Sintassi di base

Introduction to Fortran for Scientific Computing

Caterina Caravita
c.caravita@cineca.it

COSA VIENE PRESENTATO

- Struttura del programma
- Dichiarazione delle variabili:
  tipi, precisione(kind), attributi
- Operatori numerici e logici
- Manipolazione di stringhe di caratteri
- Funzioni intrinseche e moduli intrinseci

Struttura del programma

```
PROGRAM nome
  IMPLICIT NONE
  Dichiarazioni variabili
  Istruzioni
END PROGRAM nome
```

Struttura del programma

Esempio: semplice.f90

```
PROGRAM semplice
   IMPLICIT NONE

   REAL :: n, m
   INTEGER, PARAMETER :: LS = 40
   CHARACTER(LS) :: nome

   PRINT *,"Qual e' il tuo nome?"
   READ *, nome

   n = 2; m = 3

   PRINT *, "Il tuo nome e' ", nome
   PRINT *, " e ", n, " + ", m, " = ", (n + m)

   STOP
END PROGRAM semplice
```

Dichiarazioni variabili

Istruzioni

Lancia un programma Fortran (mooolto brevemente)

Compilazione
Esecuzione

```
$ gfortran program_name.f90
$ ./a.out
```

Lezione Compilazione e linking

ESERCIZIO 0

Compilare ed eseguire ***semplice.f90***

```
$ gfortran semplice.f90 -o semplice.out
$ ./semplice.out
```

Programmazione procedurale

Ogni sezione del programma inizia e termina con un'istruzione specifica.

```
PROGRAM nome
   ...
END PROGRAM nome
```

Programma principale

```
TYPE nome
   ...
END TYPE nome
```

Definizione di un tipo derivato

Programmazione procedurale

Ogni sezione del programma inizia e termina con un'istruzione specifica.

```
FUNCTION nome
  ...
END FUNCTION nome
```

Calcola una funzione e restituisce **un solo risultato**, solitamente senza modificare i suoi argomenti di input.

```
SUBROUTINE nome
  ...
END SUBROUTINE nome
```

Restituisce **zero o più risultati**, attraverso i propri argomenti di input e output, e solitamente esegue calcoli più complicati; chiamata con l'istruzione **CALL**.

```
MODULE nome
  ...
END MODULE nome
```

Contiene **definizioni** e **valori iniziali** dei dati che saranno condivisi fra varie unità di programma con l'istruzione **USE**.

Lezione di domani: Introduzione alle procedure

Nomi

I caratteri utilizzabili dal Fortran sono:
`A-Z 0-9 + * - / < > ; ! % ' " & _ () [] $ = , .`

I nomi delle entità (variabili, procedure...):
- possono avere fino a 31 caratteri, costituiti da caratteri alfanumerici (`a-z`, `0-9`) e dal carattere underscore (_)
- devono iniziare con un carattere alfabetico
  `No 1x, _x`
  `Sì x1`
- non possono essere parole riservate del linguaggio (es. `function`)

Fortran è **case insensitive**: non esiste distinzione tra maiuscole e minuscole.

Le entità utilizzate in una sezione di codice devono essere **dichiarate prima** delle istruzioni.

Dichiarazione delle variabili

L'istruzione **IMPLICIT NONE** impone la **dichiarazione esplicita di tutte le variabili** che si utilizzano nel codice.

Se non si usa, il compilatore assume il **tipo** delle variabili non dichiarate esplicitamente dall'iniziale

```
[i, j, k, l, m, n] => INTEGER
[a – h, o - z]     => REAL
```

Se si usa e una variabile viene usata ma non è stata dichiarata, si ottiene errore (**utile!**).

È caldamente consigliato l'utilizzo dell'istruzione IMPLICIT NONE.
Deve essere la prima istruzione dopo l'inizio di una sezione, può essere preceduta solo dalle istruzioni USE e FORMAT.

Dichiarazione delle variabili

```
tipo ([KIND=]kind) [attributi] :: variabile [=valore]
```

tipo INTEGER, REAL, COMPLEX, LOGICAL, CHARACTER, TYPE(nome)

kind specifica la precisione delle variabili
(dipende dal tipo e dall'architettura della macchina)

attributo PARAMETER, DIMENSION, ALLOCATABLE, PUBLIC, PRIVATE, POINTER, TARGET, INTENT, OPTIONAL, EXTERNAL, INTRINSIC

Tipi

```
tipo ([KIND=]kind) [attributi] :: variabile [=valore]
```

In Fortran esistono 5 **tipi predefiniti**

Tipi numerici:

`INTEGER`	numeri interi
`REAL`	numeri reali (floating point)
`COMPLEX`	numeri complessi (rappresentati da una coppia di numeri reali)

Tipi non numerici:

`LOGICAL`	valori booleani (`.TRUE.`, `.FALSE.`)
`CHARACTER`	stringhe di caratteri

Tipi

```
tipo ([KIND=]kind) [attributi] :: variabile [=valore]
```

In Fortran 77 si usa anche il tipo numerico **DOUBLE PRECISION** per i numeri reali in doppia precisione (in genere, 8 byte anziché 4 byte, ovvero 64 bit anziché 32 bit). Tale tipo, anche se ancora utilizzabile, è stato integrato dalla versione Fortran 90 nel tipo REAL:

- nel caso di singola precisione è sufficiente dichiarare una variabile di tipo **REAL**
- nel caso di doppia precisione si deve utilizzare l'istruzione **KIND**.

Esistono infine i **tipi derivati**, definiti dal programmatore come un insieme di più variabili di tipo predefinito o derivato, in modo da formare strutture di dati, che possono essere anche piuttosto complesse. Vengono definiti in una sezione **TYPE**.

Tipi

```
tipo ([KIND=]kind) [attributi] :: variabile [=valore]
```

Dichiarazione variabile reale in singola precisione:

```
REAL singprec
```
Fortran 77

```
REAL :: singprec
```
Fortran 90, Fortran 2003, Fortran 2008

Dichiarazione variabile reale in doppia precisione:

```
DOUBLE PRECISION doublprec
```
Fortran 77

```
REAL(8) :: doublprec
```
Fortran 90

```
USE ISO_FORTRAN_ENV
REAL(KIND=real64) :: doublprec
```
Fortran 2003, Fortran 2008

Kind

```
tipo [([KIND=]kind)] [attributi] :: variabile [=valore]
```

L'attributo KIND determina la **precisione** del dato (integer, real, complex, logical, character), ovvero l'occupazione di memoria, in base al **sistema** (architettura, compilatore).

Il valore dell'attributo è un **numero intero**, che rappresenta la precisione (**byte**) della variabile.

```
REAL :: a
```

Default: **4 bytes** (32 bit) => **singola precisione**

```
REAL(8) :: b
REAL(KIND=8) :: b
```

8 bytes (64 bit) => **doppia precisione**

Può essere espresso esplicitamente o utilizzando le funzioni intrinseche `KIND()`, `SELECTED_INT_KIND()`, `SELECTED_REAL_KIND()`.

Kind

```
tipo [([KIND=]kind)] [attributi] :: variabile [=valore]
```

La funzione intrinseca **KIND()** restituisce un numero intero, che rappresenta la precisione (**byte**) della variabile in argomento, in base al **sistema**.

Può essere usata per dichiarare una variabile

```
REAL(KIND=KIND(1.0D0)) :: a
```

Anche dichiarando una variabile con attributo PARAMETER (non modificabile)

```
INTEGER, PARAMETER :: doppia_prec = KIND(1.0D0)
REAL(KIND=doppia_prec) :: a
```

O per verificarne la precisione

```
PRINT *, KIND(a)   ->   8
```

Kind

```
tipo [([KIND=]kind)] [attributi] :: variabile [=valore]
```

Se si intende lavorare con valori interi, reali e complessi che richiedono una **determinata precisione**, indipendentemente dal sistema di calcolo utilizzato, sono a disposizione due funzioni specifiche:

`SELECTED_INT_KIND()`	interi
`SELECTED_REAL_KIND()`	reali e complessi

Queste funzioni assicurano la **portabilità** del codice da una piattaforma ad un'altra, mantenendo la precisione richiesta (purché disponibile).

Kind

```
tipo [([KIND=]kind)] [attributi] :: variabile [=valore]
```

La funzione **SELECTED_INT_KIND()** si applica a **numeri interi** e riceve in argomento **un numero intero** che rappresenta l'intervallo dell'esponente dei numeri rappresentabili: restituisce il KIND che permette (con quel sistema) di rappresentare quel numero con la precisione richiesta.

```
INTEGER, PARAMETER :: esp11 = SELECTED_INT_KIND(11)
INTEGER(KIND=esp11) :: a
```

=> possibili valori di a tra -10^{11} e 10^{11}

Kind

```
tipo [([KIND=]kind)] [attributi] :: variabile [=valore]
```

La funzione **SELECTED_REAL_KIND()** si applica a **numeri reali e complessi** e riceve in argomento **due numeri interi** che rappresentano il numero di cifre decimali e l'intervallo dell'esponente dei numeri rappresentabili: restituisce il KIND che permette (con quel sistema) di rappresentare quel numero con la precisione richiesta.

```
INTEGER, PARAMETER :: esp11 = SELECTED_REAL_KIND(9, 11)
REAL(KIND=esp11) :: a
```

=> a può avere almeno 9 cifre decimali e valori tra +/- 10^{-11} e +/- 10^{11}

Kind

```
tipo [([KIND=]kind)] [attributi] :: variabile [=valore]
```

Il valore intero restituito da SELECTED_INT_KIND() e SELECTED_REAL_KIND() coinciderà con una **precisione disponibile** nel sistema di calcolo utilizzato.
Tipicamente:

4	singola precisione
8	doppia precisione
-1	se la precisione richiesta non è rappresentabile nel sistema di calcolo

```
SELECTED_REAL_KIND(3,100)    ->    4
SELECTED_REAL_KIND(8,100)    ->    8
SELECTED_REAL_KIND(39,100)   ->   -1
```

Kind

```
tipo [([KIND=]kind)] [attributi] :: variabile [=valore]
```

Verificare precisione e dimensione di una variabile con le funzioni intrinseche

`kind()`	precisione in bytes
`range()`	esponente in base 10 del range di numeri rappresentabili
`huge()`	massimo numero positivo rappresentabile
`tiny()`	minimo numero positivo rappresentabile (solo per tipi REAL)

Kind

In Fortran tutte le **funzioni numeriche** (e.g. `abs()`) e **matematiche** (e.g. `sin()`) supportano valori reali sia a singola sia a doppia precisione: se il valore di input è a precisione singola, allora la funzione verrà calcolata con un risultato a precisione singola, se il valore di input è a doppia precisione, la funzione verrà calcolata con un risultato a precisione doppia.

Un'importante funzione intrinseca è **DBLE()**: converte qualsiasi argomento di ingresso in un valore a doppia precisione sul particolare processore in cui viene eseguito.

Perché non usiamo sempre numeri reali a 64 bit?

Perché ogni numero reale a 64 bit richiede il doppio della memoria di un numero reale a 32 bit: questo rende i programmi che li utilizzano molto più grandi, ed è necessaria più memoria per eseguire i programmi.

Dovremmo utilizzare numeri di precisione più elevata solo quando sono effettivamente necessari...

Kind

Quando sono effettivamente necessari i numeri a 64 bit?

- Quando il calcolo richiede **numeri "molto" grandi**

Type	Sign	Usual `huge()`	Usual Width (bits)	Usual Size (bytes)
`integer(selected_int_kind(2))`	+/-	127	8	1
`integer(selected_int_kind(4))`	+/-	32767	16	2

`integer`
`integer(kind(0))`
`integer(selected_int_kind(9))`
	+/-	2147483647	32	4

`integer(selected_int_kind(18))`	+/-	9223372036854775807	64	8

Type	Usual `huge()`	Usual Width (bits)	Usual Size (bytes)

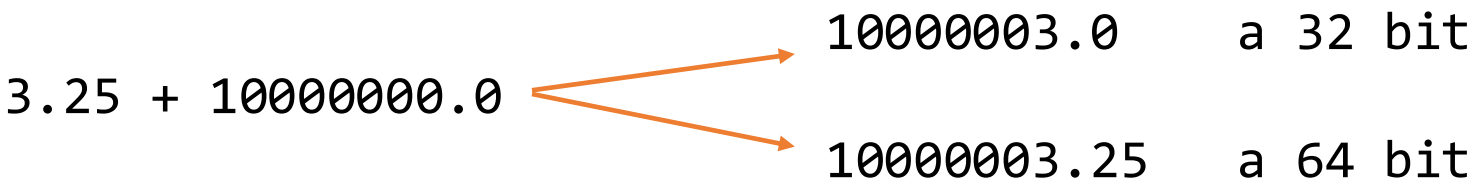
`real`
`real(kind(0.0))`
`real(selected_real_kind(6))`
	3.40282347e38	32	4

`double precision`
`real(kind(0.0d0))`
`real(selected_real_kind(15))`
	1.79769313486231573e308	64	8

Kind

Quando sono effettivamente necessari i numeri a 64 bit?

- Quando il problema richiede di **sommare o sottrarre numeri di dimensioni molto diverse**: il calcolo risultante perderebbe di precisione.



- Quando il problema richiede la **sottrazione di due numeri di dimensioni quasi uguali**: piccoli errori nelle ultime cifre dei due numeri diventano esageratamente grandi per operazioni di arrotondamento.

```
a1 = 1.0000000: errori di arrotondamento in serie di calcoli -> es. 1.0000010
a2 = 1.0000005: errori di arrotondamento in serie di calcoli -> es. 1.0000000

true_result = a1 - a2 = - 0.0000005
actual_result = a1 - a2 = 0.0000010     % error = -300%
```

Attributi

```
tipo ([KIND=]kind) [attributi] :: variabile [=valore]
```

`PARAMETER` identifica una variabile il cui valore è **costante**

```
INTEGER statico                                          Fortran 77
PARAMETER (statico = 10)

INTEGER, PARAMETER :: statico = 10, m = 1000             Fortran 90
REAL, PARAMETER    :: a = 2.4582, b = 78.512943
```

Attributi

```
tipo ([KIND=]kind) [attributi] :: variabile [=valore]
```

`DIMENSION`	specifica la dimensione di un **array** (lo spazio in memoria di un array automatico è allocato quando avviene la chiamata alla procedura e deallocato all'uscita da essa)
`ALLOCATABLE`	definisce un **array** che viene allocato/deallocato esplicitamente
`PUBLIC`	definisce un oggetto che sarà disponibile all'esterno del modulo da ogni unità di programma che usi il modulo stesso
`PRIVATE`	definisce un oggetto che sarà visibile solo all'interno del modulo
`EXTERNAL`	specifica che una procedura è definita nel programma (permettendo di sovrascrivere una procedura intrinseca)
`INTRINSIC`	specifica che una procedura è intrinseca (può poi essere un argomento)

Attributi

```
tipo ([KIND=]kind) [attributi] :: variabile [=valore]
```

`POINTER` definisce un variabile dinamica, un nome simbolico che punta all'**indirizzo di memoria** di una variabile TARGET

`TARGET` definisce una variabile statica, che contiene un valore vero e proprio, al cui **indirizzo di memoria** punta il POINTER

`INTENT` specifica il modo in cui la variabile verrà utilizzata dalla procedura

`INTENT(IN)`	letto in ingresso
`INTENT(OUT)`	scritto in uscita
`INTENT(INOUT)`	letto in ingresso e sovrascritto in uscita (default)

`OPTIONAL` definisce una variabile opzionale, non necessariamente richiesta dalla procedura

Variabile numerica

```
tipo ([KIND=]kind) [attributi] :: variabile [=valore]
```

```
INTEGER, PARAMETER :: a = 5

INTEGER :: a
a = 5

REAL, DIMENSION(2,3) :: matrix
matrix(1,1) = 5.0
matrix(1,2) = 4.3

INTEGER, PARAMETER :: prec_r = SELECTED_REAL_KIND(8,100)
REAL(KIND=prec_r)  :: a

COMPLEX(8) :: a
a = (3,-5)
```

Operatori aritmetici

tra variabili numeriche

Operatori binari:

`+`	addizione
`-`	sottrazione
`*`	moltiplicazione
`/`	divisione
`**`	esponenziale

Operatori unari:

```
+ variabile
- variabile
```

Operatore di Assegnazione:

```
variabile = espressione
```

Esempi:

```
c = a + b
c = a**b
c = - a * b
```

Operatori aritmetici

Non è possibile affiancare due operatori:

```
NO a*-b        Sì a*(-b)
NO a**-b       Sì a**(-b)
```

La moltiplicazione implicita non è ammessa:

```
NO x(y+z)      Sì x*(y+z)
```

Ordine di valutazione: parentesi (a partire dalla più interna) -> esponenziale -> moltiplicazione e divisione -> addizione e sottrazione

Se due operatori hanno la stessa priorità, si valuta da sinistra a destra.

```
y = (a + b/(c*d) - g/(5*(h-x)) ) **(e)
```

Il risultato di operazioni tra interi è un valore intero, tra reali è un valore reale e tra un intero e un reale è un valore reale (così via coi numeri complessi).

Operatori relazionali

tra variabili numeriche

`<`	`.LT.`	minore
`<=`	`.LE.`	minore o uguale
`>`	`.GT.`	maggiore
`>=`	`.GE.`	maggiore o uguale
`==`	`.EQ.`	uguale
`/=`	`.NE.`	non uguale

restituiscono un **valore booleano** (T - .TRUE., F - .FALSE.)

Esempi:

```
log1 = (int1 < int2)

if (real1 >= real2) then
    ...
```

Operatori logici

tra valori booleani (**T** - **.TRUE.**, **F** - **.FALSE.**)

`.AND.`	congiunzione
`.OR.`	disgiunzione
`.NOT.`	negazione
`.EQV.`	equivalenza
`.NEQV.`	non equivalenza

restituiscono a loro volta un **valore booleano**

Esempi:

```
log1 = .not. log2

if (a==b .and. a<=d) then
    ...

if (a==b .eqv. a<=d) then
    ...
```

x	y	.NOT. x	x .AND. y	x .OR. y	x .EQV. y	x .NEQV. y
F	F	T	F	F	T	F
T	F	F	F	T	F	T
F	T	T	F	T	F	T
T	T	F	T	T	T	F

Variabile carattere

Una stringa di caratteri è dichiarata con una lunghezza.
Se si conferisce l'attributo PARAMETER ad una stringa di caratteri, non è necessario dichiararne esplicitamente la lunghezza in quanto tale valore viene assunto in modo automatico.

Il valore di una costante di tipo stringa di caratteri può essere definito delimitando il testo tra apici singoli (' ') o doppi (" ").

Si possono anche dichiarare vettori e matrici di caratteri.

```
INTEGER, PARAMETER :: lunghezza=21
CHARACTER(LEN=lunghezza) :: nome
nome = "questa e' una stringa"
```

Variabile carattere

Dichiarazioni equivalenti

```
CHARACTER(LEN=21) :: nome
CHARACTER(21) :: nome
CHARACTER :: nome*21

CHARACTER(21), DIMENSION(10,12) :: matrice_stringhe
CHARACTER :: matrice_stringhe(10,12)*21

CHARACTER(21), PARAMETER :: "questa e' una stringa"
CHARACTER(*), PARAMETER :: "questa e' una stringa"
```

Manipolazione carattere

Estrazione di una **sotto-stringa**

```
sottostringa = stringa(inizio:fine)
```

```
PROGRAM stringhe
    IMPLICIT NONE
    CHARACTER(LEN=40) :: stringa
    CHARACTER(LEN=40) :: sottostringa

    stringa = "supercalifragilistichespiralidoso"
    sottostringa = stringa(1:5)

     PRINT *, "sottostringa: ", sottostringa

    STOP
END PROGRAM stringhe
```

Output:

```
Sottostringa: super
```

Manipolazione carattere

Concatenamento di più stringhe

```
stringa_finale = prima(inizio:fine)//prima(inizio:fine)
```

```
PROGRAM stringhe
    IMPLICIT NONE
    CHARACTER(LEN=40) :: stringa
    CHARACTER(LEN=40) :: stringa_finale

    stringa = "supercalifragilistichespiralidoso"
    stringa_finale = stringa(1:5)//stringa(23:29)

     PRINT *, "stringa concatenata: ", stringa_finale

    STOP
END PROGRAM stringhe
```

Output:

```
stringa concatenata: superspirali
```

Manipolazione carattere

Lunghezza effetiva

```
CHARACTER(LEN=10) :: stringa
stringa = "esempio"

PRINT *, "len: ", LEN(stringa)

PRINT *, "len_trim: ", LEN_TRIM(stringa)


concat = "E' un " // stringa // " semplice"
PRINT *, concat

concat_trim = "E' un " // TRIM(stringa) // " semplice"
PRINT *, concat_trim
```

Output:

```
len: 10

len_trim: 7


E' un esempio    semplice

E' un esempio semplice
```

Matrici per colonne

Fortran interpreta le matrici (vettori multi-dimensionali) **per colonne**
(a differenza di C, C++ e Python che le interpretano per righe)

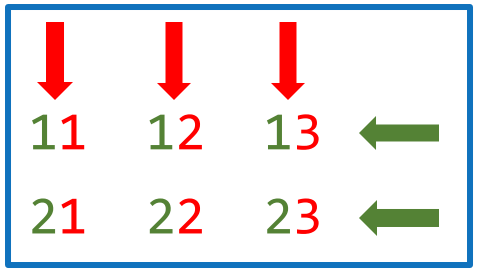
Matrice di numeri interi, 2x3 (2 righe e 3 colonne):

```
INTEGER, DIMENSION(2,3) :: mat

mat(1,1) = 11
mat(1,2) = 12
mat(1,3) = 13

mat(2,1) = 21
mat(2,2) = 22
mat(2,3) = 23

PRINT *, 'mat = ', mat
```



Output:

```
mat =   11   21   12   22   13   23
```

colonna 1 colonna 2 colonna 3

Istruzioni

Su una riga di codice possono essere scritte **più istruzioni** separate da ;

(suggerimento: solo se brevi!)

```
INTEGER :: n, m
n = 2; m = 3
```

Un'istruzione (anche una stringa) può essere scritta **su più righe** (in genere se molto lunga), utlizzando & alla fine della riga (& può essere ripetuto all'inizio della riga successiva)

```
REAL, DIMENZION(100, 50) :: vettore 1, vettore2, &
                            Vettore_somma, vettore_diff

PRINT *, 'Ho fatto delle operazioni &
          & tra due vettori bi-dimensionali'
```

Recap

Esempio: articolato.f90

```
PROGRAM articolato
   IMPLICIT NONE
   REAL :: n, m, s
   INTEGER, PARAMETER :: LS=80
   CHARACTER(LS) :: nome, testo

   PRINT *, "Qual e' il tuo nome?"
   READ *, nome

   testo = "Il tuo nome e' "//TRIM(nome)//" e "

   n = 2; m = 3
   CALL SOMMA(s, n, m)

   PRINT *, TRIM(testo), n, " + ", m, " = ", s

   STOP
END PROGRAM articolato
```

```
SUBROUTINE SOMMA(r, a, b)
   IMPLICIT NONE
   REAL :: r, a, b

   r = a + b

   RETURN
END SUBROUTINE
```

ESERCIZI

Scrivere un programma che, definite alcune variabili numeriche, esegua le seguenti operazioni e le scriva a **video**:

$$x = \left[(a+b)^2 + (3c^2)\right]^{a/b}$$

$$y = \left[\frac{ab}{c+d} - \frac{g}{5\,(h+x)}\right]^{1/r}$$

$$x + \frac{y}{z \cdot a + b^2}$$

Tipologie di funzioni in Fortran

Il linguaggio Fortran dispone di meccanismi per supportare le funzioni matematiche molto comuni, ma anche le funzioni meno comuni.

Molte di quelle più comuni sono predefinite e codificate direttamente nel linguaggio Fortran come **funzioni intrinseche**.

Le funzioni meno comuni non sono incluse nel linguaggio Fortran, ma l'utente può fornire qualsiasi funzione necessaria per risolvere specifici problemi, utilizzando le **funzioni *user-defined***.
(Lezioni di domani: Introduzione alle procedure)

***user-defined* FUNCTION:**

```
tipo FUNCTION nome (argomenti)
  (dichiarazioni variabili)
  (istruzioni)
  nome = valore di ritorno
END FUNCTION nome
```

Sono invocate con la seguente sintassi, fornendo gli argomenti di **input della funzione**:

```
nome (argomenti)  ->  valore di ritorno
```

Funzioni intrinseche

Le funzioni intrinseche possono operare su **variabili numeriche**, **logiche**, **stringhe di caratteri**, **array** e si possono classificare prevalentemente in

Elementali

Il valore viene calcolato **elemento per elemento**.
La maggior parte delle funzioni comuni sono elementali, come le funzioni numeriche (`ABS, INT`) e matematiche (`SIN, SQRT`)

Informative

Restituiscono **proprietà** degli argomenti (`LEN, SHAPE, SIZE`)

Trasformative

Trasformano un'entità in un'altra, operando sugli **array** nel loro insieme.
L'output di una funzione trasformativa spesso non avrà la stessa *shape* degli argomenti di input (`SUM, MATMUL, TRANSPOSE`)

Funzioni intrinseche

Funzioni numeriche

```
abs(a)          valore assoluto di a
aimag(z)        parte immaginaria del numero complesso z
aint(r)         tronca il numero reale r per dare un numero intero
anint(r)        arrotonda il numero reale r ad un intero reale
ceiling(a)      numero intero più grande o uguale al numero a
cmplx(a,[b])    converte (a,b) nel numero complesso a + ib
floor(a)        numero intero più piccolo o uguale al numero a
int(a)          converte il numero a in un numero intero
nint(a)         arrotonda il numero a ad un numero intero
real(a)         converte il numero a in un numero reale
conjg(z)        coniugato del numero complesso z
dim(a,b)        restituisce il valore (a-b,0) - positive difference
max(list)       valore massimo di una lista di numeri
min(list)       valore minimo di una lista di numeri
mod(a,p)        resto di a mod p
modulo(a,p)     valore di a mod p
sign(a,b)       restituisce |a| con il segno di b - sign transfer
```

Funzioni intrinseche

Funzioni matematiche

acos(r)	arcocoseno ($\cos^{-1}$) in radianti del numero reale r
asin(r)	arcoseno (sen^{-1}) in radianti del numero reale r
atan(r)	arcotangente ($\tan^{-1}$) in radianti del numero reale r
atan2(r1,r2)	arcotangente ($\tan^{-1}$) in radianti di r1/r2 con segno
cos(r)	coseno del numero reale r
cosh(r)	coseno iperbolico del numero reale r
exp(r)	esponenziale naturale del numero reale r (e^r)
log(r)	logaritmo naturale del numero reale r ($\log_e r$)
log10(r)	logaritmo in base 10 del numero reale r ($\log_{10} r$)
sin(r)	seno del numero reale r
sinh(r)	seno iperbolico del numero reale r
sqrt(r)	radice quadrate del numero reale r – square root
tan(r)	tangente del numero reale r
tanh(r)	tangente iperbolica del numero reale r

Funzioni intrinseche

NOTA: Il codice ASCII, American Standard Code for Information Interchange, è una tabella che elenca le corrispondenze tra simboli grafici e numeri

Funzioni per i caratteri e le stringhe di testo

```
achar(i)          carattere ASCII corrispondente all'intero i (0-127)
char(i)           carattere corrispondente all'intero i
iachar(c)         intero i (1-127) corrispondente al carattere ASCII c
ichar(c)          intero i corrispondente al carattere c
len(s)            lunghezza (numero di caratteri) della stringa s
trim(s)           rimuove gli spazi alla fine della stringa s
len_trim(s)       lunghezza di s, esclusi eventuali spazi finali
repeat(s,n)       unisce n copie della stringa s
index(s1,s2[,back])    posizione della prima (ultima, se back=.true.) occorrenza
                       della stringa s2 come sottostringa di s1 (altrimenti 0)
scan(s1,set[,back])    restituisce la posizione della prima (ultima, se back=.true.)
                       occorrenza di uno dei caratteri set nella stringa s1
verify(s1,set[,back])  posizione della prima (ultima se back=.true.)
                       occorrenza di un carattere di s1 non contenuto in set.
```

Funzioni intrinseche

Funzioni su variabili di tipo `kind`

```
kind(x)                  precisione di x
selected_int_kind(r)     precisione di numero intero tra -10^r e 10^r
selected_real_kind(p,r)  precisione di numero reale con almeno p cifre decimali
                         ed esponente tra ± r
```

Funzioni informative per variabili numeriche

```
digits(a)                numero di cifre significative del numero a
epsilon(r)               epsilon della macchina
huge(a)                  numero più grande che la variabile a può contenere
maxexponent(r)           valore massimo dell'esponente
minexponent(r)           valore minimo dell'esponente
precision(r)             numero di cifre significative
radix(a)                 base della rappresentazione del numero a
range(a)                 esponente decimale del numero a
tiny(a)                  numero più piccolo che la variabile a può contenere
```

Funzioni intrinseche

Funzioni informative per array

```
allocated(arr)          stato di allocazione di un array
lbound(arr,dim)         estremo inferiore di un array per una specifica dimensione
shape(source)           forma dell'array
size(arr,dim)           estensione dell'array lungo una specifica dimensione
ubound(arr, dim)        estremo superiore di un array per una specifica dimensione
```

Funzioni trasformative per array

```
count(mask)                 numero degli elementi veri in un array mask
matmul(matrixA,matrixB)     moltiplica matrici conformi

reshape(source, shape)      costruisce un array di forma specifica
sum(array, mask)            somma gli elementi di un array per cui mask è true
transpose(matrix)           trasposta per un array di rank 2
```

Moduli intrinseci

Un modulo intrinseco è un modulo Fortran predefinito e codificato direttamente dal linguaggio Fortran.

Si accede al contenuto di questo tipo di moduli mediante l'istruzione **USE**, all'inizio della programma o della procedura (funzione, subroutine, altro modulo...).

Tra i più utilizzati

- `ISO_FORTRAN_ENV` contiene costanti caratteristiche di spazio di archiviazione in un particolare computer e anche costanti che definiscono le unità I/O per il particolare computer.

- `ISO_C_BINDING` che contiene i dati necessari ad un compilatore per compilare un programma Fortran in modo che interagisca con il linguaggio C su un dato processore.

- `MODULI IEEE` descrivono le caratteristiche della virgola mobile nello standard IEEE 754 su un particolare processore: `IEEE_EXCEPTIONS`, `IEEE_ARITHMETIC` e `IEEE_FEATURES`.

ESERCIZI

- Scrivere un programma che, date le coordinate cartesiane di un punto, le trasformi in coordinate sferiche e ne verifichi il risultato, stampando i risultati a **video**:

$$\begin{cases} \rho = \sqrt{x^2 + y^2 + z^2} \\ \theta = \arccos\left(\dfrac{z}{\sqrt{x^2 + y^2 + z^2}}\right) \\ \varphi = \arctan\left(\dfrac{y}{x}\right) \end{cases}$$

$$\begin{cases} x = \rho \sin\theta \cos\varphi \\ y = \rho \sin\theta \sin\varphi \\ z = \rho \cos\theta \end{cases}$$

ESERCIZI

Scrivere un programma che, definita la variabile carattere

```
string1 = " modulo funzioni intrinseche "
```

stampi a **video**:

- il risultato della funzione `ADJUSTL` sulla variabile `string1`
- il risultato della funzione `ADJUSTR` sulla variabile `string1`
- il risultato della funzione `INDEX` sulla variabile `string1` ricercando al suo interno la stringa "in"
- il risultato della funzione `LEN_TRIM` sulla variabile `string1`
- il risultato della funzione `SCAN` sulla variabile `string1` ricercando al suo interno la stringa "oit"
- il risultato della funzione `VERIFY` sulla variabile `string1` ricercando al suo interno la stringa "modulo"

GRAZIE

Caterina Caravita
c.caravita@cineca.it