C2115 Practical Introduction to Supercomputing

4th Lesson

Petr Kulhánek, Jakub Štěpán

kulhanek@chemi.muni.cz

National Centre for Biomolecular Research, Faculty of Science
Masaryk University, Kotlářská 2, CZ-61137 Brno

esf european social fund in the czech republic
EUROPEAN UNION
MINISTRY OF EDUCATION, YOUTH AND SPORTS
OP Education for Competitivness
UNIVERSITAS MASARYKIANA BRUNENSIS

INVESTMENTS IN EDUCATION DEVELOPMENT

CZ.1.07/2.2.00/15.0233

Obsah

- **Fortran**
  Fortran language history, Hello world!, compilers, compiling, compiler options
- **Syntax**
  program, F77 differences, variables, control structures, I/O, arrays, functions, procedures
- **Exercise**
  simple programs, matrix multiplication, definite integral calculation
- **Literature**

Fortran

- Fortran language history
- Hello world!
- Compilers, compiling, compiler options

History

Fortran (abbreviation of FORmula and TRANslator) is imperative programing language, that was designed in the twenties of 20th century by IBM for **scientific calculations and numerical applications**.

Source: wikipedia

Language versions:

Fortran 77

Fortran 90

Fortran 95

Fortran 2003

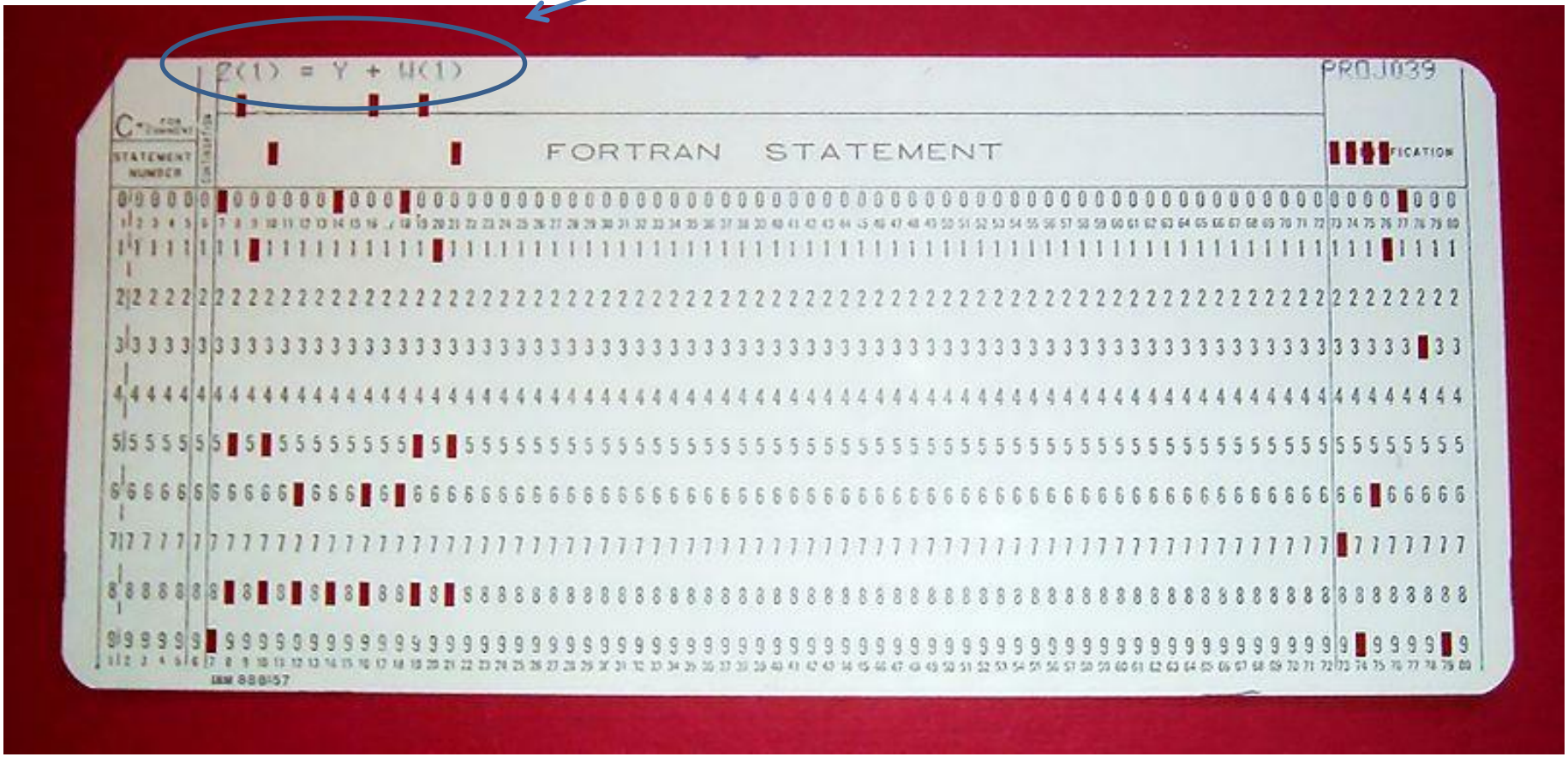
Fortran 2008

Number of libraries in Fortran of for Fortran exist. Compilers are able to produce **heavily optimized code**.

Standard mathematical libraries: BLAS, LAPACK and more on http://www.netlib.org

History

One source code



Zdroj: wikipedia

Hello world!

hello.f90

```
program Hello

write(*,*) 'Hello world!'

end program
```

Compilation:

```
$ gfortran hello.f90 -o hello
```

Running:

```
$ ./hello
```

Compiling to assembler:

```
$ gfortran hello.f90 -S
```

```
hello.s
```

Exercise LIII.1

1. Create file hello.f90. Compile it by gfortran compiler. Make sure the program works as you expect.

Compilation

GNU GCC

Compiler: **gfortran**
License type: GNU GPL (freely available)
URL: http://gcc.gnu.org/wiki/GFortran

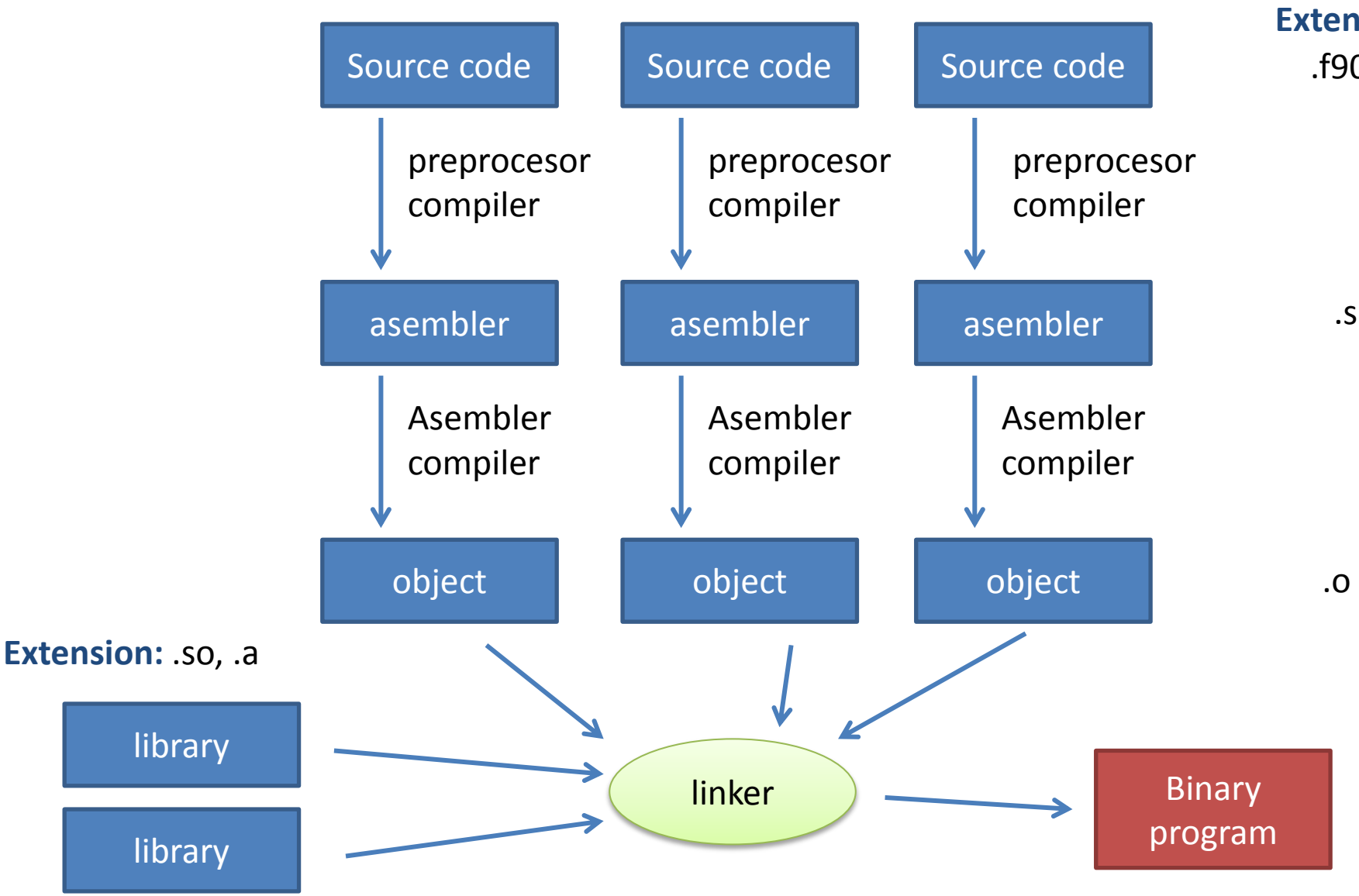
Intel® Composer XE

Compiler: **ifort**
License type: a) commercial (in MetaCenter, meta module: intelcdk)
b) free for private usage after registration (linux)
URL: http://software.intel.com/en-us/articles/intel-composer-xe/

The Portland Group

Compiler: **pgf90**, **pgf77**
License type: commercial (in MetaCenter, meta module: pgicdk)
URL: http://www.pgroup.com/

Compiling …


Extension:
.f90
Source code
Source code
Source code
preprocesor compiler
preprocesor compiler
preprocesor compiler
asembler
asembler
asembler
.s
Asembler compiler
Asembler compiler
Asembler compiler
object
object
object
.o
Extension: .so, .a
library
library
linker
Binary program

Compiler options

Compiler options:

-o	output program name
-c	translates only to object code
-S	translates only to object assembler
-Ox	optimization level, where x=0 (no), 1, 2, 3 (high)
-g	inserts additional information and code for debugging (slow down program run)
-lname	links library with *name* to program
-Lpath	path to libraries that are not in standard path

Compiler options (ifort):

-trace all	checks all arrays ranges, usage of non-initialized variables, etc.

Fortran written programs

Gaussian

http://www.gaussian.com/

Commercial program dedicated to quantum chemical calculations.

AMBER

http://www.ambermd.org/

Academic software dedicated to molecular simulations using molecular mechanics and hybrid QM/MM methods. Programs **sander** and **pmemd** are written in Fortran.

CPMD

http://www.cpmd.org/

Academic software dedicated to molecular simulations using methods of density functional theory.

Further software: Turbomole, DALTON, CP2K, ABINIT and more …
http://en.wikipedia.org/wiki/List_of_quantum_chemistry_and_solid_state_physics_software

Syntax

- program, F77 differences
- Variables
- Control structures
- I/O
- Arrays
- Functions, procedures

F77 dialect

- Fixed format
- column 1, if starts with C then line is comment
- column 1-6 is dedicated for signaling (for I/O formats, cycles)
- column 6, if contains symbol * then it is previous line continuation
- column 7-72 is program line

```
123456789012345678901234567890123456789012345678901234567890123456789

C this is comment
      implict none
      real       f
      integer    a, b
C --------------------------
C sum numbers a and b
      a = a + b
C long line
      f = a*10.0 + 11.2*b
     *+ (a+b)**2
100   format(I10)
      write(*,100) a
```

Source codes

- Fortran 90 and later uses free syntax (commands do not need to be aligned to columns as in Fortran 77).
- Possible extensions of source code files: .fpp, .**f90**, .f95, .f03, .f08
- Fortran is **not** case-sensitive
- To align text tabulator is appropriate.
- Comments may begin anywhere and are to be introduced by exclamation mark !.
- Maximum line length is limited (usually 132 chars). Ampersand & is use to extend lines.

```
implicit none
real           :: f
integer        :: A, B
! ---------------------------
! Sum numbers A and B
A = A + B
f = A*10.0 + 11.2*B &
    + (A+B)**2     ! Long line
```

Preprocessor

- Source code may contain CPP preprocessor directives (used by C and C++ languages):

#include <file>
#include “file"
#ifdef
#ifundef
#if
#else
#endif
#define
And more ...

- File processing by preprocessor may be forced by compiler option, or by change of file extension to: .fpp, .FPP, F90, .F95, .F03, .F08

http://gcc.gnu.org/onlinedocs/gfortran/Preprocessing-Options.html

Program section

```
program Hello
! Variable definition

! Program itself
write(*,*) 'Hello world!'

! Program end
end program
```

Program processing direction

Program may be stopped prematurely by command **stop**.

Variables

```
implicit none

logical                 :: f
integer                 :: a, g
real                    :: c, d
double precision        :: e
character(len=30)       :: s
```

Automatic variable declaration is switched of

Real number in single precision

Real number in double precision

String (text)

Maximum length of string in characters

Alternatives:

```
real(4)         :: c, d
real(8)         :: e
```

Variables are always defined on program, function or procedure beginning.

Variables

```
implicit none
logical                 :: f
!-----------------------
f = .TRUE.
write(*,*) f
f = .FALSE.
write(*,*) f
```

```
implicit none
real              :: a,b
!-------------------------
a = 1.0
b = 2.0
b = a + b
write(*,*) a, b
```

```
implicit none
character(len=30)    :: s
!-----------------------
s = 'sample text'
write(*,*) trim(f)
```

Variables has to be initialized (i.e. assign default value).

function **trim** crop string from one side(removes empty chars)

Variables

```
implicit none
real            :: a = 1.0
real            :: b
!-----------------------
b = 2.0
b = a + b
write(*,*) a, b
```

NEVER initialize variable on same line as is declaration.

Permitted syntax, that is translated as:

```
real,save       :: a = 1.0
```

Analog to keyword "**static**" in C and C++ language.

Math operations

Operators:

+	addition
-	subtraction
*****	multiplication
/	division
******	power

Un-direct support:

MOD(n,m) modulo (**n % m** in language C)

```
real            :: a, b, c
!-------------------------
a = 1.0
b = 2.0
c = 4.0
b = a + b
b = a * b / c
c = a ** 2 + b ** 2
```

Cycles I

```
do variable = start_value, end_value [, step]
        command1
        command2
        ...
end do
```

Variable is only **whole number (integer)**.

```
integer                    :: i
!--------------------------
do i = 1, 10
   write(*,*) i
end do
```

Prints: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10

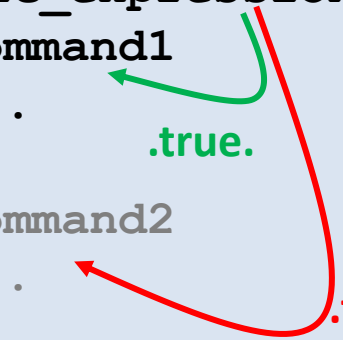
```
integer                    :: i
!--------------------------
do i = 1, 10, 2
   write(*,*) i
end do
```

Prints: 1, 3, 5, 7, 9

There exist commands **cycle** (continue in C language) and **exit** (break).

Conditions

```
if ( logic_expression ) then
        command1
        ...
else
        command2
        ...
end if
```




```
integer                 :: i = 7
!-------------------------
if( i .gt. 5 ) then
   write(*,*) 'i is greater then 5'
end if
```

Logic operators:

.and.	Logic yes
.or.	Logic or
.not.	Negation

Comparison operators (numbers):

.eq.	Equal to
.ne.	Not equal to
.lt.	Less then
.le.	Less then or equal
.gt.	Greater then
.ge.	Greater then or equal

Comparison operators (logic):

.eqv.	Equivalence
.neqv.	Non-equivalence

Cycles II

```
do while ( logic_expression )
       command1
       command2
       ...
end do
```

Cycle still iterates if **logic_expression** returns **.true.**

```
double precision     :: a
!-------------------------
a = 0.0
do while ( a .le. 5 )
   write(*,*) a
   a = a + 0.1
end do
```

Prints numbers from 0 to 5 with step 0.1

There exist commands **cycle** (continue in C language) and **exit** (break).

Functions and procedures

Function is part of program, that may be called **repeatedly** from various positions in code. **Procedure** is similar to function, difference is that **procedure does not return any value**. Using procedures and functions makes program more understandable and reduces code duplicity.

```
program Hello
! Variable definition
…
! Program itself
! Function or procedure call
…
! Program end
…
contains

! Function or procedure definitions

end program
```

Functions and procedures may be called from program itself or from functions and procedures as well.

Function and procedure **arguments** are **passed by reference**.

Function definition

```
function my_function(a,b,c) result(x)
implicit none
double precision  :: a, b, c  ! Arguments (parameters)
double precision  :: x        ! Function result
!---------------------
 integer          :: j        ! Local variable
!-------------------------------------------------------
! Function body
x = a + b + c
end function my_function
```

Alternative:

```
double precision function my_function(a,b,c)
...
my_function = a + b + c
end function my_function
```

Procedure definition

```
subroutine my_procedure(a,b,c)
implicit none
double precision  :: a, b, c  ! Arguments (parameters)
!---------------------
 integer              :: j         ! Local variable
!-------------------------------------------------------
! Procedure body
a = a + b + c
end subroutine my_procedure
```

Access permissions of function and procedure arguments may be adjusted by keyword **intent**. Default access permission is intent(**inout**).

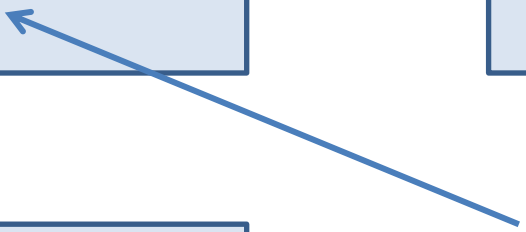
```
double precision, intent(in)      :: a      ! Argument may be read
double precision, intent(out)     :: b      ! Argument may be written
double precision, intent(inout)   :: c      ! Argument may be read and written
```

Function and procedure calls

Function call:

```
double precision  :: a
double precision  :: d
!--------------------------
a = 5.0
d = my_function_2(a)
write(*,*) d
```

```
double precision  :: a
double precision  :: d
!--------------------------
a = 5.0
my_function_2(a)
```



Result **has to be used**.

Procedure call:

```
double precision  :: a
double precision  :: d
!--------------------------
a = 5.0
d = 2.0
call my_procedure_3(a,d)
```

Passing arguments by reference

```
double precision  :: a
double precision  :: d
!--------------------------
a = 5.0
d = 2.0
write(*,*) d
call my_procedure_3(a,d)
write(*,*) d
```

2

?

```
subroutine my_procedure_3(a,b)
implicit none
double precision  :: a, b      ! Arguments (parameters)
!-----------------------------------------------------
! Procedure body
b = a + b
end subroutine my_procedure_3
```

Passing arguments by reference

```
double precision  :: a
double precision  :: d
!--------------------------
a = 5.0
d = 2.0
write(*,*) d
call my_procedure_3(a,d)
write(*,*) d
```

2

7

In C language value would be 2.

```
subroutine my_procedure_3(a,b)
implicit none
double precision  :: a, b      ! Arguments (parameters)
!-----------------------------------------------------
! Procedure body
b = a + b
end subroutine my_procedure_3
```

Some standard functions & procedures

Math functions:

sin(x)	
cos(x)	
sqrt(x)	root
exp(x)	
log(x)	**natural** logarithm
log10(x)	decadic logarithm

Random numbers:

call **random_seed**() initializes random numbers generator

call **random_number**(number) sets variable **number** to random number from range <0.0;1.0)

Time measures:

call **cpu_time**(time) sets variable **time** to program runtime in seconds (with microsecond resolution)

Arrays

Static defined arrays:

```
double precision :: a(10)
double precision :: d(14,13)
```

Single dimension array with 10 items.

Double dimension array with 14x13 items.
(14 rows and 13 columns)

Dynamic declared arrays:

```
double precision,allocatable      :: a(:)
double precision ,allocatable     :: d(:,:)
! ------------------------------------------------
! Memory allocation for array
allocate(a(10000), d(200,300))
! Array usage
! Memory dislocation
deallocate(a,d)
```

Single dimension array.

Double dimension array.

Array sizes may be defined using integer variables.

Array usage

```
double precision :: a(10)
double precision :: d(14,13)
integer          :: i
!---------------------------------

a(:) = 0.0    ! Same as a = 0.0

do i=1, 10
      write(*,*) i, ' - th item of array is ', a(i)
end do

a = d(:,1)  ! Writes first column of
            ! matrix d to vector a

a(5) = 2.3456
d(1,5) = 1.23
write(*,*) d(1,5)
```

Array items are indexed from 1.*

Array size may be checked by function **size**.

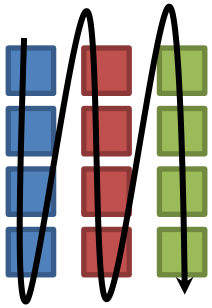
* Ranges for particular sizes may be changed.

Arrays – memory model

Fortran

a(i,j)

Following items are in columns (column based).

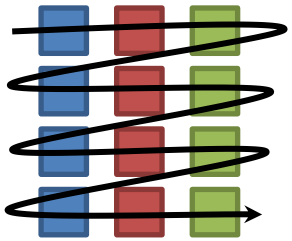


Matrix items order in memory.

C/C++

A[i][j]

Following items are in rows (row based).



If we call functions from libraries BLAS or LAPACK it is necessary to take into account different array indexing model.

Array – memory model

Fortran

```
double precision :: d(10,10)
double precision :: sum
integer          :: i,j
!-------------------------------

sum = 0.0d0
do i=1, 10
      do j=1,10
            sum = sum + d(j,i)
      end do
end do
```

index is changed fastest in rows

C/C++

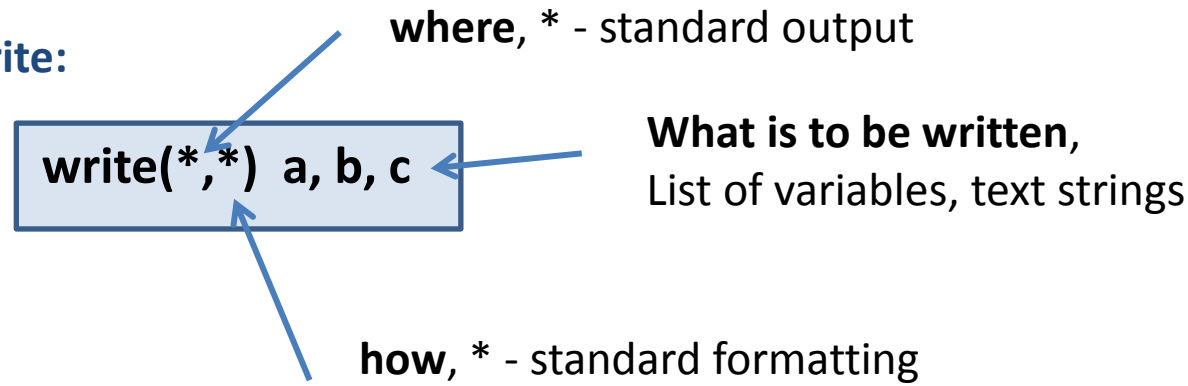
```
double* d[];
double   sum;
//-------------------------------

sum = 0.0;
for(int i=0; i < 10; i++){
      for(int j=0; j < 10; j++){
            sum += d[i][j];
      }
}
```

index is changed fastest in columns

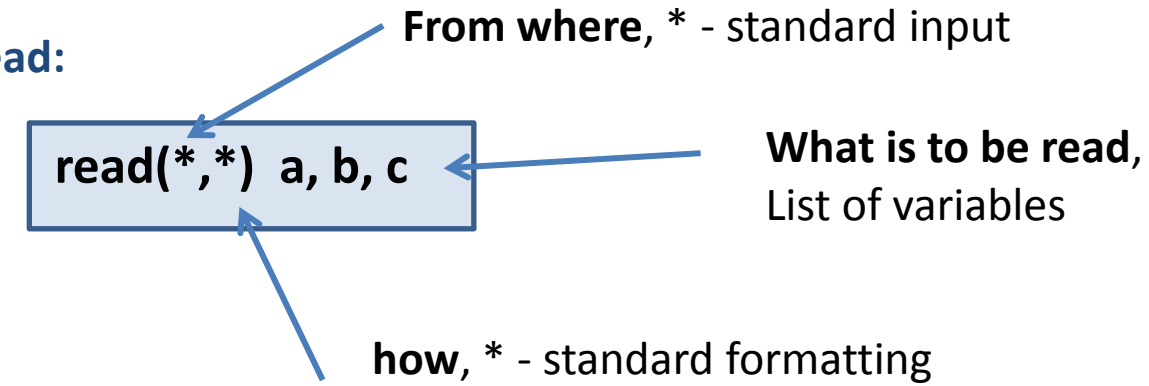
Note: difference does not influence function, but processing speed only

I/O operations

Data write:



Data read:



Files are opened by command **open**. Close by command **close**.

I/O operations - formating

Formated output:

```
write(*,10)  a, b, c
10 format('Value a=',F10.6,' Value b=',F10.6, ' Value c=',F10.6)
```

- Format may be given before or after command write or read
- Formatting types:
  - F – real number in fixed format
  - E – real number in scientific (exponential) format
  - I – whole number
  - A - string

Output with no new line:

```
write(*,10,ADVANCE='NO')  a, b, c
```

Format has to be specified

Further language properties

1. Pointer support
2. Structures
3. Object oriented programming

Exercise

All data and source codes of solved exercises should be in separate directories. It will be used in further course parts.

Exercise LIII.2

1. Write program, that prints ten 'A' symbols to terminal, each on separate line.
2. Write program, that prints ten 'A' symbols to terminal next to each other on one line.
3. Write program, that prints 'A' symbols to terminal into shape of right angled triangle.
4. Write program, that prints 'A' symbols to terminal into shape of square.

Exercise LIII.3

1. Write program, that dynamically allocates two dimensional array **A** of size **n** x **n**. Items will be initialized by random numbers from range <-10 ;20>. Print array to terminal.
2. Create two separate arrays (matrices) **A** and **B** of size **n** x **n**. Initialize arrays in same way as in previous exercise. Write code for matrix **A** and **B** multiplication, save result to matrix **C**.
3. How many floating point operations will be done during matrix multiplication? Measure time necessary for matrix multiplication (do not include matrix initiation and creation). Calculate approximate processor power in MFLOPS from operation number.
4. Calculate processor performance for different matrix **A** and **B** sizes. Create graph for values of **n** in range 10 to 1000.

Exercise LIII.4

1. Write program, that calculates definite integral given at bottom. Use trapezoid method.

$$I = \int_0^1 \frac{4}{1 + x^2} dx$$

2. What is integral value? What is reason?

Home work

http://summerofhpc.prace-ri.eu/

1. Solve first **code test** task.

Literature

- http://www.root.cz/serialy/fortran-pro-vsechny/
- http://gcc.gnu.org/onlinedocs/gfortran/
- Dokumentace ke kompilátoru ifort
- Clerman, N. S. Modern Fortran: style and usage; Cambridge University Press: New York, 2012.