Walter S. Brainerd Charles H. Goldberg
Jeanne C. Adams

Fortran 90 编程指南

(第三版)

袁 驷 叶康生 译

CHEP
高等教育出版社

Springer
施普林格出版社

Fortran 90 编程指南

(第三版)

Walter S. Brainerd　Charles H. Goldberg

Jeanne C. Adams　著

袁　驷　　叶康生　译

CHEP
高等教育出版社

Springer
施普林格出版社

图书在版编目(CIP)数据

Fortran 90 编程指南(第三版)/(美)沃尔特(Walter, S.)等著；袁驷，叶康生译 -北京：高等教育出版社；海德堡：施普林格出版社，2000.6

ISBN 7-04-007937-2

I.F… II.①沃… ②袁… III.Fortran 语言-程序设计-教材 IV. TP312

中国版本图书馆 CIP 数据核字(2000)第 60861 号

责任编辑：赵天夫　　**封面设计**：张　楠　　**责任印制**：陈伟光

Fortran 90 编程指南(第三版)

袁 驷　叶康生　译

出版发行	高等教育出版社 施普林格出版社		
社　　址	北京市东城区沙滩后街 55 号	**邮政编码**	100009
电　　话	010-64054588	**传　　真**	010-64014048
网　　址	http://www.hep.edu.cn		
经　　销	新华书店北京发行所		
印　　刷	北京民族印刷厂		
开　　本	787×1092　1/16	**版　　次**	2000 年 6 月第 1 版
印　　张	20	**印　　次**	2000 年 6 月第 1 次印刷
字　　数	500 000	**定　　价**	26.50 元

前　言

Fortran 语言的新标准(非正式地称为 Fortran 90)具有很多优秀的特性，有助于程序员编写效率高、移植性强且易于维护的程序。由于 Fortran 90 包含了全部的 Fortran 77 语言，因此程序员可以采用适于自己的进度逐步学习新引进的语言特性，而且老的 Fortran 程序在 Fortran 90 下仍然可以运行。譬如，科学计算的程序员可以首先将精力集中于数组运算的新功能，待以后再去学习如何使用模块这一新的全局数据特性对数据类型进行封装。

《Fortran 90 编程指南》在章节组织上照顾到两方面：既可以从头到尾依序通读，也可以只阅读某些中间章节来学习某些特定的内容。在尽可能合理的范围内，每个特定内容的所有有关素材都安排在一起，使得本书不仅适于用作教材，而且也适于用作参考手册。

从最简单的语言结构开始，书中对 Fortran 编程语言中所有重要的语言特性都配有程序实例。在一定程度上，本书内容多集中于 Fortran 90 编程语言中的新特性，因为在通常情况下这些新特性为具体的编程任务提供了最好的实现手段。很多程序实例的风格和细节内容的选取都意在指导读者去掌握先进的编程技巧，以便能够编制出可读性好、可移植性强以及有效率的 Fortran 程序。

书中采用实例研究的方式来说明 Fortran 90 的语言功能的实际应用以及如何组装成完整的程序。书中提供了一些简单的习题以使读者能够练习学过的知识内容。

本书中一个与众不同的特点是在第一章中对编写一个完整 Fortran 程序所必需的基本语言功能作了完整的讨论，包括：Fortran 源程序的格式、数据类型、简单表达式和赋值，以及简单的输入和输出。在余下的章节中，则对控制结构、过程、数组、字符串、数据结构和导出类型、模块、以及指针变量作了详细讨论。

Fortran 语言从一开始就有很强的输入输出功能；但是，这部分内容在很多书中未能得到充分展示，原因是相比之下这部分语言功能稍有难度，而且不太令人感兴趣。由于这些功能在程序产品中非常重要，所以本书用第九章一个章节对 Fortran 中优秀的输入输出功能作了充分的讨论。

Fortran 是一个古老的语言。随着语言的进化，很自然地，某些语言特性在多数情况下可以由更好的等效语言功能所代替，因而成为冗余的特性。第十章讨论了很多这样的语言特性。这些冗余的语言特性在编写新程序或改写老程序时应尽可能少用。但当必须要维护一个老程序时，就需要对这些语言特性有足够的了解以便能对其进行修改，因此书中保留了对这些冗余特性的介绍。

本书并不要求读者具备 Fortran 77 的知识。然而，脚注中对 Fortran 77 和 Fortran 90 之间的一些重要区别作了说明；Fortran 77 程序员对此会有特殊的兴趣。熟悉 Fortran 77 的读者通过浏览第一章中的程序实例便可了解种别参数、新的源程序格式以及声明，然后便可立即跳到第二章。

附录 A 和 B 给出了很多固有函数和有关语言的完整的语法规则，但是偶然仍会感到某个语言特性的功能模糊费解。此时，应去查询由 International Standards Organization 和 American National Standards Institute 出版的正式标准，或者参阅 Adams、Brainerd、Martin、Smith 和 Wagener 编著的由 McGraw-Hill (1992)出版的《*The Fortran 90 Handbook*》一书。

第二版

第二版中较大的改变有：

- 所有程序均通过 NAGWare f90 编译器的调试运行，因此应含有极少的错误。
- 彻底地重写了与过程和模块有关的章节。
- 更改了第十章的名称，使得与其内容更加贴切。
- 某些语言特性，如某些冗余的输入输出格式编辑描述符，被移到第十章。外部过程作为一个主要特性也被移到第十章。采用内部过程或模块过程要简单得多，而且将一个外部过程转换为一个模块过程是非常简单的，这样做还可以使编译器检验变元的相容性。

将素材划分为前九章和第十章的思路也许需要一点说明。前九章相当全面地描述了一个功能完备的 Fortran 90 的子集语言。该子集语言的选取由我们感到最有用、最有效率且最可靠的 Fortran 90 的语言特性构成，这些语言特性趋于产生最富有可读性、可维护性和效率的程序。多数情况下，对这些特性的描述是自我完备的。

第十章包含的一些语言特性，尽管曾经在 Fortran 中非常有用(甚至在某些情况下别无其他选择)，但现在与前九章中的语言特性相比，大都变成了冗余的二等“公民”。多数情况下，前九章中的新特性提供了至少与第十章中冗余特性完全相当的功能。某些冗余特性正在变得(或已经变得)同 Fortran 的其他特性不协调，在应用上显得笨拙且不成体系。除了在第十章中偶尔的简略提及之外，书中没有给出第十章与其他章节中语言特性之间的相互作用的规则。譬如，第九章中关于内部文件一节并没提及将内部文件与名字表共用是不允许的，尽管这个提法是正确的而且是将名字表放在第十章中的原因之一。

当然，一个好的 Fortran 程序员需了解完整的 Fortran 语言。但是，如果只用本书中介绍的语言特性去编写新程序的话，则对于其余部分语言特性来讲，能够读懂就比能够应用于编程更为重要。如果读者已经熟悉 Fortran 77，则可在 Fortran 90 的系统下继续原来的编程。如果读者是一个新的 Fortran 程序员，则应该首先学习掌握书中介绍的 Fortran 90 特性，而其余的特性可待以后根据需要再去学习。

第三版

Fortran 语言在某些人看来显得过时的因素之一是程序代码采用大写字母来写。这对于 Fortran 77 程序来讲是必须的，否则就难以保证可移植性。

Fortran 90 标准允许采用大写字母，但并不要求编译器只接受大写字母。前两版中采用了大写字母，以便程序可以在任何 Fortran 90 系统中均可运行。但是，就目前出现的众多的 Fortran 90 编译器来看，没有一个不接受小写字母的，因此有充分理由期望所有的 Fortran 90 编译器都将接受小写字母的程序代码。

因此，从第二版到第三版中的主要改变是所有程序都改成了小写字母。我们希望这将为源程序增加一点可读性，也使本书在面貌上显得更加现代。

有几位同仁指出了前两版中的一些错误，并提供了改进意见。其中，我们特别感谢 Hank Lauson、Brian Smith、Richard Maine、Mike Metcalf 和 Chris Smith。

Walter S. Brainerd
Charles H. Goldberg
Jeanne C. Adams
1995 年 7 月

第一章　Fortran 90 编程简介

学习程序语言的最佳途径是立即着手阅读和编写程序。如果使用计算机很方便，我们建议您模仿本章中简单的程序样例，试着编写并运行一些程序。除了本书，您或许还需要一套简短的指南，教给您如何进入您的计算设备并在其上运行您的程序。

1.1 用于计算和打印的程序

由于计算机擅长计算，而且 Fortran 也被设计成善于表达数值计算，所以学习 Fortran 90 可先用它进行一些通常要用手算或计算器完成的计算。本节讲述如何编写用于计算并将结果打印出来的程序。

1.1.1 简单计算

第一个例子是打印加法结果的程序。

```
program calculation_1
   print *, 84 + 13
end program calculation_1
```

程序 calculation_1 告诉计算机将 84 和 13 加起来，然后打印出其和：97。[1]当计算机运行该程序时，它准确地将两个数加起来，并打印出它们的和。运行后的输出结果类似如下所示。

```
run calculation_1

 97
```

1.1.2 run 命令

下面这一行

```
run calculation_1
```

不是 Fortran 的一部分，而是本地操作系统上的一条命令，该命令告诉系统编译(也就是翻译)和运行 calculation_1 程序。由于任何有经验的系统开发员都会设计出适当的风格，因此 run 命令在形式上可以有很多变化。通常命令行和程序文件名中包含字符 “f90” ，以和该系统上的其他语言区分开来。

1.1.3 缺省的打印格式

关键词 print 后的星号告诉计算机：对于打印结果，程序员没有指定具体的格式或版面编排。

---

1. Fortran 77 程序员请注意：标识符不超过 6 个字符的限制现在终于放宽为 31 个字符了。

因而，Fortran 系统将采用缺省的格式，也称作表控格式(9.8.19 节)，该格式被设计为在大多数情况下都尽可能的令人满意。Fortran 标准在缺省格式的设计中允许有一定的自由，所以您的输出可能和上面的例子稍有不同。

1.1.4 打印信息

如果您想要计算机准确地打印出您所指定的印刷字符，您可以将其括在引号(双引号)或撇号(单引号)中，[2]如程序 quotes 所示。引号在输出中不打印出来。

```
program qutoes
   print *, "84 + 13"
end program qutoes

run qutoes
 84 + 13
```

在 Fortran 程序中，包含在引号(双引号)或撇号(单引号)中的印刷字符序列为**字符串**。一个字符串既可以包含字母字符，也可以包含数值字符，还可以包含其他特殊字符，如标点符号和算术符号。

打印准确的字面字符和计算数值可得到更加易读的输出结果。

```
program calculation_1_v2
   print *,  "84 + 13 =", 84 + 13
end program calculation_1_v2

run calculation_1_v2

 84 + 13 = 97
```

在程序 calculation_1_v2(calculation_1 的第 2 个版本)，print 语句的列表中有两项，按照字面(但不包括定界引号)准确打印出的**字符常量** “84 + 13 =” ，以及按该表达式计算出来的数值。尽管这两项看起来相似，处理起来相当不同。用引号将字符串括起来将按字符的字面打印(包括 3 个空格字符)；然而，没有引号的相同表达式却需要先计算，然后打印其和。逗号用来分隔 print 语句列表中的各项。

1.1.5 program 语句

每个 Fortran 程序都以一个 program 语句开始。它包括关键词 program，后接一个由程序员选定的**程序名**。程序名必须以一个字母作为开始，并且至多包括 31 个字母、数字和下划线。除了对读者外，程序名 calculation_1 没有任何意义。

1.1.6 end 语句

end 语句以关键词 end 开始，后面可以接关键词 program，其后还可以再接一个程序名。每个 Fortran 程序必须有一个 end 语句作为结束语句。

---

2. 以前版本的 Fortran 只接受撇号为字符串定界符。

风格注释：别偷懒。在每个程序的 end 语句后要包含关键词 program 和程序名。

1.1.7 练习

1. 编写并运行一个打印您自己名字的程序。
2. 编写并运行一个程序，计算从 1 到 9 的整数的和，前面附一条简短信息，说明输出的是什么。
3. 当运行以下程序时，将得到怎样的输出结果？

```
program simple
   print *,  1, "and",   "equals", 1 + 1
end program simple
```

1.2 固有数据类型

Fortran 中有五个固有的数据类型：整型、实型、复型、逻辑型和字符型。每个数据类型都有一组可以表示的值以及对这些值的运算。我们已经列举了两种数据类型的应用例子。“84 + 13”(包括引号)是一个字符串常量，84 + 13 是一个值为整型的表达式。

以下的小节讨论了这五种固有类型以及在 Fortran 中这些类型的常量是如何表示的。

1.2.1 整型

整型用来表示整数的值。在 Fortran 中，整型常量的表示与它们通常的用法很相似。一个**整型常量**是一个只包括从 0 到 9 的一串数字，后面可能接一个下划线(_)和另一个整型常量或有名整型常量，用来指明种别参数，如 1.2.9 节所述。以下是整型常量的例子。

```
23  0  1234567  42_1  42_short  42_long
```

一个**带符号的整型常量**是一个前面带有一个 + 或 - 的整型常量。一个带符号的整型常量可以认为是一种带约束的表达式，通常可以用在整型常量允许用的地方。

每个 Fortran 系统必须具有至少一个整型种别，尽管同一种别在不同的计算机中其种别值可能不同。许多 Fortran 系统有几个整型种别，不同的种别值对应不同的可表达值的范围。

1.2.2 实型

Fortran 中，有两种形式的**实型常量**。第一种称为**位置型**，因为每个数字的值由相对的小数点的位置来决定。位置型实型常量包括一个整数，后跟一个小数点，然后是表示该值小数部分的一串数字，后面还可能接一个下划线和种别参数。假设 double 和 quad 是所用 Fortran 系统上所允许的实型种别的整型常量名(见 1.2.9 节)，以下为按位置型写出的实型常量。

```
13.5   0.1234567   123.45678   00.30_double    3.0
3.     12345.      .0          .1234567_quad
```

一个位置型的实型常量在小数点的左边可以没有数字，或者在小数点的右边没有数字，但是只有一个小数点不是合法的实型常量。

指数型的实数包括一个整数或一个位置型的实数，后跟一个字母 e 和一个可能会带符号的整数(没有种别参数)，然后可接一个下划线和种别参数。字母 e 表示“乘以 10 的多少次方”，e 后面的整数是 e 前面的数乘以 10 的方次。指数表示法对于很大或很小的数是很有用的。例如，2.3e5 表示 2.3 乘以 10 的 5 次方：2.3×10^5 或 $2.3\times100{,}000=230{,}000$。整数方次的前面可以包含一个负号或正号，如实型常量 2.3e-5 表示 2.3×10^{-5}，或 $2.3\times0.00001=0.000023$。另外两个例子：1e9_double 表示的是一千兆，具有种别参数 double；1e-3 表示千分之一。

每个 Fortran 系统必须有至少两个实型种别，[3]尽管它们的种别值、精度(粗略地讲是有效数字的位数)和指数范围可能因计算机而异。许多 Fortran 系统具有几个实型种别，不同的种别值对应不同的精度或值的范围。

1.2.3 复型

Fortran 中的**复型**用来表示数学中的复数。复数包括两个实数，通常写作 $a+bi$。第一个实数称为**实部**，第二个实数称为**虚部**。在 Fortran 中，**复型常量**写成两个(可能带符号)的整数或实数，用逗号分开，并且括在圆括号中。复型常量的例子有

```
(1, -1)
(3.14_double, -7)
(-1.0, 3.1e-27_quad)
```

如果复数的两部分是整型，复型常量的种别参数是缺省的复型种别。如果一部分是整型，另一部分是实型，该复型常量的种别参数是实型种别。如果两部分都是实型，该复型常量的种别参数与精度高的那一部分相同。

1.2.4 算术运算符

用来对两个数值(即，整型、实型、或复型)进行运算的运算符包括+、-、*、/和**。除了**，这些符号都有其通用的数学含义，分别指加法、减法、乘法和除法。两个星号表示指数：2**4 的值为 16，即计算 2 的 4 次方，或按数学表示为 2^4。符号 + 和 - 可用作一元运算符，分别表示等同运算和负值运算。

整型数除法总是切掉计算结果中的小数部分而产生一个整型结果。例如，由于 23/2 的数学结果是 11.5，Fortran 算术表达式

```
23.0/2.0
```

的值为 11.5，但是，表达式

```
23/2
```

的值为两个整型常量的商，值为 11。类似地，表达式

```
-23/2 和 23/(-2)
```

的值都是-11。

---

3. 这两种实型代表相应于 Fortran 77 中的 real 和 double precision 数据类型，但现在以种别参数的形式描述了。

1.2.5 关系运算符

数值可以用**关系运算符**来比较。各个关系运算符的两种形式在表 1-1 中给出。[4]

表 1-1 关系运算符

Fortran 90 形式		意义
<	.lt.	小于
<=	.le.	小于或等于
==	.eq.	等于
/=	.ne.	不等于
>=	.ge.	大于或等于
>	.gt.	大于

复数值只能用关系运算符 == 和 /= 进行比较。由于舍入误差，在许多情况下，用 ==或 /= 运算符来对实数或复数的值进行比较是不妥当的。在这种情况下，最好是检查它们是否近似相等。

关系运算符的结果是逻辑型的(1.2.7 节)。

1.2.6 混合表达式

在数学上，整数是实数的子集而实数又是复数的子集。因此，对两个数值进行比较的运算是有意义的，即使它们不是相同的 Fortran 类型。一个运算符的两个运算对象不一定非具有相同的数据类型；当它们不同时，在运算实施之前，将一个对象转换成另一个的类型。

如果一个是整型而另一个是实型，整型数将被转换为实型值；如果一个是实型而另一个是复型，实型数将被转换为复型值。例如，表达式

```
23.0/2
```

的值为 11.5，因为整型 2 被转换为一个实型值，然后执行两个实型数值之间的除法。

一个数值运算的两个运算对象也可以具有不同的种别参数值。在这种情况下，如果两个运算对象具有相同的类型，或者一个是实型一个是复型，结果具有运算中较高精度的种别参数。例如，如果种别 5 比种别 2 具有更高的精度，

```
1.0_2+3.0_5
```

的值是 4，种别参数值是 5。假设种别 4 比种别 2 或种别 3 的精度高，

```
1.1_4+(2.2_2,3.3_3)
```

的值是 3.3+3.3i，种别参数值为 4。如果一个运算对象是整型而另一个是实型或复型，运算结果的种别参数是实型或复型的种别参数。

1.2.7 逻辑型

Fortran 中的**逻辑型**用于表示两个真实值：“真”和“假”。一个**逻辑常量**不是.true.就是.false.。

用来对逻辑值进行运算的运算符有：.not.、.and.、.or.、.eqv.和.neqv.。除一元运算符.not.

4. 注意，Fortran 77 中的每个关系运算符在 Fortran 90 中都有新的等价表达形式。

外，它们都是二元运算符。每个逻辑运算符产生的值见表 1-2。

表 1-2 逻辑运算符的值

x_1	x_2	.not. x_2	x_1 .and. x_1	x_1 .or. x_2	x_1 .eqv. x_2	x_1 .neqv. x_2
true	true	false	true	true	true	false
true	false	true	false	true	false	true
false	true	false	false	true	false	true
false	false	true	false	false	true	false

举例来讲，

```
.false. .eqv. .false.
```

的值为真。

1.2.8 字符型

字符型用来表示字符串。**字符常量**是计算机中可以表示的、用引号(双引号)或撇号(单引号)来定界的字符序列。如果定界字符出现在字符串中，用两个定界符来表示，中间没有其他字符。一个字符常量前面可接一个种别参数和一个下划线。注意字符常量的种别参数(在下面的第二个例子中为 ascii)在常量前面，而不是像其他类型的常量那样，种别参数在常量后面。以下是字符常量的例子。

```
"Joan"
ascii_'John Q. Public'
"Don't tread on me."
'He said, "Don''t tread on me."'
```

只有一个产生字符结果的字符连接运算符，所用的符号是//，并且是一个二元运算符，其结果是包括第一个字符串以及后面的第二个字符串的一串字符。例如，"John Q." // "Public"的值是“John Q.Public”。注意在点之后没有空格，尽管本可以有的；"John Q. " // "Public"的值为“John Q. Public”。两个运算对象的种别参数必须相同。

关系运算符(1.2.5 节)可以用来比较字符串的值，这要借助字符的顺序值来进行(5.1.8 节和 5.1.11 节)。

1.2.9 种别参数

种别参数为机器中每个固有数据类型可能的不同表示提供了参数化的选择方法。如果程序员仔细的话，这将为数值精度和范围选择的可移植性提供了一种机制。对于字符数据类型，种别参数允许在程序中使用多种字符集，如日文、中文和化学符号等。

每个固有数据类型(整型、实型、复型、逻辑型和字符型)都有一个参数与其相连，称为**种别参数(kind parameter)**。种别参数试图为特定的数据类型指定一种机器表达。例如，一个编译执行系统可能具有三个实型种别，通常认为是单精度，双精度和四精度。

种别参数是一个整数，其值依赖于处理器，因而，种别参数 1、2、3 可能是单精度，双精度和四精度，或者在一个不同的系统中，同样是这些精度，所用的种别参数可能是 4、8 和 16。唯一的要求是必须具有至少两个实型和复型种别，表示缺省的实型和双精度；而且对于整型、逻辑

型和字符型数据至少要有一个种别。注意种别参数的值和决定数值精度和范围的字长无关。

您需要检查您所用的计算机系统的 Fortran 手册来确定每种类型的种别参数和缺省种别参数。在各种情况下，种别参数是可选的，因而，如果缺省类型足够适用的话，您可以始终采用缺省种别。

固有函数 selected_int_kind 和 selected_real_kind 可以用来为一个变量或有名常量选择一个合适的种别(1.3.10 节)。当程序对数值有特定的精度要求时，这些函数为程序的可移植性提供了方便；比如所要求的精度在一个机器上对应单精度，而在另一个机器上需要用双精度才行。详见 1.5.1 节。

对于逻辑数据类型，一个编译执行系统可能有不同的性质的表达：用一个单字(bit)存放一个逻辑值；或者由于寻址和指令的缘故，用一个字节(byte)存放一个逻辑值可能更为有效。这些表达可以由程序员通过指定一个种别参数值来选择。

当一个种别参数是另一个常量的一部分时，它可以是一个整型常量(没有符号)或一个有名整型常量(参数)。对于整型、实型和逻辑型常量，将它置于结尾的下划线字符(_)之后。

```
12345_4
1.345_2
.true._1
```

对于字符常量，它出现在最前面，后跟下划线：

```
ascii_ "abcde"
greek_ "αβγδε"
```

如 1.2.3 节所述，复型常量的种别由两个实型分量的种别决定。

1.2.10 练习

1. 将下面的实型数从位置记法转化为指数记法。

```
48.2613      .00241_4      38499.0
0.2717        55.0         7.000001_quad
```

2. 将下面的实型数从指数记法转化为位置记法。

```
9.503e2      4.1679e+10_double      2.881e-5
-4.421e2     -5.81e-2_8              7.000001e0
```

3. 编写一个程序，打印 .1+.2+.3+⋯+.9 的和。
4. 确定您的计算机系统上比缺省的实型种别的精度高的实型种别的种别值。
5. 对于大约一打不同的变量值，打印 selected_int_kind 和 selected_real_kind 的值，看哪些种别值在您所用的计算机上是有效的。查看您所用的 Fortran 手册，检验您的结果。
6. 编写一个程序，打印复数 (.1+.1i)+(.2+.2i)+(.3+.3i)+⋯+(.9+.9i) 的和。
7. 编写一个程序，打印以下每个表达式的逻辑值。

```
2 > 3
2 < 3
```

```
.1 + .1 = = .2
.5 + .5 / = 1.0
```

8. 编写一个程序，计算并打印您的所有名字的连接(例如：姓和名)。

1.3 变量和输入

编写一个程序进行计算，而不是用纸笔或计算器来得到答案，这样做的一个便利之处是当同种类型的问题再出现时，已经写出的程序可以再次使用。**变量**的使用为本节中的程序提供了重用的灵活性。1.1 节中的程序指示计算机对出现在 print 语句中的数值常量进行指定的算术运算。程序 add_2 计算输入的两个任意整数的和。要加的两个数并没有在程序本身中出现。相反，指定了两个整型变量 x 和 y 保存输入的值。由于 Fortran 语句既能对常量也能对变量进行操作，它们的和可以计算并打印出来。第一个例子的运行说明了如何用这个新程序计算 84 和 13 的和；此例曾在 1.1 节中用程序 calculation_1 计算过。

```
program add_2
   integer :: x, y
   read *, x
   print *, "Input data x:", x
   read *, y
   print *, "Input data y:", y
   print *, "x + y=", x + y
end program add_2

run add_2

 Input data x: 84
 Input data y: 13
 x + y = 97
```

在声明变量 x 和 y 将用来存储整数值之后，程序 add_2 告诉计算机从输入设备中读入一个数，称为 x，然后，读入另一个数，称为 y，最后打印出 x+y 的值。再加上两个 print 语句用以打印出输入的值便完成了程序 add_2。在该程序的执行中，作为 x 和 y 的值的两个数必须提供给计算机，否则计算机无法完成运行(1.3.3 节)。

1.3.1 变量的声明

类型语句出现在 program 语句和第一个可执行语句之间。每个声明包括一个关键词来指定 Fortran 的固有类型，后面跟两个冒号和用逗号分开的变量名表。例如，程序 add_2 用了类型声明

```
integer :: x, y
```

对应于 1.2 节介绍的整型、实型、复型、逻辑型和字符型常量，有整型、实型、复型、逻辑型和字符型变量。例如，在一个程序中，如果变量 q、t、k 是实型变量，n 和 b 是整型变量，以

下的语句行包括了必要的声明。[5]

```
real :: q, t, k
integer :: n, b
```

在表示数据类型的关键词后面跟一对圆括号，其中写上 kind=，后接一个种别参数值，可以将变量声明为具有某种特定的种别参数。例如，如果需要比您的系统中的缺省实型数具有更多的有效数字，并且与这种附加精度对应的种别为 2，则变量 dpq、x 和 long 可以采用以下声明将它们声明为具有附加精度的实数。

```
real (kind = 2) :: dpq, x, long
```

对于字符变量，关键词 character 后面跟一对圆括号，括号中有 len=，其后用一个整数表示字符串的长度，即字符串中的字符个数。如果字符变量有一个种别参数，而不是缺省值，也可以在同一个圆括号中指定。如果变量 name 是一个 20 个字符的字符串，可声明如下。

```
character (len = 20) :: name
```

另外，如果变量 name 的种别为 kanji (为一整型常量)，其声明可以是

```
character (len = 20, kind = kanji) :: name
```

风格注释：一个好的编程习惯是将 Fortran 程序中的每个变量都列在某个类型声明中。

当声明变量时，可以同时给变量赋一个**初值**。例如变量 count 可以用下面的语句声明为整型并置初值为 0。[6]

```
integer :: count = 0
```

变量 a、b、c 可以用下面的语句声明为实型并且分别置初值为 1.1，2.2 和 3.3：

```
real :: a = 1.1, b = 2.2, c = 3.3
```

用这种方法初始化的变量的值在程序执行过程中可以改变。[7]

1.3.2 隐含类型

在早期，Fortran 语言没有类型声明。根据变量名的第一个字母而建立的隐含规则自动为变量指定一个类型。而且，只允许用于整型和实型变量。变量名以字母 I–N 开始的变量是整型。所有其他变量为实型。尽管现代编程习惯于声明所有的变量，偶而或有意不声明的变量还是按变量名的第一个字母来分配缺省类型。以下语句可以关掉隐含类型

```
implicit none
```

当关掉隐含类型时，错误拼写或打错的变量名将被检查出来并标为错误(未声明的变量)；而若没

---

5. Fortran 77 的程序员将注意到类型声明中新的语法。旧形式的类型声明(参看第十章)仍被 Fortran 编译器所接受，但是在新的程序中使用新的形式是好的习惯。

6. 在 Fortran 77 中，数据块的初始化用一个独立的 data 语句 (参见 10.2.4 节)

7. 如果一个变量在一个子例子程序或函数中被给定一个初始值，则它的值将被保留，再次执行该子程序时也不会被重新初始化。

有关掉隐含声明，则它们将被接受为具有隐含声明的新变量，但是通常具有不期望的值或类型。

风格注释：每个程序和过程应该包括

```
implicit none
```

语句来关掉隐含类型。注意：在很短的程序中，打破这个类型规则是可接受的。

1.3.3 提供输入数据

在程序 add_2 的执行中，变量 x 和 y 的两个输入值 84 和 13 在计算机中没有出现。它们由用户键入。为了运行程序，假设要执行一个“run”命令对 Fortran 程序编译和运行，还假设输入数据文件名同程序名，但后面附上“_in”。因而，命令

```
run add_2
```

编译并运行程序 add_2，从文件 add_2_in 中读入输入数据。对于用来运行 add_2 的 Fortran 运行系统，必须准备一个名为 add_2_in 的输入文件。该文件包括以下两行

```
84
13
```

1.3.4 文件名

文件的命名规则取决于操作系统，而不是 Fortran 语言。常见的作法是生成一个命令文件 run，其文件命名遵循本书的规则；然而，不同的计算机系统上规则不同。例如，Fortran 90 程序文件名通常以字符 “.f90”作为后缀，而 Fortran 90 编译器通常取名为“f90”。在这种情况下，在 Unix 系统上的命令文件 run 可包括以下两行。

```
f90 -o $1 $1.f90
$1 < $1_in > $1_out
```

在 Uinx 中， $1 表示命令行中文件名 run 后面的第一个文本数据，也就是程序名 add_2。用 add_2 代替$1 后，第一行命令指示操作系统用编译器 f90 处理源程序文件 add_2.f90 并且将执行文件 (-o $1) 输出为 add_2。第二行命令指示操作系统执行 add_2($1)，而且用文件 add_2_in($1< _in)作为输入文件，用文件 add_2_out(> $1_out)作为输出文件。

1.3.5 输入数据的响应

和其他程序语言一样，在 Fortran 中，一个好的编程习惯是使用 print 语句对**输入数据给出响应**，以便计算过程中所用的数据可以在输出结果中记录下来。在程序 add_2 中，每一个 read 语句都后续一个对刚读入的输入数据的响应。

风格注释：使用打印语句对所有输入数据给出响应，是一个好的编程习惯。然而，某些时候遵循这个规则是不实际的，例如当有大量的数据输入时。

1.3.6 用不同的数据运行同一个程序

程序 add_2 包含响应语句，当用不同的输入数据运行同一程序时，它的重要性就显示出来了。输入数据的响应有助于判定答案与问题的对应。输入语句的其他重要用途后面还要提到。让我们演示另一个 add_2 的运行例子，这一次，输入两个不同的数相加。程序没有变化，因此我们不重复程序清单。这一次，程序 add_2 输入的数据文件包括下面两行。

```
4
7
```

执行程序的输出结果将是

```
run add_2

 Input data x: 4
 Input data y: 7
 x + y =11
```

程序 add_2 的最终打印语句包括变量 x 和 y。像这两个例子执行结果显示的那样，实际打印的是字符串常数“x + y = ”的值，后面跟着执行该打印语句那一时刻表达式 x + y 的值。

程序 add_2_reals 可以简单地通过将程序 add_2 变量声明中的关键词 integer 改为 real 而得到，这将导致变量 x 和 y 变为实型。程序 add_2_reals 可以用来对两个数作加法，而这两个数不必是整数。程序的执行还显示了输入数据可以是负值。要运行这个例子，程序的输入数据文件 add_2_reals_in 中包括两行

```
97.6
-12.9
```

程序文件包括下面几行：

```
program add_2_reals
   implicit none
   real :: x, y
   read *, x
   print *, "Input data x:", x
   read *, y
   print *, "Input data y:", y
   print *, "x + y=", x + y
end program add_2_reals
```

而且，run 命令产生了一个输出文件 add_2_reals_out。

```
run add_2_reals

 Input data x: 97.5999985
 Input data y: -12.8999996
 x + y = 84.6999969
```

某些 Fortran 系统习惯用指数形式打印实型数据。在这样的系统中，本例执行的结果更像如