Fortran 语言

－－ Fortran77 结构化程序设计

第一章 算 法

1.1 概念：

算法是为了解决一个问题所采取的步骤。

1.2 算法举例

1.3 算法的特性

有穷性、确定性、有零个或多个输入、有一个或多个输出、有效性。

1.4 算法的表示方法

1、自然语言。

2、流程图：传统流程图，改进的流程图。

3、结构化程序的三种基本结构。结构化包括两方面的内容：

（1）模块化

（2）三种基本结构为程序的基本单元

- ✓ 顺序结构
- ✓ 选择结构（分支结构）
- ✓ 循环结构

第二章 计算机和计算机程序

2.1 计算机是实现算法的有效工具

2.2 计算机的基本组成

2.3 计算机中存储信息的方法

2.4 计算机语言和计算机程序

一、计算机语言的分类

1、机器语言：面向机器

2、汇编语言（符号语言）：面向机器

3、高级语言（算法语言）：可移植性

如：BASIC，FORTRAN，C，PASCAL，C++，JAVA 等

二、计算机程序

程序 = 算法 + 数据结构

2.5 程序运行环境

一、编辑源程序

二、编译程序：将源程序翻译成机器语言的过程

1、汇编语言：

2、高级语言：

三、连接程序

2.6 程序开发的步骤

- ✓ 分析问题，画流程图，书写源程序（课堂）
- ✓ 输入、编辑源程序
- ✓ 编译
- ✓ 连接
- ✓ 运行

第三章 FORTRAN 语言程序设计初步

3.1 FORTRAN 语言发展概况

- ✓ Formula Translation
- ✓ 适用于工程及科学计算的一种高级程序设计语言
- ✓ 1951 年由约翰·贝克斯等人开始研究 Fortran 语言；
- ✓ 1957 年第一个 Fortran 程序在 IBM704 机上运行；
- ✓ 1958 和 1962 年先后推出 FortranⅡ和 FortranⅣ；
- ✓ 1966 和 1978 年先后颁布 Fortran66 和 Fortran77；
- ✓ 1991 年颁布了 Fortran90

3.2 简单的 FORTRAN 程序分析

Fortran 程序的基本结构：

⑴ 一个 Fortran 源程序由一个或多个程序单位组成，每个独立的程序单位以"end"语句结束。

⑵ 每个程序单位包括若干行（不能一行写多条语句，但可以几行写一条语句）

- ✓ 语句行（执行语句和非执行语句行）
- ✓ 非语句行（注释行）

⑶ 语句前可不设标号，也可根据需要设标号。

⑷ 各类语句在程序单位中的位置有一定规则。

⑸ 程序必须按规定格式书写。

3.3 FORTRAN 源程序的书写格式

Fortran77 源程序必须按以下格式书写：

⑴ 每行只能在 80 列内书写，并把 80 列分为 4 个区。

⑵ 1～5 列：标号区（ 1～5 位整数；第 1 列为"*"或"c"时，为注释行）

⑶ 第 6 列：续行标志区（非空格或非零字符；最多 19 个续行）

⑷ 7～72 列：语句区（书写语句；一行只能写一条语句）

⑸ 73～80 列：语句注释区（一般作编号注释）

3.4 Fortran 程序的编辑与运行

一、创建源程序文件并编写源程序

二、编译并连接源文件

三、运行程序编译生成的可执行文件

3.5 常 量

常量：在程序执行期间其值固定不变的量

Fortran 处理六种类型的常量：

⑴ 整型常量（Integer）

⑵ 实型常量（Real）

⑶ 双精度常量（Double precision）

⑷ 复型常量（Complex）

⑸ 逻辑型常量（Logical）

⑹ 字符型常量（Character）

3.5.1 整型常量

整数（Integer）：包括正、负整数和零。

说明： ①在（16 位）微机中，整数的取值范围为： -2^{15}～$2^{15}-1$（-32768～32767）

②在（32 位）微机中，整数的取值范围为： -2^{31}～$2^{31}-1$（-2147483648～2147483648 ）

3.5.2 实型常量

* 实数（real）
* 两种表示形式：
    1. 小数形式
    2. 指数形式（通常表示较大或较小的数）

说明：

⑴ 一个数值用指数形式表示时有两种表达方式：

①一个基本实数后面跟指数部分。如：0.876 → 8.76E-1

②一个整数后面跟指数部分。如：0.876 → 876E-3

⑵ 同一个数值可用不同的指数形式表示。如： 0.876→8.76E-1→87.6E-2→876E-3

⑶ 计算机在输出时，按标准化指数形式输出。如：0.28→2.800000E-01

即数字部分大于 1，且小数点前只有一位非零数字的指数输出形式。

⑷ 在微机中，一般用四个字节存放一个实数，其取值范围为：10-38～1038。

超出此范围时为"溢出"错误。

⑸ 下列形式为不合法的指数形式：

①单独的小数点和单独的指数部分。 如：.E5， E10

②指数部分只能是整数，不能带小数点。 如：8E0.5， 12.3E1.5

3.6 变 量

3.6.1 变量的概念

变量：在程序执行期间其值可以改变的量。

Fortran 为每一个变量分配一个相应的存储单元;每个变量在每个时刻只能有一个确定的值。如：x=4.5

注：在程序中用到的变量都必须要有确定的值

3.6.2 变量名

变量名：标识一个变量所用的名字。

命名规则：

- ✓ 变量名的第一个字符必须是字母;
- ✓ 在第一个字符后可跟 1～5 个字母或数字。

如：x，m1，total，k123，...

注意几点：

⑴ 变量名不区分大小写字母。如：TOTAL，Total，total 代表同一变量

⑵ 变量名中的空格不起作用。如：abc，a bc，ab c，a b c 代表同一变量

⑶ 允许变量名与语言中具有特定含义的字（"保留字"）同名。但建议不要使用。如：sin、read、end ...

⑷ 尽量"见名知义"，如:root、aver、result 等

3.6.3 变量类型

不同类型的变量用来存放不同类型的常量数据。由于常量具有六种类型，因此变量同样具有六种类型，这里只先介绍两种：

- ✓整型变量：存放整型常量
- ✓实型变量：存放实型常量

不同类型的数据在内存中的存储长度是不同的，因此，在程序中使用变量，应首先对其类型进行说明。变量类型的三种说明方法：

1. 隐含约定（I～N 规则）

以 i、j、k、l、m、n 开头的变量为整型变量，以其它字母开头的变量为实型变量（"I～N 规则"）

如：imax，m5，number，... 为整型变量； v，h2，aver，... 为实型变量

注：方便，但只能用来区分整型与实型。

2. 类型说明语句

可以用类型说明语句专门指定某些变量的类型。Fortran 中有六个类型说明语句：

⑴ integer（整型说明语句）
⑵ real（实型说明语句）
⑶ double precision（双精度说明语句）
⑷ complex（复型说明语句）
⑸ logical（逻辑型说明语句）
⑹ character（字符型说明语句）

3. IMPLICIT 语句（隐式说明语句）

可以用 IMPLICIT 语句将某一字母开头的全部变量指定为所需的类型，还可以用一个 IMPLICIT 语句同时指定几种类型。例如：

```
implicit integer (a, c, t-v)
implicit real (i, j)
implicit integer (a, b), real(i,k), integer (x-z)
```

几点说明：

⑴ 三种类型说明的优先级：

高　　类型说明语句（显式说明）
　　implicit 语句说明
低　　"I～N 规则"

⑵ 均为非执行语句，位于所有可执行语句的前面

⑶ implicit 语句应放在所有的说明语句的前面

⑷ 只在本程序单位内有效

3.7　Fortran 内部函数

在编制 Fortran 程序时，经常要用到一些基本的数学函数，如三角函数、指数函数、对数函数等。为方便用户，Fortran 编译系统已提供了这些函数的计算程序，需要时即可调用。

Fortran 内部函数调用格式：

函数名（自变量）

如：sin(90*3.14159/180)　→1.0

注意几点：

⑴ 函数自变量(参数)必须用括号括起来(一个或多个)。　　如：sinx+cosx　→　sin(x)+cos(x)

⑵ 函数自变量可以是常量、变量或表达式。　　如：　exp(3.0)，sin (x+cos(x))

⑶ 函数的类型是由函数值的类型确定的,但有些函数的类型是由自变量的类型确定的。

如：mod(8.0,3.0) →2.0；mod(8,3) →2

⑷ 三角函数的角度单位是"弧度"而不是"度"。

⑸ 注意书写方式

3.8　Fortran 算术表达式

表达式：是用一些特定的运算符将 Fortran 基本成分；连接起来的具有确定意义的式子。

基本成分包括：常量,变量,函数,字符串,数组等

Fortran 规定有四种表达式：

* 算术表达式
* 逻辑表达式（第四章）
* 字符表达式
* 关系表达式（第四章）

3.8.1 算术运算符和运算优先级

1、5 种算术运算符：　　+，-，*，/，**（乘方）

2、优先级：（）→ ** → *，/ → +，-

3.8.2 算术运算式

* 运算符：算术运算符

* 各运算元素：都是算术量

* 表达式的结果：也是算术量

例：2.34*A，　B**2-SQRT（C）/D，（A+B）/2.*H

注意：

1、书写问题：

① "/"号，如：（A+B）/（C+D）

② "*"不能省略，如：2AB - 2*A*B

③ 括号不分大小，均用（）表示，成对出现。如：((((A+1)+1)+1)+1)

④ 多次乘方按"先右后左"的原则处理。如：4**3**2 (4**（3**2）

⑤ 单边运算符："-"号

注：运算符不能连续出现,要用小括号隔开。如：A*（-B）-T/（-Q）

2、运算顺序：

（）→函数→ ** → *，/ → +，-

3.8.3 表达式运算中的类型问题

两种情况：

1、运算类型相同：结果仍为原类型

注意：两个整数相除，结果仍为整数。3/2=1.5 不正确，解决办法：先转换为实数在相除（2→2.0）

2、运算类型不同：编译系统自动转换为同一类型：整型 —> 实型

注意：类型转换从左到右进行，遇到不同类型运算时才进行转换。例：4/5+3*2.4

3.8.4 运算的误差问题

1、溢出：超出有效数值范围

解决：很大或很小的数用实型的指数形式表示

2、误差：由于有效数字的位数限制，实型数运算存在误差

解决 1：转换为双精度型

解决 2：避免因书写不当造成有效数字的丢失

如：0.001+1234567.0-1234566.0

3.9 赋值语句

作用：将一个确定的值赋给一个变量

一般格式：　V = e

变量 = 表达式

例：x=3.2

Y=SQRT(x+5.0)/2.0

说明：

1、"=" 为赋值号，其作用为：x ← 3.2

特殊：I=I+1(计数器）

2、左端只能是变量名，如：x+y=5 是不合法的

3、类型转换问题

- 类型相同，直接赋值（I=3）
- 类型不同，先计算表达式的值，然后将结果转换为左端变量的数据类型

例：　I=3.6*5+1.5

X=（5+6）/（8-5）

- 为避免出现类型转换过程中的错误，应保证两侧类型一致。

3.10 简单输出语句

* 输出语句的作用- 将内存的数据传送到显示器、打印机或保存到磁盘指定区域。
* 输入/输出三要素：
- 对象：哪些数据
- 格式
- 设备
* 输出语句的分类：
- 格式输出（第七章）
- 表控格式输出（系统标准格式）
- 无格式输出

3.10.2 表控输出语句

* 按计算机系统规定的格式输出
* 系统自动为每种类型的数据规定了列数

一、表控输出格式

1.整数的表控格式输出

规定：每个整数占 11 列，数印右端，左补空格

例：print *，123，-1128

输出结果：

␣␣␣␣␣␣␣␣123␣␣␣␣␣␣-1128

2.实数的表控格式输出

规定：每个实数占 15 列，数印右端，左补空格,小数部分占 6 列。

例：print *，15.4，-321.45

输出结果：

␣␣␣␣␣␣15.400000␣␣␣-321.450000

* 当实数值的绝对值≥107 或＜1 是时，按标准的指数形式输出。
* 共占 15 列，指数 4 列，小数 6 列

例：print *，-10000000.，0.98

输出结果：

␣␣-1.000000E+07␣␣␣9.800000E-01

二、表控格式输出语句

一般格式：

print * ,〈输出表列〉

write (* , *)〈输出表列〉

例 1： print * ,56.8,125 或 print *, 'z=', z

3.11 简单输入语句

* 输入语句的作用：将外部介质（键盘、磁盘）上的数据传送到内存变量指定的单元中。
* 输入/输出三要素：
- 对象：哪些数据
- 格式
- 设备
* 输入语句的分类：
- 格式输入（第七章）
- 表控格式输入（系统标准格式）ü

– 无格式输入

3.11.2 表控输入语句

* 自由格式输入

* 语句： read * ,〈输入表列〉

read (* , *)〈输入表列〉

注意：

1、执行输入语句时，程序会停止，等待用户从键盘输入数据。

2、输入多个数据时，数据间以逗号或空格隔开。

3、输入的数据应和输入表列中的变量个数、类型、次序严格一致。

例 1：read(*,*) a,b,I,j

输入：108.6，-37.8，5（回车）

---少一个数，则 j 没有被赋值，程序停止等待

输入：108.6，-37.8，5，6，9（回车）

---多一个数，则输入的 9 不起作用，程序正常执行

例 2： read(*,*) a,b,I,j,c, k,l,p

输入：25.8，-8.2 （回车）

5，8，2.7 （回车）

2，6，6.9 （回车）

---数据太多，可以分几个记录输入

记录：以回车结束的一批输入/输出数据

例 3：read(*,*) A,B,C,D,E,F,G

输入：7*3.5

---相同的数据可以用重复系数输入

例 4：read(*,*)A,B,C

read(*,*)D,I,J

输入：2.3，-63.5 （回车）

6.4，91.0 （回车）

5，8 （回车）

结果：A=2.3，B=-63.5，C=6.4，D=5.0，I=8

J 未被赋值

---每个 read 语句从一个新的记录开始读数

例 5：read(*,*)A,B,C

read(*,*)D,I,J

write(*,*)A,B,C,D

write(*,*)I,J

end

输入：2.3，63.5，6.4 （回车）

91.0，5，6 （回车）

输出：

-...--2.300000--...-63.500000-...-6.400000--.-91.000000

--...--5--...--6

---每个 write 语句也是从一个新的记录开始输出

3.12 参数语句（parameter 语句）

* 作用：将程序中经常用到的常数定义成一个符号常量，其值不可改变。

* 语句：parameter(p1=c1[,p2=c2,...,pn=cn])

其中： pn--符号常量； cn--常量

注意：

1、符号常量的命名规则与变量名相同,但它不同于变量,它的值不改变,在程序中不能对它赋值。

2、符号变量也有类型,也可用三种方法说明类型

3、参数语句是非执行语句,也位于所有可执行语句的前面,但位于类型说明语句后面。

4、一条语句可以定义多个符号常量。

5、优点：方便修改程序

3.13 END，STOP，PAUSE 语句

- ✓ END 语句：结束标志；有且仅有一条
- ✓ STOP 语句：停止运行程序；用于调试程序
- ✓ PAUSE 语句：暂停执行；用于调试程序

3.14 程序举例

* 顺序结构

【例 3.3】求三角形面积

公式：S= √S(S-A)(S-B)(S-C),S=（A+B+C）/2

源程序：
```
read(*,*) A,B,C
S=(A+B+C)/2
area=SQRT(S*(S-A)*(S-B)*(S-C))
write(*,*)'A=',A,'B=',B,'C=',C
write(*,*)'the area is',area
end
```

【例 3.4】求五边形面积

源程序：
```
write(*,*)'Input A,B,C,D,E,F,G'
read(*,*) A,B,C,D,E,F,G
S=(A+B+C)/2
S1=SQRT(S*(S-A)*(S-B)*(S-C))
 S=(C+D+E)/2
S2=SQRT(S*(S-C)*(S-D)*(S-E))
 S=(D+E+F)/2
S3=SQRT(S*(S-D)*(S-E)*(S-F))
AREA=S1+S2+S3
write(*,*)'area=',area
end
```

【例 3.5】编程将十进制数 407 转换成八进制数

源程序：
```
m=407
  i1=mod(m,8)
  i2=mod(m/8,8)
  i3=mod(m/8/8,8)
  write(*,*)i3,i2,i1   ；按高位到低位顺序
  end
```

第三章总结

顺序结构程序主要用于进行确定的显式计算

一般结构：由 5 部分组成

* 说明部分（说明语句，非执行语句）

* 赋初值部分（赋值语句，read 语句）

* 计算部分（数学公式→算术表达式）

* 输出部分（write，print 部分）
* 结束部分（END 语句）

第四章　逻辑运算和选择结构

4.1 引 言

* 分支结构：控制转向语句
* 分类：GOTO 类：无条件 GOTO 语句
　　计算 GOTO 语句

IF 类（条件类）：逻辑 IF 语句
　　块 IF 语句

4.2 关系表达式

* 条件：关系表达式
　　逻辑表达式
* 是构成选择结构判断条件的基本式子
* 算术表达式---由算术运算符将常量，变量，函数等基本成分连接在一起的式子
* 关系表达式---由关系运算符将算术表达式连接起来的式子

一、关系运算符

.gt.（大于）　　.ge.（大于等于）
.eq.（等于）　　.lt.（小于）
.le.（小于等于）　.ne.（不等于）

二、关系表达式的一般形式

〈算术量〉〈关系运算符〉〈算术量〉

例：x+y>15.4 ⟶ x+y.gt.15.4

注：

1、运算元素：算术量

2、运算结果:逻辑值:真（True）,假（False）

3、运算顺序：算术运算-->关系运算。　如：a+b.ne.a-b 等同于(a+b).ne.(a-b)

* 缺点：关系表达式只能表达简单的关系，如： 5≤x≤10 就不能用关系表达式表达，此时要用逻辑表达式。

4.3 逻辑表达式

一、逻辑运算符

.and.（逻辑与）
.or.　（逻辑或）
.not.（逻辑非）
.eqv.（逻辑等）
.neqv.（逻辑不等）

二、逻辑表达式的一般形式

〈逻辑量〉〈逻辑运算符〉〈逻辑量〉

⑴逻辑表达式是由逻辑运算符将两个逻辑量连接起来的式子。

⑵运算元素：逻辑量

结果：逻辑量:真(.True.)或假(.False.)

* 逻辑量包括:
- 逻辑常量
- 逻辑变量
- 关系表达式

三、逻辑常量

两个：①.true.(真) ②.false.(假)(书写)

四、逻辑变量

-用于存放逻辑常量的变量。

逻辑变量可以通过赋值语句来接受逻辑常量的值，但在使用前，要进行类型说明。

例： logical a, b

a=.true.

b=.false.

五、逻辑运算符的运算规则

若 a,b 为两个逻辑量，则：

a.and.b (当 a、b 同时为真时，为真)

a.or.b (当 a、b 中任意一个为真或同时为真时，为真)

.not.a (当 a 为真，其值为假；当 a 为假，其值真)

a.eqv.b (当 a、b 为同一逻辑常量时，为真)

a.neqv.b (当 a、b 不为同一逻辑常量时，为真)

六、逻辑表达式的运算次序

运算次序为： 算术运算→关系运算→逻辑运算

逻辑运算： .not.→.and.→.or.→.eqv.→.neqv.

4.4 用块 if 实现选择结构

* 有以下三种典型的块 if 选择结构：

⑴ 基本形式：

```
if (条件) then      （块 If 语句）
    块 1              (then 块)
else                  （else 语句）
    块 2              (else 块)
endif                 （endif 语句）
```

说明：

1、IF...THEN 语句-块 IF 结构的入口语句

2、ENDIF 语句---出口语句

3、必须一一对应，配对使用

4、THEN 块、ELSE 块是一条或多条可执行语句

5、执行过程：

条件为真（T）→THEN 块 → ENDIF 语句后的语句

条件为真（F）→ ELSE 块 → ENDIF 语句后的语句

程序举例

【例 4.1】学生考试成绩，高于 60 分为"及格"，否则为"不及格"

源程序：c

```
print score grade
read(*,*) score
if (score.ge.60.0) then
print *,'及格'
else
print *,'不及格'
end if
end
```

（2）简单结构：

```
if (条件) then      (块 if 语句)
    块                (then 块)
endif                 (endif 语句)
```

注：不包含 ELSE 块，可实现单边选择的功能

（3）嵌套结构：

```
if (条件 1) then
    块 1
else if (条件 2) then
    块 2                          (else if 块)
     ┆
else if (条件 n) then
    块 n
[else
   块(n+1)]
endif
```

说明：

⑴每个块 if 中可以完整地包含一个（或多个）块 if 结构，即构成块 if 的嵌套结构。如：

```
if (条件 1) then          if (条件 1) then
    块 1                      if (条件 2) then
else                          块 2
    if (条件 2) then       endif
      块 2                  else
    endif                     块 1
endif                     endif
```

⑵一个块 if 语句必须和一个 endif 语句相对应。

⑶块 if 中的"then 块"、"else 块"和"else if 块"可为空块。

4.5 逻辑 IF 语句

逻辑 IF 语句也是一种选择结构，但与块 if 不同，主要表现在：

①只用一行表示一个选择结构；

②仅当条件成立时执行，并且只执行一条语句。

相对而言，称为行 IF 语句。逻辑 if 语句的一般形式：

If〈条件〉语句

* 实现单边选择。例：if (n.le.100) n=n+1
* 块 IF 语句中，当 else 块是空块，then 块只有一条语句时，用逻辑 IF 语句更方便

第五章　循环结构的实现

所谓循环，是指在程序执行过程中需要重复执行的程序段。

循环结构包括：

(1) 循环体：由一些可执行语句组成

(2) 循环控制语句：控制循环的开始和结束

* 根据循环控制语句，将循环结构分两类

✓ 条件型循环

✓ 计数型循环--- DO 循环

5.1 用 goto 语句实现循环

goto 语句的一般形式：

goto <s1>　　其中：s1--语句标号。

功能：程序执行到此语句时，无条件的转向标号为 s1 的语句去执行。

5.2 用 do 语句实现循环

当循环的初值、终值和循环次数都已知时，可用 do 语句实现循环。

用 do 语句实现的循环称为"do 循环"。do 循环是由一个 do 语句和循环体组成。

一、一般形式

```
do s[,] v=e1, e2   [,e3]
    ┆
  s   <终端语句>
```

例：

```
do 10 I=1,19,2
L=I*I
write (*,*) 'L=',L
10 continue
write (*,*)'L=',L,'I=',I
end
```

注意：

1、标号为本程序单位中另一可执行语句的标号

2、步长可以省略，缺省值=1

例：do 10 n=1,15,1 → do 10 n=1,15

3、循环初值（e1），终值(e2)和步长(e3)都可以是常量，变量，表达式。

例：　do 30 k=3,16,A+1

　　do 100 n=k1,k2,k3

4、由于实数在内存中存储的误差，v, e1, e2, e3 尽可能用整型量。

5、e1,e2,e3 可正可负，e1,e2 可为 0，但 e3 不能为 0

执行过程（分四种情形）

⑴当 e2>e1 且 e3>0 ：

⑵当 e2>e1 且 e3<0 ：

⑶当 e2<e1 且 e3>0 ：

⑷当 e2<e1 且 e3<0 ：

二、具体执行过程

例：do 60 L=m,2*m-1,m/5　　（m=10)

```
    k=L+L
    write(*,*) k
60 continue
```

执行步骤：

①执行 do 语句，计算 e1,e2,e3 的值。

do 60 L=10,19,2

②将 e1 赋给循环变量 L，即执行赋值语句：L=10

③计算循环次数:r=INT((e2-e1+e3)/e3)

④检查循环次数：r=0?

r≠0-执行循环体，r=0---跳出循环体

⑤执行循环终端语句：v=v+e3

⑥r=r-1

⑦返回④继续执行

三、继续语句（continue 语句）

我们知道，循环终端语句必须是可执行语句。那么，这种作为循环终端语句的语句具有双重作用：一是作为循环终端的标志，二是要完成自身的功能。因此影响了程序的可读性。

Fortran 用一个专门的语句作为 do 循环的终端语句，即 continue 语句。

一般形式： s1 continue

四、一些规定：

1、循环变量在循环体内只能被引用，不能被赋值

2、在执行 DO 循环体期间，e1,e2,e3 的值不能改变，因为，它们决定了循环次数

3、离开 DO 循环时，循环变量可以在循环体外被引用，此时它的值为脱离循环时最后一次被赋的值。

4、程序中用到转移语句，规定：允许：从循环体内 → 体外；不允许：从体外 → 体内

5、循环终端语句必须是可执行语句（但除 goto, 块 if, endif, end 和 stop 语句外 ）

五、do 循环的嵌套

在一个 do 循环中还可以包含一套或多套完整的 do 循环，这就是 do 循环的嵌套。

一般形式:（以双重循环为例）

```
      do 10   i=1, 10
        ┆
        do 20   j=1, 10
          ┆
20      continue
        ┆
10    continue
```

例 6：打印"九九表"。

编程如下：

```
      do 10   i=1,9
        do 20   j=1,9
          k=i*j
          print*, i,'*',j, '=',k
20      continue
        print *,' '
10      continue
        end
```

注意：

1、嵌套要完整，不能交叉

2、循环变量的名字，规定：

并列的循环：循环变量可以同名

嵌套的循环：循环变量不能同名

3、若多层循环的结束语句在同一个地方，可以共用一条 continue 语句

4、控制转向语句的使用（体内→体外）

5.3 当型循环的实现

在无法确定循环次数的情况下。当型循环，是指执行循环体要依据事先给定的条件。当条件成立时执行循环，否则就不执行循环。

一、用 do while 语句实现当型循环

一般形式　　do　s1[,]　while (条件)

　　┆

　　s1　<终端语句>

结构类似 do 循环的结构

执行情况：条件：真→执行循环体；假→退出循环体

二、用块 if 和 goto 语句实现循环

一般形式：

```
s1  if (条件) then
      块
      goto s1
    endif
```

5.4 直到型循环的实现

所谓直到型循环，是指先执行循环体，再判断条件。如果条件为"假"，继续执行循环，直到条件为"真"时终止循环。

一、用逻辑 if 语句实现直到型循环

一般形式

```
s1  循环体
    if (条件) goto s1
```

5.5 几种循环形式的关系和比较

一、DO 循环适用于已知循环次数的情况

二、几种循环可以互换

DO 循环：条件型循环（可用次数作条件）

当型循环：直到型循环

当型：块 IF 语句(单边)+GOTO 语句(先判断后执行)

直到型：逻辑 IF 语句+GOTO 语句(先执行后判断)

三、各种循环可以相互嵌套。

第六章 FORTRAN 的数据结构

6.1 程序中的数据结构

1.程序代数表达式:

算法+数据结构=程序

2.对于同一个问题的求解，即一个程序的实现，可以采用不同的数据结构和不同的算法。选择合适的数据结构可以降低算法的复杂度。

* 在计算机高级语言中用数据类型来表示不同的数据结构。

数据结构一般有以下三类:

基本类型（Fortran 支持）
构造类型（Fortran 支持）
指针类型

Fortran 支持如下几种基本类型:

整型（第三章）
实型（第三章）
双精度型
复型
字符型
逻辑型（第四章）

Fortran 支持如下几种构造类型:

数组（第九章）
记录（第十三章）
文件（第十三章）

6.2 双精度数据类型

* 由于实型数据提供的有效数字的位数有限（微型计算机一般提供 7 位），一方面满足不了精度的需要，另一方面还会产生误差。

* 双精度类型以两倍于实型的字节（一般为 8 个字节）来存储数据，提供 15~17 位有效数字，解决了上述的问题。

* Fortran 中双精度常数要用指数表示：

如：12.3456789→ 1.2345656789D+1

* Fortran 中双精度变量使用前要用类型说明语句或 IMPLICIT 语句加以说明：

如：double precision A,B,C
implicit double precision(a-c)

6.3 复型类型数据

* Fortran 中复型常数要用一个括弧中的两个实数来表示，第一个实数表示复数中的实部，第二个实数表示复数中的虚部。

如：1+2.5i →（1.0，2.5）

* Fortran 中复型变量使用前要用类型说明语句或 IMPLICIT 语句加以说明：

如：complex A,B,C
implicit complex(a-c)

* 直接赋值

如： c=(3.0,4.0)
d=(8.76E+5,-6.8E-3)

* 当实部和虚部不是常数，而是表达式时，则应该用 CMPLX 函数将实部和虚部组成复型数据再赋给复型变量

如： C=cmplx(3.0*A,6.0+B)

* 如果 CMPLX 函数只有一个自变量，则它代表实部

如：cmplx(3.0) →(3.0,0.0)

* 在内存中一个复型数据占两个实数的存储单元，在 PC 中通常为 8 个字节

6.4 四种数值型数据之间的转换和运算

* 不同类型数据之间的运算的规则
* 不同类型数据的赋值规则
* 类型转换函数
* 不同类型数据间的比较规则

6.5 字符型数据

* Fortran77 不仅可以支持数值计算，而且也支持非数值处理，如文字处理。

一、字符型常量：又称字符串，是用撇号括起来的若干个字符

如：'CHINA'，'U.S.A'

* Fortran77 规定在程序语句中可以使用的字符如下：

（1）英文字母 26 个（不分大小写）

（2）数字 0-9，共 10 个字符

（3）专用字符，共 13 个： ﹀ ' $ ( ) + - * / , = . :

* 注意：

1.空格也是有效字符

2.Fortran77 规定字符和系统字符之间的区别

3.Fortran77 字符型常量允许使用系统可以使用的字符集即系统字符。如：'How are you?'，'abc@sina.com.cn' 都是合法字符串

4.当字符串本身有撇号时，用两个撇号表示。 如：'That''s right'

二、字符型变量：用来存放字符型常量的变量

* 字符型变量在使用前必须先定义。可以用 IMPLICIT 语句和 CHARACTER 语句，如：

```
Implicit character*5(A-C),character*4(x)
character*5 str1,str2,str3
character name*20,addr*30,code*10
```

* 如果在 CHARACTER 语句中不指定长度则隐含指定长度为 1
* 如果统一指定的字符长度与变量个别指定的长度不一致，则以个别指定的长度为准

如：character*5 A, B*6, C*7

则定义 A 为长度为 5 的字符型变量，B 的长度为 6，c 的长度为 7

三、字符变量的赋值

* 用赋值语句直接对字符型变量赋值
* 通过 READ 语句从键盘或其它输入设备读入字符常量给字符变量。用表控语句进行输入字符串时要用撇号将字符串括起来，如可以用 READ 语句代替上面的三个赋值语句

READ*，A，B，C

输入：

'CHINA'，'NANJING'，'NJUT'

四、子字符串：一个字符串连续的一部分

如：一个字符串 nanjing

则 na，nan，nanj，nanji，...都是其子串

* 子串表示形式

字符变量名（e1:e2）

e1 和 e2 是整型表达式，分别表示子字符串在原字符串中的起止位置,且 1≤e1≤e2≤L，L 是字

符变量的长度。

如：str='structured programming in fortran'

子字符串	子串的值	备注
str(12:22)	programming	
str(27:)	fortran	与 str(27:33)相同
str(:10)	structured	与 str(1:10)相同
str(5:5)	c	
str(:)	原字符串	与 str(1:33)相同

* 可以将一个子字符串赋给一个字符变量或另一个子字符串

如：name=str(27:33)

str(1:10)= str(27:33)

* 赋值号两侧的子串不能相互覆盖。 如：str(25:29)= str(27:33) 是不行的

五、字符表达式

* Fortran77 只提供一种字符运算符"//"，即为字符连接符，其功能是把前后两个字符型数据连接起来

如：'a'//'b' → 'ab'

六、字符关系表达式

* 格式： <字符(串)> <关系运算符> <字符(串)>

如：'A'.GT.'B' ， 'NANJING'.EQ.'NANJING'

* 比较规则： 按其字符 ASCII 码的值进行比较（P140）
* 两个单个字符比较，以其代码比较，代码大者为大。 如：'A'<'B'
* 两个字符常量（字符串）比较，将两个字符串中的字符自左向右进行比较，如有差别，则代码大者为大，否则两者相等。 如：'SHANGHAI'<'SHANGKONG'
* 如果两个字符串长度不等，则系统会自动将短的字符补以空格，使两者等长再比较。

如：'the'<'then' → 'the ␣'<'then'

七、用于字符处理的内部函数

1. LEN(A)
2. LGT(a,b)
3. LGE(a,b)
4. LLT(a,b)
5. LLE(a,b)
6. INDEX(a,b)
7. CHAR(I)
8. ICHAR(a)

第七章 数据的输入和输出

7.1 概 述

在程序设计中总是要有输入和输出。一般说来，在进行输入或输出时要确定三个基本要素：

①输入输出设备

②输入输出格式

③输入输出数据

注：系统隐含的输入输出设备为：键盘、显示器和打印机。

7.2 有格式的输出

一、有格式的输出语句

一般形式：

① write(*,s1) <输出表列>

s1 format(格式说明）

② print s1, <输出表列>

s1 format(格式说明）

其中："格式说明"是由各种"格式编辑符"构成的。

二、格式编辑符

主要介绍：I、F、E、X、H、纵向走纸、'（撇号）、r（重复系数）、/（斜杠）

1. I 编辑符。作用：用于整型数据的输出。 一般形式：

① Iw

② Iw.m

其中：I：整型输出;w-字段宽度；m：输出数据的最少数字位数。

2. F 编辑符。作用：用于实数的小数形式输出。 一般形式：

Fw.d

其中：F：实数的小数形式输出；w：字段宽度；d：输出数据的小数位数；

3. E 编辑符。作用：用于实数的指数形式输出。 一般形式：

Ew.d

其中：E：实数的指数形式输出； w：字段宽度； d：数字部分的小数位数。

4. X 编辑符。作用：用于输出空格。 一般形式：

nX

其中：X：输出空格；n：输出空格数。

5. H 编辑符。作用：用于输出字符常量。 一般形式：

nH

其中：H：输出字符常量；n：输出字符个数。

6. '（撇号编辑符)

作用：用于输出字符常量，即把撇号内的字符串原样照打。

注：如果输出的字符中包含撇号，则用两个连续的撇号代表一个要输出的撇号。

7. 纵向走纸编辑符。Fortran 规定：将输出记录中的第一个字符作为纵向走纸控制符，这个字符不再被打印出来，而从输出记录的第二个字符开始输出。记录中第一个打印字符与纵向走纸的关系（见右表）：

8. 重复系数 r。在 format 语句中，如果出现几个（或几组）相同的格式编辑符，则可以利用重复系数而只写一个（或一组）编辑符。

r--可重复使用的次数。

例 1： format(1x,2i10,f8.3,3f7.2)

9. 斜杠编辑符"/"。作用：将输出转入下一行记录。

7.3 有格式的输入

一、有格式的输入语句

一般形式：

```
        read(*,s1) <输入表列>
  s1    format(格式说明）
```

其中："格式说明"是由各种"格式编辑符"构成。

例：

```
        read(*,100)a,b,c
 100    Format(f5.1,e12.2,f7.2)
        End
```

键盘输入：

␣15.7␣2345.67e+04␣705.83↙

7.4 在 print,write 和 read 语句中包含格式说明

在格式输入输出时,也可以将 format 语句中的格式说明放到 print、write 和 read 语句中。

例：

```
        print 100,k,y
  100   format(i8,f7.2)
              ↓
        print '(i8,f7.2)',k,y
```

注意写法：'（...）'

第八章 常用算法的程序设计举例

8.1 数值积分

主要介绍求一个函数 f(x)在区间[a,b]上的定积分　　的程序设计方法。

从几何意义上看，它代表一个曲边梯形面积（见下图）。

我们知道，整个曲边梯形面积 S 可看作若干个小曲边梯形面积 si 之和。

即：

常用求解小曲边梯形面积的方法有：

①矩形法

②梯形法

③抛物线法（辛普生法）

一、梯形求解法

即应用小梯形面积代替小曲边梯形面积的的设计方法。

显然，第一个小梯形面积为：

S1=(f(a)+f(a+h))h/2

第 i 个小梯形面积为：

Si=(f(a+(i-1)h)+f(a+ih))h/2

则：

二、算法设计

画 N～S 图

三、举例

求

编程如下：

```
        read(*,*)a,b,n
        h=(b-a)/real(n)
        s=0.0
        f1=sin(a)
        do 10 i=1,n
            f2=sin(a+i*h)
            si=(f1+f2)*h/2.0
            s=s+si
            f1=f2
10      continue
        print 100,a,b,n,s
100     format (1x,'a=',f3.1,3x,
       $'b=',f3.1,3x,'n=',i4/1x,'s=',f7.3)
        end
```

8.2 解一元方程

常用求解一元方程 f(x)=0 根的方法有：

①迭代法

②牛顿迭代法

③二分法

④弦截法

一、牛顿迭代法（求 f(x)=0 在 x0 附近的实根）

基本原理和步骤为：

⑴选一个近似根 x1，有 f(x1)，见图(P179)。

⑵过点 f(x1)作切线，交于 x2。

因：f'(x1)= f(x1)/x1-x2

故：x2=x1- f(x1)/ f'(x1)

⑶再由 x2，有 f(x2)。

⑷过点 f(x2)作切线，交于 x3，得：

x3=x2- f(x2)/ f'(x2)

┆

再点 f(xn)作切线，交于 xn+1，得：

xn+1=xn- f(xn)/ f'(xn)

若：｜xn+1-xn ｜≤ε，

则： xn+1 为所求。

二、举例

用牛顿迭代法求方程

x3-2x2+4x+1=0

在 x=0 附近的实根。

（设 ε =10-6）

解例：

①画 N～S 图

②编程

8.2 解一元方程

```
          read(*,*)x
          n=1
10        x1=x
          f=x1**3-2.0*x1**2+4.0*x1+1.0
          f1=3.0*x1**2-4.0*x1+4.0
          x=x1-f/f1
          print 100,n,x1,x
100   format    (1x,'n=',i3,3x,'x1=',f15.7,3x,'x=',f15.7)
          n=n+1
          if (abs(x-x1).gt.1e-06) goto 10
         end
```

课后练习题

P192：第一、四题。

（要求编程。经上机调试后观察输出结果，然后将源程序作为附件发送。）

第九章　数 组

引例：求出全班 30 个学生的总分和平均分

循环结构：
```
      sum=0.0
      do 10 I=1,30
      read *,x
      sum=sum+x
10    continue
      aver=sum/30.0
      print *,'总分=',sum,'平均分=',aver
      end
```

注意：

1、执行时，输入数据的操作

2、X 这个变量表示一个存储单元，每次累加后就被刷新了

* 又例：要求求出每人的成绩与全班平均成绩的差值

源程序：
```
read *,x1,x2,...x30
ave=(x1+x2+...x30)/30.0
a1=x1-ave
a2=x2-ave
      ┆
write(*,*) a1,a2,...a30
end
```

在实际问题中，往往遇到要处理的数据是一组或一批同类型的量，而且它们是按一定次序进行排列的。如，

某班 30 名学生的单科成绩：

85,79,81,68,...　（分）

某年（12 个月）水文资料：

15.8,12.4,10.2,...　（m3/s）

（它们都是一组同类性质的量）

所谓数组是指一组有序同类数据的集合。对上述问题如不用数组，欲保留学生成绩则需要定义 30 个变量；保留水文资料则需要定义 12 个变量。若有 1 个班和 5 年的水文资料，就需要定义大量变量来保存这些数据，显然这会使程序笨长，也不便于阅读。使用数组可以解决这一问题。

如：对成绩，定义数组：x(30)

x(1),x(2),x(3),...,x(30)

用于存放 30 名学生的成绩。

* 对水文资料，定义数组：q(5,12)

q(1,1),x(1,2),x(1,3),...,x(1,12)

q(2,1),x(2,2),x(2,3),...,x(2,12)

┆

q(5,1),x(5,2),x(5,3),...,x(5,12)

用于存放 5 年的水文资料。这里的 x(i), q(i,j)称为数组元素(或下标变量)引例采用数组实现：

源程序：
```
dimension   x(30)
sum=0.0
read *,x
do 10 I=1,30
```

```
                sum=sum+x(I)
      10 continue
         ave=sum/30.0
```

概念：一组同类型的变量

一批连续的存储单元

表示：一个名字加不同的编号表示不同的单元

在 Fortran 中要使用数组同样必须遵循"先说明，后使用"的原则。

说明的目的是通知编译系统为数组分配相应的存贮单元。

用两种方法说明数组：

⑴用类型说明语句（"显式说明"）

⑵用 dimension 语句（"隐式说明"）

9.1 数组的说明和数组元素的引用

一、用类型语句说明数组

一般形式：

类型说明　数组说明符，...

其中,数组说明符的一般形式为：

数组名(维说明符, ...)

其中,维说明符由"下标下界:下标上界"组成

例：real xn(1:10),w(1:2,1:3),kw(10:20)

Integer bc(1:100),py(0:2,0:3,0:5)

二、用 dimension 语句说明数组

一般形式：

dimension　数组说明符，...

例：dimension x1(1:10),nw(1:2,1:3)

说明几点：

⑴ 在数组说明符中，维说明符（下标）的个数称为数组的维数。

⑵ 维说明符只能使用整型常量或整型符号常量表达式。

例：　parameter (i=1,j=10)

real kx(i:j+5)

⑶ 维说明符的下标下界为 1 时，可以省略。

⑷ 数组说明语句必须写在所有可执行语句之前。（属于非执行语句）

⑸ 数组的体积--每个下标值的乘积

三、数组元素的引用

一般形式：

数组名（下标，...）

即要有确定的数组名和下标值。

例：xn(5),w(1,3),kw(1,2,3)

* 引用数组元素时，下标可用算术表达式。

例：bc(i),py(i+j,k)

注意：数组元素与数组说明符

9.2 数组的逻辑结构和存储结构

数组的逻辑结构：数组所表示的实际数据结构。

数组的存储结构：数组在机内存储时的排列结构。

一、一维数组

此逻辑结构为一组依次排列的一串数据。　例：a(5)→ a(1),a(2),a(3),a(4),a(5)

此存储结构为一组连续存放的一列数据块。例：a(5)→ a(1) a(2) a(3) a(4) a(5)

二、二维数组

此逻辑结构为一张二维表数据。

例：a(3,4)可以看作一张 3 行 4 列的二维表数据。即：此存储结构为一组按列连续存放的数据块。

三、三维数组

此逻辑结构为若干张二维表数据。

例：a(2,2,3) ←2 行 2 列的表，共 3 页。此逻辑结构为：此存储结构为一组按页连续存放的数据块。

了解数组的逻辑结构和存储结构，对我们合理地选用数组和对数组进行输入输出有很大好处。

9.3 数组的输入和输出

有三种方式：

①用 do 循环

②用隐含 do 循环

③用数组名

一、用 do 循环进行数组的输入输出

用 do 循环进行数组的输入输出既有优点也有缺陷。

优：数组元素的输入输出次序可由用户控制。

缺：作一次循环就换行输入或输出。

二、用隐含 do 循环进行数组的输入输出

用隐 do 循环进行数组的输入输出，使我们既能控制数组元素的输入输出次序，又能控制一行内输入输出数据的个数。

仍以 2 名学生 3 门功课的成绩为例，编程如下：

```
integer g(2,3)
read(*,*) ((g(i,j),j=1,3),i=1,2)
write(*,*) ((g(i,j),j=1,3),i=1,2)
end
```

由于是一个 read 语句，所以既可以一行输入，也可以分多行输入。键盘输入如下：

```
86,75,72↙
87,70,83↙
```

注：一个 read 语句可以分多行输入，而有多个 read 语句时，每一个 read 语句必须从新的一行读数。

将上例问题编成一个较完整的程序如下：

```
      parameter (m=30,n=5)
      dimension kd(m,n)
      character num(m)*5
      read(*,*)(num(i),(kd(i,j),j=1,n),i=1,m)
      print *,'序号  学号  语文  英语  数学  物理  化学'
      print 100,(i,num(i),(kd(i,j),j=1,n),i=1,m)
100   format(1x,i3,3x,a5,2x,5i4)
      end
```

三、用数组名进行数组的输入输出

在用数组名进行数组的输入输出时，其顺序要与数组元素在机内的存储顺序一致。

例 1：dimension k(5)

```
      read *,k     ←对数组的整体进行操作
```

等价于： ↓

```
      read *,k(1),k(2),k(3),k(4),k(5)
```

也等价于： ↓

```
      read *,(k(i),i=1,5)
```

例 2： integer g(2,3)

read *,((g(i,j),j=1,3),i=1,2)

print *,g

等价于： ↓

print *,g(1,1) g(2,1) g(1,2) g(2,2)

g(1,3) g(2,3)

也等价于： ↓

print *,((g(i,j),i=1,2),j=1,3)

9.4 使用 data 语句给数组赋初值

在 Fortran 程序中，可用 data 语句给变量或数组赋初值。

一般形式：

data 变量表/初值表/,变量表/初值表,...

功能：在程序编译期间给变量或数组赋初值。其中，变量表可以是变量名、数组名、数组元素、隐 do 循环；初值表只能是常量，不允许出现任何形式的表达式。

例： data a,b/7.85,9.1/,i,j,k/5,10,15/

例： dimension k(2,3)

data((k(i,j),j=1,3),i=1,2)/90,23,20,42,14,32/

或：data k/90,42,23,14,20,32/

例： dimension a(10)

data a/10*-1.0/ (表示"10 个-1.0")

注：data 语句属说明语句，但它可以放在 end 语句之前的任意行；当程序中有多个 data 语句给同一个变量赋初值时，以最后一条 data 语句为准；程序在编译期间给变量赋定初值，在程序执行期间，data 语句不起任何作用。

9.5 数组应用举例

一、一维数组程序举例：(排序问题）

例 1：设计一个程序，对 N 个整数按升序排列。(用交换法)

①算法：先将第一个元素与后面的 N－1 个元素比较，若符合递增顺序则不变，否则两者交换。交换后的第一个元素继续与后面未比较过的元素比较，这样在第一轮扫描结束后，第一个元素就是最小的元素。第二轮扫描再对第二个元素重复以上过程。(N 个数需要 N－1 轮扫描)

②画 N～S 图：

③编程：

```
integer x(1000),temp
read *,n
read *,(x(i),i=1,n)
do 50 i=1,n-1
   k=i+1
   do 50 j=k,n
      if(x(i).le.x(j)) then
         goto 50
      else
         temp=x(i)
         x(i)=x(j)
         x(j)=temp
      endif
50  continue
print *,(x(i),i=1,n)
end
```

二、二维数组的程序举例

1、输出格式问题（输出语句）

例 2：打印乘法九九表（输出下三角阵）[杨辉三角]

```
      integer M(9,9)
        do 10 i=1,9
          do 10 j=1,9
            m(i,j)=i*j
10      continue
        do 20 i=1,9
        write(*,100) (i,'*',j,'=',m(i,j),j=1,i)
100     format(1x,9(1x,i1,a1,i1,a1,i2))
20      continue
        end
```

2、表格数据处理问题（计算，输入，输出）

例 3：10 个学生，考 3 门课的成绩，要求求出每人的平均成绩，并输出一张成绩表

分析：二维数组 G（10，3）存放学生成绩

一维数组 AVG（10）存放平均成绩

程序包含：①构造二维数组（输入）

②计算平均成绩

③输出数组（一维，二维）

源程序：

```
dimension g(10,3),avg(10)
        read *,((g(I,j),j=1,3),I=1,10)
        do 30 I=1,10
            sum=0.0
            do 20 j=1,3
                sum=sum+g(I,j)
20          continue
            avg(I)=sum/3.0
30      continue
        write (*,100)
100     format(1x,'   I',6x,'g1',6x,'g2',6x,'g3',5x,'avg')
            write(*,200)(I,(g(I,j),j=1,3),avg(I),I=1,10)
200     format(1x,i3,4f8.1)
            end
```

第十章　语句函数

在第三章§3.7 中我们介绍了 FORTRAN 的内部函数.FORTRAN 编译程序把一些常用的运算(如求平方根,绝对值,三角函数.....)编程为一个一个内部子程序，当我们程序中出现这些内部函数时，系统就会自动调用相应的子程序以完成所需的运算。但我们实际需要用到的函数是远远不止这些基本的内部函数，这个时候，我们就有必要自己定义一些我们需要的函数-语句函数或者子程序（第十一章）。

10.1 语句函数的定义

一、语句函数的定义形式

定义形式如下：f(x1,x2,...xn)=e

其中

f：语句函数名，其函数值也具有类型，所以在使用之前，必须对函数名象变量的类型说明一样对其进行类型说明；

x1,x2...xn 代表语句函数的自变量，称为虚拟参数。它们本身是没有值的，只有在函数被引用时用实在变量(实参)代替虚拟参数时，函数才能得到函数值；当然一个语句函数可以没有虚拟参数，但函数名后面的一对括号不能少；虚拟参数名可以和变量名相同，但它们之间除了名字相同外，没有任何其他联系。

e：语句函数的表达式；e 中除了包含有关虚拟参数外，还可以包含常量、变量、数组元素引用、外部和内部函数和已定义过的语句函数。

二、定义语句函数的一些规则

1.语句函数名不能与本程序单位中的任何其他对象(变量，数组...)名字相同

2.当函数可以用一条语句定义时才可以使用

3.语句函数定义语句是非执行语句.它应放在所有可执行语句的前面和所有的说明语句之后

4.语句函数只在其所在的程序单位中才有意义

5.语句函数中的虚拟参数应是变量形式,不可以是常量、表达式和数组元素

10.2 语句函数的引用

* 语句函数一旦被定义后，就可以在同一个程序单位中引用它。引用的形式和引用内部函数一样，即用实参代替虚参。实参可以是与虚参类型一致的常量、变量、或表达式，实参必须有确定的值。

例如：　f(k)=k+1

```
y=f(f(3))          //y 值为 4.0
```

第十一章 子程序

在实际工作中，我们经常要编制一些求解特定问题的程序，如：排序、求积分、解线性方程组、插值等。如果每使用一次就编写一次，程序就会显得冗长、累赘。因此常把它们编制成独立的程序单位，需要时可随时调用。即：将那些比较复杂，但又需要经常用到的操作编写成具有独立名字的子程序。由主程序随时调用。

子程序：完成某一特定功能的程序单位。

Fortran 子程序包括：函数子程序，子例行程序，数据块子程序（第十二章）

执行时：从主程序开始执行，遇到调用语句再执行相应的子程序。

不同类型的子程序，关键字不同，调用方法也不同。

11.1 函数子程序

从使用的角度讲，"函数子程序"就是一种可以作为函数来调用的子程序（也称"外部函数"）。

用法类似内部函数，如 sin(x)。

一、函数子程序的定义

一般形式：

```
类型说明  function  函数名([虚参表])
    ┆                                    (1)
end
function  函数名([虚参表])
    类型说明  函数名                (2)
        ┆
end
```

其内容由两部分组成：FUNCTION 语句

子程序体

1、FUNCTION 语句：是函数子程序的第一条语句，标志着该函数子程序的开始。

类型说明 function 函数名([虚参表])

注意：

⑴ 虚元也有类型，可在子程序体中说明

例： real function intep(x1,x2,x3)

integer x1,x2,x3

⑵ 函数名的命名规则和类型都与变量相同

⑶ 虚参可以是变量名、数组名和子程序名，但不允许用常量和数组元素。例：function fat(n,x(i))

它表示了函数自变量的个数，顺序和类型

2、子程序体：用来完成具体任务的一个程序段，结构：

说明语句

可执行语句

END 语句：独立的程序单位，单独编译

注意：

⑴若无虚参时，括号不能省。例：function proc( )

⑵函数子程序中所有变量和标号（除函数名和虚参外），与其它程序单位无任何联系。

⑶函数体的说明部分包括对虚参和本函数体内所用变量和数组的说明。

例：

```
function rdh(a,x)
integer a,b,c
dimension x(2,3)
   ┆
```

```
        end
```

⑷函数体中可设置一条或多条 return 语句，表示执行到此语句时返回调用程序。当然也可以不设 return 语句。

```
例：    function fat(n)
          ┆
        [return]
          ┆
        end
```

两种情况：

①当 return 语句和 end 语句紧挨着时，可省略 return 语句。

②需从中间返回时，return 语句必不可少

⑸函数名的作用

函数名在函数体中，至少要有一次出现在赋值语句的左边，即：函数名=表达式。（因为函数名要负责把函数值带回调用程序）

例：求一个一维数组 A(n)的各数组元素之和。

```
      Function   sum(A,n)
      Dimension A(50)
      Sum=0.0
       Do 10 i=1,n
       sum=sum+A(i)
10    Continue
        return
        end
```

这里，sum 是函数子程序的名字，它应作为一个变量名出现在程序体中，并且至少被赋值一次。这样，它才有值，在被调用后将值带回到主程序中。

二、函数子程序的调用

调用方式与内部函数相似。

一般形式：

① 函数名（实参表）

② 函数名（）

* 类似：sin(x),上例可用：fac(9)

注：在子程序中，如果函数名的类型未使用隐含规则，在主程序中则先要对子程序函数名进行显式说明，以保证函数名类型一致。

例：求 1!+3!+5!+...+9!（调用子程序）

```
C     main program
      sum=0.0
      do 10 k=1,9,2
        sum=sum+fac(k)
10    continue
      print*,sum
      end
```

函数子程序调用过程为：

```
c program main                  c subprogram
        ┆                         functiom np(x,n)
      km=np(a,m)                    ┆
        ┆                           ┆
```

end　　end

执行过程：

⑴虚实结合（k→n)

⑵执行函数子程序体，其间得到函数 fac(n)

⑶遇到返回语句，将函数值带回到主程序中 fac(k)

几点说明：

①调用程序中函数名必须与函数子程序中定义的函数名相同。

②实参与虚参在个数、类型、位置上必须一一对应，但名字可以不同。

如：主程序　　子程序

kh=erg(a,m,x)　　function erg(x,n,q)

③当虚参是变量名时，实参可以是常量、变量、数组元素或表达式；但当虚参要在函数体中被赋值时，则实参不能为常量或表达式。（因虚参和实参共用一个存储单元）

④函数子程序是一个独立的程序单位，可以作为一个单独的源文件进行存贮和编译，并与调用程序连编后才能起作用。（也可以与调用程序放在同一文件中。）

程序举例

例：分别求半径 r 为 1、3、5 的圆周长（要求把计算圆周长的部分用函数子程序编写）。

```
  c main program                    c subprogram
    integer r                         function s(a)
    do 10 r=1,5,2                     integer a
       c=s(r)                         pi=3.14159
       print 100,r,c                  s=2*pi*a
10   continue                         return
100 format(1x,'r=',i1,' c=',f6.3)     end
     end
```

11.2 子例行程序

子例行程序也是 Fortran77 的子程序，它与函数子程序的主要差别在于：

⑴ 名字的作用不同：函数子程序名除了供调用外，还代表函数值，而子例行程序名只供调用。

⑵ 要求返回值的方式不同：函数子程序是通过函数名将函数值带回调用程序，而子例行程序是通过"虚实结合"将其新值转送回调用程序。

⑶ 调用方式不同：函数子程序的调用总是出现在表达式中，而子例行程序必须用一条独立的语句调用。

一、子例行程序的定义

一般形式：

```
subroutine  子例行程序名(虚参表)
      ┆
end
```

几点说明：

⑴子例行程序名的命名规则与变量名相同。（但不标识类型）

⑵如果没有虚参，括号可以省略。 例：subroutine　abc

⑶虚参可以是变量名、数组名和子程序名，但不允许用常量和数组元素。

⑷它的说明部分包括对虚参和本子例行程序所用变量和数组的说明。

⑸子例行程序的名字只起标识作用，没有值的概念，仅仅为了调用。因此，在子例行程序体中，它不允许出现。

二、子例行子程序的调用

一般形式：

call　子例行程序名（实参表）

call　子例行程序名

* 执行过程：类似函数子程序

11.3 函数和子例行子程序的区别

函数和子例行程序的不同点：

1、 定义语句不同

2、 名字的作用不同：函数名除了供调用外，还代表函数返回值，而子例行程序名只供调用

3、 调用方法不同：函数的调用总是出现在表达式中，而子例程序必须用一条独立的语句调用

4、 返回值的方式不同：自定义函数是通过函数名将函数值带回主调程序，而子例程序是通过“虚实结合”将其新值转送回主调程序

5、 用途不同：函数只能带回一个值，而子例行子程序可以带回一批值（数组），或者完成特定的操作。（如：交换，排序等）

11.4 实参和虚参之间的数据传递

在 Fortran 程序设计中子程序的虚参可以是变量名、数组名、函数名、子例行程序名和星号'*'。

下面分别讨论当虚参为变量名、数组名、子程序名以及星号‘*’时的实参和虚参之间的数据传递问题，也称"虚实结合"问题。

一、变量名作虚参时的虚实结合

对应的实参可为同类型的常量、变量或数组元素。

1. 结合方式：是按地址结合的。即虚参和实参共用一个存储单元。

2. 结合过程

注意：当实参是数组元素时，虚实结合的过程和变量名相同。

二、数组名作虚参时的虚实结合（仅说明数值型数组）

此时，对应的实参必须为同类型的数组名或数组元素。

1. 实参为数组名时的虚实结合

是按地址结合的。即实参数组的第一个元素与对应虚参数组的第一个元素结合；

实参数组的第二个元素与对应虚参数组的第二个元素结合，...，依此类推。

几种情况：

①同维同体积（不同体积）：

例：
```
program main                subroutine ctx(b)
dimension a(8)              dimension b(5)
call ctx(a)                   ┆
 ┆                          end
end
```

若数组 a,b 均为二维数组，如 a(2,3)，b(2,3)则：

②不同维不同体积：若主程序中 a(2,3),子程序中 b(5)

2. 实参为数组元素时的虚实结合

仍按地址结合，但该数组元素与对应虚参数组的第一个元素结合；该数组元素的下一个元素与对应虚参数组的第二个元素结合，...，依此类推。

注：虚参数组的最后一个元素必须落在实参数组范围内。对上例，若定义 y(6)，则会出现"超界错"。

三、虚参是可调数组

从数组的虚实结合规则可知，如果虚参数组的下标不可调，那么就会极大地影响子程序的通用性和灵活性。设计可调数组可以较好地解决这一问题。

可调数组：只能在子程序中使用，其上、下界可以是整型虚参变量，其值可以通过对应的实参传递。如果只想求一部分元素之积，可以再多设一个虚参变量，增加灵活性。

注意：

1、可调数组只能出现在子程序中，不允许出现在主程序里。

2、可调数组的数组名和界都必须作为虚参出现在虚参表中。这样，通过虚实结合后，它们就被赋值。
例：

```
program main                    subroutine sub(x,nx,y,n1,n2)
parameter(m1=4,m2=3)            dimension x(nx),y(n1,n2)
dimension a(10),b(m1,m2)          ┆
call sub(a,10,b,m1,m2)          end
 ┆
end
```

1.在调用程序单位中，如果实在参数中出现函数子程序名或子例行程序名时，必须在调用程序单位的说明部分用 EXTERNAL 语句说明这些名字。
2.如果在实在参数表中出现内部函数名时，必须在调用程序的说明部分用 INTRINSIC 语句说明这些名字。
注意：凡在 INTRINSIC 语句中说明的名字必须是 FORTRAN 中合法的内部函数名，在 EXTERNAL 语句中说明的名字必须代表完整的程序中确实存在的子程序名。

程序举例

编写一个函数子程序，通过函数名的传送，使之既能求正切 TAN 值（X）又能求余切值 CTN（X）
取函数名:　TRIANGLE（ F1，F2，X ）
函数表达式: TRIANGLE=F1(X)/F2(X)
F1，F2 是虚拟函数名，X 是自变量（弧度）
当把 SIN 传给 F1,把 COS 传给 F2 时,求正切
当把 COS 传给 F1,把 SIN 传给 F2 时,求余切

```
      c      main    program
      intrinsic sin,cos
      x=3.14159/5.0
      y1=triangle(sin,cos,x)
      y2=triangle(cos,sin,x)
      write(*,100)'tan(',x*180/314159,')=',y1
      write(*,100)'ctn(',x*180/314159,')=',y2
100   format(1x,a,f6.2,a,e13.6)
      end
      Function triangle(F1,F2,X)
        triangle=F1(X)/F2(X)
      END
```

五、虚参是星号‘*’

当虚参表中出现一个*号时，对应的实参应该是一个冠有*号的语句标号。例如：

程序举例

例：编程实现一个一维数组的反转处理。
设一维数组的体积 n≤600 个元素。
分析：头尾倒转：n=600 时，1-600，2-599
第 i-n+1-i

```
Subroutine change(a,b)
      t=a
      a=b
      b=t
end
Subroutine run(x,n)
```

```
      Dimension x(600)
      Read *,n
      If(n.gt.600) then
      Print *,'n is too large'
      Return
      Endif
      Read *,(a(I),I=1,n)
      Do 15 I=1,n/2
        call change(x(I),x(n+1-I))
15 continue
       end
```

例：编写一个函数子程序计算二维数组 x(5,4)中任意行或任意列的元素之和。
然后调用它来计算 a(5,4),b(5,4)数组任一行或任一列元素之和。
分析：①任一行或任一列用变量 L 表示
②设置一个逻辑变量 LOC，来控制行或列：LOC=.true.----第 L 行
LOC=.false.---第 L 列

```
  Function sum(x,L,LOC)
      dimension x(5,4)
      logical   LOC
      sum=0.0
      if(LOC)then
      do 20 i=1,4
            sum=sum+x(L,i)
20   continue
      else
      do 30 i=1,5
            sum=sum+x(i,L)
30   continue
      endif
      end
```

第十二章　数据共用存储单元

数据块子程序

12.1 等价语句 (EQUIVALENCE 语句）

例 1：　EQUIVALENCE （W，ST）

例 2：　EQUIVALENCE （W，ST）,(x,y,z)

二、等价语句的使用规则：

1.等价语句是说明语句，它必须出现在程序单位的执行语句之前。

2.等价语句每对括号中的变量可以具有不同类型（但建议大家不这样使用，因为其中的变量容易失去定义），但是 FORTRAN77 规定字符变量必须与字符变量等价。

3.数组的等价。如：

dimension a(8),b(2,4),c(2,2,2)

equivalence (a,b,c)

相当于：

equivalence (a(1),b(1,1),c(1,1,1))

a,b,c 三数组在内存中的存储分配如下：

如果有以下语句：

dimension a(2,3),b(4)

equivalence (a(1,2),b(1))

a,b 两数组在内存中的存储分配如下：

4.不能利用等价关系建立矛盾的等价关系

如：　dimension A(10)

equivalence （x,A(1)）,（x,A(3)）

此处 A(1)和 A(3)是同一个数组中的元素， A(1)和 A(3)不可能分配同一个存储单元，所以 X 既要和 A(1)分配的单元相同，又要和 A(3)分配的单元相同，明显不可能。

12.2 公用语句

* 通过第十一章的学习，我们知道程序中不同程序单位之间的数据交换是通过虚实结合来完成的。另外 FORTRAN 还可以通过建立公用区的方法来实现各程序单位之间的数据交换。

* 公用区和虚实结合比较起来有其缺点--程序阅读性下降；有其优点--运行速度快（特别当有大量数据需要传送时）。

12.2.1 无名公用区

* 在不同的程序单位中，各自的变量是相互独立的，尽管它们的变量名有时相同。但如果我们在主程序和子程序中都加上一条 COMMON 语句的话，如：

COMMON　X

此时编译程序在存储区中开辟了一个公用数据区。这时主程序和子程序中的两个变量 X 就共用一个存储单元，而不是相互独立了。

* 在 FORTRAN 中，用 COMMON 语句来开辟无名和有名公用区

一、开辟无名公用区的 COMMON 语句的一般格式：

COMMON　a1，a2，...

其中 a1、a2、...允许是普通变量名、数组名和数组说明符（不是数组元素）

例如：

在主程序中写：COMMON　X，Y，K，Z（3）

在子程序中写：COMMON　A，B，J，T（3）

此时，在无名公用区中变量 X 和 A，Y 和 B，K 和 J 分别被分配在同一个存储单元中；数组 Z

和 T 同占 3 个存储单元。

解释：FORTRAN 编译程序在编译时为以上的 COMMON 语句开辟一个无名公用区，当把不同的程序单位连编在一起的时候，不同的程序单位在 COMMON 语句中的变量按其在语句中出现的先后顺序，依次连续的占用无名公用区的存储单元，也就是说每个程序单位中的 COMMON 语句总是把出现在语句中的第一个变量分配在无名公用区的第一个存储单元位置。

程序举例：解一元二次方程的根

[例 12.1]完全通过虚实结合交换数据

```
主程序：    READ*,A1,A2,A3
            CALL QUAD(A1,A2,A3,Z1,Z2)
            WRITE(*,*)Z1,Z2
            END
子程序：SUBROUTINE QUAD(A,B,C,X1,X2)
            P=-B/(2.0*A)
            Q=SQRT(B*B- 4.0*A*C)/(2.0*A)
            X1=P+Q
            X2=P- Q
            END
```

程序举例：解一元二次方程的根

[例 12.2]通过虚实结合和公用区两种方式交换数据

```
主程序：    COMMON Z1，Z2
            READ*,A1,A2,A3
            CALL QUAD(A1,A2,A3)
            WRITE(*,*)Z1,Z2
            END
子程序：SUBROUTINE QUAD(A,B,C)
            COMMON X1，X2
            P=-B/(2.0*A)
            Q=SQRT(B*B- 4.0*A*C)/(2.0*A)
            X1=P+Q
            X2=P- Q
            END
```

程序举例：解一元二次方程的根

[例 12.3]完全通过公共区交换数据

```
主程序：      COMMON   Z1,Z2,A1,A2,A3
              READ*,A1,A2,A3
              CALL QUAD
              WRITE(*,*)Z1,Z2
              END
子程序：SUBROUTINE QUAD()
              COMMON X1,X2,A,B,C
              P=-B/(2.0*A)
              Q=SQRT(B*B- 4.0*A*C)/(2.0*A)
              X1=P+Q
              X2=P- Q
              END
```

二、无名公用区的 COMMON 语句使用规则

1.COMMON 语句是说明语句，必须出现在所有可执行语句之前

2. COMMON 语句中只允许出现变量名、数组名或数组说明符，后者意味着可用 COMMON 语句说明数组。例： COMMON A，B，NP(15)，LOC(2，4)

等价于：DIMENSION NP(15)，LOC(2，4)

COMMON A，B，NP，LOC

3.用于 COMMON 语句中的变量在编译时已经被分配在实在的存储单元中，因此在 COMMON 语句中不能出现虚拟参数。同样，在编译时，可调数组的大小不定，无法分配固定的存储单元，因此可调数组名也不能出现在 COMMON 语句中。但是可调数组的维的上、下界变量可以通过 COMMON 语句传送。 例：SUBROUTINE SUB(A,B)

COMMON NA, NB

DIMENSION A(NA),B(NB)

4.一个程序在运行过程中只有一个无名公用区。在同一个程序单位中可以出现几个 COMMON 语句，它们的作用相当于一个。

例： COMMON A,B,C,D

COMMON A1,B1,C1,D1

相当于： COMMON A,B,C,D ,A1,B1,C1,D1

5.各程序单位 COMMON 语句中的变量类型必须按位置一一对应一致才能正确传送数据

例： 主程序： COMMON A(5), I

子程序： COMMON B(4),J,P

主程序中 A 数组元素是实型，而数组元素 A(5)和子程序中的变量 J 共用存储单元，还有主程序中的 I 和子程序中的 P 共用存储单元都是错误的

6.在一个程序单位中，分配在公用区中的名字只能在 COMMON 语句中出现一次

例： COMMON A，B，C

COMMON A1，B1，C

错误：因为 C 在 COMMON 语句中出现了两次

7.在不同的程序单位中，无名公用区中的变量个数可以不一样。但必须保证对应的变量类型要一致

8.如果公用区中出现字符型数据或字符数组，则要求整个公用区中的变量为字符类型，不允许字符变量与其他类型变量放在同一个公用区中

12.2.2 有名公用区

我们已经知道，无名公用区中各程序单位之间数据传送按公用区中变量名的排列顺序一一对应进行。但我们在实际使用中常常会遇到下面的问题。

例：我们的主程序要和两个子程序进行数据传送。

主程序： COMMON A, B，C，J，K，L

子程序 1 需要与主程序传送 A,B,C 三个数据；

子程序 2 需要与主程序传送 J,K,L 三个数据；

根据一一对应的原则。解决方法一：

主程序： COMMON A, B，C，J，K，L

子程序 1：COMMON A1,B1,C1

子程序 2：COMMON A2,B2,C2,J2,K2,L2

此时子程序 2 中的 A2,B2,C2，从语法角度是必需的,但它们从使用角度却是冗余的。当一个主程序和若干个子程序间进行大量的数据传送时，书写 COMMON 语句就显得很烦琐，而且也容易出错。

解决方法二：利用有名公用区

主程序： COMMON A, B，C /C2/ J,K,L

子程序 1：COMMON A1,B1,C1

子程序 2：COMMON /C2/ J2,K2,L2

此时 A,B,C 仍放在无名公用区中；而 J,K,L 放在名为 C2 的公用区中。

利用有名公用区就避免了无名公用区的弊病,使之做到公用之中有"专用",我们只需在各个程序单位中做到同名共用区中数据顺序一一对应就行了

一、开辟有名公用区的 COMMON 语句的一般格式：

COMMON /n1/变量表 1，/n2/变量表 2，...

其中，n1,n2 为公用区名，它们放在两个斜杠"/"之间，它们的取名方法与变量名同；变量表代表各有名公用区中的变量名、数组名或数组说明符。我们也可以用两个连续的斜杠来表示无名公用区，

例如： COMMON X，Y，Z/C2/A，B，C

可以写成：COMMON // X，Y，Z/C2/A，B，C

或者：COMMON /C2/A，B，C // X，Y，Z

COMMON 语句中的公用区的变量可以“化整为零”，只要它们在有名公用区中的顺序不变。

例如： COMMON X，Y，Z/C2/A，B，C

可以写成： COMMON /C2/A //X，Y，Z/C2/B，C

也可以写成下面两句： COMMON /C2/A

COMMON X，Y，Z/C2/B，C

建议：一个有名公用区用一条 COMMON 语句来完成

例如： COMMON X，Y，Z

COMMON /C1/A，B，C

COMMON /C2/J，K，L

COMMON /C3/R

二、有名公用区和无名公用区规则的区别

1.各程序单位的同名公用区的大小必须相同，而无名公用区则无此要求

2.有名公用区的成员可以通过下节§12.3 所介绍的数据块子程序赋初值，而对无名公用区的成员则无任何方法赋初值

3.在执行 RETURN 语句或 END 语句时，有时会导致有名公用区中的成员变为无定义，但无名公用区中的成员不存在这个问题。

12.3 数据块子程序

功能：是一种特殊的子程序，它只是用来给有名公用区中的变量赋初值。

数据块子程序格式和规则说明如下：

1.数据块子程序必须以 BLOCK DATA 作为第一个语句,以 END 作为最后一条语句。说明形式如下

```
BLOCK DATA 子程序名
      ⋮
END
```

可以没有子程序名,这时的数据块子程序称为无名数据块子程序；一个程序中只能有一个无名数据块子程序,可以有任意多个有名数据块子程序。

2.数据块子程序只是用来给有名公用区的变量赋值，不能被被别的程序单位调用。

3.在数据块子程序中不允许出现可执行语句，只允许出现 COMMON、 DIMMENSION、 DATA、 EQUIVALENCE 和类型说明语句。其中 DATA 和 COMMON 语句是必不可少的。（数据块子程序由 DATA 语句给指定变量赋初值，并由 COMMON 语句与其他相连接的程序单位沟通）。

4.数据块子程序中的 DATA 语句只能给有名公用区中的变量赋初值，不允许给无名共用区中的变量赋初值。

5.指定的某个有名公用区中所有变量（即使其中有些变量不要求在 DATA 语句中赋初值），都必须一一按顺序列在 COMMON 语句中。

例如：

```
BLOCK DATA INIT
DIMMENSION A(10),B(5)
COMMON /COM/A,X,Y,Z,B,I
INTEGER X,Y,Z
```

```
      DATA    X,Y,Z/3*0/,B/5*0.0/
      END
```

这是一个完整的的数据块子程序。虽然 DATA 语句只需给 COM 共用区中 X，Y，Z 变量和 B 数组赋初值，但仍需列出 COM 中所有的变量。

6.一个 FORTRAN 程序可以包含任意多个数据块子程序，但每个有名公用区中的变量只能在一个数据块子程序中赋一次初值，不允许把一个有名公用区中的变量分在几个数据块子程序中赋值。

第十三章 文 件

13.1 概述

FORTRAN 程序结构：

```
输入数据（赋值，read） ！从键盘输入
运算
输出结果（write，print）！输出到屏幕
end
```

例：求 5 个学生的平均成绩

```
read *, g1, g2, g3, g4, g5
avg=(g1+g2+g3+g4+g5)/5.0
print *, avg
end
```

输入：

① 赋值语句：不易修改，需重新编译；

② read 语句：输入量很大时，麻烦，且易出错。

输出：只能看到屏幕或打印纸上的结果，要看结果，必须运行程序。

解决办法：使用文件

输入：将要输入的数据存放在文件中（数据文件），然后在程序中用 read 语句读入，可反复使用。

输出：将结果保存在文件中，方便。

13.2 文件的基本概念

一、文件的定义

FORTRAN 中，文件指存储在外部介质上的数据集合，由文件名代表。

文件名：用字符串表示，与变量命名规则类似。作为文件的标识，可在程序中反复引用

无名文件：临时性使用

二、文件的组成

文件是由记录组成，FORTRAN 对文件的存取以记录为单位，每个记录又是由字段（数据项）组成。

三、文件的分类

按存取方式（读取记录的顺序）：

⑴ 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。

⑵ 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：

⑴ 有格式存放（也称字符形式存放）：文本文件。

⑵ 无格式存放（也称二进制形式存放）：二进制文件，直接保存数据的二进制形式。

13.3 文件使用

文件操作的一般步骤：打开文件，读取操作，关闭文件。一般形式：

```
open(说明项)
  ┆
read /write(控制项), 输入/出表列
  ┆
close(设备号)
```

例 1：建立一个存放 5 个学生的平均成绩的有格式顺序数据文件。

```
      open(5, file='score.dat', status='old')
      read (5,*)  x1, x2, x3, x4, x5
      aver=(x1+x2+x3+x4+x5)/5.0
      write(*,100)  aver
 100  format(1x,'aver=', f4.1)
      close(5)
      end
```

1、OPEN 语句

OPEN(说明项)

作用：打开一个文件，实现一个设备号和一个文件的连接。若文件不存在，则新建一个文件。

说明项：

① 设备号：[UNIT=]u （1～99 之间的整数）

② 文件名：FILE=fn （取值为字符串）

③ 状态说明：STATUS=st （取值为 'old', 'new' , 'unknown'，缺省为 'unknown'）

④ 存取方式：ACCESS=acc （取值为 'sequential', 'direct'，缺省为 'seqential'）

⑤ 格式：FORM=fm （取值为 'formatted', 'unformatted'，缺省：顺序文件有格式，直接文件无格式）

⑥ 记录长度：RECL=rl （每个记录的大小，以字节为单位（与编译器有关），仅用于直接文件）

⑦ 空格含义：BLANK=bl （指定空格的含义，仅用于有格式文件）

⑧ 出错处理：ERR=s （s 是语句标号，OPEN 出错则转向该语句，而不是直接终止）

⑨ 状态返回值：IOSTAT=ios （文件打开成功则返回 0，否则返回一个正整数）

对于有格式顺序存取文件，OPEN 语句中的 ACCESS=和 FORM=两项可以省略。

2、CLOSE 语句

OPEN(说明项)

作用：关闭文件，即解除文件和设备号的连接。

说明项：

① 设备号：[UNIT=]u

② 状态说明：STATUS=st （文件关闭后是否保留）

③ 出错处理：ERR=s

④ 状态返回值：IOSTAT=ios

3、READ/WRITE 语句

一、有格式文件的读写（文本文件）

READ(设备号，格式） 输入列表

WRITE(设备号，格式） 输出列表

格式同前述。注意：

① 如果采用有格式输入，则文件中数据的格式一定要严格按照格式的要求。

② 采用表控格式时，与普通键盘输入有区别：

键盘输入：数据可多可少

文件输入：可多，不可少，将剩余的变量赋值 0。

③同样要注意新记录的问题。

二、无格式文件的读写（二进制文件）

READ(设备号,rec=k,iostat=ferr） 输入列表

WRITE(设备号,rec=k,iostat=ferr） 输出列表

说明：

① rec 用于指定记录号，每次只能操作一条记录。
② iostat 返回状态信息，操作成功返回 0，否则为一个正整数。

4、注意事项

① 数据文件与源程序文件最好位于同一个目录下。
② 若不在同一个目录，需使用绝对路径。

5、文件读写举例

例：二进制文件的读写

调用 data1.dat 文件（二进制数据文件），将全班 30 个同学的三门课成绩求和，且保存在文件 out1.dat 文件（二进制数据文件）中。

源程序：

```
program main
implicit none
real A(30,3), sum(30)
integer i, j

open(88,file='data1.dat',access=’direct’,status='old',recl=8*3*30)
read(88,*) ((A(i,j),j=1,3),i=1,30)
do i=1,30
  sum(i)=0.0
  do j=1,3
     sum(i)=sum(i)+A(i,j)
  enddo
enddo
close(88)
open(99,file='out1.dat',access=’direct’,status='new',recl=8*30)
write(99,rec=1) sum
close(99)
end
```

说明：

① 与普通格式说明语句不同的是：Format 语句中不需要纵向走纸控制符。
② 打开文件 out1.dat 时，使用的是 status='new'，若文件已存在（如再次运行该程序时），则系统会报错，解决方法：A、运行程序前，将该文件删除；B、用 status='unknown'，此时，编译系统会自动判别文件是否存在，若存在，则取值'old'，并将原数据覆盖；否则取值'new'。
③ 输出文件 out1.dat 位于与源程序文件相同的文件夹中。

例：文本文件的读写

从文本数据文件 data2.txt（内容如下）从读取数据，求和后将结果输出到文本文件 out2.txt 中。

data2.txt 中的内容（共有 8 个数据，每个数据小数点后有 8 位）：

```
 48.77690602
-11.61860901
-93.28153003
 26.70879214
 27.26960863
-21.98318190
-70.43355709
-60.60414274
```

源程序：

```
      program main
      implicit none
      integer i, N
      parameter(N=8)
      real a(N), sum

      open(88,file='data2.txt',status='old')
      read(88,100) (a(i), i=1,N)
      sum=0.0
      do i=1,N
        sum=sum+a(i)
      enddo
      close(88)
      open(99,file='out2.txt',status='unknown')
      write(99,110) sum
 100  format(D12.8)
 110  format('sum=', D15.8)
      close(99)
      end
```

13.4 读写矩阵的三种方法

这里介绍将矩阵写入二进制文件的三种方法，以及从二进制文件中读取数据的方法。

一、将矩阵写入二进制文件

```
write(99,rec=k) A(i,j)  ! 将 A(i,j) 作为第 k 条记录写入到文件中
```

这里的 rec 表示记录的序号，每次只能写入一个记录，但长度可以自己定。

将一个双精度矩阵写入文件，可以有下面三种方式实现：

(1) 将每个元素看成一个记录，recl=8

```
      open(99,file='data.dat',access='direct',form='unformatted',recl=8)
      k=0
      do j=1,n
         do i=1,m
            k=k+1
            write(99,rec=k) A(i,j)
         enddo
      enddo
      close(99)
```

(2) 把一列看成一个记录，长度为 8*m

```
open(99,file='data.dat',access='direct',form='unformatted',recl=8*m)
do j=1,n
   write(99,rec=j) A(1,j)
enddo
close(99)
```

(3) 把整个看成一个记录，长度为 8*m*n

```
open(99,file='data.dat',access='direct',form='unformatted',recl=8*m*n)
write(99,rec=1) A
close(99)
```

二、从二进制文件读取数据

```
read(99,rec=k) B(i,j)  ！将第 k 个记录读入到 B(i,j) 中
```

我们可以通过下面三种方式读一个矩阵：

(1) 将每个元素看成一个记录，recl=8

```
open(99,file='data1.dat',access='direct',form='unformatted',recl=8)
k=0
do j=1,n
   do i=1,m
      k=k+1
      read(99,rec=k,iostat=ferr) B(i,j)
   enddo
enddo
close(99)
```

(2) 把一列看成一个记录，长度为 8*m

```
open(99,file='data1.dat',access='direct',form='unformatted',recl=8*m)
do j=1,n
   read(99,rec=j,iostat=ferr) (B(i,j), i=1,m)
enddo
close(unit=99)
```

(3) 把整个看成一个记录，长度为 8*m*n

```
open(99,file='data1.dat',access='direct',form='unformatted',recl=8*m*n)
read(99,rec=1,iostat=ferr) ((A(i,j), i=1,m), j=1,n)
close(unit=99)
```

(4) 读取整个二进制文件

```
      open(99,file='data.dat',access='direct',form='unformatted',recl=8)
      k=0
      do while (.true.)
         k=k+1
         read(99,rec=k,iostat=ferr) x(k)
         if (ferr/=0) exit
      enddo
```