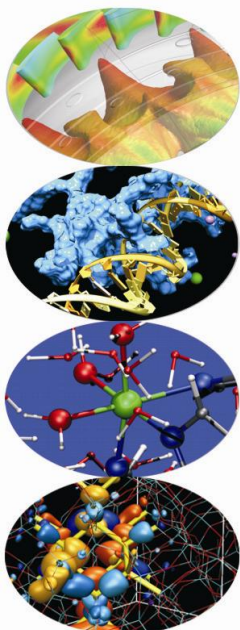
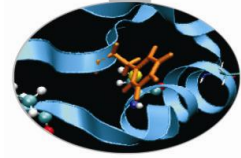


Input Output



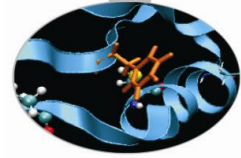


Introduzione

Il salvataggio dei dati su disco e la loro lettura è possibile dopo aver generato un "collegamento" tra le aree disco ed il programma.

Questo collegamento si realizza con l'istruzione `OPEN`, che costruisce un'*immagine* dello spazio disco internamente al programma.

L'immagine così realizzata si chiama **unità di I/O** ed è individuata da un numero intero maggiore di 0.



Introduzione

Accanto alle unità di I/O corrispondenti ad aree disco esistono **unità predefinite**, che corrispondono a dispositivi di I/O particolari, quali stampanti, schermi, tastiere.

In particolare sono spesso predefiniti:

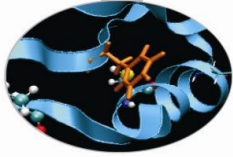
la tastiera (usualmente l'unità 5)

l'output normale su video (usualmente l'unità 6)

i messaggi di errore su video.

Le unità predefinite non devono essere accedute con l'istruzione OPEN.

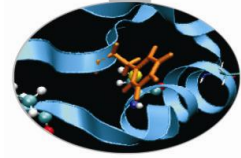
I/O avanzato



Unità predefinite (Fortran 2003)

Il Fortran 2003 introduce il concetto di modulo intrinseco, ovvero moduli predefiniti nel linguaggio Fortran. Uno di questi è `ISO_FORTRAN_ENV`, che contiene tra l'altro le costanti:

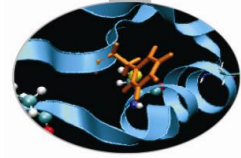
- `INPUT_UNIT` (unità associata alla tastiera)
- `OUTPUT_UNIT` (unità associata allo schermo: output normale)
- `ERROR_UNIT` (unità associata allo schermo: output degli errori)
- `IOSTAT_END` (valore di `IOSTAT=` nel caso si incontri la fine del file)
- `IOSTAT_EOR` (valore di `IOSTAT=` nel caso si incontri la fine del record)



Introduzione

Le istruzioni messe a disposizione dal FORTRAN, legate alle operazioni di I/O, sono le seguenti:

- `OPEN` permette di associare un'area di memoria su disco (file) ad un'unità di I/O del programma
- `CLOSE` rilascia l'associazione tra area disco e unità di I/O
- `INQUIRE` istruzione informativa su unità I/O e file
- `REWIND` riavvolge l'unità di I/O
- `BACKSPACE` permette di tornare indietro di un record nel file.



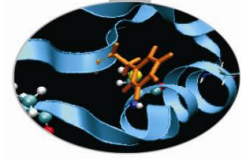
Introduzione

Le sigle che compariranno nella descrizione della sintassi dei comandi hanno i seguenti significati:

- **nn** – INTEGER
- **ss** – CHARACTER
- **bb** – LOGICAL
- **ll** – LABEL

Esempio:

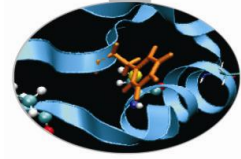
```
OPEN      (UNIT=nn,      FILE=ss,      ACCESS=ss,  
ACTION=ss,  BLANK=ss,    DELIM=ss,    ERR=ll,  
FORM=ss,    IOSTAT=nn,   PAD=ss,     POSITION=ss,  
RECL=nn,   STATUS=ss)
```



Istruzione OPEN

```
OPEN (UNIT=nn, FILE=ss, ACTION=ss,  
ERR=ll, IOSTAT=nn, STATUS=ss)
```

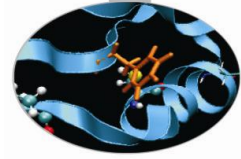
- UNIT (*input*) è l'unità di I/O generata
- FILE (*input*) è l'area disco (file) da accedere
- ACTION (*input*) può avere valori: READ/WRITE/READWRITE
- ERR (*input*) è l'etichetta cui si salta in caso di errori
- IOSTAT (*output*) è 0 se non ci sono errori
- STATUS (*input*) può avere valori:
OLD/NEW/REPLACE/SCRATCH/UNKNOWN



Istruzione CLOSE

CLOSE (UNIT=*nn*, ERR=*ll*, IOSTAT=*nn*,
STATUS=*ss*)

- UNIT (*input*) è l'unità di I/O da liberare
- ERR (*input*) è l'etichetta cui si salta in caso di errori
- IOSTAT (*output*) è 0 se non ci sono errori
- STATUS (*input*) può avere valori: KEEP/DELETE

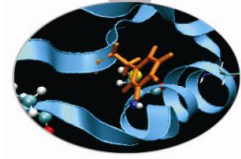


Istruzione INQUIRE

L'istruzione INQUIRE ritorna informazioni sulle aree disco e sulle unità di I/O.

Questa istruzione si utilizza in tre diverse modalità:

1. INQUIRE (UNIT=nn, ...)
 - UNIT (*input*) è un'unità di I/O.
2. INQUIRE (FILE=ss, ...)
 - FILE (*input*) è un'area disco (file).
3. INQUIRE (IOLENGTH=nn) lista
 - Quest'ultima modalità verrà presentata più avanti.



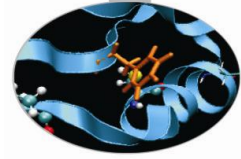
Istruzione INQUIRE

```
INQUIRE (UNIT=nn/FILE=ss, EXIST=bb,  
ACTION=ss, IOSTAT=nn, NAME=ss,  
NAMED=bb, READ=ss, WRITE=ss,  
READWRITE=ss, NUMBER=nn, OPENED=bb)
```

- **EXIST** (*output*) per verificare l'esistenza dell'unità o del file
- **ACTION** (*output*) può avere valori:

READ/WRITE/READWRITE/UNDEFINED

- **IOSTAT** (*output*) 0 se non ci sono errori
- **NAME** (*output*) ritorna il nome del file
- **NAMED** (*output*) per verificare se il file ha un nome esplicito



Modalità di Accesso

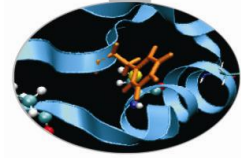
Esistono fondamentalmente due modalità di accesso ai dati delle unità di I/O:

- **sequenziale** - i dati sono letti o scritti di seguito, dall'inizio fino alla fine

```
OPEN (UNIT=nn, FILE=ss,  
      ACCESS='SEQUENTIAL')
```

- **diretta** - è possibile accedere direttamente ai dati ovunque si trovino sull'unità

```
OPEN (UNIT=nn, FILE=ss, ACCESS='DIRECT')
```



Modalità di Accesso

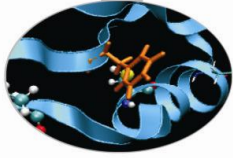
Streaming (Fortran 2003)

È una modalità di accesso a unità di I/O esterne, introdotto per compatibilità con l'ambiente C/C++.

Questa modalità viene definita con la clausola `ACCESS="stream"` dell'istruzione `OPEN`

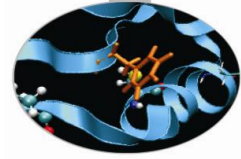
In ogni momento è possibile usare la clausola `POS` dell'istruzione `INQUIRE` per avere la posizione corrente, che quindi può essere utilizzata, anche in un secondo momento, per leggere la riga o riscriverla con le istruzioni `READ` o `WRITE`

Modalità di Accesso



I/O asincrono (Fortran 2003)

Il Fortran 2003 introduce una sintassi che permette di effettuare le operazioni di lettura/scrittura su disco in modalità asincrona rispetto ai calcoli. In generale il fatto che il compilatore riconosca la sintassi non significa che le operazioni di I/O vengano effettuate veramente in parallelo al calcolo. Inoltre il programmatore deve fare attenzione a non modificare le variabili in uso mentre le operazioni di I/O sono in corso.



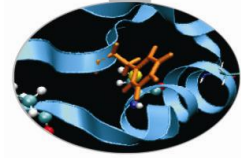
Modalità di Accesso

I/O asincrono (Fortran 2003)

Essenzialmente la richiesta di effettuare un'operazione di I/O asincrono viene fatta con la clausola `ASYNCHRONOUS="YES"` sia per l'accesso al file, che deve essere necessariamente esterno, sia per le operazioni di lettura/scrittura:

```
OPEN (11, ASYNCHRONOUS="YES")
```

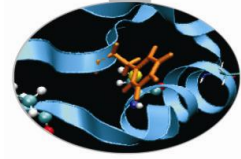
```
WRITE (11, *, ASYNCHRONOUS="YES", ID=scr_a) a
```



Modalità di Accesso

I/O asincrono (Fortran 2003)

La clausola `ID=scr_a` nella `WRITE` ritorna in `scr_a` un valore intero, identificatore dell'operazione di scrittura. Questo potrebbe essere utilizzato per controllare lo stato dell'operazione con l'istruzione `INQUIRE` o per essere sicuri che l'operazione sia terminata, con l'istruzione `WAIT`.



Modalità di Accesso

I/O asincrono (Fortran 2003)

Esempio:

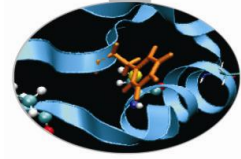
```
OPEN (11, ASYNCHRONOUS="YES")
```

```
. . .
```

```
WRITE (11, *, ASYNCHRONOUS="YES", ID=scr_a) a
```

```
. . .
```

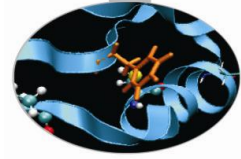
```
WAIT (11, ID=scr_a)
```

Modalità di Accesso

Istruzione FLUSH (Fortran 2003)

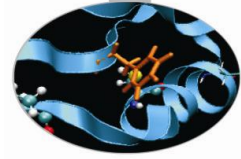
Le operazioni di I/O sono spesso bufferizzate, ovvero i dati non vengono scritti su disco finché un apposito spazio in memoria non viene riempito; questo viene fatto nei calcolatori moderni per ottimizzare l'esecuzione del programma.



Modalità di Accesso

Istruzione FLUSH (Fortran 2003)

Può tuttavia capitare di avere la necessità di leggere il contenuto di un file il più presto possibile durante la sua scrittura da parte di un programma. Per questo il Fortran 2003 mette a disposizione del programmatore l'istruzione `FLUSH (unità)`, utile soprattutto in caso di scrittura con la clausola `ADVANCE="NO"`.



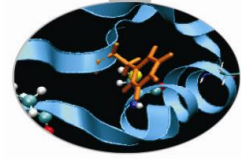
Trasferimento Dati

Istruzioni per il trasferimento dati dalle/sulle unita di I/O:

```
READ (UNIT=nn, FMT=ss, REC=nn, IOSTAT=nn,  
      ADVANCE=ss, SIZE=nn, END=ll, EOR=ll,  
      ERR=ll, NML=ss) lista variabili
```

```
WRITE (UNIT=nn, FMT=ss, REC=nn, IOSTAT=nn,  
       ADVANCE=ss, ERR=ll, NML=ss) lista  
variabili
```

```
PRINT ss, lista variabili
```



Trasferimento Dati

Esistono fondamentalmente due modalità di trasferimento dei dati da/su le unità di I/O:

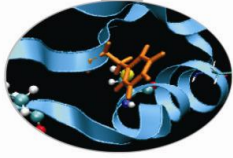
- **formattata** - i dati sono letti o scritti con una rappresentazione leggibile

```
READ (UNIT=nn, FORMAT=ll) lista
```

- **non-formattata** - i dati sono letti o scritti così come sono memorizzati

```
READ (UNIT=nn) lista
```

La modalità di scrittura formattata permette di ottenere file leggibili con un editor, mentre la modalità di scrittura non formattata permette di ottenere file generalmente piu' compatti, ma in genere non portabili da un sistema all'altro.



Accesso Sequenziale Formattato

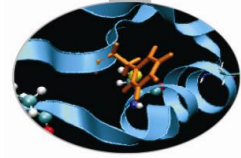
Istruzione OPEN:

```
OPEN (UNIT=nn, FILE=ss, IOSTAT=nn,
```

ACCESS=

```
' SEQUENTIAL ' , FORM=' FORMATTED ' , BLANK=ss ,  
DELIM=ss , PAD=ss , POSITION=ss)
```

- BLANK (input) può avere i valori 'NULL' / 'ZERO '
- DELIM (input) può essere
'QUOTE' / 'APOSTROPHE' / 'NONE '
- PAD (input) può essere 'YES' / 'NO '
- POSITION (input) può essere
'ASIS' / 'REWIND' / 'APPEND '

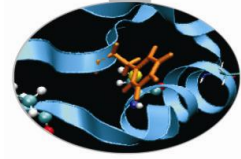


Accesso Sequenziale Formattato

Istruzione INQUIRE :

```
INQUIRE (UNIT=nn/FILE=ss,  
         IOSTAT=nn, ACCESS=ss ,  
         SEQUENTIAL=ss , FORM=ss , FORMATTED=ss ,  
         DELIM=ss , PAD=ss , POSITION=ss)
```

- ACCESS (*output*) dovrebbe essere 'SEQUENTIAL'
- SEQUENTIAL (*output*) dovrebbe essere 'YES'
- FORM (*output*) dovrebbe essere 'FORMATTED'
- FORMATTED (*output*) dovrebbe essere 'YES'
- DELIM (*output*) è il carattere delimitatore delle frasi
- PAD (*output*) è il valore indicato nella OPEN
- POSITION (*output*) è la posizione del puntatore nel file

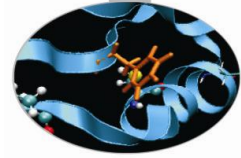


Accesso Sequenziale Formattato

Istruzione READ:

```
READ (UNIT=nn, FMT=ss, IOSTAT=nn, ERR=11,  
END=11, ADVANCE=ss, SIZE=nn, EOR=11)
```

- FMT (*input*) è il descrittore del formato dei dati
- ERR (*etichetta*) salto in caso di errore
- END (*etichetta*) salto per fine lettura
- ADVANCE (*input*) può essere 'YES' / 'NO'
- SIZE (*output*) numero di caratteri letti
- EOR (*etichetta*) salto per fine riga

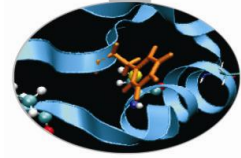


Accesso Sequenziale Formattato

Istruzione WRITE:

```
WRITE (UNIT=nn, FMT=ss, IOSTAT=nn,  
ERR=11, ADVANCE=ss)
```

- FMT (*input*) è il descrittore del formato dei dati
- ERR (*etichetta*) salto in caso di errore
- ADVANCE (*input*) può essere 'YES' / 'NO'

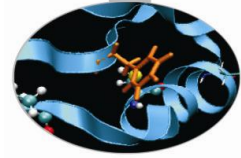


Accesso Sequenziale Formattato

Esempi di formattazione di frasi:

L'istruzione `FORMAT` permette di descrivere come i dati devono essere scritti o letti.

```
CHARACTER*16 :: frase = "Corso Fortran"  
WRITE (*,100) frase  
100 FORMAT (A) !Risultato:  
>CoursobFortranbbb  
100 FORMAT (A6) ! Risultato: >Coursob  
100 FORMAT ("La frase: ",A18)  
! Risultato: >La frase:bbbCorso Fortran  
WRITE (*, ' ("La frase: ",A16) ') frase
```



Accesso Sequenziale Formattato

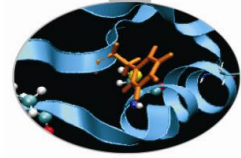
Esempi di formattazione di valori interi:

```
INTEGER :: i, j, k
i = 1; j = 10; k = 100

WRITE (*, '(i4)') k           ! Risultato: >100
WRITE (*, '(3i3)') i, j, k    ! Risultato: >bb1b10100
WRITE (*, '(3(1x,i3))') i, j, k ! Risultato: >bbb1bb10b100
WRITE (*, '(3i4.2)') i, j, k  ! Risultato: >bb01bb10b100
WRITE (*, '(3(1x,i2))') i, j, k ! Risultato: >bb1b10b**
WRITE (*, '(b5)') j           ! Risultato: >b1010
WRITE (*, '(o5)') j           ! Risultato: >bbb12
WRITE (*, '(z5)') j           ! Risultato: >bbbba
```

Se dopo il numero di caratteri si aggiunge un punto ed ancora un numero, questo rappresenta il numero di cifre comunque scritte, eventualmente zeri.

I descrittori di formato B, O, Z, sono utilizzati per le rappresentazioni in formato binario, ottale ed esadecimale.



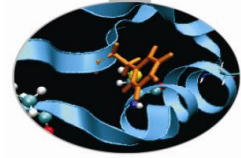
Accesso Sequenziale Formattato

Esempi di formattazione di valori in virgola mobile:

```
REAL(4) :: a, b, c
a = 1.0; b = 10.0; c = 100.0
WRITE (*, '(3f4.0)') a, b, c      ! Risultato: > 1. 10.100.
WRITE (*, '(3f5.1)') a, b, c     ! Risultato: >bb1.0b10.0100.0
WRITE (*, '(1x,e8.0)') c         ! Risultato: >b.100E+03 o
                                   ! Risultato: >bbb.1+003
WRITE (*, '(1x,e8.2)') c         ! Risultato: >bb.10+003
WRITE (*, '(1x,e8.2e2)') c      ! Risultato: >bb.10+03
c = 1.234E10
WRITE (*, '(1x,e10.2)') c       ! Risultato: >bbb.123+011
WRITE (*, '(1x,en10.2)') c     ! Risultato: >bbb12.3+009
WRITE (*, '(1x,es10.2)') c     ! Risultato: >bb1.234+010
```

Il numero che può essere indicato dopo il descrittore rappresenta, al solito, il numero di caratteri del campo numerico prodotto o letto.

In questi descrittori, il numero che compare dopo il punto rappresenta il numero di decimali stampati.



Accesso Sequenziale Formattato

Esempi di formattazione di valori generici:

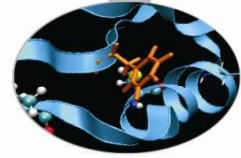
```
LOGICAL :: v, f
REAL(4) :: a, b, c

a = 10.0; b = 1000.0; c = 1.234E10
v = .TRUE.; f = .FALSE.

WRITE (*, '(3G5.1)') a, b, c      ! Risultato: >bb1.0b10.0100.0
WRITE (*, '(A,L3)') "C > B: ",v ! Risultato: >C > B: bbT
WRITE (*, '(1X,G10.2)') c
      ! Risultato: >bbbbbb10.bbbbbbb0.10E+04bbb.12E+11
WRITE (*, '(4G8.2)') "C=",c," > 0: ",v
      ! Risultato: >bbbbbbC=0.12E+11bbb>b0:bbbbbbbbT
```

Il descrittore L è specifico per trasferire valori booleani.

Il descrittore G può essere usato per valori numerici di qualsiasi tipo; la sintassi è identica al descrittore E e nella sua forma più completa ha senso solo per numeri molto grandi o piccoli.



Accesso Sequenziale Formattato

Esempi di controllo dell'incolonnamento:

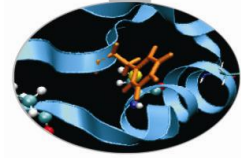
```
REAL(4) :: a, b, c
  REAL(4), dimension(3) :: m
  a = 10.0; b = 100.0; c = 1000.0

  WRITE (11, '(3G6.2)') a, b, c ! Risultato: >bbb1.0b100.01000.0
  REWIND(11)
  READ (11, '(3(G6.2,TL18))') m
  WRITE (*, '(3(TR8,G6.2,/))') m
```

Il descrittore T, TL, TR sposta la posizione del puntatore di lettura o scrittura di tanti caratteri quanti sono quelli indicati, in valore assoluto o relativo verso sinistra o destra rispettivamente.

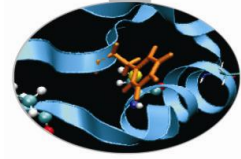
Il descrittore / genera o salta ad una nuova riga.

Accesso Sequenziale Formattato



Esempio riepilogativo di formattazione:

```
OPEN (11,STATUS='SCRATCH',IOSTAT=ioc)
  WRITE (*,'(A)',ADVANCE='no',IOSTAT=ioc) "Quanti numeri? "
  READ (*,*,IOSTAT=ioc) numeri
  DO i = 1, numeri
    WRITE (11,'(I4)',IOSTAT=ioc) (i+numeri)/2
  END DO
  ENDFILE (11)
  REWIND (11)
  DO i = 1, numeri
    READ (11,*,IOSTAT=ioc) j
    PRINT*,"Alla riga ",i,"letto il numero ",j
  END DO
  CLOSE (11)
```



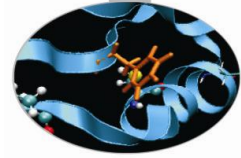
Accesso Sequenziale Formattato

Il trasferimento dei dati con READ e WRITE è possibile anche senza coinvolgere unità esterne:

```
REAL(4), DIMENSION(10) :: vr
INTEGER, DIMENSION(10) :: vi
CHARACTER :: d*1, formato*12, dati*80
INTEGER :: l

READ (*, '(A1,I1,A)') d, l, dati
WRITE (formato, '(A3,A1,I1,A1)') "(10",d,l,")"
IF ( d .EQ. "f" ) THEN
    READ (dati,formato) vr
ELSE
    READ (dati,formato) vi
END IF
```

Si deve fare attenzione tuttavia a definire bene la lunghezza dei campi ed il descrittore; in questi casi non è possibile usare il formato libero.

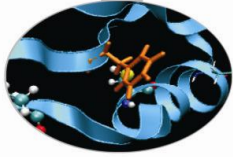


Accesso Streaming Formattato

Streaming (Fortran 2003)

Esempio:

```
OPEN(11,FILE="frasi.txt", ACCESS="STREAM", &  
      FORM="FORMATTED")  
WRITE(11,*) riga  
INQUIRE(11,POS=prec)  
PRINT*," POS = ", prec  
WRITE(11,*) riga  
INQUIRE(11,POS=prec)  
PRINT*," POS = ", prec  
WRITE(11,*) riga  
riga = "Questa riga trascrive quella precedente"  
WRITE(11,"(A)",POS=prec) riga
```

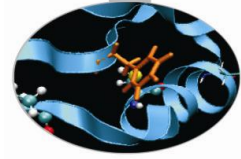



Accesso Sequenziale Formattato

Utilità predefinite (Fortran 2003)

Dal Fortran 2003 vengono introdotte altre novità, utili a gestire le operazioni di I/O.

Tra queste si segnalano la nuova clausola `IOMSG=`, che utilizzata con le istruzioni `OPEN`, `READ`, `WRITE`, ritorna un messaggio esplicativo di un eventuale errore. Utilizzata insieme alle nuove funzioni intrinseche `IS_IOSTAT_EOR(st)` (ritorna `.TRUE.` Se si è raggiunta la fine record) e `IS_IOSTAT_END(st)` (ritorna `.TRUE.` Se si è raggiunta la fine file) permette di scrivere programmi Fortran 2003 portabili ed efficaci.



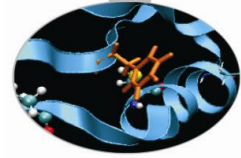
Accesso Sequenziale Formattato

Utilità predefinite (Fortran 2003)

Esempio:

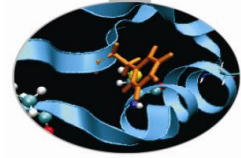
```
USE ISO_FORTRAN_ENV
. . .
CHARACTER(256) :: msg
!
OPEN(11, FILE="MatriceA.dat", &
      IOMSG=msg, IOSTAT=st, STATUS="OLD")
IF ( st /= 0 ) THEN
  WRITE (ERROR_UNIT,*) "Errore: ", &
    TRIM(msg)
  STOP
END IF
```

Accesso Sequenziale Formattato



Utilità predefinite (Fortran 2003)

```
READ (11, "(G10.2)", ADVANCE="NO", &  
      IOMSG=msg, IOSTAT=st) b(i,j)  
IF ( st /= 0 ) THEN  
  WRITE (ERROR_UNIT,*) "Errore: ", &  
    TRIM(msg)  
  IF ( IS_IOSTAT_END(st) ) THEN  
    WRITE (ERROR_UNIT,*) "Fine File"  
    . . .  
  ELSE IF ( IS_IOSTAT_EOR(st) ) THEN  
    . . .  
END IF
```



Accesso Diretto Non Formattato

Accesso diretto non-formattato.

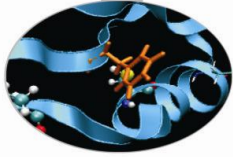
- è la modalità più efficiente per trasferire grandi quantità di dati
- è anche una delle modalità meno usate

```
OPEN (UNIT=nn, FILE=ss, IOSTAT=nn,  
      ACCESS='DIRECT' , FORM='UNFORMATTED' ,  
      RECL=nn)
```

- RECL indica la lunghezza di ogni record

Qualora si debbano trasferire grandi quantità di dati, è conveniente usare la modalità non-formattata, il processo è molto più veloce ed è possibile accedere direttamente ad un punto del file.

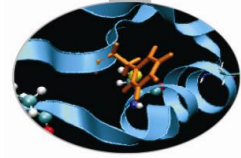
Occorre fare attenzione alla portabilità da un sistema all'altro.



Accesso Diretto Non Formattato

```
INQUIRE (UNIT=nn/FILE=ss, IOSTAT=nn,  
ACCESS=ss, DIRECT=ss, RECL=nn,  
FORM=ss, UNFORMATTED=ss,  
NEXTREC=nn)
```

- ACCESS (*output*) dovrebbe essere 'DIRECT'
- DIRECT (*output*) dovrebbe essere 'YES'
- RECL (*output*) è la lunghezza record del file
- FORM (*output*) dovrebbe essere 'UNFORMATTED'
- UNFORMATTED (*output*) dovrebbe essere 'YES'
- NEXTREC (*output*) è il numero del record successivo all'ultimo letto



Accesso Diretto Non Formattato

```
READ (UNIT=nn, IOSTAT=nn, ERR=11, REC=nn)
```

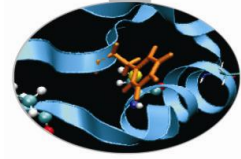
```
WRITE (UNIT=nn, IOSTAT=nn, ERR=11, REC=nn)
```

- REC (*input*) è il numero di record da leggere

```
INQUIRE (IOLENGTH=nn) lista
```

- IOLENGTH (*output*) è la lunghezza della lista

La sintassi delle istruzioni READ e WRITE è particolarmente semplice in questa modalità. Poiché i record trasferiti devono avere la lunghezza corretta, prima di leggere o scrivere su file, è utile usare la forma indicata dell'istruzione INQUIRE, che ritorna la lunghezza, usualmente in byte, dell'intera lista.



Accesso Diretto Non Formattato

Esempio:

```
INTEGER, DIMENSION(5,10) :: mat
```

```
INQUIRE (IOLENGTH=lunrec) mat
```

```
OPEN (11, FILE = 'salva.dun', RECL = lunrec, &  
      & ACCESS='direct', FORM = 'UNFORMATTED', &  
      & STATUS="REPLACE")
```

```
WRITE (11,REC=1,IOSTAT=n) mat
```

```
CLOSE (11)
```

Anche se è possibile utilizzare lo stesso file con modalità di accesso differenti, sequenziale e diretta, la cosa non è consigliabile e non può essere garantita la portabilità.