Input and output

Free format:

```
      i=10; x(i)=1.2345
      write (6,*) i, x(i)

    10 1.23450005
```

Formatted output

Old method:

```
        write (6, 10) i, x(i)
   10  format(I3,F5.2)

    10 1.23
```

Since Fortran 77 also

```
write (6, '(I3,F5.2)') i,x(i)
```

A more dynamic way:

```
character (len=80) :: form
character (len=2) :: fi
...
fi='I2'; if (i>99) fi='I3'
form='('//fi//',F5.2)'
write(6,form) i,x(i)

write (6, 10) i, x(i)
  10 format('x(',I3,')=',F5.2)
---
     write (6, "('x(',I3,')=',F5.2)") i,x(i)
---
  x( 10)= 1.23
```

Formats:

`In` integer, n digits

`Fn.d` real number, n characters altogether, d decimals

`En.d` real number using exponent notation

`In`, `Fn`, `En`: if the field is too short, print n asterisks; if the field is too wide the number is printed flush right.

`A` character string (A=alphabetic)

`An` n characters; if the field is too short print n first characters; if the string is shorter than n characters, print it flush right.

`nX` skip n characters

`Tn` (tabulator) skip to column n

`/` line feed

Automatic line feed after every `write`.

If there are more variables to be printed than formats, make a line feed and restart from the beginning of the format list.

```
write(6,'(10F5.2/10F5.1)') a(1,1:10), a(2,1:10)

real x(10)
write(6,'(F5.2)') x   ! print 10 lines
write(6,'(10F5.2)') x  ! print everything on the same line
write(6,'(30F5.2)') x  ! print only 10 values
write(6,'(3F5.2)') x   ! print 4 lines (3+3+3+1 values)
```

Format is just a string of characters that is not checked at compilation time. Excecution may crash if the format is not appropriate for the data to be printed.

`read`: formats as in output.

Free form input: variables separated by commas:

```
real x
integer n
read(5,*) x,n

1.5,20
```

Formatted input (mainly for reading files):

```
read(5,'(F4.1I2)') x,n
! 1.520
!1.5 20
!  1520
```

Line feed after an output can be omitted:

```
write(6,"('x=?')", advance='no')
read(5,*) x
```

Input and output statements can have two addresses for error situations:

```
      read(5,1, ERR=100, END=900) x
    1 format(f5.2)
      ...
  100 write(6,"('reading failed')")
      ...
  900 write(6,"('end of file')")
```

In F90 no addresses are needed:

In F90 no addresses are needed. IO statements return a status code. The cose is 0 if everything is ok, negative for end of file, positive for other errors.

```
integer status
do
 read(5,form, iostat=status) x
 if (status > 0) then
   write(6,"('reading failed')")
   exit
 else if (status < 0) then
   write(6,"('end of file')")
   exit
 end if
 ...
end do
```

```
integer status
do
 read(5,form, iostat=status) x
 if (status /= 0) then
   call error(status)
   exit
 end if
```

```
  ...
end do
...
subroutine error(code)
  integer code
  write(6,"('error',I4)") code
end subroutine
```

Logical unit numbers

Units 5 and 6 refer by default to the terminal; no need to open or close these files.

Other unit numbers may refer to some default file (`fort0001.dat` etc.); this system dependent.

A file can be opened with the `open` statement:

```
open (1, 'fyle')

! create a new file
! the file must not exits previously
open (1, file='fyle', status='new')

! open an existing file
open (1, file='fyle', status='old', ERR=900)

! if the file exists and is written to
! the old file is deleted and a new one created
open (1, file='fyle', status='unknown')
```

All files are automatically closed when the profram end.

A file can be explicitly closed with the `close` statement. Then the same unit number can be assigned to another file.

```
close(1)
```

Binary IO

Formatting is a rather slow process. Binary output can be used if the output doesn't have to be human readable.

E.g. a very long calculation should store intermediate results every now and then so that the calculation can be continued after a power failure.

```
open(1, file='bin.dat', form='unformatted')

write (1) x
```

Usually a file is read and written a record at a time sequentially from the beginning.

For a large database a direct access file is more efficient.

Declaration of a direct access file:

```
open(1, file='lista', access='direct', recl=128)
read(1, rec=i) x
```

Direct access file is a binary file by default.

For a direct access file the record lenght in bytes `recl` must always be given.

In a read or write statement the record number (`rec`) must be given.

```
program koe
type star
 real :: ra, dec, mag
 character (len=10) :: name
end type
type (star) :: s

open(1,file='catal.dat',access='direct',recl=22)
write (1,rec=1) 1.0, 2.0, 0.0, 'star1'
write (1,rec=2) 5.0, 3.0, 10.0, 'star2'
close(1)

open(1,file='catal.dat',access='direct',recl=22)
s=getstar(2)
write(6,*) s%ra, s%dec, s%mag, s%name
close(1)

contains

function getstar(n)
 implicit none
 type (star) :: getstar
```

```
      integer n
      real x,y,m
      character (len=10) :: name
      read (1,rec=n) x,y,m,name
      getstar%ra=x
      getstar%dec=y
      getstar%mag=m
      getstar%name=name
    end
  end

   5.000000     3.000000     10.00000   star2

ls -l catal.dat:

     -rw-------  1 hkarttun tah        44 Oct  8 12:29 catal.dat

od -c catal.dat
0000000  \0  \0 200   ?  \0  \0  \0   @  \0  \0  \0  \0   s   t   a   r
0000020   1  \0  \0  \0  \0  \0  \0  \0 240   @  \0  \0   @   @  \0  \0
0000040     A   s   t   a   r   2  \0  \0  \0  \0  \0
0000054
```