– **Typeset by GMNI &** FoilTEX –

PROGRAMMING IN FORTRAN LANGUAGE

GMNI — GRUPO DE MÉTODOS NUMÉRICOS EN INGENIERÍA

Departamento de Métodos Matemáticos y de Representación
Escuela Técnica Superior de Ingenieros de Caminos, Canales y Puertos
Universidade da Coruña

GMNI - Grupo de Métodos Numéricos en Ingeniería
http://caminos.udc.es/gmni

Content

- Standard computer structure
- Algorithms, computer programs and programming languages
- Fortran language

Standard computer structure (I)

- A computer is a machine capable of storing data an processing them given a sequence of established instructions fed with the objective of acquiring some information.

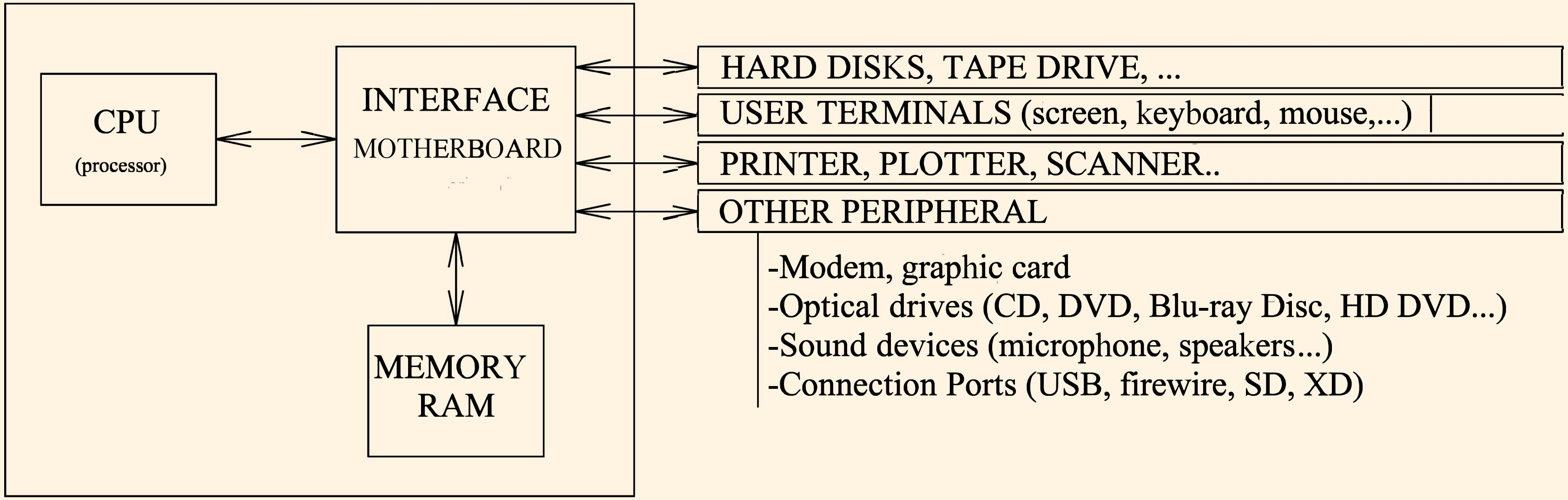
- The sequence of instructions (statements) is known as "program".

- Even though there are predecessor to computers, the actual computers emerged in the 1940s

- A computers is mainly composed of:
  - "Hardware": physical operating components (memory slots, motherboard, processor,...)
  - "Software": applications implemented on the "hardware" developed to perform a particular sequence of statements.

Standard computer structure (II)

- Hardware:



- CPU (Central Processing Unit): It is a computer part that executes the instructions of the software.
  - Logic unit that executes the operations with the data.
  - Control unit that interprets the sequence of commands and manages the associated devices.
- Main memory (RAM-"Random Access Memory"): manages the data and instructions being used by the processor at any given time.
  - It is a volatile memory.
  - Its access is faster than that of the secondary memory.

Standard computer structure (III)

- Secondary memory (Hard disk, ...): It is a non-volatile memory to storage data, software, ... Thus, its access is slower.
- Motherboard: It is the interface hat provides physical connection between all the hardware elements and the processor (CPU)
- Peripheral: external devices connected to the computer through the motherboard

  -Keyboard, mouse, monitor,...
  -Optical disc drive
  -Audio devices
  -...

▶ Software:

- Operating System (Windows, Linux, Unix, MAC OS, MS-DOS, VMS,...)
- Text Editors (Word, OpenOffice, Wordpad, Scite, vi, emacs,...)
- Spreadsheet managers (Excel, OpenOffice,...)
- ...

Standard computer structure (IV)

- **Computing performance**:
  - Clock rate of the CPU (GHz, MHz,...) usually measured in Gigaflops (Operations in Floating Point per Second)
  - Bus speed connection between the processor and the RAM memory (FSB-Front Side Bus) usually measured in MHz.
  - RAM memory: Measured in size (Mb, Gb,...) as well as in speed of access (MHz)
- **Information management**:
  - bit (BInary digiT): Number base 2 $\rightarrow \Box \left\{ \begin{array}{c} 0 \\ 1 \end{array} \right.$. It is the smallest memory unit
  - CPU: 8, 16, 32, 64, 128 bits processors
  - Measuring units:
    - 1 byte=8 bits=1 octet
    - 1 koctet=10^3 octets, 1 Moctet=10^6 octets, 1 Goctet=10^9 octets
    - 1 kbyte=2^{10} bytes = 1024 bytes, 1 Mb=2^{20} bytes = $(1024)^2$ bytes, 1 Gb=2^{30} bytes = $(1024)^3$ bytes
  - Binary base importance:
    - Computer structure
    - data storage: e.g. 23 in decimal base = $\left\{ \begin{array}{ccccc} \mathbf{1} & \mathbf{0} & \mathbf{1} & \mathbf{1} & \mathbf{1} \\ \mathbf{1}\cdot 2^4 + & \mathbf{0}\cdot 2^3 + & \mathbf{1}\cdot 2^2 + & \mathbf{1}\cdot 2^1 + & \mathbf{1}\cdot 2^0 \end{array} \right.$ in base 2

Algorithms, programs and programming languages (I)

- Algorithm:
  - A set of rules or instructions that precisely defines a sequence of operations for determining an output. They are much older than the existence of computers. (Mohammed Ibn Musa abu Djafar **Al-Khwarizmi**, mathematician of the VIII-IX century).
  - If the algorithms involves a computer language, then that set of instructions is called "computer program"
- Programming languages classification:
  - Machine language: Combination of 0 and 1 values that the computers is able to understand and interpret directly
  - Assembler language: replaces the machine language with mnemonic codes and symbolic names. The application that translates these codes into machine language is the "assembler".
  - High-level language: presents a simpler syntax for the user. Sometimes, there is an intermediate support language to make the transition. Some of these high-level languages are: Fortran, C, C++, Python, Java, Cobol, Lisp, Basic, Pascal...

♡ Each programming language is classified in terms of its level of abstraction Each programming language level is supported by programs developed at one of the lower levels.

Algorithms, programs and programming languages (II)

▶ Source code translation:

- Interpreters: they translate instruction by instruction the sequences of operations during execution. They are interactive and modifiable but are slow in execution. (Ej. Basic, Python)
- Compilators: translate the program (source code) in bulk before the execution of operations. They are not modifiable interactively but are very fast in the execution of operations.. (e.g. Fortran)

The stages of a Fortran program development are:

FORTRAN SOURCE CODE (*.for,*.f)

⇓ Compilation

OBJECT PROGRAM (*.obj)

⇓ Linking

EXECUTABLE PROGRAM (*.exe)

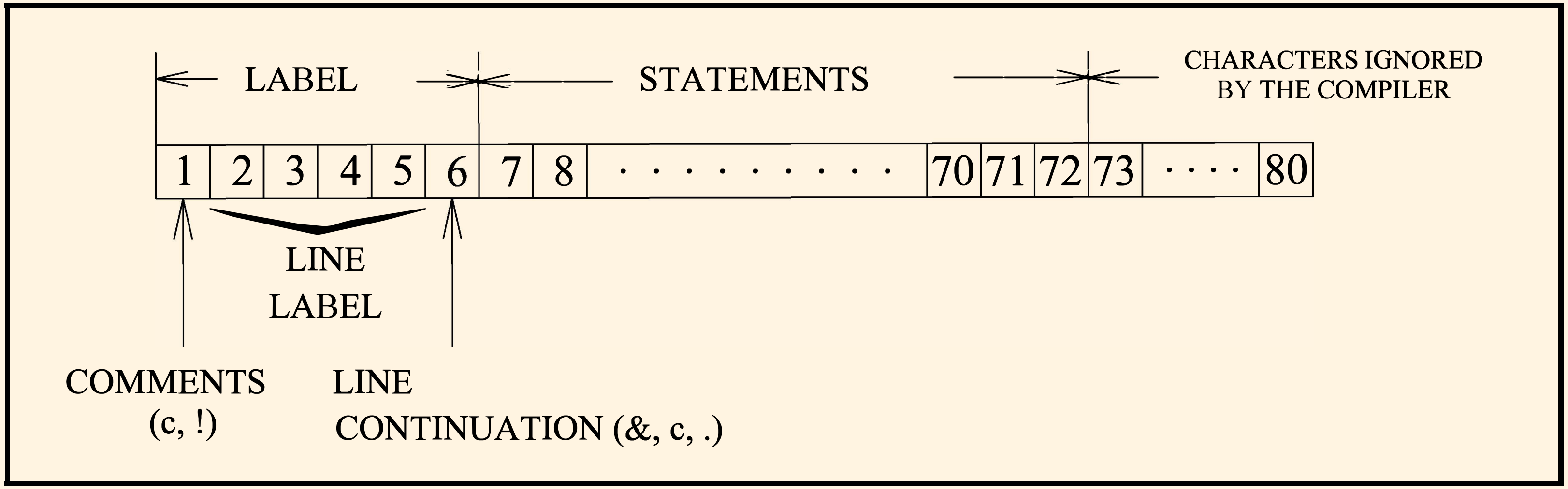
⇓ Execution

RESULTS

Fortran Language (I)

▶ Introduction:

- Fortran is the acronym for FORmula TRANslator and was created in 1954 by the IBM company as opposed to other languages very close to the machine language at the time.
- It is a standard language, easy to use, very widespread, very well adapted to engineering problems and very refined throughout its different versions: I, II, III, IV, 66, 77, 90, 95, HPF (High Performance Fortran), 2000.
- It is a sequential programming language whose sentences are incorporated into a plain text file with particular extension (*.f, *.for).
- The use of upper and lower case letters in the text file is irrelevant.
- The writing format of each line of source code is:



Fortran Language (II)

▶ Elements of a Fortran program

- Statements:
  - Non-executable — describe { -type, -characteristics, -values } of data
  - Executable — describe the instructions to perform
- Comments: They do not affect the processing of the program (source code), but help the programmer to understand it.

▶ Structure of a Fortran Program

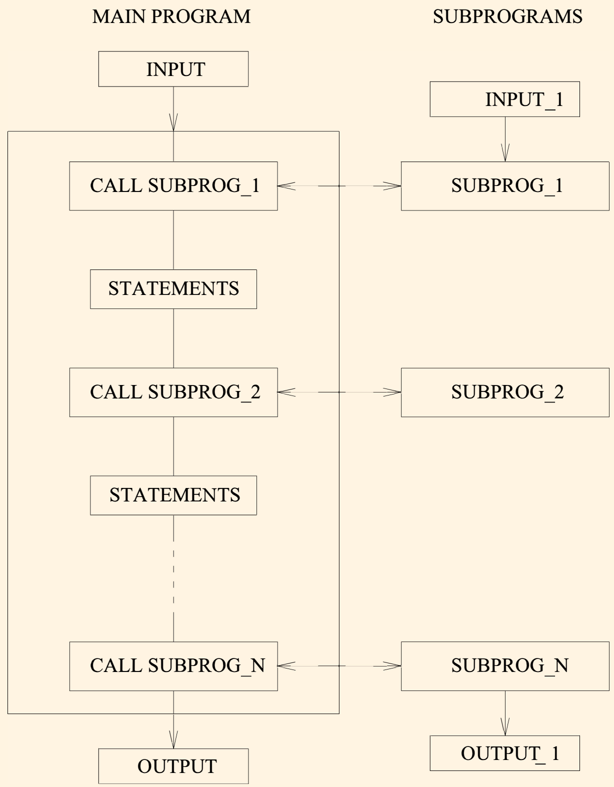
-Declaration of variables { Common; Dimension, implicit; Data, parameter }

.
.
.

-Executable statements

.
.
.

-END



Fortran Statements

- **Data:**
  - Constants:
    - Integer: 12, -37, ...
    - Real { fixed point: 6.5, -7.3, -0.12, 12.5 ... / floating point: 0.12E+04, 0.13E-01, ... }
    - Complex: (-3.7,5.4),(7E-3,5.1), ...
    - Logic: { `.TRUE.` / `.FALSE.` }
    - Character: 'Diego','problem1', ...

Fortran Language (IV)

Fortran Programming

- Variables: They are symbolic names that correspond to a certain memory location in which a value is stored (numeric, logical, alphanumeric, ...). The first character defining the name must be a letter, but the remaining characters can be other symbols. They can be declared explicitly (one by one) or implicitly (by a general criterion applicable to all of them).)

  - Integer: they store integer numbers.

    - By default, the first character must be I, J, K, L, M, N (I-N). But the default configuration can be easily changed.
    - 2 bytes of size(single precision) → INTEGER*2
      Range = (-32768, 32767)
    - 4 bytes of size (double precision) → INTEGER*4
      Range = (-2147483648, 2147483647)
    - Explicit declaration: `integer*2 ind, num`
    - Implicit declaration: `implicit integer*4(i-n)`

  Attention!

    - It is recommended to leave I-N as integers and use them only as integer counters.
    - It is not possible to operate directly with real numbers. They must first be transformed

Fortran Language (V)

- Reals:

  They are stored in floating point. By default, their name starts by (A-H,O-Z)

  May be stored in 4 bytes (single precision) → REAL*4

  Range ≈(-1.7E38,-2.9E-39), (2.9E-39,1.7E38)

  $$\underbrace{\pm 0.1234567}_{\substack{\text{MANTISSA}\\ \text{(24 bits)}}} \quad \text{E} \quad \underbrace{\pm\ 123456}_{\substack{\text{EXPONENT}\\ \text{(8 bits)}}}$$

  May be stored in 8 bytes (double precision) → REAL*8

  Range ≈(-1.0E+307,-1.E-309), (1.E-309,1.0E+307)

  If operations are to be performed between variables of different types, it is necessary to transform one of them so that they are of the same type.

  Explicit declaration: `real*8 coord, temp`
  Implicit declaration: `implicit real*8(a-h,o-z)`

- Complex:

  There is no default declaration.

  Explicit declaration: `complex*8 a1,a2` or `complex*16 a3`

  Implicit declaration: `implicit complex*8 (h-k)`

Fortran Language (VI)

- Logical: { `.true.` / `.false.` }

  There is no default declaration.

  Explicit declaration: `logical var1, var2`

  Implicit declaration: `implicit logical (a-c)`

  Operations: `.NOT.`, `.AND.`, `.OR.`

  Relational operators (they assign logic values to numerical variables)

  `.LT.` $\rightarrow <$ (or <)
  `.LE.` $\rightarrow \leq$ (or <=)
  `.EQ.` $\rightarrow =$ (or ==)
  `.NE.` $\rightarrow \neq$ (or /=)
  `.GE.` $\rightarrow \geq$ (or >=)
  `.GT.` $\rightarrow >$ (or>)

- Characters: Any set of characters between '

  Explicit declaration: `character nombre*20, apellido*30`

  Implicit declaration: `implicit character*20 (h-m)`

  Concatenation operator:

  ```
  a='PEDRO '
  b='GONZALEZ'
  c=a//b
  ```
  $\rightarrow$ c='PEDRO GONZALEZ'

Fortran Language (VII)

▷ Matrices: Matrices are stored by columns in an array

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix} \Rightarrow (a_{11}, a_{21}, a_{31}, a_{12}, a_{22}, a_{3,2}, a_{13}, a_{23}, a_{33})$$

```
integer i, j, k
real a, b, c
dimension i(10),a(3,4), b(10,10,10)
```

▷ Data: Assigns initial values to variables before the program is executed.

```
dimension a(3)
data a /1.0, 2.0, 3.0/
```

▷ Parameter: Assigns a symbolic name to a constant.

PARAMETER (identifier1=cte1, identifier2=cte2)

```
parameter (PI=3.14159265, ALPHA=2.7)
```

Arithmetic operations:

Arithmetic operations are interpreted from right to left. The operations indicated in the statements on the right side of the "=" symbol are performed and stored on the variable indicated on the left side.

```
factor=x*y
```

- Addition: `x=a+b` `a`, `b` and `c` types must coincide
- Subtraction: `x=a-b` `a`, `b` and `c` types must coincide
- Product: `x=a*b` `a`, `b` and `c` types must coincide
- Division: `x=a/b` `a`, `b` and `c` types must coincide
  Caution with dividing by 0.d+00. Overflow errors might appear given outputs of type (NaN)
- Power: `x=a**b`
  If possible, the exponent `b` should be integer. The base will usually be a real variable. Real exponent powers are slower and more imprecise.

Operations priority (lower to higher): $\begin{pmatrix} + \\ - \end{pmatrix}$, $\begin{pmatrix} * \\ / \end{pmatrix}$, $(**)$

And in case of conflict from left to right.

Fortran Language (IX)

▶ Functions

- External: arithmetic functions included in the system libraries

$sin(x)$	$asin(x)$	$log(x)$	$abs(x)$
$cos(x)$	$acos(x)$	$exp(x)$	$aint(x) \equiv E(x)$
			$cosh(x)$
$tan(x)$	$atan(x)$	$sqrt(x)$	$senh(x)$
			$tanh(x)$

- Intrinsic of the compiler: conversion functions
  - ▷ nint(x) (Nearest INTeger): (`real*4`) → Integer by approximation
  - ▷ int(x) (integer part): (`real*4` ó `real*8`) → Integer by truncation
  - ▷ ifix(x) (integer part): (`real*4`) → Integer by truncation
  - ▷ float(x) (floating point): integer → real by default, usually 4 bytes
  - ▷ dfloat(x) (floating point): integer → real in double precision
  - ▷ dble(x) any variable → real in double precision

- User-defined functions: defined by the user

Fortran Language (X)

▶ Flow control statements:

- <u>GOTO</u>

♡ GOTO UNCONDITIONAL:

GOTO ET_1

Transfers the flow of the program to the line of source code with label ET_1

It is recommendable that the line with lable ET_1 has the statement CONTINUE since some compilators require it

```
      goto 47
         non included statements
 47   continue
```

Fortran Language (XI)

♡ GOTO COMPUTED:

GOTO (ET_1,ET_2,...,ET_N), INDEX

If INDEX value is 1, 2, ..., N the program flow transfers to the lines of code with the labeling ET_1,ET_2,...,ET_N.

If there is no match the execution continues withe the line of codes that follows the GOTO

```
      goto(11,12,13), I
         statements for I ≠ 1, 2, 3
      goto 15
 11   continue
         statements for I = 1
      goto 15
 12   continue
         statements for I = 2
      goto 15
 13   continue
         statements for I = 3
 15   continue
```

- <u>IF</u>

  ♡ IF (ARITHMETIC):

  IF (INDEX)ET_1,ET_2,ET_3

  - If INDEX$<$0 the flow goes to the line with label ET_1
  - If INDEX$=$0 the flow goes to the line with label ET_2
  - If INDEX$>$0 the flow goes to the line with label ET_3

  ♡ IF (LOGIC):

  IF (ILOGIC) EXPRESSION

  ILOGIC$=$ expression or logic variable
  EXPRESSION$=$ any other statement

  Examples:

```
if ((a.eq.b).and.(c.eq.d))e=a+c

if ((a.eq.b).and.(c.eq.d))goto 10
```

♡ IF (BLOCK):

```
if (ilogic) then
  Statements to execute if ilogic is satisfied
endif
```

```
if (ilogic) then
  Statements to execute if ilogic is satisfied
else
  Statements to execute if ilogic is NOT satisfied
endif
```

```
if (ilogic1) then
  Statements to execute if ilogic1 is satisfied
elseif (ilogic2) then
  Statements to execute if ilogic2 is satisfied and not ilogic1
else
  Statements to execute if neither ilogic1 nor ilogic2 are satisfied
endif
```

```
if (model.eq.1) then
  a=x*y
elseif (model.gt.0) then
  a=x+y
else
  a=x-y
endif
```

Fortran Language (XIV)

Fortran Programming

♡ IF (NESTED BLOCKS):

```
if (cond1) then
   if (cond2) then
   ... statements to execute if cond1 and cond2 are satisfied
   else
   ... statements to execute if cond1 is satisfied and cond2 is not satisfied
   endif
else
   if (cond3) then
   ... statements to execute if cond1 is not satisfied and cond3 is satisfied
   else
   ... statements to execute if cond1 is not satisfied and cond3 is not satisfied
   endif
endif
```

```
if (a.eq.1) then
  if (b.eq.2) then
    x=a+b
  else
    x=a-b
  endif
else
  if (b.eq.1) then
    x=a*b
  endif
endif
```

Fortran Language (XV)

Example program: Computation of the factorial of 10

```
      program factorial
      implicit integer*4(i-n)

      ifact=1
      i=0

 10   continue

      i=i+1

      ifact=ifact*i

      if(i.lt.10)then
        goto 10
      endif

      end
```

Fortran Language (XVI)

- DO (LOOPS)

  ♡ Allows to repeat a sequence of operations a determined number of times:

```
do icount=imin,imax,is
  sequence to repeat from "imin" until "imax" in increments of "is"
enddo
```

  imin=initial value of the counter
  imax=maximum value of the counter (end of the loop)
  is=step between each counter value (optional, if not specified then → is=1)

```
      program factorial
      integer*4 i,fact
      fact=1
       do i=1,10
         fact=fact*i
       enddo
      end
```

```
      do i=n,1,-1
        do j=1,m
          Statements to repeat
        enddo
      enddo
```

- DO WHILE: Repeats the sequence while the specified condition is true

```
do while(logic_condition)
  Statements to repeat
enddo
```

- CALL: diverts the execution to a subprogram (to be seen later)
- RETURN: diverts the execution from a subprogram to back to the main program
- STOP: stops the execution of the program

Fortran Language (XVII)

▶ INPUT AND OUTPUT STATEMENTS. FORMATS

- <u>READ</u>: Statement for reading data (the program reads data)

  READ(NL,NF) Variables

  where: { NL = label of the logic unit to be read from (keyboard=5)
  NF = label of the line where the reading format is specified

```
      read(5,100)a,b,c
100   format(3d15.6)      → formats are explained in the next section
      read(5,*)a,b,c      → system's standard format
```

- <u>WRITE</u>: Statement for writing data (the program writes data)

  WRITE(NL,NF) Variables

  where: { NL = label of the logic unit to write int (screen=6)
  NF = label of the line where the writing format is specified

```
      write(6,100)a,b,c
100   format(3d15.6)       → formats are explained in the next section
      write(6,*)a,b,c      → system's standard format
```

Formats can be written at any point in the program, but it is advisable to write them after the READ or WRITE instruction

They should never be inserted inside control instructions (goto, if, ...) because they may not be accessible from other points in the program.

Fortran Language (XVIII)

- FORMAT: Determines the way in which the data to be read or written is to be processed (digits, decimals, type, ...)
  - ♡ Specifications:

\$	$\rightarrow$	To prevent the program from jumping a line at the end of the statement
/	$\rightarrow$	line break
,	$\rightarrow$	specifiers separator

  - ♡ Integers: n**I**m being $\begin{cases} n = \text{number of variables with that specification (optional)} \\ m = \text{number of digits to be used (sign included)} \end{cases}$

    `15i5` $\rightarrow$ 15 integer numbers with 5 digits (`-210`, `-1234`)

  - ♡ Reals: n**F**$m.d$ $\begin{cases} n = \text{number of real variables (optional)} \\ m = \text{total number of digits of the real number} \\ \quad \text{(sign included, decimal point and ``0.'')} \\ d = \text{number of decimal digits} \end{cases}$

    n**E**$m.d$ $\begin{cases} n = \text{number of reals (optional)} \\ m = \text{total number of digits of the real number} \\ \quad \text{(sign included, decimal point and 0., of the mantissa,} \\ \quad \text{E character, sign and digits of the exponent)} \\ d = \text{decimal digits of the mantissa} \end{cases}$

    n**D**$m.d$ same as the above but for `real*8`

    `17f5.1` $\rightarrow$ 17 real numbers with 5 digits and 1 decimal (`12.1`, `-12.1`)
    `3e12.5` $\rightarrow$ 3 real numbers with 12 characters and 5 decimal digits (`0.12345E+05`, `-0.12345E-05`)

Fortran Language (XIX)

♡ Characters: $nam \begin{cases} n = \text{number of variables (optional)} \\ m = \text{number of characters of each one (optional)} \end{cases}$

15a5 → 15 variables with 5 characters (Diego, series, ...)

♡ Spaces: nx where n is the number of spaces

♣ General rules about formats:

- ♡ Give each variable a suitable format
- ♡ if the format is exhausted before the variables, it is repeated
- ♡ If the list of variables is exhausted before the format, the rest of specifications are neglected
- ♡ Careful with the format capacity (the number 1000 does not fit in a i3), the character (∗) is written

♣ Alternative definition of formats:

They can be specified directly for each READ/WRITE statement

WRITE(6,'(FORMAT)') → write(6,**'(i5,5e15.6)'**)

♣ Examples

-1234.123 → { `read(5,`**10**`)a` / **10** **format**`(f9.3)` }   { `write(6,`**11**`)3.1416d+00` / **11** **format**`(d10.3)` } → 0.314E+01

Series 123 → `read(5,`**'(a6,x,i3)'**`)a,i`   `write(6,*)' Series'` → Series

- INPUT/OUTPUT DATA FILES

$$\text{OPEN(UNIT=}\left\{\text{n}^{\text{o}}\right\}\text{,FILE=}\left\{\begin{array}{c}\text{'input.txt'}\\ \text{'output.txt'}\\ \vdots\end{array}\right\}\text{, STATUS=}\left\{\begin{array}{l}\text{'old'}\\ \text{'new'}\\ \text{'unknown'}\end{array}\right\}\text{)}$$

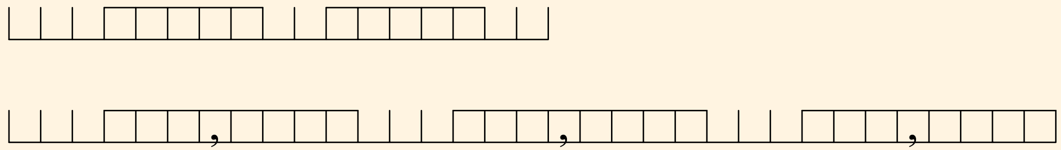
Usually, the number of the logic unit $\left\{\text{n}^{\text{o}}\right\} \in [10, 99]$

When the reading or writing is done should be closed with CLOSE($\left\{\text{n}^{\text{o}}\right\}$)

Examples:

```
      open(unit=11,file='datos.txt',status='old')
      do i=1,100
        read(11,'(i5,3e15.6)')ipoint,xpoint,ypoint,zpoint
      enddo
      close(11)

      open(unit=12,file='entrada.txt',status='unknown')
      read(12,10)i,j,x,y,z
10    format(3x,2(i5,2x),/,3(3x,f8.4))
      close(12)
```



Fortran Language (XXI)

- ARRAYS AND MATRICES (sets of structured data

Declaration → the same as the rest of variables.
The type of information to be stored is declared.

Static declaration → DIMENSION name_1(k_1,k_2,...,k_m)

- m indicates the number of dimensions of the array or matrix
  - ♡ $m = 1$ indicates that the data is stores with one-dimensional array structure
  - ♡ $m = 2$ indicates that the data is stores with two-dimensional array structure (matrix)
  - ♡ $m > 2$ indicates that the data is stores with m-dimensional array structure (hyper-matrix)
- k_i $(i = 1, ..., m)$ is the number of components that the array has in each dimension. The necessary memory is saved with that information.

```
dimension v(20),a(3,4) ! array v of 20 elements and
                       ! matrix a with 3 rows and 4 columns
```

- The data is internally stored by columns:

$$\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \quad \text{is stored as} \quad [a_{11},\ a_{21},\ a_{12},\ a_{22}]$$

Arrays (n)

Reading/writing by rows	Reading/writing by columns

`dimension a(10)`
 	
`read(5,*)(a(i),i=1,10)`
	`dimension a(10)`
	`do i=1,10`
	`  read(5,*)a(i)`
	`enddo`

Matrices (n×m)

Reading/writing in 1 row	reading/writing in rows and columns

`dimension b(10,20)`
 	
`write(6,*)((b(i,j),j=1,20),i=1,10)`
	`dimension b(10,20)`
	`do i=1,10`
	`  write(6,*)(b(i,j),j=1,20)`
	`enddo`

Reading/writing/modification by rows	**Reading/writing/modification by columns**

`dimension b(10,20)`
`do i=1,10`
`  do j=1,20`
`    read(5,*)b(i,j)`
`  enddo`
`enddo`
	`dimension b(10,20)`
	`do j=1,20`
	`  do i=1,10`
	`    write(6,*)b(i,j)`
	`  enddo`
	`enddo`

Fortran Language (XXIII)

Relevan issues:

- The storage of large arrays and matrices is very expensive because it requires a lot of memory storage capacity.

```
implicit real*8(a-h,o-z)
dimension a(10000,10000)
```

  $10000 \times 10000 \times 8$ bytes $\approx$ 763Mb

- It is necessary to be very careful with the declaration of the dimension of the matrices.:
  - Over-sizing can exceed the computer's memory limits.
  - If we fall too short we can overwrite other variables (careful! Fortran does not warn about this issue)

  ◇ **SOLUTION: DYNAMIC MEMORY ALLOCATION.**

DYNAMIC MEMORY ALLOCATION:

- It is done in two steps:
  1. In the declaration of variables it is stated that one variable will be an aray:

```
implicit integer*4(i-n), real*8(a-h,o-z)
allocatable name1(:,:,...), name2(:,:,...)
...
```

     The number of dimensions of the array is indicated with the number of times that ":" between the parenthesis.
  2. Then, in the program statements, the dimension of the array will be indicated *array*:

```
...
allocate (name1(k1, k2, ...), name2(l1, l2, ...))
...
```

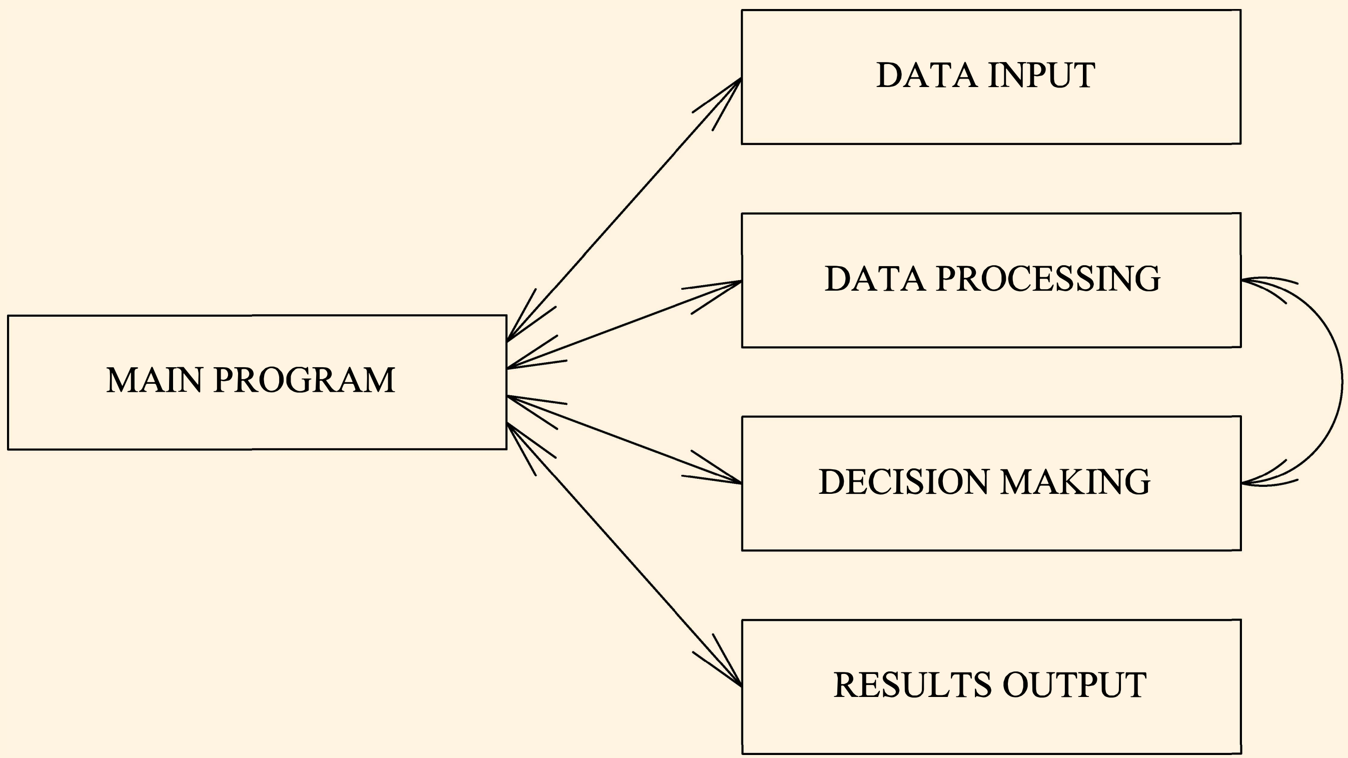
     being $k1,\ k_2, \ldots$ and $l_1,\ l_2, \ldots$ the number of elements of the arrays in each dimension
- The type of data (INTEGER, REAL, ...) contained in the array is indicated similarly than for the rest of variables: explicitly or implicitly.
- NOTE: It is recommended to use this form of dynamic sizing only in the main program.

Fortran Language (XXV)

▶ SUBPROGRAMS:

- Using subprograms allow to access to computer modules separated from the main program, accessible at any point of it.
- Once concluded, the subprogram returns the control to the main program.
- Different modules can, at the same time, be called between them.
- They allow a modular and structured programming. It is possible that in the main program there are only calls to the subprogram that execute the operations → simpler and cleaner programs
- Each module can be compiled separately (each of them is terminated by an END statement). They are connected to the main program in the linking phase.
- The exchange of information between the main program and the subprograms is key. (CAREFUL: variables in fortran indicate the position in memory where they are stored)

- Usual flowchart of a program.



Fortran Language (XXVII)

- FUNCTIONS:

  ♡ They are elemental subprograms that the user develops to evaluate functions

  Type FUNCTION function_name (list of arguments)
  List of 'COMMON'
  List of 'DIMENSION' and 'ALLOCATABLE'
  ...Statements
  RETURN
  END

  ♡ Definition

  -Tipo = INTEGER, REAL, ... If not specified depends on the first letter of the name

  -name_funcion = name of the function

  -list of arguments = (optional) name of the variables sent as arguments separated by commas

  -list of common = (optional) name of the variables sent as common blocks (to be seen later)

  -Statements = Statements to be executed. The name of the function should appear at least once as a variable

  -RETURN = returns the control to the call statement.

- FUNCTIONS:

  ♡ Call from the main program

  It is a direct assign statement such as:

  variable = name_funcion (list of arguments)

  where:

  -variable = stores the value returned by the function

  -List of arguments: list of variables separated by commas sent to the function. They must match in number, type and order. The names can be different.

Fortran Language (XXIX)

▷ e.g. function that obtains the module of a vector of m elements (m $\leq$ 10)

Main program.

```
implicit real*8(a-h,o-z)
implicit integer*4(i-n)
dimension v(10)
n=10
do i=1,n
  v(i)=1.d+00
enddo
vmod=vecmod(v,n)
end
```

Function

```
function vecmod(w,m)
implicit real*8(a-h,o-z)
implicit integer*4(i-n)
dimension w(m)
vecmod=0.d+00
do i=1,m
  vecmod=vecmod+w(i)*w(i)
enddo
vecmod=sqrt(vecmod)
return
end
```

- ♡ The variables sent as arguments keep the same position in the memory as in the main program.
- ♡ Thus, if they are modified inside the function, they are modified for the rest of modules, operations and programs.
- ♡ variables not indicated in the list of arguments are local for the function and, even if their names coincide they correspond to different memory positions
- ♡ At the end of the function execution, the local variables are deleted

Fortran Language (XXX)

- SUBROUTINES:

  Subprograms that allow to return to the main program not only one value but a set of results

  ```
  SUBROUTINE name(list of arguments separated by commas)
  list of "dimension"
  ... Statements
  RETURN
  END
  ```

  The variables sent as arguments keep the same memory position as in the main program and are global for all the program.

  The variables defined inside the subroutine not sent as arguments are local and are deleted at the end of it.

  CAREFUL: The name of the subroutine can not appear as a variable

  Call from the main program:

CALL name (list of arguments)	(input and output arguments)

Fortran Language (XXXI)

▷ Example of a program using a subroutine:

Main program.

```
implicit real*8(a-h,o-z)
implicit integer*4(i-n)
dimension v(10)
n=10
do i=1,n
  v(i)=1.d+00
enddo
call vmod(v,n,vmodulus)
end
```

Subroutine

```
subroutine vmod(w,m,wmod)
implicit real*8(a-h,o-z)
implicit integer*4(i-n)
dimension w(m)
wmod=0.d+00
do i=1,m
  wmod=wmod+w(i)*w(i)
enddo
if (wmod.eq.(0.d+00))stop
wmod=sqrt(wmod)
do i=1,m
  w(i)=w(i)/wmod
enddo
return
end
```

- COMMON block:

  Usually at the start of the declaration of variables

  COMMON /name/ list_of_variables

  - In all modules in which the COMMON block of a given name is declared, the variables in the list (which must match in type) will occupy the same memory position
  - They are usually combined with the lists of arguments of subroutines and functions
  - The COMMON block implicitly performs the DIMENSION of the variables of the list.

Fortran Language (XXXIII)

- Example:

Main Program.

```
implicit real*8(a-h,o-z)
common /vector/ v(10),n
implicit integer*4(i-n)
n=10
do i=1,n
  v(i)=1.d+00
enddo
call vmod(vmodulus)
end
```

Subroutine

```
subroutine vmod(wmod)
common /vector/ w(10),m
implicit real*8(a-h,o-z)
implicit integer*4(i-n)
wmod=0.d+00
do i=1,m
  wmod=wmod+w(i)*w(i)
enddo
wmod=sqrt(wmod)
return
end
```

Fortran Language (XXXIV)

- References:

- *Fortran 77 for engineers and scientists with an introduction to Fortran 90*, Larry Nyhoff y Sandford Leestma, Prentice Hall, Upper Saddle River, NJ, USA, 1996
- *Aprenda Fortran 8.0 como si estuviera en primero*, Javier García de Jalón, Francisco de Asís de Ribera, E.T.S. Ingenieros Industriales, Universidad Politécnica de Madrid, 2005