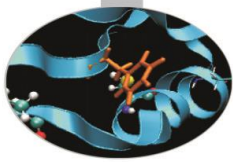
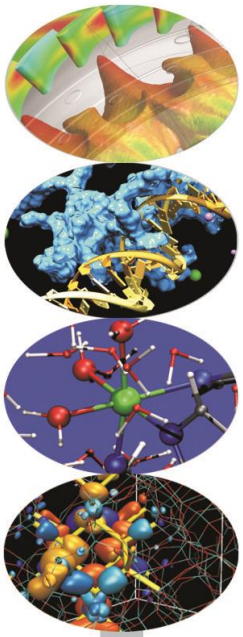
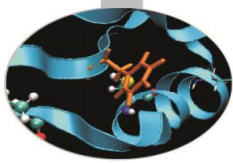


Precisione dei dati

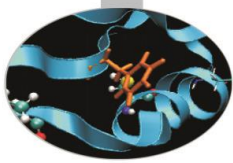


Introduzione



In Fortran abbiamo a disposizione **cinque tipi predefiniti** per dichiarare le variabili: `INTEGER`, `REAL`, `COMPLEX`, `CHARACTER`, `LOGICAL`.
Associati ai **tipi** ci sono le **specie**, definite tramite l'attributo `KIND`, che determinano la quantità di memoria da riservare.

Dal momento che i valori di kind dipendono dal sistema su cui si lavora, il Fortran 90 mette a dsposizione una serie di strumenti sintattici per permettere una completa portabilità del codice.



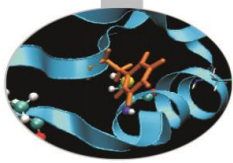
Funzione KIND()

La funzione intrinseca `KIND()` ritorna un intero, dipendente dal sistema, corrispondente alla precisione richiesta.

Tale valore può essere assegnato, in fase di dichiarazione, a un `PARAMETER` oppure direttamente all'attributo `KIND`.

Esempio:

```
INTEGER, PARAMETER :: tipo_doppio = KIND(1.0D0)
REAL(KIND=tipo_doppio) :: a,b
REAL(tipo_doppio) :: c,d
REAL(KIND=KIND(1.0D0)) :: e,f
```



Funzione KIND()

La funzione intrinseca `KIND()` può anche essere utilizzata per **verificare il kind** di una variabile.

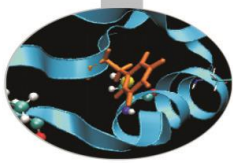
Esempio:

```
INTEGER, PARAMETER :: tipo_doppio = KIND(1.0D0)
   REAL(KIND=tipo_doppio) :: a,b,c
   WRITE(*,*) KIND(a), KIND(b), KIND(c)
```

Esempio portabilità codice:

```
INTEGER, PARAMETER :: tipo_doppio = KIND(1.0D0)
   REAL(tipo_doppio) :: a,b,c !codice portabile
   REAL *8 :: d,e,f ! Sintassi f77 non assicura
                    ! la portabilità.
```

Funzioni e portabilità

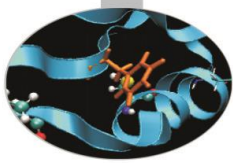


Qualora si intenda lavorare con reali, complessi e interi che richiedano una determinata precisione, indipendentemente dal sistema utilizzato, il Fortran 90 mette a disposizione del programmatore due funzioni intrinseche:

`SELECTED_REAL_KIND()` (reali e complessi)

`SELECTED_INT_KIND()` (interi)

Tramite queste funzioni è assicurata la portabilità del codice da una piattaforma ad un'altra, mantenendo la precisione richiesta.



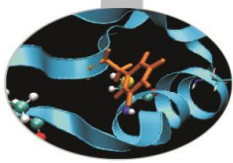
Funzioni e portabilità

La funzione `SELECTED_REAL_KIND(cifre,esp)` si applica a reali e complessi. Essa riceve in argomento due interi che indicano rispettivamente il numero di cifre decimali ed il range dell'esponente e ritorna il kind che permette di rappresentare un numero con la precisione specificata.

Esempio:

```
INTEGER, PARAMETER :: i10 = SELECTED_REAL_KIND (10,200)
REAL (KIND = i10) :: a, b, c
! Le variabili a, b, c hanno almeno 10 cifre decimali di
precisione e l'esponente nel range da -200 a +200.
```

Funzioni e portabilità

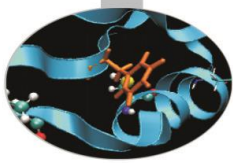


Si deve notare tuttavia che il valore di kind ritornato è quello predefinito per il sistema, perciò usualmente sarà la precisione semplice, doppia, estesa ed eventualmente la quadrupla.

Esempio :

compilatore intel	kind ritornato	precisione
SELECTED_REAL_KIND(3,1)	4	singola
SELECTED_REAL_KIND(8,100)	8	doppia
SELECTED_REAL_KIND(28,100)	-1	non rapp.

compilatore salford	kind ritornato	precisione
SELECTED_REAL_KIND(3,1)	1	singola
SELECTED_REAL_KIND(8,100)	2	doppia
SELECTED_REAL_KIND(28,100)	-1	non rapp.



Funzioni e portabilità

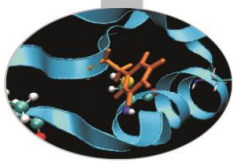
E' bene precisare che l'entità che specifica il `KIND` nelle dichiarazioni deve essere un `PARAMETER`. Essa può venire usata direttamente per specificare il `KIND` delle costanti che compaiono nel programma.

Esempio:

```
INTEGER, PARAMETER :: poca=SELECTED_REAL_KIND(3,1)
INTEGER, PARAMETER :: molta=SELECTED_REAL_KIND(8,1)
REAL(KIND=poca) :: a
REAL(KIND=molta) :: da

   a = 1.0_poca
   da = 1.0_molta
```

Funzioni e portabilità



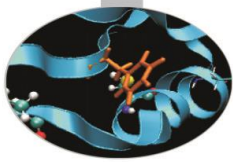
La funzione `SELECTED_INT_KIND(esp)` si applica alle variabili intere. Essa riceve in argomento un intero che indica il range dell'esponente e ritorna il kind che permette di rappresentare un numero con la precisione specificata.

Esempio:

```
INTEGER, PARAMETER :: i8 = SELECTED_INT_KIND ( 8 )
INTEGER (KIND = i8) :: ia, ib, ic
```

Le variabili ia, ib, ic avranno valori compresi fra -10^8 e $+10^8$

Funzioni Intrinseche

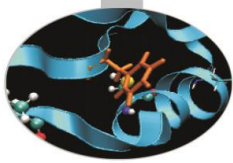


`SELECTED_INT_KIND(R)`	valore dell'attributo KIND del tipo intero che soddisfa l'esponente intero R. -1 se non esiste
`SELECTED_REAL_KIND(P,R)`	valore dell'attributo KIND del tipo reale che soddisfa la precisione decimale P e l'esponente intero R. -1 se non esiste
`KIND(X)`	valore del parametro KIND per l'entità X
`LOGICAL(L[,KIND])`	conversione tra tipi logici

Esempi:

```
INTEGER, PARAMETER :: pr_r=SELECTED_REAL_KIND(8,100)
INTEGER, PARAMETER :: pr_i=SELECTED_INT_KIND(8)
REAL(KIND=pr_r) :: a
REAL(KIND=pr_i) :: b
  write (*,*) kind(a), kind(b)
```

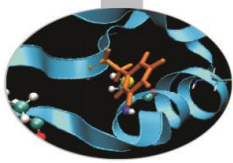
Funzioni Intrinseche



`BIT_SIZE(I)`	numero massimo di bit per il tipo intero rappresentato da I
`CEILING(A)`	il più piccolo intero non inferiore al reale A
`FLOOR(A)`	il più grande intero non superiore al reale A
`MOD(A,P)`	resto della divisione. A - INT(A/P) * P
`MODULO(A,P)`	modulo per argomenti entrambi reali o interi. A - FLOOR(A/P)*P

Esempi:

```
INTEGER :: i=-8,j=5
REAL :: a=2.16
WRITE (*,*) 'Bit size    : ', BIT_SIZE (j)      ! 32
WRITE (*,*) 'Ceiling     : ', CEILING (a)       ! 3
WRITE (*,*) 'Floor       : ', FLOOR (a)         ! 2
WRITE (*,*) 'Mod         : ', MOD (i,j)         ! -3
WRITE (*,*) 'Modulo      : ', MODULO (i,j)      ! 2
```



Funzioni Intrinseche

Le funzioni intrinseche delle slides successive sono basate sul seguente modello di rappresentazione dei reali:

$$x = sb^e \sum_{k=1}^{p} f_k b^{-k}$$

X: valore della variabile reale;

s: segno (+/-1);

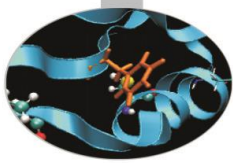
b: base il cui valore è un intero > 1 praticamente sempre uguale a 2;

e: intero pari al valore dell'esponente necessario a rappresentare il valore **x** nella base caratteristica dell'architettura della macchina;

p: numero di bit della mantissa per la rappresentazione in virgola mobile;

f_k : valore del k-esimo digit. Tale valore è compreso tra **0**<= **f_k** < **b**.

Funzioni Intrinseche

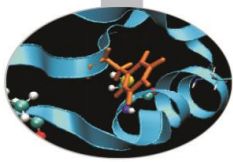


`EXPONENT(X)`	numero di bit dell'esponente nella rappresentazione del reale X ; Corrisponde al valore dell'esponente "e".
`FRACTION(X)`	parte frazionaria del reale X. Il risultato di questa funzione è dato da $X*b^{-e}$ e rappresenta il valore della sommatoria nella rappresentazione sopra riportata.
`NEAREST(X,S)`	il più vicino numero diverso da X nella direzione del segno di S
`SPACING(X)`	spacing assoluto della rappresentazione del reale X; corrisponde al valore risultante da : b^{e-p}.

Esempi:

```
INTEGER, PARAMETER :: ik = SELECTED_REAL_KIND(10)
REAL(ik) :: pi=3.14159265358979323846_ik, in = 1.0_ik
WRITE (*,*) 'Exponent   : ', EXPONENT (pi)    ! 2
WRITE (*,*) 'Fraction   : ', FRACTION (pi)    ! 0.7853..
WRITE (*,*) 'Nearest    : ', NEAREST (pi,in) ! 3.1415..
WRITE (*,*) 'Spacing    : ', RRSPACING (pi)   ! 4.44E-16
```

Funzioni Intrinseche

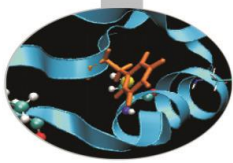


SCALE(X,I)	X*b^I
SET_EXPONENT(X,I)	numero reale la cui parte frazionaria è quella di X ed il cui esponente è I
RRSPACING(X)	reciproco dello spacing relativo nella rappresentazione del reale X

Esempi:

```
INTEGER, PARAMETER :: ik = SELECTED_REAL_KIND(10)
REAL(ik) :: pi=3.14159265358979323846_ik, in = 1.0_ik
INTEGR :: n=3
WRITE (*,*) 'Scale       : ', SCALE (pi,n)         ! 25.13
WRITE (*,*) 'Setexponent : ', SET_EXPONENT (pi,n) ! 6.28
WRITE (*,*) 'RRspacing   : ', RRSPACING (pi)    ! 7.07E+15
```

Funzioni Intrinseche

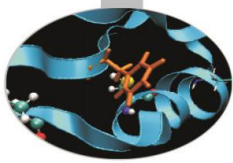


DIGITS(X)	numero di bit significativi per la rappresentazione del numero intero o reale X, tolti il segno e l'esponente
EPSILON(X)	numero più piccolo tale che, aggiunto a 1, gli cambia valore. Il risultato è dello stesso tipo del reale X ed è dato da b^{1-p}.
HUGE(X)	il più grande numero positivo rappresentabile, dello stesso tipo del reale X
MAXEXPONENT(X)	il più grande esponente del 2 per i numeri rappresentabili, dello stesso tipo del reale X

Esempi:

```
INTEGER, PARAMETER :: ik = SELECTED_REAL_KIND(10)
REAL(ik) :: pi=3.14159265358979323846_ik, in = 1.0_ik
WRITE (*,*) 'Digits     : ', DIGITS (pi)    ! 53
WRITE (*,*) 'Epsilon    : ', EPSILON (pi)   ! 2.22E-16
WRITE (*,*) 'Huge       : ', HUGE (pi)      ! 1.79E308
WRITE (*,*) 'Maxexponent  : ', MAXESPONENT (pi) ! 1024
```

Funzioni Intrinseche

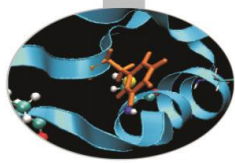


MINEXPONENT(X)	il più grande esponente negativo del 2 per i numeri di tipo X
PRECISION(X)	precisione decimale per i reali dello stesso tipo di X
RADIX(X)	base nella rappresentazione dei numeri dello stesso tipo di X ;
RANGE(X)	esponente decimale massimo per i numeri, interi, reali, complessi dello stesso tipo di X
TINY(X)	il più piccolo numero positivo reale dello stesso tipo di X

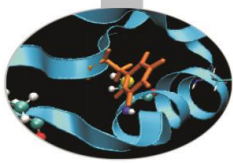
Esempi:

```
INTEGER, PARAMETER :: ik = SELECTED_REAL_KIND(10)
REAL(ik) :: pi=3.14159265358979323846_ik, in = 1.0_ik
WRITE (*,*) 'Minexponent  : ', MINEXPONENT (pi) ! -1021
WRITE (*,*) 'Precision    : ', PRECISION (pi)   ! 15
WRITE (*,*) 'Radix        : ', RADIX (pi)       ! 2
WRITE (*,*) 'Range        : ', RANGE (pi)       ! 307
WRITE (*,*) 'Tiny         : ', RANGE (pi)     ! 2.22E-308
```

Esercizi



1. Scrivere un programma per ottenere i valori di RRSPACING e SPACING usando le funzioni EXPONENT e DIGITS.
2. Scrivere un programma che stampi a video i valori di kind per le variabili intere del sistema
3. Scrivere un programma che ritorna i valori delle funzioni intrinseche DIGITS(), HUGE(), RANGE() per gli interi dei KIND disponibili.
4. Fare la stessa cosa con le funzioni DIGITS(), HUGE(), TINY(), RANGE() applicate ai reali.
5. Scrivere un programma per verificare le differenze tra le funzioni MOD e MODULO.

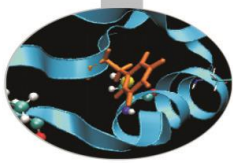


Soluzioni

Esercizio 1/0

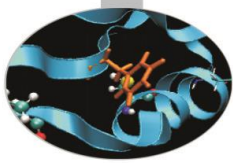
```
program test_spacing
  implicit none
  real(4) :: r4
  real(8) :: r8
  integer(4) :: i4
  real(8) :: esp, dig, rad
  r8 = 1
  do while ( r8 /= 0.0d0 )
   print*," Introdurre un numero:"
   read(*,*) r8
   esp = EXPONENT(r8)
   dig = DIGITS(r8)
   rad = RADIX(r8)
   print*," RRSPACING(",r8,") = ",RRSPACING(r8), &
          abs(r8*(rad**(-esp)))*rad**dig
```

Soluzioni



Esercizio 1/1

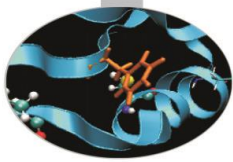
```
print*," SPACING(",r8,") = ",SPACING(r8), &
          rad**(esp-dig)
   r4 = r8
   esp = EXPONENT(r4)
   dig = DIGITS(r4)
   rad = RADIX(r4)
   print*," RRSPACING(",r4,") = ",RRSPACING(r4), &
          abs(r4*(rad**(-esp)))*rad**dig
   print*," SPACING(",r4,") = ",SPACING(r4), &
          rad**(esp-dig)
   end do
end program test_spacing
```



Soluzioni

Esercizio 2

```
PROGRAM kind_int
  IMPLICIT NONE
  INTEGER :: i
  WRITE(*,*)
  WRITE(*,*) '     I       Kind Value'
  WRITE(*,*) '             for int.  '
  WRITE(*,*) '             with range'
  WRITE(*,*) '             +/- 10**I '
  WRITE(*,*)
  DO i = 1, 10
    WRITE(*, *) " SELECTED_INT_KIND(",i,") = ", &
                SELECTED_INT_KIND(i)
  END DO
  STOP
END PROGRAM kind_int
```

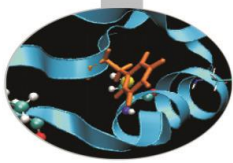


Soluzioni

Esercizio 3/0

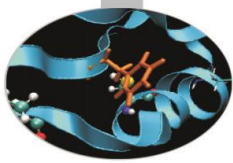
```
PROGRAM machine_constants_int
  IMPLICIT NONE
  INTEGER (KIND = 1) :: i1
  INTEGER (KIND = 2) :: i2
  INTEGER (KIND = 3) :: i3
 INTEGER (KIND = 4) :: i4
   I1 = 23; I2 = 45; I3 = 78 ; I4 = 90
  WRITE (*,*) &
    'Machine constants for integer variables with
different kind values'
  WRITE (*,*)
  WRITE (*,*) 'Default integer kind value: ',
KIND(1)
  WRITE (*,'(/''   ikv   Digits          Huge &
&Range''/)')
```

Soluzioni



Esercizio 3/1

```
WRITE(*,'(a,i7,i20,i5)') "1", DIGITS(i1), HUGE(i1),&
  & RANGE(i1)
  WRITE(*,'(a,i7,i20,i5)') "2", DIGITS(i2),
HUGE(i2),&
  & RANGE(i2)
  WRITE(*,'(a,i7,i20,i5)') "3", DIGITS(i3),
HUGE(i3),&
  & RANGE(i3)
  WRITE(*,'(a,i7,i20,i5)') "4", DIGITS(i4),
HUGE(i4),&
  & RANGE(i4)
  STOP
END PROGRAM machine_constants_int
```



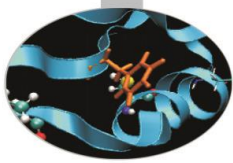
Soluzioni

Esercizio 4/0

```
PROGRAM machine_constants_real


   IMPLICIT NONE
   INTEGER, PARAMETER, DIMENSION(2) :: ikvl = (/
KIND(1.0), KIND(1.0D0) /)
   REAL (KIND = ikvl(1)) :: r1
   REAL (KIND = ikvl(2)) :: r2
   r1 = 1.0
   r2 = 2.0
   WRITE (*,*) &
     'Machine constants for real variables with &
&different kind values'
   WRITE (*,*)
   WRITE (*,*) 'Default precision has kind value: ',
&     KIND(1.0)
```

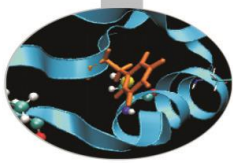
Soluzioni



Esercizio 4/1

```
WRITE (*,'(/''   ikv   Digits          Huge   &
  &         Tiny       Range''/)')
  WRITE(*,'(I5,I7, 7X, E10.2, 6X, E10.2, I7)') &
  ikvl(1), DIGITS(r1), HUGE(r1), TINY(r1), RANGE(r1)
  WRITE(*,'(I5,I7, 7X, E10.2, 6X, E10.2, I7)') &
  ikvl(2), DIGITS(r2), HUGE(r2), TINY(r2), RANGE(r2)

  STOP
END PROGRAM machine_constants_real
```



Soluzioni

Esercizio 5

```
Program mod_modulo
  IMPLICIT NONE
  INTEGER :: a = -100, p = 7
  DO a = -100, 100, 29
     DO p = -7, 7, 5
        WRITE(*,*) 'Original values of a and p: ', a, p
        WRITE(*,*)
        WRITE(*,*) 'MOD     - Remainder of a modulo p: &
&', MOD(a,p)
        WRITE(*,*) 'MODULO  - a modulo p             : &
&', MODULO(a,p)
        READ(*,*)
     END DO
  END DO
  STOP
END PROGRAM mod_modulo
```