Fortran95 程序设计

彭国伦 编著

第4章 输入输出及声明

4-1 输出命令(WRITE) (PRINT)

- Fortran通常以PROGRAM描述来开头，后面接一个自定义的程序名称。
- 最后以END表示程序代码编写结束。可以有三种方式：
  - End
  - End Program
  - End Program main

程序开始 ——→ program main ←—— MAIN是自定义的名称

…….

主程序代码 ——→ write(*,*) "Hello"

…….

程序结束 ——→ stop ←—— 这一行程序可以省略

主程序代码结束 ——→ end

[ex0401.f90]

4-1 输出命令(WRITE) (PRINT)

- Write (*，*) “就是这么简单”！简写的写法
- Write (UNIT=*，FMT=*) “就是这么简单”！完整的写法
- Write (6，*) “String”！严谨一些的写法
- Write (UNIT=6，FMT=*) “String”！最严谨的写法
  - UNIT：输出位置（第9章详细介绍）
  - FMT：输出格式
- 每执行一次WRITE，会自动换到下一行
- 用英文的双引号或单引号把输出的字符串包含起来
- 如要输出双引号，则连续使用两个双引号。例如要输出 My name is “Peter”.时，要编写为

  Write (*，*) “My name is ““Peter””.”

print *, “Hello”

- PRINT的用法和WRITE大致上相同，但其后面不使用括号，只有一个星号（表示不限定输出格式）。
- PRINT没有赋值输出位置的能力，只能针对屏幕来使用
- 建议尽量使用WRITE来做输出工作。



[ex0402.f90] [ex0403.f90]

[ex0402.f90]

```
program main
print *,"Hello"
stop
end program main
```

[ex0403.f90]

```
PROGRAM main
WRITE(*,*) 'Hello'
STOP
END
```

4-2 声明

Fortran变量是一个数据对象，它的值在程序运行期间可以改变。当Fortran编译器遇到变量时，它给变量预留已知的内存单元，无论何时程序使用变量，就引用该存贮位置。

声明：在程序代码中，程序员向编译器要求预留一些存放数据的内存空间

```
1. program ex0404
2. integer a    ！声明一块保存整数的内存空间，以a来表示
3. a=3          ！把 a 设置为3
4. write(*,*) “a=”,a     ！显示 a 的内容
5. stop
6. end
```

执行结果：　a=　　3　　[ex0404.f90]

声明变量的原则：

1.使用英文字母，可用下划线或数字，但前缀必须是英文

2.Fortran90、95中变量长度1~31个字符之间

3.不能和Fortran的执行命令同名，也不能和主程序的名称或是前面声明过的变量同名。

4.变量不分大小写。因为变量名字中不能包含空格，因此要尽可能使用下划线使变量名有意义。

5.程序单元中每个变量名字唯一。

有效变量名：time、distance、z123456、I_want_to_go_home

无效变量名：This_is_a_very_long_variable_name、This_is_a_very_very_very_very_very_very_very_very_long_variable_name、3_days、A$

有意义变量名举例：Exchange_rate

4-2-1 整数类型（INTEGERAL）

Integer a ←

A是自己取的名字，在程序代码中就以这个名字来表示这一块存放整数的空间。它习惯上称为“变量”

声明要使用整数形态数据

整型常数：

不包含小数点的任一数据。

有效的：0、-999、123456789、+17

无效的：1,000,000、-100.



[ex0405.f90]

```
program ex0405
  integer a
  a=2+2*4-3
  write(*,*) "2+2*4-3=",a
stop
end
```

因为目前所使用的都是整形数据，所以计算结果如果有小数点都会忽略不计。

```
program ex0405_2
  integer a
  a=3/2
  write(*,*) “3/2=",a
stop
end
```


```
program ex0405_3
  integer a
  a=1/2
  write(*,*) “1/2=",a
stop
end
```

[ex0406.f90]

```
program ex0406
  integer(kind=4) write !把变量取名为write是不好的做法
  write=2+2*4-3  !这一行看起来像是要输出，但却不是
  write(*,*) "2+2*4-3=",write
stop
end
```



赋值长整型的声明方法：

```
integer(kind=4) a !Fortran 90 添加
interger*4  b        !Fortran 77 传统做法
integer(4)  c        !Fortran 77 传统做法
```

赋值短整型的声明方法：

```
integer(kind=2) a !Fortran 90 添加
interger*2  b        !Fortran 77 传统做法
integer(2)  c        !Fortran 77 传统做法
```

赋值更短整型数的声明方法：

```
integer(kind=4) a !Fortran 90 添加
interger*4  b        !Fortran 77 传统做法
integer(4)  c        !Fortran 77 传统做法
```

4-2-2 浮点数（REAL）

浮点数和整数的最大不同在于浮点数可以保存小数

```
real  a
```

声明使用4 bytes单精度浮点数的方法：

```
real(kind=4) a !Fortran 90 添加
real *4  b          !Fortran 77 传统做法
real(4)  c          !Fortran 77 传统做法
```

声明使用8 bytes双精度浮点数的方法：

```
real(kind=8) a !Fortran 90 添加
real *8  b          !Fortran 77 传统做法
```

[ex0407.f90]

```
program ex0407
  real :: a ! 声明浮点数类型变量a
  a = 2.5+3.0/2.0
  write(*,*) '2.5+3.0/2.0=', A ! A和a指的是同一个变量
  stop
end
```


```
program ex0407_2
  real :: a ! 声明浮点数类型变量a
  a = 2.5+3/2
  write(*,*) '2.5+3.0/2.0=', a
  stop
end
```

[ex0408.f90]

```
program ex0408
  real(kind=4) :: a,b
  a=1 ! a=1.000000
  b=0.1
  write(*,*) a,"+",b,"=",a+b
  stop
end
```

新的用法：

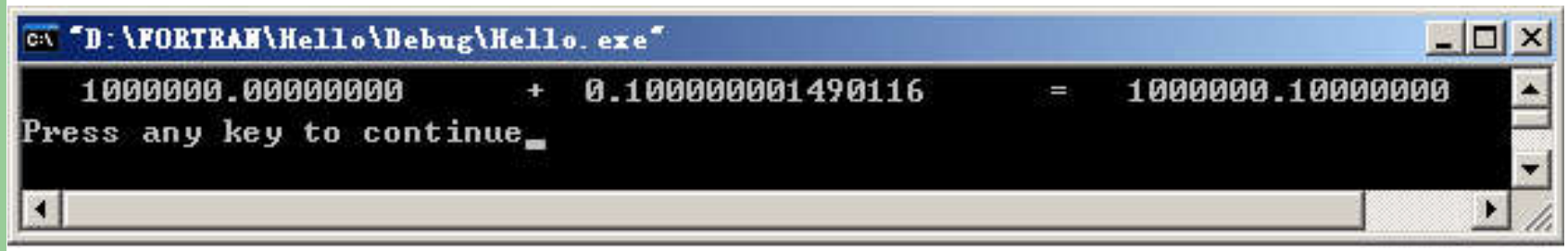
```
write(*,*) a,"+",b,"=",a+b
```

可以直接放入一个算式

```
program ex0408_2
  real(kind=4) :: a,b
  a=1000000. ! a=1000000.
  b=0.1
  write(*,*) a,"+",b,"=",a+b
  stop
end
```

计算机在保存浮点数时，会先把它转成科学计数法。单精度浮点数转换成科学计数浮点数时，数值的部分只能保存6位数字，超过的部分会被忽略。“单精度浮点数的有效位数为6位。

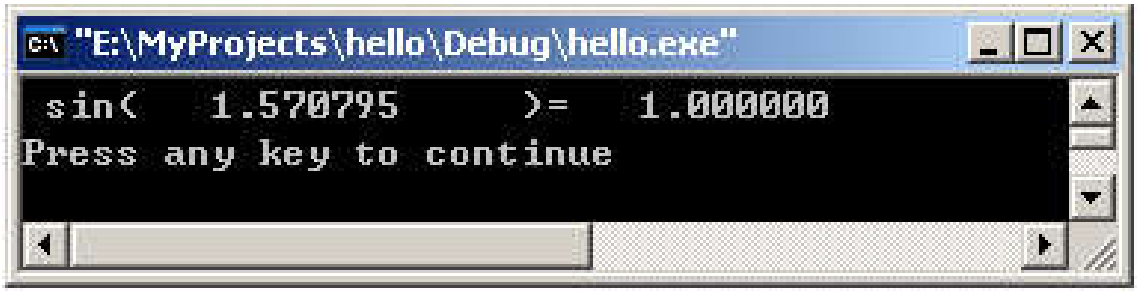
```
program ex0408_3
  real(kind=8) :: a,b
  a=1000000. ! a=1000000.
  b=0.1
  write(*,*) a,"+",b,"=",a+b
  stop
end
```



双精度浮点数的有效位数为15为，可以正确计算出a+b的结果。但运算结果如果超过15个位数，还是会出现计算误差。

[ex0409.f90]

```
program ex0409
  real :: a
  a=3.14159/2.0
  write(*,*) "sin(",a,")=",sin(a)
  stop
end
```



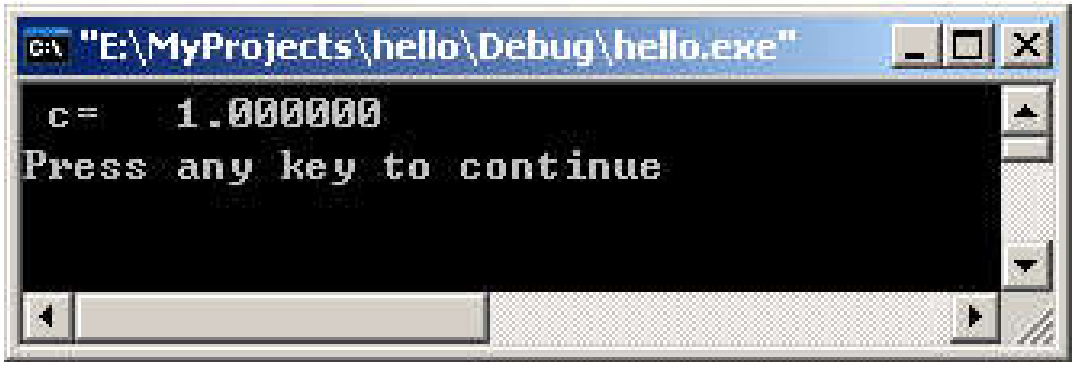
新的用法：

```
write(*,*) "sin(",a,")=",sin(a)
```

计算sin(a)

[ex0410.f90]

```
program ex0410
  real :: a,b,c
  a=0.5
  b=0.5
  c=sin(a)**2 + cos(b)**2
  write(*,*) "c=",c
  stop
end
```



4-2-3 复数（COMPLEX）

complex a
复数由实部和虚部两个部分组成，用两个浮点数来保存

复数赋值精度的方法：

```
complex (kind=4) a !单精度，Fortran 90 添加
complex (kind=8) a !双精度，Fortran 90 添加
complex *4  b      !单精度，Fortran 77 传统做法
complex *8  b      !双精度，Fortran 77 传统做法
complex(4)  c      !单精度，Fortran 77 传统做法
complex(8)  c      !双精度，Fortran 77 传统做法
```

设置复数数值的方法：

```
a=(x,y) !x为实部，y为虚部
```



[ex0411.f90]

```
program ex0411
  complex :: a,b
  a=(1.0,1.0)  !a=1.0+1.0i
  b=(1.0,2.0)  !b=1.0+2.0i
  write(*,*) "a+b=",a+b
  write(*,*) "a-b=",a-b
  write(*,*) "a*b=",a*b
  write(*,*) "a/b=",a/b
  stop
end
```

4-2-4 字符及字符串（CHARACTER）

字符类型是用来保存一个字符或一长串字符所组成的“字符串”时所使用的类型。

声明字符串的方法：

```
character(len=10) a ! Fortran 90添加
character(10) b       ! Fortran 77已有
character*10 c        ! Fortran 77已有
character*(10) d      ! Fortran 77已有
```

设置字符串变量内容的方法：

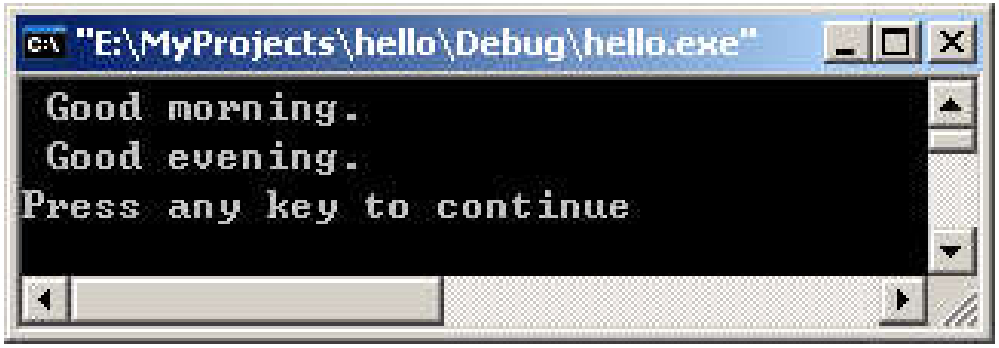
```
a=“Hello” ！Fortran 90可以用双引号来封装字符串
b=‘Hello’ ！Fortran 77只能用单引号来封装字符串
c=“That’s right.” !使用双引号封装字符串时，可以在字符串中任意使用单引号
d=‘That’’s right.’ !用单引号封装字符串时，输出单引号时要连续用两个单引号
e=“That’s ““ right””.” !用双引号封装字符串时，输出双引号时要连续用两个双引号
```

[ex0412.f90]

```
program ex0412
  character a
  character(len=10) b
  a='H'
  b="ello"
  write(*,*) a,b
end
```

[ex0413.f90]

```
program ex0413
  character(len=20) string
  string = "Good morning."
  write(*,*) string
  string(6:) = "evening." ！重设设定从第6个字符之后的字符串
  write(*,*) string
end
```



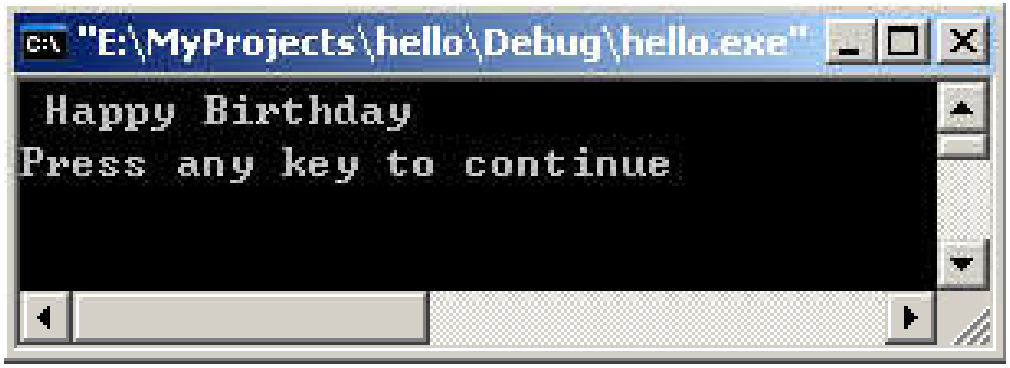
字符串变量后面加上括号，再通过冒号来区分索要重新设置的字符串位置范围，可以重新设置字符串中某一部分的内容。

string(1:2)=“Go”！字符串最前面的两个字符会变成Go

string(13:13)=“!”！字符串的第13个字符会变成叹号！

[ex0414.f90]

```
program ex0414
  character(len= 6) first
  character(len=10) second
  character(len=20) add
  first="Happy "
  second="Birthday"
  add = first//second !经由两个连续的除号可以连接两个字符串
  write(*,*) add
end
```



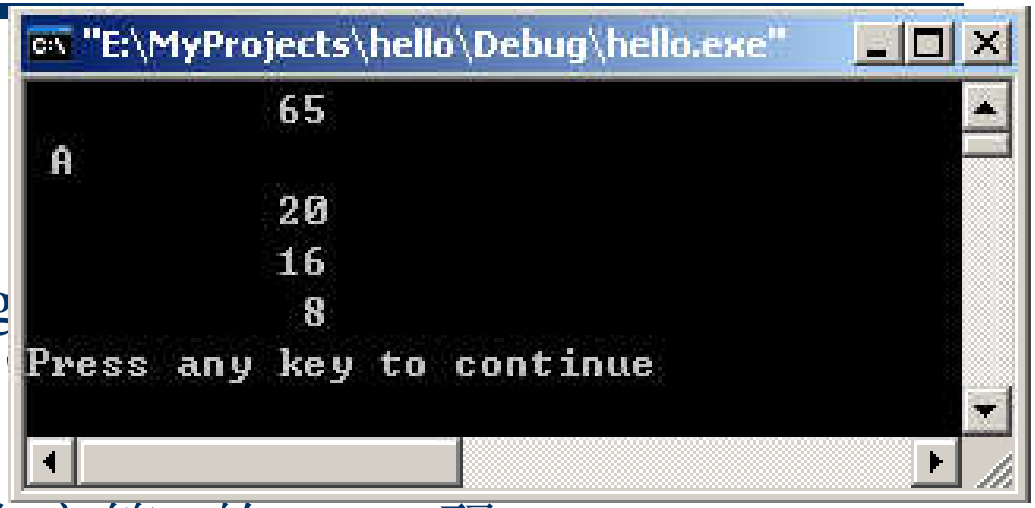
两个字符串之间可以“相加”【//】，结果是字符串合并

有关字符串运行的函数

Char(num)	返回计算机所使用的字符表上，数值num所代表的字符
IChar(char)	返回所输入的char字符在计算机所使用的字符表中所代表的编号，返回值是整数类型
Len(string)	返回输入字符串的声明长度，返回值是整数类型
Len_Trim(String)	返回字符串去除尾端空格后的实际内容长度
Index(string,key)	所输入的String和key都是字符串。这个函数会返回key这个“子字符串”在“母字符串”String中第一次出现的位置
Trim(string)	返回把string字符串尾端多余空格清除过后的字符串

[ex0415.f90]

```
program ex0415
  character(len=20) string
  character(len=5) substring
  string = "Have a nice day."
  substring = "nice"
  write(*,*) ichar('A') ! 输出字符A的ASCII码
  write(*,*) char(65)   ! 输出ASCII码65所代表的字符,也就是A
  write(*,*) len(string) ! 输出字符串string声明时的长度
  write(*,*) len_trim(string) ! 输出字符串string内容的长度
  write(*,*) index(string, substring) ! nice在Have a nice day的第
                                      ! 8个位置
end
```



4-2-5 逻辑变量（LOGICAL）

逻辑变量主要是在逻辑判断时使用
它只用来保存两种数值，“真”或“假”，最少需要1 bit的空间就足够了。通常让编译器自行选择空间大小。

```
logical  a
```

赋值逻辑变量空间大小的方法：

```
logical(kind=4) a ! 用4 bytes来记录true/false，Fortran 90做法
logical *4 b          ! Fortran 77已有
logical(4) b          ! Fortran 77已有
logical(kind=2) c ! 用2 bytes来记录true/false，Fortran 90做法
logical *2 d          ! Fortran 77已有
logical(2) c          ! Fortran 77已有
```

设置逻辑变量的方法：

```
a=.true. ！设置为“真值”，请注意true的前后要加上两个点
a=.false.！设置为“假值”，请注意false的前后要加上两个点
```

[ex0416.f90]

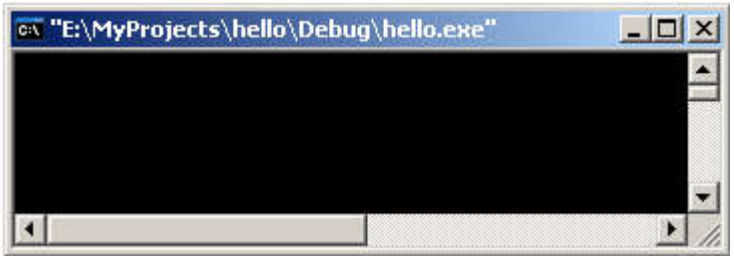
```
program ex0416
  logical a,b
  a=.true.
  b=.false.
  write(*,*) a,b
end
```

4-3 输入命令（READ）

READ：在程序进行当中实时接受用户从键盘输入数据的命令。

[ex0417.f90]

```
program ex0417
  integer a
  read(*,*) a ！由键盘读入一个整数
  write(*,*) a！写出读进变量a的内容
end
```



- read(*,*) a ！第一个星号表示输入的来源使用默认设备——键盘，第二个星号代表不指定输入格式
- read（5，*） a ！严谨一些的写法
  - 5代表输入位置是键盘，*代表不指定输入格式
- read（unit=5，fmt=*） a ！最严谨的写法
  - unit=5 代表使用输入设备-键盘
  - fmt=* 代表不赋值输入格式



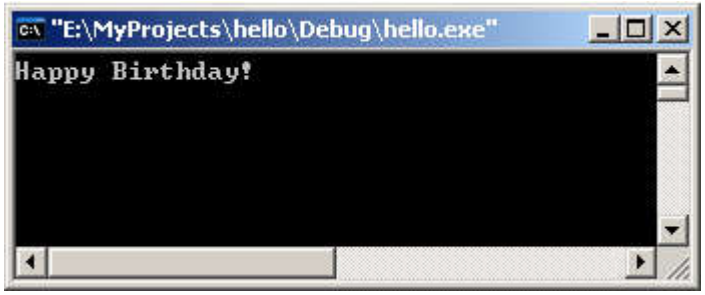
- 同一行程序代码一次读入多个数值
  - Read（*，*） a,b,c

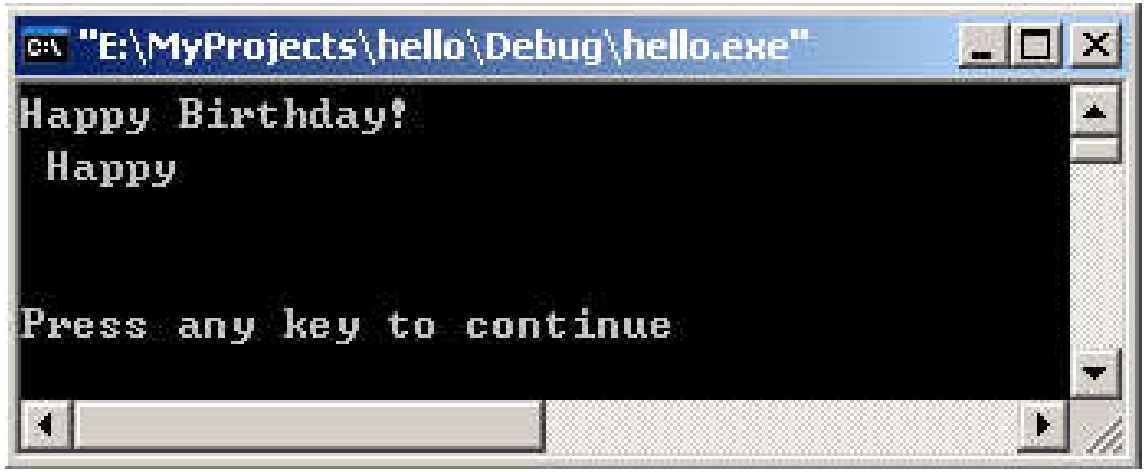
[ex0418.f90]

```
program ex0418
  real a,b,c
  read(*,*) a,b,c ！在一行中读入3个变量内容
  write(*,*) a+b+c
end
```

[ex0419.f90]

```
program ex0419
  character(len=80) a
  read(*,*) a
  write(*,*) a
end
```





4-4 格式化输入输出(FORMAT)

格式化输出的目的就是把数据经过“有计划”的版面设计显示出来

- 可以实现美观
- 某些读取数据的情况要设置恰当的输入格式

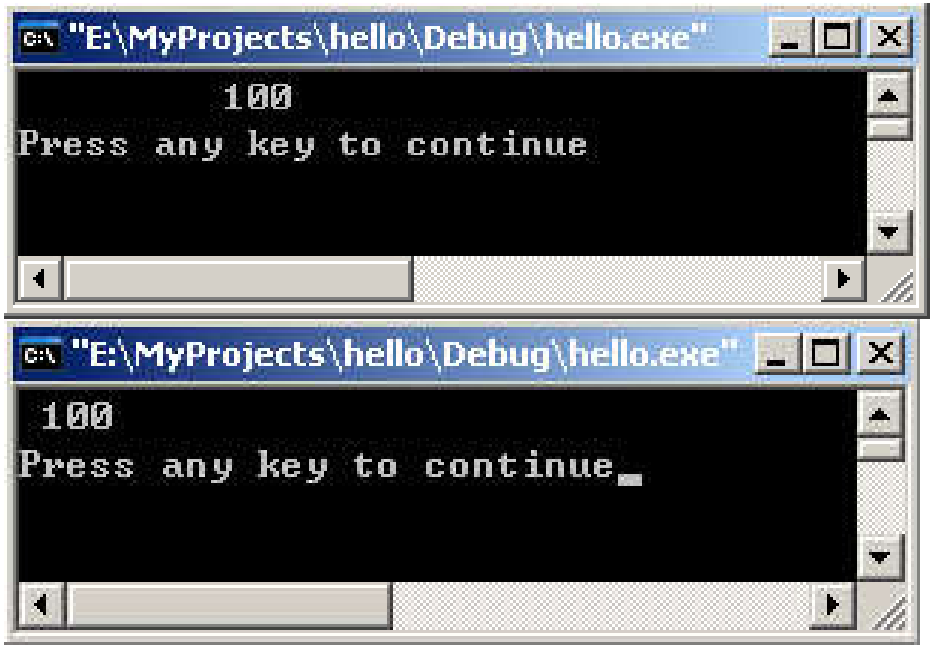


4-4-1 格式化输出概论

FORMAT命令可以用来设置输出格式

[ex0420.f90]

```
program ex0420_1
  integer a
  a=100
  write(*,100) a
End

program ex0420_2
  integer a
  a=100
  write(*,100) a  ！使用行代码100地方设定的格式来输出变量a
  100 format(I4)  ！最前面的100是行代码, 把这一行程序代码给
                  ！一个编号
end
```



Aw	以w个字符宽来输出字符串
BN	定义文本框中的空位为没有东西，在输入时才需要使用
BZ	定义文本框中的空位代表0，在输入时才需要使用
Dw.d	以w个字符宽来输出指数类型的浮点数，小数部分占d个字符宽
Ew.d[Ee]	以w个字符宽来输出指数类型的浮点数，小数部分占d个字符宽，指数部分占e个字符
ENw.d[Ee]	以指数类型来输出浮点数
ESw.d[Ee]	以指数类型来输出浮点数
Fw.d	以w个字符宽来输出浮点数，小数部分占d个字符宽
Gw.d[Ee]	以w个字符宽来输出整数，最少输出m个数字
Iw[.m]	以w个字符宽来输出整数，最少输出m个数字
Lw	以w个字符宽来输出T或F的真假值
nX	把输出的位置向右跳过n个位置
/	代表换行
:	在没有更多数据时结束输出
kP	K值控制输入输出的SCALE
Tn	输出的位置移动到本行第n列
TLn	输出的位置向左相对移动n列
TRn	输出的位置向右相对移动n列
SP	在数值为正时加上“正号”
SS	取消SP
Bw[.m]	把整数转换成二进制来输出、输出会占w个字符宽，固定输出m个数字。m值可以不给定
Ow[.m]	把整数转换成八进制来输出，输出会占w个字符宽，固定输出m个数字。m值可以不给定
Zw[.m]	把整数转换成十六进制来输出，输出会占w个字符宽，固定输出m个数字。m值可以不给定

[ex0421.f90]

```
program ex0421
  integer   a
  real      b
  complex   c
  logical   d
  character(len=20) e
  a=10
  b=12.34
  c=(1,2)
  d=.true.
  e="FORTRAN"
  write(*,"(1X,I5)")    a ！用I来格式化整数
  write(*,"(1X,F5.2)" ) b ！用F来格式化浮点数
  write(*,"(1X,F4.1,F4.1)" ) c ! complex也是浮点数
  write(*,"(1X,L3)") d ！用L来输出logical
  write(*,"(1X,A10)") e ! 用A来输出字符串
end
```

新的用法：

把输出格式的设置直接放在WRITE命令中，不再另用FORMAT描述

Write（*，“(1X，I5)”） a

输出格式可直接写在命令中

优点：1.减少程序行数

2.阅读清楚明了

3.可避免在代码中使用行号

缺点：1.输出格式复杂时编写较麻烦

2.在不同地方使用相同的输出格式时代码会重复

[ex0422.for]

```
      PROGRAM ex0422_1
      INTEGER A
      REAL    B
      COMPLEX C
      LOGICAL D
      CHARACTER*(20) E
      A=10
      B=12.34
      C=(1,2)
      D=.true.
      E="FORTRAN"
      WRITE(*,100) A ! 用I来格式化整数
      WRITE(*,200) B ! 用F来格式化浮点数
      WRITE(*,300) C ! complex也是浮点数
      WRITE(*,400) D ! 用L来输出logical
      WRITE(*,500) E ! 用A来输出字符串
100   FORMAT(1X,I5)
200   FORMAT(1X,F5.2)
300   FORMAT(1X,F4.1,F4.1)
400   FORMAT(1X,L3)
500   FORMAT(1X,A10)
      END
```

```
      PROGRAM ex0422_2
      INTEGER A
      REAL    B
      COMPLEX C
      LOGICAL D
      CHARACTER*(20) E
      A=10
      B=12.34
      C=(1,2)
      D=.true.
      E="FORTRAN"
      WRITE(*,100) A ! 用I来格式化整数
100   FORMAT(1X,I5)
      WRITE(*,200) B ! 用F来格式化浮点数
200   FORMAT(1X,F5.2)
      WRITE(*,300) C ! complex也是浮点数
300   FORMAT(1X,F4.1,F4.1)
      WRITE(*,400) D ! 用L来输出logical
400   FORMAT(1X,L3)
      WRITE(*,500) E ! 用A来输出字符串
500   FORMAT(1X,A10)
      END
```

4-4-2 详论格式化输出

在使用输出格式时，要注意“类型”是否正确。
“I、F、E、A、X”是最常用的格式

【Iw*[.m]*】:以w个字符的宽度来输出整数，至少输出m个数字。所设置的输出文本框不足时，会输出星号（*）
例：
write(*,”(I5)”)100　　□ □ 100

write(*,”(I3)”)10000　　***

write(*,”(I5.3)”)10　　□ □010

【Fw.*d*】:以w个字符文本框宽来输出浮点数，小数部分占d个字符宽，输出文本框的设置不足时会出现星号

例：

write(*,”(F9.3)”)123.45

固定使用9个字符宽来输出浮点数，小数部分固定输出3个位数

□ □ 123.450

小数不足3位部分补0，不足9个字符部分填上空白



【Ew.*d[Ee]*】:用科学计数法，以w个字符宽来输出浮点数，小数部分占d个字符宽，指数部分最少输出e个数字。

例：

```
write(*,"(E15.7)")123.45
```

使用15个字符字段，小数部分占7位

□ □ 0.1234500E+03

小数部分不足7位的部分补0，不足15个字符的部分补上空格

```
write(*, "(E9.2E3)")12.34
```

设定输出9个字符宽，小数部分占2位，指数部分输出3个数字

0.12E+002

共输出9个字符，其中有2位小数，指数部分有3位数字

【Aw】：以w个字符宽来输出字符串

例：

write(*,"(A10)")"Hello"

固定用10个字符字段来输出字符串

□ □ □ □ □ Hello

不足10个字符的部分会补上空格

write(*,"(A3)")"Hello"

固定用3个字符字段来输出字符串

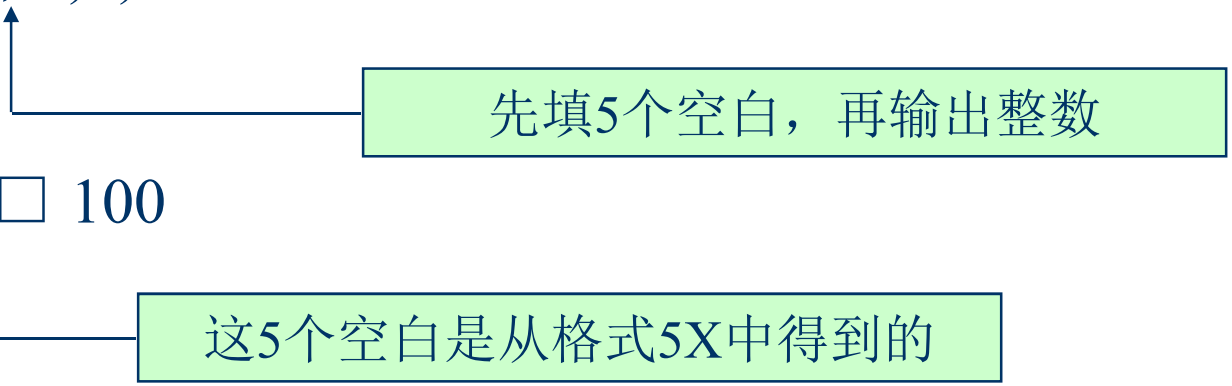
Hel

超出3个字符的部分会被略去

【nX】: 输出位置向右移动n位

例:

write(*,"(5X,I3)")100



其它格式化输出控制字符请参考P55-P57

重复地以同样的格式输出数据：

[ex0423.f90]

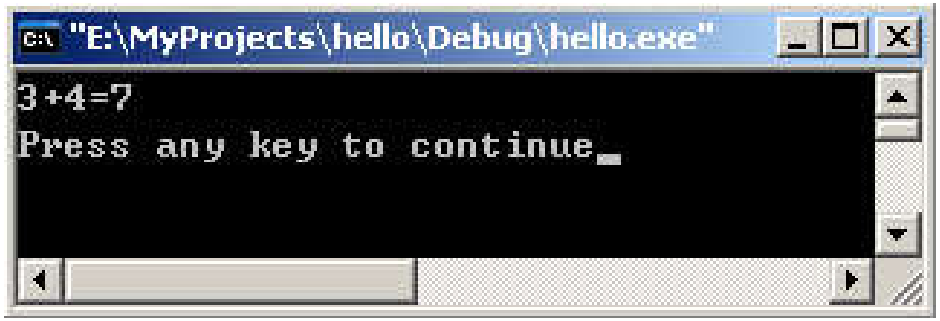
```
program ex0423
  real      a,b,c
  a=1.0
  b=2.0
  c=3.0
  write(*,"(3(1XF5.2))") a,b,c
end
```

3(1XF5.2)代表重复 (1XF5.2) 这个输出格式3次。

格式设置的字符串中，还可以放进固定要输出的字符串

[ex0424.f90]

```
program ex0424
  write(*,"('3+4=',I1)") 3+4
end
```



可以把输出格式放在字符串变量中

[ex0425.f90]

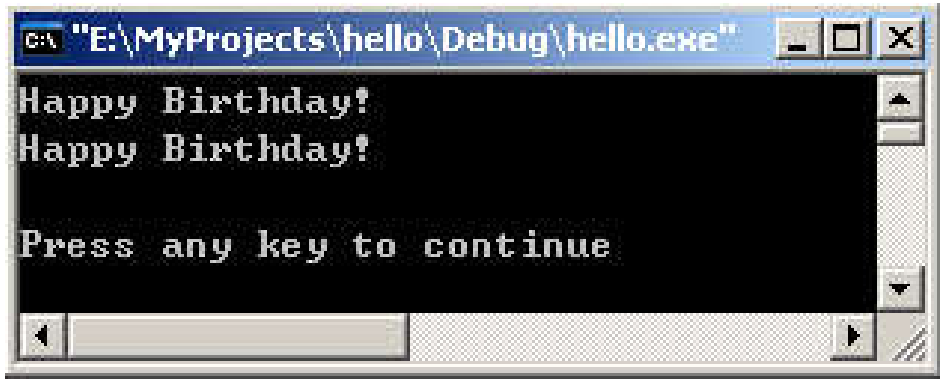
```
program ex0425
  character(len=10) fmtstring
  fmtstring = "(I2)"
  write(*,fmtstring) 3
end
```

前面提到的格式也可以使用在输入命令read中。

[ex0419.f90]

[ex0426.f90]

```
program ex0426
  character(len=80) string
  read(*,"(A80)") string
  write(*,"(A80)") string
end
```



4-5 声明的其他事项

4-5-1 变量名称的取名策略

参见　第7页

变量的名字最好是取一个有意义的英文单词，这样可以减少程序编写时出错的机会。



4-5-2 IMPLICIT命令

◆FORTRAN标准定义，变量并不一定要经过程序的声明才能使用，编译器会根据变量名称的第一个字母来自动决定这个变量的类型。

- ◆第1个字母为I、J、K、L、M、N的变量会被视为整数类型
- ◆其他字母打头会被认为是浮点数

[ex0427.f90]

```
program ex0427
  read(*,*) fa,fb
  write(*,*) fa+fb
end
```

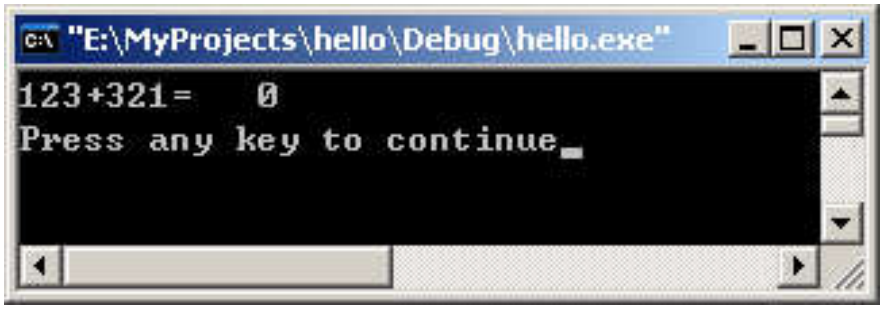
建议最好别用不经定义的方法：

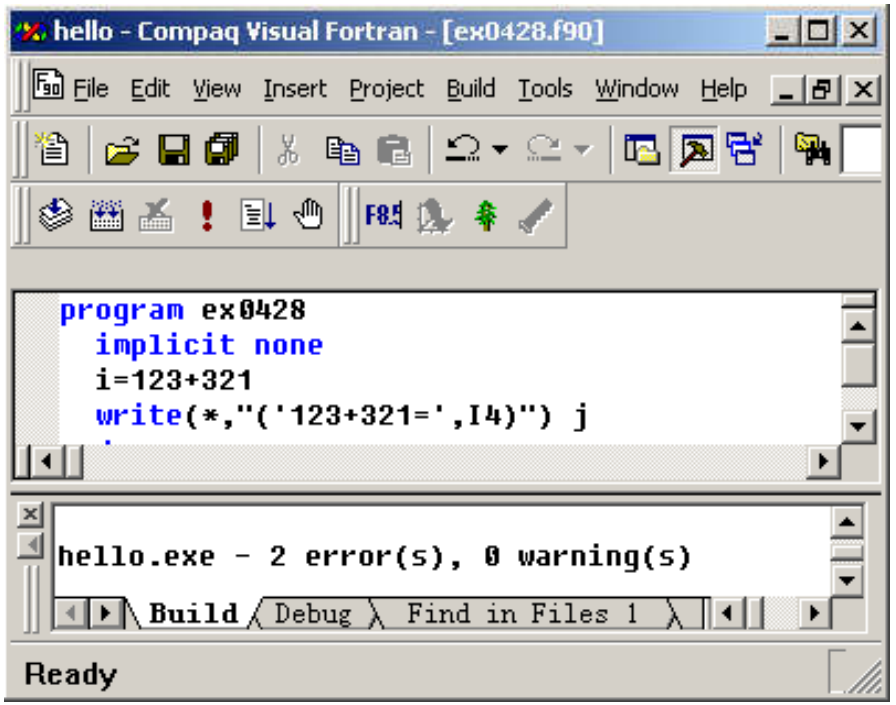
1. 先经过定义再使用变量能清楚地了解程序执行时内存的使用情况；
2. 变量不经声明就使用，写程序时很容易发生“认为错误”



[ex0428.f90]

```
program ex0421_1
  i=123+321
  write(*,"('123+321=',I4)") j
end
```






```
program ex0421_2
 implicit none
 i=123+321
  write(*,"('123+321=',I4)") j
end
```

◆IMPLICIT命令是用来设置“默认类型”，可以经过IMPLICIT描述来决定那些字母开头的变量会自动使用某种类型。

implicit integer(A,B,C) !A、B、C开头的变量都视为整形数
implicit integer(A-F,I,K) !A到F、I、K开头的变量都视为整形数
implicit integer(M-P) !M到P开头的变量都视为浮点数
implicit none ！ 关闭默认类型功能，所有变量都要事先声明

◆IMPLICIT命令必须放在PROGRAM的下一行



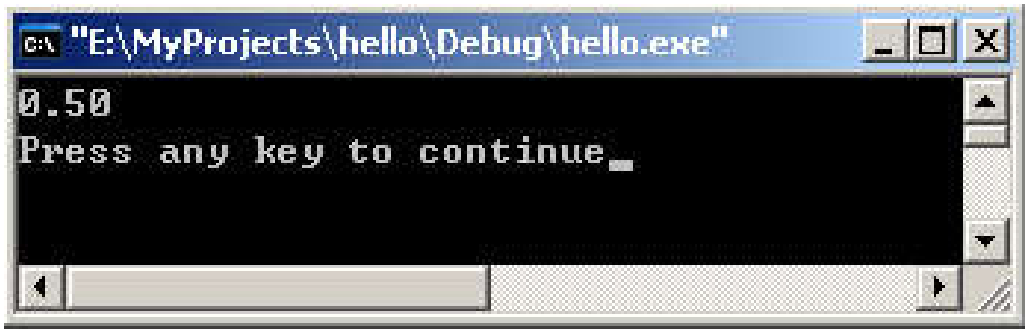
4-5-3 常数的声明方法

◆程序中有些数据是永远固定、不会改变的一常数

如：圆周率、重力加速度G值等，这些数据可以把它们声明成“常数”来处理。

[ex0429.f90]

```
program ex0429
implicit none
  real pi
  parameter(pi=3.14159)
  write(*,"(F4.2)") sin(pi/6)
end
```



常数声明的方法：

1、先声明变量类型，然后用PARAMETER命令设置数值；

```
real pi
parameter(pi=3.14159)
```

2、 PARAMETER作为形容词和变量的声明写在一起。

```
real, parameter :: pi=3.14159  !pi前面的冒号不能省略
```

声明常数的价值：

- 可以减少错误发生的机会；
- 可以增加程序执行的速度。

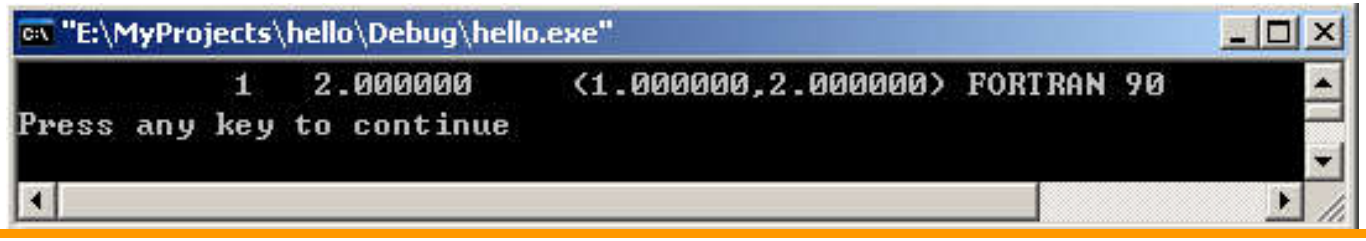


4-5-4 设置变量的初值

变量可以在声明时同时给予初值
Fortran90中直接把数值写在声明的变量后面

[ex0430.f90]

```
program ex0430
  integer :: a = 1
  real    :: b = 2
  complex :: c = (1,2)
  character(len=20) :: str = "FORTRAN 90"
  write(*,*) a,b,c,str
end
```



Fortran 77 则要使用DATA命令设置初值。在DATA命令后接上所要设置初值的变量，然后用两个斜杠包住所要设置的初值。

[ex0430.for]

```
PROGRAM ex0430
IMPLICIT NONE
INTEGER A
REAL    B
COMPLEX C
CHARACTER*(20) STR
DATA A,B,C,STR /1, 2.0, (1.0,2.0), 'FORTRAN 77'/
WRITE(*,*) A,B,C,STR
STOP
END
```

4-5-5 等价声明（EQUIVALENCE）

◆把两个以上的变量声明使用同一内存地址，就是“等价声明”。

◆使用同一内存位置的变量，只要改变其中一个变量，就会同时改变其他变量的数值

◆等价声明的方法

integer a，b

equivalence（a，b）！a，b使用同一内存空间

◆节省内存

◆精简代码



Equivalence的局限性

- 过去，计算机内存有限且昂贵，对于大型程序来说，重复使用那些为程序不同过程进行中间计算的内存，就非常普遍。
- Fortran 90之前没有动态内存分配，一块固定大小的临时存储空间就必须声明的足够大，以满足程序中所有临时计算的需要。
- 在程序的不同部分，经常用不同的名字来引用临时存储空间，但是每次使用的都是同一个物理内存。
- 为支持这样的应用，Fortran提供了一种机制，它为同一个物理地址分配两个甚至更多的名字，这种机制就是Equivalence语句。

Equivalence语句的问题

- 当先以array1为名字用某等价数组进行计算后，然后再用array2的名字重新使用这个等价数组并进行了另一个不同的计算。当访问array1中的数值时，它们有可能已经被array2的操作改变！
- 如果程序能够被其他人而不是编程者所更改，这将是一个非常大的问题，在并没有对array1赋值的情况下，数组array1中的数据就被更改了，这种错误很难被发现。
- 计算机内存已越来越便宜，而且容量也越来越大，对等价数组的需要已经急剧减少。除非有足够的理由，不要在程序中使用等价变量名。如果需要在程序中重用临时数组，使用可分配数组动态分配和释放临时空间是较好的做法。
- Equivalence语句在Fortran 90后已经不再使用。

4-5-6 声明在程序中的结构

- 声明的位置应该放在程序代码的可执行描述之前。
- 在程序代码开始出现数值计算和输入输出命令后就不能再声明变量。
- DATA也算声明的一部分

```
program main
implicit none
integer a
real b

read(*,*) a
read(*,*) b
write(*,*) a+b

end
```

从program或是implicit后面开始声明变量

声明要在程序执行指令开始之前完毕，从这一行之后就不能再出现变量声明

错误的声明结构举例：

```
program main
implicit none
integer a

read(*,*) a
real b
read(*,*) b
write(*,*) a+b
stop
end
```

声明变量要一口气在最前面完成，不能留到后面程序代码再声明

声明一定要在程序块的前面部分，这一行在编译时会出现错误，程序执行指令开始后就不能再出现声明

4-6 混合运算

◆混合运算：在算式中所进行计算的每个数字的类型不完全相同。如一个整数和一个浮点数相加。

◆进行混合运算时，最好先经过类型转换的工作，把数据类型都统一起来。

◆编译器会自动做一些类型转换的工作，但不一定能正确达到要求。

◆为避免意外，当算式变量类型不统一时，最好由程序员自行来完成数据转型的工作。



类型不统一时不能得到正确答案的例子

[ex0431.f90]

```
program ex0431
implicit none
  integer :: a=1
  integer :: b=2
  real    :: c
  c=a/b
! c=1/2=0, 虽然c是浮点数,但因为a,b
!是整数,计算a/b时会用整数去计算.
  write(*,"(F5.2)") c
end
```

通过类型转换使类型统一，得到正确结果的例子

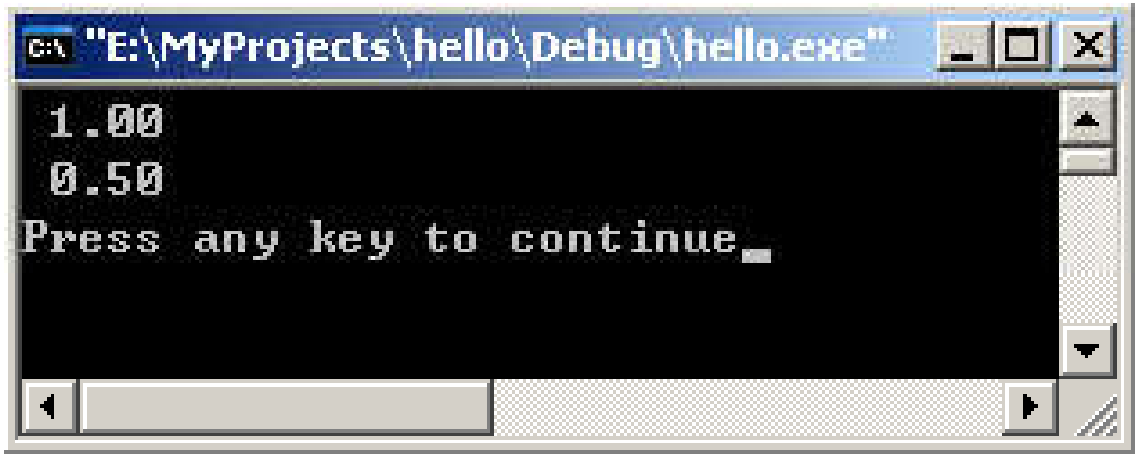
[ex0432.f90]

```
program ex0432
implicit none
  integer :: a=1
  integer :: b=2
  real    :: c
  c=real(a)/real(b) ! 经由库函数real把整数转换成浮点数
  write(*,"(F5.2)") c
end
```

编译器自动做出正确类型转换的例子

[ex0433.f90]

```
program ex0433
implicit none
  integer :: a=1
  real    :: b=2
  real    :: c
  c=a ! 整数赋值给浮点数的操作会自动转换
  write(*,"(F5.2)") c
  c=a/b ! 因为除数跟被除数类型不同, 计算
        ! 的结果会以浮点数来表示.
  write(*,"(F5.2)") c
end
```



4-7 Fortran 90的自定义数据类型

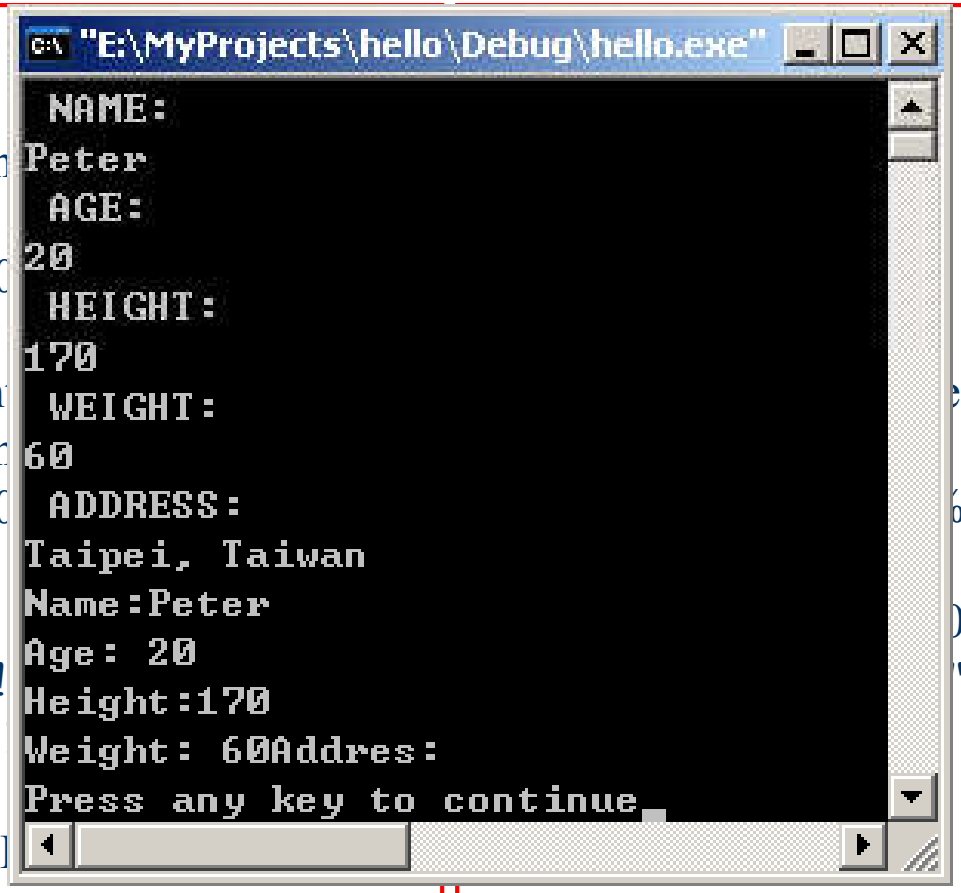
◆自定义数据类型：Fortran可以自由组合一些基本的数据类型来创造出一个更复杂类型组合的功能“TYPE”。

```
！开始创造一个叫person的数据类型
type :: person
  character(len=30) :: name    ！记录人名用
  integer  ::  age             !记录年龄用
  integer  ::  height          !记录身高用
  integer  ::  weight          !记录体重用
  character(len=80) :: address !记录地址用
end type person
！自定义数据类型结束
```

[ex0434.f90]

```
program  ex0434
implicit none
! 开始建立person
type :: person
  character(len=3(
  integer  ::  age
  integer  ::  heigh
  integer  ::  weigh
  character(len=8(
end type person

type(person) :: a !

write(*,*) "NAM
read(*,*)  a%name
write(*,*) “AGE:”
```

```
ess

age,a%height,&

)/,"Age:",I3/, &
",I3,"Addres:",A50)
```



每个类型为person的变量中，都有name、age、height、weight、address这几个元素可以使用，使用时要加上一个百分号来取用它们。

read(*,*)a%name

这表示要使用变量a中name这个元素。变量和元素间要以“%”来区隔。Visual Fortran还可以用点号“.”来区隔。

直接设置自定义类型的所有元素：

A=person（“peter”，20，170，60，“Taipei,Taiwan”）

name age height weight address

4-8 KIND的使用

◆KIND描述如果搭配一些Fortran90的库函数，可以增加程序代码的“跨平台”能力

◆声明变量所占用的内存大小，主要是根据在计算时所要使用到的有效位数以及值域范围。

在PC编译器中，各类变量所保存的值域范围：

integer(kind=1) -128 ~ +127

integer(kind=2) -32,768 ~ +32,767

integer(kind=4) -2,147,483,648 ~ +2,147,483,647

real(kind=4) $\pm 1.18*10^{-38} \sim \pm 3.40*10^{38}$

real(kind=8) $\pm 2.23*10^{-308} \sim \pm 1.79*10^{308}$



判断所要记录的数值值域范围所需kind值的库函数：

SELECTED_INT_KIND(n)

返回想记录n位整数时，所应声明的kind值。返回-1时，表示无法提供想要的值域范围。

SELECTED_REAL_KIND(n, e)

返回想要能够记录具有n位有效位数、指数达到e位的浮点数所需的kind值。返回-1表示无法满足所要求的有效位数、返回-2表示无法满足所要求的指数范围、返回-3表示两者都无法满足。

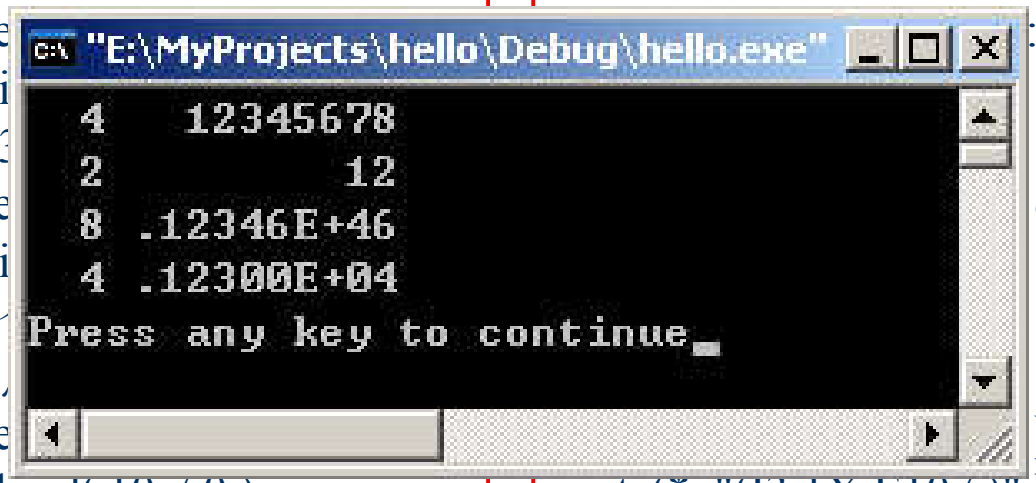


声明变量时赋值变量使用的KIND值

[ex0435.f90]

```
program ex0435
implicit none
! 判断可以记录9个位数的整数kind值
integer, paramete
&selected_int_ki
! 判断可以记录
integer, paramete
&selected_int_ki
! 判断可以有10
! 记录到50的浮
integer, paramete
&selected_real_kind( 10, 50 )
! 判断可以有3个有效位数, 指数可以
! 记录到3的浮点数kind值
```

```
integer, parameter :: short_real=&
&selected_real_kind( 3, 3 )
integer(kind=long_int)  :: a = 12345678
                        : b = 12
                        c =&

                        d = 1230

                         long_int,  a
                         short_int,  b
                        ) long_real,  c
write(*, "(I3,1X,E10.5)" ) short_real, d
stop
end
```



设置数值时在数字后加下划线赋值数字使用的KIND值

[ex0436.f90]

```
program ex0436
implicit none
real(kind=4) :: a
real(kind=8) :: b
a=1.0_4 ! 确定1.0这个数字是使用单精度
b=1.0_8 ! 确定1.0这个数字是使用双精度
write(*,*) a,b
stop
end
```

程序举例1：温度转换

- 设计一个Fortran程序，读取输入的华氏温度，转换为绝对温度并输出结果。

解决方案：

物理书上可以找到两个温度转换的关系：

T(K)=[5/9*T(°F)-32.0]+273.15

物理书上也给出了两个温度样本：

水的沸点：212°F，373.15K

干冰的升华：-110 °F，194.26K

程序功能：

提示用户输入华氏温度、读取温度、计算开氏温度、打印结果，停止。

程序举例1：温度转换

```
PROGRAM temp_conversion

! Purpose:
!   To convert an input temperature from degrees
!    Fahrenheit to an output temperature in kelvins.
!
! Record of revisions:
!     Date       Programmer         Description of change
!     ====       ==========      ====================
!   11/03/06 -- S. J. Chapman       Original code
!
```

程序举例1：温度转换

```
IMPLICIT NONE       ! Force explicit declaration of variables

! Data dictionary: declare variable types, definitions, & units
REAL :: temp_f      ! Temperature in degrees Fahrenheit
REAL :: temp_k      ! Temperature in kelvins

! Prompt the user for the input temperature.
WRITE (*,*) 'Enter the temperature in degrees Fahrenheit: '
READ  (*,*) temp_f

! Convert to kelvins.
temp_k = (5. / 9.) * (temp_f - 32.) + 273.15

! Write out the result.
WRITE (*,*) temp_f, ' degrees Fahrenheit = ', temp_k, ' kelvins'

! Finish up.
END PROGRAM temp_conversion
```



程序举例2：电子工程

- 在正弦交流电路中，电压V向负载Z供给电量，计算实际功率、无功功率和有效功率。

电路原理：负载中的电流I、实际功率P、无功功率Q、有效功率S和功率因子PF之间的计算公式为：

$$I=V/Z$$

$$P=VI\cos\theta$$

$$Q=VI\sin\theta$$

$$S=VI$$

$$PF=\cos\theta$$



程序举例2：电子工程

解决方案：

程序中需要读取电源的电压V、阻抗的幅值Z和角度θ。输入电压单位为伏特，阻抗Z的幅值为欧姆，阻抗的角度为度(°)。按照公式计算所需参量。

程序要求：

提示用户输入电源值，单位伏特；读入电压；提示用户输入阻抗的幅值和角度，单位欧姆、度；读入阻抗的幅值和角度；分别计算I、P、Q、S和PF。

程序举例2：电子工程

```
PROGRAM power
!
!  Purpose:
!    To calculate the current, real, reactive, and apparent power,
!    and the power factor supplied to a load.
!
!  Record of revisions:
!      Date       Programmer         Description of change
!      ====       ==========         =====================
!    11/03/06    S. J. Chapman       Original code
!

IMPLICIT NONE
```



程序举例2：电子工程

```
    ! Data dictionary: declare constants
REAL,PARAMETER :: DEG_2_RAD = 0.01745329
    ! Deg to radians factor

    ! Data dictionary: declare variable types, definitions, & units
REAL :: amps          ! Current in the load (A)
REAL :: p             ! Real power of load (W)
REAL :: pf            ! Power factor of load (no units)
REAL :: q             ! Reactive power of the load (VAR)
REAL :: s             ! Apparent power of the load (VA)
REAL :: theta         ! Impedance angle of the load (deg)
REAL :: volts         ! Rms voltage of the power source (V)
REAL :: z             ! Magnitude of the load impedance (ohms)
```

程序举例2：电子工程

```
! Prompt the user for the rms voltage.
WRITE (*,*) 'Enter the rms voltage of the source: '
READ  (*,*) volts

! Prompt the user for the magnitude and angle of the impedance.
WRITE (*,*) 'Enter the magnitude and angle of the impedance '
WRITE (*,*) 'in ohms and degrees: '
READ  (*,*) z, theta

! Perform calculations
amps = volts / z                          ! Rms current
p = volts * amps * cos (theta * DEG_2_RAD) ! Real power
q = volts * amps * sin (theta * DEG_2_RAD) ! Reactive power
s = volts * amps                          ! Apparent power
pf = cos ( theta * DEG_2_RAD)             ! Power factor
```

程序举例2：电子工程

```
! Write out the results.
WRITE (*,*) 'Voltage       = ', volts, ' volts'
WRITE (*,*) 'Impedance  = ', z, ' ohms at ', theta,' degrees'
WRITE (*,*) 'Current        = ', amps, ' amps'
WRITE (*,*) 'Real Power  = ', p, ' watts'
WRITE (*,*) 'Reactive Power = ', q, ' VAR'
WRITE (*,*) 'Apparent Power = ', s, ' VA'
WRITE (*,*) 'Power Factor     = ', pf

! Finish up.
END PROGRAM power
```

程序举例3：^{14}C的测定

- 放射性同位素经过一段时间后会自然衰变到另外一个元素，衰变是按照指数过程进行的。如果时间t=0时，放射性物质的初始量为Q_0，那么未来任何时刻的放射性物质的量为：

$$Q(t)=Q_0\exp(-\lambda t)$$

- λ为放射性衰变常数。如果知道放射性物质的初始量Q_0和当前量Q，那么就可以知道衰变经历的时间：

$$t=(1/\lambda)\log(Q/Q_0)$$

- 这个式子在科学界使用了很多年。例如考古学家用基于^{14}C的放射性物质可以判断曾经活着的东西死去了多长时间。



程序举例3：^{14}C的测定

- 编写程序：

  读取^{14}C在样本中的剩余百分比，计算样本的年龄，打印计算结果，并给出计量单位。已知^{14}C的衰变常数λ=0.00012097/年。

- 解决方案：

  ① 提示用户输入在样本中^{14}C的剩余百分比。

  ② 读取百分比

  ③ 把百分比转换为因子Q/Q_0。

  ④ 计算样本死去的时间。

  ⑤ 输出计算结果，结束。



程序举例3：^{14}C的测定

```
PROGRAM c14_date
!
!  Purpose:
!    To calculate the age of an organic sample from the percentage
!    of the original carbon 14 remaining in the sample.
!
!  Record of revisions:
!      Date       Programmer         Description of change
!      ====       ==========         =====================
!    11/03/06    S. J. Chapman       Original code
!
IMPLICIT NONE
```



程序举例3：^{14}C的测定

```
! Data dictionary:   declare decay constant of carbon 14,
!  in units of 1/years.
REAL,PARAMETER :: LAMDA = 0.00012097

! Data dictionary: variable types, definitions, & units
REAL :: age      ! The age of the sample (years)
REAL :: percent   ! The percentage of carbon 14 remaining
                  ! at the time of  the measurement (%)
REAL :: ratio
! The ratio of the carbon 14 remaining at the time of the
!  measurement to the original amount of  carbon 14
!  (no units)
```

程序举例3：^{14}C的测定

```
! Prompt the user for the percentage of C-14 remaining.
WRITE (*,*) 'Enter the percentage of carbon 14 remaining:'
READ  (*,*) percent

! Echo the user's input value.
WRITE (*,*) 'The remaining carbon 14 = ', percent, ' %.'

! Perform calculations
ratio = percent / 100.          ! Convert to fractional ratio
age = (-1.0 / LAMDA) * log(ratio)  ! Get age in years

! Tell the user about the age of the sample.
WRITE (*,*) 'The age of the sample is  ', age, ' years.'

! Finish up.
END PROGRAM c14_date
```



Fortran程序的调试

- 无论编写程序的大小是多少，第一时间，一般都难于顺利执行！
- 程序的错误就是所谓的bug，定位和去除bug的过程就是调试或者说debugging。
- 错误一般分三类：
  1. 语法错误：syntax error，这类错误在编译时可以被编译器检测出来。
  2. 运行错误：run-time error，这些错误导致程序执行时异常中断。
  3. 逻辑错误：logical error，运行时产生的结果错误。

作业

P70

1、2、3、4、5

6. 编写程序，计算高度h分别为1m、10m、100m的球体自由落体撞击地球时的速度。计算时不考虑空气阻力的作用。
7. 编写程序，计算不同长度钟摆的振荡周期。计算时不考虑空气阻力的作用。
8. 编写程序，计算笛卡尔坐标系中点(-1,1)到点(6,2)的距离。
9. 共振电路：由R、L、C串联组成的RLC电路有其特有的共振频率，编写程序计算R=100Ω，L=0.1mH，C=0.25nF组成的共振电路的共振频率。