1.语法

1、if...then...else语句

name均可不用

```
[name:] if (logical expression) then
   ! various statements
   . . .
   else
   !other statement(s)
   . . .
end if [name]
```

2、select case语句

```
[name:] select case (expression)
   case (selector1)
   ! some statements
   ... case (selector2)
   ! other statements
   ...
   case default
   ! more statements
   ...
end select [name]
```

其中case可以指定范围

```
case (low:high)
```

3、循环语句

(1)do循环

```
[name:]do var = start, stop [,step]
   ! statement(s)
   …
end do [name]
```

- 循环变量var为一个整数

(2)do...while循环

```
[name:]do while (logical expr)
   statements
end do [name]
```

例子：

```
program factorial
implicit none

   ! define variables
   integer :: nfact = 1
   integer :: n = 1
```

```
   ! compute factorials
   do while (n <= 10)
      nfact = nfact * n
      n = n + 1
      print*,  n, " ", nfact
   end do
end program factorial
```

(3)exit

直接跳出当前循环，继续执行下一条语句

(4)cycle

当语句进行到cycle，不再向下执行，执行指针直接拨回条件判断语句。只是在循环的中间跳过**单个特定**的数字。

(5)stop

让程序停止

2.变量

数字

数字类型	定义方法
一般整形	`integer :: x`
长整形	`integer(kind=4) :: x`
短整形	`integer(kind=2) :: x`
浮点数(单精度，有效位数6~7)	`real :: r`
浮点数(双精度，有效位数15~16)	`real(kind=8) :: r`
复数	`complex(kind=4) :: c`
复数(双精度)	`complex(kind=8) :: c`

- 长、短整形的区别在于：能存储整形数据的范围不同
- 单、双精度的区别在于：有效位数不同
- `real :: r`默认为单精度`kind = 4`浮点数
- `complex :: c`默认为单精度`kind = 4`

2、kind控制符

- 定义分配的长度/精度`integer(kind=2) :: shortval`
- 查询程序内数据精度

```
program kindCheck
implicit none

   integer :: i
   real :: r
   complex :: cp
   print *,' Integer ', kind(i)
   print *,' Real ', kind(r)
   print *,' Complex ', kind(cp)

end program kindCheck
```

字符：声明和拼接

```
program hello
explicit none

character(len=10)::title
character(len=10)::tail
character(len=10)::name

title='Good'
tail='morning'
name=title//tail
print *,title,tail
print *,name(2: 4)
```

常用字符函数

函数	作用
int len(string)	返回字符串的长度
int index(string , sustring)	返回子串的第一个字母在字符串中的位置（如果没有找到则返回0）
char achar(int)	将整数转换成一个字符
int iachar(chars)	它可将一个字符转换为整数
string trim(string)	它返回去掉尾随空格的字符串。
scan(string, chars)	它会搜索“string”由左到右（除非back=.true）包含在“string”任何字符的第一次出现。它返回一个整数，该字符，或零的位置，如果没有文字的“字符”已被找到。
verify(string, chars)	它扫描“string”由左到右（除非back=.true）不包含在“string”任何字符的第一次出现。它返回一个整数，该字符的位置，如果只在“字符”的字符被找到，或者没有找则返回零。
string adjustl(string)	将输入字符串开始的空格，移到其末尾输出
string adjustr(string)	将输入字符串末尾的空格，移到其开头输出
len_trim(string)	它返回一个整数等于“string”(len(string))减去尾随空白的数量
repeat(string,ncopy)	它返回一个字符串长度等于“ncopy”次数“string”的长度，并含有“string”的“ncopy”串联拷贝

数组

Fortran最多允许7维数组，每个数组内存储的元素类型一致

- **数组的索引值可以从任意设定开始**，例如：`integer A(5)`，从A(1)~A(5); `integer A(-1:3)`，从A(-1)~A(3)
- **多维数组低维优先**，与C相反，A(最低维，…，最高维)，即从左向右填满

```
real,dimension(5)::numbers !5个元素的一维数组，分别为number(1)--number(5)
real,dimension(5,5)::matrix !5*5矩阵
...
numbers=(/1,2,3,4,5/) !一维数组可以直接赋值，或用循环赋值
```

直接用`array(lower:[upper]:[stride])`调用数组多个元素

动态数组

声明1

```
类型，dimension(:,:),allocatable::array
...
deallocate(array) !数组使用后要解除存储器
```

声明2:可用于初始化多个阵列，或用于阵列部分的初始化(使用后不用解除存储器)

```
data variable /.../
```

例子

```
program dynamic_array
implicit none

   !rank is 2, but size not known
   real, dimension (:,:), allocatable :: darray
   integer :: s1, s2
   integer :: i, j

   print*, "Enter the size of the array:"
   read*, s1, s2

   ! allocate memory
   allocate ( darray(s1,s2) )

   do i = 1, s1
      do j = 1, s2
         darray(i,j) = i*j
         print*, "darray(",i,",",j,") = ", darray(i,j)
      end do
   end do

   deallocate (darray)
end program dynamic_array
```

```
program dataStatement
implicit none

   integer :: a(5), b(3,3), c(10),i, j
   data a /7,8,9,10,11/

   data b(1,:) /1,1,1/
   data b(2,:)/2,2,2/
   data b(3,:)/3,3,3/
   data (c(i),i=1,10,2) /4,5,6,7,8/
   data (c(i),i=2,10,2)/5*2/

   Print *, 'The A array:'
   do j = 1, 5
      print*, a(j)
   end do

   Print *, 'The B array:'
   do i = lbound(b,1), ubound(b,1)
      write(*,*) (b(i,j), j = lbound(b,2), ubound(b,2))
   end do

   Print *, 'The C array:'
   do j = 1, 10
      print*, c(j)
```

```
    end do

end program dataStatement
```

向量乘法/矩阵乘法

函数	作用
real dot_product(vector_a, vector_b)	向量点乘，两个输入向量必须同样长
matrix matmul (matrix_a, matrix_b)	矩阵相乘，两个相乘的矩阵必须大小相似，例如(m,k)和(k,n)

（跳过了还原~~操作函数）

结构体

声明

```
type type_name
  !变量声明
  character(len=5)::title
  integer::a,b
  complex::com
 end type
```

访问数据

结构体后用%可访问变量

```
book1%title='name'
book1%a=1
book1%b=2
book1%complex=(1,2)
```

结构体类

有以下两种方法：

1. 声明和已定义结构体(**BOOK**)数据相同的另一个变量(**book1**) `type(BOOK)::book1`
2. 声明派生数组 `type(BOOK),dimension(2)::list`，即定义了两个结构体**list(1)、list(2)**

3.指针

见工程test2
在Fortran中，指针不仅仅存储地址，而存储多功能性的数据对象。它包含有关特定对象的详细信息，如类型，等级，扩展和存储器地址。

声明

```
integer, pointer :: p1 !声明指向整数的指针
real, pointer, dimension (:) :: pra !指向一维实向量的指针
real, pointer, dimension (:,:) :: pra2 !指向二维实向量的指针
...
integer,target :: a !声明可以被指针指向的变量
```

```
! 使指针指向变量
integer,target::a
integer,poiter::p1
```

```
pi=>a
a=2
p1=3
```

```
! 给指针分配内存空间
integer,poiter::p2
allocate(p2)
p=100
···
deallocate(p2) !释放分配的空间
```

指针和变量

在fortran中，改变指针的值就是改变指向变量(或者分配内存)的值

1. 指针→变量：`p => a`,可以随时设置指针指向，即，同一指针可随时指向不同的变量
2. 指针分配内存(等价于一个变量使用)：改变指针指向之前要先释放空间`deallocate(pointer)`
3. 多个**pointer**可以指向同一个**target**

4.程序

子程序：subroutine

```
subroutine name(子程序的输入变量)
 ···
 ···
 return !表示返回调用处继续执行主程序，可省略，会自动return
end[subroutine name]
```

- 子程序可以在程序的**任何地方**被调用，包括：主程序、其他子程序、甚至自己调用自己(递归)
- 子程序中，变量声明独立
- 调用方法：`call subroutine`

自定义函数：function

```
function fun1(输入的处理参数)
 ···
 real/integer/complex :: fun1 !这个与函数名相同的变量代表此函数返回的数值类型
 ···
 fun1=···
 [return]
end
```

或者可以简化为

```
real/integer/complex function fun1(inputs)
 ···
 fun1=···
 [return]
end
```

以上`return`都可以省略

自定义函数与子程序的不同：

1. 自定义函数调用：和变量一起声明`类型,external ::f unction`，不用`call`命令

```
program test
···
 real,external :: fun1 ! 声明fun1是函数，不是变量
```

```
 ...
end
```

2. 自定义函数有一个返回值：与函数名称相同

全局变量

使用**内存地址对应**的方法来传递数据，common内部区域中也遵循此原则

```
programme test
 real :: x,y,z
 common x,y,z !说明在不署名的全局变量区域中的3个内存位置依次存储了3个变量
 ! 将common细分为彼此独立的区域的方法
 ! common /group1/ a
 ! common /group2/ b
 ...
end

subroutine sub()
 ...
 real :: a,b,c
 common a,b,c !a=x,b=y,c=z
end subroutine sub
```

为全局变量赋值：使用block data模块来设置，**全局变量不能声明成常量(parameter)**

```
programme test
implicit none
 real :: a,b
 common a,b
 ...
end

block data !此模块的功能仅仅用来填写全局变量的数据内容
implicit none
 real :: a,b
 common a,b

 data a,b /1,2/ ! a=1,b=2
end block data
```

模块：module

用以封装程序模块

```
module module_name
 ...
end module
```

通过 `use module_name` 来调用这个模块，若不加说明，则module中的变量为局部变量，使之变为全局变量的两种方法:

```
real :: a,b
common a,b !声明为全局变量的一般方法
```

```
real,save :: a,b !通过加入save来做到记录的功能
```

使用多个文件：可以直接在VS中添加入*source files*，注意，**多个文件中只能有一个主程序存在**

1. 独立常用函数，给其他程序使用
2. 加快编译速度

5.文件

见工程test3

写数据到txt文件

```
open(unit=number,file='hello.txt')
write(number,[fmt=]*)"hello" !默认输出格式
```

`integer :: unit`是打开待写入文件的代码，最好避开1、2、5、6

write、print、read

`write(*,*)"text"`，完整写法为`write(unit=*,fmt=*)"text"`

- 第一个星号表示：默认的输出位置 即，屏幕，unit=* *等价于unit=6*
- 第二个星号表示：默认的输出格式

`print *,"text"`，与write的不同在于**少了第一个星号，只能输出到屏幕，不能指定为其他输出位置**