



Fortran 95, Exercises

Introductory course, 8-9 Oct 2009

Raimo Uusvuori, Tommi Bergman, Jarmo Pirhonen, Sami Saarinen

`firstname.lastname[at]csc.fi`

CSC - IT Center for Science Ltd.

CSC - Tieteen tietotekniikan keskus Oy

Espoo, Finland

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

Practical arrangements

Computers used

- The computing server `hippu.csc.fi` (Hippu) and the classroom PC workstations running Linux are used for exercises. For `hippu.csc.fi` there are user ids `trng01...trng20`. Passwords are given during the course.
- You can login to `hippu.csc.fi` using the SSH protocol from the terminal window of a PC workstation or from your own laptop via LAN connection, e.g.:

```
% ssh -X hippu.csc.fi -l trng07
```

Supplementary material

- You can find supplementary material of the course from the directory `$DOC/f90/examples` of `hippu.csc.fi` (`/fs/app1/gen/courses/f90/examples`) and in the subdirectory `f90/examples` of the workstation.
- In Hippiu copy example programs from the directory `$DOC/f90/examples` to your own storage when needed.
- The example answers will be available on-line when the course ends. The exercises and answers can also be found later from the course material:

http://www.csc.fi/english/csc/courses/archive/fortran_f2009

About exercises

- The course will follow closely the CSC's guide *Fortran 95/2003* (only in Finnish). See:
<http://www.csc.fi/csc/julkaisut/oppaat>
<http://www.csc.fi/csc/julkaisut/oppaat/pdfs/fortran2003>
- The number of the exercise inside the parenthesis refers to the numbering of the exercise in the Guide *Fortran 95/2003*.

Hands-on Exercises

- Recommended Fortran exercises for the hands-on exercises of the course:
 - hands-on exercise: A (Fortran 95 overview): exercises 1, 2, 4
 - hands-on exercise: B (Functions and subroutines, File I/O and format descriptors): exercises 3, 5, 6
 - hands-on exercise: C (Dynamic memory allocation and arrays as arguments): exercises 7, 8, 9, 10
 - hands-on exercise: D (Modules and derived data types): exercises 11, 12, 13
 - extra exercises: (Operators, Conversion to Fortran95/(2003)): exercises 14, 15, 16

Fortran compiler used

- The GNU Compiler Collection (GCC) Fortran 95 compiler `gfortran` is used. It supports also FORTRAN 77 and Fortran 90. The newest versions implement also some or all Fortran 2003 and Fortran 2008 features. The version available on Linux workstations is 4.1.2 and the newest version on Hippu is 4.3.2 (default).
- The GCC online documentation of the latest versions: <http://gcc.gnu.org/onlinedocs/>
- On Linux workstations the command `gfortran` can be used directly to launch the GCC Fortran compiler, but on Hippu this command should be used only after its module for a desired GCC version is loaded (see later).

Other compiler commands and compilers

- On Linux workstation there may be the command `£95` and/or `£90` which refer to the command `gfortran`. On Hippu these commands refer to the Fortran compiler command of the loaded programming environment. In the case of the GCC (GNU) environment these refer to `gfortran`.
- In addition PGI (it is the default when logging on Hippu), PathScale and Intel compilers and their programming environments are available on Hippu, see Hippu User's Guide:
http://www.csc.fi/english/pages/hippu_guide

How to use the modules on Hippu

- On Hippu the program execution and development environment is managed by using the environment modules package. For example, user's shell (`/bin/tcsh`) environment variables for different software packages, compiler suites and libraries are set up by using the module tool.
- Currently loaded modules are listed with command `module list`
- All available modules are listed with command `module avail`
- The environment changes induced by a particular module, *e.g.*, `gcc`, are shown with the command `module show gcc`

Loading modules on Hippu

- Loading and unloading module files is done with module commands `load` and `unload`. Changing from the default PGI programming environment to the default GNU (GCC) programming environment is accomplished with the command

```
module swap PrgEnv-pgi PrgEnv-gnu
```

- Among other things, this loads the default GCC compiler module `gcc`, which modifies user's command search path `$PATH` so that `f90` and `f95` commands point to GNU `gfortran` instead of `pgf90` and `pgf95`.

More about modules and compilers

- The module commands `list`, `avail [module]` and `show [module]` display module names and their versions in the format like `gcc/4.3.2(default)` and `PrgEnv-pgi/9.0.3` and, which is the default, if any. The operation of the `load`, `unload` and `swap` module commands apply to the default versions, if a version number is not given, and to the specified version, if the version number is given.
- More information about Hippu module environment and usage of compilers see:
http://www.csc.fi/english/pages/hippu_guide/using_hippu/modules
http://www.csc.fi/english/pages/hippu_guide/program_development/index.html

Emacs editor, f90 and fortran modes

- For editing Fortran program files you can use Emacs editor without (the option `-nw`) or with X-windows:

```
emacs -nw test.f90  
emacs prog.f90
```

- On Hippu the **F90** (`.f90` files) or **Fortran** (`.f` files) mode is on by default. You must however use the corresponding menu for indentation, highlighting and so on. On some Linux workstations it may be necessary first load f90 lisp library: `Esc X load-library` and then `f90`. Indentation and highlighting works then by default.

Simple compilation

● Example program test.f90:

```
PROGRAM Test
  IMPLICIT NONE
  INTEGER, PARAMETER :: result = 42
  ! Simple output statement...
  PRINT *, 'Hello world!'
  PRINT *, 'The result is ', result
END PROGRAM Test
```

● Copying to home directory, compilation and execution:

```
% cp $DOC/f90/examples/test.f90 .
% gfortran test.f90 -o test
% ./test
Hello world!
The result is          42
```

Exercise 1: Input format of the code

1. (4.1) What means the following (free format) code line

```
A = 0.0 ; B = 370 ! Initialization ; C = 17.0 ; D = 33.0
```

2. (4.2) Is the following syntactically correct Fortran 90 code?

```
Y = SIN(MAX(X1,X2)) * EXP(-COS(X3)**I) - TAN(AT  
& AN(X4))
```

3. (4.3) Is the following declaration correct?

```
REAL :: real; REAL :: price
```

4. (4.5) Are the following Fortran statements written correctly?

```
character_string = 'Awake in the morning,  
                  & asleep in the evening.'  
x = 0.4-6  
answer = 'True & false'  
low-limit = 0.0005E10  
y = E6
```

Exercise 2: Declaration of variables

1. (5.1) Are the following declarations legal in Fortran:

```
DOUBLE :: x
```

```
CHARACTER(LEN=*), PARAMETER :: "Name"
```

```
REAL :: pi = 22/7
```

```
REAL :: x = 2., y = -3
```

```
REAL :: pi = 22.0/7.0
```

```
REAL x = 1.0
```

2. (5.2) Declare the constant κ , whose value is 0.75.

3. (5.5) Find out, what are the ranges of the values of integer and real numbers, which the Fortran compiler you use can handle. Hint: study the values which the following subprogram calls return

```
SELECTED_REAL_KIND  
SELECTED_INT_KIND
```

e.g., by the following way:

```
PROGRAM kindtest  
  IMPLICIT NONE  
  INTEGER :: i, ikind  
  DO i = 1, 24  
    ikind = SELECTED_INT_KIND(i)  
    WRITE (*,*) 'i = ', i, ' kind = ', ikind  
  END DO  
END PROGRAM kindtest
```

Please, note that the values of the kind parameter depend on the Fortran compiler used!

4. (5.6) Print the value of the statement

$$\frac{1}{3}\ln(1 + \sqrt{2}) + \frac{1}{15}(2 + \sqrt{2}) \approx 0.5214 \dots$$

using at least the precision of 6 decimals. Use the function calls `LOG` and `SQRT`. In addition, print the value of the statement using the highest precision of real numbers which the compiler knows.

This slide is intentionally left empty.

Exercise 3: Input and output

1. (12.2) In the following there are declared Fortran format statements in various ways. What the program does? Which declarations are most functional?

```
PROGRAM form
  IMPLICIT NONE
  REAL :: x
  CHARACTER(LEN=11) :: form1
  CHARACTER(LEN=*), PARAMETER :: &
    form2 = '(F12.3,A)'
  x = 12.0
  form1 = '(F12.3,A)'
  WRITE (*, form1) x, ' hello '
  WRITE (*, form2) 2*x, ' hi '
  WRITE (*, '(F12.3,A)') 3*x, ' hi hi '
END PROGRAM form
```

2. (12.3) Write output statements for the following arrays:

```
REAL, DIMENSION(6,10) :: a
```

```
INTEGER, DIMENSION(24) :: h
```

```
CHARACTER(LEN=10), DIMENSION(6) :: marray
```

```
LOGICAL, DIMENSION(2) :: condition
```

```
COMPLEX, DIMENSION(20) :: z
```

3. (12.4) Write list directed input routines, which read values for the arrays of the previous exercise. Give an example of your program's input file.

Exercise 4: Control structures

1. (7.3) What are the iteration counts of the following DO loops, the values of loop variable i inside the loop, and the value of the loop variable after the DO construct?

```
DO i = 1, 5
```

```
DO i = 5, 0, -1
```

```
DO i = 10, 1, -2
```

```
DO i = 0, 30, 7
```

```
DO i = 3, 2, 1
```

2. (7.2) Write a `SELECT CASE` structure, which does different operations when an integer variable is negative, it is zero, or it is one of the prime numbers (3, 5, 7, 11, 13). In other cases nothing is done.

3. (7.4) Write the DO loop, which sums the square roots of 100 given numbers, excluding negative numbers, and stops summing if the given number is zero. Make two versions so that one uses a CYCLE statement whereas another one does not use it.
4. (7.6) Print the transformation table for converting inches to millimeters ($1'' = 25.4\text{mm}$). Start from the value 0.0 and print values at half inch intervals up to 12 inches.

Exercise 5: Declaration of procedures

1. (13.3) Write a subprogram, which transforms the polar coordinate presentation (r, ϕ) to the Cartesian coordinates (x, y) . Write also a subprogram which does the reverse transformation. Use the definitions

$$\sin \phi = y/r, \quad \cos \phi = x/r, \quad r = \sqrt{x^2 + y^2}$$

2. (13.7) Write a logical function numbers, which tells is a character string given as an argument composed only of numerals. Use the standard function `VERIFY` as an aid.
3. Write a function, which calculates the logarithm of the number x to the base b . Please, note:

$$\log_b x = \log_{10} x / \log_{10} b = \ln x / \ln b$$

Think about also, which types and ranges you allow for the arguments of the function!

4. (8.5) Write a recursive function for calculating the greatest common factor for two integers. Utilize the identities

$$\begin{cases} \text{gcf}(m, n) = \text{gcf}(n, \text{MOD}(m, n)), \\ \text{gcf}(m, 0) = m. \end{cases}$$

5. (8.7) Write a recursive function which calculates "Tribonacci numbers":

$$\begin{cases} x_1 = 1, & x_2 = 1, & x_3 = 1, \\ x_n = x_{n-1} + x_{n-2} + x_{n-3}. \end{cases}$$

Calculate x_{12} . Study behavior of the computation time as a function of n by using, for example, the standard program `SYSTEM_CLOCK` or by measuring it with the Unix/Linux command `time`. Is the routine efficient? Carry out the computation also using a loop structure.

6. Write a function `degrees (f , x)` , which returns the value of standard functions `ASIN`, `ACOS` and `ATAN` as degrees and not as radians. Use these standard functions as arguments for the function `f` you declared.

This slide is intentionally left empty.

Exercise 6: Arguments of a procedure

1. (8.9) In one of the lectures concerning Arguments of a procedure there was introduced the routine `FUNCTION gauss` for integration of scalar function $f(x)$ using the Gauss integration formula

$$\int_a^b f(x)dx \approx \frac{b-a}{2} \left[f\left(\frac{a+b}{2} - \frac{b-a}{2\sqrt{3}}\right) + f\left(\frac{a+b}{2} + \frac{b-a}{2\sqrt{3}}\right) \right]$$

where the integration interval is $[a, b]$. Add optional arguments to the program so that

- if the lower integration limit a is not given, it is set to $a = 0$,
- if the upper integration limit b is not given, it is set to $b = a + 1$.

Test the program first using some standard function as the integrand.

2. Change the integrand to the (external) function declared by yourself. Specify its interface using the `INTERFACE` block.
3. *Additional exercise.* Extend the routine you declared above so that the user can define desired number of integration points using an optional argument.

Exercise 7: Arrays

1. (11.1) Is the following declaration correct?

```
REAL DIMENSION(1:3,2:3) :: aa
```

2. (11.2) Declare the integer array `iarray`, which contains 3 rows and 4 columns. Initialize the first row of the array with integers 1-4 in this order from left to right, the second line with integers 5-8, and the lowest line putting integer -2 to every column. Print `iarray` row by row – so that each output line contains the elements of one row of the array.
3. Build also the 3×8 integer array `bigarray`, where the first 4 columns are identical with the array `iarray` in the previous exercise, and the 4 last columns are obtained from the columns of the array `iarray` by multiplying them with the number 3 and adding 5 to the product. Use array syntax in statements.

4. Write a subprogram which can do automatically the operation described in the previous exercise. The smaller array is given to the subprogram as input and it must return the double sized array, whose elements are constructed as described above
5. (11.4) Declare two pointer variables. Set one of them to point to the one-dimensional array of real numbers

```
REAL, DIMENSION(10) :: x
```

and another one to the sixth element of the same array.

6. (11.5) Use a pointer variable to substitute in the array `x` of the previous exercise the value `-1.0` for the elements, whose ordinal number is even, and the value `1.0` for the odd elements. The values of the array must be thus accordant with the following list: `(1.0, -1.0, 1.0, -1.0, ...)`.

7. (11.6) Make a subprogram which returns the array of the desired size in a pointer variable. Give the size of the array as an argument for the subprogram. The subprogram could be used, e.g., by the following way:

```
CALL reserve(p,n)
```

Exercise 8: Array syntax

Modify the program `table1.f` (next slide and in the directory `$DOC/f90/example/`) so that the array syntax is used in it. The function `rand` appearing in the program returns a random number. The call `CALL RANDOM_NUMBER(x)` fills in the array `x` with random numbers, which are evenly distributed between the interval `[0, 1]`.

```
program table1
parameter (n=10)
real b(n), c(n,n), d(n,n)

do 10 i = 1,n
b(i) = rand()
  do 10 j = 1,n
  c(j,i) = rand()
10  continue
write (*,*) b
do 20 i = 1,n
do 20 j = 1,n
d(j,i) = 1.0-c(j,i)
20  continue
```

```
do 100 j = 2, n
a = b(j) - b(j-1)
do 100 i = 2, n
    c(i,j) = a * c(i-1,j) + d(i,j)
100 continue

print *, ' answer = ', c(n,n)
end
```

Exercise 9: Standard functions

Modify the program `table2.f` so that array syntax and standard functions are used in it. Compile the original program using `£77` and the new one using `£90`. Confirm, that they work similarly.

```
program table2
  implicit none
  real x(100,100), y(100), z(100)
  real a, s
  integer i,j,k
  do 1 j=1,100
    y(j)=0.1*j
    do 1 i=1,100
      x(i,j)=1.0+y(j)
1  continue
```

```
do 2 j=1,100
  s=0.0
  do 3 i=1,100
    s=s+x(i,j)
3    continue
    z(j)=sqrt(y(j))/s
2  continue
do 5 j=1,100
5  if (z(j).lt.0.001) z(j)=0.0
a=z(1)
k=1
do 6 i=2,100
  if (z(i).gt.a) then
    k=i
    a=z(i)
  endif
6  continue
write (*,*) a,k
end
```

Exercise 10: Standard procedures

1. (13.1) Solve all complex roots of the equation $x^n = 1$ (hint: $e^{2\pi k i} = 1, k = 0, 1, \dots$).
2. (13.4) Calculate the inner product of five element real type vectors using the standard function **DOT_PRODUCT**.
3. (13.5) Calculate the product $C = AB$ of the 5×4 and 4×3 matrices A and B using standard function **MATMUL**. Calculate also the product $C^T = B^T A^T$, where B^T is the transpose of the matrix B , etc.
4. (13.6) Find the location of the first numeral in the character string at most 80 characters long, which the user gives as an input, using the **SCAN** function.

This slide is intentionally left empty.

Exercise 11: Derived types

1. (10.7) Declare the derived type which can save the birth date in the following form:

```
21 01 1990
```

This derived type thus contains three integers, which have different `KIND` values: `SELECTED_INT_KIND(2)` and `SELECTED_INT_KIND(4)`.

2. (10.8) Add the field the for the name to the derived type of the previous exercise.

3. (10.9) Write the function, which returns the name and date in a character string in the following form:

```
Jaska Jokunen (01.01.1999)
```

Use the derived type constructed in the previous exercise.

4. (10.14) In the Chapter 3.12 (page 33) of the book “Fortran 95/2003” (you can read it online) there is a suitable derived type `kurssilainen` (= course participant) for saving the names (`nimi` = name) and marks (`arvosana` = mark, grade) of the students (= `oppilaat`). Declare the module, which contains operations needed to handle this kind of derived type. Declare the input and output operations for the student data and a procedure for calculating the average of the marks.

Exercise 12: Modules

1. Declare a derived type in the main program and a variable which is of this type. Initialize the variable and pass it to an external subroutine, which prints the components of the variable. Compile and run the program. What happens? Next, put the subprogram to a module and compile the program. What happens? If there was a problem, correct it.
2. Make the program, which sums up two real numbers in an external subprogram (function). Make a module, which declares the kind type parameter for double precision real number. Use a module for defining precision of the main program and the summing function.
3. Change the another addend to integer in the summing function. Compile and run the program. What happens? Put then the summing function a module and use it in the main program. Compile the program. What happens?

4. Implement the global `point` type presented on lectures (Derived types and modules) so that it contains two real numbers (x and y coordinates). Make a function, which sums two `points`. Try how the module works using it in a simple main program.
5. Object oriented programming. One idea in the object oriented programming is, that data structures can be used only with specified functions (*member functions* in C++ terminology) and data elements cannot be accessed directly. The interface of the data structure is accurately defined in this kind of programming, which reduce programming errors and make the localization of them easier.
 - Change the `point` type of the previous exercise so that the elements x and y cannot be accessed outside the module.

- Try initialization and printing of the components of a `point` type variable. What happens?
- Add the subprograms to the module for initialization and printing. Use a simple main program to test the operation of the module.

This slide is intentionally left empty.

Exercise 13: Generic procedures

1. This is continuation of object oriented programming introduced in the previous exercise. Modify `point` module so that initialization can be done both for integer and real numbers using the same subprogram call.
2. The module file `swapmod.f90` in the directory `$DOC/f90/examples/` contains the module `swap` which defines a generic subroutine `switch`, which can be used for the switching the values of real and character string arguments. Modify the module so that it works also with integer arguments. What about one and two dimensional array arguments? Use the main program `swap.f90` in testing.

Exercise 14: Operator

1. The resistances R_s and R_r of the resistors connected series or parallel are calculated using the formulas

$$R_s = R_1 + R_2,$$
$$1/R_r = 1/R_1 + 1/R_2.$$

Define operators for handling real number variables, which can be used as follows:

```
REAL :: r, r1, r2
r = r1 .series. R2
r = r1 .parallel. r2
```

2. Add the operator `+` to the module `point` for summing the elements of the points, and the operator `.distance.` for calculating the distance from the origin.

Exercise 15: F77 → F90/F95

1. Compile one of your own FORTRAN 77 programs with the Fortran 90/95 compiler using fixed form input. You can also use the example program `$DOC/f90/examples/avgar.f` .
2. Try to modify the program to use free form input of Fortran 90/95. Change the comment lines starting with the letter C or with the character * to start with the exclamation mark ! , etc.

3. What difference there is in the behavior of the following program when you use free or fixed form.

```
LOGICAL L
L = .FALSE.
Z = 0.0
Y = 1.0
IF (L) THEN Z = 1.0
ELSE Y = Z ENDIF
PRINT *, Y, Z
END
```

Correct the errors found in the program.

Exercise 16: Old Fortran-codes

Additional Exercise. The file `$DOC/f90/examples/fzero.f` contains an subprogram from the book Kahaner, Moler, Nash: *Numerical Methods and Software*. The code follows old Fortran 66 syntax. Clean it using features and structures of Fortran 90/95, if possible. Write for testing a small program which calls the subprogram.