Fortran 实用简介


Shu-yi Wei

魏树一

http://shuyi.eu

shuyi [AT] mail.sdu.edu.cn


目 录

第一章
Fortran 文件与编译器

本书以 Fortran77 版本为例介绍如何利用 fortran 语言实现数值计算。

Fortran77 文件扩展名为 ".f"。所有源代码都是以文本形式存储这些文件中。一套完整的代码必须包含且仅能包含一个主程序 program，可以包含多个 subroutine 和 function。这些 program、subroutine 和 function 可以存储在不同的 "*.f" 文件中。例如，一套完整的代码可以包含以下几个文件：

```
main.f                          ! 用以存储主程序
sub1.f                          ! 用以存储一个或几个 subroutine
sub2.f                          ! 用以存储一个或几个 subroutine
func1.f                         ! 用以存储一个或几个 function
func2.f                         ! 用以存储一个或几个 function
```

编译这些 "*.f" 文件需要预先安装 fortran 语言的编译器。编译器可以随个人喜好有不同选择。常用的编译器有 gfortan、ifort 等。编译上面这样一套代码可以使用如下命令：

```
gfortran -o prog.exe main.f sub1.f sub2.f func1.f func2.f
```

这样就可以生成一个名为 "prog.exe" 的可执行文件。然后在终端内执行 "./prog.exe" 就可以运行该程序。

第二章
代码格式与数据类型

首先，fortran 代码并不区分大小写，可以混合使用。

在 Fortran77 语言环境下写 Fortran 代码时，格式要求非常严格。首先，每行前面必须有六个空格，且不可超过 72 列。前六个空格是有起作用的。如果第一个列有 c，* 或者! 字符，那么改行就被注释掉了。所有代码不起作用。在第 2-5 列可以认为分配一个数字号。该数字号在一个 program，subroutine 或者 function 中需是唯一的。这个数字号是用来配合 goto 跳转使用的。第 6 列是给链接符预留的。由于 fortran77 不允许一行代码超过 72 列。因此，当一行代码过长的时候可以分成几行去写，后面几行代码中在第 6 列需要写上任意字符。下面是一段简单的代码示例。

```
! 这一行被注释掉了。
      PROGRAM main
      implicit double precision (a-h,o-z)

      i1 = 1
      i2 = 2
      i3 = 3
      i4 = 4

      goto 91

      write(*,*)  i1
     v          * i2

 91   continue

      write(*,*)  i3
     v          * i4

      END PROGRAM
```

Fortran 语言是从 program 开始一行一行的开始向下执行的。当执行到第 10 行的时候，由于 goto 的存在，就会直接跳转至被 91 标记的那一行（即第 15 行）。然后，从这一行继续执行。12

和 13 行的代码就不会起作用。因此，当编译运行上面这行代码的时候，输出的结果是 i3×i4 的值也就是 12。而第 18 行第 6 列的 v 字符就表示 17 行与 18 行是连在一起的。

Fortran 语言的默认类型非常强大。使用 “implicit double precision (a-h,o-z)” 语句后，所有未另行声明的，以 a-h 或 o-z 开头的变量都是双精度，i-n 开头的变量都是整数。这样在 fortran 编程过程中，如果要用到一些变量的时候就无需预先声明其数据类型了。另外一种声明方法是使用 “implicit none”，也就是说关闭 fortran 默认的变量类型。这时 code 中用到的所有变量都需要在该 subroutine、function、program 里预先声明其变量类型。这样上面的这段 code 就变成：

```
! 这一行被注释掉了。
      PROGRAM main
      implicit none
      integer i1, i2, i3, i4

      i1 = 1
      i2 = 2
      i3 = 3
      i4 = 4

      goto 91

      write(*,*)  i1
     v          * i2

 91   continue

      write(*,*)  i3
     v          * i4

      END PROGRAM
```

所有没有放入 common 块里的变量都是局域变量。也就是说这些变量只能在本 subroutine、function 内用。如果要想跨 subroutine 使用这些变量，有两种办法：（1）使用函数的变量传递；（2）将其放入 common 块中。

```
      PROGRAM main
      implicit double precision (a-h,o-z)

      common /id/ i1

      i1 = 1
      i2 = 2
      i3 = 3
      i4 = 4

      call printer1
```

```
12      call printer2(i3, i4)
13
14      END PROGRAM
15
16      subroutine printer1
17      implicit double precision (a-h,o-z)
18      common /id/ i1
19      write(*,*) "i1=", i1, ", i2=", i2
20      end subroutine
21
22      subroutine printer2(i3,i4)
23      implicit double precision (a-h,o-z)
24      common /id/ i1
25      write(*,*) "i3=", i3, ", i4=", i4
26      end subroutine
```

运行上面这段代码，你就会发现在 printer1 中，i1 因为放在了 common 块中，所以它的数值从主程序中正确的传递到了 subroutine 中。而 i2 的值就出现错误，没能正确传递过去。在 printer2 中，由于 i3 和 i4 都通过 subroutine 的参数正确的传递了过去。

数组是通过 dimension 定义的：

```
1       dimension a(-5:5)      ! 定义一个从 a(-5) 到 a(5) 的局域数组
2       common /dis/ b(-5:5)   ! 定义一个从 b(-5) 到 b(5) 的全局数组
```

另外，“read、write” 语句可以用来输入输出。其后跟的 “(*,*)” 中，第一个 * 号代表输出到的媒体。* 号表示默认媒体，也就是输出在终端中，第二个 * 号代表输出格式。* 号为默认格式。当然你也可以选择自定义格式，不过通常情况下，使用 * 号就能满足我们的日常需求。

下面是一个简单的代码用以读入输出数据到特定文件内。

```
1       PROGRAM main
2       implicit double precision (a-h,o-z)
3
4       open(UNIT=13,FILE="input.dat")     ! 打开 input.dat 文件，文件编号为 13
5       open(UNIT=14,FILE="output.dat")    ! 打开 output.dat 文件，文件编号为 14
6
7       read(13,*) i1, i2                  ! 从 input.dat（编号：13）文件中读入两个
            整数，并将其分别赋值给 i1, i2
8       write(14,*) i1*i2                  ! 将 i1*i2 的结果输出到 output.dat 文件
            中。（编号：14）
9
10      close(13)                          ! 关闭 input.dat 文件
11      close(14)                          ! 关闭 output.dat 文件
12
13      END PROGRAM
```

第三章
逻辑判断

Fortran 使用 if 语句实现逻辑判断。常用的判断语句有：equal（.eq.），greater than（.gt.），less than（.lt.），greater than or equal to（.ge.），less than or equal to（.le.），并且（.and.），或（.or.）等。

如果满足条件后，只执行一条命令的话，可以使用如下方式：

```
      PROGRAM main
      implicit double precision (a-h,o-z)

      ix = 5
      x = ix * 10d0

      if (ix .ge. 5) write(*,*) x

      END PROGRAM
```

但如果满足条件后需要执行一条或多条命令，可以使用如下方式：

```
      PROGRAM main
      implicit double precision (a-h,o-z)

      ix = 2

      if (ix .eq. 1) then
      x = ix * 1d0
      write(*,*) x
      else if (ix .eq. 2) then
      write(*,*) ix
      else
      write(*,*) "ix is larger than 2."
      end if

      END PROGRAM
```

注意，最后一个 else 语句如果没有要加条件的话，其后不可加 then。

第四章

循环

Fortran 的循环是通过 do 语句实现的：

```
      PROGRAM main
      implicit double precision (a-h,o-z)

      do iX = 1, 10    ! 循环开始
          x = iX * 0.1
          write(*,*) x
          if (ix .eq. 5) goto 91
      end do

 91   continue

      END PROGRAM
```

这个循环是从 iX = 1 开始的，一直循环到 iX = 10。每次循环之后，iX 都会自动加一。整个循环中 iX 不可以重新赋值。

结束循环有两种方式，一种是让该循环进行到底，自然结束。另外一种就是用 if 判断语句，当满足某些条件时，可以使用 goto 语句，跳转出来。

第五章
子程序与子函数

子程序和子函数的区别可能会比较困扰人。二者比较类似，基本上可以实现的功能也很接近。我在这里讲一下实用的用法。

subroutine 是没有返回值的。在主程序中通过 call <subroutine name> 实现功能执行。

```
      PROGRAM main
      implicit double precision (a-h,o-z)

      x = 1.0d0
      call doublex(x, y)
      write(*,*) y

      END PROGRAM

      subroutine doublex(x, y)
      implicit double precision (a-h,o-z)

      y = 2d0 * x
      end subroutine
```

这个 subroutine 的参数是 x，y。在示例中，doublex 子程序将接收到的 x 参数乘 2 后赋值给 y，然后将 x，y 的值传递回主程序。

而函数则是通过赋值的方式将结果传递回主程序。

```
      PROGRAM main
      implicit double precision (a-h,o-z)

      x = 1.0d0
      y = doublex(x)
      write(*,*) "x = ", x, "y = ", y

      END PROGRAM

      function doublex(x)
```

```
11      implicit double precision (a-h,o-z)
12
13      x = 2d0 * x
14      doublex = x
15      return
16      x = 0d0              ! 该语句在 ruturn 之后，不起任何作用。
17      end function
```

在这个程序中，函数首先将 x 的值乘 2 并将结果重新赋给 x 变量，然后将新 x 值赋给 doublex 函数，并通过 return 返回主程序。这样一个程序就可以看到，x 的值变成了 2，通常函数的计算结果也赋给了 y 值。通常不建议在函数内部修改变量的大小。