FSOURCE,MASK) 在 MASK 控制下合并两个数组

PACK(ARRAY,MASK, VECTOR) 在 MASK 控制下打包一个数组放进一个秩为 1 的数组

VECTOR 任选

SPREAD(SOURCE, DIM,NCOPIES) 用增加一维的办法重复数组

UNPACK(VECTOR, MASK,FIELD) 在 MASK 控制下把一个秩为 1 的数组解包放进一个数组

十七、数组重构函数

RESHAPE(SOURCE,SHAPE, PAD,ORDER) 重构一个数组

PAD, ORDER 任选

十八、数组操作函数

CSHIFT(ARRAY,SHIFT,DIM) 循环移位

DIM 任选

EOSHIFT(ARRAY,SHIFT, BOUNDARY,DIM) 去端移位

BOUNDARY, DIM 任选

TRANSPOSE(MATRIX) 矩阵转置

十九、数组定位函数

MAXLOC(ARRAY,MASK) 求数组中值最大的元素的位置

MASK 任选

MINLOC(ARRAY,MASK)　求数组中值最小的元素的位置

MASK 任选

二十、指针结合状态查函数

ASSOCIATED(POINTER,TARGET)　查询结合状态

TARGET 任选

第三节　内在子程序

内在子程序是由处理系统提供的具有特定定义的子程序过程。程序员无需定义即可用CALL 语句按名调用它们。但应注意，内在子程序的名一定不能作为过程的实元。

一、内在子程序的功能分类

现在先按功能分类简单介绍一下各种内在子程序。

1. 日期和时间子程序

子程序 DATE_AND_TIME 和 SYSTEM_CLOCK 用来从实时时钟中得到当时的日期和时间。所得的时间是当地时间，但用一个实用程序来找到当地时间与坐标通用时间(Coordinated Universal Time)之间的时差。

2. 伪随机数子程序

子程序 RANDOM_NUMBER 返回一个伪随机数或一伪随机数的数组。子程序 RANDOM_SEED 用来初始化或重新开始一个随机数序列。

3. 位拷贝子程序

子程序 MVBITS 用来将一个整型对象的指定位置处的一个位字段拷贝到另一对象的指定位置去。它是一个可作用于数组变元的元素上的元素子程序。

二、内在子程序的调用形式

DATE _ AND _ TIME(DATE, TIME,ZONE,VALUES) DATE, TIME, ZONE, VALUES 任选	得到日期和时间
MVBITS(FROM,FROMPOS, LEN,TO,TOPOS)	从一个整数拷贝一些位到另一整数
RANDOM _ NUMBER (HARVEST)	回送随机数
RANDOM _ SEED (SIZE,PUT,GET) SIZE, PUT, GET 任选	初始化或重新开始随机数产生器
SYSTEM _ CLOCK(COUNT, COUNT _ RATE,COUNT _ MAX) COUNT,COUNT _ RATE,COUNT _ MAX 任选	从系统时钟得到数据

第四节 内在函数的特定名和类属名对照表

本节我们来列出所有内在函数的特定名，指出它们相应的类属名，并标明当用特定名引用函数时变元的类型。

特定名	类属名	变元类型
ABS(A)	ABS(A)	默认实型
ACOS(X)	ACOS(X)	默认实型
AIMAG(Z)	AIMAG(Z)	默认复型
AINT(A)	AINT(A)	默认实型
ALOG(X)	LOG(X)	默认实型
ALOG10(X)	LOG10(X)	默认实型
· AMAX0(A1,A2,A3,…) A3,…任选	REAL(MAX(A1,A2,A3,…)) A3,…任选	默认整型
· AMAX1(A1,A2,A3,…) A3,…任选	MAX(A1,A2,A3,…) A3,…任选	默认实型
· AMIN0(A1,A2,A3,…) A3,…任选	REAL(MIN(A1,A2,A3,…)) A3,…任选	默认整型
· AMIN1(A1,A2,A3,…)	MIN(A1,A2,A3,…)	默认实型

A3,…任选	A3,…任选	
AMOD(A,P)	MOD(A,P)	默认实型
ANINT(A)	ANINT(A)	默认实型
ASIN(X)	ASIN(X)	默认实型
ATAN(X)	ATAN(X)	默认实型
ATAN2(Y,X)	ATAN2(Y,X)	默认实型
CABS(A)	ABS(A)	默认复型
CCOS(X)	COS(X)	默认复型
CEXP(X)	EXP(X)	默认复型
• CHAR(I)	CHAR(I)	默认整型
CLOG(X)	LOG(X)	默认复型
CONJG(Z)	CONJG(X)	默认复型
COS(X)	COS(X)	默认实型
COSH(X)	COSH(X)	默认实型
CSIN(X)	SIN(X)	默认复型
CSQRT(X)	SQRT(X)	默认复型
DABS(X)	ABS(X)	双精度实型
DACOS(X)	ACOS(X)	双精度实型
DASIN(X)	ASIN(X)	双精度实型
DATAN(X)	ATAN(X)	双精度实型
DATAN2(Y,X)	ATAN2(Y,X)	双精度实型
DCOS(X)	COS(X)	双精度实型
DCOSH(X)	COSH(X)	双精度实型
DDIM(Y,X)	DIM(Y,X)	双精度实型
DEXP(X)	EXP(X)	双精度实型
DIM(X,Y)	DIM(X,Y)	默认实型
DINT(A)	AINT(A)	双精度实型
DLOG(X)	LOG(X)	双精度实型
DLOG10(X)	LOG10(X)	双精度实型
• DMAX1(A1,A2,A3,…)	MAX(A1,A2,A3,…)	双精度实型
A3,…任选	A3,…任选	
• DMIN1(A1,A2,A3,…)	MIN(A1,A2,A3,…)	双精度实型
A3,…任选	A3,…任选	
DMOD(A,P)	MOD(A,P)	双精度实型
DNINT(A)	ANINT(A)	双精度实型
DPROD(X,Y)	DPROD(X,Y)	默认实型
DSIGN(A,B)	SIGN(A,B)	双精度实型
DSIN(X)	SIN(X)	双精度实型
DSINH(X)	SINH(X)	双精度实型

DSQRT(X)	SQRT(X)	双精度实型
DTAN(X)	TAN(X)	双精度实型
DTANH(X)	TANH(X)	双精度实型
EXP(X)	EXP(X)	默认实型
• FLOAT(A)	REAL(A)	默认整型
IABS(A)	ABS(A)	默认整型
• ICHAR(C)	ICHAR(C)	默认字符型
IDIM(X,Y)	DIM(X,Y)	默认整型
• IDINT(A)	INT(A)	双精度实型
IDNINT(A)	NINT(A)	双精度实型
• IFIX(A)	INT(A)	默认实型
INDEX(STRING,SUBSTRING)	INDEX(STRING,SUBSTRING)	默认字符型
• INT(A)	INT(A)	默认实型
ISIGN(A,B)	SIGN(A,B)	默认整型
LEN(STRING)	LEN(STRING)	默认字符型
• LGE(STRING_A,STRING_B)	LGE(STRING_A,STRING_B)	默认字符型
• LGT(STRING_A,STRING_B)	LGT(STRING_A,STRING_B)	默认字符型
• LLE(STRING_A,STRING_B)	LLE(STRING_A,STRING_B)	默认字符型
• LLT(STRING_A,STRING_B)	LLT(STRING_A,STRING_B)	默认字符型
• MAX0(A1,A2,A3,…)	MAX(A1,A2,A3,…)	默认整型
A3,… 任选	A3,…任选	
• MAX1(A1,A2,A3,…)	INT(MAX(A1,A2,A3,…))	默认实型
A3,… 任选	A3,…任选	
• MIN0(A1,A2,A3,…)	MIN(A1,A2,A3,…)	默认整型
A3,…任选	A3,…任选	
• MIN1(A1,A2,A3,…)	INT(MIN(A1,A2,A3,…))	默认实型
A3,… 任选	A3,…任选	
MOD(A,P)	MOD(A,P)	默认整型
NINT(A)	NINT(A)	默认实型
• REAL(A)	REAL(A)	默认整型
SIGN(A,B)	SIGN(A,B)	默认实型
SIN(X)	SIN(X)	默认实型
SINH(X)	SINH(X)	默认实型
• SNGL(A)	REAL(A)	双精度实型
SQRT(X)	SQRT(X)	默认实型
TAN(X)	TAN(X)	默认实型
TANH(X)	TANH(X)	默认实型

值得注意，这些标明圆点的特定内在函数名一定不能作过程

的实元。只有那些前头不标明圆点的内在函数的特定名才能作为过程的实元。

第五节　内在过程的详细说明

本节我们将给出所有内在过程的详细说明。每个说明将包括描述、分类、变元、结果类型和类型参数(可能还有形状)、结果值和例子等项，并按过程名的字母顺序排列。

1. ABS(A)

描述:求绝对值。

分类:元素函数。

变元:A 必须是整型、实型或复型。

结果类型和类型参数:与 A 的相同，除了当 A 是复型时结果为实型。

结果值:当 A 是整型或实型时，结果为|A|，当 A 为复型值(x,y)时，结果为(x * * 2+y * * 2) * * (1/2) 的依赖处理系统的近似值。

例子:ABS((3.0,4.0)) 近似地有值 5.0 。

2. ACHAR(I)

描述:回送在 ASCII 理序序列中指定位置的字符。它是 IACHAR 的反函数。

分类:元素函数。

变元:I 必须是整型。

结果类型和类型参数:长度为 1 的字符，具有类型参数值 KIND('A') 。

结果值:如果 I 有值 0≤I≤127,倘若处理系统能表示该字符，则结果是位于 ASCII 理序序列中第 I 个位置的字符；否则，结果是依赖处理系统的。如果处理系统不能同时表示大写和小写字母，当 I 对应位置的字母不能被表示时，结果就是处理系统能表示的相应字母，即大小写不分。对于处理系统能表示的任意字母 C,

ACHAR(IACHAR(C))必须得值C。

例子:ACHAR(88) 有值'X' 。

3. ACOS(X)

描述:反余弦函数。

分类:元素函数。

变元:X 必须是实型的，且值满足不等式$|X|\leqslant 1$ 。

结果类型和类型参数:与 X 的相同。

结果值:结果为以弧度表示的 arccos(x)的依赖处理系统的近似值,满足 $0\leqslant ACOS(X)\leqslant\pi$ 。

例子:ACOS(0.54030231) 近似地有值 1.0 。

4. ADJUSTL(STRING)

描述:去掉前头的空格，补进尾部的空格，使得左面对齐。

分类:元素函数。

变元:STRING 必须是字符型的。

结果类型:与 STRING 具有相同长度和种别类型参数。

结果值:结果是将 STRING 所有领头的空格删掉并在尾部补进相同个数空格后得到的字符串。

例子:ADJUSTL('WORD') 有值 'WORD' 。

5. ADJUSTR(STRING)

描述:去掉尾部的空格，补进领头的空格，使得右面对齐。

分类:元素函数。

变元:STRING 必须是字符型的。

结果类型:与 STRING 具有相同长度和种别类型参数。

结果值:结果是将 STRING 所有尾部的空格删掉并在前头补进相同个数空格后得到的字符串。

例子:ADJUSTR('WORD') 有值 'WORD' 。

6. AIMAG(Z)

描述:求复数的虚部。

分类:元素函数。

变元:Z 必须是复型的。

结果类型和类型参数：与 Z 具有相同种别类型参数的实型数。

结果值：如果 Z 为(x,y)，则结果值是 y 。

例子：AIMAG(2.0，3.0)的值为 3.0 。

7. AINT(A,KIND) KIND 任选

描述：截断为整数。

分类：元素函数。

变元：A 必须是实型的。

KIND(任选)必须是一个整型标量初始化表达式。

结果类型和类型参数：结果为实型的。如果有 KIND，则种别类型参数由 KIND 指明；否则种别类型参数与 A 的相同。

结果值：若 $|A|\leqslant 1$，AINT(A)有值 0；若 $|A|\geqslant 1$，AINT(A)的值为大小是不超过 A 的绝对值的最大整数，符号与 A 相同的整数。

例子：AINT(2.863)的值为 2.0；AINT(－2.863)的值为－2.0 。

8. ALL(MASK,DIM) DIM 任选

描述：决定在 MASK 中沿着 DIM 指示的维内是否所有值都是真。

分类：变换函数。

变元：MASK 必须是逻辑型，且一定不是标量。

DIM(任选)必须是值在 $1\leqslant DIM\leqslant n$ 之间的整型标量，其中 n 是 MASK 的秩，且与 DIM 对应的实元一定不能是一个任选的虚元。

结果类型、类型参数和形状：结果是逻辑型的，具有与 MASK 相同种别类型参数。若 DIM 缺省或 MASK 的秩为 1，则结果是标量；否则结果是一个秩为 n－1 的，形状为 $(d_1,d_2,\cdots,d_{DIM-1},d_{DIM+1},\cdots,d_n)$ 的数组，其中 $(d_1,d_2,\cdots,d_n)$ 是 MASK 的形状。

结果值：情况 1　若 MASK 的所有元素为真或 MASK 的大小是 0，则 ALL(MASK)的值为真；若 MASK 有一个元素为假，

则结果值为假。

情况 2　若 MASK 的秩为 1，ALL(MASK,DIM)的值等于 ALL (MASK)；否则(即当 MASK 的秩不为 1 时)，ALL (MASK,DIM)中下标为($s_1, s_2, \cdots, s_{DIM-1}, s_{DIM+1}, \cdots, s_n$)的元素的值等于 ALL(MASK($s_1, s_2, \cdots, s_{DIM-1}$, : , $s_{DIM+1}, \cdots, s_n$))的值。

例子：情况 1　ALL((/. TRUE. ,. FALSE. ,. TRUE. /))的值为假。

情况 2　B 和 C 分别是数组：

$$\begin{bmatrix}1 & 3 & 5\\ 2 & 4 & 6\end{bmatrix} \text{ 和 } \begin{bmatrix}0 & 3 & 5\\ 7 & 4 & 8\end{bmatrix}$$

则 ALL(B . NE. C，DIM＝1)是〔真，假，假〕；ALL(B . NE. C，DIM＝2)是〔假，假〕。

9. ALLOCATED(ARRAY)

描述：指出一个可分配数组当前是否已分配。

分类：查询函数。

变元：ARRAY 必须是一个可分配数组。

结果类型、类型参数和形状：默认逻辑型标量。

结果值：若当前 ARRAY 已被分配，则结果为真；若尚未分配，则结果值为假；若数组的分配状态是不定义的，则结果也不定义。

10. ANINT(A,KIND) KIND 任选

描述：截断为最靠近变元 A 的整数。

分类：元素函数。

变元：A 必须是实型的。

KIND(任选)必须是一个整型标量初始化表达式。

结果类型和类型参数：结果为实型。如果有 KIND，则其种别类型参数由 KIND 指明；否则种别类型参数与 A 的相同。

结果值：若 A＞0，则 ANINT(A)有值 AINT(A＋0. 5)；若 A ≤0，则 ANINT(A) 有值 AINT(A－0. 5)。

例子：ANINT(2. 863)的值为 3. 0；ANINT(－2. 863)的值为

−3.0 。

11. ANY(MASK,DIM) DIM 任选

描述:决定在 MASK 中沿着 DIM 指示的维内是否有一个值为真。

分类:变换函数。

变元:MASK 必须是逻辑型的，且一定不是标量。

DIM(任选)必须是值在 $1 \leqslant DIM \leqslant n$ 之间的整型标量，其中 n 是 MASK 的秩，且与 DIM 对应的实元一定不能是一个任选的虚元。

结果类型、类型参数和形状:结果是逻辑型的，具有与 MASK 相同的种别类型参数。若 DIM 缺省或 MASK 的秩为 1，则结果是标量；否则结果是一个秩为 $n-1$ 的，形状为$(d_1,d_2,\cdots,d_{DIM-1},d_{DIM+1},\cdots,d_n)$的数组，其中$(d_1,d_2,\cdots,d_n)$是 MASK 的形状。

结果值:情况 1　若 MASK 有某元素为真，则 ANY(MASK) 的值为真；若 MASK 没有一个元素为真或 MASK 的大小是 0，则结果值为假。

情况 2　若 MASK 的秩为 1，ANY(MASK,DIM)的值等于 ANY(MASK)；否则(即当 MASK 的秩不为 1 时)，ANY(MASK,DIM)中下标为$(s_1,s_2,\cdots,s_{DIM-1},s_{DIM+1},\cdots,s_n)$的元素的值等于 ANY(MASK$(s_1,s_2,\cdots,s_{DIM-1}, :, s_{DIM+1},\cdots,s_n)$))的值。

例子:情况 1　ANY((/.TRUE.,.FALSE.,.TRUE./))的值为真。

情况 2　B 和 C 分别是数组:

$$\begin{bmatrix}1 & 3 & 5\\2 & 4 & 6\end{bmatrix} \text{ 和 } \begin{bmatrix}0 & 3 & 5\\7 & 4 & 8\end{bmatrix}$$

则 ANY(B .NE. C, DIM=1)是〔真,假,真〕；ANY(B .NE. C, DIM=2)是〔真,真〕。

12. ASIN(X)

描述:反正弦函数。

分类:元素函数。

变元:X 必须是实型的，且值满足不等式 $|X|\leqslant 1$ 。

结果类型和类型参数:与 X 的相同。

结果值:结果为以弧度表示的 arcsin(x)的依赖处理系统的近似值，满足 $-\pi/2\leqslant$ ASIN(X) $\leqslant\pi/2$ 。

例子:ASIN(0.84147098) 近似地有值 1.0 。

13. ASSOCIATED(POINTER,TARGET) TARGET 任选

描述:回送指针变元的结合状态或指出与目标结合的指针的状态。

分类:查询函数。

变元:POINTER 必须是一个指针，但可为任意类型。它的指针结合状态一定不能是不定义的。

TARGET 必须是一个指针或一个目标。如果它是一个指针，它的指针结合状态一定不能是不定义的。

结果类型:结果是默认逻辑型的。

结果值:情况 1　若 TARGET 省缺，则当 POINTER 当前已与一个目标结合时结果为真，当 POINTER 当前尚未与一个目标结合时结果为假。

情况 2　若存在 TARGET 并且是一个目标，则当 POINTER 当前已与 TARGET 结合时结果为真，当 POINTER 当前尚未与 TARGET 结合时结果为假。

情况 3　若存在 TARGET 并且是一个指针，则当 POINTER 和 TARGET 当前已与同一目标结合时结果为真，否则为假。当 POINTER 或 TARGET 有一个为未结合状态时，结果也为假。

例子:如果 CURRENT 指向目标 HEAD，则 ASSOCIATED (CURRENT,HEAD)为真；在执行了语句:

```
A_PART=>A(:N)
```

之后，如果 N 等于 UBOUND(A,DIM=1)，则 ASSOCIATED (A_PART,A)为真；在执行了语句:

```
NULLIFY(CUR); NULLIFY(TOP)
```

之后，ASSOCIATED(CUR,TOP)为假。

14. ATAN(X)

描述:反正切函数。

分类:元素函数。

变元:X 必须是实型的。

结果类型和类型参数:与 X 的相同。

结果值:结果为以弧度表示的 arctan(x)的依赖处理系统的近似值,满足 $-\pi/2 \leqslant \text{ATAN}(X) \leqslant \pi/2$ 。

例子:ATAN(1.5574077) 近似地有值 1.0 。

15. ATAN2(Y,X)

描述:反正切函数。结果是非零复数变元(X,Y)的主值(principle value)。

分类:元素函数。

变元:Y 必须是实型的。

X 必须与 Y 具有相同类型和种别类型参数。如果 Y 的值为 0,则 X 的值不能也是 0 。

结果类型和类型参数:与 X 的相同。

结果值:结果为以弧度表示的复数变元(X,Y)的主值的依赖处理系统的近似值,满足 $-\pi \leqslant \text{ATAN2}(Y,X) \leqslant \pi$,并且等于 arctan(Y/X)的依赖处理系统的近似值(如果 $X \neq 0$)。若 $Y>0$,则结果为正。若 $Y=0$,则当 $X>0$ 时结果为 0,当 $X<0$ 时结果为 π。若 $Y<0$,结果为负。若 $X=0$,则结果的绝对值为 $\pi/2$。

例子:ATAN2(1.5574077,1.0) 近似地有值 1.0 。如果 Y 和 X 分别有值:

$$\begin{bmatrix} 1 & 1 \\ -1 & -1 \end{bmatrix} \text{ 和 } \begin{bmatrix} -1 & 1 \\ -1 & 1 \end{bmatrix}$$

则 ATAN2(Y,X)的值近似地为:

$$\begin{bmatrix} 3\pi/4 & \pi/4 \\ -3\pi/4 & -\pi/4 \end{bmatrix}$$

16. BIT_SIZE(I)

描述:回送在本章(11. I)定义的模型中的位数 s。

分类:查询函数。

变元:I 必须是整型数。

结果类型、类型参数和形状:与 I 具有相同种别类型参数的整型标量。

结果值:结果是在本章(11. I)中为位操作定义的整数模型中的位数 s 的值。

例子:若在模型中 s 的值为 32,则 BIT_SIZE(1)有值 32。

17. BTEST(I,POS)

描述:测试一个整数值的位的状态。

分类:元素函数。

变元:I 必须整型。

POS 也必须是整型的,并且是小于 BIT_SIZE(I)的非负整数。

结果类型:结果为默认逻辑型。

结果值:若 I 的第 POS 位为 1,则结果为真;若为 0,则结果为假。这里 I 的值按本章(11. I)中的模型解释为一个位序列。

例子:BTEST(8,3)的值为真。若 A 的值为:

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}.$$

则 BTEST(A,2) 的值为:

$$\begin{bmatrix} 假 & 假 \\ 假 & 真 \end{bmatrix}$$

18. CEILING(A)

描述:回送大于等于其变元的最小整数。

分类:元素函数。

变元:A 必须是实型。

结果类型和类型参数:默认整型。

结果值:结果是大于等于变元 A 的最小整数。如果处理系统不能把这个数表示为默认整型,则结果不定义。

例子:CEILING(3.7)的值为 4;CEILING(-3.7)的值为

—3。

19. CHAR(I,KIND) KIND 任选

描述:根据与指定种别类型参数相联的处理系统理序序列回送一个给定位置上的字符。它是 ICHAR 的反函数。

分类:元素函数。

变元:I 必须是整型的，其值在范围 0≤I≤n—1 之中，这里 n 是与指定种别类型参数相联的理序序列中字符的个数。

KIND(任选) 必须是一个整型的标量初始化表达式。

结果类型和类型参数:长度为 1 的字符。如果有 KIND，则种别类型参数由 KIND 指明；否则默认为字符型。

结果值:结果是位于与指定种别类型参数相联的理序序列中第 I 个位置处的字符。对满足 0≤I≤n—1 的任意的 I，ICHAR (CHAR(I,KIND(C))) 必须有值 I；而且对任意处理系统中可表示的字符 C，CHAR(ICHAR(C)，KIND(C)) 必须有值 C。

例子:在采用 ASCII 理序序列的处理系统中，CHAR(88)的值为'X'。

20. CMPLX(X,Y,KIND) Y，KIND 任选

描述:转换为复型。

分类:元素函数。

变元:X 必须是整型、实型或复型。

Y(任选) 必须是整型或实型。如果 X 为复型，Y 必须不存在。

KIND(任选) 必须是一个整型的标量初始化表达式。

结果类型和类型参数:结果为复型。如果有 KIND，则种别类型参数由 KIND 指明；否则为默认实型。

结果值:如果 Y 存在并且 X 不是复型，结果值如同 Y 有零值那样求得。如果 Y 不存在并且 X 为复型，结果值如同 Y 有值 AIMAG(X)那样求得。CMPLX(X,Y,KIND) 具有复数值，其实部为 REAL(X,KIND) 而其虚部为 REAL(Y,KIND)。

例子:CMPLX(—3) 有值 (—3.0，0.0)。

21. CONJG(Z)

描述:求复数的共轭。

分类:元素函数。

变元:Z 必须是复型。

结果类型和类型参数:与 Z 相同。

结果值:如果 Z 有值(x,y),则结果有值(x,—y)。

例子:CONJG((2.0,3.0)) 有值 (2.0,—3.0)。

22. COS(X)

描述:余弦函数。

分类:元素函数。

变元:X 必须是实型或复型。

结果类型和类型参数:与 X 的相同。

结果值:结果为依赖处理系统的 COS(X)的近似值。如果 X 为实型,则变元被理解为 X 所表示的弧度;如果 X 为复型,则变元被理解为 X 的实部所表示的弧度。

例子:COS(1.0) 近似地有值 0.54030231。

23. COSH(X)

描述:双曲余弦函数。

分类:元素函数。

变元:X 必须是实型的。

结果类型和类型参数:与 X 的相同。

结果值:结果为依赖处理系统的 COSH(X)的近似值。

例子:COSH(1.0) 近似地有值 1.5430806。

24. COUNT(MASK,DIM) DIM 任选

描述:沿着 DIM 指示的维计算在 MASK 中值为真的元素的个数。

分类:变换函数。

变元:MASK 必须是逻辑型,且一定不是标量。

DIM(任选)必须是值在 1≤DIM≤n 之间的整型标量,其中 n 是 MASK 的秩,且与 DIM 对应的实元一定不能是一个任选的虚元。

结果类型、类型参数和形状：结果是默认整型的。如果 DIM 缺省或 MASK 的秩为 1，则结果是标量；否则结果是一个秩为 $n-1$ 的，形状为 $(d_1, d_2, \cdots, d_{DIM-1}, d_{DIM+1}, \cdots, d_n)$ 的数组，其中 $(d_1, d_2, \cdots, d_n)$ 是 MASK 的形状。

结果值：情况 1　若 MASK 的大小是 0，则 COUNT (MASK) 为零；否则结果为 MASK 中值为真的元素的个数。

情况 2　若 MASK 的秩为 1，COUNT(MASK,DIM) 的值等于 COUNT(MASK)；否则 COUNT(MASK,DIM) 中下标为 $(s_1, s_2, \cdots, s_{DIM-1}, s_{DIM+1}, \cdots, s_n)$ 的元素的值等于 COUNT(MASK $(s_1, s_2, \cdots, s_{DIM-1}, :, s_{DIM+1}, \cdots, s_n)$) 的值。

例子：情况 1　COUNT((/.TRUE.,.FALSE.,.TRUE./)) 的值为 2。

情况 2　若 B 和 C 分别是数组：

$$\begin{bmatrix} 1 & 3 & 5 \\ 2 & 4 & 6 \end{bmatrix} \text{ 和 } \begin{bmatrix} 0 & 3 & 5 \\ 7 & 4 & 8 \end{bmatrix}$$

则 COUNT(B .NE. C, DIM=1) 是〔2,0,1〕；COUNT(B .NE. C, DIM=2) 是〔1,2〕。

25. CSHIFT(ARRAY,SHIFT,DIM) DIM 任选

描述：在一个秩为 1 的数组表达式上作循环移位，或沿着秩为 2 或更大秩的数组表达式的给定维在所有秩为 1 的完整数组片段上作循环移位。从一个数组片段的一端移出的元素被移入另一端。不同的片段可按不同的方向移不同的位数。

分类：变换函数。

变元：ARRAY 可是任何类型，但必须不是标量。

SHIFT 当 ARRAY 秩为 1 时必须为整型标量；否则，它必须是标量，或者是秩为 $n-1$，形状为 $(d_1, d_2, \cdots, d_{DIM-1}, d_{DIM+1}, \cdots, d_n)$ 的数组，其中 $(d_1, d_2, \cdots, d_n)$ 是 ARRAY 的形状。

DIM（任选）必须是值在 $1 \leqslant DIM \leqslant n$ 之间的整型标量，其中 n 是 ARRAY 的秩。如果 DIM 被省略，它就认为具有值 1。

结果类型、类型参数和形状：结果的类型、类型参数和形状与

ARRAY 的相同。

结果值:情况 1　若 ARRAY 有秩 1，则结果的第 i 个元素为 ARRAY(1+MODULO(i+SHIFT−1, SIZE(ARRAY)))。

情况 2　若 ARRAY 的秩大于 1，则结果的片段($s_1,s_2,\cdots,s_{DIM-1}$, : , $s_{DIM+1},\cdots,s_n$)的值等于 CSHIFT(ARRAY($s_1,s_2,\cdots,s_{DIM-1}$, : , $s_{DIM+1},\cdots,s_n$), 1, sh)，其中 sh 为 SHIFT 或 SHIFT($s_1,s_2,\cdots,s_{DIM-1},s_{DIM+1},\cdots,s_n$)。

例子:情况 1　如果 V 是数组[1,2,3,4,5,6]，则用 CSHIFT (V, SHIFT=2)循环左移两位的结果为[3,4,5,6,1,2]；而用 CSHIFT(V, SHIFT=−2)循环右移两位的结果为[5,6,1,2,3,4]。

情况 2　一个秩为 2 的数组的各行可被移相同的位数，也可被移不同的位数。设 M 是数组

$$\begin{bmatrix}1 & 2 & 3\\4 & 5 & 6\\7 & 8 & 9\end{bmatrix}$$

则 CSHIFT(M, SHIFT=−1, DIM=2)的值为

$$\begin{bmatrix}3 & 1 & 2\\6 & 4 & 5\\9 & 7 & 8\end{bmatrix}$$

CSHIFT(M, SHIFT=(/−1, 1, 0/), DIM=2)的值为

$$\begin{bmatrix}3 & 1 & 2\\5 & 6 & 4\\7 & 8 & 9\end{bmatrix}$$

26. DATE _ AND _ TIME(DATE,TIME,ZONE,VALUES)　DATE,TIME,ZONE,VALUES 任选

描述:回送实时时钟上的各种数据，并且日期的表示形式与 ISO 8601:1988 标准规定的形式兼容。

分类:子程序。

变元:DATE(任选)必须是默认字符类型的标量，并且为了

能包含完整的值其长度必须至少为8。这是一个INTENT(OUT)变元。它的最左面的8个字符被置为形式为CCYYMMDD的值，其中CC为世纪数，YY为该世纪中的年份，MM为该年中的月份，DD为该月中的日子。如果在时钟上无日期可取，则它们被置成空格。

TIME(任选)必须是默认字符类型的标量，并且为了能包含完整的值其长度必须至少为10。这是一个INTENT(OUT)变元。它的最左面的10个字符被置为形式为hhmmss.sss的值，其中hh为该日中的小时数，mm为该小时中的分数，ss.sss为该分中的秒数和毫秒数。如果没有时钟，则它们被置成空格。

ZONE(任选)必须是默认字符类型的标量，并且为了能包含完整的值其长度必须至少为5。这是一个INTENT(OUT)变元。它的最左面的5个字符被置为形式为±hhmm的值，其中hh和mm表示相对于坐标通用时间(Coordinated Universal Time)的时差，分别以小时数和分数表示。如果没有时钟，则它们被置成空格。

VALUES(任选)必须是默认整型的，并且秩为1。这是一个INTENT(OUT)变元。它的大小必须至少为8。在VALUES中回送的值如下：

VALUES(1)年份(例如1990)，或者若在时钟上无日期可取，则回送－HUGE(0)。

VALUES(2)该年份中的月份，或者若在时钟上无日期可取，则回送－HUGE(0)。

VALUES(3)该月份中的日子，或者若在时钟上无日期可取，则回送－HUGE(0)。

VALUES(4)相对于坐标通用时间(Coordinated Universal Time)的时差，以分数表示。或者若这种信息取不到，则回送－HUGE(0)。

VALUES(5)该日中的小时数，范围在0到23之间；或者若没有时钟，则回送－HUGE(0)。

VALUES(6)该小时中的分数，范围在 0 到 59 之间；或者若没有时钟，则回送－HUGE(0)。

VALUES(7)该分中的秒数，范围在 0 到 60 之间；或者若没有时钟，则回送－HUGE(0)。

VALUES(8)该秒中的毫秒数，范围在 0 到 999 之间；或者若没有时钟，则回送－HUGE(0)。

例子：

```
INTEGER DATE_TIME(8)
CHARACTER (LEN=10) BIG_BEN(3)
CALL DATE_AND_TIME(BIG_BEN(1), BIG_BEN(2), &
                   BIG_BEN(3), DATE_TIME)
```

如果在瑞士日内瓦，在1985年4月12日时刻15:27:35.5时调用，则BIG_BEN(1)的值为19850412，BIG_BEN(2)的值为152735.500，BIG_BEN(3)的值为＋0100，而DATE_TIME的元素的值分别为：1985，4，12，60，15，27，35，和 500。

注：坐标通用时间(Coordinated Universal Time)又称格林威治时间(GMT)，由 CCIR Recommendation 460－2定义。

27. DBLE(A)

描述：转换为双精度实型。

分类：元素函数。

变元：A 必须是整型、实型、或复型。

结果类型和类型参数：双精度实型。

结果值：情况1　如果 A 为双精度实型，则 DBLE(A)＝A。

情况2　如果 A 为整型或实型，则结果是把 A 的有效位数部分作为一个双精度实型数后求得的结果。

情况3　如果 A 为复型，则结果是把 A 的实部的有效位数部分作为一个双精度实型数后求得的结果。

例子：DBLE(－3)有值－3.0D0。

28. DIGITS(X)

描述：回送表示与变元具有相同类型和种别类型参数的数的

模型中有效数字的位数 s。

分类：查询函数。

变元：X 必须是整型或实型的。它可是标量或数组。

结果类型、类型参数和形状：默认整型标量。

结果值：如果 X 是整型，结果的值为 q；如果 X 是实型，结果的值为 p；其中 q 和 p 就是在本章（11. Ⅱ，Ⅲ）中表示与 X 具有相同类型和种别类型参数的数的模型中所包含的 q 和 p。

例子：若 X 是在本章（11. Ⅴ）中定义的模型所表示的实型数，则 DIGITS(X)的值为24。

29. DIM(X,Y)

描述：如果 X 与 Y 的差 X－Y 为正，则结果就是 X－Y；否则为0。

分类：元素函数。

变元：X 必须是整型或实型。

Y 必须与 X 有相同的类型和种别类型参数。

结果类型和类型参数：与 X 的相同。

结果值：如果 X>Y，则结果为 X－Y；否则为0。

例子：DIM(－3.0,2.0)有值 0.0 。

30. DOT _ PRODUCT(VECTOR _ A，VECTOR _ B)

描述：执行数值或逻辑向量的点积(或称内积)乘法。

分类：变换函数。

变元：VECTOR _ A 必须是数值类型(整型、实型或复型)的或逻辑型的，而且它必须是秩为1的数组。

VECTOR _ B 当 VECTOR _ A 为数值类型时必须是数值类型的；当 VECTOR _ A 为逻辑类型时必须是逻辑型的。而且它必须是秩为1的数组，并与 VECTOR _ A 具有相同的大小。

结果类型、类型参数和形状：如果变元是数值类型的，则结果的类型和种别类型参数就是表达式 VECTOR _ A * VECTOR _ B 的相应类型和种别类型参数，它按照关于表达式的类型和类型参数的章节所述确定。如果变元是逻辑型的，则结果是逻辑型的，

其种别类型参数就是表达式 VECTOR _ A. AND. VECTOR _ B 的相应种别类型参数，它按照关于表达式的类型和类型参数的章节所述确定。结果都是一个标量。

结果值：情况1　如果 VECTOR _ A 为整型或实型，则结果有值 SUM(VECTOR _ A * VECTOR _ B)。如果向量的大小为0，则结果的值为0。

情况2　如果 VECTOR _ A 为复型，则结果有值 SUM (CONJG(VECTOR _ A) * VECTOR _ B)。如果向量的大小为0，则结果的值为0。

情况3　如果 VECTOR _ A 为逻辑型，则结果有值 ANY (VECTOR _ A. AND. VECTOR _ B)。如果向量的大小为0，则结果的值为假。

例子：DOT _ PRODUCT((/1,2,3/),(/2,3,4/))的值为20。

31. DPROD(X,Y)

描述：双精度实数乘。

分类：元素函数。

变元：X 必须是默认实型。

Y 必须是默认实型。

结果类型和类型参数：双精度实型。

结果值：结果等于 X 和 Y 的乘积的依赖处理系统的近似值。

例子：DPROD(－3.0，2.0) 有值 －6.0D0。

32. EOSHIFT (ARRAY，SHIFT，BOUNDARY，DIM) BOUNDARY，DIM 任选

描述：在一个秩为1的数组表达式上作去端(end _ off)移位，或沿着秩为2或更大秩的数组表达式的给定维在所有秩为1 的完整数组片段上作去端移位。在一个数组片段的一端被移出的元素被丢弃，并在另一端移入相同个数的 BOUNDARY 的值。不同的片段可以有不同的 BOUNDARY 值，并可在不同的方向移不同的位数。

分类：变换函数。

变元：ARRAY 可是任何类型，但必须不是标量。

SHIFT 当 ARRAY 秩为1时必须为整型标量；否则它必须是标量，或者是秩为 n－1，形状为$(d_1,d_2,\cdots,d_{DIM-1},d_{DIM+1},\cdots,d_n)$的数组，其中$(d_1,d_2,\cdots,d_n)$是 ARRAY 的形状。

BOUNDARY(任选)必须与 ARRAY 有相同的类型和类型参数，并且当 ARRAY 秩为1时必须为标量；否则它必须或者是标量，或者是秩为 n－1，形状为$(d_1,d_2,\cdots,d_{DIM-1},d_{DIM+1},\cdots,d_n)$的数组，其中$(d_1,d_2,\cdots,d_n)$是 ARRAY 的形状。BOUNDARY 对于下表中的数据类型可被省略，此时，等价于 BOUNDARY 是表中所示的标量。

数组的类型	BOUNDARY 的值
整型	0
实型	0.0
复型	(0.0,0.0)
逻辑型	假
字符型(len)	len 个空格

DIM(任选)必须是值在1≤DIM≤n 之间的整型标量，其中 n 是 ARRAY 的秩。如果 DIM 被省略，它就认为具有值1。

结果类型、类型参数和形状：结果的类型、类型参数和形状与 ARRAY 的相同。

结果值：结果的元素$(s_1,s_2,\cdots,s_n)$的值等于 ARRAY$(s_1,s_2,\cdots,s_{DIM-1},sh+s_{DIM},s_{DIM+1},\cdots,s_n)$，其中 sh 为 SHIFT 或 SHIFT$(s_1,s_2,\cdots,s_{DIM-1},s_{DIM+1},\cdots,s_n)$，倘若不等式 LBOUND(ARRAY，DIM)$\leq sh+s_{DIM}\leq$UBOUND(AR _ RAY,DIM)成立。否则，sh 为 BOUNDARY 或 BOUNDARY$(s_1,s_2,\cdots,s_{DIM-1},s_{DIM+1},\cdots,s_n)$。

例子：情况1　如果 V 是数组[1,2,3,4,5,6]，则 EOSHIFT(V，SHIFT＝3)去端左移三位的结果为[4,5,6,0,0,0]；而用 EOSHIFT(V，SHIFT＝－2，BOUNDARY＝99)去端右移两位的结果为[99,99,1,2,3,4]。

情况2　一个秩为2的数组的各行可被移相同的位数，也可被移不同的位数，并且可以相同也可不同。设 M 是数组

$$\begin{bmatrix} A & B & C \\ D & E & F \\ G & H & I \end{bmatrix}$$

则 EOSHIFT(M, SHIFT=-1, BOUNDARY='*', DIM=2)的值为

$$\begin{bmatrix} * & A & B \\ * & D & E \\ * & G & H \end{bmatrix}$$

EOSHIFT(M, SHIFT=(/-1, 1, 0/), BOUNDARY=(/'*', '/', '?'/), DIM=2)的值为：

$$\begin{bmatrix} * & A & B \\ E & F & / \\ G & H & I \end{bmatrix}$$

33. EPSILON(X)

描述：返回一个与变元具有相同类型和种别类型参数的模型表示数中的单位相比几乎可以忽略的正的小模型数。

分类：查询函数。

变元：X 必须是实型的。它可是标量或数组。

结果类型、类型参数和形状：与变元 X 具有相同类型和种别类型参数的标量。

结果值：结果的值为 b^{1-p}，其中 b 和 p 就是在本章(11. Ⅲ)中定义的与 X 具有相同类型和种别类型参数的模型表示数中的那些参数。

例子：若 X 是在本章(11. Ⅴ)定义的模型所表示的实型数，则 EPSILON(X)的值为2^{-23}。

34. EXP(X)

描述：指数函数。

分类：元素函数。

变元：X 必须是实型或复型的。

结果类型和类型参数：与 X 的相同。

结果值：结果为依赖处理系统的 e^X 的近似值。如果 X 为复型的，则它的虚部被认为是一个弧度为单位的值。

例子：EXP(1.0) 近似地有值 2.7182818。

35. EXPONENT(X)

描述：当变元 X 表示为模型数时，回送它的指数部分。

分类：元素函数。

变元：X 必须是实型的。

结果类型：默认整型。

结果值：结果的值为 X 值的本章(11. Ⅲ)模型表示的指数部分 e，并且 e 在默认整数的范围之内。如果处理系统不能把 e 表示成默认整型，结果不定义。如果 X 为0，EXPONENT(X)的值也是0。

例子：如果实数按本章(11. Ⅴ)的模型定义，则 EXPONENT(1.0)有值1，EXPONENT(4.0) 有值 3。

36. FLOOR(A)

描述：回送小于等于其变元值的最大整数。

分类：元素函数。

变元：A 必须是实型的。

结果类型和类型参数：默认整型。

结果值：结果的值为小于等于其变元 A 的值的最大整数。如果处理系统不能把这个值表示成默认整型，结果不定义。

例子：FLOOR(3.7)有值 3。FLOOR(−3.7)有值 −4。

37. FRACTION(X)

描述：回送变元值的模型表示的分数部分。

分类：元素函数。

变元：X 必须是实型的。

结果类型和类型参数：与 X 的相同。

结果值：结果的值为 $X * b^{-e}$，其中 b 和 e 就是如同在本章(11. Ⅲ)中定义的 X 的模型表示所包含的 b 和 e。如果 X 为0，结果的值也是0。

例子：如果实数按本章(11. Ⅴ)的模型定义，则 FRACTION (3.0) 有值 0.75。

38. HUGE(X)

描述：返回一个与变元具有相同类型和种别类型参数的模型表示数中的最大的数。

分类：查询函数。

变元：X 必须是整型或实型的。它可以是标量或数组。

结果类型、类型参数和形状：与变元 X 具有相同类型和种别类型参数的标量。

结果值：如果 X 为整型，结果的值为 r^q-1；如果 X 为实型，结果的值为 $(1-b^{-p})*b^{emax}$。其中 r、q、b、p 和 e^{max} 就是在本章(11. Ⅱ，Ⅲ)中定义的与 X 具有相同类型和种别类型参数的模型表示数中的那些参数。

例子：若 X 是在本章(11. Ⅴ)定义的模型所表示的实型数，则 HUGE(X)的值为 $(1-2^{-24})*2^{127}$。

39. IACHAR(C)

描述：回送一个字符在 ASCII 理序序列中的位置数。

分类：元素函数。

变元：A 必须是长度为1 的默认字符类型。

结果类型和类型参数：默认整型。

结果值：如果 C 在理序序列中由 ISO 646：1983(国际参考版)指明的代码定义，则结果是 C 在该序列中的位置，且满足不等式：0≤IACHAR(C)≤127。如果 C 不在 ASCII 理序序列中，则回送一个依赖于处理系统的值。结果应与词法比较函数 LGE、LGT、LLE 和 LLT 的相应结果一致。例如，若 LLE(C,D)为真，则 IACHAR(C).LE.IACHAR(D)也应真，其中 C 和 D 为两个在处理系统中可表示的字符。

例子：IACHAR('X')有值88。

40. IAND(I,J)

描述：执行逻辑 AND。

分类:元素函数。

变元:I 必须是整型的。

J 必须是整型的，且与 I 具有相同的种别类型参数。

结果类型和类型参数:与 I 相同。

结果值:结果的值等于把 I 和 J 按下列真值表逐位运算而得到的值:

I	J	IAND(I,J)
1	1	1
1	0	0
0	1	0
0	0	0

其中，整数值按(11. I)的模型解释为位的序列。

例子:IAND(1,3) 有值 1。

41. IBCLR(I,POS)

描述:把一个位清成0。

分类:元素函数。

变元:I 必须是整型。

POS 也必须是整型，并且是小于 BIT _ SIZE(I)的非负整数。

结果类型和类型参数:与 I 相同。

结果值:结果的值就是 I 的位序列，唯一的区别就是 I 中的第 POS 位已被清成0。这里一个整数值按(11. I)的模型解释为一个位序列。

例子:IBCLR(14,1)有值12。若 V 有值[1,2,3,4]，则 IBCLR(POS=V, I=31)的值为[29,27,23,15]。

42. IBITS(I,POS,LEN)

描述:抽出一个位的序列。

分类:元素函数。

变元:I 必须是整型。

POS 必须是整型的且为非负的，并且 POS+LEN 必须小于

等于 BIT _ SIZE(I)。

LEN 必须是整型的且为非负的。

结果类型和类型参数:与 I 相同。

结果值:结果的值为长度为 LEN 的 I 中的一个位序列，右边对齐，所有其它位为0。这里一个整数值按(11. I)的模型解释为一个位序列。

例子:IBITS(14,1,3)有值7。

43. IBSET(I,POS)

描述:把一个位置成1。

分类:元素函数。

变元:I 必须是整型。

POS 也必须是整型的和非负的，并且小于 BIT _ SIZE(I)。

结果类型和类型参数:与 I 相同。

结果值:结果的值就是 I 的位序列，唯一的区别就是 I 中的第 POS 位已被置成1。这里一个整数值按(11. I)的模型解释为一个位序列。

例子:IBSET(12,1)有值14。若 V 有值[1,2,3,4]，则 IBSET(POS=V，I=0)的值为[2,4,8,16]。

44. ICHAR(C)

描述:回送一个字符在处理系统理序序列中的位置数，以及该字符的种别类型参数。

分类:元素函数。

变元:A 必须是长度为1 的字符类型，它的值必须是处理系统能表示的一个字符。

结果类型和类型参数:默认整型。

结果值:结果是 C 在处理系统理序序列中的位置以及 C 的种别类型参数，且满足不等式：0≤ICHAR(C)≤n，n 为处理系统理序序列中字符的个数。对任意的处理系统能表示的字符 C 和 D，C.LE.D 为真，当且仅当 ICHAR(C).LE.ICHAR(D)为真，并且 C.EQ.D 为真，当且仅当 ICHAR(C).EQ.ICHAR(D)为真，

例子：在用 ASCII 理序序列作为默认字符类型的处理系统上 ICHAR('X')有值88。

45. IEOR(I,J)

描述：执行排除 OR。

分类：元素函数。

变元：I 必须是整型的。

J 必须是整型的，且与 I 具有相同的种别类型参数。

结果类型和类型参数：与 I 相同。

结果值：结果的值等于把 I 和 J 按下列真值表逐位运算而得到的值：

I	J	IEOR(I,J)
1	1	0
1	0	1
0	1	1
0	0	0

其中，整数值按(11. I)的模型解释为位的序列。

例子：IEOR(1,3) 有值 2。

46. INDEX(STRING,SUBSTRING,BACK) BACK 任选

描述：回送一个子串在一个串中的开始位置。

分类：元素函数。

变元：STRING 必须是字符型。

SUBSTRING 必须是字符型的，且与 STRING 有相同的种别类型参数。

BACK(任选)必须是逻辑型的

结果类型：默认整型。

结果值：情况1　如果 BACK 省缺或其值为假，结果是使得 STRING(I:I+LEN(SUBSTRING)－1)＝SUBSTRING 的最小正整数 I；或者若不存在这样的正整数 I，则回送0。如果 LEN(STRING)＜LEN(SUBSTRING)，则回送0；如果 LEN(SUB-

STRING)=0，则回送1。

情况2 如果BACK的值为真，结果是小于等于LEN(STRING)－LEN(SUBSTRING)＋1的最大整数，使得STRING(I:I+LEN(SUBSTRING)－1)=SUBSTRING；或者若不存在这样的正整数I，则回送0。如果LEN(STRING)≤LEN(SUBSTRING)，则回送0；如果LEN(SUBSTRING)=0，则回送LEN(STRING)+1。

例子 INDEX('FORTRAN','R')有值3。INDEX('FORTRAN','R',BACK=.TRUE.)有值5。

47. INT(A,KIND) KIND任选

描述:转换为整型。

分类:元素函数。

变元:A必须是整型、实型或复型。

KIND(任选)必须是一个整型的标量初始化表达式。

结果类型和类型参数:结果为整型。如果有KIND，则种别类型参数由KIND指明；否则为默认整型。

结果值:情况1 如果A为整型，则INT(A)=A。

情况2 如果A为实型，则可分两种情形：若=|A|<1，INT(A)=0；若|A|≥1，INT(A)是一个大小为不超过A的大小的最大整数，其符号与A的相同。

情况3 如果A为复型，则INT(A)是将A的实部作为实型变元应用情况2.求得的结果。

如果处理系统不能把这个结果表示成指定的整型，结果不定义。

例子:INT(－3.7)有值－3。

48. IOR(I,J)

描述:执行OR。

分类:元素函数

变元:I必须是整型的。

J必须是整型的，且与I具有相同的种别类型参数。

结果类型和类型参数：与 I 相同。

结果值：结果的值等于把 I 和 J 按下列真值表逐位运算而得到的值：

I	J	IOR(I,J)
1	1	1
1	0	1
0	1	1
0	0	0

其中，整数值按（11. I ）的模型解释为位的序列。

例子：IOR(1,3) 有值 3。

49. ISHFT(I,SHIFT)

描述：执行一个逻辑移位。

分类：元素函数。

变元：I 必须是整型的。

SHIFT 必须是整型的，且其绝对值必须小于等于 BIT _ SIZE(I)。

结果类型和类型参数：与 I 相同。

结果值：结果的值为将 I 移 SHIFT 位后得到的值。如果 SHIFT 是正的，则左移；如果 SHIFT 是负的，则右移；如果 SHIFT 是0，则不做移位。移出的位被丢失，而在另一端移入0。其中 I 的整数值按（11. I ）的模型解释为位的序列。

例子：ISHFT(3,1) 有值 6。

50. ISHFTC(I,SHIFT,SIZE) SIZE 任选

描述：执行最右一些位的一个循环移位。

分类：元素函数。

变元：I 必须是整型的。

SHIFT 必须是整型的，且其绝对值必须小于等于 SIZE。

SIZE（任选）必须是整型的，并且其值必须是正的且一定不能超过 BIT _SIZE(I)。如果 SIZE 省缺，则就如 SIZE 具有值 BIT _

SIZE(I)。

结果类型和类型参数：与I相同。

结果值：结果的值为将I的最右的SIZE个位循环地移SHIFT位后得到的值。如果SHIFT是正的，则左移；如果SHIFT是负的，则右移；如果SHIFT是0，则不做移位。移位中没有位被丢失，没被移动的位是不改变的。其中I的整数值按(11.1)的模型解释为位的序列。

例子：ISHFTC(3,2,3)有值15。

51. KIND(X)

描述：返回变元X的种别类型参数。

分类：查询函数。

变元：X可是任意内在类型的。

结果类型、类型参数和形状：默认整型标量。

结果值：结果的值是变元X的种别类型参数。

例子：KIND(0.0)有默认实型的种别类型参数值。

52. LBOUND(ARRAY,DIM) DIM任选

描述：返回一个数组的所有下界或某指定的下界。

分类：查询函数。

变元：ARRAY可是任意类型的，但一定不是标量，也不能是一个不结合的指针或者没被分配的可分配数组。

DIM(任选)必须是值在1≤DIM≤n之间的整型标量，其中n是ARRAY的秩，且与DIM对应的实元一定不能是一个任选的虚元。

结果类型、类型参数和形状：结果是默认整型的。如果DIM存在，则结果是标量；否则结果是一个秩为1而大小为n的数组，其中n是ARRAY的秩。

结果值：情况1　对一个不同于完整数组或一个数组结构分量的数组片段或数组表达式，LBOUND(ARRAY,DIM)有值1；否则值为ARRAY的第DIM个下标的下界(当ARRAY的第DIM维的大小不是0时)，值为1(当ARRAY的第DIM维的大小是0

时）。

情况2　LBOUND(ARRAY)的值是一个数组，其第 i 个分量为 LBOUND(ARRAY,i),i=1,2,…,n，其中 n 为 ARRAY 的秩。

例子:如果 A 是由下列语句说明的数组：

REAL A (2:3, 7:10)

则 LBOUND(A)是[2,7]，而 LBOUND(A, DIM=2)是7。

53. LEN(STRING)

描述:返回一个字符实体的长度。

分类:查询函数。

变元:STRING 必须是字符型的。它可是标量或数组。

结果类型和类型参数:默认整型标量。

结果值:当 STRING 为标量时,结果的值为 STRING 中字符的个数；当 STRING 为数组时,结果的值为 STRING 的一个元素包含的字符的个数。

例子:如果 C 是由下列语句说明的数组：

CHARACTER (11) C (100)

则 LEN(C)是11。

54. LEN_TRIM(STRING)

描述:返回字符变元不计尾部空格字符在内的长度。

分类:元素函数。

变元:STRING 必须是字符型的。

结果类型和类型参数:默认整型。

结果值:结果的值为把 STRING 的所有尾部空格去掉之后余下的字符的个数。如果变元不包含非空格字符,则结果的值为0。

例子:LEN_TRIM('AB ')的值为2。LEN_TRIM(' ')的值为0。

55. LGE(STRING_A, STRING_B)

描述:根据 ASCII 理序序列，测试一个串是否在词法上大于等于另一个串。

分类:元素函数。

变元:STRING _ A 必须是默认字符型的。

STRING _ B 必须是默认字符型的。

结果类型和类型参数:默认逻辑型。

结果值:如果两个串不等长，则应将较短的串在其右边补空格使两者等长后再比较。如果被比较的串中包含非 ASCII 字符，则结果的值是依赖处理系统的。如果两个串相同或按 ASCII 理序序列 STRING _ A 跟在 STRING _ B 之后，则结果为真;否则为假。注意:若两个串的长度均为0,则结果的值为真。

例子:LGE('ONE','TWO')有值假。

56. LGT(STRING _ A, STRING _ B)

描述:根据 ASCII 理序序列，测试一个串是否在词法上大于另一个串。

分类:元素函数。

变元:STRING _ A 必须是默认字符型的。

STRING _ B 必须是默认字符型的。

结果类型和类型参数:默认逻辑型。

结果值:如果两个串不等长，则应将较短的串在其右边补空格使两者等长后再比较。如果被比较的串中包含非 ASCII 字符，则结果的值是依赖处理系统的。如果两个串按 ASCII 理序序列 STRING _ A 跟在 STRING _ B 之后，则结果为真;否则为假。注意:若两个串的长度均为0,则结果的值为假。

例子:LGT('ONE','TWO')有值假。

57. LLE(STRING _ A, STRING _ B)

描述:根据 ASCII 理序序列，测试一个串是否在词法上小于等于另一个串。

分类:元素函数。

变元:STRING _ A 必须是默认字符型的。

STRING _ B 必须是默认字符型的。

结果类型和类型参数:默认逻辑型。

结果值:如果两个串不等长，则应将较短的串在其右边补空格使两者等长后再比较。如果被比较的串中包含非 ASCII 字符，则结果的值是依赖处理系统的。如果两个串相同或按 ASCII 理序序列 STRING _ A 在 STRING _ B 之前，则结果为真；否则为假。注意：若两个串的长度均为0,则结果的值为真。

例子:LLE('ONE','TWO')有值真。

58. LLT(STRING _ A, STRING _ B)

描述:根据 ASCII 理序序列，测试一个串是否在词法上小于另一个串。

分类:元素函数。

变元:STRING _ A 必须是默认字符型的。

STRING _ B 必须是默认字符型的。

结果类型和类型参数:默认逻辑型。

结果值:如果两个串不等长，则应将较短的串在其右边补空格使两者等长后再比较。如果被比较的串中包含非 ASCII 字符，则结果的值是依赖处理系统的。如果两个串按 ASCII 理序序列 STRING _ A 在 STRING _ B 之前，则结果为真；否则为假。注意：若两个串的长度均为0,则结果的值为真。

例子:LLT('ONE','TWO')有值假。

59. LOG(X)

描述:自然对数函数。

分类:元素函数。

变元:X 必须是实型或复型的。如果 X 为实型，则其值必须大于零；如果 X 为复型，则其值必须不是零。

结果类型和类型参数:与 X 的相同。

结果值:结果为依赖处理系统的 LOG(X)的近似值。复型的结果是虚部 ω 在范围 $-\pi\leqslant\omega\leqslant\pi$ 之中的主值。仅当变元的实部小于0而且虚部为0时结果的虚部才为 π。

例子:LOG(10.0) 近似地有值 2.3025851。

60. LOG10(X)

描述：公用对数函数。

分类：元素函数。

变元：X 必须是实型的，其值必须大于零。

结果类型和类型参数：与 X 的相同。

结果值：结果为依赖处理系统的 $LOG_{10}X$ 的近似值。

例子：LOG10(10.0) 近似地有值 1.0。

61. LOGICAL(L,KIND) KIND 任选

描述：在逻辑种别之间进行转换。

分类：元素函数。

变元：L 必须是逻辑型的。

KIND(任选) 必须是一个整型的标量初始化表达式。

结果类型和类型参数：结果为逻辑型。如果有 KIND，则其种别类型参数由 KIND 指明；否则为默认逻辑型。

结果值：结果的值就是 L 的值。

例子：不管逻辑变量 L 的种别类型参数如何，LOGICAL(L.OR..NOT.L))的值都为真，且为默认逻辑型。

62. MATMUL(MATRIX_A，MATRIX_B)

描述：执行数值或逻辑矩阵的矩阵乘法。

分类：变换函数。

变元：MATRIX_A 必须是数值类型(整型、实型或复型) 的或逻辑型的，而且它必须是秩为1或2的数组。

MATRIX_B 当 MATRIX_A 为数值类型时必须是数值类型的；当 MATRIX_A 为逻辑型时必须是逻辑型的。而且它必须是秩为1或2的数组。如果 MATRIX_A 有秩1，则 MATRIX_B 必须有秩2。如果 MATRIX_B 有秩1，则 MATRIX_A 必须有秩2。MATRIX_B 的第一个(或唯一的)维的大小必须等于 MATRIX_A 的最后一个(或唯一的)维的大小。

结果类型、类型参数和形状：如果变元是数值类型的，则结果的类型和种别类型参数由变元的类型按照关于表达式的类型和类型参数的章节所述确定。如果变元是逻辑型的，则结果是逻辑型

的，其种别类型参数由变元的种别类型参数按照关于表达式的类型和类型参数的章节所述确定。结果的形状按如下的规则依赖于变元的形状：

情况1　如果 MATRIX_A 的形状为(n,m)，MATRIX_B 的形状为(m,k)，则结果的形状为(n,k)。

情况2　如果 MATRIX_A 的形状为(m)，MATRIX_B 的形状为(m,k)，则结果的形状为(k)。

情况3　如果 MATRIX_A 的形状为(n,m)，MATRIX_B 的形状为(m)，则结果的形状为(n)。

结果值：情况1　如果变元为数值类型，则结果的元素(i,j)有值 SUM(MATRIX_A(i,:) * MATRIX_B(:,j))。如果变元为逻辑型，则结果的元素(i,j)有值 ANY(MATRIX_A(i,:) .AND. MATRIX_B(:,j))。

情况2　如果变元为数值类型，则结果的元素(j)有值 SUM(MATRIX_A(:) * MATRIX_B(:,j))。如果变元为逻辑型，则结果的元素(j)有值 ANY(MATRIX_A(:). AND. MATRIX_B(:,j))。

情况3　如果变元为数值类型，则结果的元素(i)有值 SUM(MATRIX_A(i,:) * MATRIX_B(:))。如果变元为逻辑型，则结果的元素(i)有值 ANY(MATRIX_A(i,:). AND. MATRIX_B(:))。

例子：设 A 和 B 为矩阵：

$$\begin{bmatrix} 1 & 2 & 3 \\ 2 & 3 & 4 \end{bmatrix} \text{和} \begin{bmatrix} 1 & 2 \\ 2 & 3 \\ 3 & 4 \end{bmatrix}$$

X 和 Y 为向量[1，2]和[1，2，3]。

情况1　结果 MATMUL(A,B)是矩阵一矩阵乘积 AB 具有值：

$$\begin{bmatrix} 14 & 20 \\ 20 & 29 \end{bmatrix}$$

情况2　结果 MATMUL(X,A)是向量一矩阵乘积 XA 具有值[5,8,11]。

情况3　结果 MATMUL(A,Y)是矩阵一向量乘积 AY 具有值[14,20]。

63. MAX(A1,A2,A3,…) A3,…任选

描述:求极大值。

分类:元素函数。

变元:变元必须是整型或实型的，并且所有变元必须具有相同的类型和种别类型参数。

结果类型和类型参数:与变元的相同。

结果值:结果的值为最大变元的值。

例子:MAX(-9.0，7.0，2.0)有值7.0。

64. MAXEXPONENT(X)

描述:回送表示与变元具有相同类型和类型参数的数的模型的最大指数部分。

分类:查询函数。

变元:X 必须是实型的。它可以是标量或数组。

结果类型、类型参数和形状:默认整型标量。

结果值:结果的值为表示与变元 X 具有相同类型和类型参数的数的本章(11. Ⅲ)模型的最大指数部分 e_{max}。

例子:如果实数按本章(11. Ⅴ)的模型定义，则 MAXEXPONENT(X) 有值127。

65. MAXLOC(ARRAY,MASK) MASK 任选

描述:根据 MASK 的真元素确定 ARRAY 的第一个最大元素的位置。

分类:变换函数。

变元:ARRAY 必须是整型或实型的，但一定不是标量。

MASK(任选)必须是逻辑型的，并与 ARRAY 有相同的形状。

结果类型、类型参数和形状:结果是默认整型的。结果是一个

秩为1而大小与 ARRAY 的秩相同的数组。

结果值:情况1　如果 MASK 省缺，则结果为秩为1 的数组，其元素的值是 ARRAY 中具有 ARRAY 的所有元素的最大值的一个元素的下标值。其中第 i 个下标在1到 e_i 之间取值，这里 e_i 是 ARRAY 第 i 个维的大小。如果有多个元素具有最大值，则按数组元素的次序选送第一个最大元素的下标值作为结果。如果 ARRAY 的大小为0,则结果的值依赖于处理系统。

情况2　如果 MASK 存在，则结果为秩为1 的数组，其元素的值按如下的步骤获得：首先从 ARRAY 中去掉 MASK 中对应位置为假的那些元素;然后在处理后的 ARRAY 中按情况1. 所述回送具有最大值的一个元素的下标值。其中第 i 个下标在1到 e_i 之间取值，这里 e_i 是 ARRAY 第 i 个维的大小。如果在处理后的 ARRAY 中有多个元素具有最大值，则按数组元素的次序选送第一个最大元素的下标值作为结果。如果没有这样的最大元素，即或者 ARRAY 的大小为0，或者 MASK 的每个元素都是假，则结果的值依赖于处理系统。

如果处理系统不能把结果的某个元素的值表示成默认整型，则结果的该元素不定义。

例子:情况1　MAXLOC((/2,6,4,6/))的值为[2]。

情况2　如果 A 的值是下列数组：

$$\begin{bmatrix} 0 & -5 & 8 & -3 \\ 3 & 4 & -1 & 2 \\ 1 & 5 & 6 & -4 \end{bmatrix}$$

则 MAXLOC(A,MASK=A.LT.6)有值[3,2]。注意：即使 A 在说明中下界不是1，结果也一样。

66. MAXVAL(ARRAY,DIM,MASK) DIM，MASK 任选

描述:沿着第 DIM 维对应 MASK 的真元素确定 ARRAY 的最大值。

分类:变换函数。

变元:ARRAY 必须是整型或实型的，但一定不是标量。

DIM(任选)必须是值在$1 \leqslant DIM \leqslant n$之间的整型标量，其中 n 是 ARRAY 的秩，且与 DIM 对应的实元一定不能是一个任选的虚元。

MASK(任选)必须是逻辑型的，并与 ARRAY 有相同的形状。

结果类型、类型参数和形状：结果与 ARRAY 具有相同的类型和种别类型参数。如果 DIM 省缺或 ARRAY 的值为1，则结果是一个标量；否则结果是秩为 n－1，形状为$(d_1, d_2, \cdots, d_{DIM-1}, d_{DIM+1}, \cdots, d_n)$的数组，其中$(d_1, d_2, \cdots, d_n)$是 ARRAY 的形状。

结果值：情况1　如果 MASK 省缺，则 MAXVAL(ARRAY) 的结果为 ARRAY 的所有元素的最大值。如果 ARRAY 的大小为0，则 MAXVAL(ARRAY)的结果为处理系统所支持的与 ARRAY 具有相同的类型和种别类型参数的最大数的负值。

情况2　如果 MASK 存在，则结果的值按如下的步骤获得：首先从 ARRAY 中去掉 MASK 中对应位置为假的那些元素；然后在处理后的 ARRAY 中按情况1所述回送处理后的 ARRAY 中的最大值。如果 ARRAY 的大小为0或 MASK 中都是假元素，则 MAXVAL(ARRAY)的结果为处理系统所支持的与 ARRAY 具有相同的类型和种别类型参数的最大数的负值。

情况3　若 ARRAY 的秩为1，则 MAXVAL(ARRAY，DIM〔,MASK〕)的值等于 MAXVAL(ARRAY〔,MASK＝MASK〕)；否则 MAXVAL(ARRAY，DIM〔,MASK〕)的元素$(s_1, s_2, \cdots, s_{DIM-1}, s_{DIM+1}, \cdots, s_n)$的值等于 MAXVAL$(ARRAY(s_1, s_2, \cdots, s_{DIM-1}, :, s_{DIM+1}, \cdots, s_n)$〔, MASK ＝ MASK$(s_1, s_2, \cdots, s_{DIM-1}, :, s_{DIM+1}, \cdots, s_n)$〕)。

例子：情况1　MAXVAL((/1,2,3/))的值为3。

情况2　MAXVAL(C,MASK＝C. LT. 0. 0)将形成 C 的负元素的最大值。

情况3　如果 B 的值是下列数组：

$$\begin{bmatrix} 1 & 3 & 5 \\ 2 & 4 & 6 \end{bmatrix}$$

则 MAXVAL(B,DIM＝1)有值〔2,4,6〕。MAXVAL(B,DIM＝2)有值〔5,6〕。

67. MERGE(TSOURCE,FSOURCE,MASK)

描述：根据 MASK 的值选择交替的值。

分类：元素函数。

变元：TSOURCE 可是任意类型的。

FSOURCE 必须与 TSOURCE 具有相同的类型和类型参数。

MASK 必须是逻辑型的。

结果类型和类型参数：与 TSOURCE 相同。

结果值：如果 MASK 真，结果为 TSOURCE；否则结果为 FSOURCE。

例子：如果 TSOURCE、FSOURCE 和 MASTK 分别为数组：

$$\begin{bmatrix} 1 & 6 & 5 \\ 2 & 4 & 6 \end{bmatrix}、\begin{bmatrix} 0 & 3 & 2 \\ 7 & 4 & 8 \end{bmatrix}和\begin{bmatrix} T & \cdot & T \\ \cdot & \cdot & T \end{bmatrix}$$

这里"T"表示真，"·"表示假，则 MERGE(TSOURCE,FSOURCE,MASK)的值为：

$$\begin{bmatrix} 1 & 3 & 5 \\ 7 & 4 & 6 \end{bmatrix}$$

当 K＝5时，MERGE(1.0,0.0,K>0)的值为1.0，而当 K＝－2时为0.0。

68. MIN(A1,A2,A3,…) A3,…任选

描述：求极小值。

分类：元素函数。

变元：变元必须是整型或实型的，并且所有变元必须具有相同的类型和种别类型参数。

结果类型和类型参数：与变元的相同。

结果值：结果的值为最小变元的值。

例子：MIN(－9.0，7.0，2.0)有值－9.0。

69. MINEXPONENT(X)

描述:回送表示与变元具有相同类型和类型参数的数的模型的最小(最负)指数部分。

分类:查询函数。

变元:X 必须是实型的。它可以是标量或数组。

结果类型、类型参数和形状:默认整型标量。

结果值:结果的值为表示与变元 X 具有相同类型和类型参数的数的本章(11. Ⅲ)模型的最小指数部分 e_{min}。

例子:如果实数按本章(11. Ⅴ)的模型定义,则 MINEXPONENT(X)有值－126。

70. MINLOC(ARRAY,MASK) MASK 任选

描述:根据 MASK 的真元素确定 ARRAY 的第一个最小元素的位置。

分类:变换函数。

变元:ARRAY 必须是整型或实型的,但一定不是标量。

MASK(任选)必须是逻辑型的,并与 ARRAY 有相同的形状。

结果类型、类型参数和形状:结果是默认整型的。结果是一个秩为1而大小与 ARRAY 的秩相同的数组。

结果值:情况1　如果 MASK 省缺,则结果为秩为1 的数组,其元素的值是 ARRAY 中具有 ARRAY 的所有元素的最小值的一个元素的下标值。其中第 i 个下标在1到 e_i 之间取值,这里 e_i 是 ARRAY 第 i 个维的大小。如果有多个元素具有最小值,则按数组元素的次序选送第一个最小元素的下标值作为结果。如果 ARRAY 的大小为0,则结果的值依赖于处理系统。

情况2　如果 MASK 存在,则结果为秩为1 的数组,其元素的值按如下的步骤获得:首先从 ARRAY 中去掉 MASK 中对应位置为假的那些元素;然后在处理后的 ARRAY 中按情况1. 所述回送具有最小值的一个元素的下标值。其中第 i 个下标在1到 e_i 之间取值,这里 e_i 是 ARRAY 第 i 个维的大小。如果在处理后的

ARRAY 中有多个元素具有最小值，则按数组元素的次序选送第一个最小元素的下标值作为结果。如果没有这样的最小元素，即或者 ARRAY 的大小为0，或者 MASK 的每个元素都是假，则结果的值依赖于处理系统。

如果处理系统不能把结果的某个元素的值表示成默认整型，则结果的该元素不定义。

例子：情况1　MINLOC((/4,3,6,3/))的值为[2]。

情况2　如果 A 的值是下列数组：

$$\begin{bmatrix} 0 & -5 & 8 & -3 \\ 3 & 4 & -1 & 2 \\ 1 & 5 & 6 & -4 \end{bmatrix}$$

则 MINLOC(A,MASK=A.GT.-4)有值[1,4]。注意：即使 A 在说明中下界不是1，结果也一样。

71. MINVAL(ARRAY,DIM,MASK) DIM，MASK 任选

描述：沿着第 DIM 维对应 MASK 的真元素确定 ARRAY 的最小值。

分类：变换函数。

变元：ARRAY 必须是整型或实型的，但一定不是标量。

DIM(任选)必须是值在1≤DIM≤n 之间的整型标量，其中 n 是 ARRAY 的秩，且与 DIM 对应的实元一定不能是一个任选的虚元。

MASK(任选)必须是逻辑型的，并与 ARRAY 有相同的形状。

结果类型、类型参数和形状：结果与 ARRAY 具有相同的类型和种别类型参数。如果 DIM 省缺或 ARRAY 的值为1，则结果是一个标量；否则结果是秩为 n－1，形状为 $(d_1,d_2,\cdots,d_{DIM-1},d_{DIM+1},\cdots,d_n)$的数组，其中$(d_1,d_2,\cdots,d_n)$是 ARRAY 的形状。

结果值：情况1　如果 MASK 省缺，则 MINVAL(ARRAY)的结果为 ARRAY 的所有元素的最小值。如果 ARRAY 的大小为0，则 MINVAL(ARRAY)的结果为处理系统所支持的与 AR-

RAY 具有相同的类型和种别类型参数的最大数的正值。

情况2　如果 MASK 存在，则结果的值按如下的步骤获得：首先从 ARRAY 中去掉 MASK 中对应位置为假的那些元素；然后在处理后的 ARRAY 中按情况1所述回送处理后的 ARRAY 中的最小值。如果 ARRAY 的大小为0或 MASK 中都是假元素，则 MINVAL(ARRAY)的结果为处理系统所支持的与 ARRAY 具有相同的类型和种别类型参数的最大数的正值。

情况3　若 ARRAY 的秩为1，则 MINVAL(ARRAY，DIM〔,MASK〕)的值等于 MINVAL(ARRAY〔,MASK＝MASK〕)；否则 MINVAL(ARRAY，DIM〔,MASK〕)的元素($s_1, s_2, \cdots, s_{DIM-1}, s_{DIM+1}, \cdots, s_n$)的值等于 MINVAL(ARRAY($s_1, s_2, \cdots, s_{DIM-1}, :, s_{DIM+1}, \cdots, s_n$)〔,MASK = MASK($s_1, s_2, \cdots, s_{DIM-1}, :, s_{DIM+1}, \cdots, s_n$)〕)。

例子：情况1　MINVAL((/1,2,3/))的值为1。

情况2　MINVAL(C,MASK＝C.GT.0.0)将形成 C 的正元素的最小值。

情况3　如果 B 的值是下列数组：

$$\begin{bmatrix} 1 & 3 & 5 \\ 2 & 4 & 6 \end{bmatrix}$$

则 MINVAL(B,DIM＝1)有值〔1,3,5〕。MINVAL(B,DIM＝2)有值〔1,2〕。

72. MOD(A,P)

描述：余数函数。

分类：元素函数。

变元：A 必须是整型或实型的。

P 必须与 A 具有相同的类型和种别类型参数。

结果类型和类型参数：与 A 相同。

结果值：如果 P≠0，则结果的值等于 A－INT(A/P) * P。如果 P＝0，则结果依赖于处理系统。

例子：MOD(3.0,2.0)近似地有值1.0。MOD(8,5)有值3。

MOD(−8,5)有值−3。MOD(8,−5)有值3。MOD(−8,−5)有值−3。

73. MODULO(A,P)

描述:模数函数。

分类:元素函数。

变元:A 必须是整型或实型的。

P 必须与 A 具有相同的类型和种别类型参数。

结果类型和类型参数:与 A 相同。

结果值:当 A 为整型时，如果 $P \neq 0$，则 MODULO(A,P)的值 R 使得 $A = Q \times P + R$ 成立，其中 Q 是一个整数。当 $P > 0$时，不等式$0 \leqslant R \leqslant P$ 成立；当 $P < 0$时，不等式 $P \leqslant R \leqslant 0$成立；当 $P = 0$时，结果依赖于处理系统。

例子:MODULO(8,5)有值3。MODULO(−8,5)有值2。MODULO(8,−5)有值−2。MODULO(−8,−5)有值−3。

74. MVBITS(FROM,FROMPOS,LEN,TO,TOPOS)

描述:从一个数据对象拷贝一个位的序列到另一个数据对象。

分类:元素子程序。

变元:FROM 必须整型的，它是一个 INTENT(IN)变元。

FROMPOS 必须是整型的且为非负的，它是一个 INTENT(IN)变元。并且 FROMPOS+LEN 必须小于等于 BIT_SIZE(FROM)。这里一个整数值按本章(11.1)的模型解释为一个位序列。

LEN 必须是整型的且为非负的，它是一个 INTENT(IN)变元。

TO 必须是一个与 FROM 具有种别类型参值的整型变量，并且可以就是变量 FROM。它是一个 INTENT(OUT)变元。从 FROM 的第 FROMPOS 位开始拷贝长度为 LEN 的一个位的序列到 TO 的第 TOPOS 位处，TO 的其它位不变。在返回时，TO 中从 TOPOS 开始的 LEN 个位等于 FROM 在进入该子程序时从 FROMPOS 开始的 LEN 个位的值。这里一个整数值按本章(11.

1)的模型解释为一个位序列。

TOPOS 必须是整型的且为非负的，它是一个 INTENT(IN) 变元。并且 TOPOS+LEN 必须小于等于 BIT _ SIZE(TO)。

结果类型和类型参数：与 TO 相同。

结果值：从 FROM 的第 FROMPOS 位开始拷贝长度为 LEN 的一个位的序列到 TO 的第 TOPOS 位处，TO 的其它位不变。故在返回时，TO 中从 TOPOS 开始的 LEN 个位等于 FROM 在进入该子程序时从 FROMPOS 开始的 LEN 个位的值。这里一个整数值按本章(11. 1)的模型解释为一个位序列。

例子：如果 TO 有初始值6，则在执行 CALL MVBITS(7,2,2,TO,0)后变为5。

75. NEAREST(X,S)

描述：回送在给定方向上最近的一个机器可表示的数。

分类：元素函数。

变元：X 必须是实型的。

S 必须是实型的且不等于0。

结果类型和类型参数：与 X 相同。

结果值：结果的值为在由 S 的符号指定的方向上最靠近 X 且与 X 不同的一个机器可表示的数，。

例子：MODULO(8,5)有值3。MODULO(-8,5)有值2 。MODULO(8,-5)有值-2。MODULO(-8,-5)有值-3。

76. NINT(A,KIND) KIND 任选

描述：回送最近的整数。

分类：元素函数。

变元：A 必须是实型的。

KIND(任选) 必须是一个整型的标量初始化表达式。

结果类型和类型参数：结果为整型。如果有 KIND，则种别类型参数由 KIND 指明；否则为默认整型。

结果值：如果 A>0，则 NINT(A)有值 INT(A+0.5)；如果 A≤0，则 NINT(A) 有值 INT(A-0.5)。如果处理系统不能把这

个结果表示成指定的整型，结果不定义。

例子:NINT(2.783) 有值 3。

77. NOT(I)

描述:执行逻辑补。

分类:元素函数。

变元:I 必须是整型的。

结果类型和类型参数:与 I 相同。

结果值:结果的值等于把 I 按下列真值表逐位求补而得到的值:

I	NOT(I)
1	0
0	1

其中的整数值按本章(11. I)的模型解释为位的序列。

例子:如果 I 的值为位串 01010101，则 NOT(I) 有值 10101010。

78. PACK(ARRAY,MASK,VECTOR) VECTOR 任选

描述:在 MASK 的控制下，将一个数组 ARRAY 包入一个秩为1的数组。

分类:变换函数。

变元:ARRAY 可是任意类型的，但一定不是标量。

MASK 必须是逻辑型的，并与 ARRAY 有相同的形状。

VECTOR(任选)必须与 ARRAY 具有相同的类型和类型参数，且秩为1。VECTOR 的元素个数一定不能少于 MASK 中真元素的个数。如果 MASK 为标量且值为真，则 VECTOR 的元素个数一定不能少于 ARRAY 中元素的个数。

结果类型、类型参数和形状:结果是一个秩为1且与 ARRAY 具有相同的类型和类型参数的数组。如果 VECTOR 存在，则结果的大小是 VECTOR 的大小；否则其大小是 MASK 中真元素的个数 t，除非 MASK 是标量且值为真，这时结果的大小与 ARRAY 的相同。

结果值:按数组中元素的次序，结果的第i个元素是ARRAY中对应于MASK的第i个真元素的数组元素，i=1,2,…,t。如果VECTOR存在且其大小n大于t，则结果的第i个元素为VECTOR(i)，i=t+1,…,n。

例子:如果M的值是下列数组:

$$\begin{bmatrix} 0 & 0 & 0 \\ 9 & 0 & 0 \\ 0 & 0 & 7 \end{bmatrix}$$

则其非零元素可用PACK来收集。PACK(M,MASK=A.NE.0)的值为[9,7]。PACK(M,MASK=A.NE.0,VECTOR=(/2,4,6,8,10,12/))的值为[9,7,6,8,10,12]。

79. PRECISION(X)

描述:返回表示与变元具有相同种别类型参数的实数的模型的十进精度。

分类:查询函数。

变元:X必须是实型或复型的。它可以是标量或数组。

结果类型、类型参数和形状:默认整型标量。

结果值:结果的值为INT((P-1)*LOG10(b))+k，其中b和p就是在本章(11.Ⅱ,Ⅲ)定义的表示与X具有相同种别类型参数和相同值的实数的模型中的那些参数，而k为1(当b是10的整数次幂时)，否则为0。

例子:对于在本章(11.Ⅴ)的模型所表示的实型数X而言，PRECISION(X)的值为INT(23*LOG10(2.0))=INT(6.92...)=6。

80. PRESENT(A)

描述:查询一个任选的变元是否存在。

分类:查询函数。

变元:A必须是一个在该PRESENT函数引用所在的过程中的任选变元。

结果类型和类型参数:默认逻辑型标量。

结果值：如果 A 是存在的，则结果的值为真，否则为假。

例子：如果实数按本章(11. V)的模型定义，则 MINEXPONENT(X)有值 −126。

81. PRODUCT(ARRAY,DIM,MASK) DIM，MASK 任选

描述：沿着第 DIM 维计算对应 MASK 的真元素的所有 ARRAY 的元素的乘积。

分类：变换函数。

变元：ARRAY 必须是整型、实型或复型的，且一定不是标量。

DIM(任选)必须是值在1≤DIM≤n 之间的整型标量，其中 n 是 ARRAY 的秩，且与 DIM 对应的实元一定不能是一个任选的虚元。

MASK(任选)必须是逻辑型的，并与 ARRAY 有相同的形状。

结果类型、类型参数和形状：结果与 ARRAY 具有相同的类型和种别类型参数。如果 DIM 省缺或 ARRAY 的秩为1，则结果是一个标量；否则结果是秩为 n−1、形状为$(d_1, d_2, \cdots, d_{DIM-1}, d_{DIM+1}, \cdots, d_n)$的数组，其中$(d_1, d_2, \cdots, d_n)$是 ARRAY 的形状。

结果值：情况 1　如果 MASK 省缺，则 PRODUCT(ARRAY)的结果为 ARRAY 的所有元素的乘积的依赖于处理系统的一个近似值。如果 ARRAY 的大小为0，则其值为1。

情况2　如果 MASK 存在，则结果 PRODUCT(ARRAY, MASK=MASK)的值按如下的步骤获得：首先从 ARRAY 中去掉 MASK 中对应位置为假的那些元素；然后在处理后的 ARRAY 中按情况1所述计算处理后的 ARRAY 中的元素的乘积。如果 ARRAY 的大小为0或 MASK 中都是假元素，即处理后的 ARRAY 中已没有元素，则结果的值为1。

情况3　若 ARRAY 的秩为1，则 PRODUCT(ARRAY, DIM [, MASK])的值等于 PRODUCT(ARRAY [, MASK = MASK])；否则 PRODUCT(ARRAY, DIM [, MASK])的元素

$(s_1, s_2, \cdots, s_{DIM-1}, s_{DIM+1}, \cdots, s_n)$的值等于 PRODUCT(ARRAY $(s_1, s_2, \cdots, s_{DIM-1}, :, s_{DIM+1}, \cdots, s_n)$〔,MASK＝MASK $(s_1, s_2, \cdots, s_{DIM-1}, :, s_{DIM+1}, \cdots, s_n)$〕)。

例子:情况1　PRODUCT((/1,2,3/))的值为6。

情况2　PRODUCT(C,MASK＝C. LT. 0. 0)将形成 C 的所有正元素的乘积。

情况3　如果 B 的值是下列数组:

$$\begin{bmatrix} 1 & 3 & 5 \\ 2 & 4 & 6 \end{bmatrix}$$

则 PRODUCT(B,DIM＝1)有值〔2,12,30〕。PRODUCT(B,DIM＝2)有值〔15,48〕。

82. RADIX(X)

描述:返回表示与变元具有相同类型和种别类型参数的数的模型的基数。

分类:查询函数。

变元:X 必须是整型或实型的。它可是标量或数组。

结果类型、类型参数和形状:默认整型标量。

结果值:如果 X 为整型,则结果的值为 r;如果 X 为实型,则结果的值为 b。其中 r 和 b 就是在本章(11. Ⅱ,Ⅲ)定义的表示与 X 具有相同类型和种别类型参数的数的模型中的那些参数。

例子:对于在本章(11. Ⅴ)的模型所表示的实型数 X 而言,RADIX(X)的值为2。

83. RANDOM_NUMBER(HARVEST)

描述:返回一个在〔0,1)上均匀分布的伪随机数或伪随机数数组。

分类:子程序。

变元:HARVEST 必须是实型的。它是一个 INTENT(OUT)变元,可是一个标量或数组变量。每次返回时,它被置为在〔0,1)区间上均匀分布的随机数。

例子:

```
REAL X, Y(10,10)
  !用一个伪随机数把 X 初始化
CALL RANDOM_NUMBER(HARVEST=X)
CALL RANDOM_NUMBER(Y)
  !X 和 Y 中包含了均匀分布的随机数
```

84. RANDOM_SEED(SIZE, PUT, GET) SIZE, PUT, GET 任选

描述：从新开始或查询 RANDOM_NUMBER 伪随机数产生器。

分类：子程序。

变元：必须或者恰好只有一个变元或者没有变元。

SIZE（任选）必须是默认整型标量。这是一个 INTENT(OUT)变元。它被置成处理系统为了存放种子(SEED)的值所用的整数的个数 N。

PUT（任选）必须是秩为1大小大于 N 的默认整型数组。这是一个 INTENT(IN)自变量。它被处理系统用来置种子(SEED)的值。

GET（任选）必须是秩为1大小大于 N 的默认整型数组。这是一个 INTENT(OUT)自变量。它被处理系统置成种子(SEED)当前的值。

如果没有变元存在，处理系统把种子置成一个依赖处理系统的值。

例子：

```
CALL RANDOM_SEED                     !处理系统初始化
CALL RANDOM_SEED(SIZE=K)             !置 K=N
CALL RANDOM_SEED(PUT=SEED(1:K))      !置用户种子
CALL RANDOM_SEED(GET=OLD(1:K))       !读当前的种子
  !X 和 Y 中包含了均匀分布的随机数
```

85. RANGE(X)

描述：返回表示与变元具有相同种别类型参数的整数或实数的模型中的十进指数范围。

分类:查询函数。

变元:X 必须是整型、实型或复型的。它可以是标量或数组。

结果类型、类型参数和形状:默认整型标量。

结果值:情况1　对一个整型变元而言，结果的值为 INT(LOG10(huge))，其中 huge 就是在本章(11. Ⅲ)定义的表示与 X 具有相同种别类型参数的整数的模型中的最大正整数。

情况2　对一个实型或复型变元而言，结果的值为 INT(MIN(LOG10(huge),－LOG10(tiny)))，其中 huge 和 tiny 就是在本章(11. Ⅱ，Ⅲ)定义的表示与 X 具有相同种别类型参数和相同值的实数的模型中的最大和最小正整数。

例子:对于在本章(11. Ⅴ)的模型所表示的实型数 X 而言，RANGE(X)的值为38,因为在此情形中，$huge=(1-2^{-24})\times 2^{127}$；$tiny=2^{-127}$。

86. REAL(A,KIND) KIND 任选

描述:转换为实型。

分类:元素函数。

变元:A 必须是整型、实型或复型的。

KIND(任选)必须是一个整型的标量初始化表达式。

结果类型和类型参数:结果为实型。

情况1　如果 A 是整型或实型且存在 KIND，则结果的种别类型参数由 KIND 指明；如果 A 是整型或实型但不存在 KIND，则结果的种别类型参数是对应默认实型的一种依赖处理系统的种别类型参数。

情况2　如果 A 是复型且存在 KIND，则结果的种别类型参数由 KIND 指明；如果 A 是复型但不存在 KIND，则结果的种别类型参数就是 A 的种别类型参数。

结果值:情况1　如果 A 是整型或实型，则结果等于 A 的依赖处理系统的近似值。

情况2　如果 A 是复型，则结果等于 A 的实部的依赖处理系统的近似值。

例子：REAL(－3) 有值 3.0。对复型变量 Z 而言，REAL(Z) 与复变量 Z 的实部具有相同的种别类型参数和相同的值。

87. REPEAT(STRING, NCOPIES)

描述：把一个串的几个拷贝拼接在一起。

分类：转换函数。

变元：STRING 必须是字符型的标量。

NCOPIES 必须是整型的标量，且其值必须不是负的。

结果类型、类型参数和形状：结果是长度为 STRING 的 NCOPIES 倍的字符型标量，且与 STRING 具有相同的种别类型参数。

结果值：结果的值是把 NCOPIES 个 STRING 拼接在一起得到的字符串。

例子：REPEAT('H', 2)的值为 HH。REPEAT('XYZ', 0)的值为长度为0的串。

88. RESHAPE(SOURCE,SHAPE,PAD,ORDER) PAD, ORDER 任选

描述：用一个给定数组的元素来构造一个指定形状的数组。

分类：变换函数。

变元：SOURCE 可是任意类型的，但必须是一个数组。如果省缺 PAD 或大小为0，则 SOURCE 的大小必须大于等于 PRODUCT(SHAPE)。结果的大小为 SHAPE 的各元素的值的乘积。

SHAPE 必须是整型的秩为1且大小为常数的数组。其大小必须是正的且小于8，其中一定不能有负值元素。

PAD(任选)必须与 SOURCE 具有相同的类型和类型参数。PAD 必须是一个数组。

ORDER (任选)必须是整型的且与 SHAPE 具有相同的形状，并且它的值必须是(1,2,…,n)的一种排列，其中 n 是 SHAPE 的大小。如果省缺 ORDER，就默认它具有值(1,2,…,n)。

结果类型、类型参数和形状：结果是形状为 SHAPE 的一个数

组，就是说，SHAPE(RESHAPE(SOURCE，SHAPE，PAD，ORDER))的值为 SHAPE。结果与 SOURCE 具有相同的类型和类型参数。

结果值：结果的元素就是将 SOURCE 的元素按数组元素的次序排列后，如必要再拼接上附加的若干个 PAD 包含的元素(也按数组元素的次序排列)中的元素是按 ORDER 所指的下标次序排列的。

例子：RESHAPE((/1,2,3,4,5,6/)，(/2,3/))的值是

$$\begin{bmatrix} 1 & 3 & 5 \\ 2 & 4 & 6 \end{bmatrix}$$

RESHAPE((/1,2,3,4,5,6/))，(/2,4/)，(/0,0/)，(/2,1/))的值为

$$\begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 0 & 0 \end{bmatrix}$$

89. RRSPACING(X)

描述：回送靠近变元值的模型数的相对空格的倒数。

分类：元素函数。

变元：X 必须是实型的。

结果类型和类型参数：与 X 的相同。

结果值：结果的值为 $|X \times b^{-e}| \times b^{p}$，其中 b、e 和 p 就是如同在本章(11. Ⅲ)定义的 X 的模型表示中所包含的 b、e 和 p。

例子：如果实数按本章(11. Ⅴ)的模型定义，则 RRSPACING(−3.0) 有值 0.75×2^{24}。

90. SCALE(X,I)

描述：回送 $X \times b^{I}$，其中 b 为 X 的模型表示中的基数。

分类：元素函数。

变元：X 必须是实型的

I 必须是整型的。

结果类型和类型参数：与 X 的相同。

结果值：结果的值为 $X \times b^{I}$，其中 b 就是如同在本章(11. Ⅲ)

定义的 X 的模型表示中所包含的基数 b，倘若这个结果值没有超出数的表示范围。如果这个结果值已超出表示范围，则结果依赖处理系统。

例子：如果实数按本章(11. V)的模型定义，则 SCALE(3.0, 2) 有值12.0。

91. SCAN(STRING,SET,BACK) BACK 任选

描述：在一个串中扫描一个字符集中的任一字符。

分类：元素函数。

变元：STRING 必须是字符型的。

SET 必须是字符型的，且与 STRING 具有相同种别类型参数。

BACK(任选)必须是逻辑型的。

结果类型和类型参数：默认整型。

结果值：情况1　如果 BACK 省缺或不省缺而有假值，并且在 STRING 中至少包含一个 SET 中的字符，则结果的值为包含在 STRING 中的 SET 的位于最左面一个字符的位置。

情况2　如果 BACK 不省缺而有真值，并且在 STRING 中至少包含一个 SET 中的字符，则结果的值为包含在 STRING 中的 SET 的位于最右面一个字符的位置。

情况3　如果在 STRING 中不包含 SET 中的字符，或者 STRING 或 SET 的大小为0，则结果的值为0。

例子：情况1　SCAN('FORTRAN','TR')的值为3。

情况2　SCAN('FORTRAN','TR',BACK=.TRUE.)的值为5。

情况3　SCAN('FORTRAN','BCD')的值为0。

92. SELECTED_INT_KIND(R)

描述：回送表示所有满足 $-10^R<n<10^R$ 的整数值 n 的整型数据类型的种别类型参数的一个值。

分类：转换函数。

变元：R 必须是整型的标量。

结果类型、类型参数和形状:默认整型标量。

结果值:结果的值为表示所有满足$-10^R<n<10^R$的整数值n的整数数据类型的种别类型参数的一个值。如果在处理系统中没有这样一个种别类型参数,则结果的值为-1。如果有多个满足条件的种别类型参数,则回送其中具有最小十进指数范围的那个值,除非此时还有多个这样的值,这时回送其中最小的那个。

例子:在一个支持具有r=2、q=3的默认整型表示方法的机器上,SELECTED_REAL_KIND(6) 有值 KIND(0)。

93. SELECTED_REAL_KIND(P,R) P,R 任选

描述:回送至少具有P位十进精度和十进指数范围至少为R的实型数据类型的种别类型参数的一个值。

分类:转换函数。

变元:至少要有一个变元。

P(任选)必须是整型的标量。

R(任选)必须是整型的标量。

结果类型、类型参数和形状:默认整型标量。

结果值:结果的值为至少具有P位如同函数PRECISION所回送的那种十进精度和如同函数RANGE所回送的那种十进指数范围至少为R的实型数据类型的种别类型参数的一个值。如果在处理系统中没有这样一个种别类型参数,则结果的值为-1。如果有多个满足条件的种别类型参数,则回送其中具有最小十进精度的那个值,除非此时还有多个这样的值,这时回送其中最小的那个。

例子:在一个支持具有$b=16$、$p=6$、$e_{min}=-64$和$e_{max}=63$的默认实型近似表示方法的机器上,SELECTED_REAL_KIND(6,70) 有值 KIND(0.0)。

94. SET_EXPONENT(X,I)

描述:回送一个其分数部分为X的模型表示的分数部分,其指数部分为I的模型数。

分类:元素函数。

变元:X 必须是实型的。

I 必须是整型的。

结果类型和类型参数:与 X 的相同。

结果值:结果的值为 $X\times b^{I-e}$，其中 b 和 e 就是如同在本章(11. Ⅲ)定义的 X 的模型表示中所包含的那些数，倘若这个结果值没有超出数的表示范围。如果这个结果值已超出表示范围，则结果依赖处理系统。如果 X 值为0，则结果的值也是0。

例子:如果实数按本章(11. Ⅴ)的模型定义，则 SET_EXPONENT(3.0,1)有值1.5。

95. SHAPE(SOURCE)

描述:回送一个数组或标量的形状。

分类:查询函数。

变元:SOURCE 可是任何类型的。它可是标量或数组，但它一定不能是未被结合的指针或者没被分配的可分配数组，并且不是一个僭取大小数组。

结果类型、类型参数和形状:默认整型标量。

结果值:结果是一个秩为1大小等于 SOURCE 的秩的默认整型数组。

例子:SHAPE(A(2:5,-1:1)) 有值[4,3]。SHAPE(3)的值为大小为0秩为1的数组。

96. SIGN(A,B)

描述:A 的绝对值乘 B 的符号。

分类:元素函数。

变元:A 必须是整型或实型的。

B 必须与 A 具有相同的类型和种别类型参数。

结果类型和类型参数:与 A 的相同。

结果值:如果 B≥0，则结果是|A|；如果 B<0，则结果是-|A|。

例子:SIGN(-3.0,2.0)有值3.0。

97. SIN(X)

描述：正弦函数。

分类：元素函数。

变元：X 必须是实型或复型的。

结果类型和类型参数：与 X 的相同。

结果值：结果为依赖处理系统的 SIN(X)的近似值。如果 X 为实型，则它被认为是一个弧度值。如果 X 为复型，则它的实部被认为是一个弧度值。

例子：SIN(1.0) 近似地有值 0.84147098。

98. SINH(X)

描述：双曲正弦函数。

分类：元素函数。

变元：X 必须是实型的。

结果类型和类型参数：与 X 的相同。

结果值：结果为依赖处理系统的 SINH(X)的近似值。

例子：SINH(1.0) 近似地有值 1.1752012。

99. SIZE(ARRAY,DIM) DIM 任选

描述：回送一个数组第 DIM 维的大小或者整个数组的全部元素的个数。

分类：查询函数。

变元：ARRAY 可是任何类型的，但一定不是标量。它一定不能是未被结合的指针或者没被分配的可分配数组，如果 ARRAY 是一个假定大小数组，则 DIM 必须存在，且值小于 ARRAY 的秩。

DIM(任选)必须是值在1≤DIM≤n 之间的整型标量，其中 n 是 ARRAY 的秩。

结果类型、类型参数和形状：默认整型标量。

结果值：结果是数组 ARRAY 的第 DIM 维的大小或者当 DIM 省缺时，是整个数组的全部元素的个数。

例子：SIZE(A(2:5，－1:1)，DIM＝2) 的值是3；SIZE(A(2:5，－1:1)) 的值是12。

100. SPACING(X)

描述:回送靠近变元值的模型数的绝对空格的数。

分类:元素函数。

变元:X 必须是实型的。

结果类型和类型参数:与 X 的相同。

结果值:结果的值为 $X \times b^{e-p}$,其中 b、e 和 p 就是如同在本章(11. Ⅲ)定义的 X 的模型表示中所包含的 b、e 和 p,倘若这个结果值没有超出数的表示范围。如果这个结果值已超出表示范围,则结果等于 TINY(X)。

例子:如果实数按本章(11. Ⅴ)的模型定义,则 SPACING(3.0) 有值 2^{-22}。

101. SPREAD(SOURCE, DIM, NCOPIES)

描述:以加一个维的方式复制一个数组多次形成一个高一维的数组。

分类:转换函数。

变元:SOURCE 可是任意类型的。它可是标量或数组,但其秩必须小于7。

DIM 必须是值在1≤DIM≤n+1之间的整型标量,其中 n 是 SOURCE 的秩。

NCOPIES 必须是整型的标量。

结果类型、类型参数和形状:结果是一个与 SOURCE 具有相同类型和类型参数的秩为 n+1的数组,其中 n 为 SOURCE 的秩。

情况1 如果 SOURCE 是标量,则结果的形状是(MAX(NCOPIES,0))。

情况2 如果 SOURCE 是形状为(d_1,d_2,…,dn)的数组,则结果的形状是(d_1,d_2,…,d_{DIM-1}, MAX(NCOPIES,0),d_{DIM},…,d_n)。

结果值:情况1 如果 SOURCE 是标量,则结果的各元素的值都是 SOURCE 的值。

情况2　如果 SOURCE 是数组，则结果中具有下标$(r_1,r_2,\cdots,r_{n+1})$的元素的值是 SOURCE$(r_1,r_2,\cdots,r_{DIM-1},r_{DIM+1},\cdots,r_{n+1})$的值。

例子：如果 A 是数组〔2，3，4〕，则当 NC 的值为3时，SPREAD(A,DIM=1,NCOPIES=NC)的值为数组：

$$\begin{bmatrix}2 & 3 & 4\\2 & 3 & 4\\2 & 3 & 4\end{bmatrix}$$

如果 NC 的值为0，则结果为大小为0的一个数组。

102. SQRT(X)

描述：平方根函数。

分类：元素函数。

变元：X 必须是实型或复型的。除非 X 为复型，它的值必须大于等于0。

结果类型和类型参数：与 X 的相同。

结果值：结果为依赖处理系统的 X 的平方根的近似值。复型结果的值是实部大于等于0的主值。当结果的实部为0时，其虚部大于等于0。

例子：SQRT(4.0) 近似地有值 2.0。

103. SUM(ARRAY,DIM,MASK) DIM，MASK 任选

描述：沿着第 DIM 维计算对应于 MASK 的真元素的所有 ARRAY 的元素的和。

分类：变换函数。

变元：ARRAY 必须是整型、实型或复型的，且一定不是标量。

DIM(任选)必须是值在1≤DIM≤n 之间的整型标量，其中 n 是 ARRAY 的秩，且与 DIM 对应的实元一定不能是一个任选的虚元。

MASK(任选)必须是逻辑型的，并与 ARRAY 有相同的形状。

结果类型、类型参数和形状．结果与 ARRAY 具有相同的类型和种别类型参数。如果 DIM 省缺或 ARRAY 的秩为1，则结果是一个标量；否则结果是秩为 $n-1$，形状为 $(d_1, d_2, \cdots, d_{DIM-1}, d_{DIM+1}, \cdots, d_n)$的数组，其中$(d_1, d_2, \cdots, d_n)$是 ARRAY 的形状。

结果值：情况1　如果 MASK 省缺，则 SUM(ARRAY)的结果为 ARRAY 的所有元素的和数的依赖于处理系统的一个近似值。如果 ARRAY 的大小为0，则其值为0。

情况2　如果 MASK 存在，则结果 SUM(ARRAY,MASK=MASK)的值按如下的步骤获得：首先从 ARRAY 中去掉 MASK 中对应位置为假的那些元素；然后在处理后的 ARRAY 中按情况1．所述计算处理后的 ARRAY 中的元素的和数。如果 ARRAY 的大小为0或 MASK 中都是假元素，即处理后的 ARRAY 中已没有元素，则结果的值为0。

情况3　若 ARRAY 的秩为1，则 SUM(ARRAY, DIM [, MASK])的值等于 SUM(ARRAY [,MASK=MASK])；否则 SUM(ARRAY, DIM [,MASK])的元素$(s_1, s_2, \cdots, s_{DIM-1}, s_{DIM+1}, \cdots, s_n)$的值等于 SUM(ARRAY$(s_1, s_2, \cdots, s_{DIM-1}, :, s_{DIM+1}, \cdots, s_n)$,[,MASK=MASK$(s_1, s_2, \cdots, s_{DIM-1}, :, s_{DIM+1}, \cdots, s_n)$])。

例子：情况1　SUM((/1,2,3/))的值为6。

情况2　SUM(C,MASK=C.LT.0.0)将形成 C 的所有正元素的算术和。

情况3　如果 B 的值是下列数组：

$$\begin{bmatrix} 1 & 3 & 5 \\ 2 & 4 & 6 \end{bmatrix}$$

则 SUM(B,DIM=1)有值[3,7,11]。SUM(B,DIM=2)有值[9,12]。

104. SYSTEM_CLOCK(COUNT,COUNT_RATE,COUNT_MAX) COUNT,COUNT_RATE,COUNT_MAX 任选

描述：根据实时时钟返回一些整型数据。

分类:子程序。

变元:COUNT(任选)必须是默认整型的标量。它是一个 INTENT(OUT)变元。根据处理系统时钟的当前值被置成一个依赖处理系统的值;当没有时钟时置成-HUGE(0)。这个依赖处理系统的值对时钟的每个计数加1,直到 COUNT_MAX 后在时钟的下一个计数时被重置成0。所以,如果有一个时钟,则其值在0到 COUNT_MAX 之间。

COUNT_RATE(任选)必须是默认整型的标量。它是一个 INTENT(OUT)变元。它被置成每一秒处理系统时钟的计数数目;当没有时钟时置成0。

COUNT_MAX(任选)必须是默认整型的标量。它是一个 INTENT(OUT)变元。它被置成 COUNT 可以具有的最大值;当没有时钟时置成0。

例子:如果时钟是以24小时计时一秒一计数的,则在上午11时30分时调用:

```
CALL SYSTEM_CLOCK(COUNT=C, COUNT_RATE
      =R, COUNT_MAX=M)
```

将把 C、R 和 M 置成:

$C=11\times3600+30\times60=41400$,

$R=1$,

$M=24\times3600-1=86399$。

105. TAN(X)

描述:正切函数。

分类:元素函数。

变元:X 必须是实型的。

结果类型和类型参数:与 X 的相同。

结果值:结果为依赖处理系统的 tan(X)的近似值,其中 X 被认为是一个弧度值。

例子:TAN(1.0) 近似地有值 1.5574077。

106. TANH(X)

描述：双曲正切函数。

分类：元素函数。

变元：X 必须是实型的。

结果类型和类型参数：与 X 的相同。

结果值：结果为依赖处理系统的 tanh(X)的近似值。

例子：TANH(1.0) 近似地有值 0.76159416。

107. TINY(X)

描述：返回一个与变元具有相同类型和种别类型参数的模型表示数中的最小正数。

分类：查询函数。

变元：X 必须是实型的。它可是标量或数组。

结果类型、类型参数和形状：与变元 X 具有相同类型和种别类型参数的标量。

结果值：结果的值为($1-b^{e_{min}-1}$，其中 b 和 e_{min} 就是在本章(11.Ⅲ)中定义的为了表示与 X 具有相同类型和种别类型参数的数的模型中的那些参数。

例子：若 X 是在本章(11.Ⅴ)定义的模型所表示的实型数，则 TINY(X)的值为2^{-127}。

108. TRANSFER(SOURCE, MOLD, SIZE) SIZE 任选

描述：回送一个结果，其物理表示完全与 SOURCE 相同，但是用 MOLD 的类型和类型参数来解释。

分类：转换函数。

变元：SOURCE 可是任意类型的。它可是标量或数组。

MOLD 可是任意类型的。它可是标量或数组。

SIZE(任选)必须是整型标量，且与其对应的实元一定不能是一个任选的虚元。

结果类型、类型参数和形状：结果是一个与 MOLD 具有相同类型和类型参数：

情况1　如果 SIZE 省缺且 MOLD 是标量，则结果是标量。

情况2　如果 SIZE 省缺且 MOLD 是数组，则结果是秩为1的

数组。它的大小要尽可能小使得其物理表示不比 SOURCE 的大小短。

情况3　如果 SIZE 存在，则结果是秩为1大小为 SIZE 的数组。

结果值：如果结果的物理表示与 SOURCE 具有相同的长度，则结果的物理表示与 SOURCE 的相同。如果结果的物理表示的长度比 SOURCE 的长，则结果的物理表示的前头部分与 SOURCE 的相同，余下部分是无定义的。如果结果的物理表示的长度比 SOURCE 的短，则结果的物理表示就是 SOURCE 的前头部分。如果 D 和 E 是标量变量，D 的物理表示的长度大于等于 E 的物理表示的长度，则 TRANSFER(TRANSFER(E,D),E)必须是 E 的值。如果 D 是一个数组，E 是一个秩为1的数组，则 TRANSFER(TRANSFER(E,D),E,SIZE(E))必须是 E 的值。

例子：情况1　在把值4.0和1082130432表示为二进位串0100 0000 1000 0000 0000 0000 0000 0000的处理系统上，TRANSFER (1082130432,0.0)的值为4.0。

情况2　TRANSFER((/1.1,2.2,3.3/),(/(0.0,0.0)/))是秩为1长度为2的复型数组，其第一个元素是(1.1,2.2)，而第二个元素的实部为3.3，虚部无定义。

情况3　TRANSFER((/1.1,2.2,3.3/),(/(0.0,0.0)/),1)的值为[1.1+2.2i]。

109. TRANSPOSE(MATRIX)

描述：秩为2的数组的转置。

分类：变换函数。

变元：MATRIX 可是任何类型的，但必须是秩为2的数组。

结果类型、类型参数和形状：结果是一个秩为2而且与 MATRIX 具有相同的类型、类型参数和形状的数组。

结果值：结果中下标为(i,j)的元素就是 MATRIX 中具有下标(j,i)的元素，i=1,2,…,n；j=1,2,…,m，其中(m,n)是 MATRIX 的形状。

例子：如果 A 的值是下列数组：

$$\begin{bmatrix}1 & 2 & 3\\4 & 5 & 6\\7 & 8 & 9\end{bmatrix}$$

则 TRANSPOSE(A)有值：

$$\begin{bmatrix}1 & 4 & 7\\2 & 5 & 8\\3 & 6 & 9\end{bmatrix}$$

110. TRIM(STRING)

描述：返回去掉了尾部空格字符后的变元值。

分类：转换函数。

变元：STRING 必须是字符型的标量。

结果类型和类型参数：字符类型，它与 STRING 具有相同的种别类型参数值，长度是 STRING 的长度减去其尾部空格字符数。

结果值：结果的值为把 STRING 的所有尾部空格去掉之后余下的字符组成的字符串。如果变元不包含非空格字符，则结果值的长度为0。

例子：TRIM('AB ')的值为'AB'。

111. UBOUND(ARRAY,DIM) DIM 任选

描述：返回一个数组的所有上界或某指定的上界。

分类：查询函数。

变元：ARRAY 可是任意类型的，但一定不是标量，也不能是一个未结合的指针或者没被分配的可分配数组。

DIM(任选)必须是值在1≤DIM≤n 之间的整型标量，其中 n 是 ARRAY 的秩，且与 DIM 对应的实元一定不能是一个任选的虚元。如果 ARRAY 是一个僭取大小数组，DIM 必须存在，且是值小于 ARRAY 的秩。

结果类型、类型参数和形状：结果是默认整型的。如果 DIM 存在，则结果是标量；否则结果是一个秩为1而大小为 n 的数组，其

中 n 是 ARRAY 的秩。

结果值:情况1　对一个不同于整个数组或一个数组结构分量的数组段或数组表达式，UBOUND(ARRAY,DIM)的值等于在给定维中的元素的个数；否则值为 ARRAY 的第 DIM 个下标的上界(当 ARRAY 的第 DIM 维的大小不是0时)，值为0(当 ARRAY 的第 DIM 维的大小是0时)。

情况2　UBOUND(ARRAY)的值是一个数组，其第 i 个分量为 UBOUND(ARRAY,i),i=1,2,...,n，其中 n 为 ARRAY 的秩。

例子:如果 A 是由下列语句说明的数组：

REAL A (2:3, 7:10)

则 UBOUND(A)是[3,10]，而 UBOUND(A, DIM=2)是10。

112. UNPACK(VECTOR,MASK,FIELD)

描述:在 MASK 的控制下，将一个秩为1的数组拆包为一个数组。

分类:变换函数。

变元:VECTOR 可是任意类型的，但必须有秩为1且大小至少为 t，其中 t 为 MASK 中真元素的个数。

MASK 必须是逻辑型的数组。

FIELD 必须与 VECTOR 具有相同的类型和类型参数，且与 MASK 有相同的形状。

结果类型、类型参数和形状:结果是一个与 VECTOR 具有相同的类型和类型参数与 MASK 有相同形状的数组。

结果值:按数组中元素的次序，结果中对应于 MASK 的第 i 个真元素处的数组元素的值为 VECTOR(i)，i=1,2,…,t，其中 t 为 MASK 中真元素的个数。其它位置的数组元素为 FIELD(如果 FIELD 为标量)；或者为 FIELD 中对应元素的值(如果 FIELD 为数组)。

例子:应用 UNPACK 可将一些指定的值散布到一个数组的指定位置去。如果 M 的值是下列数组：

$$\begin{bmatrix}1 & 0 & 0\\0 & 1 & 0\\0 & 0 & 1\end{bmatrix}$$

V 是数组〔1,2,3〕，而 Q 是逻辑矩阵：

$$\begin{bmatrix}. & T & .\\T & . & .\\. & . & T\end{bmatrix}$$

其中，T 表示真，"."表示假，则 UNPACK(V,MASK＝Q,FIELD＝M)的值为：

$$\begin{bmatrix}1 & 2 & 0\\1 & 1 & 0\\0 & 0 & 3\end{bmatrix}$$

而 UNPACK(V,MASK＝Q,FIELD＝0)的值为

$$\begin{bmatrix}0 & 2 & 0\\1 & 0 & 0\\0 & 0 & 3\end{bmatrix}$$

113. VERIFY(STRING,SET,BACK) BACK 任选

描述：验证一个字符集是否包含一个串中的所有字符，标出该字符串中第一个不出现在该给定字符集中的字符的位置。

分类：元素函数。

变元：STRING 必须是字符型的。

SET 必须是字符型的，且与 STRING 有相同的种别类型参数。

BACK(任选)必须是逻辑型的。

结果类型：默认整型。

结果值：情况1　如果 BACK 省缺存在但其值为假，并且如果 STRING 至少包含一个不在 SET 中的字符，则结果的值是最左一个不在 SET 中的 STRING 的字符的位置。

情况2　如果 BACK 存在且其值为真，并且如果 STRING 至少包含一个不在 SET 中的字符，则结果的值是最右一个不在

SET 中的 STRING 的字符的位置。

情况3　如果 STRING 中的每个字符都包含在 SET 中，或者 STRING 的长度为0，则结果的值是0。

例子：情况1　VERIFY('ABBA','A')有值2。

情况2　VERIFY('ABBA','A',BACK=.TRUE.)有值3。

情况3　VERIFY('ABBA','AB')有值0。

第十二章　作用域、结合和定义

本章将给出Fortran 90的一些很重要的概念，如作用域的含意及使用方法、如何使变量变成定义的或无定义的等问题。

第一节　作　用　域

长期以来，使用Fortran语言书写源程序时，我们都已熟知Fortran语言提供了一个可以分离编译各个过程与主程序的便利条件。当某个过程需要做改动时，我们只需对该过程做单独的再调试与编译即可。这个功能给大型应用程序的编制提供了极大的方便。但应该说同时也带来了某些不便，例如对一个变量名的说明必须在它出现的各个部分都要进行说明，如在某一过程中忘记说明时，就可能出现无定义或发生其它不可预见的情况，这就是所谓名字的作用域问题。

若一个名字在主程序中说明了它的类型及含意，它代表的是一个变量、或过程等，那么对它有关的说明就在从PROGRAM语句开始到END PROGRAM语句这个范围内有意义，这就是它的作用域。类似地，如果在SUBROUTINE或FUNCTION语句之后对它进行了说明，那么它的作用域就是从SUBROUTINE或FUNCTION语句开始直到END SUBROUTINE或END FUNCTION语句为止。而在其它范围内，即使你书写了同样一个名字，它也可以有其它的含意，不会受到其它程序单元中有关说明的任何影响。

一个程序单元由一组非覆盖的作用域单位组成。作用域单位有下列几种：

(1)导出类型定义。

(2)过程接口体，此处应把其包括的任何导出类型定义及过程接口体除外。

(3)一个程序单元或辅程序，此处应把其包括的导出类型定义、过程接口体以及内部辅程序除外。

包含另一个作用域单位的作用域单位称为被包含单位的宿主作用域单位。

如果作用域是一个可执行程序，则其中的实体称为“全局实体”；若作用域是一个作用域单位，则其中的实体称为“局部实体”；若作用域是一个语句或语句的一部分，则称其中的实体为语句实体。

一个实体可用一个名字、一个标号、一个外部输入/输出部件号、一个运算符符号、一个赋值符号等方式标识。通过结合，对一个实体的引用可以通过不同作用域单位中的相同的标识符或不同的标识符、或相同作用域单位中的不同的标识符来进行。

以下的例子是包含五个作用域单位的情况：

```
MODULE SCOPE1                    !作用域1
    …                            !作用域1
CONTAINS                         !作用域1
  SUBROUTINE SCOPE2              !作用域2
    TYPE SCOPE3                  !作用域3
      …                          !作用域3
    END TYPE _SCOPE3             !作用域3
    INTERFACE                    !作用域3
      …                          !作用域4
    END INTERFACE                !作用域3
      …                          !作用域2
CONTAINS                         !作用域2
    FUNCTION SCOPE5(…)           !作用域5
      …                          !作用域5
    END FUNCTION                 !作用域5
  END SUBROUTINE                 !作用域2
```


13

```
END MODULE                              !作用域1
```

以上例子比较清晰地说明了作用域的含意。一旦在一个作用域单位中说明了一个实体，就可以用它的名字在其它作用域单位中引用它。而在另一个作用域单位中说明的实体总是一个不同的实体，甚至它有相同的名字并且有精确地相同的性质也还是一个不同的实体。每一个都看作是一个局部实体。这对导出类型也是一样的，即两个导出类型若有相同的名字及相同的成分，而由它们说明的实体也还处理成不同的类型。这种做法对程序设计者是有很大好处的，他不需要去考虑偶然发生的名字方面的冲突问题了。

第二节　结　　合

结合可以是名结合、指针结合或存储结合。名结合可以是变元结合、宿主结合或使用结合。

存储结合引起不同的实体使用相同的存储。任何结合可使一个实体用同一作用域单位中不同的名或用不同作用域单位中相同的名或不同的名来标识。

一、名　结　合

名结合有三种形式：变元结合、宿主结合及使用结合，它提供了在一个作用域单位中所知的实体在另一作用域单位中可以进行访问的机制。

变元结合是在过程引用的执行中建立的实元及其相应虚元之间的一种结合，它们可以是顺序结合。虚元名和实元名可以相同也可以不相同。结合之后在被引用的过程中即可使用，而当过程执行完毕之后上述建立的这种结合也就终止了。每一个实元必须和一个虚元相结合，例如：

```
SUBROUTINE SOLVE(FUNCT, SOLUTION, METHOD,&
                 STRATEGY, PRINT)
  INTERFACE
```

```
    FUNCTION FUNCT(X)
      REAL FUNCT,X
    END FUNCTION FUNCT
  END INTERFACE
  REAL SOLUTION
  INTEGER, OPTIONAL::METHOD, STRATEGY, PRINT
  ...
```

是一个过程。引用它可以用：

```
CALL SOLVE (FUN, SOL, PRINT=6)
```

进行。

使用结合是由 USE 语句指明的不同的作用域单位中名字间的结合。USE 语句的形式是：

```
USE module-name
```

这可被看作是在其所在的作用域单位内对其中全部模块实体的一个重新说明，它们具有相同的名字和相同的性质。这时，把这些模块实体称作是根据使用结合而能访问的。

在模块过程中可以出现一种称为可访问性的语句，这种语句只能出现在模块过程中，它说明实体的可访问性，共用可访问性或私用可访问性。一个具有共用类属标识符的过程，即使它说明的名字是私用的，也可以通过类属标识符进行访问。

如果在模块的作用域单位中出现可访问性语句 PUBLIC 或 PRIVATE，则称对该模块实体设置了缺省的可访问性，则这种可访问性适用于该模块的作用域单位中那些未专门为自己声明某种可访问性实体。

例如，语句：

```
PUBLIC
```

把缺省情况置为具有共用可访问性。

例如，语句：

```
PRIVATE
```

把缺省情况置为具有私用可访问性。

若在模块中没有这种语句出现，则就默认为具有共用可访问

性。

可访问性语句的例子：

```
MODULE EX
  PRIVATE
  PUBLIC::A,B,C,ASSIGNMENT(=),OPERATOR(+)
```

可访问性也可做为类型说明语句中的一种属性形式出现，当然这种属性只能出现在模块的作用域单位中。

可访问性属性说明了实体的可访问性和导出类型定义的可访问性。用 PRIVATE 属性说明的实体不能在模块外被访问。而用 PUBLIC 属性说明的实体可通过 USE 语句在其它程序单元中被访问。缺省意味着具有 PUBLIC 性质的可访问性。例如：

```
REAL, PRIVATE::X,Y,Z
```

就是一个说明 X,Y 和 Z 具有私用可访问性属性的例子。

一个内部辅程序、模块辅程序或导出类型定义通过宿主结合对其宿主中的有名实体进行访问。这些被访问的实体具有与宿主实体同样的名字和同样的属性，它们可以是变量、常量、包含接口的过程、导出类型、类型参数、导出类型成份及名表组。

如果一个通过使用结合进行访问的实体具有与宿主实体相同的非类属名，宿主实体就是不可访问的。

宿主过程和接口过程可以包含相同的和不相同的使用结合的实体，由下例可以阐明：

```
MODULE B; REAL BX, Q; INTEGER IX, JX; END MODULE
MODULE C; REAL CX; END MODULE
MODULE D; REAL DX, DY, DZ; END MODULE
MODULE E; REAL EX, EY, EZ; END MODULE
MODULE F; REAL FX; END MODULE
MODULE G; USE F; REAL GX; END MODULE
PROGRAM A
USE B; USE C; USE D
...
CONTAINS
```

```
   SUBROUTINE INNER_PROC(Q)
   USE C            !Not needed
   USE B, ONLY:BX   !Entities accessible are BX, IX, and JX
                    !if no other IX or JX
                    !is accessible to INNER_PROC
                    !Q is local to INNER_PROC
                    !since Q is a dummy argument
   USE D, X=>DX     !Entities accessible are DX, DY, and DZ
                    !X is local name for DX in INNER_PROC
                    !X and DX denote same entity if no other
                    !entity DX is local to INNER_PROC
   USE E, ONLY:EX   !EX is accessible in INNER_PROC, not in
                    !program A, EY and EZ are not accessible
                    !in INNER_PROC or in program A
   USE G            !FX and GX are accessible in INNER_PROC
     ...
   END SUBROUTINE INNER_PROC
 END PROGRAM A
```

注意:因为程序 A 包含了语句:

```
USE B
```

除了 Q 以外,模块 B 中的全部实体在 INNER_PROC 中都是可访问的,即使 INNER_PROC 包含了下列语句:

```
USE B, ONLY:BX
```

也是如此。带有 ONLY 关键词的 USE 语句意味着这个特殊语句只引入 ONLY 后的变量。

以下例子说明在子程序 INNER 中,根据宿主结合 X 是可访问的,但 Y 是 INNER 中的一个局部变量,因而宿主中的 Y 是不可访问的:

```
SUBROUTINE OUTER
  REAL X, Y
  ...
CONTAINS
```

```
  SUBROUTINE INNER
    REAL Y
    Y=X+1
    ...
  END SUBROUTINE INNER
END SUBROUTINE OUTER
```

注意，宿主并不对它所包含的任何子程序的局部实体都是可访问的。

二、指 针 结 合

指针结合是指针和目标之间的结合，它允许通过引用指针来引用目标。在程序执行的不同的时刻里，指针可以是无定义的、同各种目标结合的或分离的。指针的初始结合状态是无定义的。若指针和目标相结合，指针的定义状态或是定义的或是无定义的，这取决于目标的定义状态。

一个指针的结合状态可为下列情况之一：

(1)结合的。当做为引用该指针的 ALLOCATE 语句的成功执行的结果，该指针被分配时，或者当指针由指针赋值语句赋给所结合的目标时，或用具有 TARGET 属性来指定的目标如果是可分配的，则就被当时分配时，一个指针变为结合的。

(2)分离的。当指针是空的，或者当指针是去分配的，或者当指针由指针赋值语句赋予分离的指针时，一个指针变为分离的。

(3)无定义的。当指针从未被结合或分离时，或者当其目标从未被分配时，或者当其目标不是通过指针而被去分配时，或者当由于 RETURN 或 END 语句的执行而引起指针的目标变为无定义时，一个指针的结合状态是无定义。

指针的定义状态就是它的目标的定义状态。若一个指针和一个能定义的目标相结合，则指针的定义状态可以是定义的或无定义的，这要根据变量的规则来决定。

如果指针的结合状态是分离的或无定义的，则指针就不得被

引用或去分配。不论其结合状态是什么，一个指针总可以是空的、分配的或指针赋值的。空指针是分离的。当指针是分配的，它就变成结合的但无定义的。当指针是被指针赋值的，它的结合和定义状态由它的目标来确定。

三、存 储 结 合

存储序列用于描述变量、公用块结果变量之间存在的关系。当两个或多个数据对象出现在两个或多个存储序列中，这些序列共享或对齐一个或多个存储单元时出现的结合称存储结合。

存储序列是存储单元的序列。存储序列的大小是该存储序列中存储单元的个数。存储单元有字符存储单元、数值存储单元或未说明的存储单元。

我们说两个非零大小的存储序列S1和S2是存储结合的，如果S1的第 i 个存储单元和 S2的第 j 个存储单元是相同的就叫它们是存储结合的。这就使得 S1的第(i+k)个存储单元和 S2的第(j+k)个存储单元是相同的，对每一个整数 k 都如此，k 满足：$1\leqslant i+k\leqslant$ S1的大小，并且$1\leqslant j+k\leqslant$S2的大小。

而两个标量数据对象是存储结合的，若它们的存储序列是存储结合的。两个标量实体是整体结合的，若它们有相同的存储序列。两个标量实体是部分结合的，若它们是结合的而又不是整体结合的。

数据对象的定义状态的值影响着任何与它存储结合实体的定义状态和值。EQUIVALENCE 语句、COMMON 语句或 ENTRY 语句都可以引起存储序列的存储结合。

EQUIVALENCE 语句引起的数据对象的存储结合只能在一个作用域单位内，除非等价实体之一也在一个公用块内才例外。

COMMON 语句引起一个作用域单位中的数据对象和另一个作用域单位中的数据对象变成存储结合的。

有名公用块允许包含不同存储单元的序列，只要它提供的每个访问该公用块的作用域单位都说明存储单元的同一个序列。对

空白公用块也使用相同的规则。若两个空白公用块的大小不同，短的块的存储单元序列必须和长的块的存储单元的初始序列相同。

函数辅程序中的 ENTRY 语句引起结果变量的存储结合。

部分结合只可能存在于：

(1)默认字符或字符顺序型的对象和默认字符或字符顺序型的对象之间。

(2)默认复型、双精度实型或数值顺序型的对象和默认整型、默认实型、默认逻辑型、双精度实型、默认复型或数值顺序型的对象之间。

对于非字符实体，部分结合只能通过使用 COMMON、EQUIVALENCE 或 ENTRY 语句实现。对于字符实体，部分结合只能通过变元结合或使用 COMMON、EQUIVALENCE 或 ENTRY 语句实现。例如：

```
REAL A(4),B
COMPLEX C(2)
DOUBLE PRECISION D
EQUIVALENCE(C(2),A(2),B),(A,D)
```

C 的第三个存储单元、A 的第二个存储单元、B 的存储单元和 D 的第二个存储单元被指定为同一个存储单元。此存储序列可用表12－1说明。

表12－1

存储单元	1	2	3	4	5
	C(1)		C(2)		
		A(1)	A(2)	A(3)	A(4)
			B		
		D			

表中，A(2)和 B 是整体结合，其余都是部分结合：即 A(1)和 C(1)、A(2)和 C(2)、A(3)和 C(2)、B 和 C(2)、A(1)和 D、A(2)和 D、B 和 D、C(1)和 D 以及 C(2)和 D 之间都是部分结合的。注意，

虽然C(1)和C(2)各自都和D结合，但C(1)和C(2)不是彼此结合的。

当某些字符实体的部分存储单元相同但不是全部相同时，就发生字符实体的部分结合。

下面的例子说明A、B和C是部分结合的：

```
CHARACTER A*4,B*4,C*3
EQUIVALENCE(A(2:3),B,C)
```

结合问题是程序设计者在编制程序中必然要遇到的问题，必须考虑大批的数据如何最有效的使用，而且最节省存储单元等问题。为此，我们必须对结合的定义和规则十分清楚。

第三节 定义和无定义

变量可以是定义的也可以是无定义的，在可执行程序的执行期间其定义状态可以改变。导致一个变量变为无定义的动作并不意味着该变量以前一定是定义的。导致一个变量变为定义的动作并不意味着该变量以前一定是无定义的。

数组、数组片段以及导出类型、字符型或复型变量都是对象，它们由零个或多个对象组成。在变量和子对象之间以及不同变量的子对象之间可以建立结合。从而使这些子对象可以变为定义的或无定义的。Fortran中规定：

(1)当且仅当它的全部子对象都是定义的，对象才是定义的。

(2)若对象是无定义的，则其子对象中至少有一个(不必是全部)是无定义的。

(3)零大小的数组和零长度的串总是定义的。

下列变量称为是初始定义的：

(1)由DATA语句指定具有初始值的变量。

(2)由类型声明语句指定具有初始值的变量。

(3)总是定义的变量。

所有其它的变量都是初始无定义的。

一、导致变量变为定义的事件

导致变量变为定义的事件可以是：

(1)除了屏蔽数组赋值语句外，执行一个内在赋值语句导致赋值号前面的变量变为定义的。执行一个定义的赋值语句可以导致赋值号前面的全部或部分变量变成定义的。

(2)执行屏蔽数组赋值语句可以导致赋值语句中某些或全部数组元素变成定义的。

(3)作为输入语句执行的结果，从输入文件上赋值给它的每个变量在把数据传送给它时变为定义的。执行一个由部件说明符标识的内部文件的 WRITE 语句时，导致所写的每个记录都变成定义的。

(4)执行 DO 语句导致 DO 变量(若有的话)变为定义的。

(5)输入/输出语句中由隐 DO 表规定的动作在开始执行时导致隐 DO 变量变成定义的。

(6)ASSIGN 语句的执行导致语句中的变量变成定义的并带有语句标号值。

(7)如果整个实元是定义的且其值不是语句标号，则引用一个过程导致整个相应虚元数据对象变成定义的。

如果实元的相应子对象是定义的，则引用一个过程导致相应虚元的子对象变成定义的。

(8)包含输入/输出 IOSTAT＝说明符的输入/输出语句的执行导致说明的整型变量变成定义的。

(9)包含 SIZE＝说明符的 READ 语句的执行导致说明的整型变量变成定义的。

(10)执行 INQUIRE 语句，若没有差错条件存在，导致在语句执行期间被赋予值的任何变量变成定义的。

(11)当字符存储单元变成定义的，则所有与它结合的字符存储单元变成定义的。

当一个数值存储单元变成定义的，则所有与它结合的相同类

型的数值存储单元就变成定义的，除了执行 ASSIGN 语句时和在 ASSIGN 语句中的变量相结合的变量变为无定义的之外。当双精度实型的实体变为定义的，所有与该双精度实型的整体结合的实体都变为定义的。

当未说明的存储单元变为定义的，所有与它结合的未说明的存储单元变为定义的。

(12)当默认复型实体变为定义的，所有与它部分结合的默认实型实体变为定义的。

(13)当默认复型实体的两部分变为定义的，这是作为与它们变为定义的默认实型或默认复型实体部分结合的结果，则默认复型实体变为定义的。

(14)当数值顺序结构或字符顺序结构的全部成分变成定义的，作为与它们变为定义的部分结合对象的结果，则该结构变成定义的。

(15)带有 STAT＝说明符的 ALLOCATE 或 DEALLOCATE 语句的执行导致由 STAT＝说明符说明的变量变成定义的。

(16)零大小数组的分配导致该数组变成定义的。

(17)引用一个过程导致该过程中零大小的任何自动对象变成定义的。

(18)执行一个与有定义的目标相结合的指针的指针赋值语句，导致该指针变成定义的。

以上列出了导致变量变成定义的所有可能的18种事件。

二、导致变量变成无定义的事件

导致变量变成无定义的事件可以是：

(1)当给定类型的变量变成定义的，则所有与它结合的不同类型的变量变成无定义的。然而，当默认实型的变量和默认复型的变量部分结合时，实变量变成定义时相应的复变量并不变为无定义的，且当复变量变成定义时，与它结合的实变量不变为无定义的。

当默认复型的变量和另一默认复型的变量部分结合时，其中一个的定义并不导致另一个变为无定义的。

(2)ASSIGN 语句的执行导致语句中的变量变成作为整型量是无定义的。与该变量结合的各个变量也都变为无定义的。

(3)若函数的求值可能导致函数的变元或模块中的变量或公用块中的变量变成定义的，当引用出现在表达式中的函数时，在表达式中并不需要函数值来确定表达式的值，则表达式求值时该变元或变量变为无定义的。

(4)在辅程序内执行 RETURN 语句或 END 语句，导致局部于它的作用域单位的所有变量，或对递归引用它的作用域单位局部于当前实例的所有变量都变为无定义的，但下列情况除外：带有 SAVE 属性的变量；在空白公用块中的变量；出现在辅程序内或出现在直接或间接引用该辅程序的另一个作用域单位内的有名公用块中的变量；由宿主作用域单位访问的变量；由模块访问的变量，若至少一个作用域单位直接或间接引用该模块并此模块直接或间接地引用该辅程序；在已初始定义的且已不继续定义或再定义的有名公用块中的变量。

(5)在输入语句执行期间，当出现一个差错条件或文件结束条件时，由语句的输入项表或名表组说明的所有变量变为无定义的。

(6)当执行输入/输出语句期间出现差错条件、文件结束条件或记录结束条件时，某些或全部隐 DO 变量可能变为无定义的。

(7)执行定义的赋值语句，在赋值号前面可能遗留全部或部分无定义的变量。

(8)对以前未写过的一个记录的执行一个直接访问输入语句导致由该语句输入项表说明的全部变量变成无定义的。

(9)执行 INQUIRE 语句可能导致 NAME＝、RECL＝和 NEXTREC＝变量变为无定义的。

(10)当一个字符存储单元变为无定义时，所有与它结合的字符存储单元变为无定义的。

当一个数值存储单元变为无定义时，所有与它结合的数值存

储单元变为无定义的，除了是定义与一个不同类型的数值存储单元相结合的结果之外。

当双精度实型的一个实体变为无定义时，所有与它整体结合的双精度实型实体变为无定义的。

当未说明的存储单元变为无定义时，所有与它结合的未说明的存储单元变为无定义的。

(11)若用语句标号值来定义实元，则该过程引用导致相应的虚元部分变成无定义的。

(12)当可分配数组被去分配时，它变为无定义的。成功地执行一个 ALLOCATE 语句导致已分配的数组变为无定义的。

(13)执行 INQUIRE 语句时若存在差错条件，导致除了可能有的 IOSTAT＝说明符中的变量之外的所有查询说明符变量变为无定义的。

(14)当引用一个过程时：不和实元结合的可选虚元是无定义的；具有 INTENT(OUT)的虚元是无定义的；与具有 INTENT(OUT)的虚元结合的实元变为无定义的；若实元的子对象是无定义的，则相应虚元的子对象是无定义的；函数的结果变量是无定义的。

(15)当指针的结合状态变为无定义的或分离的时，该指针变为无定义的。

以上列出了导致变量变成无定义的所有可能出现的15种事件。

本节中对于定义与无定义给出了较为详细的阐述。这些对于程序设计者在程序的编制中是很有用的。

小　结

本章中对于 Fortran 90中一些比较重要而在使用中又较难处理的概念给出了较为详细的说明，如作用域的含意、什么情况下使用使用域、变量在什么情况下会变成定义的或无定义的、结合有几种类型、在什么情况下会使用结合等。这些问题对初学者可能较难

理解，不过随着学习的深入和编程序经验的丰富，读者会越来越感到这些问题的重要性。

习　　题

1. 作用域单位有那几种？

2. 什么是宿主作用域单位？

3. 有多少种结合？它们的差别是什么？

4. 什么情况下导致变量是定义的？什么情况下导致变量是无定义的？

第十三章　数据导出类型和说明

随着程序设计语言的广泛应用，Fortran 77过去所定义的各种类型已有一定的局限性，例如要描述人员的有关信息，它是一个多种类型的有序组合的结果。姓名是字符型的，性别又是字符型的，年龄是整型的……，对此用原有的类型描述是不行的。Fortran 90引进了导出类型来解决这类问题。本章将介绍有关导出类型的定义、值的表示以及其上可施行的运算等。

第一节　导出类型的定义

导出类型的对象具有若干成分，每个成分是某种内在类型的或是已定义的导出类型的。内在类型是语言预定义的，而导出类型是由用户定义的，所以必须在可执行程序中定义。内在类型总是可访问的，而导出类型只可在其定义的作用域内访问它们。

导出类型定义的形式是：

```
TYPE 〔〔，访问说明〕：：〕类型名
  〔私用—顺序语句〕：：
  成分定义语句
  〔成分定义语句〕…
END TYPE 〔类型名〕
```

其中，私用—顺序语句是 PRIVATE 或 SEQUENCE，在一给定的导出类型定义中，私用—顺序语句可以不出现，可以出现其中的一个，可以这两个都出现。但不允许以其它的形式出现，例如出现两个 PRIVATE 语句。

若出现 SEQUENCE 语句，则在成分定义中说明的所有导出类型必须也是顺序类型的（即其中也必须出现 SEQUENCE），并

且在定义中成分定义的次序就说明了该类型的对象的存储序列。

关于可访问性说明和 PRIVATE 语句的作用将在下一节中说明。

注意:类型名禁止与内在类型的名字相同,也禁止与其它可访问的导出类型的类型名相同。在 END TYPE 中出现类型名的话,一定要与开头的定义语句中的类型名相同。

从定义的形式可知成分定义语句至少要出现一个,其形式是:

ts 〔〔, cas 表〕::〕cd 表

其中,cas 或是

POINTER

或是

DIMENSION(cars)

cd 是成分名〔(cars)〕〔* 字符长度〕

cars 是显形说明表或迟形说明表

所谓"表"是指用逗号隔开的一些项。注意在一个给定的成分定义语句中,cas 不得出现多次,而且如果对一个成分没有说明 POINTER 属性,则 ts 应是内在类型或先前已定义的导出类型,并且 cars 必须是显形说明表。若一个成分说明了 POINTER 属性,则 ts 需说明为内在类型或包括正在定义的类型在内的任何可访问的导出类型,并且 cars 必须是迟形说明表。* 字符长度可选项只允许出现在类型说明是字符型时,出现时必须是常量说明表达式。

若成分是定义语句中包含 DIMENSION 属性,或成分名后含 cars,则该成分是数组。若成分名后有 cars,则它们说明该成分数组的界,否则按 DIMENSION 属性的 cars 说明该成分数组的界。

导出类型的例子是:

〔例1〕
```
TYPE PERSON
   INTEGER AGE
   CHARACTER (LEN=50) NAME
END TYPE PERSON
```

这个例子说明导出类型 PERSON 有两个成分构成,一为默认

整型标量，一为默认字符型标量。声明变量 CHAIRMAN 是导出类型 PERSON 的例子是：

```
TYPE (PERSON) :: CHAIRMAN
```

独立引用成分时用指定符 CHAIRMAN% AGE，或 CHAIRMAN% NAME。

〔例2〕
```
TYPE LINE
  REAL, DIMENSION (2,2)::COORD
  REAL            ::WIDTH
  INTEGER         ::PATTERN
END TYPE LINE
```

这个例子中说明导出类型 LINE 有三个成分构成，一为默认实型的二维数组，一为默认实型的标量，一为默认整型的标量。声明变量 LINE_SEGMENT 是 LINE 类型的例子是：

```
TYPE (LINE) :: LINE_SEGMENT
```

这样标量变量 LINE_SEGMENT 具有一个成分是数组。该数组是标量的子对象。

〔例3〕 具有私用成分的导出类型例子：

```
MODULE DEFINITIONS
  TYPE POINT
    PRIVATE
    REAL :: X,Y
  END TYPE POINT
END MODULE
```

此类型定义在通过 USE 语句访问了该模块的任何作用域单位中是可访问的；然而，成分 X 和 Y 仅在该模块中是可访问的。

〔例4〕 导出类型定义的成分可以是导出类型的。

```
TYPE TRIANGLE
  TYPE (POINT) :: A,B,C
END TYPE TRIANGLE
```

〔例5〕 私用类型的例子是：

```
TYPE, PRIVATE :: AUXILIARY
```

```
    LOGICAL :: DIAGNOSTIC
    CHARACTER (LEN=20) :: MESSAGE
  END TYPE AUXILIARY
```

这样的类型只能在定义所在的模块中是可访问的。

〔例6〕 顺序类型的导出类型的例子：

```
TYPE NUMERIC_SEQ
  SEQUENCE
  INTEGER :: INT_VAL
  REAL    :: REAL_VAL
  LOGICAL :: LOG_VAL
END TYPE NUMERIC_SEQ
```

这是一个顺序类型的例子，它规定编译程序给它分配存储单元时应按其中成分出现的次序安排存储序列。

第二节　几种属性说明

在第四章及其它一些地方已分别介绍了多种数据的属性及其说明的方法，这些属性既可在类型语句中加以说明，也可用相应的属性语句来加以说明。在此我们再介绍几个属性。

一、可访问性属性

可访问性属性说明的形式是：

PUBLIC

或

PRIVATE

这种可访问性属性说明只能出现在模块作用域单位中。它说明实体或导出类型定义的可访问性。PRIVATE 属性已在前一节中说明，在此就有关内容再简要复述一下。声明具有 PRIVATE 属性的实体在模块外是不可访问的，声明具有 PUBLIC 属性的实体在其它程序单元中用了 USE 语句就可访问。对于没有显式说明可访问性的实体具有默认的可访问性，即除非用 PRIVATE 语句

来改变它的属性，实体总是默认可访问的。

可访问性说明的形式也有两种，一种是出现在类型语句中：

ts，可访问性说明:: 实体声明表

例如：

REAL，PRIVATE :: X，Y，Z

另一种形式是可访问性语句：

PUBLIC〔〔::〕可访问的标识表〕

或

PRIVATE〔〔::〕可访问的标识表〕

每个可访问的标识或是使用名或是类属说明。在模块的作用域单位中省略可访问的标识表的可访问性语句只允许一个，而且说明该模块作用域单位中所有潜在可访问的实体所具有的可访问性由此语句给出，例如语句：

PUBLIC

则设置共用的可访问性，而语句：

PRIVATE

则设置私用的可访问性。若在模块中设有这样的语句出现，则默认是共用的可访问性。

可访问性语句每个使用名必须是有名变量、过程、导出类型、有名常量或名表组的名字。

一个具有 PRIVATE 可访问性的虚元或函数结果的模块过程必须具有 PRIVATE 可访问性，该模块过程不得有具有 PUBLIC 可访问性的类属标识符。

一个过程具有共用的类属标识符，通过类属标识符总是可访问的，即使其特定名具有私用的可访问性也不例外。

在导出类型定义中要注意访问说明所在的位置，若 TYPE 后有私用的可访问性，则说明该导出类型定义是私用的，只能在定义导出类型的模块中使用此导出类型定义。若在私用—顺序语句中出现 PRIVATE 语句，是说明类型定义中的成分是私用的，但类型定义本身是共用的，而其成分只能在定义该导出类型的模块中

访问。

二、POINTER 属性

POINTER 属性说明数据对象是指针，若该指针是一数组，则该数组必须是迟形数组。POINTER 属性说明办法也有两种，一种是在类型语句中用属性说明指出具有 POINTER 属性，具体形式是：

ts〔,〕POINTER :: 实体声明表

另一种形式是 POINTER 语句，其形式是：

POINTER 〔::〕对象名表

每项对象的形式是：

对象名〔(迟形说明表)〕

对象名被禁止说明 INTENT 属性及 PARAMETER 属性，对象名除非在该作用域单位的别处说明 DIMENSION 属性，否则必须有迟形说明表。

说明了具有 POINTER 属性的对象，只有当执行了指针赋值或 ALLOCATE 语句后，使它与目标对象相结合，变成与可以引用或定义的目标对象相结合的指针之后才能对它引用和定义。例如：

```
TYPE (NODE), POINTER :: CURRENT, TAIL
REAL, DIMENSION (:,:), POINTER :: IN, OUT
TYPE (NODE) :: CURRENT
POINTER :: CURRENT, A(:,:)
```

最后对数组指针还要重复说几句。在介绍迟形数组时曾指出，迟形数组除了可分配数组之外还有数组指针。数组指针在与目标结合之前，数组指针的目标的大小、界和形状都是无定义的。这时，这样的一个数组的任何一部分都不可被定义或引用。但可作一些内在查询函数的变元，以便查询关于变元的出现性、类型、类型参数或结合的状态等。

指针虚元是只能与指针实元相结合的变元。而指针实元则有

时可以与非指针虚元相结合。

三、TARGET 属性

TARGET 属性说明对象可以作为与指针结合的目标。没有TARGET 属性或 POINTER 属性的对象不得与可访问的指针相结合。TARGET 属性也有两种说明方法，其一是通过类型说明：

ts，TARGET :: 实体声明表

另一种形式是用 TARGET 语句，其形式是：

TARGET :: 对象名表

每个对象的形式是：

对象名〔(数组说明)〕

其中对象名被禁止说明具有 PARAMETER 属性。这个语句说明其中的对象名具有 TARGET 属性，因而可以有与其结合的指针。例如：

```
TYPE (NODE), TARGET :: HEAD
REAL, DIMENSION (1000,1000), TARGET :: A
TARGET :: A(1000,1000)
```

四、指 针 赋 值

指针赋值可导致一个指针变成与目标相结合的，也可导致指针的结合状态变成分离的（即不结合的）或无定义的。指针赋值语句的形式是：

指针对象 => 目标

其中目标是一个变量或表达式。

注意：指针赋值的符号形状上像一个箭头，先等号后大于号，如果交换次序就变成关系运算符>=，这点一定不能弄错。

指针对象必须有 POINTER 属性，目标必须是有 TARGET 属性的，或是具有 TARGET 属性的对象的子对象，或者它又有 POINTER 属性。目标必须和指针具有相同的类型、类型参数和秩，但目标不得是具有向量下标的数组片段，作为目标的表达式的

结果必须是指针。

若目标不是指针，指针赋值语句把指针对象和目标相结合。若目标是一个结合的指针，则指针对象和与目标所结合的对象结合。若目标是一个非结合的指针，则指针对象也变成非结合的。若目标是一个具有无定义的结合状态的指针，则指针对象也变成一个无定义的结合状态。任何以前在指针对象和目标之间的结合被破坏。例如：

```
NEW_NODE % LEFT=>CURRENT_NODE
SIMPLE_NAME=>STRUCTURE % SUBSTRUCT % &
    COMPONENT
ROW=>MAT2D(N,:)
WINDOW=>MAT2D (I-1:I+1,J-1:J+1)
POINTER_OBJECT=>POINTER_FUNCTION (ARG_1,&
    ARG_2)
EVERY_OTHER=>VECTOR (1:N:2)
```

都是指针赋值语句的例子。

五、ALLOCATE 语句的进一步介绍

ALLOCATE 语句除了动态地建立可分配数组，还可动态地建立指针目标。在前面介绍的 ALLOCATE 语句中，凡可出现可分配数组的地方都可出现指针。一个指针随着相应的 ALLOCATE 语句的成功执行，使该指针成为与目标相结合的，且成为可引用或定义的。指针的分配建立了一个隐含具有 TARGET 属性的对象。此外，指针还可通过指针赋值变成与指针目标或指针目标的一部分相结合的。分配当前已与目标结合的一个指针不是错误，这时，按 ALLOCATE 语句中说明的任何数组界和指针属性所要求的建立一个新的指针目标。然后这指针与新的目标相结合。该指针与任何先前结合的目标被断开。若先前的目标已用分配语句所建立，则它变为不可访问的，除非它仍然能用当前与其结合的其他指针来引用。内在函数 ASSOCIATED 可以用来确定指针当前是否是结合的。

六、NULLIFY 语句

NULLIFY 语句的形式是：

NULLIFY（指针对象表）

每个指针对象是变量名或结构成分。该语句的执行导致指针变为分离的，即变为不结合的。

注意：每个指针对象需具有 POINTER 属性。

七、DEALLOCATE 语句

DEALLOCATE 语句除了可导致可分配数组变为去分配的，也能导致指针目标变为去分配的并且指针变为分离的。在语句中可出现可分配数组的地方也可出现指针。当一个指针出现在该语句中，那么指针的结合状态必须是定义的。去分配一个已分离的或其目标没用 ALLOCATE 语句分配的指针，将导致该语句的一个错误条件。若一指针与一可分配数组当前是结合的，则该指针禁止去分配。对一个当前不是与已分配的目标对象的整体结合的指针也禁止去分配。若当前一个指针是与一个目标对象的一部分结合的，而且这一部分独立于该目标对象的任何其它部分，则它也禁止去分配。去分配一个指针目标将导致与该目标或目标的一部分结合的任何其它的指针的结合状态变成无定义的。

第三节　导出类型的确定

在一作用域单位中一个具体的类型名至多能定义一次。具有相同类型名的导出类型定义可以出现在不同的作用域单位中，在这种情况下它们可能描述独立不同的导出类型，也可能描述相同的导出类型。

若引用相同导出类型定义声明的两个数据实体，则它们具有相同的类型。可从一个模块或一个宿主作用域单位访问此定义，不同作用域单位中数据实体也可以具有相同的类型，只要它们声明

引用不同导出类型定义，而这些定义具有相同的名字、都具有 SEQUENCE 性质、结构成分不具有 PRIVATE 可访问属性、结构成分在次序、名字和属性都一致。否则，它们是不同导出类型的。一个使用具有 SEQUENCE 性质的类型所声明的数据实体与 PRIVATE 的或具有成分是 PRIVATE 的所声明的类型的实体是不同类型的。例如：

```
TYPE POINT
  REAL X,Y
END TYPE POINT
TYPE (POINT)::X1
CALL SUB (X1)
    ...
CONTAINS
    SUBROUTINE SUB (A)
      TYPE (POINT)::A
      ...
    END SUBROUTINE SUB
      ...
```

子程序 SUB 的作用域单位是被包含在其宿主程序单元的作用域单位中，所以子程序 SUB 中导出类型 POINT 的定义是知道的，因此 X1和 A 的声明都引用了相同的导出类型定义，所以 X1 和 A 具有相同的类型。

```
PROGRAM PGM
  TYPE EMPLOYEE
    SEQUENCE
    INTEGER      ID_NUMBER
    CHARACTER (50) NAME
  END TYPE EMPLOYEE
  TYPE (EMPLOYEE) PROGRAMMER
  CALL SUB (PROGRAMMER)
  ...
END PROGRAM PGM
```

```
SUBROUTINE SUB (POSITION)
  TYPE EMPLOYEE
    SEQUENCE
    INTEGER      ID_NUMBER
    CHARACTER (50) NAME
  END TYPE EMPLOYEE
  TYPE (EMPLOYEE) POSITION
  ...
END SUBROUTINE SUB
```

例中两个导出类型定义具有相同的名字、SEQUENCE 性质、各成分均不具有 PRIVATE 属性、各成分的名字、次序、类型、类型参数和秩都相同，所以实元 PROGRAMMER 和虚元 POSITION 具有相同的类型。如果有一个成分名为 ID_NUMBER，而另一个定义中相应成分名为 ID_NUM，则这实元和虚元具有不同的类型。

第四节　导出类型的值

导出类型值的集合由与导出类型定义相一致所有可能的成分值的序列所构成。

一、导出类型值的构造

导出类型的定义隐含地定义了相应的结构构造符，构造符允许从值的序列构造导出类型的标量值。该标量值的序列中各个值就是导出类型各个成分的值。导出类型的结构构造符在实际书写时写成类型名后跟用括号括着的表达式表，其中每个表达式的类型要与相应成分的类型相一致。这里所谓相一致是指：当成分是算术型时表达式也要是算术型，当成分是逻辑型时表达式也要是逻辑型，当成分是字符型时要求表达式是种别类型参数相同的字符型。所以表达式求值后，可能各个值需按内在赋值规则进行转换，转换为与成分的类型和类型参数相一致的值。对于非指针成分

还要求表达式的形状需符合相应成分的形状。当表达式都是常量表达式时，结构构造符是导出类型常量表达式。结构构造符必须出现在其导出类型定义之后。

例如，对导出类型定义：

```
TYPE PERSON
  INTEGER AGE
  CHARACTER (LEN=50) NAME
END TYPE PERSON
```

可有结构构造符：

```
PERSON (21,'JOHN SMITH')
```

当导出类型的一个成分是数组时，或一个对象是导出类型的数组时，对这样的数组可以使用数组构造符来构造。

例如，对类型语句

```
TYPE (PERSON), DIMENSION (2) :: A
```

可有数组构造符为：

```
(/ PERSON (21,'JOHN SMITH'), PERSON (22,'MARY &
BLANCK')/)
```

例如：

```
TYPE PERSON1
  INTEGER AGE
  INTEGER, DIMENSION (2) :: TEL _ NUM
  CHARACTER (LEN=50) NAME
END TYPEPERSON1
```

可有结构构造符：

```
PERSON1(21,(/3014567, 2568899/),'JOHN SMITH')
```

当成分是指针时，则表达式需对一个对象求值，这个对象将是在指针赋值语句中对此指针能允许的目标。例如：

```
TYPE REFERENCE
  INTEGER  ::VOLUME, YEAR, PAGE
  CHARACTER (LEN=50) :: TITLE
  CHARACTER, DIMENSION(:), POINTER :: ABS
END TYPE REFERENCE
```

```
CHARACTER, DIMENSION (1:400), TARGET::TEXT
TYPE (REFERENCE) ::BIBLIO
```

则语句：

```
BIBLIO=REFERENCE (5,1993,35, &
'This is the title of the referenced paper', TEXT)
```

是允许的，且语句的执行将对 BIBLIO 的 ABS 成分赋值，即赋以目标 TEXT 所具有的值。

二、导出类型的运算和赋值

在导出类型的实体上能施行的运算或对导出类型的实体的非内在赋值都要由用户自己定义。但是，其成分或成分的子对象可以作为一般表达式的初等量参与运算，也可以作为赋值语句的左部量。例如：

```
BIBLIO % VOLUME=5
BIBLIO % TITLE='This is a paper'
```

若有

```
TYPE (PERSON1) :: BOY
```

则语句

```
BOY % TEL_NUM(2)=9217468
A(2) % AGE=30
IF(A(1) % AGE<A(2) % AGE) THEN
   ...
```

都是合法的语句。

第五节　导出类型的说明语句

其实，导出类型的说明语句与一般类型说明语句相类似，仅是将一般类型说明语句中的 ts 写为 TYPE(类型名)，在前面大家已看到过不少的例子。在此我们只想再补充一点，就是 IMPLICIT 语句中也可出现导出类型的名，例如：

```
IMPLICIT (PERSON) (A-C)
```

TYPE PERSON

...

这个语句说明首字母隐含类型法则的内容为以 A、B 或 C 三个字母开头的名是 PERSON 类型的。

最后，我们再罗列一下各种类型说明：

整型	INTEGER 〔kind－selector〕
实型	REAL 〔kind－selector〕
双精度实型	DOUBLE PRECISION
复型	COMPLEX 〔kind－selector〕
字符型	CHARACTER 〔char－selector〕
逻辑型	LOGICAL 〔kind－selector〕
导出类型	TYPE(type－name)

此外，再将各种属性说明罗列于下：

PARAMETER

PUBLIC

PRIVATE

ALLOCATABLE

DIMENSION (array－spec)

EXTERNAL

INTENT (intent－spec)

INTRINSIC

OPTIONAL

POINTER

SAVE

TARGET

对有些属性说明，可有相应的说明语句：

(1)PARAMETER 语句。

(2)可访问性(PUBLIC，PRIVATE)语句。

(3)ALLOCATABLE 语句。

(4)DIMENSION 语句。

(5)EXTERNAL 语句。

(6)INTENT 语句。

(7)INTRINSIC 语句。

(8)OPTIONAL 语句。

(9)POINTER 语句。

(10)SAVE 语句。

(11)TARGET 语句。

小　　结

本章主要介绍了导出类型的定义、值、运算和赋值，类型的确定以及类型说明语句等。与其相关地介绍了指针(POINTER 属性)及相应的目标(TARGET 属性)，及有关的指针赋值语句，NULLIFY 语句，补充介绍了 ALLOCATE 语句及 DEALLOCATE 语句。这些都是 Fortran 90新增加的内容，它们对于 Fortran 语言用于新的应用领域起着重要的作用。

习　　题

1. 举出三个例子说明导出类型的用途。

2. 指针与目标这两个对象的引进有什么好处。试编程序举例说明。

3. 说明 ALLOCATE 语句和 DEALLOCATE 语句与指针的关系。

4. 试写出类型语句的完整形式。

第十四章　表达式和赋值语句(二)

在前面的有关章节中已对表达式和赋值语句作了不少的介绍，本章对 Fortran 中的一些与此有关的新成分作进一步的介绍和说明。

第一节　定义的运算

Fortran 90对运算扩充了定义的运算。这种扩充首先是因为对数据类型作了扩充，除了内在类型外还有导出类型，在内在类型中还扩充了种别类型参数。这些扩充使 Fortran 语言描述的数据对象范围更广，这就导致了定义的运算的概念。先来看一个例子。由于有了导出类型，我们可以定义实数的区间：

```
TYPE INTERVAL
  REAL LOW, UP
END TYPE INTERVAL
```

区间由下界和上界构成。对于两个有部分重叠的区间可以作并运算，现在来定义运算：

```
FUNCTION COMBINE_INTERVALS (A,B)
    TYPE (INTERVAL) COMBINE_INTERVALS, A, B
    COMBINE_INTERVALS %LOW= MIN(A%LOW, &
      B%LOW)
    COMBINE_INTERVALS %UP= MAX(A%UP, B%UP)
END FUNCTION COMBINE_INTERVALS
```

再写出接口块：

```
INTERFACE OPERATOR (.COMB.)
    MODULE PROCEDURE COMBINE_INTERVALS
END INTERFACE
```

这样，对于区间 X，Y 和 Z：

```
TYPE (INTERVAL) X, Y, Z
```

可用赋值语句：

```
X=Y.COMB.Z
```

就可得到区间 Y 和 Z 之并，且存于 X 之中。事实上用定义的运算还可定义集合的和、差、并、交等，总之定义的运算可以使得用户方便地表达系统提供内在运算之外的许多运算。

一、定义的二元运算

定义的二元运算有两种形式，它们是：

X_1定义的二元运算符 X_2

它用一个函数和一个类属接口块定义，或

X_1内在运算符 X_2

其中，X_1或 X_2或两者的类型和秩不是原内在二元运算符所要求的，这种定义的二元运算由一个函数和一个类属接口块定义。

关于函数和接口块在下一章有详细的介绍。这种定义的二元运算符是一种扩充的运算符，其书写是二个园点之间的一个字母串，其字母个数最多不得超过31个，且不能与逻辑型字面常量、关系运算符、逻辑运算符等相同。上述的.COMB.就是一例。

但是，如果将上例中接口块改写为：

```
INTERFACE OPERATOR (+)
    MODULE PROCEDURE COMBINE_INTERVALS
END INTERFACE
```

则

```
X=Y+Z
```

就表示先求区间 Y 和 Z 之并，然后存于 X 中。这也是一种正确的定义二元运算的方法。但是要注意这里 Y 和 Z 是导出类型的，符合前面介绍的其类型不是原内在二元运算＋所要求的，且有一个函数和类属接口块来定义的，所以是正确的。但是，当 Y 和 Z 是实型时，就不能这样来定义二元运算了，而.COMB.所定义的二元运算符就没有这条限制。

在下一章中我们还可看到，用类属接口块时可以说明多个函数，定义多个运算。

在第六章中我们已经说过定义的二元运算符在运算符的优先级中是最低的。

用函数定义的二元运算 X_1.OP.X_2还要注意下列几点：

(1)该函数是用有两个虚元 d_1和 d_2的 FUNCTION 或 ENTRY 语句说明的；

(2)接口块用 OPEATOR(.OP.)的说明提供函数；

(3)X_1和 X_2的类型分别与相应虚元 d_1和 d_2的相同；

(4)如果有类型参数的话，X_1和 X_2的类型参数也分别与相应虚元 d_1和 d_2相匹配；

(5)如果 X_1和 X_2中的一个或两个是数组，X_1和 X_2的秩和形状也分别与相应虚元 d_1和 d_2相匹配。

(6)运算结果的类型、类型参数等由函数中定义。

二、定义的一元运算

定义的一元运算也有两种形式，它们是：

定义的一元运算符 X_2

它由一个函数和一个类属接口块定义，或

内在运算符 X_2

其中，X_2的类型不是原内在运算所要求的，这种定义的一元运算由一个函数和一个类属接口块定义。

定义的一元运算符也是扩充的运算符，其书写规定与定义的二元运算符完全一样。例如：

```
FUNCTION EQUAL_INTERVAL (A)
    TYPE (INTERVAL) EQUAL_INTERVAL,A
    EQUAL_INTERVAL % LOW = A % LOW
    EQUAL_INTERVAL % UP = A % UP
END FUNCTION EQUAL_INTERVAL
```

接口块：

```
INTERFACE OPERATOR (.EQUAL.)
```

```
        MODULE PROCEDURE EQUAL _ INTERVAL
    END INTERFACE
```

这样，对于区间 M，N

```
    TYPE (INTERVAL) M, N
```

可用赋值语句

```
    M=.EQUAL.N
```

得到 M 是与 N 相同的区间。

若接口块写为：

```
    INTERFACE OPERATOR (+)
        MODULE PROCEDURE EQUAL _ INTERVAL
    END INTERFACE
```

则

```
    M=+N
```

得到与上述同样的结果。

在这里要注意的是当用内在运算符来定义一元运算时，一定要使 X_2 的类型不是原内在运算所要求的，且用一个函数和一个类属接口块定义。前述的例子是符合要求的。例如当 N 是实型时就不能用＋来作定义的运算符。

在第六章中我们已经说过定义的一元运算符在运算符的优先级中是最高的。

用函数定义的一元运算.OP. X_2 还要注意下列几点：

(1)该函数是用有一个虚元 d_2 的 FUNCTION 或 ENTRY 语句说明的；

(2)接口块用 OPERATOR (.OP.)的类属说明提供函数；

(3)X_2 的类型与虚元 d_2 的类型相同；

(4)如果 X_2 有类型参数，它与虚元 d_2 的相匹配；

(5)如果 X_2 是数组，则它的秩和形状应与虚元 d_2 的相匹配；

(6)运算结果的类型等是由相应函数定义的。

第二节　定义的赋值语句

Fortran 90中还扩充了定义的赋值语句。这种扩充同样是基

于对数据类型扩充后，以扩充定义的赋值语句满足不同用户的需要。

在导出类型中我们介绍过有一种内在赋值语句，但它限制赋值号两边具有相同类型的变量和表达式，类型不同是不行的。但实际问题中确实存在要求两边类型不同的问题，这就需要借助于定义的赋值语句来解决问题。例如对前面定义的实数区间，如果要求将一个区间的中点赋给一个实型标量变量，就可以编制这样的定义的赋值语句来实现。

```
SUBROUTINE REAL_MIDDLE_POINT_INTERVAL (A,B)
    REAL A
    TYPE (INTERVAL) B
    A=(B%LOW+B%UP)/2
END SUBROUTINE
```

和接口块：

```
INTERFACE ASSIGNMENT (=)
    MODULE PROCEDURE REAL_MIDDLE_POINT_&
      INTERVAL
END INTERFACE
```

则对于区间 X 及实型量 MID，这定义的赋值语句

```
MID=X
```

就是把 X 区间的中点值赋给 MID。

再假定一个导出类型为另一个导出类型的子集，其成分是抽取了后一导出类型的部分成分，这同样也可用定义的赋值语句来实现，例如：

```
TYPE RECORD1
    CHARACTER (LEN=30) NAME
    CHARACTER (LEN=1) SEX
    INTEGER YEAR, MONTH, DAY
    CHARACTER (LEN=20) ADDRESS
    INTEGER TEL_NUM_0, TEL_NUM_H
END TYPE RECORD1
```

```
TYPE RECORD2
    CHARACTER (LEN=30) NAME
    CHARACTER (LEN=20) ADDRESS
    INTEGER TEL_NUM_0, TEL_NUM_H
END TYPE RECORD2
SUBROUTINE COMMUNICATION (A,B)
    TYPE (RECORD2) A
    TYPE (RECORD1) B
    A% NAME=B% NAME
    A% ADDRESS=B% ADDRESS
    A% TEL_NUM_0=B% TEL_NUM_0
    A% TEL_NUM_H=B% TEL_NUM_H
END SUBROUTINE
INTERFACE ASSIGNMENT (=)
    MODULE PROCEDURE COMMUNICATION
END INTERFACE
```

若说明：

```
   TYPE (RECORD1) X
   TYPE (RECORD2) Y
```

则 Y=X 的执行结果是在 Y 中存有 X 导出类型的四个成分组成一个新的导出类型变量。从上面的例子可以看出，定义的赋值语句是一个赋值语句，但不是内在赋值语句，它是由子程序和说明 ASSIGNMENT(=)的接口块定义的。

关于子程序定义所定义的赋值语句 $X_1=X_2$，要注意下列几点：

(1)子程序是用有两个虚元 d_1和 d_2的 SUBROUTINE 或 ENTRY 语句说明的。

(2)接口块中采用带有 ASSIGNMENT(=)的类属说明的子程序。

(3)X_1和 X_2的类型与相应虚元 d_1和 d_2的相同。

(4)如果 X_1和 X_2有类型参数，分别与相应虚元 d_1和 d_2的相匹配。

(5)如果 X_1和 X_2中有一个或两个都是数组，它们的秩和形状，分别与相应虚元 d_1和 d_2的相匹配。

(6)X_1和 X_2不能两者都是具有同样种别类型参数的数值型、逻辑型或字符型的。

第三节　屏蔽数组赋值语句

在第六章中已介绍过数组赋值语句，这种数组赋值语句为向量或并行处理系统发挥其效率提供了方便。但是在有些情况下这种赋值往往是有条件的，而带有条件的赋值语句在向量化或并行化时会遇到一定的困难。Fortran 90中为此提供了屏蔽数组赋值语句，它实质上是一种带条件的数组赋值语句。

屏蔽数组赋值语句或者是 WHERE 语句，或者是 WHERE 构造。

WHERE 语句的形式是：

```
WHERE (屏蔽表达式)赋值语句
```

其中，屏蔽表达式是数组逻辑表达式。

WHERE 构造的形式是：

```
WHERE (屏蔽表达式)
  [赋值语句]...
[ELSE WHERE]
  [赋值语句]...]
END WHERE
```

其中，屏蔽表达式也是数组逻辑表达式。

这里要求所有数组赋值语句中被定义的变量和屏蔽表达式必须都是数组，而且形状相同。而且不能是定义的赋值语句。

这屏蔽数组赋值要根据屏蔽表达式，也就是逻辑表达式的值来决定是否要对赋值语句中表达式求值以及是否要赋值。

说详细一些是，在执行 WHERE 语句时，当数组逻辑表达式的某个数组元素的值为.TRUE.时，要求对赋值语句中的表达式

的相应元素求值，并按内在赋值规则赋给对应的被定义的数组元素。

在执行 WHERE 构造时，逻辑表达式求值其结果由处理系统加以保持。对保持的数组，其元素的值为.TRUE.时，相应的屏蔽表达式之后的每个赋值语句被顺序地求值，并赋给相应的元素，直到可能有的 ELSE WHERE 语句或一定有的 END WHERE 语句前一语句执行完为止。若其元素为.FALSE.，并且有 ELSE WHERE 语句后的赋值语句时，则相应的每个赋值语句被顺序求值，并赋给相应的元素，直到 END WHERE 语句前一个语句执行完为止。

下面我们举几个简单的例子来说明：如果我们对数组 A 中，每个元素绝对值大于1都取其倒数，而其余元素不变，用 WHERE 语句可表示为：

```
WHERE ( ABS(A)>= 1.0 ) A=1.0/A
```

如果我们还要记录那些元素作了变换，则可用：

```
B=.FALSE.
WHERE ( ABS(A)>=1.0 ) B=.TRUE.
WHERE ( ABS(A)>=1.0 ) A=1.0/A
```

完成。实际上最后一个语句也可写为：

```
WHERE (B) A=1.0/A
```

但是不能写成：

```
WHERE ( ABS(A)>=1.0 ) A=1.0/A
WHERE ( ABS(A)>=1.0 ) B=.TRUE.
```

这就得不到原先希望的结果。

很显然写两个 WHERE 语句要两次计算屏蔽表达式，在运行时效率可能不高，这时就可应用 WHERE 构造，例如：

```
B=.FALSE.
WHERE ( ABS(A)>=1.0 )
    B=.TRUE.
    A=1.0/A
END WHERE
```

注意这里 B=.TRUE. 和 A=1.0/A 两个语句的顺序可以任意。这还可以写为

```
WHERE ( ABS(A)>=1.0 )
    B=.TRUE.
    A=1.0/A
ELSE WHERE
    B=.FALSE.
END WHERE
```

第四节　表达式的进一步说明

表达式的数据类型和形状，依赖于表达式中的运算符。在表达式中使用的初等量的数据类型和形状，在前面的介绍中已经知道，这是由表达式的语法形式递归地确定的。

一、初等量的数据类型、类型参数和形状

初等量的数据类型、类型参数和形状是按照该初等量的具体形式来确定的。

当初等量是常量时，其类型、类型参数和形状就是该常量的类型、类型参数和形状。

当初等量是结构构造符时，它是标量，其类型由结构名决定。

当初等量是数组构造符时，其类型和类型参数是数组构造符表达式的类型和类型参数，其形状是秩为1的数组，其大小是数组构造符表达式的个数。当数组构造符中含隐 DO 控制时，表达式的个数是按隐 DO 展开后得到的个数。它的形状也可用内在函数 RESHAPE 来改变。

当初等量是变量或函数引用时，其类型、类型参数和形状分别是变量或函数引用的类型、类型参数和形状。但特别要注意，在函数引用时，函数可以是类属的，此时函数引用的类型、类型参数和形状由实元的类型、类型参数和形状所决定的。

当初等量是括号括著的表达式时，则其类型、类型参数和形状就是该表达式的类型、类型参数和形状。

当初等量是指针时，如果它对应非指针的虚元，结合的目标对象就可被引用，初等量的类型、类型参数和形状就是当前目标的类型、类型参数和形状。如果它不与目标结合，它只能作为实元出现，而且这个实元是与引用的过程说明的虚元相对应的实元。

二、运算结果的类型、类型参数和形状

对于内在运算结果的类型、类型参数和形状在第六章已有较详细的叙述。我们总结列于表14—1中。

表14—1

内在运算符 OP	X_1的类型	X_2的类型	$[X_1]OPX_2$的类型	$[X_1]OPX_2$的类型参数	$[X_1]OPX_2$的形状
一元+,-		I,R,Z	I,R,Z	①	⑧
二元+,-,	I	I,R,Z	I,R,Z	④①①	⑨
*,/,	R	I,R,Z	R,R,Z	②③③	
**	Z	I,R,Z	Z,Z,Z	②③③	
//	C	C	C	⑤	⑨
.EQ.,	I	I,R,Z	L,L,L	⑥	⑨
.NE.,	R	I,R,Z	L,L,L		
==,	Z	I,R,Z	L,L,L		
/=	C	C	L		
.GT.,>, .GE.,>=,	I	I,R	L,L	⑥	⑨
.LT.,<,	R	I,R	L,L		
.LE.,<=	C	C	L		
.NOT.		L	L	①	⑧
.AND.,.OR., .EQV.,.NEQV.	L	L	L	⑦	⑧

符号说明类型中字母I,R,Z,C和L分别表示整型、实型、复

型、字符型和逻辑型：

①与X_2的类型参数相同。

②与X_1的类型参数相同。

③当X_1与X_2有相同的种别类型参数时，与X_1或X_2的类型参数相同。当X_1与X_2有不同的种别类型参数时，若十进制精度不同，则与十进制精度大的那个操作数的种别类型参数相同。当X_1与X_2有不同的种别类型参数时，且十进制精度相同，则由处理系统决定。

④当X_1与X_2有相同的种别类型参数时，与X_1或X_2的类型参数相同。当X_1与X_2有不同的种别类型参数时，若十进制阶码范围不同，则与十进制阶码范围大的那个操作数的种别类型参数相同。当X_1与X_2有不同的种别类型参数时，且十进制阶码范围相同，则由处理系统决定。

⑤X_1，X_2的种别类型参数必须相同，其种别类型参数与X_1或X_2的相同。其长度类型参数为X_1和X_2的长度类型参数之和。

⑥种别类型参数为 KIND(.FALSE.)。

⑦当X_1与X_2有相同的种别类型参数时，与X_1或X_2的类型参数相同。当X_1与X_2有不同的种别类型参数时，则由处理系统决定。

⑧与X_2的形状相同。

⑨当X_1是标量时，与X_2的形状相同。否则与X_1的形状相同。

对于定义的运算结果的类型、类型参数和形状由定义该运算的函数说明来确定。

三、运算的求值

1. 对操作数的要求

表达式求值的基础是运算的求值，在表达式中各种操作数在执行求值时，有一些必须的要求：

(1)当操作数是变量时，它必须已被定义。

(2)当操作数是函数引用时，它必须已被定义。

(3)当操作数是指针时，它必须与已定义的目标对象相结合。

(4)当操作数是整型时，它必须用整型值来定义。这里强调这一点的原因是，整型标量变量可能用标号定义，这是一种过时的功能，会导致很隐蔽的错误。

(5)当操作数为字符数据对象时，则引用的所有字符必须已被定义。

(6)当操作数为数组或数组片段时，则被选择的所有数组元素必须已被定义。

(7)当操作数为一结构时，则其所有成分必须已被定义。

2. 求值时一些被禁止的情况

在求值时，禁止其结果在数学上没有定义的数值运算。例如任何数除以零，零值的零次幂，零值的负数次幂，负的实型数的实型次幂等都是不允许的。

在函数引用求值时，除了下述特殊情况之外，它必须不影响语句中任何其它实体的求值。同时也不受语句中任何其它实体求值的影响。例如：

```
A=X+F(X,Y)+C*H
```

函数引用 F(X,Y)执行时不得影响 X、C 和 H，同时在 X 和 C * H 求值时也不得影响 F(X,Y)的求值。这特别在使用了 COMMON 语句时，有时会发生隐蔽的错误。

特殊例外的情况是在 IF 语句或 WHERE 语句的逻辑表达式求值时，该表达式的函数引用允许定义该逻辑表达式值为 .TRUE. 时要执行的语句中的变量。例如：

```
IF (L(X)) A=X+Y
WHERE (L2(X)) B=C*X
```

在逻辑表达式中 L(X)和 L2(X)函数引用求值时，它允许定义相应的 X，甚至 Y 或 C。

在函数引用求值时，若引起函数的变元变为定义的或无定义的，则任何与该变元结合的实元不得出现在同一语句中别的地方。例如：

```
A(I)=F(J,I)
```

```
Y=F2(X,Z)+X
```

在函数引用 F(J,I)求值时，如果函数定义中对 I 对应的虚元进行定义或变为无定义时，这个语句就是不允许的。同样在 F2(X,Z)求值时，如果函数引用中对 X 对应的虚元进行定义或变为无定义时，这个语句同样是不允许的。

在表达式中当有数组元素引用、数组片段引用或子串引用时，其下标表达式、片段下标或子串表达式的求值时不影响表达式的类型，表达式的类型也不影响它们的求值。

3. 操作数的求值

数组表达式中两个同形状的数组进行二元运算、或对一个数组进行一元运算时，都是对数组每个对应的元素之间逐个进行的。语言允许处理系统按任何的次序进行，先进行第一个，后进行第二个，再进行第三个……这仅是一种处理的次序，但决不是唯一的。

对于零大小的数组或零长度的字符串来讲，其数组元素的下标表达式或子串的始点和终点的表达式均可以不必计算。

如果一个表达式的值在计算了一部分表达式的值之后就能确定的话，可以不必计算其它部分的值。例如：

```
X.GT.Y.OR.L(Z)
```

当计算出 X 大于 Y 时，可以不求逻辑函数 L(Z)的值。又如表达式 F(Z)+B 中，当 B 是零大小的数组时，函数 F(Z)就不必求值。再如

```
CHARACTER (LEN=2) C1,C2,C3,CF
    ...
C1=C2//CF(C3)
```

因为 C2与 C1的长度相同，所以不必计算字符函数 CF(C3)的值。

但是要注意上述函数 L(Z)、F(Z)和 CF(C3)中，若只有执行函数引用时才对虚元进行定义，所以才对相应的实元有定义的话，那么在上例中会导致变元 Z 和 C3是无定义的。这就再次提醒编程序者，对函数来讲只有函数结果是必须加以定义的。良好的习惯是不对任何虚元加以定义，也就是说尽量不用函数的副作用，否则会导致程序出现隐蔽的错误，而且很难发现。

4. 括号的完整性

当程序写有括号时必须先计算括号内的表达式，当有多层括号嵌套时必须从内到外逐层计算。例如 A+(B−C)，其中 A、B 和 C 均是数值型的，必须先求 B−C 的值，然后将结果再加上 A。而绝对不能处理成数学上等价的(A+B)−C。

但在不违反括号完整性的原则下，处理系统可以求任何数学上等价的表达式的值。当两个表达式对各初等量的所有可能的值，其数学值都相等，则称这两个表达式在数学上是等价的。但等价的表达式可能产生不同的结果，例如(1.0/3.0) * 3.0与1.0的值是不同的结果，但这种差别是计算上的误差而不是数学上的差别。而整除与非整除是数学上的差别，5/2与5.0/2.0这结果上的差别就不是计算上的误差而是数学上的差别，这两个表达式是不等价的。

下面名字 A、B 和 C 表示任何实型或复型操作数，I 和 J 为整型操作数，X、Y 和 Z 是数值型操作数，L1和 L2为逻辑型操作数。表达式中有的可写成另一种允许的形式，有的则不允许，下面是一些例子：

表达式	允许的另一种形式
X+Y	Y+X
X * Y	Y * X
−X+Y	Y−X
X+Y+Z	X+(Y+Z)
X−Y+Z	X−(Y−Z)
X * A/Z	X * (A/Z)
X * Y−X * Z	X * (Y−Z)
A/B/C	A/(B * C)
A/5.0	0.2 * A
I>J	J−I<C
L1. AND. L2. AND. L3	L1. AND. (L2. AND. L3)

表达式	不允许的另一种形式
I/2	0.5 * I

X * I/J	X * (I/J)
X/J/A	X/(J * A)
(X+Y)+Z	X+(Y+Z)
(X * Y)-(X * Z)	X * (Y-Z)
X * (Y-Z)	X * Y-X * Z

小　　结

本章主要介绍了定义的运算、定义的赋值，它们分别用函数和类属接口块及子程序和类属接口块来定义，接口块的具体定义将在下一章详细叙述。此外还介绍了屏蔽数组赋值语句，这用来有条件地对数组的部分或全部元素进行赋值。最后对表达式作了进一步说明，提醒读者需特别注意的一些问题，以免发生一些隐蔽的错误。表达式和赋值语句是语言的重要核心内容，在程序中也是用得最多的内容，正确地使用将使读者得到预想的结果，读者一定要熟练地掌握有关内容，以便编写出正确和高质量的程序来。

习　　题

1. 试编出集合的交、并、差等定义的运算。

2. 编写从两种(数据库的)记录中抽取部分内容形成一个新记录的程序。

3. 用不同的方法编写给单位矩阵、上三角矩阵或下三角矩阵赋初值的程序。

4. 仔细分析上述表达式中不允许的另一种形式的例子，指出原因。

第十五章　模块与过程接口

模块(module)是 Fortran 90中新增加的一种程序单元。增加它的主要目的在于在过程之间提供另一种方便有效的共享常量、变量、类型定义以及过程的途径。我们知道过去在 Fortran 中在过程之间共享数据主要靠 COMMON 语句、EQUIVALENCE 语句以及通过变元来传递值。而前两种方法是作为 Fortran 语言特有的历史遗物保留下来的，为了保持语言的向上兼容性才保留了它们。因为运用它们并不是好的程序风格，所以在 Fortran 90中并不推荐使用它们。在过程之间通过变元互传数据自然是一种不可少的手段，但当需要共享的数据量很多时，这也不是一种方便有效的办法。而且为了保持虚元和实元之间正确的对应次序，当变元很多时也非易事，很容易出错。特别在某些场合下过程间数据的传递比较频繁时，这种共享数据的方法将会明显降低效率。此外，模块还可用来将一些过程打包成一个"库"，供许多不同的过程引用。

第一节　模块的定义

模块是一种程序单元但不能被直接执行。一个模块包含说明和定义两部分，其一般形式为：

模块语句

　　[说明部分]

　　[模块辅程序部分]

模块的 END 语句

其中，模块语句具有形式：

MODULE 模块名

模块的 END 语句具有形式：

END[MODULE[模块名]]

模块辅程序部分以 CONTAINS 语句开始，由若干个内部辅程序组成，形式上与外部辅程序中的内部辅程序部分相同。关于内部辅程序的定义和解释已在介绍辅程序（第九章）时作过说明，模块中的内部辅程序并没有什么特别的地方。

对模块说明有三个限制：

(1)如果模块的 END 语句中包含一个模块名，则它必须和模块语句中指明的模块名相同。

(2)在一个模块的说明部分中一定不能包含语句函数语句、ENTRY 语句或 FORMAT 语句。

(3)一个自动对象一定不能出现在一个模块的说明部分中。

由于模块名是全局于整个可执行程序的，所以它既不能与该可执行程序中的任何其它程序单元的名、外部过程名、或公用块名相同，也不能与本模块中的任何局部名相同。

虽然语句函数语句、ENTRY 语句和 FORMAT 语句都不能出现在模块的说明部分，但它们仍可出现在模块所包含的一个模块内部辅程序的说明部分。

模块是其包含的任一内部模块过程的“宿主(host)”，所以模块中说明的实体可被其内部模块过程通过宿主结合来访问。

例如，一个关于矩阵运算的程序包可用下列模块来定义：

```
MODULE MATRIX _ MODULE
  TYPE MATRIX
    INTEGER::M,N ! M×N 的矩阵
    REAL,DIMENSION(:,:),ALLOCATABLE::A
  END TYPE MATRIX
  CONTAINS
    FUNCTION MATRIX _ SUM(MATRIX _1,MATRIX _2) &
               RESULT(MATRIX _ SUM _ RESULT)
      ......
    FND FUNCTION MATRIX _ SUM
    FUNCTION MATRIX _ PROD(MATRIX _1,MATRIX _2) &
```

```
            RESULT(MATRIX_PROD_RESULT)
      ......
    END FUNCTION MATRIX_PROD
    FUNCTION ARRAY_TO_MATRIX(VALUES) &
            RESULT(ARRAY_TO_MATRIX_RESULT)
      ......
    END FUNCTION ARRAY_TO_MATRIX
    SUBROUTINE PRINT_MATRIX
      ......
     END SUBROUTINE PRINT_MATRIX
END MODULE MATRIX_MODULE
```

在此值得指出，上述模块中的函数和子程序的具体实现是外部不可见的，只要其调用形式不变，实现方法的改变不影响其它程序单元对它的引用。而且还可用 PRIVATE 属性来限制模块中的某些实体成为外部不可见，例如，下一例子中的变量 AMOUNT 就是这样。

在模块中可以只包含数据的说明，以供其它程序单元来共享。下面的模块就是这样一个例子：

```
MODULE SHARED_DATA_MODULE
  INTEGER,DIMENSION(:),ALLOCATABLE::LOST_CARD
  INTEGER::NUMBER_OF_LOST_CARDS
  REAL,PRIVATE::AMOUNT
  CHARACTER(LEN=10)::NAME
  CHARACTER(LEN=30)::ADDRESS
END SHARED_DATA_MODULE
```

上面我们陈述了模块定义的方法，接着我们来介绍如何运用它们。

第二节　USE 语句

前面已经讲到，模块的用途就是为在程序单元之间提供另一种方便有效的共享常量、变量、类型定义以及过程的途径。那么如何来共享它们呢？这就要用到 USE 语句。USE 语句的一般形式

为：

USE 模块名[,更名表]

或

USE 模块名,ONLY:[only 表]

其中更名表具有形式：

局部名＝＞使用名,局部名＝＞使用名,……

only 表具有形式：

访问标识,访问标识,……

或

[局部名＝＞]使用名,[局部名＝＞]使用名,……

在此要求：各个访问标识必须是该模块中的公共实体；各个使用名必须是该模块中的公共实体的名。

如果在更名表或 only 表中出现一个局部名，则它是由其相应的使用名指明的实体的局部名；否则局部名就是使用名。

USE 语句在没有 ONLY 选择时将提供对其指明的模块中的全部公共实体的访问权；在有 ONLY 选择时将仅提供对其指明的模块中的作为访问标识或 only 表中的使用名出现的公共实体的访问权。

在一个作用域中可以出现多个对于给定模块的 USE 语句。如果其中之一没有 ONLY 选择，则该模块中的全部公共实体是可访问的，而且，出现在多个 USE 语句中的更名表和 only 表被拼接在一起作为单个更名表来使用。如果几个 USE 语句都有 ONLY 选择，则仅对该模块中那些出现在一个或多个 only 表内的实体是可访问的，而且，出现在多个 USE 语句中的全部 only 表被拼接在一起作为单个更名表来使用。

一个实体的局部名在被一个 USE 语句引入成为可访问的之后，除了可以出现在一个模块的作用域内的 PUBLIC 或 PRIVATE 语句中之外，不再需要其它说明语句来重新说明其任何属性，即可出现在该 USE 语句所在的作用域中。值得注意，上述规定不允许这种局部名出现在 COMMON 和 EQUIVALENCE 说明

中，但是允许它们出现在名表组的表中。如果这样一个局部名出现在一个模块的 PUBLIC 语句中，则其标识的实体在该模块中成为一个公共实体。如果这样一个局部名出现在一个模块的 PRIVATE 语句中，则其标识的实体在该模块中就不是一个公共实体。如果这样一个局部名既不出现在一个模块的 PRIVATE 语句中也不出现在该模块的 PUBLIC 语句中，则其标识的实体在该模块中具有该作用域的默认访问属性。

例如，对于上面定义的 SHARED _ DATA _ MODULE 模块，使用语句：

```
USE SHARED _ DATA _ MODULE
```

将使得 SHARED _ DATA _ MODULE 模块中的所有公共实体成为可访问的。语句：

```
USE MOD1;USE SHARED _ DATA _ MODULE,SURNAME=>NAME
```

中，如果模块 MOD1 内也包含一个实体名为 NAME，为了避免与模块 SHARED _ DATA _ MODULE 内的实体同名，上面语句中把 SHARED _ DATA _ MODULE 模块内的 NAME 实体更名为 SURNAME。语句：

```
USE SHARED _ DATA _ MODULE,ONLY:NAME
```

将仅仅使公共实体 NAME 成为可访问的。

USE 语句就是为了把由模块说明的常量、变量、类型定义以及过程等提供给整个可执行程序共享。所以一当模块被定义之后，在各程序单元中就可用 USE 语句来引入模块中说明的实体。下面进一步举例说明。

```
MODULE AJAX _2002_ PARAMETERS
  INTEGER PARAMETER::SINGLE=4,&
                     DOUBLE=8,&
                     QUAD=12, &
                     SHORT=2, &
                     ASCII=1
  ......
END MODULE AJAX _2002_ PARAMETERS
```

当上述模块被定义之后，就可在需要这些参数的程序单元中用 USE 语句引入它们。例如：

```
PROGRAM EQUATION _ SOLVER
  USE AJAX _2002_ PARAMETERS
  REAL(KIND=QUAD)::A,B,C
  INTEGER(KIND=SHORT)::INDEX
  CHARACTER(LEN=20,KIND=ASCII)::NAME
  ……
  NAME=ASCII _"JOHN DOE"
  CALL CREATE _ ARRAY(A,100_ SHORT)
  ……
END PROGRAM EQUATION _ SOLVER
```

如果该程序中只需要 QUAD、SHORT 和 ASCII，则可将上面的 USE 语句改成：

```
USE AJAX _2002_PARAMETERS,ONLY:QUAD,&
                SHORT,ASCII
```

如果还要求把它们更名为 QUD、SHT 和 ASC，则可：

```
USE AJAX _2002_ PARAMETERS,ONLY:QUD=>QUAD,&
                SHT=>SHORT,ASC=>ASCII
```

来实现。当模块又包含 USE 语句时，采用 USE 语句来更名需特别小心，下面的例子说明这个问题。有两个模块：

```
MODULE J;REAL JX,JY,JZ;END MODULE
```

和

```
MODULE K
  USE J,ONLY:KX=>JX,KY=>JY
              !KX 和 KY 是模块 K 的局部名
  REAL KZ  !KZ 是模块 K 的局部名
  REAL JZ  !JZ 是模块 K 的局部名
END MODULE
```

于是：

```
PROGRAM RENAME
  USE J
```

```
USE K
    ! 模块 J 的实体 JX 可用名 JX 和 KX 访问
    ! 模块 J 的实体 JY 可用名 JY 和 KY 访问
    ! 模块 K 的实体 KZ 可用名 KZ 访问
    ! 模块 J 的实体 JZ 和模块 K 的实体 JZ 是不同的实体，
    ! 因此一定不能引用 JZ
END PROGRAM RENAME
```

第三节 过程接口

过程接口用来确定过程被调用的形式。它由过程的特性、过程名、各虚元的名和特性以及过程的类属标识符（可以省缺）组成，一般它们都被写在一个过程的开头部分。因为一个内部过程总是由同一程序单元中的一个语句来调用的，所以可认为编译程序在处理对该过程的调用时知道该内部过程的所有情况，特别是它当然也知道这些内部过程的接口。也就是说，它知道该过程是函数还是子程序、过程名、各虚元的名和特性、以及函数结果的特性（如果是函数的话）等情况。由于在这种情形中，过程接口对编译程序而言是明显给出的，故称显式接口。一个模块过程或者在本模块内被调用，或者在另一程序单元中的相应 USE 语句之后被调用，两种情形中编译程序在处理对该过程的调用时都明显地知道该过程的所有情况，因此其接口也是显式的，类似地，语句函数和所有内在过程的接口也都是显式的。

然而，当对一个外部过程或虚过程的调用进行编译时，一般并没有一种机制来立即得到该过程的代码，从而有关该过程的各种属性，所以我们说它们的接口是隐式的。在 Fortran 90之前的 Fortran 语言中，为了指明一个名是一个外部过程或虚过程的名，可用 EXTERNAL 语句，其形式为：

EXTERNAL 外部名表

要求它出现在任意 USE 语句或 IMPLICIT 语句之后，任意可执

行语句之前。可是,EXTERNAL 语句只说明外部名表中的各外部名是一个外部过程或虚过程的名,并没指明过程的接口,所以接口仍是隐式的。然而,在 Fortran 90中提供了一种明显地指明过程接口的机制,即所谓"接口块"。通过它可为一个外部过程或虚过程指明一个显式的接口,我们将在下面介绍这种机制。

综上所述,如果一个过程在一个作用域单位中可见,则它的过程接口在该作用域单位中或者是隐式的或者是显式的。一个内部过程、模块过程或内在过程的接口在这样一个作用域单位中总是显式的。一个语句函数的接口也总是显式的。一个外部过程或虚元过程的接口,如果已为它们提供了一个接口块,则在它自己之外的作用域单位中是显式的;否则如果没为它们提供接口块,则其接口仍是隐式的。此外,显然一个具有分开的结果名的递归子程序或递归函数的接口在定义它的辅程序中是显式的。

一、过程接口说明

过程的接口决定过程被调用的形式。一个内部过程、外部过程、模块过程或虚过程的接口用一个 FUNCTION 语句、SUBROUTINE 语句或 ENTRY 语句后跟关于该过程的虚元和函数结果的说明语句来指明。除了 ENTRY 语句一定不能出现在一个接口块中之外,这些说明语句可以出现在其过程定义中、一个接口块中、或同时出现在两者中。注意,内部过程是一定不能出现在一个接口块中。

过程接口块的一般形式为:

```
接口语句
  [接口体]…
  [模块过程语句]…
接口 END 语句
```

其中,接口语句具有形式:

```
INTERFACE[类属说明]
```

接口 END 语句具有形式:

```
END INTERFACE
```

接口体的形式为：

```
函数语句
  [说明部分]
函数 END 语句
```

或

```
子程序语句
  [说明部分]
子程序 END 语句
```

模块过程语句的形式为：

```
MODULE PROCEDURE 过程名表
```

类属说明的形式为：

```
类属名
```

或

```
OPERATOR(定义的运算)
```

或

```
ASSIGNMENT(=)
```

对此有四个限制条件：

(1)接口体中一定不能包含 ENTRY 语句、DATA 语句、FORMAT 语句、或语句函数语句。

(2)只有当在接口块中有一个类属说明，并且其"宿主"是一个模块或用 USE 语句使一个模块可见时，才允许有模块过程语句(MODULE PROCEDURE 语句)。其中的各过程名必须是其"宿主"中可见的模块过程的名。例如：

```
MODULE INTERVAL_ARITHMETIC
  TYPE INTERVAL
    REAL LOWER,UPPER
  END TYPE INTERVAL
  INTERFACE OPERATOR(+)
    MODULE PROCEDURE ADD_INTERVALS
  END INTERFACE
    ...
```

```
CONTAINS
  FUNCTION ADD_INTERVALS(A,B)
    TYPE(INTERVAL) ADD_INTERVALS,A,B
    ADD_INTERVALS % LOWER = A% LOWER + B % &
      LOWER
    ADD_INTERVALS % UPPER = A % UPPER + B % &
      UPPER
  END FUNCTION ADD_INTERVALS
    ...
END MODULE INTERVAL_ARITHMETIC
```

(3)接口块一定不能出现在 BLOCK DATA 程序单元中。

(4)一个辅程序中的接口块一定不能包含一个被该辅程序定义的过程的接口体。

其实,一个辅程序的接口体可认为就是将其可执行部分和内部辅程序去掉后余下的部分。然而可允许如下的改变:其虚元可更名;只需留下关于虚元和函数结果的说明和信息,其它无关的说明和信息(例如关于局部变量的说明)都可去掉;相同的信息允许用不同的语句组合来给出,例如一些说明语句可以重排次序。

把一个过程名放在 EXTERNAL 语句中或给它一个接口体,可以保证它作为一个外部过程或虚过程,而避免编译程序把它作为一个内在过程处理。因为 Fortran 90是允许一个处理系统增加外加的内在过程的。

例如,下面是一个没有类属说明的接口块:

```
INTERFACE
  SUBROUTINE EXT1(X,Y,Z)
    REAL DIMENSION (100,100):: X,Y,Z
      END SUBROUTINE EXT1
      SUBROUTINE EXT2(X,Z)
        REAL X
        COMPLEX (KIND=4) Z(2000)
      END SUBROUTINE EXT2
      FUNCTION EXT3(P,Q)
```

```
        LOGICAL EXT3
        INTEGER P (1000)
        LOGICAL Q (1000)
      END FUNCTION EXT3
    END INTERFACE
```

这个接口块为三个过程 EXT1、EXT2和 EXT3说明了三个显式接口。这些过程都将可使用关键词调用，例如：

```
EXT3 (Q=P _ MASK(N+1:N+1000),P=ACTUAL _ P)
```

引入过程接口块来为过程提供显式的调用接口比用 EXTERNAL 语句提供了更多的信息，它们都是在编译时作各种语法或语义的检查所必须的，否则这些检查只能采用生成目标的办法留待执行时动态地去检查。因此，用接口块提供过程的显式接口，不但在程序的可读性上得到了改善，而且对提高编译目标的质量方面也大有好处。

二、类属过程接口块

具有类属说明的过程接口块用来说明块中的过程是一些类属过程，并提供各过程的类属接口形式。根据上述的定义，过程接口块中类属说明分为三种情形。

1. 类属标识

当类属说明是一个类属标识时，例如：

```
INTERFACE SWITCH
  SUBROUTINE INT _ SWITCH(X,Y)
    INTEGER,INTENT (INOUT):: X,Y
  END SUBROUTINE INT _ SWITCH
  SUBROUTINE REAL _ SWITCH(X,Y)
    REAL,INTENT (INOUT) :: X,Y
  END SUBROUTINE REAL _ SWITCH
  SUBROUTINE COMPLEX _ SWITCH (X,Y)
    COMPLEX,INTENT (INOUT) :: X,Y
  END SUBROUTINE COMPLEX _ SWITCH
```

```
    END INTERFACE
```

为三个子程序 INT _ SWITCH、REAL _ SWITCH 和 COMPLEX _ SWITCH 说明了一个类属接口。接口块中的类属标识指明所有块中过程的类属名，因此它们中的任意一个除了可用各自的特定名调用之外，都还可用其类属名 SWITCH 来调用。例如，对 INT _ SWITCH 而言，即可用语句：

```
    CALL INT _ SWITCH(MAX _ VAL,LOC _ VAL)
```

调用，也可用语句：

```
    CALL SWITCH (MAX _ VAL,LOC _ VAL)
```

来调用，只要保证 MAX _ VAL 和 LOC _ VAL 是整型即可。

如果接口块的"宿主"是一个模块或者通过 USE 语句可见到一个模块，则必须用其中的 MODULE PROCEDURE 语句列出在该模块中定义的，或通过 USE 语句可见的模块过程。例如：

```
  MODULE INTERVAL _ ARITHMETIC _ SWITCH _ MATRIX &
      _ MODULE
    TYPE INTERVAL
      REAL LOWER,UPPER
    END TYPE INTERVAL
    USE MATRIX _ MODULE
    INTERFACE
      MODULE PROCEDURE MATRIX _ SUM,MATRIX &
            _ PROD ARRAY _ TO _ MATRIX,PRINT _ MATRIX
    END INTERFACE
    INTERFACE OPERATOR (+)
      MODULE PROCEDURE ADD _ INTERVALS
    END INTERFACE
      ...
    INTERFACE SWITCH
      MODULE PROCEDURE INT _ SWTCH ,REAL _ SWITCH,&
             COMPLEX _ SWITCH
    END INTERFACE
      CONTAINS
```

```
    FUNCTION ADD _ INTERVALS(A,B)
      TYPE(INTERVAL) ADD _ INTERVALS,A,B
      ADD _ INTERVALS % LOWER = A % LOWER + &
        B % LOWER
      ADD _ INTERVALS % UPPER = A % UPPER + &
        B % UPPER
    END FUNCTION ADD _ INTERVALS
      ...
    SUBROUTINE INT _ SWITCH(X,Y)
      INTEGER,INTENT (INOUT) :: X,Y
    END SUBROUTINE INT _ SWITCH
    SUBROUTINE REAL _ SWITCH(X,Y)
      REAL,INTENT (INOUT)::X,Y
    END SUBROUTINE REAL _ SWITCH
    SUBROUTINE COMPLEX _ SWITCH (X,Y)
      COMPLEX,INTENT (INOUT):: X,Y
    END SUBROUTINE COMPLEX _ SWITCH
END MODULE INTERVAL _ ARITHMETIC _ SWTCH
```

其中 MATRIX _ MODULE 是第一节中所定义的那个模块。

2. OPERATOR 类属说明

如果类属说明是 OPERATOR(定义的运算),则接口块可用来定义运算,而且允许运算是可重载的。这时块中说明的过程必须都是函数,而且它们不能没有变元,也不能有两个以上的变元。其虚元必须是非任选的且具有属性 INTENT(IN)。函数结果必须没有僭取的字符长度。如果 OPERATOR 指明的是一个内在运算,则相应的函数的变元的个数必须与该内在运算的操作数个数相同。

一个一元函数定义一个一元运算,函数的变元就是其操作数。一个二元函数定义一个二元运算,函数的第一个变元就是其左操作数,函数的第二个变元就是其右操作数。下面是一个具有 OPERATOR 类属说明的接口块:

```
INTERFACE OPERATOR( * )
```

```
    FUNCTION BOOLEAN _ AND (B1,B2)
      LOGICAL,INTENT(IN)::B1(:),B2(SIZE(B1))
      LOGICAL :: BOOLEAN _ AND (SIZE(B1))
    END FUNCTION BOOLEAN _ AND
  END INTERFACE
```

于是就可用：

```
    SENSOR (1:N) * ACTION (1:N)
```

来运用运算 *，这等价于函数调用：

```
    BOOLEAN _ AND(SENSOR(1:N),ACTION(1:N)
                 ! SENSOR 和 ACTION 为
                 ! 逻辑型
```

又如，

```
  INTERFACE OPERATOR (+)
    FUNCTION MATRIX _ SUM (MATRIX _1,MATRIX _2) &
                 RESULT (MATRIX _ SUM _ RESULT)
      TYPE(MATRIX),INTENT(IN):: MATRIX _1,MATRIX _2
      TYPE(MATRIX)::MATRIX _ SUM _ RESULT
    END FUNCTION MATRIX _ SUM
    FUNCTION MATRIX _ SCALAR _ SUM (MATRIX _1,X _2)&
                 RESULT (MATRIX _ SCALAR _ SUM&
          _ RESULT)
      TYPE(MATRIX),INTENT(IN)::MATRIX _1
      TYPE(MATRIX)::MATRIX _ SCALAR _ SUM _ RESULT
      REAL,INTENT(IN):: X _2
    END FUNCTION MATRIX _ SCALAR _ SUM
    FUNCTION SCALAR _ MATRIX _ SUM (X _1,MATRIX _2)&
                 RESULT (SCALAR _ MATRIX _ SUM&
          _ RESULT)
      TYPE (MATRIX),INTENT(IN)::MATRIX _2
      TYPE(MATRIX)::SCALAR _ MATRIX _ SUM _ RESULT
      REAL,INTENT(IN) ::X _1
    END FUNCTION SCALAR _ MATRIX _ SUM
    FUNCTION MATRIX _ IDENT(MATRIX _1)&
```

```
                    RESULT(MATRIX_IDENT_RESULT)
        TYPE (MATRIX),INTENT(IN)::MATRIX_1
        TYPE(MATRIX)::MATRIX_IDENT_RESULT
      END FUNCTION MATRIX_IDENT
    END INTERFACE
```

上述接口块说明了四个函数，分别做矩阵加法、矩阵加实数、实数加矩阵和恒等矩阵运算，因它们被包在一个接口块中，故都可用运算＋来引用它们。例如，下面的程序就是一个合法的测试程序：

```
    PROGRAM OPERATOR_TEST
      IMPLICIT NONE
      USE MATRIX_MODULE
      TYPE(MATRIX)::M1,M2
      REAL,DIMENSION (3,3)::A1,A2
      REAL,X
        ! 先建立两个数组用来给矩阵 M1和 M2赋值
      DO I=1,3
        A1(I,:)=I
          A2(:,I)=I
        END DO
        M1=ARRAY_TO_MATRIX(A1)
        M2=ARRAY_TO_MATRIX(A2)
        CALL PRINT_MATRIX (M1+M2)
        CALL PRINT_MATRIX (M1+X)
        CALL PRINT_MATRIX (X+M2)
        CALL PRINT_MATRIX (+M1)
      END PROGRAM OPERATOR_TEST
```

同样地，可以定义各种运算一和 * 等。用这种方法可由用户自己扩充各自需要的运算。

3. ASSIGNMENT 类属说明

我们还可用接口块来给赋值号(=)定义不同的语义，这就是所谓 ASSIGNMENT 类属说明。例如：

```
INTERFACE ASSIGNMENT (=)
  SUBROUTINE BIT _ TO _ NUMERIC (N,B)
    INTEGER,INTENT(OUT) :: N
    LOGICAL,INTENT(IN)::B(:)
  END SUBROUTINE BIT _ TO _ NUMERIC
  SUBROUTINE CHAR _ TO _ STRING(S,C)
    USE STRING _ MODULE !模块中包含类型 STRING 的定义
    TYPE (STRING),INTENT(OUT):: S!一个变长的串
    CHARACTER(*),INTENT(IN)::C
  END SUBROUTINE CHAR _ TO _ STRING
END INTERFACE
```

接口块中定义了两个不同含义的赋值运算，以后在程序中都可用＝号来使用它们。例如，合法的赋值：

```
COUNT=SENSOR(J:K)
  ! 等价于 CALL BIT _ TO _ MATRIX (COUNT,(SENSOR
  !(J:K)))
NOTE='12AB'
  !等价于 CALL CHAR _ TO _ STRING(NOTE,('12AB'))
```

在此值得指出，在具有 ASSIGNMENT 选择的接口块中，包含的过程都必须是子程序，而且都只能正好有两个变元，第一个变元具有属性 INTENT(OUT)或 INTENT(INOUT)，第二个变元具有属性 INTENT(IN)，并且都不能是任选的。当作为赋值运算用时，第一个变元为被赋值的对象，第二个变元对应被括号括起来的赋值号的右边部分。可见，ASSIGEMENT 类属说明既可用来扩充赋值的语义，也可用来重定义原赋值的含义(当等号两边是相同的导出类型时)。

关于过程的显式接口我们可做如下的总结：如果下列的任意一条成立，则一个过程必须要有一个显式接口：

(1)对该过程的引用中变元附带关键词时。

(2)对该过程的引用是作为用户定义的赋值使用时(只对子程序而言)。

(3)在表达式中对该过程的引用是作为用户定义的运算使用时(只对函数而言)。

(4)对该过程的引用是被其类属名引用时。

(5)该过程具有一个任选虚元时。

(6)该过程的结果为数组时(只对函数而言)。

(7)该过程有一个虚元是僭取形数组、指针或目标时。

(8)该过程的结果为长度类型参数值既不是僭取的也不是常量时(只对函数而言)。

(9)该过程的结果为一个指针时(只对函数而言)。

要求为这些过程提供显式接口的目的是为了给编译程序提供足够的编译信息,使得编译能够进行,或达到更高的编译质量。

如果在一个作用域单位中,一个函数的接口是隐式的,则该函数的结果的类型和类型参数由该函数名的隐式的或显式的类型说明确定。从接口是隐式的作用域调用过程必须使相应实元的类型、类型参数和形状与该过程的虚元的特性一致。

第四节　模块的应用

模块的引入给程序设计带来了很多方便,可有各种应用,下面我们来举一些例子说明。

一、全 局 数 据

前面已经讲到,可把一些将在整个可执行程序中都要用到的全局数据封装在一个模块中,即把它们放在一个模块中统一说明,在需要这些数据的程序单元内用 USE 语句导出它们即可。例如:

```
MODULE DATA_MODULE
  SAVE
  REAL A(10),B,C(20,20)
  INTEGER,PARAMETER :: I=0
  INTEGER,PARAMETER :: J=10
```

```
    COMPLEX D(J,J)
  END MODULE
```
于是,为了可访问全部数据对象,可用
```
      USE DATA _MODULE
```
如果为了只用其中的 A、B 和 D,则可用
```
      USE DATA _MODULE,ONLY:A,B,D
```
为了避免名字冲突,还可以适当更名,例如可用
```
      USE DATA _MODULE,AMODULE=>A,DMODULE=>D
```
将 A 和 D 分别更名为 AMODULE 和 DMODULE。

其实,上节中模块 AJAX _2002_ PARAMETERS 也是这样一种全局数据模块。

二、全 局 过 程

模块还可把一些将在整个可执行程序中都要用到的全局过程封装在一个模块中,即只需简单地把这些过程放在一个模块中统一说明,而在需要这些过程的程序单元内用 USE 语句导出它们即可。例如我们可以把 Bessel 函数、Gamma 函数等特殊函数放在一个模块中构造一个计算特殊函数的程序包。因为模块也是一种程序单元,可有其自己的说明部分,所以,还不妨将各模块过程公用的实体抽出来,统一放在模块的说明部分说明。这时,模块中说明的实体可通过宿主结合而被其中的模块过程所引用。例如,上节中的模块 MATRIX _ MODULE 就是这样一个例子。一当定义好模块 MATRIX _ MODULE 之后,我们就可用 USE 语句来导出做各种矩阵运算的过程了。今用下面的例子来说明:
```
  PROGRAM MATRIX _ TEST
    IMPLICIT NONE
    USE MATRIX _MODULE
    TYPE(MATRIX)::M1,M2
    REAL,DIMENSION(3,3)::A1,A2
      !下面建立两个测试矩阵 A1和 A2,用来给矩阵 M1和 M2赋值
    DO I=1,3
```

```
    A1(I,:)=I
    A2(:,I)=I
  END DO
  M1=ARRAY_TO_MATRIX(A1)
  M2=ARRAY_TO_MATRIX(A2)
  CALL PRINT_MATRIX(MATRIX_SUM(M1,M2))
  CALL PRINT_MATRIX(MATRIX_PROD(M1,M2))
END PROGRAM MATRIX_TEST
```

其中,通过 USE 语句引入了 MATRIX_MODULE 中的四个对矩阵进行操作的过程,于是在程序体部分就可引用它们了。而且,模块 MATRIX_MODULE 还可被程序中的任何其它外部辅程序来引用,所以其中的模块过程其实就是一些全局过程。

三、公用的导出类型

模块还可用来封装一些导出类型,供其它程序单元来公用。例如:

```
MODULE SPARSE
  TYPE NONZERO
    REAL A
    INTEGER I,J
  END TYPE
END MODULE
```

定义了一个由一个实数和两个整数构成的数据类型,用以描述一个稀疏矩阵的非零元素。其中的一个实数表示元素的值,两个整数表示元素的下标。于是,导出类型 NONZERO 就可被其它单元用 USE 语句来共用。

四、全局可分配数组

在许多程序中需要一些全局公用的可分配数组,它们的大小在程序执行之前是不知道的,这时也可用模块来实现。例如,

```
PROGRAM GLOBAL_WORK
```

```
  CALL CONFIGURE_ARRAYS  ! 实施适当的数组分配
  CALL COMPUTE           ! 用分配好的数组做计算
END PROGRAM GLOBAL_WORK
MODULE WORK_ARRAYS
  INTEGER N
  REAL,ALLOCATABLE,SAVE :: A(:),B(:,:),C(:,:,:)
END MODULE WORK_ARRAYS
SUBROUTINE CONFIGURE_ARRAYS
  USE WORK_ARRAYS
  READ (*,*) N
  ALLOCATE (A(N),B(N,N),C(N,N,2*N))
END SUBROUTINE CONFIGURE_ARRAYS
SUBROUTINE COMPUTE
  USE WORK_ARRAYS
   ...                   !对数组 A,B,C 的计算
END SUBROUTINE COMPUTE
```

这是很典型的一种应用。这时,所有需要使用公用工作区 WORK_ARRAYS 的辅程序中都将包含语句:

```
      USE WORK_ARRAYS
```

从而达到共享存储和相互交换信息的目的。

五、抽象数据类型

程序员可以用模块来任意定义自己的抽象数据类型,为此只需在模块中首先定义一个导出数据类型,然后把表示可在这种类型的值上进行的运算的过程紧跟着放在后面即可。例如:

```
MODULE INTEGER_SETS
  INTEGER,PARAMETER :: MAX_SET_CARD=200
  TYPE SET              !定义数据类型 SET
    PRIVATE
    INTEGER CARD
    INTEGER ELEMENT (MAX_SET_CARD)
  END TYPE SET
```

```
INTERFACE OPERATOR(.IN.)
  MODULE PROCEDURE ELEMENT
END INTERFACE
INTERFACE OPERATOR(<=)
  MODULE PROCEDURE SUBSET
END INTERFACE
INTERFACE OPERATOR(+)
  MODULE PROCEDURE UNION
END INTERFACE
INTERFACE OPERATOR(-)
  MODULE PROCEDURE DIFFERENCE
END INTERFACE
INTERFACE OPERATOR(*)
  MODULE PROCEDURE INTERSECTION
END INTERFACE
CONTAINS
INTEGER FUNCTION CARDINALITY (A) !回送 A 的基数
  TYPE (SET)A
  CARDINALITY = A%CARD
END FUNCTION CARDINALITY
LOGICAL FUNCTION ELEMENT (X,A)!确定 X 是否在 A 中
  INTEGER X
  TYPE (SET)A
  ELEMENT =ANY(A % ELEMENT(1:A % CARD).EQ.X)
END FUNCTION ELEMENT
FUNCTION UNION (A,B)!集合的并
  TYPE(SET) A,B,UNION
  INTEGER J
  UNION =A
  DO J=1,B %CARD
    IF(.NOT.(B %ELEMENT (J).IN.A))THEN
        IF(UNION % CARD <MAX_SET_CARD)THEN
            UNION % CARD =UNION % CARD+1
```

```
            UNION % ELEMENT(UNION %CARD)=&
          B% ELEMENT (J)
           ELSE
            !超过集合允许的大小,…
           END IF
         END IF
     END DO
   END FUNCTION UNION
   FUNCTION DIFFERENCE (A,B)!集合的差
     TYPE(SET)A,B,DIFFERENCE
     INTEGER J,X
     DIFFERENCE % CARD = 0!空集
     DO J=1,A%CARD
       X= A % ELEMENT (J)
       IF (.NOT.(X.IN.B)) DIFFERENCE=DIFFERENCE &
                              +SET(1,X)
     END DO
   END FUNCTION DIFFERENCE
   FUNCTION INTERSECTION(A,B)!集合的交
     TYPE(SET) A,B,INTERSECTION
     INTERSECTION =A-(A-B)
   END FUNCTION INTERSECTION
   FUNCTION SUBSET(A,B)!确定 A 是否是 B 的子集
     TYPE(SET)A,B
     INTEGER I
     SUBSET = A% CARD≤ B % CARD
     IF(.NOT. SUBSET)RETURN ! 为了运行更有效而设的控制
     DO I=1, A% CARD
       SUBSET =SUBSET .AND.(A%ELEMENT (I) .IN. B)
   END DO
   END FUNCTION SUBSET
   TYPE(SET)FUNCTION SETF(V)
         ! 向量转换成没有重复元素的集合
```

```
        TYPE (SET) V(:)
        INTEGER J
        SETF % CARD =0
        DO J=1,SIZE(V)
          IF(.NOT.(V(J).IN.SETF))THEN
            IF(SETF % CARD <MAX _ SET _ CARD)THEN
                SETF % CARD = SETF % CARD +1
                SETF % ELEMENT (SETF % CARD) =V(J)
          ELSE
                !超过集合允许的大小,…
          END IF
        END IF
      END DO
    END FUNCTION SETF
    FUNCTION VECTOR(A)!转换集合为一个值递增的向量
      TYPE (SET)A
      INTEGER,POINTER :: VECTOR(:)
      INTEGER I,J,K
      ALLOCATE(VECTOR(A % CARD))
      VECTOR = A % ELEMENT (1: A & CARD)
      DO I=1,A % CARD-1
            DO J=I+1,A % CARD
              IF(VECTOR(I)>VECTOR(J)) THEN
                K=VECTOR(J):VECTOR(J)=VECTOR(I);&
                    VECTOR(I)=K
              END IF
            END DO
          END DO
        END FUNCTION VECTOR
      END MODULE INTEGER _ SETS
```

上述模块首先定义一个由200个整数构成的整数集体的类型SET;然后给出说明五种集合运算的五个显式接口块(.IN.,<=,+,-,*);接着具体定义实现这些运算的模块过程;所以这可

看成已定义了一种有限(200个)整数集合抽象数据类型。

采用上述方法可以方便地来组织程序包,把一些类似的或用于同一领域的过程封装在一起,供全局范围内调用。

小　结

本节我们介绍了 Fortran 90中新增加的两个语言成份,模块和过程接口。模块是一种程序单元,它为过程之间提供了另一种方便有效的共享常量、变量、类型定义以及过程的途径。在过去的 Fortran 语言中只有用虚元和实元的结合一种较有效的手段能来实现在过程之间共享常量、变量、以及过程的传递。而 COMMON 语句和 EQUIVALENCE 语句如今已认为不是一种好的共享数据的办法,虽然为了保持向上兼容的目的保留了它们,但作为一种好的编程序风格并不推荐使用它们。本节中用举例的形式介绍了多种关于模块单元和 USE 语句的应用实例,这对了解如何运用模块单元来编程序是很有帮助的。

过程接口块能为一些没有显式调用接口的过程提供必要的属性和信息(即所谓显式接口),使得在要调用它们时知道如何来引用它们。此外,显式的调用接口中所包含的信息都是在编译时作各种语法或语义的检查所必须的,否则这些检查只能采用生成目标的办法留待执行时动态地去检查,那样就势必降低运行效率。因此,用接口块提供过程的显式接口,不但在程序的可读性和编程序风格上得到了改善,而且对提高编译目标的质量方面也大有好处。

习　题

1. 用模块单元定义一个做复数四则运算的程序包。

2. 用过程接口块定义一个类属运算+,用它可做整数、实数、复数和布尔量的加法(或逻辑加)。

3. 试述用模块来提供公用数据比用 COMMON 语句来提供

的优点。

4. 把由 $M \times N$ 的矩阵(元素为整数)构成的集体定义为一个抽象数据类型,在其上能做矩阵的加法、乘法、转置、元素求和、以及求元素的极大和极小等运算。

附　　录

本书有四个附录，其前三个附录都取自我国国家标准草案：Fortran 90。其中的附录一是草案附录 D 中的 D.1 语法规则和约束；附录二是草案附录 F 索引；附录三是草案附录 B 缩减的功能。附录四是关于软盘说明。

附录一　　语法规则

本附录是全部语法规则和约束的一个抽取，按照它们出现在国家标准草案中的次序来排列。

除了以-name 结束的之外，不定义的非终结符只有 letter、digit、special-character 和 rep-char。假定下列的规则，字母“xyz”表示任何合法的语法类短语：

xyz-list　　is *xyz*〔,*xyz*〕…

xyz-name　　is *name*

scalar-xyz　　is *xyz*

约束：scalar-xyz 必须是标量。

语法规则和约束

下面每一段所包含的语法规则和约束来自标准草案中的某一章；即：第 1 条包含了第 1 章中的规则和约束，第 2 条包含了来自第 2 章的规则和约束，等等。注意，第 1、13 和 14 章没包含语法规则。

1. 概述

没有语法规则和约束。

2. Fortran 术语和概念

R201　*executable-program*　　　is *program-unit*

[*program-unit*]...

R202 *program-unit* is *main-program*
or *external-subprogram*
or *module*
or *block-data*

R1101 *main-program* is [*program-stmt*]
[*specification-part*]
[*execution-part*]
[*internal-subprogram-part*]
end-program-stmt

R203 *external-subprogram* is *function-subprogram*
or *subroutine-subprogram*

R1215 *function-subprogram* is *function-stmt*
[*specification-part*]
[*execution-part*]
[*internal-subprogram-part*]
end-function-stmt

R1219 *subroutine-subprogram* is *subroutine-stmt*
[*specification-part*]
[*execution-part*]
[*internal-subprogram-part*]
end-subroutine-stmt

R1104 *module* is *module-stmt*
[*specification-part*]
[*module-subprogram-part*]
end-module-stmt

R1110 *block-data* is *block-data-stmt*

[*specification-part*]
end-block-data-stmt

R204 *specification-part*	is [*use-stmt*]···
	[*implicit-part*]
	[*declaration-construct*]···
R205 *implicit-part*	is [*implicit-part-stmt*]···
	implicit-stmt
R206 *implicit-part-stmt*	is *implicit-stmt*
	or *parameter-stmt*
	or *format-stmt*
	or *entry-stmt*
R207 *declation-construct*	is *derived-type-def*
	or *interface-block*
	or *type-declaration-stmt*
	or *specification-stmt*
	or *parameter-stmt*
	or *format-stmt*
	or *entry-stmt*
	or *stmt-function-stmt*
R208 *execution-part*	is *executable-construct*
	[*execution-part-construct*]···
R209 *execution-part-construct*	is *executable-construct*
	or *format-stmt*
	or *data-stmt*
	or *entry-stmt*
R210 *internal-subprogram-part*	is *contains-stmt*

internal-subprogram
[*internal-subprogram*]...

R211 *internal-subprogram* is *function-subprogram*
or *subroutine-subprogram*

R212 *module-subprogram-part* is *contains-stmt*
module-subprogram
[*module-subprogram*]...

R213 *module-subprogram* is *function-subprogram*
or *subroutine-subprogram*

R214 *specification-stmt* is *access-stmt*
or *allocatable-stmt*
or *common-stmt*
or *data-stmt*
or *dimension-stmt*
or *equivalence-stmt*
or *external-stmt*
or *intent-stmt*
or *intrinsic-stmt*
or *namelist-stmt*
or *optional-stmt*
or *pointer-stmt*
or *save-stmt*
or *target-stmt*

R215 *executable-construct* is *action-stmt*
or *case-construct*
or *do-construct*
or *if-construct*
or *where-construct*

R216 *action-stmt* is *allocate-stmt*
or *assignment-stmt*
or *backspace-stmt*
or *call-stmt*
or *close-stmt*
or *computed-goto-stmt*
or *continue-stmt*
or *cycle-stmt*
or *deallocate-stmt*
or *endfile-stmt*
or *end-function-stmt*
or *end-program-stmt*
or *end-subroutine-stmt*
or *exit-stmt*
or *goto-stmt*
or *if-stmt*
or *inquire-stmt*
or *nullify-stmt*
or *open-stmt*
or *pointer-assignment-stmt*
or *print-stmt*
or *read-stmt*
or *return-stmt*
or *rewind-stmt*
or *stop-stmt*
or *where-stmt*
or *write-stmt*
or *arithmetic-if-stmt*
or *assign-stmt*
or *assigned-goto-stmt*
or *pause-stmt*

约束：*execution-part* 必须不包含 *end-function-stmt*、*end-program-stmt* 或 *end-subroutine-stmt*。

3. 字符、词法记号和源程序形式

R301 *character*	is *alphanumeric-character*
	or *special-character*
R302 *alphanumeric-character*	is *letter*
	or *digit*
	or *underscore*
R303 *underscore*	is _
R304 *name*	is *letter*〔 *alphanumeric-character*〕…

约束:*name* 的最大长度是 31 个字符。

R305 *constant*	is *literal-constant*
	or *named-constant*
R306 *literal-constant*	is *int-literal-constant*
	or *real-literal-constant*
	or *complex-literal-constant*
	or *logical-literal-constant*
	or *char-literal-constant*
	or *boz-literal-constant*
R307 *named-constant*	is *name*
R308 *int-constant*	is *constant*

约束:*int-constant* 必须是整型的。

R309 *char-constant*	is *constant*

约束:*char-constant* 必须是字符型的。

R310 *intrinsic-operator*	is *power-op*
	or *mult-op*
	or *add-op*
	or *concat-op*
	or *rel-op*

or *not-op*
or *and-op*
or *or-op*
or *equiv-op*

R708 *power-op* is * *

R709 *mult-op* is *
or /

R710 *add-op* is +
or -

R712 *concat-op* is //

R714 *rel-op* is .EQ.
or .NE.
or .LT.
or .LE.
or .GT.
or .GE.
or ==
or /=
or <
or <=
or >
or >=

R719 *not-op* is .NOT.

R720 *and-op* is .AND.

R721 *or-op* is .OR.

R722 *equiv-op* is .EQV.
or .NEQV.

R311 *defined-operator* is *defined-unary-op*
or *defined-binary-op*
or *extended-intrinsic-op*

R704 *defined-unary-op* is .*letter*〔*letter*〕….

R724 *defined-binary-op* is .*letter*〔*letter*〕….

R312 *extended-intrinsic-op* is *intrinsic-operator*

约束：*defined-unary-op* 和 *defined-binary-op* 必须不得包含 31 个以上字母并且必须不和任何 *intrinsic-operator* 或 *logical-literal-constant* 相同。

R313 *label* is *digit*〔*digit*〔*digit*〔*digit* ■
■〔*digit*〕〕〕〕

约束：在 *label* 中至少有一个数字必须是非零的。

4. 内在的和导出的数据类型

R401 *signed-digit-string* is 〔*sign*〕*digit-string*

R402 *digit-string* is *digit*〔*digit*〕…

R403 *signed-int-literal-constant* is 〔*sign*〕*int-literal-constant*

R404 *int-literal-constant* is *digit-string*〔_*kind-param*〕

R405 *kind-param* is *digit-string*
or *scalar-int-constant-name*

R406 *sign* is +
or -

约束：*kind-param* 的值必须是非负的。

约束：*kind-param* 的值必须说明存在于处理系统的表示方法。

R407 *boz-literal-constant* is *binary-constant*
or *octal-constant*

or *hex-constant*

约束：*boz-literal-constant* 只能出现在 DATA 语句中。

R408 *binary-constant* is B′ *digit*〔 *digit*〕…′
or B″ *digit*〔 *digit*〕…″

约束：*digit* 必须具有值 0 或 1 之一。

R409 *octal-constant* is O′ *digit*〔 *digit*〕…′
or O″ *digit*〔 *digit*〕…″

约束：*digit* 必须具有值 0 到 7 之一。

R410 *hex-constant* is Z′ *hex-digit*〔 *hex-digit*〕…′
or Z″ *hex-digit*〔 *hex-digit*〕…″

R411 *hex-digit* is *digit*
or A
or B
or C
or D
or E
or F

R412 *signed-real-literal-constant* is 〔 *sign*〕 *real-literal-constant*

R413 *real-literal-constant* is *significand*〔 *exponent-letter*■
■*exponent*〕〔_ *kind-param*〕
or *digit-string exponent-letter*■
■*exponent* 〔_ *kind-param*〕

R414 *significand* is *digit-string*. 〔 *digit-string*〕
or. *digit-string*

R415 *exponent-letter* is E
or D

R416 *exponent* is *signed-digit-string*

约束：若 *kind-param* 和 *exponent-letter* 都出现，则 *exponent-letter* 必须是 E。

约束：*kind-param* 的值必须说明存在于处理系统的近似方法。

R417 *complex-literal-constant* is (*real-part*, *imag-part*)

R418 *real-part* is *signed-int-literal-constant*
or *signed-real-literal-constant*

R419 *imag-part* is *signed-int-literal-constant*

or *signed-real-literal-constant*

R420 *char-literal-constant* is 〔*kind-param* _〕'〔*rep-char*〕…'

or 〔*kind-param* _〕"〔*rep-char*〕…"

约束:*kind-param* 的值必须说明存在于处理系统的表示方法。

R421 *logical-literal-constant* is .TRUE. 〔_ *kind-param*〕

or.FALSE.〔_ *kind-param*〕

约束:*kind-param* 的值必须说明存在于处理系统的表示方法。

R422 *derived-type-def* is *derived-type-stmt*

〔*private-seqnence-stmt*〕…

component-def-stmt

〔*component-def-stmt*〕…

end-type-stmt

R423 *private-seqnence-stmt* is PRIVATE

or SEQUENCE

R424 *derived-type-stmt* is TYPE〔〔,*access-spec*〕::〕■

■ *type-name*

约束:在给定的 *derived-type-del* 中,同样的 *private-seqnence-stmt* 禁止出现多次。

约束:若 SEQUENCE 出现,则在成分定义中说明的所有导出类型必须是顺序类型。

约束:仅当类型定义是在模块的说明部分中时,在定义中的 *access-spec* 或 PRIVATE 语句是允许的。

约束:若导出类型的一个成分是声明为私用的类型,则或导出类型定义必须包含 PRIVATE 语句,或导出类型必须是私用的。

约束:导出类型的 *type-name* 禁止与任何内在类型的名字相同,也禁止与任何其它可访问的导出类型的 *type-name* 相同。

R425 *and-type-stmt* is END TYPE〔*type-name*〕

约束:若 END TYPE 后跟 *type-name*,则该 *type-name* 必须与在相应 *derived-type-stmt* 中的 *type-name* 相同。

R426 *component-def-stmt* is *type-spec*〔〔, *component-attr*■

■*-spec-list*〕::〕■

■*component-decl-list*

R427 *component-attr-spec* is POINTER

or DIMENSION (*component-array*■

■ *-spec*)

约束:在给定的 *component-def-stmt* 中,*component-attr-spec* 不得出现多次。

约束:若对一成分没说明 POINTER 属性,则在 *component-def-stmt* 中 *type-spec* 必须说明为内在类型或先前已定义的导出类型。

约束:若对一成分没说明 POINTER 属性,则在 *component-def-stmt* 中 *type-spec* 必须说明为内在类型或包括正在定义类型在内的任何可访问的导出类型。

R428 *component-array-spec* is *explicit-shape-spec-list*

or *deferred-shape-spec-list*

R429 *component-decl* is *component-name*[(*component*■

■ *-arry-spec*)][* *char-length*]

约束:若未说明 POINTER 属性,则每个 *component-array-spec* 必须是 *explicit-shape-spec-list*。

约束:若说明了 POINTER 属性,则每个 *component-array-spec* 必须是 *deferred-shape-spec-list*。

约束:仅当说明的类型是字符型时,才允许有 * *char-length* 可选项。

约束:在 *component-decl* 中 *char-length* 必须是常量说明表达式。

约束:在 *explicit-shape-spec*(见 R428)中每个界必须是常量说明表达式。

R430 *structure-constructor* is *type-name*(*expr-list*)

R431 *array-constructor* is (/ *ac-value-list*/)

R432 *ac-value* is *expr*

or *ac-implied-do*

R433 *ac-implied-do* is (*ac-value-list*,*ac-implied* ■

■ *-do-control*)

R434 *ac-implied-do-control* is *ac-do-variable*=*scalar-int* ■

■ *-expr*,*scalar-int-expr* ■

■ [, *scalar-int-expr*]

R435 *ac-do-variable* is *scalar-int-variable*

约束:*ac-do-variable* 必须是有名变量。

约束:在 *array-constructor* 中的每个 *ac-value* 表达式必须有相同的类型

和类型参数。

5. 数据对象的声明与说明

R501 *type-declaration-stmt*	is *type-spec* [[,*attr-spec*]… ■
	■ ::]*entity-decl-list*
R502 *type-spec*	is INTEGER[*kind-selector*]
	or REAL[*kind-selector*]
	or DOUBLE PRECISION
	or COMPLEX[*kind-selector*]
	or CHARACTER[*char-selector*]
	or LOGICAL[*kind-selector*]
	or TYPE(*type-name*)
R503 *attr-spec*	is PARAMETER
	or *access-spec*
	or ALLOCATABLE
	or DIMENSION(*array-spec*)
	or EXTERNAL
	or INTENT(*intent-spec*)
	or INTRINSIC
	or OPTIONAL
	or POINTER
	or SAVE
	or TARGET
R504 *entity-decl*	is *object-name* [(*array-spec*)]■
	■[* *char-length*] ■
	■[=*initialization-expr*]
	or *function-name*[(*array-spec*)]■
	■[* *char-length*]
R505 *kind-selector*	is ([KIND=*scalar-*■
	■ *int-initialization-expr*)

约束：在一给定的 *type-declaration-stmt* 中同样的 *attr-spec* 不得出现多次。

约束：*function-name* 必须是外部函数名、内在函数名、函数虚拟过程名或语句函数名。

约束：若语句包含 PARAMETER 属性，则＝*initialization-expr* 必须出现。

约束：若＝ *initialization-expr* 出现，则双冒号分隔符必须出现在 *entity-decl-list* 之前。

约束：若 *object-name* 是虚元、函数结果，除了在块数据程序单元中类型声明之外的有名公用块中的对象、空白公用块中的对象、可分配数组、指针、外部名、内在名或自动对象，则＝ *initialization-expr* 不得出现。

约束：仅当说明的类型是字符型时，才允许出现 * *char-length* 可选项。

约束：仅当声明数组，且数组既不是虚元也不是函数结果时，才可使用 ALLOCATABLE 属性。

约束：声明的数组具有 POINTER 或 ALLOCATABLE 属性时，必须说明具有 *array-spec*，且是 *deferred-shape-spec-list*。

约束：对于不具有 POINTER 属性的 *function-name*，其 *array-spec* 必须是 *explicit-shape-spec-list*。

约束：对于具有 POINTER 属性的 *function-name*，其 *array-spec* 必须是 *deferred-shape-spec-list*。

约束：若说明了 POINTER 属性，则禁止说明 TARGET、INTENT、EXTERNAL 或 INTRINSIC 属性。

约束：若说明了 TARGET 属性，则禁止说明 POINTER、EXTERNAL、INTRINSIC 或 PARAMETER 属性。

约束：PARAMETER 属性禁止说明于虚元、指针、可分配数组、函数或在公用块中的对象。

约束：INTENT 和 OPTIONAL 属性只能说明于虚元。

约束：若实体的类型具有 PRIVATE 属性，则它不得具有 PUBLIC 属性。

约束：对于在公用块中的对象、虚元、过程、函数结果或自动数据对象禁止说明 SAVE 属性。

约束：若实体具有 INTRINSIC 属性，则它不得具有 EXTERNAL 属性。

约束：在 *type-declaration-stmt* 中的实体，除非该实体是函数，不得说明具有 EXTERNAL 或 INTRINSIC 属性。

约束：数组不得既有 ALLOCATABLE 属性又有 POINTER 属性。

约束：在作用域单位中，一实体不得显式地多次给定一属性。

约束：*scalar-int-initialization-expr* 的值必须是非负的，且必须说明存在

于处理系统的表示方法。

R506 *char-selector* is *length-selector*

or (LEN= *type-param-value*,■

■ KIND= *scalar-int-initialization-expr*)

or (*type-param-value*,■

■[KIND=]*scalar-int-initialization-expr*)

or (KIND= *scalar-int-initialization-expr*■

■ [,LEN= *type-param-value*])

R507 *length-selector* is ([LEN=] *type-param-value*)

or * *char-length*[,]

R508 *char-length* is (*type-param-value*)

or *scalar-int-literal-constant*

约束:在 *length-selector* 中可选的逗号仅允许在 *type-declaration-stmt* 的 *type-spec* 中。

约束:仅当在 *type-declaration-stmt* 中不出现双冒号分隔符时,才允许 *length-selector* 中可选的逗号。

约束:*scalar-int-initialization-expr* 的值必须是非负的,且必须说明存在于处理系统的表示方法。

约束:*scalar-int-literal-constant* 不得包含 *kind-param*。

R509 *type-param-value* is *specification-expr*

or *

约束:若函数是内部函数或模块函数、数组值的、指针值的或递归的,则函数名禁止声明具有星号 type-param-value。

R510 *access-spec* is PUBLIC

or PRIVATE

约束:access-spec 属性只能出现在模块作用域单位中。

R511 *intent-spec* is IN

or OUT

or INOUT

约束:INTENT 属性禁止说明是虚拟过程或虚拟指针的虚元。

R512 *array-spec* is *explicit-shape-spec-list*

or *assumed-shape-spec-list*

or *deferred-shape-spec-list*

or *assumed-size-spec*

约束：最大的秩是七。

R513 *explicit-shape-spec* is 〔*lower-bound* :〕 *upper-bound*

R514 *lower-bound* is *specification-expr*

R515 *upper-bound* is *specification-expr*

约束：显形数组的界依赖于非常量表达式的值，该显形数组必须是虚元、函数结果或过程的自动数组。

R516 *assumed-shape-spec* is 〔*lower-bound*〕:

R517 *deferred-shape-spec* is :

R518 *assumed-size-spec* is 〔explicit-shape-spec-list，〕 ■

■〔*lower-bound* :〕 *

约束：数组值函数的函数名禁止声明为假取大小数组。

R519 *intent-stmt* is INTENT（*intent-spec*）〔::〕 ■

■*dummy-arg-name-list*

约束：*intent-stmt* 只能出现在辅程序或接口体的 *specification-part* 中。

约束：*dummy-arg-name* 禁止是虚拟过程或虚拟指针的名字。

R520 *optional-stmt* is OPTIONAL〔::〕*dummy-arg-name-list*

约束：*optional-stmt* 只能出现在辅程序或接口体的作用域单位中。

R521 *access-stmt* is *access-spec* 〔〔::〕 *access-id-list*〕

R522 *access-id* is *use-name*

or *generic-spec*

约束：*access-stmt* 只能出现在模块的作用域单位中。在模块的作用域单位中只允许一个具有省略 *access-id-list* 的可访问性语句。

约束：每个 *use-name* 必须是有名变量、过程、导出类型、有名常量或名表组的名字。

约束：具有 PRIVATE 可访问性类型的虚元或函数结果的模块过程必须具有 *PRIVATE* 可访问性，且不得具有 *PUBLIC* 可访问性的类属标识符。

R523 *save-stmt* is SAVE〔〔::〕*saved-entity-list*〕

R524 *saved-entity* is *object-name*

or /common-block-name/

约束：*object-name* 禁止是虚元名、过程名、函数结果名、自动数据对象名

或在公用块中的实体名。

约束:若一个省略保留实体表的 SAVE 语句出现在一作用域单位中,而且没有其他 SAVE 属性或 SAVE 语句明显地出现在同一作用域单位中,则是允许的。

R525 *dimension-stmt* is DIMENSION〔::〕 *array-name* ■
■(*array-spec*) ■
■〔,*array-name*(*array-spec*)〕…

R526 *allocatable-stmt* is ALLOCATABLE〔::〕 *array-name* ■
■〔(*deferred-shape-spec-list*)〕■
■〔,*array-name*■
■〔(*deferred-shape-spec-list*)〕〕…

约束:*array-name* 禁止是虚元或函数结果。

约束:对于 *array-name* 除非在该作用域单位中别的地方说明了 DIMENSION 属性,否则 *array-spec* 必须是 *deferred-shape-spec-list*。

R527 *pointer-stmt* is POINTER〔::〕*object-name*■
■〔(*deferred-shape-spec-list*)〕■
■〔,*object-name*■
■〔(*deferred-shape-spec-list*)〕〕…

约束:对于 *object-name* 禁止说明 INTENT 属性。

约束:对于 *object-name* 除非在该作用域单位中别的地方说明 DIMENSION 属性,否则 *array-spec* 必须是 *deferred-shape-spec-list*。

约束:对于 *object-name* 禁止说明 PARAMETER 属性。

R528 *target-stmt* is TAEGET〔::〕*object-name*■
■〔(*array-spec*)〕■
■〔,*object-name*〔(*array-spec*)〕〕…

约束:对于 *object-name* 禁止说明 PARAMETER 属性。

R529 *data-stmt* is DATA *data-stmt-set*■
■〔〔,〕 *data-stmt-set*〕. . .

R530 *data-stmt-set* is *data-stmt-object-list* ■
■/*data-stmt-value-list*/

R531 *data-stmt-object* is *variable*
or *data-implied-do*

R532 *data-stmt-value* is 〔*data-stmt-repeat* * 〕■

■ *data-stmt-constant*

R533 *data-stmt-constant* is *scalar-constant*

or *signed-int-literal-constant*

or *signed-real-literal-constant*

or *structrue-constructor*

or *boz-literal-constant*

R534 *data-stmt-repeat* is *scalar-int-constant*

R535 *data-implied-do* is (*data-i-do-object-list*,■

■ *data-i-do-variable*=■

■*scalar-int-expr*,*scalar-int-expr*■

■[,*scalar-int-expr*])

R536 *data-i-do-object* is *array-element*

or *scalar-structure-component*

or *data-implied-do*

约束:*array-element* 不得具有常量父辈。

约束:*scalar-structure-component* 不得具有常量父辈。

R537 *data-i-do-variable* is *scalar-int-variable*

约束:*data-i-do-variable* 必须是有名变量。

约束:DATA 语句重复因子必须是正的或零。若 DATA 语句重复因子是有名常量,则它必须在作用域单位中先前已声明或由使用结合或宿主结合构成可访问的。

约束:若 *data-stmt-constant* 是 *structure-constructor*,则每个成分必须是初始化表达式。

约束:在作为 *data-stmt-object* 的 *variable* 中,任何下标、片段下标、子串始点和子串终点必须是初始化表达式。

约束:其名或指定符是包括在 *data-stmt-object-list* 或 *data-i-do-object-list* 中的变量禁止是:虚元,由使用结合或宿主结合构造可访问的,除了在块数据程序单元的 DATA 语句以外的在有名公用块中的变量,在空白公用块中的变量,函数名,函数结果名,自动对象,指针或可分配数组。

约束:在作为 *data-i-do-object* 的 *array-element* 或 *scalar-structure-component* 中,任何下标必须是表达式,其初等量或是常量、或是包含 *data-implied-do* 的 DO 变量,且每个运算必须是内在的。

约束:*data-implied-do* 的 *scalar-int-expr* 必须仅是常量或包含 *data-im-*

plied-do 的 DO 变量作为初等量，且每个运算必须是内在的。

R538 *parameter-stmt* is PARAMETER(*named-constant-def-list*)

R539 *named-constant-def* is *named-constant*=*initialization-expr*

R540 *implicit-stmt* is IMPLICIT *implicit-spec-list*

or IMPLICIT NONE

R541 *implicit-spec* is *type-spec*(*letter-spec-list*)

R542 *letter-spec* is *letter*[- *letter*]

约束：在一作用域单位中若说明 IMPLICIT NONE，则它必须在该作用域单位中出现在任何 PARAMETER 语句之前，且在该作用域单位中必须没有其它的 IMPLICIT 语句。

约束：若减号和第二个字母出现，则第二个字母必须在字母表中的第一个字母之后。

R543 *namelist-stmt* is NAMELIST/*namelist-group-name*/■

■*namelist-group-object-list*■

■[[,]/*namelist-group-name*/■

■*namelist-group-object-list*]…

R544 *namelist-group-object* is *variable-name*

约束：*namelist-group-object* 禁止是具有非常量界的数组虚元、非常量字符长度的变量、自动对象、指针、具有终极成分是指针类型的变量或可分配数组。

约束：若 *namelist-group-name* 具有 PUBLIC 属性，则在 *namelist-group-object-list* 中没有一个项能具有 PRIVATE 属性。

R545 *equivalence-stmt* is EQUIVALENCE *equivalence-set-list*

R546 *equivalence-set* is (*equivalence-object*,■

■*equivalence-object-list*)

R547 *equivalence-objcet* is *variable-name*

or *array-element*

or *substring*

约束：*equivalence-object* 禁止是虚元、指针、可分配数组、非顺序导出类型的对象、在任何层成分选择上包含指针的顺序导出类型的对象、自动对象、函数名、入口名、结果名、有名常量、结构成分或前述对象的任何子对象。

约束：在 *equivalence-object* 中每个下标或子串范围表达式必须是整的初始化表达式。

约束：若 *equivalence-object* 是默认整型、默认实型、双精度实型、默认复型、默认逻辑型数值顺序类型，则在该等价集中所有对象都必须是这些类型的。

约束：若 *equivalence-object* 是默认字符型或字符顺序类型的，则在该等价集中所有对象都必须是这些类型的。

约束：若 *equivalence-object* 是非数值顺序或非字符顺序类型的导出类型的，则在该等价集中的所有对象必须是同样类型的。

约束：若 *equivalence-object* 是内在类型的，而此内在类型不是默认整型、默认实型、双精度实型、默认复型、默认逻辑型或默认字符型，则在该等价集中所有对象必须是具有相同种别类型参数值的相同类型的。

R548 *common-stmt* is COMMON〔/〔*common-block-name*〕/〕■
■ *common-block-object-list*■
■〔〔,〕/〔*common-block-name*〕/■
■*common-block-object-list*〕...

R549 *common-block-object* is *variable-name*■
■〔(*explicit-shape-spec-list*)〕

约束：在一作用域单位中所有 *common-block-object-list* 中给定的 *variable-name* 只允许出现一次。

约束：*common-block-object* 禁止是虚元、可分配数组、自动对象、函数名、入口名或结果名。

约束：在 *explicit-shape-spec* 中每个界必须是常量说明表达式。

约束：若一 *common-block-objcet* 是导出类型的，则它必须是顺序类型。

约束：若一 *variable-name* 出现时具有 *explicit-shape-spec-list*，则它不得具有 POINTER 属性。

6. 数据对象的使用

R601 *varialbe* is *scalar-variable-name*
or *array-variable-name*
or *subobject*

约束：*array-variable-name* 必须是数组数据对象的名字。

约束：*array-variable-name* 不得具有 PARAMETER 属性。

约束：*scalar-variable-name* 不得具有 PARAMETER 属性。

约束：*subobject* 禁止是其父为常量的子对象指定符(例如，子串)。

R602 *subobject* is *array-element*

or *array-section*

or *structure-component*

or *substring*

R603 *logical-variable* is *variable*

约束：*logical-variable* 必须是逻辑型的。

R604 *default-logical-variable* is *variable*

约束：*default-logical-variable* 必须是默认逻辑型的。

R605 *char-variable* is *variable*

约束：*char-variable* 必须是字符型的。

R606 *default-char-variable* is *variable*

约束：*default-char-variable* 必须是默认字符型的。

R607 *int-variable* is *variable*

约束：*int-variable* 必须是整型的。

R608 *default-int-variable* is *variable*

约束：*default-int-variable* 必须是默认整型的。

R609 *substring* is *parent-string*(*substring-range*)

R610 *parent-string* is *scalar-variable-name*

or *array-element*

or *scalar-structure-component*

or *scalar-constant*

R611 *substring-range* is [*scalar-int-expr*]:■

■ [*scalar-int-expr*]

约束：*parent-string* 必须是字符型的。

R612 *data-ref* is *part-ref*[% *part-ref*]…

R613 *part-ref* is *part-name* [(*section-subscript-list*)]

约束：在 *data-ref* 中，每个 *part-name* 除最右的一个外，必须是导出类型的。

约束：在 *data-ref* 中，每个 *part-name* 除最左的一个外，必须是由前一个 *part-name* 的类型的导出类型定义的一个成分的名字。

约束：在包含 *section-subscript-list* 的 *part-fef* 中，*sction-subscript* 的个数必须等于 *part-name* 的秩。

约束：在 *data-ref* 中禁止多个带非零秩的 *part-ref*。对具有非零秩 *part-ref* 的右部 *part-name* 不得具有 POINTER 属性。

R614 *structure-component* is *data-ref*

约束：在 *structure-component* 中，必须有多个 *part-ref*，且最右的 *part-ref* 必须为 *part-name* 形式。

R615 *array-element* is *data-ref*

约束：在 *array-element* 中，每个 *part-ref* 必须具有秩零且最后一个 *part-ref* 必须包含一个 *subscript-list*。

R616 *array-section* is *data-ref*〔(*substring-range*)〕

约束：在 *array-section* 中，确切地一个 *part-ref* 必须有非零秩，并且最终 *part-ref* 具有非零秩的 *section-subscript-list* 或另一个 *part-ref* 具有非零秩。

约束：在有 *substring-range* 的 *array-section* 中，最右的 *part-name* 必须是字符型的。

R617 *subscript* is *scalar-int-expr*

R618 *section-subscript* is *subscript*
or *subscript-triplet*
or *vector-subscript*

R619 *subscript-triplet* is 〔*subscript*〕：〔*subscript*〕〔：*stride*〕

R620 *stride* is *scalar-int-expr*

R621 *vector-subscript* is *int-expr*

约束：*vector-subscript* 必须是秩 1 的整数组表达式。

约束：在一个假定大小数组的最后一维中的 *subscript-triplet* 其第二个 *subscript* 禁止省略。

R622 *allocate-stmt* is ALLOCATE (*allocation-list*■
■〔,STAT= *stat-variable*〕)

R623 *stat-variable* is *scalar-int-variable*

R624 *allocation* is *allocate-object* ■
■〔(*allocate-shape-spec-list*)〕

R625 *allocate-object* is *variable-name*
or *structure-component*

R626 *allocate-shape-spec* is 〔*allocate-lower-bound*：〕■
■ *allocate-upper-bound*

R627 *allocate-lower-bound* is *scalar-int-expr*

R628 *allocate-upper-bound* is *scalar-int-expr*

约束：每个 *allocate-object* 必须是指针或可分配数组。

约束：在 *allocate-shape-spec-list* 中 *allocate-shape-spec* 的个数必须等于指针或可分配数组的秩。

R629 *nullify-stmt* is NULLIFY（*pointer-object-list*）

R630 *pointer-object* is *variable-name*

or *structure-component*

约束：每个 *pointer-object* 必须具有 POINTER 属性。

R631 *deallocate-stmt* is DEALLOCATE(*allocate-object-list* ■

■〔,STAT= *stat-variable*〕)

约束：每个 *allocate-object* 必须是指针或可分配数组。

7. 表达式和赋值

R701 *primary* is *constant*

or *constant-subobject*

or *variable*

or *array-constructor*

or *structure-constructor*

or *function-reference*

or (*expr*)

R702 *constant-subobject* is *subobject*

约束：*subobject* 必须是一个子对象指定符，其父辈是一个常量。

约束：作为 *primary* 的 *variable* 不得是一个假定大小的数组。

R703 *level-1-expr* is 〔*defined-unary-op*〕 *primary*

R704 *defined-unary-op* is . *letter* 〔 *letter*〕… .

约束：一个 *defined-unary-op* 不得包含多于 31 个字母并且不得与 *intrinsic-operator* 或 *logical-literal-constant* 相同。

R705 *mult-operand* is *level-1-expr*〔*power-op mult-operand*〕

R706 *add-operand* is 〔*add-operand mult-op*〕*mult-operand*

R707 *level-2-expr* is 〔〔*level-2-expr*〕*add-op*〕 *add-operand*

R708 *power-op* is * *

R709 *mult-op* is *

or /

R710 *add-op* is +

or -

R711 *level-3-expr* is 〔*level-3-expr concat-op*〕■

■*level-2-expr*

R712 *concat-op* is //

R713 *level-4-expr* is 〔*level-3-expr rel-op*〕*level-3-expr*

R714 *rel-op* is .EQ.
or .NE.
or .LT.
or .LE.
or .GT.
or .GE.
or ==
or /=
or <
or <=
or >
or >=

R715 *and-operand* is 〔*not-op*〕*level-4-expr*

R716 *or-operand* is 〔*or-operand and-op*〕 *and-operand*

R717 *equiv-operand* is 〔*equiv-operand or-op*〕*or-operand*

R718 *level-5-expr* is 〔*level-5-expr equiv-op*〕■
■*equiv-operand*

R719 *not-op* is .NOT.

R720 *and-op* is .AND.

R721 *or-op* is .OR.

R722 *equiv-op* is .EQV.
or .NEQV.

R723 *expr* is 〔*expr defined-binary-op*〕■
■ *level-5-expr*

R724 *defined-binary-op* is .*letter* 〔*letter*〕….

约束:*defined-binary-op* 不得包含多于 31 个字母并且不得与 *intrinsic-operator* 或 *logical-literal-constant* 相同。

R725 *logical-expr* is *expr*

约束:*logical-expr* 必须是逻辑型的。

R726 *char-expr* is *expr*

约束:*char-expr* 必须是字符型的。

R727 *default-char-expr* is *expr*

约束:*default-char-expr* 必须是默认字符型的。

R728 *int-expr* is *expr*

约束:*int-expr* 必须是整型的。

R729 *numeric-expr* is *expr*

约束:*numeric-expr* 必须是整型、实型或复型。

R730 *initialization-exp* is *expr*

约束:*initialization-expr* 必须是一个初始化表达式。

R731 *char-initialization-expr* is *char-expr*

约束:*char-initialization-expr* 必须是一个初始化表达式。

R732 *int-initialization-expr* is *int-expr*

约束:*int-initialization-expr* 必须是一个初始化表达式。

R733 *logical-initialization-expr* is *logical-expr*

约束:*logical-initialization-expr* 必须是一个初始化表达式。

R734 *specification-expr* is *scalar-int-expr*

约束:*scalar-int-expr* 必须是一个受限表达式。

R735 *assignment-stmt* is *variable*= *expr*

约束:在 *assignment-stmt* 中的 *variable* 不得是假定大小的数组。

R736 *pointer-assignment-stmt* is *pointer-object*=> *target*

R737 *target* is *variable*

or *expr*

约束:*pointer-object* 必须有 POINTER 属性。

约束:*variable* 必须有 TARGET 属性,或者是一个具有 TARGET 属性的对象的子对象,或者它必须有 POINTER 属性。

约束:*target* 必须和指针有同样的类型、类型参数和秩。

约束:*target* 不得是具有向量下标的数组片段。

约束:*expr* 必须交付一个指针结果。

R738 *where-stmt* is WHERE(*mask-expr*) *assignment-stmt*

R739 *where-construct* is *where-construct-stmt*

[*assignment-stmt*]…

[*elsewhere-stmt*

[*assignment-stmt*]…]

end-where-stmt

R740 *where-construct-stmt* is WHERE(*mask-expr*)

R741 *mask-expr* is *logical-expr*

R742 *elsewhere-stmt* is ELSEWHERE

R743 *end-where-stmt* is END WHERE

约束：在每个 assignment-stmt 中，mask-expr 和正被定义的变量必须是同形数组。

约束：*assignment-stmt* 禁止是定义的赋值。

8. 执行控制

R801 *block* is 〔*execution-part-construct*〕…

R802 *if-construct* is *if-then-stmt*

block

〔*else-if-stmt*

block〕…

〔*else-stmt*

block〕

end-if-stmt

R803 *if-then-stmt* is 〔*if-construct-name*:〕 IF ■

■ 〔*scalar-logical-expr*〕 THEN

R804 *else-if-stmt* is ELSE IF(*scalar-logical-expr*)■

■ THEN 〔*if-construct-name*〕

R805 *else-stmt* is ELSE〔*if-construct-name*〕

R806 *end-if-stmt* is END IF〔*if-construct-name*〕

约束：如果 *if-construct* 的 *if-then-stmt* 用 *if-construct-name* 标识，那么相应的 *end-if-stmt* 必须说明同样的 *if-construct-name*。如果 *if-construct* 的 *if-then-stmt* 没用 *if-construct-name* 标识，则相应的 *end-if-stmt* 不得说明 *if-construct-name*。如果 *else-if-stmt* 或 *else-stmt* 用 *if-construct-name* 标识，则相应的 *if-then-stmt* 必须说明同样的 *if-construct-name*。

R807 *if-stmt* is IF (*scalar-logical-expr*)*action-stmt*

约束：*if-stmt* 的 *action-stmt* 不得是 *if-stmt*、*end-program-stmt*、*end-function-stmt* 或 *end-subroutine-stmt*。

R808 *case-construct* is *select-case-stmt*

〔*case-stmt*

block〕…

end-select-stmt

R809 *select-case-stmt* is 〔*case-construct-name*:〕 ■

■ SELECT CASE(*case-expr*)

R810 *case-stmt* is CASE*case-selector* ■

■ 〔*case-construct-name*〕

R811 *end-select-stmt* is END SELECT*case-construct-name*〕

约束:如果 *case-construct* 的 *select-case-stmt* 用 *case-construct-name* 标识,则相应的 *end-select-stmt* 必须说明同样的 *case-construct-name*。如果 *case-construct* 的 *select-case-stmt* 没有用 *case-construct-name* 标识,则相应的 *end-select-stmt* 不得说明 *case-construct-name*。如果 *case-stmt* 用 *case-construct-name* 标识,则相应的 *select-case-stmt* 必须说明同样的 *case-construct-name*。

R812 *case-expr* is *scalar-int-expr*

or *scalar-char-expr*

or *scalar-logical-expr*

R813 *case-selector* is (*case-value-range-list*)

or DEFAULT

约束:一个 CASE 语句不得有一个以上选择符为 DEFAULT。

R814 *case-value-range* is *case-value*

or *case-value*:

or : *case-value*

or *case-value* :*case-value*

R815 *case-value* is *scalar-int-initialization-expr*

or *scalar-char-initialization-expr*

or *scalar-logical-initialization-expr*

约束:对给定的 *case-construct*,每个 *case-value* 必须与 *case-expr* 同样类型。对字符型,允许长度不同,但种别类型参数必须相同。

约束:如果 *case-expr* 是逻辑型的,则不得使用有冒号的 *case-value-range*。

约束:对给定的 *case-construct*、*case-value-ranges* 不得重迭;也就是不可能有 *case-expr* 的值与不止一个的 *case-value-range* 相匹配。

R816 *do-construct* is *block-do-construct*

or *nonblock-do-construct*

R817 *block-do-construct* is *do-stmt*

do-block

end-do

R818 *do-stmt* is *label-do-stmt*

or *nonlable-do-stmt*

R819 *label-do-stmt* is [*do-construct-name*:]DO *lable* ■

■ DO [*loop-control*]

R820 *nonlabel-do-stmt* is [*do-construct-name*:] ■

■ DO [*loop-control*]

R821 *loop-control* is [,] *do-variable* =■

■ *scalar-numeric-expr*,■

■ *scalar-numeric-expr*■

■[, *scalar-numeric-expr*]■

or [,] WHILE(*scalar-logical-expr*)

R822 *do-variable* is *scalar-variable*

约束:*do-variable* 必须是整型的有名标量变量,默认实型或双精度实型的有名标量变量。

约束:在 *loop-control* 中的每个 *scalar-numeric-expr* 必须为整型的,或默认实型或双精度实型。

R823 *do-block* is *block*

R824 *end-do* is *end-do-stmt*

or *continue-stmt*

R825 *end-do-stmt* is END DO[*do-construct-name*]

约束:如果 *block-do-construct* 的 *do-stmt* 用 *do-construct-name* 标识,则相应的 *end-do* 必须是说明同样 *do-construct-name* 的 *end-do-stmt*。如果 *block-do-construct* 的 *do-stmt* 没用 *do-construct-name* 标识,则相应的 *end-do* 不得说明 *do-construct-name*。

约束:如果 *do-stmt* 是 *nonlabel-do-stmt*,则相应的 *end-do* 必须是一个 *end-do-stmt*。

约束:如果 *do-stmt* 是 *label-do-stmt*,则相应的 *end-do* 必须用同样的 *label* 标识。

R826 *noublock-do-construct* is *action-term-do-construct*

or *outer-shared-do-construct*

R827 *action-term-do-construct* is *label-do-stmt*

do-body

do-term-action-stmt

R828 *do-body* is 〔*execution-part-construct*〕…

R829 *do-term-action-stmt* is *action-stmt*

约束:*do-term-action-stmt* 不得是 *continue-stmt*、*goto-stmt*、*return-stmt*、*stop-stmt*、*exit-stmt*、*cycle-stmt*、*end-function-stmt*、*end-subroutine-stmt*、*end-program-stmt*、*arithmetic-if-stmt* 或 *assigned-goto-stmt*。

约束:*do-term-action-stmt* 必须有标号来标识,相应的 *label-do-stmt* 必须引用同样的标号。

R830 *outer-shared-do-construct* is *label-do-stmt*

do-body

shared-term-do-construct

R831 *shared-term-do-construct* is *outer-shared-do-construct*

or *inner-shared-do-construct*

R832 *inner-shared-do-construct* is *label-do-stmt*

do-body

do-term-shared-stmt

R833 *do-term-shared-stmt* is *action-stmt*

约束:*do-term-shared-stmt* 不得是 *goto-stmt*、*return-stmt*、*stop-stmt*、*exit-stmt*、*cycle-stmt*、*end-function-stmt*、*end-subroutine-stmt*、*end-program-stmt*、*arithmetic-if-stmt* 或 *assigned-goto-stmt*。

约束:*do-term-shared-stmt* 必须由标号标识,*shared-term-do-construct* 的所有 *label-do-stmt* 必须引用相同标号。

R834 *cycle-stmt* is CYCLE 〔*do-construct-name*〕

约束:如果 *cycle-stmt* 引用一个 *do-construct-name*,它必须在 *do-construct* 的范围之内;否则它必须在至少一个 *do-construct* 范围之内。

R835 *exit-stmt* is EXIT〔*do-construct-name*〕

约束:如果 *exit-stmt* 引用 *do-construct-name*,它必须在 *do-construct* 的范围之内,否则必须至少在一个 *do-construct* 的范围之内。

R836 *goto-stmt* is GOTO *label*

约束:*label* 必须为出现在与 *goto-stmt* 同一作用域单位中的分支目标语句的语句标号。

R837 *computed-goto-stmt* is GOTO (*label-list*)〔,〕■

■*scalar-int-expr*

约束:*label-list* 中每个 *label* 必须是出现在与 *computed-goto-stmt* 同一作用域单位的分支目标语句的语句标号。

R838 *assign-stmt* is ASSIGN *label* TO *scalar-int-variable*

约束:*label* 必须是与 *assign-stmt* 同一作用域单位中出现的分支目标语句或 *format-stmt* 语句标号。

约束:*scalar-int-variable* 必须是有名的,并为默认整型的。

R839 *assigned-goto-stmt* is GOTO*scalar-int-variable*[[,](*label-list*)]

约束:*label-list* 中每个 *label* 必须是与 *assigned-goto-stmt* 同一作用域单位中出现的分支目标语句的语句标号。

约束:*scalar-int-variable* 必须为有名的和默认整型的。

R840 *arithmetic-if-stmt* is IF(*scalar-numeric-expr*)*label*,*label*, *label*

约束:每个 *label* 必须是出现在与 *arithmetic-if-stmt* 同一作用域单位中的分支目标语句的标号。

约束:*scalar-numeric-expr* 不得为复型。

R841 *continue-stmt* is CONTINUE

R842 *stop-stmt* is STOP[*stop-code*]

R843 *stop-code* is *scalar-char-constant*

or *digit*[*digit*[*digit* [*digit*■

■[*digit*]]]]

约束:*scalar-char-constant* 必须为默认字符型的。

R844 *pause-stmt* is PAUSE[*stop-code*]

9. 输入/输出语句

R901 *io-unit* is *external-file-unit*

or *

or *internal-file-unit*

R902 *external-file-unit* is *scalar-int-expr*

R903 *internal-file-unit* is *default-char-variable*

约束:*default-char-variable* 不得是带有向量下标的数组片段。

R904 *open-stmt* is OPEN(*connect-spec-list*)

R905 *connect-spec* is [UNIT=] *external-file-unit*

or IOSTAT= *scalar-default-int-variable*

or ERR= *label*

or FILE = *file-name-expr*

or STATUS= *scalar-default-char-expr*

or ACCESS= *scalar-default-char-expr*

or FORM= *scalar-default-char-expr*

or RECL= *scalar-int-expr*

or BLANK= *scalar-default-char-expr*

or POSITION= *scalar-default-char-expr*

or ACTION= *scalar-default-char-expr*

or DELIM= *scalar-default-char-expr*

or PAD= *scalar-default-char-expr*

R906 *file-name-expr* is *scalar-default-char-expr*

约束：如果从部件说明符中省略了可选字符 UNIT= ，则在 *connect-spec-list* 中的第一项必须是部件说明符。

约束：在给定的 *open-stmt* 中，每个说明符最多只能出现一次；必须说明 *external-file-unit*。

约束：在 ERR= 说明符中使用的 *label* 必须是在与 OPEN 语句同一作用域单位中出现的分支目标语句的语句标号。

R907 *close-stmt* is CLOSE (*close-spec-list*)

R908 *close-spec* is [UNIT=]*external-file-unit*

or IOSTAT= *scalar-default-int-variable*

or ERR= *label*

or STATUS= *scalar-default-char-expr*

约束：如果在部件说明符中省略了可选字符 UNIT=，则部件说明符必须是 *close-spec-list* 中的第一项。

约束：在给定的 *close-stmt* 中，每个说明符不得出现一次以上；必须说明 *external-file-unit*。

约束：在 ERR=说明符中使用的 *label* 必须是出现在与 CLOSE 语句同一个作用域单位内的分支目标语句的语句标号。

R909 *read-stmt* is READ(*io-control-spec-list*)■

■[*input-item-list*]

or READ *format*[,*input-item-list*]

R910 *write-stmt* is WRITE(*io-control-spec-list*)■

■[*output-item-list*]

R911 *print-stmt* is PRINT *format*[,*output-item-list*]

R912 *io-control-spec* is [UNIT=]*io-unit*

or [FMT=] *format*

or [NML=] *namelist-group-name*

or REC= *scalar-int-expr*

or IOSTAT= *scalar-default-int-variable*

or ERR= *label*

or END= *label*

or ADVANCE= *scalar-default-char-expr*

or SIZE= *scalar-default-int-variable*

or EOR= *label*

约束:*io-control-spec-list* 必须严格地包含一个 *io-unit*,且可以包含每个其它说明符最多一次。

约束:END=、EOR=或 SIZE=说明符不得出现在 *write-stmt* 中。

约束:在 ERR=、EOR=或 END=说明符中的 *label* 必须是出现在此数据传输语句同一作用域单位中的分支目标语句的语句标号。

约束:若在数据传输语句中存在有 *input-item-list* 或 *output-item-list*,则不得存在 *namelist-group-name*。

约束:*io-control-spec-list* 不得包含 *format* 和 *namelist-group-name* 二者。

约束:若从部件说明符中省略了可选的字符 UNIT=,则部件说明符必须是控制信息表中的第一项。

约束:若从格式说明符中省略了可选字符中 FMT=,则格式说明符必须是控制信息表中的第二项,且第一项必须是不带可选字符 UNIT=的部件说明符。

约束:若从名表说明符中省略了可选字符 NML=,则名表说明符必须是控制信息表中的第二项,且第一项必须是不带可选字符 UNIT=的部件说明符。

约束:若部件说明符说明了一个内部文件,则 *io-control-spec-list* 不得包含 REC=说明符或 *namelist-group-name*。

约束:若存在 REC=说明符,则不得出现 END=说明符和 *namelist-group-name*,且 *format*(若有的话)禁止是说明表控输入/输入的星号。

约束:ADVANCE=说明符仅可出现在具有显式格式说明的格式顺序输入/输出语句中,它的控制信息表不包含内部文件部件说明符。

约束:若存在 EOR=说明符,则也必须出现 ADVANCE=说明符。

R913 *format* is *default-char-expr*

or *label*

or *

or *scalar-default-int-variable*

约束:*label* 必须是一个与包含此格式说明符的语句出现于同一作用域单位的 FORMAT 语句的标号。

R914 *input-item* is *variable*
or *io-implied-do*

R915 *output-item* is *expr*
or *io-implied-do*

R916 *io-implied-do* is (*io-implied-do-object-list*,■
■*io-implied-do-control*)

R917 *io-implied-do-object* is *input-item*
or *output-item*

R918 *io-implied-do-control* is *do-variable*=*scalar-numeric-expr*,■
■*scalar-numeric-expr*■
■[,*scalar-numeric-expr*]

约束:作为 *input-item* 的 *variable* 禁止是一个假定大小的数组。

约束:*do-variable* 必须是整型、默认实型或双精度实型的标量。

约束:在 *io-implied-do-control* 中的每一个 *scalar-numeric-expr* 必须是整型、默认实型或双精度实型的标量。

约束:在 *input-item-list* 中,*io-implied-do-object* 必须是 *input-item*。在 *output-item-list* 中,*io-implied-do-object* 必须是 *output-item* 。

R919 *backspace-stmt* is BACKSPACE *external-file-unit*
or BACKSPACE(*position-spec-list*)

R920 *endfile-stmt* is ENDFILE *external-file-unit*
or ENDFILE(*position-spec-list*)

R921 *rewind-stmt* is REWIND *external-file-unit*
or REWIND(*position-spec-list*)

R922 *position-spec* is [UNIT=] *external-file-unit*
or IOSTAT=*scalar-default-int-variable*
or ERR=*label*

约束:在 ERR=说明符中的 *label* 必须是与此文件定位语句在同一个作用域单位中出现的分支目标语句的语句标号。

约束：若从部件说明符中省略了可选字符 UNIT＝，则部件说明符必须是 *position-spec-list* 中的第一项。

约束：*position-spec-list* 必须严格地包含一个 *external-file-unit*，且可以包含每一个其它说明符至多一次。

R923 *inquire-stmt* is INQUIRE(*inquire-spec-list*)

or INQUIRE(IOLENGTH= ■

■*scalar-default-int-variable*) ■

■*output-item-list*

R924 *inquire-spec* is [UNIT=]*external-file-unit*

or FILE=*file-name-expr*

or IOSTAT=*scalar-default-int-variable*

or ERR=*label*

or EXIST= ■

■ *scalar-default-logical-variable*

or OPENED= ■

■ *scalar-default-logical-variable*

or NUMBER=*scalar-default-int-variable*

or NAMED= ■

■ *scalar-default-logical-variable*

or NAME=*scalar-default-char-variable*

or ACCESS=*scalar-default-char-variable*

or SEQUENTIAL= ■

■*scalar-default-char-variable*

or DIRECT= ■

■*scalar-default-char-variable*

or FORM=*scalar-default-char-variable*

or FORMATTED= ■

■*scalar-default-char-variable*

or UNFORMATTED= ■

■*scalar-default-char-variable*

or RECL=*scalar-default-int-variable*

or NEXTREC=

■scalar-default-int-variable

or BLANK=*scalar-default-char-variable*

or POSITION=■

■*scalar-default-char-variable*

or ACTION=■

■*scalar-default-char-variable*

or READ=*scalar-default-char-variable*

or WRITE=■

■*scalar-default-char-variable*

or READWRITE=■

■*scalar-default-char-variable*

or DELIM=*scalar-default-char-variable*

or PAD=*scalar-default-char-variable*

约束:*inquire-spec-list* 必须包含一个 FILE=说明符或一个 UNIT=说明符,但不得包含两者,且包含其它查询说明符中的每一种最多一次。

约束:在 INQUIRE 语句的按部件形式的查询中,如果从部件说明符中省略了可选字符 UNIT=,则部件说明符必须是 *inquire-spec-list* 中的第一项。

10. 输入/输出编辑

R1001 *format-stmt* is FORMAT*format-specification*

R1002 *format-specification* is ([*format-item-list*])

约束:*format-stmt* 必须带标号。

约束:在 *format-item-list* 中用于分隔 *format-item* 的逗号可按如下规则省略:

(1)在 *P* 编辑描述符与紧随的 F、E、EN、ES、D 或 G 编辑描述符之间;

(2)当可选的重复说明不出现时,在斜线编辑描述符之前;

(3)在斜线编辑描述符之后;

(4)在冒号编辑描述符的前后。

R1003 *format-item* is [*r*]*data-edit-desc*

or *control-edit-desc*

or *char-string-edit-desc*

or [*r*](*format-item-list*)

R1004 *r* is *int-literal-constant*

约束:r 必须为正。

约束:r 不得具有为它说明的种别参数。

R1005 *data-edit-desc*	is I w[. m]
	or B w[. m]
	or O w[. m]
	or Z w[. m]
	or F $w.d$
	or E $w.d$[E e]
	or EN $w.d$[E e]
	or ES $w.d$[E e]
	or G $w.d$[E e]
	or L w
	or A[w]
	or D $w.d$
R1006 w	is *int-literal-constant*
R1007 m	is *int-literal-constant*
R1008 d	is *int-literal-constant*
R1009 e	is *int-literal-constant*

约束:w 和 e 必须为正。

约束:w、m、d 及 e 不得具有为它们说明的种别参数。

R1010 *control-edit-desc*	is *position-edit-desc*
	or [r]/
	or :
	or *sign-edit-desc*
	or kP
	or *blank-interp-edit-desc*
R1011 k	is *signed-int-literal-constant*

约束:k 不得具有为它说明的种别参数。

R1012 *position-edit-desc*	is Tn
	or TL n
	or TR n
	or nX
R1013 n	is *int-literal-constant*

约束:*n* 必须为正。

约束:*n* 不得具有为它说明的种别参数。

R1014 *sign-edit-desc* is S
or SP
or SS

R1015 *blank-interp-edit-desc* is BN
or BZ

R1016 *char-string-edit-desc* is *char-literal-constant*
or *c*H *rep-char*[*rep-char*]…

R1017 *c* is *int-literal-constant*

约束:*c* 必须为正。

约束:*c* 不得具有为它说明的种别参数。

约束:*c*H 形式中的 *rep-char* 必须是默认字符型的。

约束:*char-literal-constant* 不得具有为它说明的种别类型参数。

11. 程序单元

R1101 *main-program* is [*program-stmt*]
[*specification-part*]
[*execution-part*]
[*internal-subprogram-part*]
end-program-stmt

R1102 *program-stmt* is PROGRAM *program-name*

R1103 *end-program-stmt* is END[PROGRAM[*program-name*]]

约束:在 *main-program* 中,*execution-part* 不得包含 RETURN 语句或 ENTRY 语句。

约束:仅当使用了可选的 *program-stmt* 语句时,才可在 *end-program-stmt* 中包含 *program-name*,若包含了它,则必须与 *program-stmt* 中说明的 *program-name* 相同。

约束:自动对象不得出现在主程度的 *specification-part*(见 R204)中。

R1104 *module* is *module-stmt*
[*specification-part*]
[*module-subprogram-part*]
end-module-stmt

R1105 *module-stmt* is MODULE *module-name*

R1106 *end-module-stmt* is END[MODULE[*module-name*]]

约束：若 *module-name* 在 *end-module-stmt* 中说明，则它必须与 *module-stmt* 中说明的 *module-name* 相同。

约束：模块的 *specification-part* 不得包含 *stmt-function-stmt*、*entry-stmt* 或 *format-stmt*。

约束：自动对象不得出现在模块的 *specification-part*（见 R204）中。

R1107 *use-stmt* is USE *module-name*[,*rename-list*]
or USE *module-name*,ONLY:[*only-list*]

R1108 *rename* is *local-name*=> *use-name*

R1109 *only* & is *access-id*
or [*local-name*=>] *use-name*

约束：每一个 *access-id* 必须是模块内的共用实体。

约束：每一个 *use-name* 必须是模块内共用实体的名字。

R1110 *block-data* is *block-data-stmt*
[*specification-part*]
end-block-data-stmt

R1111 *block-data-stmt* is BLOCK DATA[*block-data-name*]

R1112 *end-block-data-stmt* is END[BLOCK DATA[*block-data-name*]]

约束：仅当在 *block-data-stmt* 中提供了 *block-data-name* 时，它才可被包含在 *end-block-data-stmt* 内，并且，当包含它时，必须与 *block-data-stmt* 内的 *block-data-name* 相同。

约束：*block-data specification-part* 可以只包含 USE 语句、类型声明语句、IMPLICIT 语句、PARAMETER 语句、导出类型定义以及下列说明语句：COMMON、DATA、DIMENSION、EQUIVALENCE、INTRINSIC、POINTER、SAVE 和 TARGET。

约束：在 *block-data specification-part* 内的类型声明语句不得包含 ALLOCATABLE、EXTERNAL、INTENT、OPTIONAL、PRIVATE 或 PUBLIC 属性说明符。

12. 过程

R1201 *interface-block* is *interface-stmt*
[*interface-body*]…
[*module-procedure-stmt*]…
end-interface-stmt

R1202 *interface-stmt*　　is INTERFACE[*generic-spec*]

R1203 *end-interface-stmt*　　is END INTERFACE

R1204 *interface-body*　　is *function-stmt*

　　[*specification-part*]

　　end-function-stmt

　　or *subroutine-stmt*

　　[*specification-part*]

　　end-subroutine-stmt

R1205 *module-procedure-stmt* is MODULE PROCEDURE■

　　■*procedure-name-list*

R1206 *generic-spec*　　is *generic-name*

　　or OPERATOR(*defined-operator*)

　　or ASSIGNMENT(=)

约束：*interface-body* 不得包含 *entry-stmt*、*data-stmt*、*format-stmt* 或 *stmt-function-stmt*。

约束：仅当 *interface-block* 有 *generic-spec* 且有一个宿主，而宿主是一模块或通过使用结合访问模块的情况下，才允许 MODULE PROCEDURE 说明；每一 *procedure-name* 必须是在宿主中可以访问的模块过程的名字。

约束：*interface-block* 不得出现在 BLOCK DATA 程序单元中。

约束：在辅程序的 *interface-block* 不得包含由该辅程序所定义的过程的 *interface-body*。

R1207 *external-stmt*　　is EXTERNAL *external-name-list*

R1208 *intrinsic-stmt*　　is INTRINSIC■

　　■*intrinsic-procedure-name-list*

约束：每一个 *intrinsic-procedure-name* 必须是一个内在过程的名字。

R1209 *function-reference*　　is *function-name*■

　　■([*actual-arg-spec-list*])

约束：函数引用的 *actual-arg-spec-list* 不得包含 *alt-return-spec*。

R1210 *call-stmt*　　is CALL *subroutine-name*■

　　■[([*actual-arg-spec-list*])]

R1211 *actual-arg-spec*　　is [*keyword* =]*actual-arg*

R1212 *keyword*　　is *dummy-arg-name*

R1213 *actual-arg*　　is *expr*

or *variable*

or *procedure-name*

or *alt-return-spec*

R1214 *alt-return-spec* is * *label*

约束:若在作用域单位内过程的接口是隐式的,则不得出现 *keyword* =。

约束:仅当在变元表中已经从每一先前的 *actual-arg-spec* 中把 *keyword* = 省略,则可以从 *actual-arg-spec* 中省略 *keyword* =。

约束:每一个 *keyword* 必须是过程显式接口中的虚元名。

约束:*procedure-name actual-arg* 禁止是内部过程名或语句函数名,而且禁止是过程的类属名。

约束:用在 *alt-return-spec* 中的 *label* 必须是与 *call-stmt* 出现在同一作用域单位内的分支目标语句的语句标号。

R1215 *function-subprogram* is *function-stmt*

[*specification-part*]

[*execution-part*]

[*internal-subprogram-part*]

end-function-stmt

R1216 *function-stmt* is [*prefix*] FUNCTION *function-name*■

■([*dummy-arg-name-list*])■

■[RESULT(*result-name*]

约束:若说明 RESULT,则 *function-name* 不得出现在函数辅程序的作用域单位内的任何说明语句中。

R1217 *prefix* is *type-spec*[RECURSIVE]

or RECURSIVE[*type-spec*]

R1218 *end-function-stmt* is END[FUNCTION[*function-name*]]

约束:若说明 RESULT,则 *result-name* 禁止与 *function-name* 相同。

约束:在内部函数或模块函数的 *end-function-stmt* 必须出现 FUNCTION。

约束:内部函数不得包含 ENTRY 语句。

约束:内部函数不得包含 *internal-subprogram-part*。

约束:若 *function-name* 出现在 *end-function-stmt* 中,则它必须与 *function-stmt* 中说明的 *function-name* 相同。

R1219 *subroutine-subprogram* is *subroutine-stmt*

[*specification-part*]

[*execution-part*]

[*internal-subprogram-part*]

end-subroutine-stmt

R1220 *subroutine-stmt* is [RECURSIVE]SUBROUTINE■

■*subroutine-name*■

■[([*dummy-arg-list*])]

R1221 *dummy-arg* is *dummy-arg-name*

or *

R1222 *end-subroutine-stmt* is END[SUBROUTINE[*subroutine-name*]]

约束:*SUBROUTINE* 必须出现在内部子程序或模块子程序的 *end-subroutine-stmt* 中。

约束:内部子程序不得包含 ENTRY 语句。

约束:内部子程序不得包含 *internal-subprogram-part*。

约束:若 *subroutine-name* 出现在 *end-subroutine-stmt* 中,则它必须与 *subroutine-stmt* 中说明的 *subroutine-name* 相同。

R1223 *entry-stmt* is ENTRY*entry-name*■

■[([*dummy-arg-list*])■

■[RESULT *result-name*]]

约束:若说明了 RESULT,则 *entry-name* 不得出现在辅程序的作用域单位内的任何说明语句中。

约束:*entry-stmt* 只可以出现在 *external-subprogram* 或 *module-subprogram* 中。在 *executable-construct* 内不得出现 *entry-stmt*。

约束:仅当函数辅程序中包含 *entry-stmt* 时才可以出现 RESULT。

约束:在包含 *entry-stmt* 的辅程序内,*entry-name* 不得作为虚元出现在 FUNCTION 语句,SUBROUTINE 语句中,或在另一个 ENTRY 语句中,并也不得出现在 EXTERNAL 语句或 INTRINSIC 语句中。

约束:仅当 ENTRY 语句被包含在一个子程序辅程序中时,*dummy-arg* 才可以是交错返回指示符。

约束:若说明了 RESULT,则 *result-name* 禁止与 *entry-name* 相同。

R1224 *return-stmt* is RETURN[*scalar-int-expr*]

约束:函数或子程序辅程序的作用域单位内必须包含 *return-stmt*。

结束:仅在子程序辅程序的作用域单位内才允许 *scalar-int-expr*。

R1225 *contains-stmt* is CONTAINS

R1226 *stmt-function-stmt* is *function-name*■

■([*dummy-arg-name-list*])=■

■*scalar-expr*

约束:*scalar-expr* 只可以由常量(字面的和有名的)、对标量变量的引用和对数组元素的引用、对函数的引用和函数虚过程的引用、以及内在运算符构成。若在 *scalar-expr* 中出现对语句函数的引用,则在作用域单位内必须预先提供它的定义以及不得是正在定义的语句函数名。

约束:在作用域单位内必须预先声明 *scalar-expr* 中的有名常量、或者通过宿主结合或使用结合使其是可访问的。若数组元素出现在 *scalar-expr* 中,则在作用域单位内必须预先声明其父辈数组是一数组、或者通过宿主结合或使用结合使其是可访问的。若标量变量、数组元素、函数引用或虚函数引用被隐含类型规则确定其类型,则在任何后继的类型声明语句中的出现必须确认这种隐含类型及隐含类型参数的值。

约束:必须显式或隐式说明 *function-name* 和每个 *dummy-arg-name* 是标量数据对象。

约束:一个给定的 *dummy-arg-name* 只可以在某个 *dummy-arg-name-list* 中出现一次。

约束:在 *scalar-expr* 中每一个标量变量引用既可以是对语句函数虚元的引用也可以是对与该语句函数语句属同一作用域单位的局部变量的引用。

13. 内在过程

没有语法规则和约束。

14. 作用域、结合和定义

没有语法规则和约束。

附录二 索引

accessibility attribute 可访问性属性

access-id R522 可访问性标识符

access-spec R510 可访问性说明

access-stmt R521 可访问性语句

ac-do-variable R435	数组构造符的隐 do 变量
ac-implied-do R433	数组构造符的隐 do
ac-implied-do-control R434	数组构造符的隐 do 控制
action-stmt R216	动作语句
action-term-do-construct R827	动作终结 do 构造
active	活跃的
actual-arg R1213	实元
actual-arg-spec R1211	实元说明
ac-value R432	数组构造符的值
add-op R710	加法运算符
add-operand R706	加法操作数
advancing input/output statement	推进式输入/输出语句
allocatable array	可分配数组
ALLOCATABLE attribute	ALLOCATABLE 属性
allocatable-stmt R526	可分配语句
ALLOCATE statement	ALLOCATE 语句
allocate-lower-bound R627	分配下界
allocate-object R625	分配对象
allocate-shape-spec R626	分配形说明
allocate-stmt R622	分配语句
allocate-upper-bound R628	分配上界
allocation R624	分配
alphanumeric-character R302	字母数字字符
alt-return-spec R1214	交错返回说明
and-op R720	与运算符
and-operand R715	与操作数
approximation methods	近似方法
argument keyword	变元关键词
arithmetic-if-stmt R840	算术 if 语句

array	数组
array constructor	数组构造符
array element	数组元素
array element order	数组元素次序
array elements	数组元素
array intrinsic assignment statement	数组内在赋值语句
array pointer	数组指针
array section	数组片段
array-constructor R431	数组构造符
array-element R615	数组元素
array-section R616	数组片段
array-spec R512	数组说明
ASCII collating sequence	ASCII 理序序列
assigned-goto-stmt R839	赋值 go to 语句
assignment-stmt R735	赋值语句
assign-stmt R838	(语句标号)赋值语句
associated	被结合的
Association	结合
assumed type parameter	假取类型参数
assumed-shape array	假取形数组
assumed-shape-spec R516	假取形说明
assumed-size array	假取大小数组
assumed-size-spec R518	假取大小说明
attributes	属性
attr-spec R503	属性说明
automatic array	自动数组
automatic data object	自动数据对象
backspace-stmt R919	回退语句
belongs	属于

binary-constant R408	二进制常量
blank common	空白公用(块)
blank-interp-edit-desc R1015	空格控制编辑描述符
block	块
block data program unit	块数据程序单元
block R801	块
block-data R1110	块数据
block-data-stmt R1111	块数据语句
block-do-construct R817	块 do 构造
boz-literal-constant R407	二、八、十六进制字面常量
branch target statement	分支目标语句
Branching	分支
c R1017	整字面常量 c
call-stmt R1210	调用语句
CASE construct	CASE 构造
case index	分情况下标
case-construct R808	分情况构造
case-expr R812	分情况表达式
case-selector R813	分情况选择符
case-stmt R810	分情况语句
case-value R815	分情况值
case-value-range R814	分情况值范围
charcter constant expression	字符常量表达式
character context	字符上下文
character intrinsic assignment statement	字符内在赋值语句
character intrinsic operation	字符内在运算
character intrinsic operator	字符内在运算符
character literal constant	字符字面常量
character R301	字符

character relational intrinsic operation	字符关系内在运算
character sequence structure	字符顺序结构
character sequence type	字符顺序类型
character storage unit	字符存储单元
character string	字符串
character string edit descriptor	字符串编辑描述符
character type	字符类型
characteristics of a procedure	过程的特征
char-constant R309	字符常量
char-expr R726	字符表达式
char-initialization-expr R731	字符初始化表达式
char-length R508	字符长度
char-literal-constant R420	字符字面常量
char-selector R506	字符选择符
char-string-edit-desc R1016	字符串编辑描述符
char-variable R605	字符变量
CLOSEstatement	CLOSE 语句
close-spec R908	关闭说明
close-stmt R907	关闭语句
collating sequence	理序序列
comment	注解
common block storage sequence	公用块存储序列
common blocks	公用块
COMMON statement	COMMON 语句
common-block-object R549	公用块对象
common-stmt R548	公用语句
complex type	复型
complex-literal-constant R417	复字面常量
component-array-spec R428	成分数组说明

component-attr-spec R427	成分属性说明
component-decl R429	成分声明
component-def-stmt R426	成分定义语句
components	成分
computed-goto-stmt R837	计算 goto 语句
concatenation	并置
concat-op R712	并置运算符
conformable	可符合的
connected	连接的
connect-spec R905	连接说明
constant	常量
constant expression	常量表达式
constant R305	常量
constant subobject	常量子对象
constant-subobject R702	常量子对象
CONTAINS statement	CONTAINS 语句
contains-stmt R1225	包含语句
continue-stmt R841	继续语句
Control characters	控制字符
control edit descriptor	控制编辑描述符
control information list	控制信息表
control-edit-desc R1010	控制编辑描述符
create a file	创建(一个)文件
current record	当前记录
currently allocated	当前已分配的
cycle-stmt R834	循环中断语句
d R1008	整字面常量 d
data edit descriptor	数据编辑描述符
data entity	数据实体

data object	数据对象
data object reference	数据对象引用
DATA statement	DATA 语句
data transfer input statement	数据传输输入语句
data transfer output statement	数据传输输出语句
data type	数据类型
data—edit—desc R1005	数据编辑描述符
data—i—do—object R536	数据隐 do 对象
data—i—do—variable R537	数据隐 do 变量
data—implied—do R535	数据隐 do
data—ref R612	数据引用
data—stmt R529	数据语句
data—stmt—constant R533	数据语句常量
data—stmt—object R531	数据语句对象
data—stmt—repeat R534	数据语句重复因子
data—stmt—set R530	数据语句集
data—stmt—value R532	数据语句值
DEALLOCATE statement	DEALLOCATE 语句
deallocate—stmt R631	去分配语句
declaration	声明
declaration—construct R207	声明构造
default character	默认字符的
default complex	默认复的
default integer	默认整的
default logical	默认逻辑的
default real	默认实的
default—char—expr R727	默认字符表达式
default—char—variable R606	默认字符变量
default—int—variable R608	默认整变量

default—logical—variable R604	默认逻辑变量
deferred—shape array	迟形数组
deferred—shape—spec R517	迟形说明
defined	定义的
defined assignment statement	定义的赋值语句
defined binary operation	定义的二元运算
defined operation	定义的运算
defined unary operation	定义的一元运算
defined—binary—op R724	定义的二元运算符
defined—operator R311	定义的运算符
defined—unary—op R704	定义的一元运算符
definition	定义
delete a file	删除(一个)文件
deleted features	被删除的功能
Delimiters	界限符
derived type	导出类型
derived—type intrinsic assignment statement	导出类型内在赋值语句
derived—type—def R422	导出类型定义
derived—type—stmt R424	导出类型语句
digits	数字
digit—string R402	数字串
DIMENSION attribute	DIMENSION 属性
dimension—stmt R525	维数语句
direct access	直接访问
direct acces input/output statement	直接访问输入/输出语句
disassociated	分离
DO termination	DO 终结
do—block R823	do 块
do—body R828	do 体
do—construct R816	do 构造

do—stmt R818	do 语句
do—term—action—stmt R829	do 终结动作语句
do—term—shared—stmt R833	do 终结共享语句
double precision real	双精度实型
do—variable R822	do 变量
dummy procedure	虚过程
dummy—arg R1221	虚元
e R1009	整字面常量 e
edit descriptor	编辑描述符
element sequence	元素序列
elemental	初等的
elemental function	初等函数
elemental reference	初等引用
elemental subroutine	初等子程序
else—if—stmt R804	else—if 语句
else—stmt R805	else 语句
elsewhere—stmt R742	elsewhere 语句
END statement	END 语句
end—block—data—stmt R1112	块数据结束语句
end—do R824	do 结束
end—do—stmt R825	do 结束语句
endfile record	文件结束记录
endfile—stmt R920	文件结束语句
end—function—stmt R1218	函数结束语句
end—if—stmt R806	if 结束语句
ending point	终点
end—interface—stmt R1203	接口结束语句
end—module—stmt R1106	模块结束语句
end—of—file condition	文件结束条件
end—of—record condition	记录结束条件
end—program—stmt R1103	程序结束语句
end—select—stmt R811	选择结束语句
end—subroutine—stmt R1222	子程序结束语句
end—type—stmt R425	类型结束语句

end-where-stmt R743	where 结束语句
entity-decl R504	实体声明
ENTRY statement	ENTRY 语句
entry-stmt R1223	入口语句
EQUIVALENCE statement	EQUIVALENCE 语句
equivalence-object R547	等价对象
equivalence-set R546	等价集
equivalence-stmt R545	等价语句
equiv-op R722	等值运算符
equiv-operand R717	等值操作数
executable program	可执行程序
executable statement	可执行语句
executable-construct R215	可执行构造
executable-program R201	可执行程序
execution cycle	可执行循环中断
execution-part R208	执行部分
execution-part-construct R209	执行部分构造
exist	现存
exit-stmt R835	出口语句
explicit	显式的
explicit-shape array	显形数组
explicit-shape-spec R513	显形说明
exponent R416	指数
exponent-letter R415	指数字母
expr R723	表达式
expression	表达式
extended-intrinsic-op R312	扩展的内在运算符
extension operation	扩充运算
extension operator	扩充运算符
extent	维展
EXTERNAL attribute	EXTERNAL 属性
external file	外部文件
external procedure	外部过程
EXTERNAL statement	EXTERNAL 语句

external subprogram	外部辅程序
external unit	外部部件
external — file — unit R902	外部文件部件
external — stmt R1207	外部语句
external — subprogram R203	外部辅程序
field	字段
field width	字段宽度
file	文件
file connection statements	文件连接语句
file inquiry statement	文件查询语句
file positioning statements	文件定位语句
file — name — expr R906	文件名表达式
fixed source form	固定源程序形式
format control	格式控制
format R913	格式
format — item R1003	格式项
format — specification R1002	格式说明
format — stmt R1001	格式语句
formatted input/output statement	格式输入/输出语句
formatted record	(有)格式记录
Fortran character set	Fortran 字符集
free source form	自由源程序形式
function	函数
function reference	函数引用
function subprogram	函数辅程序
function — reference R1209	函数引用
function — stmt R1216	函数语句
function — subprogram R1215	函数辅程序
generic identifier	类属标识符
generic interface	类属接口
generic name	类属名
Generic names	类属名
generic — spec R1206	类属说明
global entity	全局实体

goto—*stmt* R836	goto 语句
Graphic characters	图形字符
hex—*constant* R410	十六进制常量
hex—*digit* R411	十六进制数字
host	宿主
host association	宿主结合
host scoping unit	宿主作用域单位
IF construct	IF 构造
IF statement	IF 语句
if—*construct* R802	if 构造
if—*stmt* R807	if 语句
if—*then*—*stmt* R803	if—then 语句
imaginary part	虚部
imag—*part* R419	虚部
implicit	隐式的
IMPLICIT statement	IMPLICIT 语句
implicit—*part* R205	隐式部分
implicit—*part*—*stmt* R206	隐式部分语句
implicit—*spec* R541	隐式说明
implicit—*stmt* R540	隐式语句
inactive	非活跃的
INCLUDE line	INCLUDE 行
initial point	初始点
initialization expression	初始化表达式
initialization—*expr* R730	初始化表达式
inner—*shared*—*do*—*construt* R832	内共享 do 构造
Input statement	输入语句
input—*item* R914	输入项
inquire by file	按文件查询
inquire by output list	按输出表查询
inquire by unit	按部件查询
inquire—*spec* R924	查询说明
inquire—*stmt* R923	查询语句
inquiry function	查询函数

instance	事例
int—constant R308	整型常量
integer constant expression	整型常量表达式
integer type	整型
INTENT attribute	INTENT 属性
intent—spec R511	意向说明
intent—stmt R519	意向语句
interface	接口
interface body	接口体
interface—block R1201	接口块
interface—body R1204	接口体
interface—stmt R1202	接口语句
internal procedure	内部过程
internal subprogram	内部辅程序
internal unit	内部部件
internal—file—unit R903	内部文件部件
internal—subprogram R211	内部辅程序
internal—subprogram—part R210	内部辅程序部分
int—expr R728	整型表达式
int—initialization—expr R732	整型初始化表达式
int—literal—constant R404	整字面常量
intrinsic	内在的
intrinsic assignment statement	内在赋值语句
INTRINSIC attribute	INTRINSIC 属性
intrinsic binary operation	内在二元运算
intrinsic function	内在函数
intrinsic operation	内在运算
intrinsic procedure	内在过程
INTRINSIC statement	INTRINSIC 语句
intrinsic type	内在类型
intrinsic unary operation	内在一元运算
intrinsic—operator R310	内在运算符
intrinsic—stmt R1208	内在语句
int—variable R607	整变量

io-control-spec R912	输入输出控制说明
io-implied-do R916	输入输出隐 do
io-implied-do-control R918	输入输出隐 do 控制
io-implied-do-object R917	输入输出隐 do 对象
io-unit R901	输入输出部件
iteration count	重复计数
k R1011	带正负号整字面常量 k
keyword	关键词
keyword R1212	关键词
kind	种别
kind type parameter	种别类型参数
kind-param R405	种别参数
kind-selector R505	种别选择符
label R313	标号
label-do-stmt R819	标号 do 语句
left tab limit	左制表限
length	长度
length type parameter	长度类型参数
length-selector R507	长度选择符
letters	字母
letter-spec R542	字母说明
level-1-expr R703	1 级表达式
level-2-expr R707	2 级表达式
level-3-expr R711	3 级表达式
level-4-expr R713	4 级表达式
level-5-expr R718	5 级表达式
Lexical tokens	词法记号
line	行
list-directed input/output statement	表控输入/输出语句
literal constant	字面常量
literal-constant R306	字面常量
local entity	局部实体
logical constant expression	逻辑常量表达式
logical intrinsic assignment statement	逻辑内在赋值语句

logical intrinsic operation	逻辑内在运算
logical intrinsic operator	逻辑内在运算符
logical type	逻辑型
logical−expr R725	逻辑表达式
logical−initialization−expr R733	逻辑初始化表达式
logical−literal−constant R421	逻辑字面常量
logical−variable R603	逻辑变量
loop	循环
loop−control 821	循环控制
lower−bound R514	下界
low−level syntax	低级语法
m R1007	整字面常量 m
main program	主程序
main−program R1101	主程序
many−one array section	多对一数组片段
masked array assignment	屏蔽数组赋值
mask−expr R741	屏蔽表达式
module	模块
module procedure	模块过程
module R1104	模块
module reference	模块引用
module subprogram	模块辅程序
modul-procedure-stmt R1205	模块过程语句
module-stmt R1105	模块语句
module-subprogram R213	模块辅程序
module-subprogram-part R212	模块辅程序部分
mult-op R709	乘法运算符
mult-operand R705	乘法操作数
n R1013	整字面常量 n
name	名(字)
name association	名结合
name R304	名(字)
named common blocks	有名公用块
named constant	有名常量

named file	有名文件
named-constant R307	有名常量
named-constant-def R539	有名常量定义
namelist input/output statement	名表输入/输出语句
NAMELIST statement	NAMELIST 语句
namelist-group-object R544	名表组对象
namelist-stmt R543	名表语句
Names	名(字)
name-value-subsequences	名值子序列
next effective item	下一有效项
next record	下一记录
nonadvancing input/output statement	非推进式输入/输出语句
nonblock-do-construct R826	非块 do 构造
nonexecutable statement	不可执行语句
nonlable-do-stmt R820	非标号 do 语句
nonnumeric types	非数值类型
not-op R719	非运算符
NULLIFY statement	NULLIFY 语句
nullify-stmt R629	取消语句
numeric constant expression	数值常量表达式
numeric intrinsic assignment statement	数值内在赋值语句
numeric intrinsic operation	数值内在运算
numeric intrinsic operator	数值内在运算符
numeric relational intrinsic operation	数值关系内在运算
numeric sequence structure	数值顺序结构
numeric sequence type	数值顺序类型
numeric storage unit	数值存储单元
numeric types	数值类型
numeric—exp R729	数值表达式
object	对象
obsolescent features	过时的功能
octal—constant R409	八进制常量
only R1109	仅(只)
OPEN statement	OPEN 语句

open—stmt R904 打开语句
operator 运算符
OPTIONAL attribute OPTIONAL 属性
optional—stmt R520 可选语句
or—op R721 或运算符
or—operand R716 或操作数
outer—shared—do—construct R830 外共享 do 构造
Output statement 输出语句
output—item R915 输出项
PARAMETER attribute PARAMETER 属性
PARAMETER statement PARAMETER 语句
parameter—stmt R538 参数语句
parent object 父对象
parent—string R610 父串
partially associated 部分结合
part—ref R613 部分引用
pause—stmt R844 暂停语句
pointer 指针
pointer association status 指针结合状态
POINTER attribute POINTER 属性
pointer—assignment—stmt R736 指针赋值语句
pointer—object R630 指针对象
pointer—stmt R527 指针语句
position 位置
positional arguments 位置变元
position—edit—desc R1012 位置编辑描述符
position—spec R922 位置说明
power—op R708 幂运算符
preceding record 前一记录
Preconnection 预连接
prefix R1217 前缀
present 呈现，提供
primary R701 初等量
PRINT statement PRINT 语句

printing	打印
print－stmt R911	打印语句
PRIVATE	PRIVATE
private－sequence－stmt R423	私用顺序语句
procedure	过程
procedure interface block	过程接口块
procedure reference	过程引用
processor	处理系统
processor dependent	依赖于处理系统
program name	程序名
program unit	程序单元
program－stmt R1102	程序语句
program－unit R202	程序单元
PUBLIC	PUBLIC
r R1004	整字面常量 r
range	范围
rank	秩
READ statement	READ 语句
reading	读
read－stmt R909	读语句
real part	实部
real type	实型
real－literal－constant R413	实字面常量
real－part R418	实部
record	记录
record number	记录号(记录个数)
RECURSIVE	RECURSIVE(一个关键词)
reference	引用
relational intrinsic operation	关系内在运算
relational intrinsic operator	关系内在运算符
rel－op R714	关系运算符
rename R1108	更名
repeat specification	重复说明
representable character	可表示字符

representation method 表示方法
representation methods 表示方法
restricted expression 受限表达式
result variable 结果变量
RETURN statement RETURN 语句
return-stmt R1224 返回语句
rewind-stmt R921 反绕语句
SAVE attribute SAVE 属性
saved object 保留对象
saved-entity R524 保留实体
save-stmt R523 保留语句
scalar 标量
scalar factor 标量因子
scope 作用域
scoping unit 作用域单位
section-subscript R618 片段下标
select-case-stmt R809 选择分情况语句
sequence 序列
SEQUENCE statement SEQUENCE 语句
sequence structure 顺序结构
sequence type 顺序类型
sequential access 顺序访问,顺序存取
sequential access input/output statement 顺序访问输入/输出语句
set of allowed access methods 允许访问方法组(集)
set of allowed actions 允许动作组(集)
set of allowed forms 允许形式组(集)
set of allowed record lengths 允许记录长度组(集)
shape 形状
shape conformance 形状符合性
share 共享
shared-term-do-construct R831 共享终结 do 构造
sign R406 正负号
signed-digit-string R401 带正负号的数字串
signed-int-literal-constant 403 带正负号的整字面常量

sign-edit-desc R1014	正负号编辑描述符
signed-real-literal-constant R412	带正负号的实字面常量
significand R414	有效数
size	大小
size of a common block	公用块的大小
size of a storage sequence	存储序列的大小
source forms	源程序形式
special characters	特殊字符
specific interface	特定接口
Specific names	特定名
specification expression	说明表达式
specification-expr R734	说明表达式
specification-part R204	说明部分
specification-stmt R214	说明语句
standard module	标准模块
standard-conforming program	符合标准的程序
starting point	始点
statement	语句
statement entity	语句实体
statement function	语句函数
statement keyword	语句关键词
statement label	语句标号
stat-variable R623	状态变量
stmt-function-stmt R1226	语句函数语句
stop-code R843	停代码
stop-stmt R842	停语句
storage associated	存储结合的
Storage association	存储结合
storage sequence	存储序列
storage unit	存储单元
stride	跨距
stride R620	跨距
structure	结构
structure component	结构成分

structure constructor	结构构造符
structure—component R614	结构成分
structure—constructor R430	结构构造符
Subobject	子对象
subobject designator	子对象指定符
subobject R602	子对象
subroutine	子程序
subroutine subprogram	子程序辅程序
subroutine—stmt R1220	子程序语句
subroutine—subprogram R1219	子程序辅程序
subscript R617	下标
subscript—triplet R619	下标三元组
substring	子串
substring R609	子串
substring—range R611	子串范围
Syntax rules	语法规则
TARGET attribute	TARGET 属性
target R737	目标
target—stmt R528	目标语句
terminal point	终止点
totally associated	整体结合
transformational functions	变换函数
type declaration statement	类型声明语句
type parameter	类型参数
type specifier	类型说明符
type—declaration—stmt R501	类型声明语句
type—param—value R509	类型参数值
type—spec R502	类型说明
ultimate components	终极成分
undefined	无定义的
underscore R303	下划线
unformatted input/output statement	无格式输入/输出语句
unformatted record	无格式记录
unit	部件、单元

unspecified storage unit	未说明的存储单元
upper—bound R515	上界
use associated	使用结合的
Use association	使用结合
USE statement	USE 语句
use—stmt R1107	使用语句
value separator	值分隔符
variable	变量
variable R601	变量
vector subscript	向量下标
vector—subscript R621	向量下标
w R1006	整字面常量 w
where—construct R739	where 构造
where—construct—stmt R740	where 构造语句
where—stmt R738	where 语句
whole array	完整数组
whole array named constant	完整数组有名常量
whole array variable	完整数组变量
WRITE statement	WRITE 语句
write—stmt R910	写语句
writing	写

附录三 缩减的功能

1. 已删除的功能

已删除的功能是Fortran77 中那些冗余的以及被认为几乎不再使用的功能。在国家标准中,已删除的功能的清单是空的。

2. 过时的功能

过时的功能是Fortran77 中那些冗余的、并且Fortran90 中有更好的方法可用的功能。过时的功能包括:

(1)算术 IF——用 IF 语句或 IF 构造代替。

(2)实型及双精度型 DO 控制变量以及 DO 循环控制表达式

——用整型代替。

(3)共享的 DO 终结以及在除 END DO 或 CONTINUE 以外的语句上的终结——对每个 DO 语句都用一个 END DO 或 CONTINUE 语句作为终结。

(4)从 IF 块外面到相应 END IF 语句的转移——用转移到后随 END IF 的语句代替。

(5)交错返回——见 a.

(6)PAUSE 语句——见 b.

(7)ASSIGN 及赋值 GO TO 语句——见 c.

(8)赋值的 FORMAT 说明符——见 d.

(9)*c*H 编辑描述符——见 e.

a. 交错返回

交错返回把标号引入到一个变元表中,以允许在返回时由被调用过程去控制调用过程的执行。同样的效果能通过一个在返回时能用于计算 GO TO 语句或 CASE 构造的返回码来达到。这就避免了变元结合中语法及语义上的不正规性。

```
    CALL SUBR_NAME(X,Y,Z,*100,*200,*300)
```

可由下面的程序代替:

```
CALL SUBR_NAME(X,Y,Z,RETURN_CODE)
SELECT CASE(RETURN_CODE)
  CASE(1)
  ...
  CASE(2)
  ...
  CASE(3)
  ...
  CASE DEFAULT
  ...
END SELECT
```

b. PAUSE 语句

执行一个 PAUSE 语句后,需要操作员或系统特定的干预,以

重新执行。在大多数情况下，使用一个等待某些输入数据的适当的READ 语句就可以得到同样的功能，因而，此方法同样有效，且是一种更可移植的方法。

c. ASSIGN 及赋值 GO TO 语句

ASSIGN 语句允许把一个标号动态地赋给一个整型变量，且赋值 GO TO 语句允许通过此变量进行“间接转移”。这就损害了程序的可读性，当此整型变量也被用于算术运算时尤其如此。整型变量的两种完全不同的用法可能成为一种不易发现的出错来源。

这些语句曾普遍地用于模拟内部过程，而现在已能对这些内部过程直接编码了。

d. 赋值的 FORMAT 说明符

ASSIGN 语句也允许把格式语句的标号动态地赋给一个整型变量，然后，此变量可用作 READ、WRITE 或 PRINT 语句中的格式说明符。这就损害了可读性。允许非一致的使用整型变量，并且可能成为一种不易发现的出错来源。

这一功能可以通过字符型的变量、数组及常量得到。

e. H 编辑

这个编辑描述符可能成为一种出错来源。同样的功能可以用字符型常量编辑描述符得到。

附录四 软盘说明

如果您购买了本书的运行软盘，并且您的微型计算机上配有Fortran 90 编译程序，您可根据每个问题调入相应的文件做下列的练习：

(1)比较下列三个程序的运行结果，注意不同的 PRINT 语句表示法。(file 1)

(2)体会以下输入/输出语句的用法。(file 2)

(3)字符串的输入/输出。(file 3)

(4)简单算术运算的程序写法。(file 4)

(5)求二次方程根的程序编制。自己给出具体值运行。(file 5)

(6)求二次方程复根的程序编制。(file 6)

(7)对$f=(1/2\pi)\times(g/L)^{1/2}$，$T=1/f$ 编写程序，其中$g=9.80665$。对给定的程序在运行中给出查错消息，应如何改此程序，试修改它，并再次运行，直至找出全部问题可正确运行时为止。(file 7)

(8)在你做了上题之后，请比较你修改后的程序和以下程序的差别。(file 8)

(9)计算$h=(R_E)/(1-V^2/2gR_E)$，$V_{final}=(V^2-2gR_E)^{\frac{1}{2}}$，其中$g=9.80$。$R_E=6.366\times10^6$. (file 9)

(10)用税率表计算应交纳的税款(用 IF 语句)。(file 10)。税率表为：

收入多于	收入少于	应纳税款
￥0	￥17,850	收入的 15%
￥17,850	￥43,150	￥2,677.50 加上超过￥17,850 部分的 28%
￥43,150	￥81,560	￥9,761.50 加上超过￥43,150 部分的 33%

(11)仿照以下程序，把相应数值改为 45，75 和 95 再运行你自己的程序。(file 11)

(12)求 2 的从 0 到 1000 的幂级数的程序(用 DO 语句)。(file 12)

(13)若一个文件每行包含一个整数，除文件最后一行的整数是负的以外，其它各行都是正的，以下程序为读文件并计算这些数的平均值。用 DO 语句和 CYCLE 语句各编一个程序，并比较之。(file 13)

(14)用倒计数法编写流行歌曲“墙上有 99 瓶啤酒”的程序。(file 14)

(15)用梯形法编制$\int_0^{\pi}\sin x dx$的求近似值程序。(file 15)

(16)下列程序所产生的输出是什么？(file 16)

(17)编写把任意三个数读入，按升序排列并打印出来的程序。(file 17)

(18)用子程序方法编写把任意三个数读入，按升序排列并打印出来的程序(子程序是把两个变量进行交换)。(file 18)

(19)打印函数 $f(x)=[1+1/x]^x$，当 $x=1,10,100,\cdots,10^{10}$时的函数值的程序。(file 19)

(20)用函数方法编写求 $f(x)=[1+1/x]^x$，当 $x=1,10,100$ $\cdots,10^{10}$时的函数值的程序。(函数表示 $f(x)=[1+1/x]^x$)。(file 20)

(21)编写把任意三个数读入，按升序排列并打印出来的程序(每部分都用子程序表示，交换也用子程序，再用主程序把它们连在一起)。(file 21)

(22)求两个骰子掷出的点数和为 7 或 11 的概率(使用内部函数 RANDOM，并编写一个函数，表示一个值小于等于 HIGH 而大于等于 LOW)。(file 22)

(23)问题同上，RANDOM _ INT 函数加一变元指明生成的伪随机数的个数。(file 23)

(24)使用 PARAMETER 属性编写一个简单的代替字符字面常量的程序。(file 24)

(25)在 PRINT 语句中把一个字符串赋给一个字符变量以代替执行包含不同消息的交替的 PRINT 语句。(file 25)

(26)下列程序使你在一次不同种属的一个字符可以确定其理序。(file 26)

(27)令计算机按照输入提供的次序打印字符串。(file 27)

(28)令计算机按照输入提供的次序两两打印字符串(每对的首字符重复前面的末字符)。(file 28)

(29)把输入的字词变成复数表示的程序。(file 29)

(30)用子程序方法编写列出文章中所用的全部字词。(file 30)

(31)用子程序方法编写列出文章中所用的全部字词(改变子程序写法，与上题比较之)。(file 31)

(32)搜寻文章中用了些什么字词。(file 32)

(33)求文章中所用字母的出现频率(第一个程序是说明思路的,第二个程序才是真正的运行程序)。(file 33)

(34)用内部子程序方法求文章中所用字词的平均长度。(file 34)

(35)求文章中所用字母的出现频率(前两个程序是说明思路的,第三个程序才是实现第二个思路的真正的运行程序)。(file 35)

(36)查看文章是否是回文,若不是回文,查前面第 5 个字符和从后数第 5 个字符是什么字符。(file 36)

(37)用模块方法测试矩阵的各种运算操作,请仿照 MATRIX _ SUM 函数编写 MATRIX _ PROD 函数并把此函数填入模块中,以便运行本程序。(file 37)

(38)用接口块方法测试矩阵的各种运算操作。请仿照 MATRIX _ SUM 函数编写 MATRIX _ PROD 函数并把此函数填入模块中,并仿照 INTERFACE OPERATOR(+)编写矩阵乘法,即 INTERFACE OPERATOR(*),以便运行本程序(运行时接口要插入主 程序中,可参照 40 题自己做)。(file 38)

(39)越南河内的一座庙前有三支旗杆,建庙时第一支旗杆上放了 64 个金盘,最大的金盘在最下面,越往上放越小,最小的金盘在最上面。庙里的和尚要把金盘从第一支旗杆移至第二支旗杆再移至第三支旗杆。当全部移好后,就是世界末日的来临。移动的规则是:

(1)金盘从一支旗杆移到另一支旗杆时,一次只能移一个金盘。

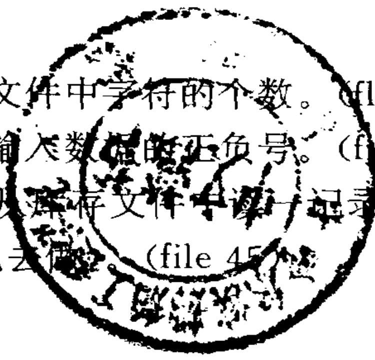
(2)在同一支旗杆处,较大的金盘永远不允许在较小的金盘的上面。(file 39)

(40)测试函数积分的精确值与近似值。(file 40)

(41)已有一数值表,现读入一批数据,按序插入表内。(file 41)

(42)从左到右打印树形节点的值。(file 42)

378049



(43)检测文件中字符的个数。(flie 43)

(44)检测输入数据的正负号。(file 44)

(45)用户从库存文件中读一记录，检查库存中给定零件的量(参照以下情况去做)。(file 45)