

# Fortran 95 程序设计讲义<sup>1</sup>

潘建瑜

<sup>1</sup>本讲义仅供课堂教学使用



# 目 录

第一讲	Fotran 介绍 .....	1
1.1	Fortran 的发展历史 .....	1
1.2	Fortran 77 的不足 .....	4
1.3	Fortran 90 的新特性 .....	4
1.4	Fortran 95 的新特性 .....	5
1.5	Fortran 的优点 .....	6
1.6	Fortran 编译器 .....	7
1.6.1	Windows 操作系统下较流行的 Fortran 编译器 .....	7
1.6.2	Linux/Unix 操作系统下较流行的 Fortran 编译器 .....	8
1.6.3	Fortran 编译器使用推荐 .....	9
1.6.4	高性能程序库 .....	9
1.7	程序设计方法 .....	10
1.8	推荐参考资料 .....	10

第二讲	Fortran 编程基础	12
2.1	Fortran 基础知识	13
2.1.1	字符集	14
2.1.2	标识符	14
2.1.3	关键字	15
2.1.4	语句	15
2.1.5	书写格式	17
2.2	基本数据类型	19
2.3	常量与变量	20
2.3.1	常量	20
2.3.2	变量	21
2.3.3	变量的声明	21
2.3.4	DATA 初始化	25
2.3.5	NAMelist	25
2.4	表达式与赋值语句	25
2.4.1	表达式	25

2.4.2	赋值语句	26
2.5	类型转换	26
2.5.1	自动转换	26
2.5.2	显式转换	27
2.6	END 语句和 STOP 语句	27
2.7	内置过程 INTRINSIC PROCEDURES	27
2.8	输入输出	28
2.8.1	表控输出	28
2.8.2	表控输入	29
2.8.3	格式化输出	30
2.8.4	格式化输入	35
2.9	选择/分支结构	36
2.9.1	IF 结构	36
2.9.2	SELECT CASE 结构	38
2.10	循环语句	40
2.10.1	DO 结构	40

2.10.2	DO WHILE 结构	41
2.10.3	隐式 DO 循环	41
2.11	数组	41
2.11.1	一维数组	41
2.11.2	二维数组	45
2.11.3	多维数组	46
2.11.4	数组运算	46
2.11.5	可调数组	50
2.12	字符数据	51
2.12.1	字符变量声明	51
2.12.2	字符串操作	52
2.13	内部文件—数值与字符转换	54
第三讲	过程 PROCEDURE	55
3.1	主程序一般结构	55
3.2	自定义函数	57

3.3	子例程序	59
3.3.1	数组作为形参	61
3.3.2	字符串作为形参	63
3.3.3	函数或子例程序作为形参	64
3.3.4	关键字参数和可选参数	64
3.3.5	星号作为形参	66
3.4	递归函数和递归子例程序	66
3.5	内部过程 CONTAINS	69
3.6	PURE 过程	70
3.7	ELEMENTAL 过程	71
3.8	模块 MODULE	72
3.8.1	模块的定义	72
3.8.2	模块的使用 USE	73
3.8.3	USE 高级选项	74
3.9	接口块 INTERFACE	74

3.10	通用过程	77
3.11	使用多个文件	77
3.12	作用域	77
第四讲	派生结构和指针	78
4.1	派生数据类型	78
4.2	指针	79
4.2.1	指针的声明	79
4.2.2	指针的引用	79
4.2.3	指针与派生类型	79
4.2.4	指针与动态控制	79
4.2.5	指针与数组	79
4.2.6	指针与过程	79
第五讲	文件操作	80
5.1	文件的分类	80



5.2	文件读写 .....	80
5.3	打开文件 OPEN .....	81
5.4	关闭文件 CLOSE .....	83
5.5	文件状态查询 INQUIRE .....	84
5.6	读文件 .....	87
5.7	写文件 .....	88
5.8	矩阵读写举例 .....	88
5.9	文件定位 .....	91
5.10	ENDFILE 语句 .....	92
5.11	有名 I/O 列表 .....	92
第六讲	其他 .....	94
6.1	面向对象编程 .....	94

6.2	重载 .....	94
6.2.1	子程序重载 .....	94
6.2.2	运算符重载 .....	94
6.2.3	赋值号重载 .....	94
6.3	公共数据 .....	94
6.3.1	公共区 COMMON .....	94
6.3.2	共享存储 EQUIVALENCE .....	94
6.3.3	数据块 BLOCK DATA .....	94
6.4	好的编程习惯 .....	94
6.5	Fortran 内置函数和子例程序 .....	96
6.6	Fortran 90/95 语句 .....	106

# 第一章 Fortran 介绍

FORTRAN 或 Fortran，是“FORmula TRANslation”的缩写，即“公式翻译”。它是世界上最早出现的计算机高级程序设计语言，至今仍广泛应用于科学和工程计算领域。

## 1.1 Fortran 的发展历史

Fortran 语言是为了满足数值计算的需求而发展出来的。1953 年 12 月，IBM 工程师 John Warner Backus (1924.12.03–2007. 03.17) 因深深体会编写程序很困难，写了一份备忘录给董事长 Cuthbert Hurd，建议为 IBM704 系统设计全新的电脑语言以提升开发效率。当时 IBM 公司的顾问冯·诺伊曼 (John von Neumann, 1903.12.28–1957.02.08) 强烈反对，因为他认为不切实际而且根本不必要。但 Hurd 还是批准了这项计划。1957 年，IBM 公司开发出第一套 FORTRAN 编译器，并在 IBM704 电脑上运作通过。之后陆续推出了 FORTRAN II, III, IV, Fortran 66。直至 1978 年，ANSI (American National Standards Institute, 美国标准化协会) 正式公布了 Fortran 77。Fortran 77 是具有结构化特性的编程语言，在短时间内取得了巨大的成功，广泛地应用于科学和工程计算，几乎统治了数值计算领域。但这之后，Fortran 一直没有更新，直至 1991 年 Fortran 90 的出现。目前的最新标准是 Fortran 2008。

- 1951 年，Backus 开始着手开发 FORTRAN 语言。
- 1954 年，Backus 在纽约正式对外发布 FORTRAN I。

- 1957 年, 第一个 FORTRAN 编译器在 IBM704 计算机上实现, 并首次成功运行了 FORTRAN 程序。
- 1958 年, 在 FORTRAN I 基础上进行扩充和完善, 引进子函数等概念, 推出了商业化的 FORTRAN II。之后开始研发 FORTRAN III, 但由于种种问题, 最后没有发布。
- 1962 年, 推出 FORTRAN IV, 但没有充分考虑兼容性问题, 导致 FORTRAN II 程序不能在 FORTRAN IV 系统中运行, 使其应用受到了很大限制。
- 1962 年, 随着 FORTRAN 语言版本的不断更新和变化, 语言不兼容性问题日益突出, 语言标准化工作被提上了日程, ANSI 开始着手进行 FORTRAN 语言的标准化工作, 并将其命名为 “American Standard Fortran”。
- 1966 年, ANSI 正式公布了两个 FORTRAN 标准文本: FORTRAN (ANSI X3.9-1966) 和 Basic FORTRAN (ANSI X3.10-1966), 前者相当于 FORTRAN IV, 后者相当于 FORTRAN II。规定 Basic FORTRAN 是 FORTRAN 的一个子集, 从而实现了语言的向下兼容, 初步解决了语言的兼容性问题。这就是 FORTRAN 第一套标准, 后来被称为 FORTRAN 66。FORTRAN 66 有效的成功第一套工业标准版的 FORTRAN, 在国际上产生了广泛影响。
- 1972 年, ISO (International Organization for Standardization, 国际标准化组织) 在 FORTRAN 66 基础上制定了 FORTRAN 语言三级国际标准: 基本级, 中间级和完全级。
- 1978 年, ANSI 正式公布了新的 FORTRAN 标准 (ANSI X3.9-1978), 同时宣布撤消 ANSI FORTRAN 3.9-1966。这就是 FORTRAN 77。FORTRAN 77 兼容 FORTRAN 66。

- 1980 年, FORTRAN 77 被 ISO 正式确定为国际标准 ISO 1539-1980.
- 1991 年, Fortran 90 ISO 标准发布, ANSI 的 Fortran 90 标准于 1992 年发布。FORTRAN 90 添加了许多新特性, 如自由格式源代码输入, 运算符重载, 面向对象, 指针, 数组运算, 行内注释等。
- 1997 年, Fortran 95 发布。这是一个小改版, 主要修正了 Fortran 90 一些较为显著的问题, 并有适当扩充。<sup>1</sup>
- 2004 年, Fortran 2003 发布, 有着较大幅度的改版。
- 2010 年, Fortran 2008 发布, 是 Fortran 2003 的一个小改版。



关于 FORTRAN 的大小写问题:

习惯上, 旧版 FORTRAN (77 之前) 用大写, 如 FORTRAN 77, 而新版用 “Fortran”, 如 Fortran 90。

Fortran 90/95/03/08 的相继推出使的 Fortran 语言具备了现代高级编程语言的一些特性。Fortran 语言目前仍被广泛应用于科学与工程计算领域, 它具有以下特性:

- Fortran 语言的最大特性是接近数学公式的自然描述, 在计算机里具有很高的执行效率;
- 易学, 语法严谨;

---

<sup>1</sup>本讲义以 Fortran 95 为主

- 可以直接对矩阵和复数进行运算，这点 Matlab 有继承；
- 自诞生以来广泛地应用于数值计算领域，积累了大量高效而可靠的源程序；
- 很多专用的大型数值运算计算机针对 Fortran 做了优化；
- 广泛地应用于并行计算和高性能计算领域。

## 1.2 Fortran 77 的不足

- 不能动态分配内存
- 不能自定义新的数据类型
- 易出错，且编译器无法检测，特别是在调用子例程序或函数时
- 某些程序可能依赖于操作平台，影响可移植性
- 控制结构较弱，有时需要使用 GOTO 语句
- 其它陈旧特性：书写格式太严格，关键字大写，变量名太短（6 个字符），等等

## 1.3 Fortran 90 的新特性

- 自由格式源代码输入，关键字可小写，行内注释
- 最长 31 个字符的标识符
- 可以在表达式和赋值语句中对数组进行整体操作，极大地简化了数学和工程计算

- 可以动态分配内存（指针）
- 可以自定义新的数据类型
- 运算符重载
- 支持模块（MODULE），将有关联的过程和数据封装在一起，使它们可以被其它程序单元调用
- 更多控制语句：SELECT CASE, EXIT, CYCLE
- 递归（RECURSIVE）过程
- 用户可控制的数字精度
- 新的和增强的内部过程和输入输出特性
- 强化了语法检测功能

## 1.4 Fortran 95 的新特性

- 处理修正 Fortran 90 的一些错误外，Fortran 95 还有以下主要特性：
  - 提供 FORALL 结构，实行矢量化运算，提高程序在并行机上的执行效率
  - 用户可以自定义的 PURE 过程和 ELEMENTAL 过程
  - 指针的初始化和派生数据类型的缺省初始化
- 其它修改的地方有：
  - 提供了更多的内置过程

- 废弃了实型变量作为 DO 循环的循环控制变量
- 废弃了 PAUSE, ASSIGN, 带标签的 GOTO 语句
- 废弃了输入输出的 H 编辑符
- 下面是一些过时的特性:
  - 算术 IF 语句 (建议使用 IF 结构)
  - 多个 DO 循环共用一条终止语句 (建议使用 ENDDO)
  - 交替 RETURN 语句 (Alternate return)
  - 计算 GOTO 语句
  - 语句函数
  - DATA 语句可以出现在可执行语句之后 (建议放在可执行语句之前)
  - Assumed character length functions
  - 书写源程序的固定格式 (建议使用自由格式)
  - 使用 CHARACTER\* 声明字符变量的长度 (建议使用 CHARACTER(len=k))

## 1.5 Fortran 的优点

- 历史悠久, 已在各科学领域中积累了大量的优秀的 Fortran 源程序
- 语法严格 (如数组越界检测), 非常适合严谨的科学计算领域



- 可以直接复数运算和数组运算
- 在并行计算方面有独特优势，如 PURE 过程，ELEMENTAL 过程，FORALL 过程等
- Fortran 是编译型语言，效率高

## 1.6 Fortran 编译器

### 1.6.1 Windows 操作系统下较流行的 Fortran 编译器

- Microsoft Fortran PowerStation 4.0  
微软公司开发的 Fortran 编译器，1997 年转让给 DEC 公司。
- Digital Visual Fortran  
Fortran PowerStation 的 DEC 公司版本，版本号为 5.0.x ~ 6.0.x，1998 年 DEC 公司被康柏公司收购，Digital Visual Fortran 更名为 Compaq Visual Fortran。
- Compaq Visual Fortran (CVF)  
Digital Visual Fortran 的进一步发展，版本号 6.5.x ~ 6.6.B。2002 年康柏公司并入惠普公司，但仍然称为 Compaq Visual Fortran，版本号升级到 6.6.C。
- Intel Visual Fortran (IVF)  
英特尔公司开发的 Fortran 编译器。事实上，惠普购买了 Compaq 的 Fortran 编译器技术之后不久，便留下了用于 Linux/UNIX 系统的相关技术，而将 Windows 平台上的 Fortran 编译器相关

权利全部转售给 Intel。因此从 CVF 6.6.C 之后，Windows 台下的 Visual Fortran 编译器就改由 Intel 生产和销售。需要指出的是，IVF 需要微软 Visual Studio 外壳的支持才能实现 Visual IDE 功能，否则只提供命令行界面的编译方式。

- PGI Visual Fortran (PVF)

由意法半导体旗下子公司，世界领先的独立的高性能计算技术编译器及开发工具供应商 Portland Group 开发，也需要微软 Visual Studio 外壳的支持。

- Absoft Pro Fortran，由 Absoft 开发的 Fortran 编译器。
- Lahey Fortran，由 Lahey 开发的 Fortran 编译器。
- G95，免费，开放源代码，基于命令行方式。

### 1.6.2 Linux/Unix 操作系统下较流行的 Fortran 编译器

- GFortran，GNU Fortran，免费，开放源代码；GCC（GNU Compiler Collection, GNU 编译器套装）中的 Fortran 编译器；是 Linux 下的优秀编译器，用于替代 g77；目前支持 Fortran 95，以及部分 Fortran 2003 功能。
- G95，免费，开放源代码，Fortran 95 编译器。
- Intel Fortran，英特尔公司开发的 Fortran 编译器。
- PGI Fortran，由 Portland Group 开发的 Fortran 编译器。
- Absoft Pro Fortran，由 Absoft 开发的 Fortran 编译器。

- Sun Studio, 由 Oracle 公司开发和维护, 提供经过优化的 C、C++ 和 Fortran 编译器, 有 Solaris 和 Linux 版本。

### 1.6.3 Fortran 编译器使用推荐

- Windows: Code::Blocks (含 gcc 的跨平台集成开发环境), 使用时将 GNU Fortran 设为缺省编译器。也可以和 G95 配合使用 (先安装 G95, 然后安装 Code::Blocks)
- Linux: GFortran 或 Code::Blocks (需图形界面)。

### 1.6.4 高性能程序库

Fortran 编程的另一个显著特点就是可以利用一些现成的高性能程序库 (package), 如:

- BLAS — Basic Linear Algebra Subroutines
- LAPACK — Linear Algebra PACKage
- IMSL — 国际数学和统计链接库
- MKL — Intel 数学核心库
- ... ..

## 1.7 程序设计方法


目前常用的程序设计方法主要有: 模块化, 结构化和面向对象。

- 模块化程序设计方法：在设计和编写大程序时，按功能将程序划分成若干子程序，每个子程序实现一定的功能，这就是模块化。模块化可以降低程序的复杂性，提高程序的正确性，可靠性，可读性和可维护性。
- 结构化程序设计方法：由计算机科学家 E. W. Dijkstra 于 1965 年提出，是指采用三种基本控制结构（顺序结构，选择结构和循环结构），自顶向下，逐步求精的程序设计方法。
- 面向对象程序设计方法：模块化和结构化属于传统的程序设计方法，是基于面向过程的方法，把数据和处理数据的过程分离为相互独立的实体，当数据结构改变时，所有相关的处理过程都要进行修改，因此不利于当前大型复杂项目的开发和设计。而面向对象程序设计方法则将数据及对数据的操作方法封装在一起，成为一个整体（对象），并将其作为程序的基本单元，有效提高了软件的重用性、灵活性和扩展性。

## 1.8 推荐参考资料

- [1] S. J. Chapman, **Fortran 95/2003 for Scientists and Engineers**, 3rd edition, McGraw-Hill, 2007.  
中文翻译：Fortran 95/2003 程序设计 (第三版), 中国电力出版社, 2009  
很好的入门教材, 仔细周到, 规范先进, 内容丰富, 中文翻译质量一般, 可结合英文版一起看。
- [2] 彭国伦, **Fortran 95 程序设计**, 中国电力出版社, 2002.
- [3] 白云等, **Fortran 95 程序设计**, 清华大学出版社, 2011.

- [4] I. Chivers and J. Sleightholme, **Introduction to Programming with Fortran: With Coverage of Fortran 90, 95, 2003, 2008 and 77**, 2nd Edition, Springer, 2012.  
较全面, 涉及 F77/F90/F95/F2003, 是一本很好的案头参考, 适合有一定基础的读者.  
第一版有中文翻译, 书名“Fortran 权威指南”(人民邮电出版社, 2009).
- [5] M. Metcalf, J. Reid and M. Cohen, **Modern Fortran Explained**, Oxford University Press, 2011.  
Fortran explained 系列, 经典 Fortran 书籍, 介绍了 Fortran 的新特性.
- [6] **Fortran 95 Standard**, 1997. <http://www.nag.co.uk/sc22wg5/links.html>

 本讲义主要参考 [1, 2, 3, 6]

## 第二章 Fortran 编程基础

一个简单的 Fortran 程序，文件取名为 f01.f95。

Fortran 源代码 2.1 一个简单的 Fortran 程序

```
1 PROGRAM main ! 程序名
2
3 ! this is an example 注解
4
5 WRITE(*,*) 'Hello world!' ! 可执行语句
6
7 END PROGRAM main ! 程序结束
```

- 编译: `gfortran -O2 -o f01 f01.f95` 或 `ifort -O2 -o f01 f01.f95`
- 运行: `./f01`
- Fortran 程序基本结构:
  - 一个程序由一个或多个程序单元组成
  - Fortran 95 的四种基本程序单元: 主程序, 外部过程, 模块和数据块单元
  - 程序单元一般分以下四部分 (以主程序单元为例)

```
PROGRAM main    ! 程序单元声明部分  
REAL :: x      ! 数据类型声明部分, 不可执行语句  
x=2.0          ! 可执行语句部分  
END PROGRAM main ! 程序单元结束声明部分
```

## 2.1 Fortran 基础知识

每一个程序设计语言都有严格的词法、语法和语义规定, 且不允许出现二义性和不确定性。词法、语法和语义是程序设计语言的基本概念。

- 词法: 构成合法单词的规则称为词法, 编译器根据词法来判断一个单词是否合法。
- 语法: 具有特定含义的字符串或句子称为语句, 构成合法语句的规则称为语法。
- 语义: 合法语句的实际含义称为语义, 对语义的理解是否正确, 直接影响程序的可靠性和正确性。

### 2.1.1 字符集

FORTRAN 95 基本字符集

英文字母	A ~ Z, a ~ z
数字	0 1 2 3 4 5 6 7 8 9
下划线	_
特殊字符 (21 个)	␣(空格): = ! + - * / ( ) , . ' " % & ; < > ? \$

- 英文字母不区分大小写, 但作为字符串时除外
- “?” 和 “\$” 没有特殊含义
- Fortran 中的空格: 通常被忽略
- 基本字符集以外的可打印字符 (如汉字, 希腊字母, Tab 符等), 只允许出现在注释、字符串和输入输出的记录中

### 2.1.2 标识符

用于标识变量、函数、数组、过程等。

- 只能包含字母、数字、下划线
- 第一个字符必须是字母
- 长度不超过 31 (有的编译器可能支持更长的标识符)



- 不区分大小写

### 2.1.3 关键字

具有特殊功能的标识符。Fortran 95 不保留关键字，即用户可以用关键字来命名变量等，但不建议这样做！

### 2.1.4 语句

- 分可执行语句和非执行语句
- 语句有一定的排列顺序：从上往下，同一水平顺序任意，注释可以出现在任意位置

PROGRAM, FUNCTION, SUBROUTINE, MODULE, BLOCK DATA 语句		
USE 语句		
IMPLICIT NONE		
FORMAT 语句 ENTRY 语句	PARAMETER 语句	IMPLICIT 语句
	PARAMETER 语句 DATA 语句	派生数据类型定义，接口块， 类型声明语句，说明语句，语句函数语句
	DATA 语句	可执行结构
CONTAINS 语句		
内部子过程，模块（module）		
END 语句		

- 语句出现规则

Statements allowed in scoping units							
	主程序	模块	数据块	外部子程序	模块子程序	内部子程序	接口体
USE 语句	Yes	Yes	Yes	Yes	Yes	Yes	Yes
ENTRY 语句	No	No	No	Yes	Yes	No	No
FORMAT 语句	Yes	No	No	Yes	Yes	Yes	No
DATA 语句	Yes	Yes	Yes	Yes	Yes	Yes	No
派生类型定义	Yes	Yes	Yes	Yes	Yes	Yes	Yes
接口块	Yes	Yes	No	Yes	Yes	Yes	Yes
可执行语句	Yes	No	No	Yes	Yes	Yes	No
CONTAINS 语句	Yes	Yes	No	Yes	Yes	No	No
语句函数	Yes	No	No	Yes	Yes	Yes	No
其它语句 *	Yes	Yes	Yes	Yes	Yes	Yes	Yes

\* 其它语句包括：PARAMETER 语句，IMPLICIT 语句，类型声明，属性声明

### 2.1.5 书写格式

- 固定格式（FIXED FORMAT）和自由格式（FREE FORMAT）

- FORTRAN 77 源代码需按固定格式书写，扩展名为 .f 或 .for

**固定格式：**程序代码每一行中每个字符字段的意义

第 1 个字符	如果是字母 c, C 或星号 *, 表示此行是注释
第 1-5 个字符	如果是数字, 则表示这一行的语句号, 否则只能是空白
第 6 个字符	如果是 0 以外的字符, 表示这一行程序会接续上一行
第 7-72 个字符	FORTRAN 程序代码的编写区域
第 73 个字符之后	不使用, 超过部份会被忽略, 有的编译器会有错误讯息

- Fortran 90 之后的源代码按自由格式书写，扩展名为 .f90, .f95, 等等
- 自由格式不再规定每行的第几个字符有什么作用，但需要注意以下几点：
  - 惊叹号 “!” 可以出现在任意位置，其后面的文字都是注释
  - 每行可以写 132 个字符（注意! 并不是无限长）
  - 行号（如果需要的话）放在每行程序的最前面
  - 如果一行代码的最后是符号 &, 则表示下一行代码会和这一行连接
  - 如果一行代码的最前面是 &, 则表示与上一行代码连接
  - 如果一行中写多条语句，则语句之间用分号 “;” 隔开，最后一条语句后面不能有分号

## 2.2 基本数据类型

Fortran 中的基本数据类型主要有：整型（INTEGER），实型（REAL），复型（COMPLEX），字符型（CHARACTER）和逻辑型（LOGICAL）。每种数据类型都可以通过 kind 进行细化。

内部数据类型

类型	关键字	示例与说明
整型	INTEGER	INTEGER :: a ! 缺省占 4 个字节
	INTEGER(k)	INTEGER(2) :: b, c ! 短整型, k=1, 2, 4, 8
实型	REAL	REAL :: x, y ! 缺省占 4 个字节
	REAL(k)	REAL(8) :: x, y ! k=4, 8, 16
	DOUBLE PRECISION	DOUBLE PRECISION :: x, y ! 同 REAL(8)
复型	COMPLEX	COMPLEX :: z ! 缺省占 8 个字节
	COMPLEX(k)	COMPLEX(8) :: z ! k=4, 8, 16
	DOUBLE COMPLEX	
字符型	CHARACTER	CHARACTER :: str ! 缺省占 1 个字节
逻辑型	LOGICAL	LOGICAL :: flag ! 缺省占 4 个字节
	LOGICAL(k)	LOGICAL(1) :: flag ! k=1, 2, 4, 8

## 2.3 常量与变量

### 2.3.1 常量

常量是指在整个程序运行过程中不变的量，如常数，字符串，符号常量等。

- 整数：不带小数点，如 3, 412。(注意：整数与整数的运算结果仍为整数)
- 实数（浮点数）分单精度和双精度，可以用小数形式或指数形式表示，如：

`x=3.14, y=3.14E2, z=3.14D-3` ! 实数缺省为单精度，E 表示指数，D 表示双精度  
`x=3.14, y=3.14_8` ! 可以用下划线来指定实数的 kind 值

- 复数用两个实数来表示实部和虚部，如 (3.0, 4.0)；复数可以直接参加算术运算
- 字符串可以使用单引号或双引号表示（FORTRAN 77 只能使用单引号）
- 符号常量：通过关键字 `PARAMETER` 将变量声明成常量

`REAL(8),PARAMETER :: pi=3.141593` ! 声明常量时必须初始化  
`REAL(8) :: e`  
`PARAMETER(e=2.718282)` ! `PARAMETER` 语句必须出现在所有可执行语句之前

### 2.3.2 变量

变量是程序设计语言的基本元素，其值可以根据需要随时改变。


- 变量名必须以字母开头，可以含字母、数字和下划线
- 变量名长度不要超过 31 个字符
- 变量名不要与其它标识符重复，如关键词，主程序名等
- 变量名不区分大小写，最好能见名识意

### 2.3.3 变量的声明

变量使用前必须先声明，声明方式有隐式声明和显式声明。

#### 隐式声明

- **I-N 规则**：Fortran 默认以 I ~ N (或 i ~ n) 开头的变量为整型，其它为实型
- Fortran 允许通过 **IMPLICIT** 语句取消 I-N 规则，或重新定义 I-N 规则
  - **IMPLICIT NONE**：取消 I-N 规则，此时所有变量必须显式声明
  - **IMPLICIT 数据类型 (字母范围)**：指定以“字母范围”开头的变量的缺省类型，如：  
**IMPLICIT REAL(a-c), INTEGER(d-g)** ! 以 a,b,c 开头的变量为实型，以 d,e,f,g 开头的为整型
  - **IMPLICIT** 必须出现在所有显式声明和可执行语句之前

 为了减少错误，建议尽量使用 **IMPLICIT NONE**

## 显式声明


类型关键字 [(kind 值)][, 属性说明] :: 变量名表

- 类型声明语句必须出现在可执行语句之前
- **kind** 和“属性说明”是可选的
- 属性说明有（可同时使用多个）：

<b>ALLOCATABLE</b>	可以动态分配空间
<b>DIMENSION</b>	数组形状大小
<b>EXTERNAL</b>	外部函数
<b>INTENT</b>	形参属性
<b>INTRINSIC</b>	允许内部函数作实参
<b>OPTIONAL</b>	可选形参
<b>PARAMETER</b>	声明常量
<b>POINTER</b>	指针
<b>PRIVATE</b>	模块私有对象
<b>PUBLIC</b>	模块中的对象在模块外也可见
<b>SAVE</b>	保留局部变量的值（静态变量）
<b>TARGET</b>	目标对象，指针只能指向具有 TARGET 属性的变量

- 整型变量的声明


INTEGER :: a	! 缺省 KIND=4
INTEGER(2) :: a, b	! 可同时声明多个变量, 用分号隔开
INTEGER*2 :: a, b	! 同上
INTEGER(KIND=2) :: a, b	! 同上, Fortran 90 用法
INTEGER*2 a, b	! 同上, FORTRAN 77 用法
INTEGER :: a=2, b=3	! 变量声明时可以初始化, F77 需使用 DATA 初始化
INTEGER, PARAMETER :: c=8	! 声明变量时, 可以指定一些属性

 从 Fortran 90 开始才使用双冒号“::”, 可以省略。但如果省略双冒号的话, 则声明变量时不能初始化。

- 实型变量的声明

REAL :: x, y	! 缺省 KIND=4
REAL(8) :: x=3.14, y=3.14D-3	! 可同时声明多个变量, 用分号隔开



 给双精度实型变量赋值时，尽量用双精度常数。

```
REAL(8) :: x=6.6666666666666666 ! 实型常数缺省是单精度，只保留 7 位有效数字
REAL(8) :: y=6.6666666666666666_8 ! 最后面的_8 代表该常数的 kind 值
```

- 复型变量的声明

```
COMPLEX :: z ! 缺省 KIND=4，即实部和虚部均占 4 个字节
COMPLEX(8) :: z=(3.0, 4.0) ! 复型变量初始化
```
- 字符型变量的声明

```
CHARACTER :: str ! 缺省 KIND=1，单个字符
CHARACTER(LEN=8) :: str ! 可以存放多个字符，即字符串
CHARACTER(8) :: str="hello" ! 初始化，可以用单引号或双引号
```
- 逻辑型变量的声明

```
LOGICAL :: flag ! 缺省 KIND=4，占 4 个字节
LOGICAL(1) :: flag=.TRUE. ! 初始化，逻辑常量：.TRUE. 和 .FALSE.
```

### 2.3.4 DATA 初始化

- 变量在声明时可以直接初始化

- 也可以使用 DATA 语句进行显式初始化

DATA 变量列表/初值列表/, 变量列表/初值列表/, ...

### 2.3.5 NAMELIST

## 2.4 表达式与赋值语句

### 2.4.1 表达式

Fortran 95 表达式有：算术表达式，关系表达式，逻辑表达式，字符串表达式。

- 算术运算符：+，-，\*，/，\*\*（指数运算）
- 关系运算符：<，<=，>，>=，==，/=（.LT.，.LE.，.GT.，.GE.，.EQ.，.NE.）
- 逻辑运算符：.NOT.，.AND.，.OR.，.XOR.，.EQV.，.NEQV.（优先级依次降低，后三者相同）
- 运算优先级：圆括号 → 指数运算 → 乘除 → 加减 → 关系运算 → 逻辑运算
- 字符串运算：
  - 取字符串的片段（字串）：str(i:j), str(:j), str(i:)
  - 字符串连接：str1//str2
  - 字符串可以进行比较运算



整型数据做运算时要特别注意，特别是除法

### 2.4.2 赋值语句

变量名=表达式

## 2.5 类型转换

### 2.5.1 自动转换

- 同类型数据做运算时，结果的类型不变
- 不同类型数据做运算时，低级别类型向高级别类型转换

(低) INTEGER(1) → INTEGER(2) → INTEGER(4) → INTEGER(8)  
→ REAL(4) → REAL(8) → COMPLEX(4) → COMPLEX(8) (高)

### 2.5.2 显式转换

- 通过系统提供的函数进行类型转换

```
INT(x[,kind]), REAL(x[,kind]), DBLE(x), CMPLX(x,y[,kind])
```

## 2.6 END 语句和 STOP 语句

- END 语句只能在主程序中使用，标志程序结束
- STOP 语句可在主程序，模块和外部子例程序中使用，表示终止程序运行，返回操作系统

```
STOP [提示信息] !提示信息可以是数字或字符串，供调试使用
```

## 2.7 内置过程 INTRINSIC PROCEDURES

- Fortran 95 提供了许多内置过程（函数和子例程序）供调用，除了上面的类型转换函数外，常见的有

```
sin(x), cos(x), tan(x), abs(x), sqrt(x), exp(x), log(x), mod(m,n), ...
```

- 更多内置过程参见 6.5 节。

## 2.8 输入输出

### 2.8.1 表控输出

- 按照缺省的格式，向缺省的输出设备输出数据。

**PRINT** \*, 输出列表 ! 输出列表中可以含变量或表达式  
**WRITE**(\*,\*) 输出列表

- WRITE** 中第一个星号表示向缺省输出设备（屏幕）输出数据，第二个星号代表缺省输出格式
- 输出列表中可以含变量或表达式，用逗号分隔
- 缺省输出格式

整型	kind=1 时域宽为 4 个字符
	kind=2 时域宽为 6 个字符
	kind=4 时域宽为 11 个字符
	kind=8 时域宽为 21 个字符
实型	kind=4 时总域宽为 14 个字符，数值部分为 10，指数部分为 4
	kind=8 时总域宽为 23 个字符，数值部分为 18，指数部分为 5
复数	用圆括号显示
字符型	字符（串）按实际长度输出
逻辑型	占一个字符：T 或 F

### 2.8.2 表控输入

- 按照缺省的格式，从缺省的输入设备输入数据。

#### READ(\*,\*) 变量列表

- 第一个星号表示从缺省输入设备（键盘）接受数据，第二个星号代表使用缺省格式
- 变量列表中各个变量用逗号隔开
- 输入数据个数不能少于变量个数，可以分多行输入
- 输入数据个数多于变量个数时，多余部分被忽略
- 输入数据时可以用“/”表示输入结束，未得到输入数据的变量保持原值
- 不同数据用逗号或空格隔开
- 连续两个逗号表示输入一个空数据
- 输入数据时可以使用“\*”表示重复输入某个数据，如： $3*5$  表示 3 个 5



输入时要有提示信息，并尽量回显输入的数据，以便确保输入正确。

### 2.8.3 格式化输出

- 按用户指定的格式输出。

**PRINT** 格式说明语句号, 输出列表

**WRITE**(\*, 格式说明语句号) 输出列表

- 这里的星号代表缺省输出（屏幕）
- 这里“格式说明语句号”代表描述格式的 **FORMAT** 语句

语句号 **FORMAT**(格式说明)

- 举例


```
WRITE(*,100) x, y
```


```
100 FORMAT(1X, 'x=', F8.3, ', y=', D20.6)
```

- “格式说明”通过“格式描述符”描述
- Fortran 将输出缓冲区中的第一个字符作为纵向控制符，即输出的第一个字符是纵向控制符：

## 纵向控制符

字符	含义
空格	换行，即新起一行
0	前进两行，即在当前行下面空一行
1	换页
+	将输出位置 (光标) 移到当前行开头，即覆盖当前行
其它字符	功能同空格

 不要将你想要输出的数据作为第一个输出字符。每个格式描述符最好单独写。

 使用表控输出时，系统会自动在每个输出缓冲区的最前面插入一个空格。

- 格式描述符分两类：可重复（即可指定重复系数，也称数据描述符）和不可重复（也称控制描述符）

## 可重复格式描述符

格式	含义	输入	输出
[r]Iw[m]	整型数据, r 为重复系数, w 为域宽, 至少输出 m 个数字, 不够时用 0 填充	Y	Y




[r]Bw[.m]	按二进制输出	Y	Y
[r]Ow[.m]	按八进制输出	Y	Y
[r]Zw[.m]	按十六进制输出	Y	Y
[r]Fw.d	按小数形式输出实型数据, 小数部分占 d 位	Y	Y
[r]Ew.d[Ee]	按指数形式输出实型数据, 尾数在 0.1 至 1.0 之间, 小数部分占 d 位, 指数部分占 e 位, 一般要求 $w \geq d + 7$	Y	Y
[r]ENw.d[Ee]	同上, 工程计数法, 尾数在 1.0 至 1000 之间	Y	Y
[r]ESw.d[Ee]	同上, 科学计数法, 尾数在 1.0 到 10.0 之间	Y	Y
[r]Dw.d	按指数形式输出双精度实型数据 (功能同 E, 已过时)	Y	Y
[r]A[w]	输出字符型数据 (字符串), w 为域宽 (缺省为实际长度)	Y	Y
[r]Lw	输出逻辑型数据, w 为域宽, 缺省为实际长度	Y	Y
[r]Gw.d[Ee]	输出任意类型数据的通用描述符	Y	Y

## 不可重复格式描述符


单引号	输出字符串		Y
双引号	输出字符串		Y

/	换行	Y	Y
nX	将输出位置向右跳过 n 个位置	Y	Y
Tn	将输出位置移到本行第 n 个位置	Y	Y
TLn	将输出位置向左移 n 个位置	Y	Y
TRn	将输出位置向右移 n 个位置	Y	Y
kP	设置 F 和 E 编辑符指数比例因子 (obsolescent)	Y	Y
SP	输出正数时加正号 “+”		Y
SS	取消 SP 功能		Y
:	在没有更多数据时结束输出		Y

 通常将 1X 作为 FORMAT 格式说明中的第一个格式编辑符，代表换行。

- 可以将多个格式编辑符组合成一组，重复使用，也可以嵌套，如：

```
120 format(1X, 2I3, 3(I4,F10.5), 4(I3,2(I4,F10.5)))
```

 复杂的 FORMAT 语句会给理解和调试带来一定的困难，因此建议不要使用太复杂的 FORMAT 语句。

- 格式说明也可以使用字符常量或变量，下面三种写法等价：

```
WRITE(*,100) x, y
100 FORMAT(1X, F8.3, D20.6)    ! FORMAT 语句可以出现在 WRITE 之前或之后

str='(1X, F8.3, D20.6)'
WRITE(*,str) x, y

WRITE(*,'(1X, F8.3, D20.6)') x, y
```

- 输出时，数据和格式说明是同时被处理的，程序从最左端的变量和最左端的格式说明开始，从左至右依次扫描，一个数据对应一个格式描述符，输出列表中的变量与格式说明中的格式描述符必须是一一对应，特别是数据类型。
- 将记录发送到输出设备之前，计算机在内存中为其建立完整的映射，即输出缓冲区（output buffer）。
- 格式说明的重复使用：若格式描述符用完后，仍有变量没有输出，则自动重复使用格式说明：如果格式说明中没有圆括号，则从头开始；如果格式说明中有圆括号，则从最后一个不带重复次数的圆括号开始。

```
200 FORMAT(1X, 'x=', F10.6) ! 重复使用格式说明时从头开始
```

```
210 FORMAT(1X, 'x=', (1X, I3, 2(3X, I5)))
```

! 格式说明中的有两个圆括号, 重复使用的格式是 (1X, I3, 2(3X, I5))

```
220 FORMAT(1X, 'x=', 1X, I3, 2(3X, I5)) ! 重复使用时从头开始, 注意与上面  
语句的区别
```

## 2.8.4 格式化输入

**READ(\*, 格式说明语句号) 输入变量列表**

- 这里的星号代表缺省输入设备 (键盘)

 使用格式化输入时容易引起混乱, 要谨慎使用。

## 2.9 选择/分支结构

### 2.9.1 IF 结构

IF (条件) THEN

语句块

END IF !END IF 也可以写出 ENDIF

IF (条件) THEN

语句块

ELSE

语句块

END IF

IF (条件) THEN

语句块

ELSE IF (条件) THEN

语句块

... !可以有多个 ELSE IF

ELSE !可选

语句块

END IF

- 逻辑 if 语句

**IF (条件) 语句**    ! 只能是单条语句

- IF 语句可以嵌套
- 可以给 IF 语句命名 (标记), 以增加程序的可读性, 特别是嵌套时

**[name:] IF (条件) THEN**

... ..

**END IF [name]**    ! 这里 name 可省, 若给出, 则必须保持一致性

- 举例

#### Fortran 源代码 2.2 带标记的 IF 语句

```
1      ...  
2      outer: IF (...) THEN ! 这里的 outer 是 IF 语句的标记  
3          inner: IF (...) THEN ! 这里的 inner 是另一个 IF 语句的标记  
4              ...  
5              ELSE inner ! 表示 ELSE 是 inner 代表的 IF 语句  
6              ...  
7          END IF inner ! 表示解释 inner 代表的 IF 语句
```

```
8      ELSE outer
9          ...
10     END IF outer
11     ...
```


### 2.9.2 SELECT CASE 结构

```
[name:] SELECT CASE (case_exp)  ! 也可以给该结构做标记
CASE (值列表)
    语句块
CASE (值列表)
    语句块
... ...    ! 可以有多个 CASE
CASE DEFAULT  ! 可选
    语句块
END SELECT [name]
```

- 将表达式的值依次与 CASE 后面的“值列表”进行匹配，若匹配成功，则执行相应的语句
- `case_exp` 可以是整数、字符(串)或逻辑表达式

- 值列表只能是整数, 字符 (串), 逻辑值, 或数值范围

值列表	含义
<code>a</code>	<code>case_exp==a</code> 时, 执行相应的语句块
<code>a:</code>	<code>case_exp&gt;=a</code> 时, 执行相应的语句块
<code>:b</code>	<code>case_exp&lt;=b</code> 时, 执行相应的语句块
<code>a:b</code>	<code>a&lt;=case_exp&lt;=b</code> 时, 执行相应的语句块
<code>a1,a2,...,an</code>	与其中任何一个相等时, 执行相应的语句块
<code>str1,str2,...,strn</code>	与其中任何一个字符串匹配时, 执行相应的语句块

 在 CASE 结构中尽量使用 `CASE DEFAULT`, 以便捕捉可能存在的逻辑错误或非法输入

## 2.10 循环语句

### 2.10.1 DO 结构

`[name:] DO 循环变量=初始值, 终止值, 步长` ! 步长可省略, 缺省为 1  
语句块 (循环体)





```
END DO [name]
```

- 非正常退出: CYCLE, EXIT

```
CYCLE [name] ! name 为循环标记  
EXIT [name]
```

- DO 循环可以嵌套, 非正常退出通常与选择结构配合使用

 循环嵌套套时, 给循环结构做标记, 可以方便退出多重循环

 不要在循环体内修改循环变量的值; 也不要再在循环后面使用循环变量的值 (有可能非正常退出)

### 2.10.2 DO WHILE 结构

```
[name:] DO WHILE (条件)  
    语句块 (循环体)  
END DO [name]
```

```
[name:] DO      !无限循环，循环体中必须有退出机制  
    语句块（循环体）  
END DO [name]
```

### 2.10.3 隐式 DO 循环

通常用在输入输出语句中

## 2.11 数组

### 2.11.1 一维数组

- 一维数组的声明


```
类型关键字 [kind], DIMENSION(k)[, 其他属性说明] :: 数组名列表  
类型关键字 [kind], DIMENSION(k1:k2)[, 其他属性说明] :: 数组名列表  
类型关键字 [kind][, 其他属性说明] :: 数组名 (大小)
```

- 举例:

```
REAL(8), DIMENSION(5) :: x, y    ! 下标缺省从 1 开始  
REAL(8), DIMENSION(0:9) :: z  
REAL(8) :: a(5), b(-2:2)    ! 老用法, 仍然支持  
REAL(8), DIMENSION(5) :: u=(/1.0,2.0,5.0,4.0,6.0/)    ! 声明时可以初始化
```

- 声明数组大小时可以使用符号常量:

```
INTEGER, PARAMETER :: N=100  
REAL(8), DIMENSION(N) :: x    ! N 必须是常量  
REAL(8) :: a(2*N)
```

 声明数组大小时使用符号常量是一个很好的编程习惯。

- 数组元素的引用:

```
x(k)    ! 引用单个元素  
x(i:j)  ! 引用多个元素, 从第 i 个到第 j 个  
x(i:j:step) ! step 是步长  
x(:j); x(:j:step) ! i 省略时, 默认从第一个元素开始
```

```
x(i:); x(i::step) !j 省略时, 默认到最后一个元素  
x(::step) !i 和 j 都可以省略  
x(:) !所有元素  
idx=(/1,3,2,1/)  
x(idx) !下标也可以是整型数组, 可以有重复, 但赋值是不能有重复下标
```

- 数组赋值:

```
x=(/1.0, 2.0, 3.0, 4.0, 5.0/)  
x(1:5)=(/sqrt(i),i=1,5/) !隐式 DO 循环  
x(0:8:2)=(/y(1),y(5),y(3),y(2),y(4)/)  
x=(/z(1:9:2)/)
```

- 数组的输出: 可以使用隐式 DO 循环

隐式 DO 循环

```
1 WRITE(*,*) (x(i),i=1,6) ! 隐式 DO 循环  
2  
3 100 FORMAT(1X, 'x=', 6F10.6)  
4 WRITE(*,100) (x(i),i=1,6)  
5
```


```
6  ! 可同时对多个输出量使用隐式 DO 循环
7  WRITE(*,120) (x(i), sin(x(i)), i=1,6,2)
8  120 FORMAT(1X, 2F10.6)
```

- 数组的输出：数组名

WRITE(\*,\*) x ! 表示输出所有元素

100 FORMAT(1X, 'x=', 3F10.6)

WRITE(\*,100) x(1:5:2) ! 也可以只输出部分元素

 为了防止数组越界，在调试程序时应打开编译器的越界检测选项；由于越界测试会降低程序运行速度，因此完成调试后可以关闭越界测试选项。

### 2.11.2 二维数组

类型关键字 [kind], DIMENSION(d1,d2)[, 其他属性说明] :: 数组名列表  
类型关键字 [kind] [, 其他属性说明] :: 数组名 (d1,d2)

- 举例：

```
REAL(8), DIMENSION(5,5) :: A    !下标缺省从1开始
REAL(8), DIMENSION(0:5,0:5) :: B  !指定下标范围
REAL(8) :: C(-2:2,-2:2)    !老用法, 仍然支持
```

- 存储方式: 按列存储, 如  $2 \times 2$  矩阵 **A** 的存储顺序是 **A(1,1)**, **A(2,1)**, **A(1,2)**, **A(2,2)**
- 初始化, 输入输出: 隐式 DO 循环, 可嵌套

```
! INTEGER, DIMENSION(3,3) :: A=(/1,1,1,2,2,2,3,3,3/)
INTEGER, DIMENSION(3,3) :: A=RESHAPE((/1,1,1,2,2,2,3,3,3/), (/3,3/))
100 FORMAT(1X,3I2)
WRITE(*,100) ((A(i,j), j=1,3), i=1,3)    !嵌套的隐式 DO 循环
WRITE(*,100) A    !也可以直接用数组名
```

### 2.11.3 多维数组

```
类型关键字 [kind], DIMENSION(d1,...,dn)[, 其他属性说明] :: 数组名列表
类型关键字 [kind] [, 其他属性说明] :: 数组名 (d1,...,dn)
```

- Fortran 最多可声明七维数组
- 多维数组的存储:

`DIMENSION(i1,i2,...,in) :: A`

! 按维数从低到高 ( $i1 \rightarrow i2 \rightarrow \dots \rightarrow in$ ) 依次存放

- 多维数组较复杂，慎用。

#### 2.11.4 数组运算

- 数组的加减

`A=B-C` ! A, B, C 为同维数的数组

`A=B+3`

`B=sin(A)` ! 函数作用在数组上时，等价于作用在每个元素上

- 函数作用在数组上：等价于作用在每个元素上，如：`sin(A)`
- 数组的乘法：矩阵乘法
- 内部数组函数

`allocated, lbound, ubound, shape, size` ! 查询函数

`all, any, count, matmul, reshape, product, dot_product`

`maxval, minval, maxloc, minloc, sum, transpose`

- 数组部分元素操作：冒号

```
x(3:5) = 8  
x(3:) = 8  
x(3:5) = b(4:2:-1)  
x(:) = A(:,3) B=A(1:3,5:2:-1)
```

- 加掩码的数组赋值：WHERE 结构

```
WHERE (数组关系表达式) 一条语句    ! WHERE 语句
```

```
WHERE (数组关系表达式)  
  语句块  
END WHERE
```

```
WHERE (数组关系表达式)  
  语句块  
ELSE WHERE  
  语句块  
END WHERE
```



[name:] WHERE (数组关系表达式) ! 也可以给 WHERE 结构做标记

语句块

ELSEWHERE (数组关系表达式) [name]

! ELSE WHERE 可以写在一起; 也可以有多个 ELSE WHERE

语句块

ELSE WHERE [name]

语句块

END WHERE [name]

- WHERE 只能用于数组
- WHERE 不能嵌套
- 举例

WHERE (A/=0) B=0 ! 等价于 IF (A(i)>0) B(i)=0

WHERE (A>0)

B=1.0/A

ELSE WHERE (A<0)

B=-1.0/A

```
ELSE WHERE  
  B=0  
END WHERE
```

- 隐含式循环：FORALL 结构

FORALL (下标范围 [, 下标范围, ...] [, mask]) 一条语句


```
[name:] FORALL (下标范围 [, 下标范围, ...] [, mask])  
  语句块  
END FORALL [name]
```

- 满足条件（下标范围，**mask**）的数组元素才会被操作
- FORALL 可以嵌套，里面只能出现跟设置数组数值相关的程序命令
- FORALL 中可以使用 WHERE，但 WHERE 中不能使用 FORALL
- 举例

```
FORALL (i=1:5:2) x(i)=0
```

```
FORALL (i=1:5,j=1:5,i==j)  ! 只给对角线元素赋值
```

```
A(i,j)=3  
B(i,j)=A(i,j)+5  
END FORALL
```

 FORALL 结构可以用 DO 循环实现，但 DO 循环必须按一定的顺序处理数组元素，因此只能串行运算，而 FORALL 可以按任意顺序处理数组元素，因此能并行运算。

### 2.11.5 可调数组

**类型关键字, ALLOCATABLE :: x(:), y(:), A(:, :)**

- 使用占位符声明数组的维数，而不是大小，使用时需要用 **ALLOCATE** 语句分配内存空间。


**ALLOCATE(x(100))** ! 动态分配空间


**ALLOCATE(y(-3:3), stat=flag)** ! 关键词 **STAT** 用于返回是否分配成功

**DEALLOCATE(x)** ! 释放动态分配的数组

**ALLOCATED(A)** ! 检查一个可调数组是否已分配空间

- flag** 是整型变量，返回值等于 0 时表示分配成功，否则表示分配空间失败。

 在 `ALLOCATE` 语句中, 应始终使用 `stat=flag`, 以便检测内存空间是否分配成功。

 可调数组用完后使用 `DEALLOCATE` 释放其所占有的内存空间是一个很好的编程习惯。

## 2.12 字符数据

### 2.12.1 字符变量声明

`CHARACTER(LEN=k[,kind]) [, 属性说明列表] :: 变量名列表`

```
CHARACTER(LEN=15,KIND=2) :: str1, str2
```

```
CHARACTER(15,2) :: str1, str2
```

```
CHARACTER(len=10) :: str1, str2*5, str3*20
```

```
CHARACTER*10 :: str1
```

```
CHARACTER(LEN=10), DIMENSION(10) :: str1, str2
```

### 2.12.2 字符串操作

- 字符串子串: `str(i:j)`

- 字符串连接: `str1//str2`
- 字符串可以直接进行比较运算, 如: `flag='ABC'>'AA'`, 比较结果与系统所使用的字符集有关, 一般采用 ASCII 码字符集。


```
flag='ABC'>'AA'    !flag 的值为真
flag='AAA'>'AA'    !flag 的值为真
flag='A'=='a'      !flag 的值为假
flag='AA'=='AA'    !flag 的值为真
```

- 内部字符串函数

Fortran 95 字符串函数

函数名	参数类型	返回值类型	说明
ACHAR(k)	int	char	返回 ASCII 码为 <code>k</code> 的字符
CHAR(k)	int	char	返回系统所用字符集中对应于 <code>k</code> 的字符
IACHAR(c)	char	int	返回字符 <code>c</code> 的 ASCII 码
ICHAR(c)	char	int	返回系统中对应于字符 <code>c</code> 的整数
LEN(str)	char	int	返回字符串 <code>str</code> 的长度
LEN_TRIM(str)	char	int	返回字符串 <code>str</code> 的长度, 不含尾部的空格

TRIM(str)	char	char	去掉 str 后面的空格
INDEX(str1, str2 [,back])	char[,logical]	int	返回 str1 中包含 str2 的第一个字符的序号, back 是可选参数, 如果提供且值为正, 则表示从 str1 末尾开始搜索
LLT(str1,str2)	char	logical	按 ASCII 码大小, 若 $\text{str1} < \text{str2}$ , 返回真
LLE(str1,str2)	char	logical	按 ASCII 码大小, 若 $\text{str1} \leq \text{str2}$ , 返回真
LGT(str1,str2)	char	logical	按 ASCII 码大小, 若 $\text{str1} > \text{str2}$ , 返回真
LGE(str1,str2)	char	logical	按 ASCII 码大小, 若 $\text{str1} \geq \text{str2}$ , 返回真

 CHAR 和 ICHAR 的返回值与系统采用的字符集有关, 因此尽量使用 ACHAR 和 IACHAR。

## 2.13 内部文件 — 数值与字符转换

- Fortran 提供了一种特殊机制, 即内部文件 (internal files), 来实现数值数据与字符数据之间的转换。例:

```
CHARACTER(LEN=5) :: str='123.4'
```

```
REAL :: x
```

```
READ(str,*) x    ! 将字符串 str 包含的数据转化为数值数据
```

```
CHARACTER(LEN=12) :: str
```

```
REAL :: x=12.3456
```

```
WRITE(str,100) x    ! 将数值数据 x 转换为字符数据
```

```
100 FORMAT(F12.6)
```

- 字符型变量和字符型数组都可以视作内部文件。

## 第三章 过程 PROCEDURE

### 3.1 主程序一般结构

- 一些基本概念：程序，文件，程序单元，过程，外部过程，内部过程，内置过程
- 主程序一般结构

**PROGRAM MAIN**

主程序的声明部分

主程序的执行部分

**CONTAINS**    !内部过程

**SUBROUTINE myroutine(...)**

子例程序的声明部分

子例程序的执行部分

**END SUBROUTINE myroutine**

**FUNCTION myfun(...)**

函数的声明部分

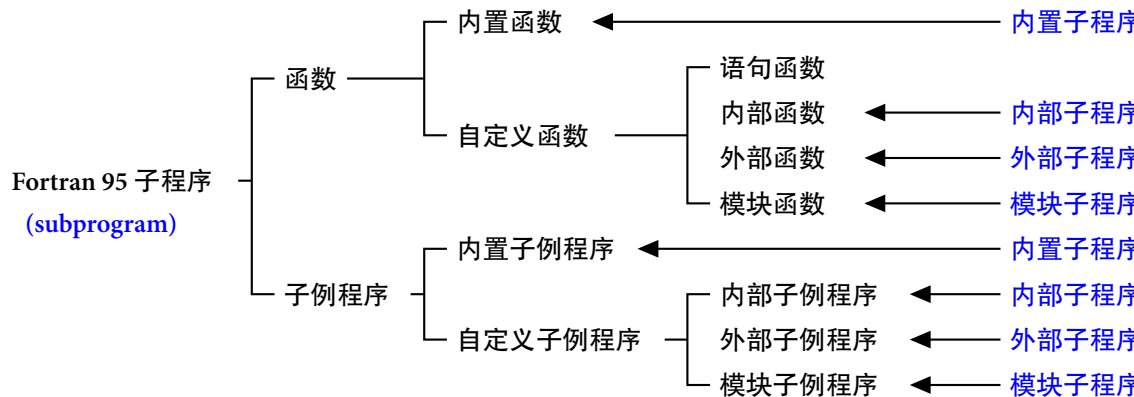
函数的执行部分

**END FUNCTION myfun**

**END PROGRAM main**



- Fortran 中的过程分函数子程序 (function, 简称函数) 和子例程序 (subroutine, 简称子例程序)
  - 函数通过函数名调用, 出现在表达式中
  - 子例程序通过 CALL 调用
- 内部过程: 定义在某个程序单元之中, 只有其所在的程序单元及所在程序单元的其他内部过程可以调用
- 外部过程: 一个独立的程序单元, 或其它语言编写的过程, 主程序和其它过程均可调用
- 模块过程: 包含在模块中的过程
- 过程接口块: 提供外部过程的接口信息
- 内置函数和内置子例程序: Fortran 自带的函数和子例程序



### 3.2 自定义函数

**FUNCTION** 函数名 (形参列表)

函数体

**END [FUNCTION [函数名] ]**

- 函数调用

变量 = 函数名 (实参列表)

- 函数名即代表函数的返回值，在函数体内必须声明并赋值
- 也可以在定义函数时直接声明其数据类型，即

```
FUNCTION myfun(...)
```

```
REAL :: myfun
```

```
REAL FUNCTION myfun(...)    ! 效果与上面等价
```

- 在主调程序单元中也必须声明其数据类型
- 函数只能有一个输出值
- 形参可以是变量、数组、过程、指针等
- 可以没有形参，但圆括号不能省
- 实参与形参结合时，传递的是地址，因此它们的类型必须一致，否则会出现不可预料的错误
- 数组名作为实参时，传递的是首地址
- 程序执行时，遇到 **RETURN** 或 **END**，则返回主调程序



一般来说，函数不会修改形参的值，因此将形参声明为 `INTENT(in)` 是一个好的编程习惯。

### 3.3 子例程序

**SUBROUTINE** 子例程序名 (形参列表)

程序体

**END** [SUBROUTINE [子例程序名] ]

- 子例程序的调用

**CALL** 子例程序 (实参列表)

- 子例程序中使用 **RETURN** 返回主调过程
- 子例程序通过形参与主调过程进行参数传递，包括传入数据和传出数据
- 实参与形参结合时传递的是地址，类型要一致
- INTENT**: 形参可用 `INTENT` 来声明参数的值能否被修改

**SUBROUTINE**(arg1,arg2,arg3)

**REAL** **INTENT**(in) :: arg1

```
REAL INTENT(out) :: arg2
REAL INTENT(inout) :: arg3
REAL :: arg4, arg5
INTENT(in) :: arg4, arg5    ! INTENT 语句
```


- **SAVE**: 保留局部变量的值, 以便下次调用时使用, **SAVE** 的用法为:

```
REAL, SAVE :: x, y    ! 和类型声明一起使用

REAL :: x, y
SAVE :: x, y    ! 也可以单独使用

SAVE    ! 将所以局部变量都声明为带有 SAVE 属性
```

- 局部变量如果被初始化, 则自动具有 **SAVE** 属性
- 参数结合的一般规则: 形参与实参个数, 类型保持一致
- 数组作为参数: 确定大小, 可调数组, 假定形状数组
- 子例程序可以嵌套
- 子例程序中, 除形参外, 其他变量缺省为局部变量
- 参数个数不定时可使用: **OPTIONAL**
- 改变参数传递位置的方法

 子例程序中不建议使用 STOP 语句。

### 3.3.1 数组作为形参

- 显式结构的形参数组 (explicit-shaped dummy arrays)
  - 将数组的维数和大小显式传给子例程序, 例:


```
SUBROUTINE myroutine(A,m,n)
  INTEGER, INTENT(in) :: m, n
  REAL(8), INTENT(in), DIMENSION(m,n) :: A
  ... ..
END SUBROUTINE myroutine
```

- 假定维数的形参数组 (assumed-shaped dummy arrays)
  - 形参数组结构不定, 只有子例程序或函数有显式接口, 才能使用这种数组, 通常采用的方式是将子例程序放在模块 (MODULE) 中, 然后在主调函数中使用该模块。例:

```
MODULE mymodule
CONTAINS
```

```
SUBROUTINE myroutine(A,B)
  REAL(8), INTENT(in), DIMENSION(:, :) :: A
  REAL(8), INTENT(iout), DIMENSION(:, :) :: B
  ... ..
END SUBROUTINE myroutine


END MODULE mymodule
```

 当使用以上两种方法传递数组时，函数或子例程序可以对整个数组或部分数组元素进行操作，即可以进行数组运算。

- 不定大小的形参数组
  - 显式声明除最后一个维度以外的所有维度的长度，最后一个维度用星号代替。
  - 由于编译器不知道传递给子例程序的实际数组的长度，因此无法对整个数组或部分数组内容进行操作，如提取数组的某一列，或与其他数组进行数组运算等。

```
SUBROUTINE myroutine(A)
  REAL(8), DIMENSION(10,*) :: A
  ... ..
```

```
END SUBROUTINE myroutine
```

 这是 Fortran 早期版本的用法，无法获知实参的实际长度，因此不建议使用。

### 3.3.2 字符串作为形参

- 字符串作为形参时，可用星号来声明字符串的长度（形参仅仅是一个占位符，系统不需要给形参分配实际内存）
- 如果在过程中需要用到字符串长度，可通过内部函数 `len` 获取，如：

```
SUBROUTINE test_string(str)
  CHARACTER(LEN=*), INTENT(in) :: str
  CHARACTER(LEN=len(str)) :: str_tmp  ! 利用 str 的长度声明一个局部字符变量

  WRITE(*,'(1X,A,I3)') 'The length of str =', len(str)

END SUBROUTINE test_string
```

### 3.3.3 函数或子例程序作为形参

- 函数和子例程序也可以作形参



- 当函数或子例程序作为实参时，传递的是一个指向它们的指针
- 外部函数或子例程序作为实参时，需事先声明为外部的

```
EXTERNAL :: fun1, routine1, ...
```

- 内置函数或子例程序作为实参时，也需要事先声明

```
INTRINSIC :: fun1, routine1, ...
```

### 3.3.4 关键字参数和可选参数

- 通过关键字参数改变参数次序
  - 一般情况下，调用过程时，实参与形参必须匹配（个数，次序，类型），但如果过程接口是显式的，则可以通过关键字参数（keyword arguments）改变参数的次序。
  - 关键字参数一般形式

```
keyword=实参 ! 这里的 keyword 是与实参关联的形参
```

- 举例


```
myfun(a,b,c) ! 带显式接口的过程
```

CALL myfun(1.0,2.0,3.0) ! 正常调用

CALL myfun(b=2.0,c=3.0,a=1.0) ! 通过关键字参数改变次序

CALL myfun(1.0,c=3.0,b=2.0) ! 改变部分参数的次序

- 关键字参数通常和可选参数结合使用

 若调用过程时，实参列表中出现关键字参数，则该关键字参数后面的其他参数都必须用关键字参数。

- 可选参数

- 在定义显式接口的过程时，可将形参声明为可选的

**类型关键字 [(kind)], OPTIONAL [, 其它属性说明] :: 变量名**

- 判断可选参数是否出现：PRESENT 函数
- 若可选参数没有出现，但其后面的参数需要出现，此时需要使用关键字参数

### 3.3.5 星号作为形参

- 作用：为程序单元提供多个 RETURN 返回机制，如

```
SUBROUTINE myroutine(a,b,c,*,*,*)
```

- 程序单元中 RETURN 语句一般形式

```
RETURN k
```

- 当  $k < 1$  或  $k > n$  时, 正常返回, 这里  $n$  是形参中星号的个数
  - 当  $1 \leq k \leq n$  时, 返回与第  $k$  个星号所结合的实参所标识的语句处
- 调用时, 与星号相结合的实参是星号后面跟一个语句标号

```
CALL myroutine(x1,x2,x3,*100,*150,*20)
```

## 3.4 递归函数和递归子例程序

```
RECURSIVE FUNCTION 函数名 (形参列表)
```

```
... ..
```

```
END [FUNCTION [函数名] ]
```

```
RECURSIVE SUBROUTINE 子例程序名 (形参列表)
```

```
... ..
```

**END [SUBROUTINE [子例程序名]]**

- RECURSIVE: 在表明函数/子例程序是递归的
- 递归子例程序举例

Fortran 源代码 3.1 递归子例程序

```
1 RECURSIVE SUBROUTINE factorial (n, result)
2 IMPLICIT NONE
3
4 INTEGER, INTENT(IN) :: n ! Value to calculate
5 INTEGER, INTENT(OUT) :: result ! Result
6 INTEGER:: temp ! Temporary variable
7
8 IF (n>=1) THEN
9 CALL factorial(n-1, temp)
10    result = n*temp
11 ELSE
12    result 1
13 END IF
14 END SUBROUTINE factorial
```

- 为了避免递归函数中函数名的两张使用方法带来的麻烦，Fortran 允许为递归函数名和返回值指定不同的名字。

**RECURSIVE FUNCTION** 函数名 (形参列表) **RESULT(返回值变量名)**

- 递归函数举例

Fortran 源代码 3.2 递归函数

```
1 RECURSIVE FUNCTION fact(n) RESULT(answer)
2 IMPLICIT NONE
3 INTEGER, INTENT(IN) :: n ! Value to calculate
4 INTEGER :: answer ! Result variable
5
6 IF (n>=1) THEN
7     answer=n*fact(n-1)
8 ELSE
9     answer=1
10 END IF
11
12 END FUNCTION fact
```

### 3.5 内部过程 CONTAINS

- 包含在程序单元内部的过程

```
...  
CONTAINS    !表示后面定义的过程是内部过程  
  SUBROUTINE ...    !内部子例程序，只能被包含它的程序单元调用  
    ... ..  
  END SUBROUTINE ...  
  FUNCTION ...    !内部函数，只能被包含它的程序单元调用  
    ... ..  
  END FUNCTION ...  
END ...
```

- 内部过程必须放在其所在程序单元的所有可执行语句的后面
- 内部过程与外部过程的区别
  - 内部过程只能被其所在的程序单元调用
  - 内部过程继承其所在程序单元的所有数据
  - 内部过程名不能作为命令行参数传递给其他过程
- 如果内部过程中声明了与其所在程序单元同名的变量，则优先使用自己声明的变量

## 3.6 PURE 过程

- PURE 过程和 ELEMENTAL 过程都是为了适应并行计算而引入的，目的是提高程序在并行机上的执行效率。
- 进行并行计算的基本要求是不同运算之间保持相对独立。
- PURE 过程定义方式：

```
PURE FUNCTION/SUBROUTINE ...
```

```
... ..
```

```
END FUNCTION/SUBROUTINE ...
```

- PURE 函数有以下要求：
  - 局部变量不可以有 SAVE 属性，也不能初始化
  - 所有形参必须声明为 INTENT(in)
  - 被 PURE 函数调用的函数或子例程序也必须是 PURE 过程
  - 不能有任何的外部文件读写操作（PRINT, READ, WRITE, OPEN 等）
  - 不能有指针赋值语句和 STOP 语句
- 在 FORALL 结构中只能使用 PURE 函数
- 在声明语句中可以使用 PURE 函数
- Fortran 所有内置函数都是 PURE 函数

- 除了可以修改用 INTENT(out) 或 INTENT(inout) 声明的参数外, PURE 子例程序的限制和 PURE 函数一样。

### 3.7 ELEMENTAL 过程

- 数组运算

```
ELEMENTAL FUNCTION/SUBROUTINE ...  
...  
END FUNCTION/SUBROUTINE ...
```

- ELEMENTAL 函数的限制
  - 用户自定义的 ELEMENTAL 函数必须是 PURE 函数
  - 所有形参必须是标量, 不能带有 POINTER 属性
  - 函数返回值也必须是标量, 不能带有 POINTER 属性
  - 形参不能用在变量声明中, 因此无法使用自动数组
- ELEMENTAL 子例程序的限制同 ELEMENTAL 函数



## 3.8 模块 MODULE

- Fortran 可以通过模块实现各个程序单元之间的数据传递和共享
- 模块是一个独立编译的程序单元，可以包含数据和过程
- 其他程序单元可以通过 USE 语句来使用模块
- 使用同一个模块的不同程序单元可以访问同样的数据和过程（即模块中的数据和过程）
- 模块的作用：共享与复制
  - 取代 F77 中的共用区，提供共享常量、变量、类型声明和过程
  - 其他程序单元引用模块时，实际上就是把模块内的全部语句复制到程序中

### 3.8.1 模块的定义

- 模块定义方式：

**MODULE 模块名**

**类型声明部分**

**[ CONTAINS**     ! 模块可以只包含数据，也可以有内部过程


**内部过程 1**

**... ...**

**内部过程 n ]**

### END [MODULE [模块名]]

- 模块中的过程称为**模块过程**，必须在数据声明之后，并在前面加上 CONTAINS 语句
- MODULE 可以用来封装程序模块，通常把具备相关功能的函数及变量封装在一起


 为了通过模块达到共享数据的目的时，应将模块中的数据声明为 SAVE 属性。

### 3.8.2 模块的使用 USE

- 模块的使用：

#### USE 模块名 1, 模块名 2, ..., 模块名 N

- USE 语句需要放在程序单元最前面（除 PROGRAM 等语句）

 使用模块时，程序单元中不能再声明与模块中同名的变量。

### 3.8.3 USE 高级选项

- 限制对模块内容的访问：**PUBLIC**, **PRIVATE**
- 限制对模块中特定数据项的访问：**ONLY**
- 重命名数据项和过程

## 3.9 接口块 INTERFACE

- 关键字参数和可选参数只能在带显式接口的过程中使用
- EXTERNAL 和 INTRINSIC 声明外部过程时只能提供过程名，但不能提供接口细节（形参个数，类型，形状等）。由于主调过程与外部过程独立编写，分别编译，所以仅声明外部过程难以获取详细准确的调用信息，从而无法正确编译。
- 创建**显式接口**的方法：
  - 将过程放在模块中（模块中的过程始终具有显式接口），然后在主调过程中使用该模块
  - 但有些过程无法放入模块中，如程序库中的过程，或者其他语言编写的函数等，解决方法可以使用接口块
- 接口块指定了外部过程所有的接口特征，编译器根据接口块中的信息执行一致性检查
- 接口块的一般形式

```
INTERFACE
  interface body A
  interface body B
  ... ..
END INTERFACE
```

- **interface body** 用来指定外部过程的接口信息，内容包括过程声明，形参声明和过程

结束标志, 这些语句给编译器提供了检查接口一致性的足够信息。过程中的可执行语句不能出现在接口块中。如

```
INTERFACE
  SUBROUTINE mysort(A, n)
    IMPLICIT NONE
    REAL, DIMENSION(:), INTENT(inout) :: A
    INTEGER :: n
  END SUBROUTINE
END INTERFACE
```

- 这个类似于 C/C++ 中的函数声明。
- 将这段代码放入主调过程声明区 (紧跟 IMPLICIT NONE 之后, 类型声明语句之前), 就可以使用关键字参数了。
- 接口块中过程名, 形参个数和类型必须一致, 但形参名称可以不同。
- 接口块使用注意事项
  - 如果过程已经在所使用的模块中 (即已经有显式接口), 则不能再放入接口块中, 否则会引起显式接口的二次定义, 引起编译错误
  - 接口块常用于旧版本 FORTRAN 或其他语言的过程 (函数) 提供显式接口

- 如果需要为多个外部过程提供接口，则可以将它们放在一个模块中

```
MODULE module_interface  
  
INTERFACE  
  SUBROUTINE mysort(A, n)  
    IMPLICIT NONE  
    REAL, DIMENSION(:), INTENT(inout) :: A  
    INTEGER :: n  
  END SUBROUTINE  
  
  ...    ! 其他接口  
  
END INTERFACE  
  
END MODULE
```

## 3.10 通用过程

## 3.11 使用多个文件

**INCLUDE 文件名** !将相应的文件插入当前位置

### 3.12 作用域

- 一个对象（变量，符号常量，过程，语句标号等）的作用范围是程序中定义该对象的那一部分。
- Fortran 中有三层作用范围：
  - 全局：在整个程序执行过程中有定义，如程序名，外部过程名，模块名等
  - 局部：只在某个作用域内有定义，如局部变量
  - 语句：只在某条语句中有定义，如隐式 DO 循环和 FORALL 结构中的循环变量

## 第四章 派生结构和指针

### 4.1 派生数据类型

- 派生数据类型的定义

**TYPE** 派生类型名

成员组 1 类型说明

成员组 2 类型说明

... ..

**END TYPE** [派生类型名]

- TYPE** 是定义派生数据类型的关键字
  - TYPE** 与 **END TYPE** 之间只能关于结构成员的类型说明，不能有任何可执行语句
  - 派生类型成员为字符型时，其长度必须确定，不可用 \*
- 用派生数据类型声明变量

**TYPE(派生类型名) :: 变量列表**

- 使用派生类型成员方法：派生类型变量% 成员
- 可对派生类型变量直接赋值
- 构造函数：每定义一个派生类型，会自动生成与派生类型名相同的构造函数

## 4.2 指针

### 4.2.1 指针的声明

### 4.2.2 指针的引用

### 4.2.3 指针与派生类型

### 4.2.4 指针与动态控制

### 4.2.5 指针与数组

### 4.2.6 指针与过程



## 第五章 文件操作

### 5.1 文件的分类

- 按存取方式（读取记录的顺序）：
  - 顺序存取文件：从第一个记录开始，一个接一个地依次存取记录
  - 直接存取文件：可对任意指定的记录进行存取（也称随机文件）
- 按存放形式
  - 有格式存放（也以字符形式存放），通常称为文本文件，适合人类阅读。
  - 无格式存放（以二进制形式存放），适合计算机读取

### 5.2 文件读写

- 文件操作的一般步骤为：打开文件，读写操作，关闭文件。
- Fortran 95 文件操作函数有：

OPEN	打开文件
CLOSE	关闭文件
INQUIRE	查询文件属性
READ	从文件读取数据

WRITE	向文件写入数据
REWIND	将文件指针一道开头（顺序文件）
BACKSPACE	将文件指针往回移一个记录（顺序文件）
ENDFILE	将文件指针移到末尾（顺序文件）

### 5.3 打开文件 OPEN

#### OPEN(说明项)


- OPEN 语句的说明项有：

[UNIT=]n	设备号，代表所关联的文件，非负整数，一般不超过 99
FILE=fname	文件名，字符串表示
STATUS=char_expr	文件状态，取值有：'UNKNOWN'（缺省），'OLD'，'NEW'，'REPLACE'，'SCRATCH'
ACTION=char_expr	读写方式，取值有：'READWRITE'（缺省），'READ'，'WRITE'
IOSTAT=flag	返回打开状态，如果打开成功则 flag 的值为 0，否则表示不成功
ACCESS=char_expr	存取方式，取值有：'SEQUENTIAL'（缺省），'DIRECT'

FORM=char_expr	记录格式，取值有：'FORMATTED'，'UNFORMATTED'，缺省值：顺序文件为有格式，直接文件为无格式
RECL=int_expr	指定每条记录的长度，仅用于直接文件：对于有格式文件，以字符个数为单位，而对于无格式文件，以处理器相关的度量单位（通常为字节）为单位
POSITION=char_expr	文件打开后文件指针的位置，取值有：'ASIS'（缺省），'REWIND'，'APPEND'
DELIM=char_expr	指定表控输出中，字符串之间的分隔符，取值有：'NONE'（缺省），'APOSTROPHE'，'QUOTE'
PAD=char_expr	格式化输入的记录是否填充空格，取值有：'YES'（缺省），'NO'
BLANK=char_expr	空格被视为空或零，取值有：'NULL'（缺省），'ZERO'
ERR=label	文件打开失败时，转向标号为 label 的语句

- 文件状态：'SCRATCH' 表示创建一个临时文件，关闭后自动被删除
- 顺序文件是指对文件的读写是按照从头到尾的顺序进行，而直接文件是指每次可以访问指定的记录，而不必按顺序依次访问，直接文件中每条记录的长度必须相等
- 有格式文件是指将数据按人所能识别的字符存储，而无格式文件中的数据是计算机内存中数据的精确拷贝

- 文件指针的位置：'REWIND' 表示指向第一条记录，'APPEND' 表示指向最后一条记录之后，文件结束符之前，'ASIS' 表示不定，与系统有关
- IOSTAT=flag 是唯一一个带返回值的选项，其中 flag 是事先已经声明的整型变量
- BLANK 和 ERR 选项建议不再使用

 尽量使用 IOSTAT 选项来捕获可能存在的文件打开错误。

## 5.4 关闭文件 CLOSE

### CLOSE(说明项)

- CLOSE 语句的说明项有：

[UNIT]=n	设备号，代表所需要关闭的文件
STATUS=char_expr	是否保留文件，取值有：'KEEP'（缺省），'DELETE'
IOSTAT=flag	返回关闭状态，如果关闭成功则 flag 的值为 0，否则表示不成功
ERR=label	文件关闭失败时，转向标号为 label 的语句，建议不再使用

## 5.5 文件状态查询 INQUIRE

### INQUIRE(说明项)

- INQUIRE 语句的说明项有：

[UNIT=]n	设备号
FILE=fname	文件名
IOSTAT=int_var	返回 I/O 状态, 0 表示成功, 否则表示不成功
EXIT=log_var	文件是否存在, 返回值有: <b>.TRUE.</b> , <b>.FALSE.</b>
OPENED=log_var	文件是否打开, 返回值有: <b>.TRUE.</b> , <b>.FALSE.</b>
NAMED=log_var	文件是否有名, 返回值有: <b>.TRUE.</b> , <b>.FALSE.</b>
NAME=char_var	若文件有名, 则返回文件名, 否则无意义
NUMBER=int_var	若文件打开, 返回文件关联的设备号, 否则没有意义
ACCESS=char_var	返回文件的存取方式
SEQUENTIAL=char_var	文件能否按顺序方式访问, 返回值有: <b>'YES'</b> , <b>'NO'</b> , <b>'UNKNOWN'</b>
DIRECT=char_var	文件能否按直接方式访问, 返回值有: <b>'YES'</b> , <b>'NO'</b> , <b>'UNKNOWN'</b>

FORM=char_var	若文件打开, 返回文件的格式
FORMATED=char_var	文件是不是有格式的, 返回值有: 'YES', 'NO', 'UNKNOWN'
UNFORMATED=char_var	文件是不是无格式的, 返回值有: 'YES', 'NO', 'UNKNOWN'
RECL=int_var	返回记录的长度, 仅用于直接文件
NEXTREC=int_var	返回最后从文件读出或写入的记录个数 + 1, 仅用于直接文件
BLANK=char_var	空格被视为空还是 0, 返回值有: 'NULL', 'ZERO'
POSITION=char_var	文件首次打开时, 文件指针的位置, 返回值有: 'ASIS', 'REWIND', 'APPEND', 'UNDEFINED'
ACTION=char_var	返回文件读写方式, 返回值有: 'READWRITE', 'READ', 'WRITE', 'UNDEFINED'
READ=char_var	文件是否以只读方式打开, 返回值有: 'YES', 'NO', 'UNKNOWN'
WRITE=char_var	文件是否以只写方式打开, 返回值有: 'YES', 'NO', 'UNKNOWN'
READWRITE=char_var	文件是否以读写方式打开, 返回值有: 'YES', 'NO', 'UNKNOWN'

DELIM=char_var	返回表控输出或有名 I/O 列表中，字符串之间的分隔符，返回值有：'NONE'，'APOSTROPHE'，'QUOTE'，'UNKNOWN'
PAD=char_var	格式化输入的记录是否填充空格，返回值有：'YES'，'NO'
ERR=label	返回如果操作失败，程序应执行的语句的标号（建议不再使用）

- INQUIRE 语句另一种用法

#### INQUIRE(IOLENGTH=int\_var) 输出列表

- 返回输出列表中无格式记录的长度

## 5.6 读文件

#### READ(说明项) 输入变量列表

- READ 语句的说明项有：

[UNIT=]n	设备号
[FMT=]fm	格式说明，可以是 FORMAT 语句的标号，或 *，或字符串表示的格式说明

IOSTAT=int_var	返回操作状态，0 表示成功，正数表示读文件失败，-1 表示文件结束，-2 表示记录结束
REC=int_expr	指定读取记录的个数，仅用于直接文件
NML=namelist	指定要读取的 I/O 实体的命名表
ADVANCE=char_expr	指定是否进行高级 I/O，仅用于顺序文件
SIZE=int_var	指定非高级 I/O 中读取的字符个数，仅用于非高级 I/O
DELIM=char_expr	暂时重载 OPEN 语句中的定界原值，取值有：'NONE'，'APOSTROPHE'，'QUOTE'
EOR=label	非高级 I/O 中，达到文件结束标志，程序转向标号为 label 的语句（不再使用）
END=label	达到文件结束标志，程序转向标号为 label 的语句（不再使用）
ERR=label	文件操作失败，程序转向标号为 label 的语句（不再使用）

- 举例：

```
read(99,REC=k,IOSTAT=ferr)
```



## 5.7 写文件

**WRITE([UNIT=] 设备号, 格式说明) 输出列表**

- WRITE 语句的说明项与 READ 相同, 除了 END, SIZE, EOR。

## 5.8 矩阵读写举例

- 将一个双精度矩阵  $A$  写入二进制文件, 可以有三种方式实现

Fortran 源代码 5.1 将双精度矩阵写入文件

```
1  ! 1) 将每个元素看成一个记录, recl=8
2  OPEN(UNIT=99, FILE='data.dat', ACCESS='direct', &
3      & FORM='unformatted', RECL=8)
4  k=0
5  DO j=1,n
6      DO i=1,m
7          k=k+1
8          WRITE(99, REC=k) A(i,j)
9      ENDDO
```

```
10 ENDDO
11 CLOSE(UNIT=99)
12
13 ! 2) 把一列看成一个记录, 长度为 8*m
14 OPEN(UNIT=99, FILE='data.dat', ACCESS='direct', &
15      & FORM='unformatted', RECL=8*m)
16 DO j=1,n
17     WRITE(99, REC=j) A(1,j)
18 ENDDO
19 CLOSE(UNIT=99)
20
21 ! 3) 把整个看成一个记录, 长度为 8*m*n
22 OPEN(UNIT=99, FILE='data.dat', ACCESS='direct', &
23      & FORM='unformatted', RECL=8*m*n)
24 WRITE(99, REC=1) A
25 CLOSE(UNIT=99)
```

- 从二进制文件中读取数据, 存入矩阵  $B$  中

Fortran 源代码 5.2 将双精度矩阵写入文件

```
1  ! 1) 将每个元素看成一个记录, recl=8
2  OPEN(UNIT=99, FILE='data1.dat', ACCESS='direct', &
3      & FORM='unformatted', RECL=8)
4  k=0
5  DO j=1,n
6      DO i=1,m
7          k=k+1
8          READ(99, REC=k, IOSTAT=ferr) A(i,j)
9      ENDDO
10 ENDDO
11 CLOSE(UNIT=99)
12
13 ! 2) 把一列看成一个记录, 长度为 8*m
14 OPEN(UNIT=99, FILE='data1.dat', ACCESS='direct', &
15     & FORM='unformatted', RECL=8*m)
16 DO j=1,n
17     READ(99, REC=j, IOSTAT=ferr) (A(i,j), i=1,m)
18 ENDDO
19 CLOSE(UNIT=99)
20
```

```
21 ! 3) 把整个看成一个记录, 长度为 8*m*n
22 OPEN(UNIT=99, FILE='data1.dat', ACCESS='direct', &
23      & FORM='unformatted', RECL=8*m*n)
24 READ(99, REC=1, IOSTAT=ferr) ((A(i,j), i=1,m), j=1,n)
25 CLOSE(UNIT=99)
```

## 5.9 文件定位

**BACKSPACE([UNIT=] 设备号)** ! 回退一个记录  
**REWIND([UNIT=] 设备号)** ! 返回到文件开头

## 5.10 ENDFILE 语句

- ENDFILE 语句在顺序文件的当前位置写入一个文件结束记录, 然后定位在该记录之后

**ENDFILE(说明项)**

## 5.11 有名 I/O 列表

- A NAMELIST is just a list of variable names that are always read or written as a group.

**NAMELIST/nl-group-name/变量列表**

- NAMELIST 是声明语句，必须位于可执行语句之前
- 表控有名列表的 WRITE 和 READ 语句：

**WRITE(UNIT=n,NML=namelist, ...)**

**READ(UNIT=n,NML=namelist, ...)**

- 举例

### Fortran 源代码 5.3 有名 I/O 举例

```
1 PROGRAM write_namelist
2 IMPLICIT NONE
3 INTEGER :: i=1, j=2
4 REAL :: a=-99.0, b=0.0
5 CHARACTER(LEN=12) :: str='test string'
6 NAMELIST / mylist / i, j, string, a, b
7
```

```
8 OPEN(8, FILE='output.nml', DELIM='apostrophe')
9 WRITE(UNIT=8, NML=mylist)
10 CLOSE(8)
11 END PROGRAM write_namelist
```

## 第六章 其他

### 6.1 面向对象编程

### 6.2 重载

#### 6.2.1 子程序重载

#### 6.2.2 运算符重载

#### 6.2.3 赋值号重载

### 6.3 公共数据

#### 6.3.1 公共区 COMMON

#### 6.3.2 共享存储 EQUIVALENCE

#### 6.3.3 数据块 BLOCK DATA

### 6.4 好的编程习惯

- 使用有意义的变量名（见名识意）
- IMPLICIT NONE

- 输入数据时给出提示信息
- 将不变的数据声明为常量
- 初始化所有变量



## 6.5 Fortran 内置函数和子例程序

Fortran 95 内置函数和子例程序

函数名	说明
<b>参数存在查询函数</b>	
PRESENT	存在参数
<b>数值函数</b>	
ABS (A)	绝对值
AIMAG (Z)	复数的虚部
AINIT (A [, KIND])	整数截尾
ANINT (A [, KIND])	最近的整数
CEILING (A [, KIND])	大于或等于数值的最小整数
CMPLX (X [, Y, KIND])	转换为复数类型
CONJG (Z)	共轭复数
DBLE (A)	转换为双精度实数类型
DIM (X, Y)	正偏差
DPROD (X, Y)	双精度实数乘积

FLOOR (A [, KIND])	小于或等于数值的最大整数
INT (A [, KIND])	转换为整数类型
MAX (A1, A2 [, A3,...])	最大值
MIN (A1, A2 [, A3,...])	最小值
MOD (A, P)	余数函数
MODULO (A, P)	模数函数
NINT (A [, KIND])	最近的整数
REAL (A [, KIND])	转换为实数类型
SIGN (A, B)	符号传输

### 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)	常用对数 ( 10 为基数 )
SIN (X)	正弦
SINH (X)	双曲正弦
SQRT (X)	平方根
TAN (X)	正切
TANH (X)	双曲正切

### 字符函数

ACHAR (I)	按 ASCII 整理序列排列时给定位置的字符
ADJUSTL (STRING)	齐左调整
ADJUSTR (STRING)	齐右调整
CHAR (I [, KIND])	按处理器整理序列排列时给定位置的字符
IACHAR (C)	按 ASCII 整理序列排列时字符的位置
ICHAR (C)	按处理器整理序列排列时字符的位置
INDEX (STRING, SUBSTRING [, BACK])	子串的起始位置

LEN_TRIM (STRING)	长度不包含结尾的空白字符
LGE (STRING_A, STRING_B)	词法上大于或等于
LGT (STRING_A, STRING_B)	词法上大于
LLE (STRING_A, STRING_B)	词法上小于或等于
LLT (STRING_A, STRING_B)	词法上小于
REPEAT (STRING, NCOPIES)	重复并置
SCAN (STRING, SET [, BACK])	扫描字符串以查找集中的某个字符
TRIM (STRING)	删除结尾的空白字符
VERIFY (STRING, SET [, BACK])	检验字符串中的字符集
LEN (STRING)	字符实体的长度

### KIND ( 种类 ) 函数

KIND (X)	返回给定数据的 kind ( 种类类型参数 ) 值
SELECTED_INT_KIND (R)	返回满足指定范围的整数 kind 值
SELECTED_REAL_KIND ([P, R])	返回满足指定精度和范围的实数 kind 值

### 逻辑函数

LOGICAL (L [, KIND])	在种类类型参数不相同的逻辑类型对象之间转
----------------------	----------------------

### 数值查询函数

DIGITS (X)	模型的有效数字数
EPSILON (X)	与此相比几乎可以忽略的数值
HUGE (X)	模型中最大的数值
MAXEXPONENT (X)	模型的最大指数
MINEXPONENT (X)	模型的最小指数
PRECISION (X)	十进制精度
RADIX (X)	模型的基数
RANGE (X)	十进制指数范围
TINY (X)	模型中最小的正数

### 位查询函数

BIT_SIZE (I)	模型中的位数
--------------	--------

### 位操作函数

BTEST (I, POS)	位测试
IAND (I, J)	逻辑 AND
IBCLR (I, POS)	清除位

IBITS (I, POS, LEN)	提取位
IBSET (I, POS)	设置位
IEOR (I, J)	互斥 OR
IOR (I, J)	包容 OR
ISHFT (I, SHIFT)	逻辑移位
ISHFTC (I, SHIFT [, SIZE])	循环移位
NOT (I)	逻辑补充

### 传送函数

TRANSFER (SOURCE, MOLD [, SIZE])	处理第一个参数，就好象它与第二个参数属于同一种类型
----------------------------------	---------------------------

### 浮点处理函数

EXPONENT (X)	型号的指数部分
FRACTION (X)	数值的小数部分
NEAREST (X, S)	指定的方向最近的不同处理器
RRSPACING (X)	接近指定数值的型号相对间隔的倒数
SCALE (X, I)	实数乘以基数得出整数幂

SET_EXPONENT (X, I)	设置数值的指数部分
SPACING (X)	接近指定数值的型号的绝对间隔

### 矩阵向量乘法函数

DOT_PRODUCT (VECTOR_A, VECTOR_B)	两个一级数组的点乘积
MATMUL (MATRIX_A, MATRIX_B)	矩阵乘法

### 约简数组函数

ALL (MASK [, DIM])	如果所有的值为 True 则为 True
ANY (MASK [, DIM])	如果任意值为 True 则为 True
COUNT (MASK [, DIM])	数组中 True 元素数
MAXVAL (ARRAY, DIM [, MASK]) 或 MAXVAL (ARRAY [, MASK])	数组中的最大值
MINVAL (ARRAY, DIM [, MASK]) 或 MINVAL (ARRAY [, MASK])	数组中的最小值
MAXLOC (ARRAY, DIM [, MASK]) 或 MAXLOC (ARRAY [, MASK])	数组中最大值的位置
MINLOC (ARRAY, DIM [, MASK]) 或 MINLOC (ARRAY [, MASK])	数组中最小值的位置

PRODUCT (ARRAY, DIM [, MASK]) 或 PRODUCT (ARRAY [, MASK])	数组元素的乘积
SUM (ARRAY, DIM [, MASK]) 或 SUM (ARRAY [, MASK])	数组元素的求和
ALLOCATED (ARRAY)	数组分配状态
LBOUND (ARRAY [, DIM])	数组的维数下界
SHAPE (SOURCE)	数组或标量的形式
SIZE (ARRAY [, DIM])	数组中的元素总数
UBOUND (ARRAY [, DIM])	数组的维数上界

### 数组构造函数

MERGE (TSOURCE, FSOURCE, MASK)	在屏蔽下合并
PACK (ARRAY, MASK [, VECTOR])	在屏蔽下将数组压缩为一级数组
SPREAD (SOURCE, DIM, NCOPIES)	增加维数以复制数组
UNPACK (VECTOR, MASK, FIELD)	在屏蔽下将一级数组解压缩为数组

### 数组整形函数

RESHAPE (SOURCE, SHAPE[, PAD, ORDER])	数组整形
---------------------------------------	------



### 数组处理函数

CSHIFT (ARRAY, SHIFT [, DIM])	循环移位
EOSHIFT (ARRAY, SHIFT [, BOUNDARY, DIM])	结束移位
TRANSPOSE (MATRIX)	调换两级数组

### 指针关联状态函数

ASSOCIATED (POINTER [, TARGET])	关联状态查询或比较
NULL ([MOLD])	返回分离的指针

### 系统环境调节过程

COMMAND_ARGUMENT_COUNT ()	返回命令参数的数目
GET_COMMAND ([COMMAND, LENGTH, STATUS])	返回调用程序的整个命令
GET_COMMAND_ARGUMENT (NUMBER [, VALUE, LENGTH, STATUS])	返回一个命令参数
GET_ENVIRONMENT_VARIABLE (NAME [, VALUE, LENGTH, STATUS, TRIM_NAME])	获得环境变量的值

### 内部子例程序

CPU_TIME (TIME)	获取处理器的时间
DATE_AND_TIME ([DATE, TIME, ZONE, VALUES])	获取日期和时间
MVBITS (FROM, FROMPOS, LEN, TO, TOPOS)	将位从一个整数复制到另一个整数
RANDOM_NUMBER (HARVEST)	返回伪随机数值
RANDOM_SEED ([SIZE, PUT, GET])	初始化或重新启动伪随机数据产生器
SYSTEM_CLOCK ([COUNT, COUNT_RATE, COUNT_MAX])	从系统时钟中获取数据

## 6.6 Fortran 90/95 语句

### Fortran 90/95 statements

#### 非执行语句 NON-EXECUTABLE STATEMENTS

#### 程序单元和子程序 Program Units and Subprograms

**PROGRAM** program-name

**MODULE** module-name

**END [MODULE [module-name]]**

**USE** module-name [,rename-list]

**USE** module-name, only: [only-list]

**PRIVATE** [[:] access-id-list]

**PUBLIC** [[:] access-id-list]

**EXTERNAL** external-name-list

**INTRINSIC** [[:] intrinsic-name-list    **! :: in Fortran 95 only**

[prefix] **SUBROUTINE** subroutine-name ([形参列表])  
where prefix is recursive, pure or elemental

[prefix] **FUNCTION** function-name ([形参列表]) [result(result-name)]  
where prefix is type [recursive] or recursive [type] or also, for Fortran 95, pure or elemental

---

ENTRY entry-name [(形参列表)] [result(result-name)]

---

INTENT (inout) [::] 形参列表 where inout is in, out, or in[ ]out

---

OPTIONAL [::] 形参列表

---

SAVE [::] 变量列表

---

INTERFACE [generic-spec]  
where generic-spec is 通用名, 自定义运算符或赋值号 (=)

---

END interface [generic-spec]

---

MODULE PROCEDURE procedure-name-list

---

### Data Specification

---

TYPE [[, attribute] ... ::] entity-list  
where type is integer, real, logical, complex, character, double precision, or type(type-name)  
attribute is parameter, public, private, pointer, target, allocatable, dimension, intent, external,  
intrinsic, optional or save

---

IMPLICIT NONE

---

IMPLICIT TYPE(letter-spec-list) [,type(letter-spec-list)] ...

---

TYPE [[, access] ::] type-name    **! access is public or private**

---

TYPE[,component-attr] ... ::] component-decl-list

---

where component-attr is pointer or dimension and component-decl is component-name

END type [type-name]

SEQUENCE

DATA object-list/value-list/ [[,] object-list/value-list/]...

BLOCK DATA [block-data-name]

END [block[ ]data [block-data-name]]

PARAMETER (named-constant-definition-list)

NAMelist /nl-group-name/variables [[,]/nl-group-name/variables] ...

DIMENSION [::] array(array-spec) [, array(array-spec)] ...

ALLOCATABLE [::] array-name [(array-spec)] [,array-name [(array-spec)]]...

POINTER [::] object-name [(array-spec)] [,object-name [(array-spec)]]...

TARGET [::] object-name [(array-spec)] [,object-name [(array-spec)]]...

EQUIVALENCE (object, object-list) [, (object, object-list)]...

COMMON [/[cname]/]vlist [[,]/[cname]/ vlist]...

## EXECUTABLE STATEMENTS

### Assignment

variable=expr

---

pointer=>target

---

IF (scalar-logical-expr) action-stmt

---

WHERE (logical-array-expr) array-variable=expr

---

FORALL (index-spec [,index-spec] ... [,scalar-logical-expr]) assignment

---

### Program Units and Subprograms

---

CALL subroutine-name ([[actual-argument-list]])

---

RETURN

---

END [unit [unit-name]] where unit is program, subroutine, or function

---

### Dynamic Storage Allocation

---

ALLOCATE(allocation-list [, stat=stat] )

---

DEALLOCATE(allocate-object-list [, stat=stat] )

---

NULLIFY(pointer-object-list)

---

### Control Constructs

---

[do-name:] DO do-variable=integer-expr,integer-expr[,integer-expr]

---

[do-name:] DO WHILE(scalar-logical-expr)

---

CYCLE [do-name]

---

---

EXIT [do-name]

---

CONTINUE

---

END DO [do-name]

---

[if-name:] IF(scalar-logical-expr) THEN

---

ELSE IF(scalar-logical-expr) THEN [if-name]

---

ELSE [if-name]

---

END IF [if-name]

---

[select-name:] SELECT[ ]CASE (scalar-expr)

---

CASE (case-value-list) [select-name]

---

CASE DEFAULT [select-name]

---

END SELECT [select-name]

---

GOTO label

---

STOP [access-code]

---

[where-name:] WHERE (logical-array-expr)

---

ELSEWHERE [where-name]

---

ELSEWHERE (logical-array-expr) [where-name] (Fortran 95)

---

END WHERE [where-name]

---

---

[forall-name:] FORALL(index-spec[,index-spec] ... [,scalar-logical-expr])

---

END FORALL [forall-name]

---

### Input-Output

---

READ(control-list) [input-list]

---

READ format [, input-list]

---

WRITE(control-list) [output-list]

---

PRINT format [, output-list]

---

REWIND external-file-unit

---

REWIND(position-list)

---

END file external-file-unit

---

END file (position-list)

---

BACKSPACE external-file-unit

---

BACKSPACE(position-list)

---

OPEN(connect-list)

---

CLOSE(close-list)

---

INQUIRE(inquire-list)

---



---

INQUIRE(iolength = length) olist

---

FORMAT([format-list])

---