

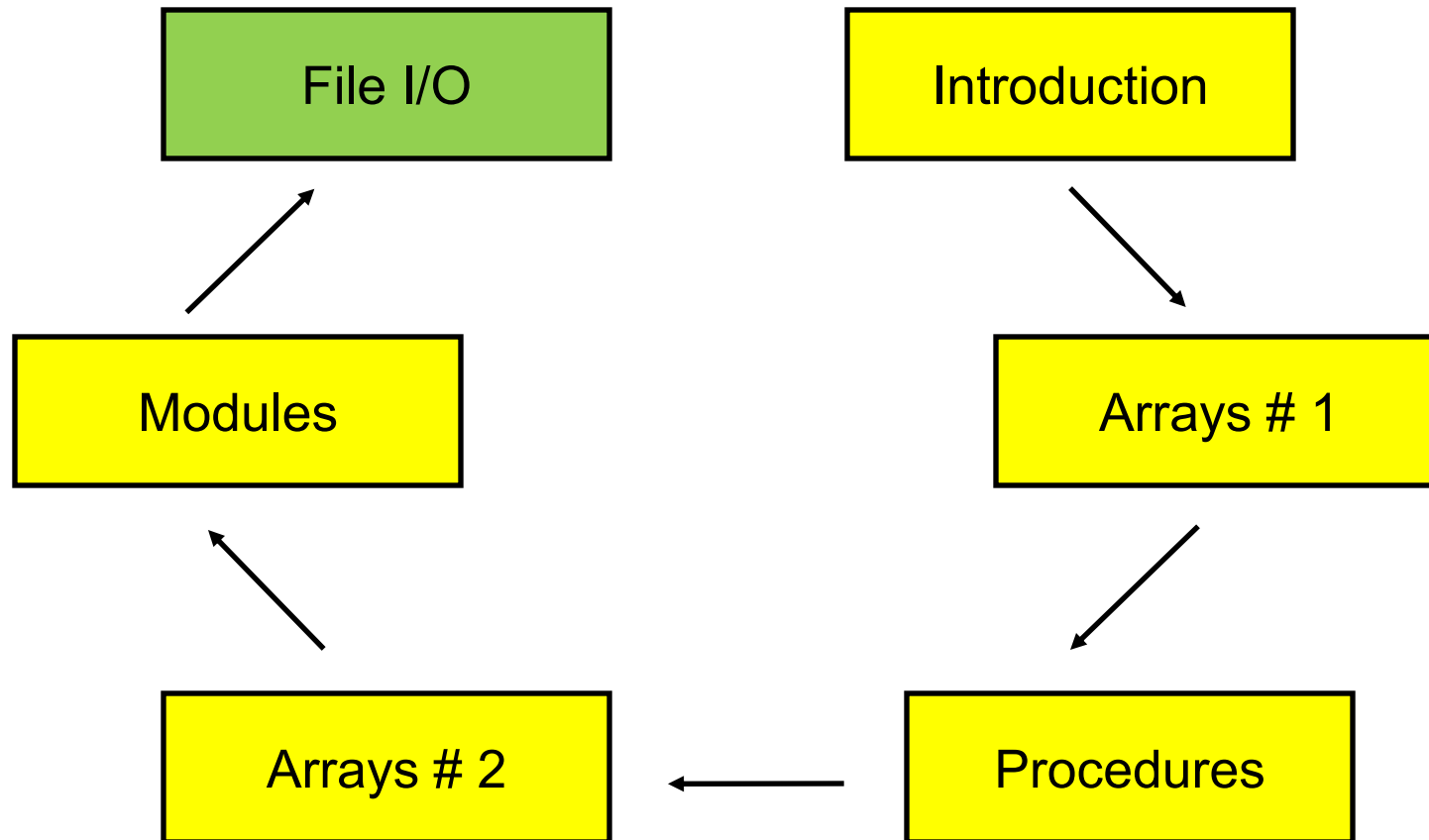
# Fortran95

## *File I/O*

CSC – IT Center for Science Ltd.  
Espoo, Finland

CSC – Tieteen tietotekniikan keskus Oy  
CSC – IT Center for Science Ltd.

# Fortran 95 course



# File I/O Motivation



- File interface with other applications
- Data reading
- Data writing

# Contents



- Open / Close / Inquire - functions
- Read / Write / Format - statements
- File types (text, binary, direct access)
- NAMELIST I/O
- Internal I/O

# Basic concepts

- Writing to or reading from a file is basically similar to writing onto a terminal screen or reading from a keyboard
- Differences
  - File must be opened first with OPEN-statement, in which the unit number and (optionally) a file name are given
  - Subsequent writes (or reads) must refer to the given unit number

# Basic concepts (cont'd)

- ...differences (cont'd)
  - File should finally be closed
- For example :

```
INTEGER :: iu
```

```
iu = 12
```

```
OPEN(unit=iu, file = 'foo')
```

```
WRITE(iu, ...) ! or READ(iu, ...)
```

```
CLOSE(iu)
```

# Contents



- **Open / Close / Inquire - functions**
- Read / Write / Format - statements
- File types (text, binary, direct access)
- NAMELIST I/O
- Internal I/O

# Opening & closing a file

- The syntax is (the brackets [ ] indicate optional keywords or arguments) :

```
OPEN([unit=]iu, file='name' [, options])
```

```
CLOSE([unit=]iu [, options])
```

- For example :

```
OPEN(10, file= 'output.dat', status='new')
```

```
CLOSE(unit=10, status='keep')
```

# Opening & closing a file (cont'd)



- The first parameter is the unit number
- The keyword `unit=` can be omitted
- The unit numbers 0, 5 and 6 are predefined
  - *0 is output for standard (system) error messages*
  - *5 is for standard (user) input*
  - *6 is for standard (user) output*
- These units are opened by default and should not be closed

# Opening & closing a file (cont'd)



- You can also refer to the default output or input unit with asterisk

**WRITE(\*, ...)** ! or **READ(\*, ...)**

- Note that they are NOT necessarily the same as the unit numbers 5 and 6
- If the file name is omitted in the OPEN, the a file based on unit number will be opened, f.ex. for unit=12 → **'fort.12'**

# File opening options

- Investigating the **options**-flags in `OPEN([unit=]iu, file='name' [,options])`
- The **options**-flags can be one (or a suitable combination) of the following :
  - **status** , **position** , **action** , **form**
  - **access** , **iostat** , **err** , **recl**

# File opening options (cont'd)

- **status** : existence of the file

'old', 'new', 'replace', 'scratch', 'unknown'

- **position** : offset, where to start writing

'append'

- **action** : file operation mode

'write', 'read', 'readwrite'

- **form** : text or binary file

'formatted', 'unformatted'

# File opening options (cont'd)

- **access** : direct or sequential file access  
**'direct', 'sequential'**
- **iostat** : error indicator, (output) integer
  - Non-zero only upon error
- **err** : the label number to jump upon error
- **recl** : record length, (input) integer
  - For direct access files only
  - Warning / check : **may be in bytes or words**

# Obtaining file properties

- Use **INQUIRE** -statement to find out information about file existence, file unit open status, various attributes etc.
- The syntax has two forms, one based on file name, the other for unit number

```
INQUIRE(file='name', options  
...)
```

```
INQUIRE(unit=iu, options ...)
```

# Obtaining file properties (cont'd)



- The **options** contains one or more (**keyword** , **variable**) – pairs
- The corresponding **variable** contains the information that was inquired
  - Depending on context, the **variable** is either LOGICAL, CHARACTER-string or INTEGER
- There are plenty of various **keywords** , of which probably the most useful ones are described next

# Obtaining file properties (cont'd)



- **exists** : file existence ? (LOGICAL)
- **opened** : file / unit is opened ? (LOGICAL)
- **form** : 'formatted' or 'unformatted' (CHAR)
- **access** : 'sequential' or 'direct' (CHAR)
- **action** : 'read', 'write', 'readwrite' (CHAR)
- **recl** : record length (INTEGER)
- **size** : file size in bytes (INTEGER)

# Example of file properties

- Find out about file existence

```
logical :: exfile
```

```
INQUIRE (file='foo.dat', exists=exfile)
```

```
if (.not. exfile) then
```

```
    write(*,*) 'The file does not exist'
```

```
else
```

```
    ...
```

```
endif
```

# Contents



- Open / Close / Inquire - functions
- **Read / Write / Format - statements**
- File types (text, binary, direct access)
- NAMELIST I/O
- Internal I/O

# Using WRITE statement

- Writing to a file is done by giving the corresponding unit number (iu) as a parameter :

```
write(iu,*) str
```

```
write(unit=iu, fmt=*) str
```

- Formats and other options can be used as needed
- If keyword 'unit' used, also 'fmt' keyword must be used (for formatted, text files)

# Using WRITE statement (cont'd)



- If the file unit (iu) has not been explicitly OPENed, the very first WRITE on that unit will trigger an implicit OPEN
- In most UNIX systems this means opening a file named as 'fort.<iu>', where <iu> is the unit number in concern
- Star ('\*') format indicates list directed output (a programmer do not choose the output style)

# Using WRITE statement (cont'd)



- For unformatted (binary) WRITES to a sequential file, syntax is as follows :

**WRITE(iu) array**

- For unformatted direct access WRITES the syntax requires also a record number (rec)

**WRITE(iu,rec=78) array**

# Using READ statement

- Reading from a file is done by giving the corresponding unit number (iu) as a parameter :

```
read(iu,*) str
```

```
read(unit=iu, fmt=*) str
```

- Formats and other options can be used as needed
- Star (“\*”) format indicates free format input (list directed input)

# Using READ statement (cont'd)



- If the file unit (*iu*) has not been explicitly OPENed, the very first READ on that unit will trigger an implicit OPEN
- In most UNIX systems this means opening a file named as 'fort.<iu>', where <iu> is the unit number in concern

# Using READ statement (cont'd)



- For unformatted (binary) READs from a sequential file, syntax is as follows :

**READ(iu) array**

- For unformatted direct access READs the syntax requires also a record number (rec)

**READ(iu,rec=78) array**

- You can READ & WRITE such files in any order, virtually at any record

# FORMAT descriptors

- To prettify output and to make it human readable, use FORMAT descriptors in connection with WRITE-statement
- Can be used with READ as well as to input data at fixed positions and using predefined field lengths
- Use either through FORMAT-statements, CHARACTER variable or embedded in READ / WRITE fmt-keyword

# FORMAT descriptors...

w=number of characters to use, d=number of digits to the right of decimal point, m=minimum number of characters to be used, e=number of digits in the exponent. Variables: Integer :: J, Real :: R, Character :: C, Logical :: T

Data type	Basic data edit descriptors	Example	
Integer	Iw, Iw.m	I5, I5.3	WRITE(*,'(I5)') J WRITE(*,'(I5.3)') J
Real (decimal and exponential forms)	Fw.d Ew.d, Ew.dEe	F7.4 E12.3, E12.3E4	WRITE(*,'(F7.4)') R WRITE(*,'(E12.3E4)') R
Character	A, Aw	A, A20	WRITE(*,'(A)') C
Logical	Lw	L7	WRITE(*,'(L7)') T

# FORMAT descriptors...



## Control edit descriptors:

Variables: Integer :: I, J

(n = number of characters)

Task	Descriptor	Example
New line	/	<code>write(*,'(I5,/,I5)') I, J</code>
Tabbing (absolute)	Tn	<code>write(*,'(I5,T20,I5)') I, J</code>
Tabbing (relative,right)	TRn	<code>write(*,'(I5,TR5,I5)') I, J</code>
Tabbing (relative,left)	TLn	<code>write(*,'(I5,TL3,I5)') I, J</code>
Number of blanks	nX	<code>write(*,'(I5,5X,I5)') I, J</code>
Do not read blanks	BN	<code>read(*,'(BN,I5)') I</code> (default way)
If blanks -> zeros	BZ	<code>read(*,'(BZ,I5)') I</code>
Switch on plus sign	SP	<code>write(*,'(SP,I5)') I</code>
Switch off plus sign	SS	<code>write(*,'(SP,I5,SS,I5)') I, J</code>

# FORMAT descriptors...



Complex number case, give data format for both parts:

Variable: Complex :: Z

Example: WRITE(\*,'(F6.6,E9.2)') Z

It is possible that an edit descriptor will be repeated a specified number of times.  
For example

-five integer values: WRITE(\*,'(5I8)')

-three times an integer-real value pair: WRITE(\*,'(3(I5,F8.3))')

Matrix style (2D), row by row, output example (4x3 matrix):

```
INTEGER :: j
```

```
INTEGER, PARAMETER :: ind1=4, ind2=3
```

```
REAL, DIMENSION(ind1,ind2) :: R
```

```
DO j=1,ind1
```

```
  WRITE(*,'(3(F7.3,1X))') R(j,:)
```

```
! WRITE(*, 100) R(j,:)      ! same as above, see FORMAT statement below
```

```
END DO
```

```
! 100 FORMAT (3(F7.3,1X))
```

# Contents



- Open / Close / Inquire - functions
- Read / Write / Format - statements
- **File types (text, binary, direct access)**
- NAMELIST I/O
- Internal I/O

# Formatted vs. unformatted files

- Text or **formatted files** are
  - Human readable
  - Portable i.e. machine independent
- Binary or **unformatted files** are
  - Machine readable only
  - Much faster to access than formatted files
  - Suitable for large amount of data due to reduced file sizes

# Formatted vs. unformatted files

- Binary or **unformatted files** are (cont'd)
  - Internal data representation used for numbers, thus no number conversion, no rounding of errors compared to formatted data
  - Not necessarily portable (need to have the same internal data representation between machines that are writing / reading such files)

**Beware of BIG- & LITTLE-endians !!**

# Examples of unformatted I/O



- Write to a *sequential binary file*

```
REAL rval
CHARACTER(len = 60) string
OPEN(10, file='foo.dat', form='unformatted')
WRITE(10) rval
WRITE(10) string
CLOSE(10)
```

- No FORMAT-descriptors (FMT= keyword)
- Reading is done similarly :

```
READ(10) rval
READ(10) string
```

# Examples of unformatted I/O



- In case of *direct access files*, format is by default binary (unformatted)
  - Override via **FORM='formatted'** options
- You can read and write data in any order, unlike in sequential files
- Used for database accesses
- Record length needs to be known in **OPEN**

# Examples of unformatted I/O



- The record length is given using **recl** – keyword in **OPEN**
  - # of characters for formatted files
  - # of bytes for unformatted files (beware: on some machines it is the # of **words** !!)
- The record number is given using **rec** – keyword in **read** / **write** -statements
  - Record numbering starts from 1

# Examples of unformatted I/O



- An example of (binary) direct access I/O

```
REAL, dimension(10) :: r
INTEGER :: iu = 20
OPEN(iu,file='data.db',access='direct',recl=40)
READ(iu,rec=33) r
CALL update(r)
WRITE(iu,rec=33) r
CLOSE(iu)
```

# Contents



- Open / Close / Inquire - functions
- Read / Write / Format - statements
- File types (text, binary, direct access)
- **NAMelist I/O**
- Internal I/O

# NAMELIST I/O

- NAMELIST is a handy mechanism to input data values using directly Fortran (*variable, value*)-pairs through NAMELIST– group :

```
REAL :: mondeo=1.8, volvo=2.3, honda=2.0
NAMELIST /volume/ mondeo, volvo, honda
READ(*,NML=volume)
WRITE(*,*) 'Car engine sizes : '
WRITE(*,NML=volume)
```

# NAMELIST I/O (cont'd)

- NAMELIST input begins with &-character and group name, ends with /-character :

```
&volume mondeo=2.0, honda=2.2 /
```

- Output

```
&VOLUME
```

```
MONDEO=2.0, VOLVO=2.3, HONDA=2.2, /
```

# Contents



- Open / Close / Inquire - functions
- Read / Write / Format - statements
- File types (text, binary, direct access)
- NAMELIST I/O
- **Internal I/O**

# Internal I/O

- Often it is necessary to filter out data from a given character string
- Or to pack values into a character string
- For these situations Fortran internal I/O with **READ** / **WRITE** becomes handy
- No actual (physical) files are used

# Internal I/O : An example

```

character(len=8), :: tstamp = '20100613'
integer :: yr, mon, day
character(len=3), parameter :: mons(12) = (/ &
'Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', &
'Jul', 'Aug', 'Sep', 'Oct', 'Nov', 'Dec' /)
character(len=11) :: pretty
READ(tstamp, fmt='(i4,i2,i2)') yr, mon, day
WRITE(*,*) 'yr, mon, day = ', yr, mon, day
WRITE(pretty, fmt='(i2.2,"-",a3,"-",i4)') &
    day, mons(mon), yr
WRITE(*,*) pretty

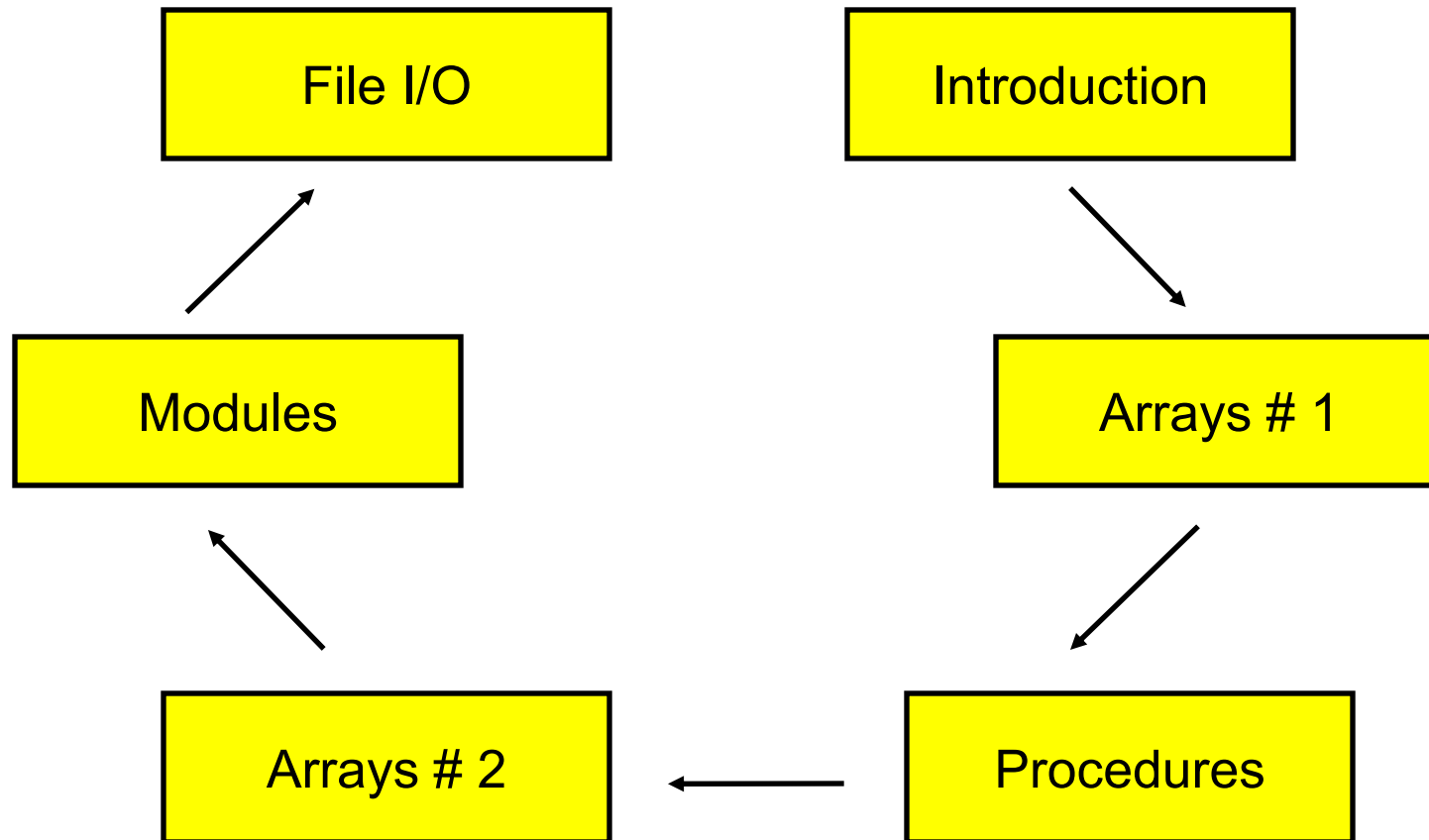
```

# Summary



- Communication between a program and outside world (disk files).
- Data reading
- Data writing

# Fortran 95 course

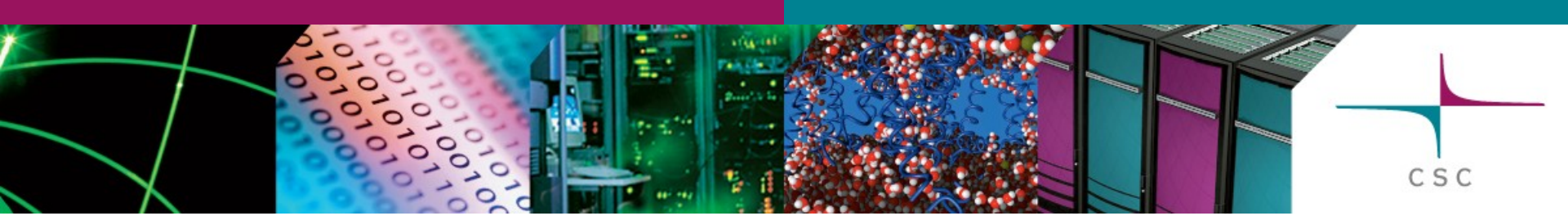


# The goal

- ✓ Attendees should be able to write small Fortran 95 programs

# References

- Get CSC's Fortran95/2003 Guide for free
  - <http://www.csc.fi/csc/julkaisut/oppaat>
  - In Finnish
- GNU Fortran online documents
  - <http://gcc.gnu.org/onlinedocs/gcc-4.5.0/gfortran>
- Others
  - Search from IBM & Cray websites for example



# Fortran95

Tommi Bergman  
Jarmo Pirhonen  
Sami Saarinen

CSC – IT Center for Science Ltd.  
Espoo, Finland

CSC – Tieteen tietotekniikan keskus Oy  
CSC – IT Center for Science Ltd.