

第五章 子程序

2025.11.13 建档

5.1 基本概念

基本概念 程序代码中需要重复某个功能或重复使用某一段程序代码时，可用子程序。在其它语言中，可以被理解为函数。一个程序包含一个主程序（程序入口）和若干个被按需调用的子程序。

子程序包括：函数子程序(function)、子例行程序(subroutine)、数据块子程序(block data)

分类 子程序又分为：外部子程序和内部子程序。

5.2 外部子程序

分类与描述 外部子程序包括**外部函数子程序**和**外部子例行程序**。外部子程序的定义**位于调用程序单元的外部**。也就是说，它位于主程序的 **END** 语句之外。

5.2.1 外部函数子程序

描述 由一个实现某种特定功能的子程序组成，调用程序单元调用它得到**一个函数值（仅仅只有一个）**。

定义形式 **[类型说明] FUNCTION 函数名([虚参 1,虚参 2,...])** 类型说明定义了返回的函数值的类型

说明语句 变量定义

执行语句 具体的可执行代码

END [FUNCTION [函数名]] 返回值存放在函数名中

调用形式 **函数名(实参 1,实参 2,...)** 或者 **函数名()**

执行步骤

- ① 在调用程序单元的函数调用表达式中，计算实参值。
- ② 将实参值传递给对应虚参，即虚实结合。
- ③ 执行函数体，计算函数值，**并将函数值赋给函数名**。
- ④ 执行 **END** 语句，将函数值带回（即返回）调用程序单元的函数调用表达式。

注意

- ① 实参的**个数、类型和位置顺序**必须与所调用的，函数子程序对应的虚参一致。
- ② 实参名称与虚参名称可以相同，也可以不同。
- ③ 实参与虚参主要由**位置顺序**建立其对应关系。

例题：已知上海、南京、武汉三地的气温，编制外部函数子程序计算三地的平均气温

```
program $251113  主程序
  implicit none
  external average
  real a,b,c,average
  read *, a,b,c
  print *, average(a,b,c)
end program $251113

real function average(x,y,z)  子程序部分
  implicit none
  real x,y,z,average
  average = (x+y+z)/3
end function average
```

对三个虚参进行数据类型定义
注意：这里 average 可以是个变量

external 语句说明 **average** 是一个外部子程序，可以省略。
主程序中也必须对 **average** 进行数据定义。

5.2.2 外部子例行程序

| | |
|------|---|
| 描述 | 子例行程序不仅可求一个值，还可求多个值或不求值而执行某种操作，因此具有更广泛的用途。 |
| 定义形式 | SUBROUTINE 子例行程序名([虚参 1,虚参 2,...]) 不存在类型定义了，程序名不承担变量作用 说明语句 变量定义 执行语句 具体的可执行代码 END [SUBROUTINE [子例行程序名]] |
| 调用形式 | 外部子例行程序的调用必须由调用语句（即 CALL 语句）实现。 一般形式为： CALL 子例行程序名(实参 1,实参 2,...) |
| 执行步骤 | ① 在调用程序单元中，计算实参值。 ② 将实参值传递给对应虚参，即虚实结合。 ③ 执行子程序体，实现其功能。 ④ 执行 END 语句，将返回值带回给调用程序单元。 |

例题：已知上海、南京、武汉三地的气温，编制外部子例行程序计算三地的平均气温

```
program $251113 主程序
  implicit none
  external average
  real a,b,c,temp
  read *, a,b,c
  call average(a,b,c,temp)
  print *, temp
end program $251113

subroutine average(x,y,z,ave)
  implicit none
  real x,y,z,ave
  ave = (x+y+z)/3
end subroutine average
```

按照位置传递数据，这里把 ave→temp。

| | |
|------|---|
| 两者区别 | ① 函数子程序会返回一个数值，并对储存返回值的函数名要有类型声明；子例行程序可以返回 1 个、多个或不返回。 ② 函数子程序调用时不用 call，但调用前要对函数子程序名声明；子例行程序必须用 call 命令调用 ③ 函数子程序体中，由于用函数名存放函数值，故函数名具有类型；而由于子例行程序的计算结果均存放在虚参表中，故子例行程序名既不存放数值也没有类型。 |
|------|---|

5.3 内部子程序

| | |
|-------|---|
| 分类与描述 | 内部子程序包括内部函数子程序和内部子例行程序。内部子程序的定义一般位于调用程序单元中 CONTAINS 语句之后，只有包含内部子程序的程序单元才能调用它们。 |
|-------|---|

5.3.1 内部函数子程序

| | |
|------|--|
| 定义形式 | [类型说明] FUNCTION 函数名([虚参 1,虚参 2,...]) 说明语句 执行语句 END FUNCTION [函数名] |
| 调用形式 | 函数名(实参 1,实参 2,...) 或者 函数名() 与外部函数子程序相同 |

例题：已知上海、南京、武汉三地的气温，编制内部函数子程序计算三地的平均气温

```
program
  implicit none
  real a,b,c
  read *, a,b,c

  因为是内部子程序，无需 external 语句
  real 中也无需定义 average 的类型
```

| | |
|---|--|
| <code>print *, average(a,b,c)</code> | |
| <code>contains</code> | 由该行定义其之下的语句都属于子程序部分 |
| <code>real function average(x,y,z)</code> | |
| <code>implicit none</code> | |
| <code>real x,y,z,average</code> | 必须要对 <code>average</code> 进行类型声明 |
| <code>average = (x,y,z)/3</code> | |
| <code>end function average</code> | 为了和主程序区别, <code>end</code> 后必须增加 <code>function</code> 。 |
| <code>end program \$251113</code> | |

5.3.2 内部子例行程序

定义形式 **SUBROUTINE** 子例行程序名([虚参 1,虚参 2,...])

说明语句

执行语句

END SUBROUTINE [子例行程序名]

调用形式 **CALL** 子例行程序名(实参 1,实参 2,...) 与外部函数子程序相同

例题：已知上海、南京、武汉三地的气温，编制内部子例行程序计算三地的平均气温

| | |
|--|--|
| <code>program</code> | |
| <code>implicit none</code> | |
| <code>real a,b,c,ave</code> | 这里需要定义 <code>ave</code> 的类型 |
| <code>read *, a,b,c</code> | |
| <code>call average(a,b,c,ave)</code> | |
| <code>print *, ave</code> | |
| <code>contains</code> | 由该行定义其之下的语句都属于子程序部分 |
| <code>subroutine average(x,y,z,ave)</code> | |
| <code>implicit none</code> | |
| <code>real x,y,z,ave</code> | |
| <code>ave = (x,y,z)/3</code> | |
| <code>end subroutine average</code> | 为了和主程序区别, <code>end</code> 后必须增加 <code>function</code> 。 |
| <code>end program \$251113</code> | |

- 注意**
- ① 内部子程序的定义应当位于调用程序单元中的 **CONTAINS** 语句之后。
 - ② 内部子程序中不能再包含内部子程序。
 - ③ **END** 语句中的关键字 **FUNCTION** (或 **SUBROUTINE**) 在外部子程序中是可选项，但在内部子程序中是**必选项**。

5.4 实参和虚参之间的数据传递

虚实结合 调用子程序时，用**实参取代虚参**的过程,是不同程序单元数据传递的主要方式。

主要方式 ① **地址传递**：将实参的地址传递给虚参，虚参和实参**拥有相同的内存地址**，虚参值的变化会改变实参值，如 Fortran 语言。

② **值传递**：将实参的值传递给虚参，虚实结合后**虚参不会改变实参的值**，如 C 语言。

虚参类型 变量作为虚参、数组名作为虚参、子程序名作为虚参、星号 (*) 作为虚参

5.4.1 变量作为虚参

概述 子程序用变量作虚参时，对应的实参可以是**同类型**的**常量、表达式、变量或数组元素**。

常量/表达式 实参是常量或表达式时，采用**值传递方式**，将实参常量、表达式之值传递给虚参变量。

变量/数组 实参是变量或数组元素时，采用**地址传递方式**，将实参变量或数组元素的地址传递给虚参变量。调用之前，虚参没有值。调用时，虚参的值就是实参的值。

示例

```
program $251113
  implicit none
  external sub
  integer :: a(5)=(/1,2,3,4,5/), b=8    创建了一个有五个元素的数组
  call sub(a(3),b)                      a(3),b 作为实参往虚参传递
end program $251113

subroutine sub(x,y)
  implicit none
  integer :: x,y
  x=x+3                                  虚参改变，实参一并改变，此处改变了 a(3)的值
  y=y+4                                  此处改变了 b 的值
end subroutine sub
```

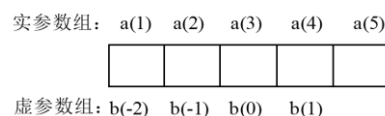
5.4.2 数组名作为虚参

概述 子程序用数组名作虚参时，对应实参可以是**同类型**的**数组名或数组元素**。实参数组和虚参数组可以有**不同的维数和不同的长度**。这时实参和虚参之间以**地址传递方式**实现数据传递。

5.4.2.1 实参是相同维数的数组名

情况 实参与虚参维度相同，但个数可以不同 `call sub(a)`。

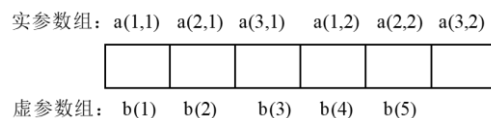
传递规则 虚参数组与实参数组的**第一个数组元素首先结合**，虚、实数组中的其余元素按其**在内存的排列顺序（按列存放）**依次结合。



5.4.2.2 实参是不同维数的数组名

情况 实参与虚参维度不同，实参二维，虚参一维。

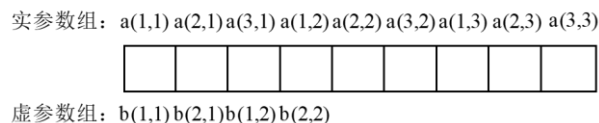
传递规则 主程序调用子程序时，二维实参数组的第一个元素与一维虚参数组的第一个元素首先结合，其余元素按其**在内存的排列顺序**依次结合（注意**列优先规则**）。



5.4.2.3 实参是维数相同，长度不同的数组名

情况 实参二维有 9 个数据，虚参二维有 4 个数据。

传递规则 实参数组的第一个元素与虚参数组的第一个元素首先结合，其余元素按其**在内存的排列顺序**依次结合（注意**两个数组都遵循列优先规则**）。

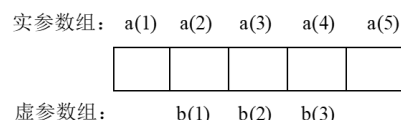


特别注意 **虚参数组元素的个数必须小于等于实参数组元素的个数**，否则，多余的虚参数组元素会导致子程序数据处理出错。**传入的数据只能更多，不能更少。**

5.4.2.4 实参是相同维数的数组元素

情况 右图所示，我们从 a(2)开始传入数据 `call sub(a(2))`。

传递规则 将虚参数组第一个数组元素与实参数组元素首先结合，虚参数组的其余元素与实参数组的后续元素按其**内存排列顺序**依次结合。



5.4.2.5 实参是不同维数的数组元素

传递规则 将虚参数组第一个数组元素与实参数组元素首先结合，虚参数组的其余元素与实参数组的后续元素按其**在内存的排列顺序**依次结合。原则和 5.4.2.2 完全一致。

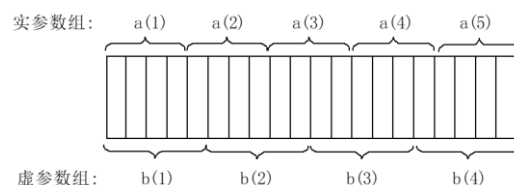
特别注意 虚实结合后，虚参数组的所有元素均应在实参数组的范围之内，否则，由于虚实结合的错误，导致子程序数据处理出错。也就是说实参必须大于等于虚参个数。

5.4.2.6 虚参是字符型数组，实参也是字符型数组

情况 此处 a 为 `character *4 a(5)` 有 5 个长度为 4 的元素

此处 b 为 `character *5 b(4)` 有 4 个长度为 5 的元素

传递规则 虚参数组和实参数组 **不是** 按数组元素的顺序一一对应结合，而是 **按字符位置一一对应结合**。虚实结合时，不仅允许虚实数组的维数不同，同时还允许虚、实数组元素字符的长度可以不同。



5.4.2.7 虚参是可调数组

传递规则 所谓可调数组，是指该数组的维界表达式（即数组的界）是个尚未确定值的整型变量，调用子程序时，该整型变量由调用程序单元给定。

示例

```
subroutine sub2(b,n,m)
  implicit none
  integer i,j,n,m,s
  integer b(n,m)           调用主程序时可以使用 call sub2(a,5,5)
```

5.4.3 子程序名作为虚参

情况 用 **外部函数** 子程序名或 **外部子例行** 程序名作为实参，**必须** 在其说明语句部分用 `EXTERNAL` 语句声明它们是外部子程序名（不可以省略），`EXTERNAL` 语句应位于 `IMPLICIT` 语句之后。如果实参是系统定义的内部函数名（例如 `SIN`），则 **必须** 在其说明语句部分用 `INTRINSIC` 语句声明它们是内部函数名。

一般形式 `EXTERNAL` 语句的一般形式：`EXTERNAL` 外部子程序名 1[, 外部子程序名 2, ...]

`INTRINSIC` 语句的一般形式：`INTRINSIC` 内部函数名 1[, 内部函数名 2, ...]

5.4.3.1 内部函数名作为实参

情况 调用程序单元中用 `INTRINSIC` 语句对内部函数名予以声明。

示例

```
PROGRAM ex10_09
  IMPLICIT NONE
  INTRINSIC SIN, COS, TAN           声明内部函数名
  REAL triangle, y1, y2, y3
  y1 = triangle(60.0, SIN)          可以传递函数 SIN
  y2 = triangle(30.0, COS)
  y3 = triangle(45.0, TAN)
  PRINT *, "y1=", y1, "y2=", y2, "y3=", y3
END PROGRAM
FUNCTION triangle(deg, fun)
  IMPLICIT NONE
  REAL triangle, deg, fun           此处定义的不是函数，是 fun 返回的值的类型
  triangle = fun(deg * 3.1415926 / 180)  可以直接调用传入的函数名
END FUNCTION triangle
```

5.4.3.2 外部子程序名作为实参

情况 调用程序单元中用 `EXTERNAL` 语句对外部子程序名予以声明

示例

```
PROGRAM ex10_10
  IMPLICIT NONE
```

```

EXTERNAL absy, sqrty, test      全部声明外部子程序
REAL y, r
CHARACTER*1 c
PRINT *, "输入变量的值: "
READ *, y
PRINT *, "输入选择符('A'或'S'): "
READ *, c
IF (c == 'A' .OR. c == 'a') THEN
    CALL test(y, absy, r)
    PRINT *, "y 的绝对值是: ", r
ELSEIF (c == 'S' .OR. c == 's') THEN
    CALL test(y, sqrty, r)
    PRINT *, "y 的平方根是: ", r
ELSE
    PRINT *, "选择符输入错误!"
ENDIF
END PROGRAM

选择 'A': 调用 absy 计算绝对值
选择 'S': 调用 sqrty 计算平方根

SUBROUTINE test(x, sub, z)
    IMPLICIT NONE
    REAL x, z
    CALL sub(x, z)
END SUBROUTINE test

SUBROUTINE absy(a, z)
    IMPLICIT NONE
    REAL a, z
    z = ABS(a)
END SUBROUTINE absy

SUBROUTINE sqrty(b, z)
    IMPLICIT NONE
    REAL b, z
    z = SQRT(b)
END SUBROUTINE sqrty

```

5.4.4 变量的作用域和生存周期

变量情况

外部子程序中用到的所有变量:

- ① 被调用前通常都没有确定的存储单元。
- ② 被调用时才临时分配存储单元。
- ③ 调用后这些存储单元又都被释放。因此子程序中变量的生存期仅局限在该子程序被调用期间，而变量的作用域也仅限于该子程序的范围之内。在变量声明中加 **SAVE** 可以增加变量的生存周期、保留住所保存的数据。（事情上有些编译器不管声明中有没有指明 **SAVE**，都会让变量永远记住数值，例如 visual fortran，不过为了确保正确性、增加代码的可移植性，最好加上 **save**）

```

SUBROUTINE sub(a, b, *, *)
    IMPLICIT NONE
    REAL a, b
    IF (a < 0 .and. b < 0) THEN
        RETURN 1
    ELSE
        RETURN 2
    ENDIF
END SUBROUTINE sub

```