Fortran95 程序设计

彭国伦 编著

第9章 文件

9-0 文 件

- 目前已经编写的程序中，输入/输出的数据量相对很少。从键盘输入数据、由终端输出数据，对于小型的数据集合来说，是可以接受的，但是当使用大量的数据时，标准的键盘输入屏幕输出的方法就不可用。
- 解决这个局限性的方法就是使用文件。 Fortran可以在程序中使用文件中的数据。一个文件由许多相互关联的数据行组成，可以作为一个整体被存取。Fortran可以一次一个记录地从一个文件中读取信息或者向文件中写入信息。
- 写入文件中的信息可以长期保存下来。



- 计算机中的文件可以保存在各种类型的设备上面，相对于计算机的RAM主存储器而言，这些设备都称为辅助存储器。
- 辅助存储器的读写速度比计算机的主存储区要慢很多，但仍能实现相对快速的数据存取。
- 辅助存储器包括硬磁盘、软磁盘、USB存储器、CD或DVD以及磁带。
- 计算机的早期时代，磁带是最常见的辅助存储设备。计算机必须按照从磁带的开始到结束的顺序读取磁带中的数据。由于历史原因，即使现在使用能够直接访问的设备，Fortran中的默认访问方式仍然为顺序存取。



- 为了在Fortran程序内使用文件，需要选择一些读写文件的方法。
- Fortran读取辅助存储器上面的文件的方法，使用的是输入/输出通道机制，即通常所说的I/O通道机制。
- 常见的文件读取命令：
  - Open：将I/O通道号分配给文件
  - Close：将I/O通道号与文件分离
  - Read：通过I/O通道号读取文件内数据
  - Write：通过I/O通道号将数据写入文件
  - Rewind、Backspace：用于改变在打开的文件中的读写位置。



9-1 文件读取的概念

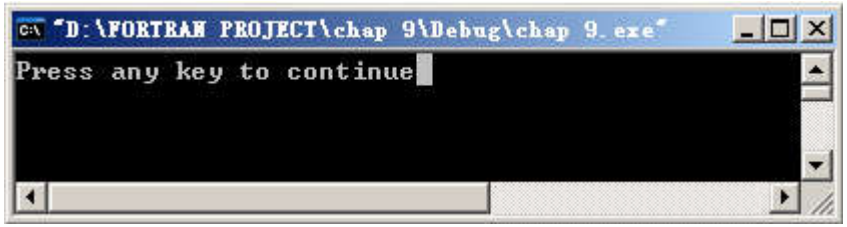
- Fortran语言中，读取文件的操作可以有“顺序读取”及“直接读取”两种方法。
  - **顺序读取**：读写一个文件时，只能从头开始，顺序读写文件中的数据，不能跳跃。
  - **直接读取**：读写文件时，可以任意跳跃到文件的任何一个位置来读写。
- 文件的“保存格式”有两种方法，“文本文件”和“二进制文件”
  - **文本文件**：所有的数据都使用ASCII码字符来保存。文本文件可以很直观的使用文本编辑器来查看或修改。
  - **二进制文件**：把数据在内存中的保存内容直接写入文件中。要使用特别的工具程序来阅读。
    - 读取速度比较快，不需要转换；
    - 较省空间，和变量所占用的内存空间相同。

存储长整型1234567890文件时，文本格式需要10个字节的空间；二进制格式只要4个字节。

9-2-1 打开文件--OPEN

使用open命令打开文件之后，就可以对文件进行输入输出操作。

[EX0901.F90]




```
program ex0901
implicit none
open(unit=10, file='hello.txt')
        ! 打开hello.txt文件,
        ! unit指定设备通道号，file指定文件名称。
write(10, *) "hello"
        !在通道号为10的设备中（文件）中输出hello
stop
end program ex0901
```

Open的最简格式

open(unit=N, file=string)

File是关键字，后面的string是字符串，用来设定文件的名字。

Unit是关键字，N为整数，用来给后面的文件指定一个通道号。

- 文件打开以后，可以使用READ、WRITE命令对文件进行读写操作，操作时的命令跟原来相同，仅仅把输入、输出的通道号写成open中定义的通道号（N）即可。
- 不建议把通道号写为5和6，一般这两个通道号对应于计算机的标准输入输出设备。

Open格式详解

OPEN(**UNIT=number**, **FILE='filename'**, &
FORM='...', **STATUS='...'**, ACCESS='...', &
RECL=length, *ERR=label,* **IOSTAT=iostat**, &
BLANK='...', POSITION='...', **ACTION=action**,&
PAD='...', DELIM='... ')

[UNIT=]number： **I/O通道号**，number必须是一个正整数，可以使用变量或是常量来赋值。通道号最好避开5, 6。因为6是默认的输出位置，也就是屏幕。5则是默认的输入位置，键盘。

FILE='filename'： 指定要打开的文件名称，文件名要符合系统规定。windows下不区分大小写，unix下则会区分大小写，最好不要使用中文文件名。

Open格式详解

STATUS='NEW' or 'OLD' or 'REPLACE'
or 'SCRATCH' or 'UNKNOWN'

用来说明打开一个新的文件或已经存在的旧文件。

STATUS='NEW'，打开一个原本不存在的新文件

STATUS='OLD'， 打开一个原来已经存在的文件

STATUS='REPLACE'，若文件已经存在则重新创建一次，原来的内容消失；若不存在则会创建新文件。

STATUS='SCRATCH'，表示要打开一个暂存文件，这个时候不需要指定文件名称。暂存文件会在程序结束后自动删除。

STATUS='UNKNOWN' 由各编译器自定义。通常会同REPLACE的效果。在程序中应该避免使用这个状态。

默认情况下为“读写”方式，与UNKNOWN相同。

Open格式详解

POSITION='ASIS' or 'REWIND' or 'APPEND'

设置文件的读写位置：

'REWIND'：文件打开时的读取位置移到文件开头。

'APPEND'：文件打开时的读取位置移到文件的结尾。

'ASIS'：文件打开时的读取位置不指定，与处理机相关默认值为ASIS，通常就是在文件的开头。

ACTION='READ' or 'WRITE' or 'READWRITE'

设置文件的读写权限：

'READWRITE'：被打开的文件可以读写，默认值。

'READ'：表示所打开的文件只能用来读取数据。

'WRITE'：表示所打开的文件只能用来写入数据。

Open格式详解

FORM='FORMATTED' OR 'UNFORMATTED'

FORM字段只有两个值可以设置：

FORM='FORMATTED' “文本文件”格式来保存

FORM='UNFORMATTED' “二进制文件”格式保存

这一栏不给定时候的**默认值是: FORM='FORMATTED'**

ACCESS='SEQUENTIAL' or 'DIRECT'

设置读写文件的方法：

ACCESS='SEQUENTIAL'， “顺序读取文件”。

ACCESS='DIRET'， “直接读取文件”。

不赋值时候，**默认为： ACCESS='SEQUENTIAL'**。

Open格式详解

IOSTAT=var

这个字段会设置一个整数值给后面的整型变量，用来说明文件打开状态，数值大小与处理机相关：

var>0 表示读取操作错误

var=0 表示读取操作正常——与处理机无关

var<0 表示文件终了

RECL=length

适用于“直接读取文件”，RECL＝length的length值是用来设置文件中每一个读取单元的分区长度。

length的单位与处理机相关。

Open格式详解

PAD='YES' or 'NO'

PAD='YES'，在格式化输入时，最前面的不足字段会自动以空格填满，默认值是PAD='YES'。

PAD='NO'，在格式化输入时，输入数据行必须要和格式描述符指明的记录一样长，否则会发生错误。

DELIM='APOSTEROPHE' or 'QUOTE' or 'NONE'

DELIM='NONE' 输出的字符串内容没有定界符。

DELIM='QUOTE' 输出字符串内容的定界符为双引号

DELIM='APOSTEROPHE' 字符串内容定界符为单引号

默认值为 "NONE"。

9-2-2 关闭文件--CLOSE

CLOSE([UNIT=]number, STATUS=string, &

ERR=errlabel, **IOSTAT=int_var**)

一旦不再需要文件时，就应该使用Close将其和I/O通道的连接断开。Close语句执行后，该I/O通道又可用于其它open语句。

- [UNIT=]number：同open
- STAT='KEEP'：文件关闭后，在存储设备中保留这个文件。默认状态。
- STAT='DELETE'：文件关闭后，从存储设备中删除文件。
- Err=errlabel，不再使用，由IOSTAT接替。
- IOSTAT：操作结束后的I/O状态，与处理机相关，0=成功，正数=失败。



9-2-3 文件的读写--READ、WRITE

WRITE/READ ([UNIT=]number, [FMT=]format, &
NML=namelist, REC=record, **IOSTAT=stat**,&
ERR=errlabel, END=endlabel,&
ADVANCE=advance, SIZE=size)

- [UNIT=]number 指定read/write所使用的I/O通道号。
- [FMT=]format 指定输入输出格式的使用，其中有：
  - Statement_label：Format语句的标号。
  - Char_expr：包含格式化信息的字符串。
  - *：表控I/O。

  如果FMT=是read语句中的第二个子句，并且第一个是缩写的通道号形式，那么格式化子句必须缩写为命名的语句代号。

Read(10, 100) data_list

READ、WRITE命令参数详解

- IOSTAT=stat 设置一个数值给在它后面的变量，用来说明文件的读写状态。

  stat>0 表示读写操作发生错误。

  stat=0 表示读写操作正常。

  stat<0 表示文件终了(-1或-2)。

```
Open(unit=8,file='test.txt',status='old')
Nvals=0
Do
   Read(8,100,iostat=istat) temp
   If (istat<0) exit
   Nvals=nvals+1
   Array(nvals)=temp
End do
```



READ、WRITE命令参数详解

- NML=namelist 指定读写某个I/O有名列表的内容。

  REC=record 设置直接读取文件中的记录数。仅对直接读取文件有效。

  ERR=errlabel 读写过程中发生错误时，转移到某个行代码继续执行程序。不再使用。

  END=endlabel 读到文件末尾时，转移到某个行代码来继续执行程序。不用于write语句，也不再使用。



READ、WRITE命令参数详解

以下是fortran 90添加功能

ADVANCE='YES' or 'NO'

此子句指明在读写结束时是否放弃当前缓冲区的内容。当为YES时，完成读写操作后，当前缓冲区的内容将被丢弃。否则，当前缓冲区的内容将得到保存，并用于下一条读写语句，默认值为YES。此子句仅仅对顺序文件有效。

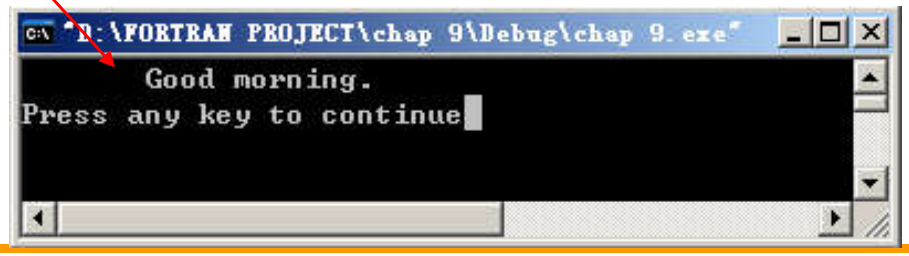
！使用这个字段时候一定要设置输出入格式，**在屏幕输出时可以使用这个设置来控制write命令是否会自动换行。**

SIZE=count

在ADVANCE='NO'时，才可以使用这个字段。它会把输入缓冲区的字符数目设置给后面的整型变量。不能用于write语句中。

[EX0902.F90]

```
program ex0902
  implicit none
  character(len=20) :: string

  open(10, file="test.txt")
  write(10,"(A20)") "Good morning." ! 写到文件中
  rewind(10)
  read(10,"(A20)") string ! 从文件中读出来
  write(*,"(A20)") string ! 写到屏幕上
  Close(10)
  stop
end
```



9-2-4 文件状态查询--INQUIRE

在使用open打开文件的前后，可以通过inquire命令来查询文件的状态，命令中的各个参数与open参数类似。

EX0903.F90


```
program ex0903
  implicit none
  character(len=20) :: filename = "ex0903.f90"
  logical alive
  inquire(file=filename, exist=alive)
  if ( alive ) then
    write(*,*) filename," exist."
  else
    write(*,*) filename," doesn't exist."
  end if
  stop
end
```

inquire使用方法详解

```
INQUIRE (UNIT=number, FILE=filename, IOSTAT=stat, &
         EXIST=exist, OPENED=opened, &
         NUMBER=number, NAMED=named, &
         ACCESS=access, SEQUENTIAL=sequential,&
         DIRECT=direct, FORM=form,&
         FORMATTED=formatted, ERR=label, &
         UNFORMATTED=unformatted, RECL=recl)
```

UNIT=number I/O通道

FILE=filename 文件名

IOSTAT=stat 查询文件读取情况，设置一个整数给后面变量：

stat>0 文件读取操作错误

stat=0 文件读取操作正常

stat<0 文件终了

EXIST=exist 检查文件是否存在，返回值为逻辑值。

OPENED=opened 检查文件是否已经被打开，返回逻辑值。

NUMBER=number 被打开文件的通道号。

NAMED=named 查询文件是否有名字，也就是检查文件是否为临时保存的文件，返回值为逻辑值。

ACCESS=access 检查文件的读取顺序，返回一个字符串，可以是：

‘SEQUENTIAL’ 代表文件使用顺序读取方式

‘DIRECT’ 代表文件使用直接读取方式

‘UNDEFINED’ 代表没有定义

ERR=errlabel 发生错误时转移到指定的代码行继续执行。不再使用。

SEQUENTIAL=sequential 查看文件是否顺序读取，返回一个字符串，字符串的意义：

‘YES’：文件是顺序读取的。

‘NO’：文件不是顺序读取的。

‘UNKNOWN’：不知道方式。

DIRECT＝direct 查看文件是否直接读取，返回一个字符串，字符串的意义：

‘YES’：文件是直接读取文件

‘NO’： 文件是非直接读取文件

‘UNKNOWN’：不知道方式

FORM=form 查看文件的保存格式，返回字符串，意义为：

‘FORMATTED’ 打开的是文本文件

‘UNFORMATTED’ 打开的是二进制文件

‘UNDEFINED’ 没有定义

FORMATTED=fmt 查看文件格式是否为文本格式的文件，
返回字符串，可以是：
‘YES’：本文件是文本文件
‘NO’： 本文件非文本文件
‘UNDEFINED’： 无法判断
UNFORMATTED=fmt 查看文件是否为二进制格式的文件，
返回字符串，可以是：
‘YES’：本文件是二进制文件
‘NO’： 本文件非二进制文件
‘UNKNOWN’： 无法判断
RECL=length 直接读取文件中的记录长度。
NEXTREC=nr 返回下一次文件读写的位置。
BLANK=blank 指定数字域中的空格被视为空还是零。

以下是fortran 90的添加功能：

POSITION=position 文件首次打开时，指定文件指针的位置值。position字段所给定的字符串值可能是：
'REWIND'、'APPEND'、'ASIS'或 'UNDEFINED'

ACTION=action 返回文件打开时ACTION字段所赋值的字符串。
'READ'、'WRITE'、'READWRITE'。

READ=read 检查文件是否为只读文件：
'YES'、'NO'或者'UNKNOWN'。

WRITE=write 检查文件是否为只写文件：
'YES'、'NO'或者 'UNKNOWN'。

READWRITE=readwrite 检查文件是否可同时读写：
'YES'、'NO'或者'UNKNOWN'。

DELIM=delim 指定表控和有名I/O列表中字符串定界符类型：
‘APOSTROPHE’、‘QUOTE’、‘NONE’或
‘UNDEFINED’。

PAD=pad 指定输入行是否需要使用空格填充。此值通常为
‘YES’除非使用PAD=‘NO’显式打开文件。

IOLENGTH=int_var：返回未格式化记录的长度，和处理机单元相关。此子句特别用于第三种INQUIRE语句。

Inquire语句的第三种形式为：

inquire(IOLENGTH=int_var) output-list

其中，int_var是整型变量，output_list是变量、常量和表达式的列表，此语句的目的是返回包含在输出列表中实体的未格式化记录的长度，该长度使用与处理机相关的单元来度量。



例题：防止输出文件覆盖已有数据

```
1.  PROGRAM open_file
2.  !
3.  !  Purpose:
4.  !    To illustrate the process of checking before overwriting an
5.  !    output file.
6.  !
7.  IMPLICIT NONE

8.  ! Data dictionary: declare variable types & definitions
9.  CHARACTER(len=20) :: file_name          ! File name
10. CHARACTER :: yn                        ! Yes / No flag
11. LOGICAL :: lexist                      ! True if file exists
12. LOGICAL :: lopen = .FALSE.             ! True if file is opened
```



```
13. ! Do until file is open
14.  openfile: DO

15.    ! Get output file name.
16.    WRITE (*,*) 'Enter output file name: '
17.    READ (*,'(A)') file_name
18.
19.    ! Does file already exist?
20.    INQUIRE ( FILE=file_name, EXIST=lexist )
21.    exists: IF ( .NOT. lexist ) THEN
22.      ! It's OK, the file didn't already exist.  Open file.
23.     OPEN (UNIT=9, FILE=file_name, &
24.                STATUS='NEW', ACTION='WRITE')
25.      lopen = .TRUE.
```

获取文件名字

查询文件状态

如果文件不存在

那么就打开文件

如果文件存在

```
25.  ELSE
26.     ! File exists.  Should we replace it?
27.     WRITE (*,*) 'Output file exists.  Overwrite it? (Y/N) '
28.     READ (*,'(A)') yn
29.     CALL ucase ( yn )              ! Shift to upper case

30.     replace: IF ( yn == 'Y' ) THEN
31.       ! It's OK.  Open file.
32.       OPEN (UNIT=9, FILE=file_name, STATUS='REPLACE', ACTION='WRITE')
33.       lopen = .TRUE.
34.     END IF replace

35.   END IF exists
36.   IF ( lopen ) EXIT
37.  END DO openfile

38.  ! Now write output data, and close and save file.
39.  WRITE (9,*) 'This is the output file!'
40.  CLOSE (9,STATUS='KEEP')

41.  END PROGRAM open_file
```

提醒用户“文件已经存在，是否覆盖，并得到确认

确认覆盖，以替换的方式重新打开文件

不论建新文件还是替换旧文件，此时都退出

如果不想覆盖原来文件，则回到开始，重新输入一个新的文件名，然后重复上面过程

在新文件中写入字符串，关闭并显式保存文件

```
SUBROUTINE ucase ( string )
IMPLICIT NONE

CHARACTER(len=*), INTENT(INOUT) :: string

INTEGER :: I              ! Loop index
INTEGER :: length           ! Length of input string

! Get length of string
length = LEN ( string )

! Now shift lower case letters to upper case.
DO I = 1, length
   IF ( ( string(I:I) >= 'a' ) .AND.  &
        ( string(I:I) <= 'z' ) ) THEN
      string(I:I) = CHAR ( ICHAR ( string(I:I) ) - 32 )
   END IF
END DO

END SUBROUTINE
```



9-2-5 文件定位语句

Fortran中有两条定位语句：rewind和backspace。Rewind语句将文件指针定位在文件的最开始，以便下一条read语句读取文件的第一行。Backspace语句将文件指针回退一行。这两个语句仅对顺序文件有效。

REWIND(contol_list)

BACKSPACE(control_list)

其中control_list由一个或多个由逗号隔开的子句构成。文件定位语句中出现的子句如下：

[unit=]int_expr：要操作的I/O通道

IOSTAT=int_var：操作后的I/O状态，0=成功

ERR=statement_lable：如果发生错误，转向当前作用域内的语句标号。现代Fortran中已经不再使用。

9-2-6 EndFile语句

- EndFile语句：顺序文件的当前位置写入一个文件结束记录，然后定位在文件的该记录之后。
- 在对文件使用EndFile语句之后，除非执行BackSpace或Rewind语句，否则read或write语句都不能继续对该文件进行操作。语句格式为：

  ENDFILE(control_list)

- 子句简要格式如下：

  [unit=]int_expr：要操作的I/O通道

  IOSTAT=int_var：操作后的I/O状态，0=成功

  ERR=statement_lable：如果发生错误，转向当前作用域内的语句标号。现代Fortran中已经不再使用。

删除文件程序实例

[ex0904.F90]

```
program ex0904
  implicit none
  integer, parameter :: fileid = 10
  logical alive
  character(len=20)  :: filename

  write(*,*) "filename:"
  read(*,"(A20)") filename

  inquire(file=filename, exist=alive)
  if( alive ) then
    open( unit=fileid, file=filename )
    close(fileid, status="DELETE")
  else
    write(*,*) TRIM(filename),&
    " doesn't exist."
  end if

  stop
end
```



9-3 有名I/O列表

- 有名I/O列表是输出一个固定变量和数值列表的好方法，也是读入固定变量名和数值列表的好方法。
- 有名列表经常是作为一组来读写的变量名列表。

  NAMELIST /nl_group_name/ var1 [,var2,…]

  其中，nl_group_name是有名列表的名字，var1等式列表中的变量。
- NAMELIST是说明语句，必须出现在程序的第一条可执行语句之前。如果有多条具有同样名字的NAMELIST语句，那么所有语句中的变量就会被连接起来，如同在一条大型语句中一样。
- NAMELIST I/O语句与格式化I/O语句类似，区别在于仅仅用NML=子句替换FMT=子句。



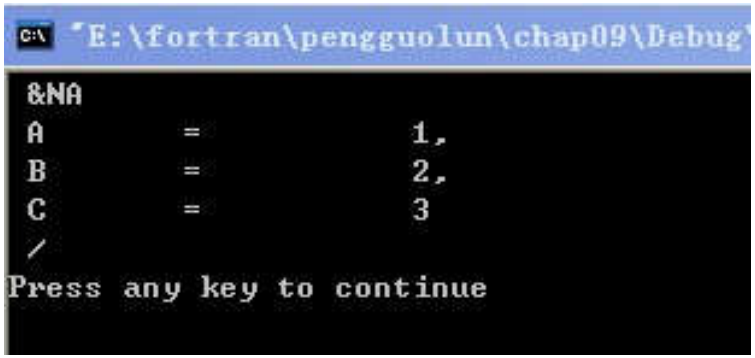
- NAMELIST是很特殊的输入输出方法，收录在f90标准当中。有名I/O列表经常用于read/write语句。

  Read/write(UNIT=unit,NML=nl_group_name,[…])

- 当执行表控有名列表的write语句时，有名列表中的所有变量名都会和其值一起按照特定的顺序输出。输出的第一项是&，&后随“有名列表”名字，然后是一系列按照“NAME=Value”格式的输出值。最后，列表以一个斜杠“/”结束。



```
program ex0918
  implicit none
  integer :: a = 1, b = 2, c = 3
  namelist /na/ a,b,c
  !把a,b,c这三个变量放在名字叫做na的namelist中
  write(*,nml=na)
  stop
end program
```



namelist也算是声明的一部分，必须编写在程序执行命令前面。

```
PROGRAM write_namelist

! Purpose:  To illustrate a NAMELIST-directed WRITE statement.

IMPLICIT NONE

! Data dictionary: declare variable types & definitions
INTEGER :: i = 1, j = 2                          ! Integer variables
REAL :: a = -999., b = 0.                        ! Real variables
CHARACTER(len=12) :: string = 'Test string.'  ! Char variables
NAMELIST / mylist / i, j, string, a, b       ! Declare namelist

OPEN (8,FILE='output.nml',DELIM='APOSTROPHE')

WRITE (UNIT=8, NML=mylist)                   ! Write namelist
CLOSE (8)                                    ! Close file

END PROGRAM write_namelist
```

```
&MYLIST
I      =           1,
J      =           2,
STRING = 'Test string.',
A      =  -999.0000    ,
B      =  0.0000000E+00
/
```

- NAMELIST也可以用来输入数据，不过通常都会用来读取文件，不会用在键盘输入。

```
program ex0919
  implicit none
  integer :: a, b, c
  namelist /na/ a,b,c
  read (*,nml=na)
  write(*,nml=na)
  stop
end program
```



- 输入格式需要按照前面的格式。先输入符号&，紧接namelist的名字。再输入变量的名称，等号以及内容，要结束时还要加上除号。

  &na a=1 b=2 c=3/

- 读取namelist时，可以不填入所有变量的值，只要以&na开始输入，除号结束输入。
- 不需要按照变量顺序输入，程序会自动按照变量名称来设置数值。变量甚至可以重复输入，不过变量会得到最后一次设置的数值。



namelist通常使用在文本文件的输入 / 输出中，使用read从文件中读取数据时，会自动从目前的位置向下寻找存放namelist的地方。

```
program ex0920
  implicit none
  integer :: a(3)
  namelist /na/ a

  open(10, file="ex0920.txt")
  read (10,nml=na)
  write(*,"(3I2)") a

end program
```

```
&MYLIST
I =           -111
STRING = 'Test 1.'
STRING = 'Different!'
B =      123456.
/
```

通过NAMELIST读写数据

- PROGRAM read_namelist
- ! Purpose: To illustrate a NAMELIST-directed READ statement
- IMPLICIT NONE

! Data dictionary: declare variable types & definitions

- INTEGER :: i = 1, j = 2 ! Integer variables
- REAL :: a = -999., b = 0. ! Real variables
- CHARACTER(len=12) :: string = 'Test string.' ! Char variables
- NAMELIST / mylist / i, j, string, a, b ! Declare namelist

- OPEN (7,FILE='input.nml',DELIM='APOSTROPHE') ! Open input file.
- ! Write NAMELIST before update
- WRITE (*,'(1X,A)') 'Namelist file before update: '
- WRITE (UNIT=*, NML=mylist)
- READ (UNIT=7,NML=mylist) ! Read namelist file.
- ! Write NAMELIST after update
- WRITE (*,'(1X,A)') 'Namelist file after update: '
- WRITE (UNIT=*, NML=mylist)
- END PROGRAM read_namelist

```
"F:\book_files\chap14\Debug\
Namelist file before update:
&MYLIST
I       =           1,
J       =           2,
STRING  = Test string.,
A       =  -999.0000    ,
B       =  0.0000000E+00
/
Namelist file after update:
&MYLIST
I       =        -111,
J       =           2,
STRING  = Different!  ,
A       =  -999.0000    ,
B       =   123456.0
/
Press any key to continue
```

- 如果打开有名列表输出文件时使用的是引号作为定界符，那么由有名列表write语句所写入的输出文件是可以被有名列表read语句直接读取的。这一事实给相互独立的程序之间或者同一程序不同运行之间的数据交换带来了方便。
- 数组名、部分数组以及数组元素可出现在NAMELIST中。如果数组名出现在有名列表中，那么当执行有名列表write时，数组的每个元素都会一次输出到有名列表中。当执行有名列表read时，可以单独给每个数组元素赋值。
- 形参和动态创建的变量不能出现在NAMELIST中，包括无上下界数组的形参、长度不定的字符变量、自动变量以及指针。



在屏幕上浏览文本文件内容

[ex0905.F90]

```
program ex0905
  implicit none
  character(len=79)  :: filename
  character(len=79)  :: buffer
  integer, parameter :: fileid = 10
  integer :: status = 0
  logical alive

  write(*,*) "Filename:"
  read (*,"(A79)") filename
  inquire( file=filename, exist=alive)
```

输入并查询文件存在与否



```
if ( alive ) then
  open(unit=fileid, file=filename, &
               access="sequential", status="old")
  do while(.true.)
    read(unit=fileid, fmt="(A79)", iostat=status ) buffer
    if ( status/=0 ) exit ！没有数据就跳出循环
    write(*,"(A79)") buffer
  end do
else
  write(*,*) TRIM(filename)," doesn't exist."
end if

 stop
end
```

如果文件存在，就以顺序文件的形式打开

以“死循环”的方式读取文件中的内容，并在屏幕上面显示出来。遇到文件结尾，直接停止

如果文件不存在，屏幕提示



记录全部同学考试成绩的程序

[ex0906.F90]

```
 module typedef
   type student
     integer Chinese,English,Math
   end type
 end module

 program ex0906
   use typedef
   implicit none
   integer :: students
   type(student), allocatable :: s(:)
   character(len=80) :: filename = "data.txt"
   integer, parameter :: fileid = 10
   integer :: i
```

使用module，并创建自定义数据类型



```
write(*,*) "班上有多少学生?"
 read (*,*) students
 allocate( s(students), stat=i )
 if ( i/=0 ) then
   write(*,*) "Allocate buffer fail."
              stop
 end if
```

动态分配数组

```
 open(fileid, file=filename)
 do i=1,students
   write(*,"('请输入'I2'号同学的中文、英文及数学成绩')") i
   read(*,*) s(i)%Chinese, s(i)%English, s(i)%Math
   write(fileid,"('座号:'I2/'中文:'I3' 英文:'I3' 数学:'I3)") i,s(i)
 end do
 close(fileid)

  stop
end
```

键盘读入成绩并写到文件中

从文件中读取学生成绩的程序

[ex0907.F90]

```
module typedef
  type student
    integer Chinese,English,Math
  end type
end module

program ex0907
  use typedef
  implicit none
  type(student) :: s
  character(len=80) :: filename = "data.txt"
  integer, parameter :: fileid = 10
  logical alive
  integer :: error
  integer :: no
```

```
inquire(file=filename, exist=alive)
 if ( .not. alive ) then
   write(*,*) trim(filename)," doesn't exist."
      stop
 end if

 open(fileid, file=filename)
 do while(.true.)
      read(fileid,"(5X,I2,/,5XI3,6XI3,6XI3)",iostat=error) no,s
      if ( error/=0 ) exit
      write(*,"(I2'号 中文:'I3' 英文:'I3' 数学:'I3)") no,s
 end do
 close(fileid)

 stop
end
```

从版面比较自由的文件中读入选手数据并显示特定数据的程序

[ex0908.F90]

```
module typedef
  type player
    character(len=80) :: name
    real weight, height
         real score
  end type
end module

program ex0908
  use typedef
  implicit none
  character(len=20)  :: filename = "ex0908dat.txt"
  integer, parameter :: fileid = 20
  logical :: alive   ！检查文件是否存在
  type(player) :: p  ！读取选手数据
  character(len=10)  :: title ！读取数据项目
  real tempnum       ！读取数据
  logical, external :: GetNextPlayer ！找出下一位球员数据的函数
  integer i          ！循环记数器
  integer error      ！检查文件读取是否正常
```

ex0908dat.txt - 记事本

文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

```
姓名 王天才
体重 80
身高 195
得分 15.8

姓名 李天才
身高 190
体重 85
得分 10.8


姓名 洪天才
体重 90
身高 201
得分 19.8

姓名 彭天才
体重 70
得分 22.2
身高 185

姓名 黄天才
得分 20.1
体重 85
身高 190
```

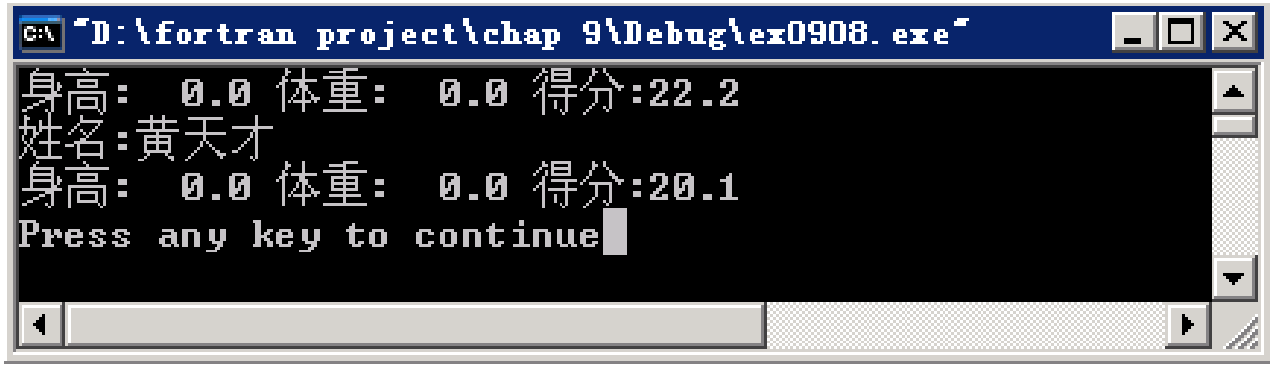
```
inquire(file=filename, exist=alive)
 if ( .not. alive ) then ! 文件不存在就结束程序
   write(*,*) trim(filename)," doesn't exist."
       stop
 end if

 open(unit=fileid, file=filename)
 do while(.true.)
   if ( GetNextPlayer(fileid, p%name) ) then
         do i=1,3
           read(fileid, "(A4,1X,F)", iostat=error ) title, tempnum
              if ( error/=0 ) then
                write(*,*) "文件读取错误"
                stop
              end if
           ! 要经由每一行最前面两个中文字来判断读入的是什么数据
           select case(title)
           case("身高")
             p%height = tempnum
```

```
        case("体重")
          p%weight = tempnum
        case("得分")
          p%score = tempnum
            case default
              write(*,*) "出现不正确的数据"
              stop
      end select
        end do
      else
        exit
      end if


      if ( p%score > 20.0 ) then ! 印出得分高于20分的选手数据
        write(*,"('姓名:',A8/,'身高:',F5.1,' 体重:',F5.1,' 得分:',F4.1)") p
      end if
  end do

end
```

```
! GetNextPlayer函数会从文件中找出下一位球员的数据位置
! 如果文件中还有数据需要读取, 传回.true.
! 如果文件中没有数据需要读取, 传回.false.
logical function GetNextPlayer(fileid, name)
  implicit none
  integer, intent(in) :: fileid
  character(len=*), intent(out) :: name
  character(len=80) title
  integer  error

  do while(.true.)
    read(fileid,"(A80)",iostat=error) title
```



```
    if ( error/=0 ) then ! 文件中已经没有数据了
      GetNextPlayer = .false.
      return
    end if

    if ( title(1:4)=="姓名" ) then
      name = title(6:)
      GetNextPlayer = .true.
      return
    end if
  end do

  return
end function
```



把文本文件内容的最前面加上行号后，输出到另一个文件的程序

```
program ex0909
  implicit none
  integer, parameter :: inputfileid = 10, outputfileid = 11
  integer, parameter :: maxbuffer = 200
  character(len=80)  :: inputfile, outputfile
  character(len=maxbuffer) buffer
  integer count
  integer error
  logical alive

  write(*,*) "Input Filename:"
  read(*,"(A80)") inputfile
  write(*,*) "Output Filename:"
  read(*,"(A80)") outputfile

  inquire(file=inputfile, exist=alive)
  if ( .not. alive ) then
    write(*,*) trim(inputfile)," doesn't exist."
      stop
  end if
```

```
  open(unit=inputfileid, file=inputfile, status="old")
  open(unit=outputfileid, file=outputfile, status="replace")
  count = 1
  do while(.true.)
   ! 读入一整行的数据
   read(inputfileid,"(A200)",iostat=error) buffer
    if ( error/=0 ) exit ! 没有数据了,离开循环
    ! 再最前面加上行号再输出到另一个文件中
   write(outputfileid, "(I3,'.',A)") count,trim(buffer)
     count=count+1        ! 计算行数
  end do
  close(inputfileid)
  close(outputfileid)

  stop
end program
```



9-4 直接（读取）访问文件

- 直接读取文件是指使用直接读取模式读写的文件。顺序访问文件中的记录必须按照顺序从第一条记录访问到最后一条。相反，直接读取文件中的记录可以按照任意顺序进行访问。
- 操作直接读取文件的关键：直接读取文件中的每条记录的长度都必须相等。
- 通过在open语句中指定access='DIRECT'的方式打开直接读取文件。
- 必须在open语句中使用RECL=子句来指明直接读取文件中的记录长度。



示例：格式化直接读取文件

```
1.  PROGRAM direct_access_formatted
2.  !
3.  !  Purpose:
4.  !    To illustrate the use of direct access Fortran files.
5.  !
6.  !  Record of revisions:
7.  !      Date       Programmer         Description of change
8.  !      ====       ==========         =====================
9.  !    12/11/06    S. J. Chapman       Original code
10. !
11. IMPLICIT NONE

12. ! Data dictionary: declare variable types & definitions
13. INTEGER :: i                        ! Index variable
14. INTEGER :: irec                     ! Number of record in file
15. CHARACTER(len=40) :: line           ! String containing current line.
```

```
16. ! Open a direct access formatted file with 40 characters per record.
17. OPEN ( UNIT=8, FILE='dirio.fmt', ACCESS='DIRECT', &
18.        FORM='FORMATTED', STATUS='REPLACE', RECL=40 )
19.
20. ! Insert 100 records into this file.
21. DO i = 1, 100
22.   WRITE ( 8, '(A,I3,A)', REC=i ) 'This is record ', i, '.'
23. END DO

24. ! Find out which record the user wants to retrieve.
25. WRITE (*,'(A)',ADVANCE='NO') ' Which record would you like to see? '
26. READ (*,'(I3)') irec
27.
28. ! Retrieve the desired record.
29. READ ( 8, '(A)', REC=irec ) line
30.
31. ! Display the record.
32. WRITE (*, '(A,/,5X,A)' ) ' The record is: ', line

33. END PROGRAM direct_access_formatted
```

写入的记录：

This is record i.

输出字符串，不换行等待键盘输入

从文件中读取输入的记录号所对应的记录内容

屏幕输出读取的记录内容



示例：格式化与未格式化直接读取文件

```
1.  PROGRAM direct_access
2.  !
3.  !  Purpose:
4.  !    To compare direct access formatted and unformatted files.
5.  !
6.  !  Record of revisions:
7.  !      Date       Programmer          Description of change
8.  !      ====       ==========          =====================
9.  !    12/11/06    S. J. Chapman        Original code
10. !
11. IMPLICIT NONE
12. ! List of parameters:
13. INTEGER, PARAMETER :: SINGLE = SELECTED_REAL_KIND(p=6)
14. INTEGER, PARAMETER :: DOUBLE = SELECTED_REAL_KIND(p=14)
15. INTEGER, PARAMETER :: MAX_RECORDS = 20000     ! Max # of records
16. INTEGER, PARAMETER :: NUMBER_OF_READS = 50000 ! # of reads
17. ! Data dictionary: declare variable types & definitions
18. INTEGER :: i, j                               ! Index variable
19. INTEGER :: length_fmt = 80                    ! Length of each record in  formatted file
20. INTEGER :: length_unf                         ! Length of each record in unformatted file
```



```
21. INTEGER :: irec                    ! Number of record in file
22. REAL(KIND=SINGLE) :: time_fmt  ! Time for formatted reads
23. REAL(KIND=SINGLE) :: time_unf  ! Time for unformatted reads
24. REAL(KIND=SINGLE) :: value     ! Value returned from random0
25. REAL(KIND=DOUBLE), DIMENSION(4) :: values
26.                                    ! Values in record
27. ! Get the length of each record in the unformatted file.
28. INQUIRE (IOLENGTH=length_unf) values
29. WRITE (*,'(A,I2)') ' The unformatted record length is ', length_unf
30. WRITE (*,'(A,I2)') ' The formatted record length is ', length_fmt
31. ! Open a direct access unformatted file.
32. OPEN ( UNIT=8, FILE='dirio.unf', ACCESS='DIRECT', &
33.    FORM='UNFORMATTED', STATUS='REPLACE', RECL=length_unf )
34. ! Open a direct access formatted file.
35. OPEN ( UNIT=9, FILE='dirio.fmt', ACCESS='DIRECT', &
36.    FORM='FORMATTED', STATUS='REPLACE', RECL=length_fmt )
```

Inquire语句的第三种形式

分别打开格式化和未格式化直接读取文件



```
37. ! Generate records and insert into each file.
38. DO i = 1, MAX_RECORDS
39.    DO j = 1, 4
40.       CALL random0(value)              ! Generate records
41.       values(j) = 30._double * value
42.    END DO
43.    WRITE (8,REC=i) values              ! Write unformatted
44.    WRITE (9,'(4ES20.14)',REC=i) values   ! Write formatted
45. END DO

46. ! Measure the time to recover random records from the unformatted file.
47. CALL set_timer
48. DO i = 1, NUMBER_OF_READS
49.    CALL random0(value)
50.    irec = (MAX_RECORDS-1) * value + 1
51.    READ (8,REC=irec) values
52. END DO
```

20000个记录，每个记录包含4个随机数据

随机数子程序参看第8章过程中的例子

分别在格式化和未格式化文件中写入记录

调用子程序计算读取时间，见后面过程

随机读取50000个未格式化文件中的记录



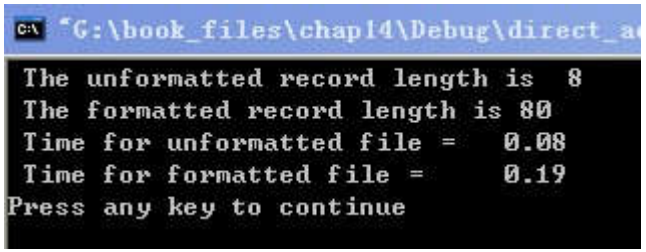
名称	大小	类型	修改日期
dirio.unf	625 KB	UNF 文件	2010-5-24 16:23
dirio.fmt	1,563 KB	FMT 文件	2010-5-24 16:23

```
53. CALL elapsed_time (time_unf)

54. ! Measure the time to recover random records from the formatted file.
55. CALL set_timer
56. DO i = 1, NUMBER_OF_READS
57.    CALL random0(value)
58.    irec = (MAX_RECORDS-1) * value + 1
59.    READ (9,'(4ES20.14)',REC=irec) values
60. END DO
61. CALL elapsed_time (time_fmt)
62. ! Tell user.
63. WRITE (*,'(A,F6.2)') ' Time for unformatted file = ', time_unf
64. WRITE (*,'(A,F6.2)') ' Time for formatted file =   ', time_fmt

65. END PROGRAM direct_access
```

计算出随机读取50000个记录的时间

调用子程序计算读取时间，见后面过程

随机读取50000个格式化文件中的记录



计算出随机读取50000个记录的时间

```
MODULE save_timer
!
!  Purpose:
!    To hold the time from the call to set_timer for use
!    in calls to subroutine elapsed_time.
!
IMPLICIT NONE
! Time of call to set_timer
INTEGER,SAVE,DIMENSION(8) :: t1
! Time of call to elapsed_time
INTEGER,SAVE,DIMENSION(8) :: t2
END MODULE save_timer
```



```
SUBROUTINE set_timer
!
!  Purpose:
!    To start (or reset) the elapsed time counter.  Elapsed
!    time is returned by subsequent calls to subroutine
!    elapsed_time.
!
!  Record of revisions:
!      Date       Programmer        Description of change
!      ====       ==========        =====================
!    11/22/06    S. J. Chapman       Original code
!
!  List of calling arguments:
!     none
!
USE save_timer
IMPLICIT NONE

! Get current time.
CALL DATE_AND_TIME ( VALUES=t1 )

END SUBROUTINE set_timer
```

调用系统当前的时间，设为t1

```
SUBROUTINE elapsed_time ( sec )
!  Purpose:   To return the time in seconds since the last call to subroutine set_timer.
USE save_timer
IMPLICIT NONE

! Dummy arguments
REAL :: sec             ! Elapsed time in seconds

! Local variables
INTEGER(KIND=SELECTED_INT_KIND(9)) :: msec

! Get current time.
CALL DATE_AND_TIME ( VALUES=t2 )

! Now get the elapsed time in milliseconds.
msec = (3600000*t2(5) + 60000*t2(6) + 1000*t2(7) + t2(8) ) &
     - (3600000*t1(5) + 60000*t1(6) + 1000*t1(7) + t1(8) )

! Convert to floating point seconds and return result.
sec = 0.001 * REAL(msec)

END SUBROUTINE elapsed_time
```

再次调用系统当前的时间，设为t2

计算出时间差，并把毫秒转换成秒



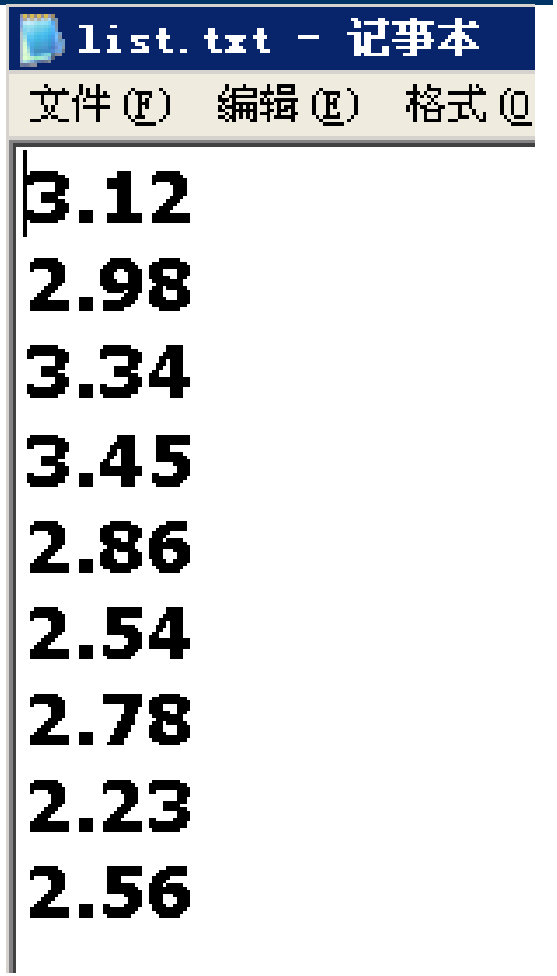
格式化文件与未格式化文件比较

格式化文件	未格式化文件
可以在输出设备上显示数据	不能在输出设备上显示数据
可以在不同计算机之间传递数据	无法轻易的在使用不同内部数据表示法的计算机之间传送数据。
需要大量的磁盘空间	需要的空间相对要小
速度慢，需要大量的计算机时间	速度快，需要很少的计算机时间
在格式化时可能导致截尾或舍入误差	不存在截尾或舍入误差

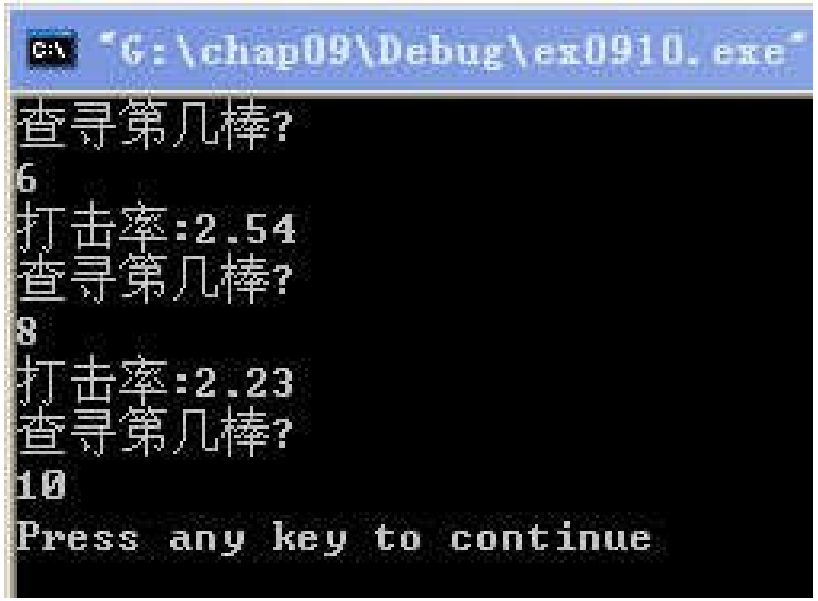


经过棒次来查询选手打击率的程序

```
program ex0910
  implicit none
  integer, parameter :: fileid = 10
  character(len=20)  :: filename = "list.txt"
  integer player
  real::    hit
  integer:: error
  logical:: alive

  inquire(file=filename, exist=alive)
  if ( .not. alive ) then
    write(*,*) trim(filename)," doesn't exist."
     stop
  end if
```



```
open(unit=fileid, file=filename, access="direct",&
     form="formatted", recl=6, status="old")
 do while(.true.)
   write(*,“(‘查寻哪一个选手的打击率?')")
    read (*,*) player
    read(fileid, fmt="(F4.2)", rec=player, IOSTAT=error) hit
    if ( error/=0 ) exit
    write(*,"('打击率:'F4.2)") hit
 end do
 close(fileid)

 stop
end program
```



9-5 二进制文件的操作

- 二进制文件是直接把内存的二进制数据写入文件，也就没有所谓的格式化输入／输出。
- Fortran中的所谓“二进制文件”，其实就是直接读取的未格式化文件。直接读取时，open命令中的recl字段所设置的整数n值会随编译器不同而改变。
- 存放“精确”及“大量”的数据时，使用二进制文件是比较好的选择。用文本文件保存数字，很可能会造成部分数据流失。如格式F8.3值允许存放3位小数，第3位以下的小数都会被舍去。而使用二进制文件则不会有任何的数据流失。



使用二进制文件来记录输入棒球选手打击率的程序

```
program ex0912
  implicit none
  integer, parameter :: fileid = 10
  character(len=20)  :: filename = "list.bin"
  integer player
  real  :: hit(9) = (/ 3.2, 2.8, 3.3, 3.2, 2.9, 2.7, 2.2, 2.3, 1.9 /)

  open(unit=fileid, file=filename, form="unformatted", &
       access="direct", recl=4, status="replace")
  do player=1,9
    write(fileid, rec=player) hit(player)
  end do

  close(fileid)
end program
```

默认情况下，是formatted格式

经过棒次来查询选手打击率的程序（使用二进制文件）

```
program ex0913
  implicit none
  integer, parameter :: fileid = 10
  character(len=20)  :: filename = "list.bin"
  real    hit
  integer player
  logical alive
  integer error

  inquire(file=filename, exist=alive)
  if ( .not. alive ) then
    write(*,*) trim(filename)," doesn't exist."
     stop
  end if
```



```
open(unit=fileid, file=filename, form="unformatted",&
      access="direct", recl=4, status="old")

 do while(.true.)
   write(*,"('第几棒?')")
    read (*,*) player
    read(fileid, rec=player, iostat=error) hit
   if ( error/=0 ) exit
    write(*,"('打击率:',F5.2)") hit
 end do

 close(fileid)

end program
```



9-6 字符串文件

使用写入文件的方法，把数据写到一个字符串变量中。

```
program ex0914
  implicit none
  integer :: a=2
  integer :: b=3
  character(len=20) :: string

  write( unit=string, fmt="(I2,'+',I2,'=',I2)" ) a,b,a+b
  !把字符串变量当作输出的目的
  write(*,*) string

  stop
end program
```

可以经过read命令从字符串读入数据：

```
program ex0915
  implicit none
  integer :: a
  character(len=20) :: string="123"

  read( string, * ) a !从字符串string中读出一个数字
  write(*,*) a

  stop
end program
```



- 在某些情况下需要使用内部文件来设置数据：使用read命令从键盘输入数据时，如果用户输入错误的数据，会导致死机。
- 如需要输入整数时却输入英文字母，就可能会死机。比较好的处理办法是，程序先暂时把数据当作字符串读入，检查字符串中是否含有不合理的字符，如果字符串中都是0～9的数字字符，就把字符串转成整数，不然就请用户再输入一次。



```
program ex0916
  implicit none
  integer i
  integer, external :: GetInteger
  i = GetInteger()
  write(*,*) i
  stop
end program

integer function GetInteger()
  implicit none
  character(len=80) :: string
  logical :: invalid
  integer i, code
```



```
invalid = .true.
 do while( invalid )
   write(*,*) "请输入正整数"
   read(*, "(A80)" ) string
   invalid = .false.
   do i=1, len_trim(string)
              ! 检查输入的字符是否包含'0'～'9'以外的字符
     code = ichar(string(i:i))
     if ( code<ichar('0') .or. code>ichar('9') ) then
                 invalid=.true.
                          exit
               end if
   end do
 end do

 read( string, * ) GetInteger
 return
end function
```



内部文件还可应用在动态改变输出格式，输出格式可以事先存放在字符串中，程序进行时，动态改变字符串内容就可以改变输出格式。

```
program ex0917
  implicit none
  integer :: a=2
  integer :: b=3
  character(len=20) :: string
  character(len=20) :: fmtstring="(I??+I??=I??)"

  write(fmtstring(3:4),"(I2.2)") int(log10(real(a))+1)
  write(fmtstring(7:8),"(I2.2)") int(log10(real(b))+1)
  write(fmtstring(11:12),"(I2.2)") int(log10(real(a+b))+1)
  write(*,*) fmtstring

  stop
end program
```

作 业

P274：

1、2、3、4、5

所需要的文件从ftp上面下载。

将“综合训练1”和“综合训练2”的数据输出到文件中，然后利用第三方软件（如excel、origin等）进行作图。