Fortran-Ein-/Ausgabe

Quell-Programm

E/A-Anweisungen

```
READ
WRITE
PRINT
OPEN
CLOSE
REWIND
INQUIRE
BACKSPACE
ENDFILE
...
FORMAT
```

logische Dateinummer (LUN)

Geräte-/Dateizuordnung
Quelltext $\leftrightarrow$ Betriebssystem
(explizit oder implizit)

LUN	–	phys. Datei
Fortran IV (IBM/OS)		
1	–	LKL
2	–	LKS
3	–	Drucker
Fortran 77 ff.		
5	–	`stdin`
6	–	`stdout`
0	–	`stderr`
`1-4,7-?`	–	beliebig

Peripherie

E/A-Geräte

Tastatur
Bildschirm
Drucker
Plotter
Harddisk (Datei)
CD
...

Programmstart:
`stdin` $\leftarrow$ Tastatur
`stdout` $\rightarrow$ Terminal
`stderr` $\rightarrow$ Terminal

Fortran-Ein-/Ausgabe

Quell-Programm

E/A-Anweisungen

```
READ(5,*) n,m
WRITE
PRINT
OPEN
CLOSE
REWIND
INQUIRE
BACKSPACE
ENDFILE
...
FORMAT
```

logische Dateinummer (LUN)

Geräte-/Dateizuordnung
Quelltext $\leftrightarrow$ Betriebssystem
(explizit oder implizit)

LUN	–	phys. Datei
Fortran IV (IBM/OS)		
1	–	LKL
2	–	LKS
3	–	Drucker
Fortran 77 ff.		
5	–	stdin
6	–	stdout
0	–	stderr
1-4,7-?	–	beliebig

Peripherie

E/A-Geräte

Tastatur
Bildschirm
Drucker
Plotter
Harddisk (Datei)
CD
...

Programmstart:
stdin $\leftarrow$ Tastatur
stdout $\rightarrow$ Terminal
stderr $\rightarrow$ Terminal

Fortran-Ein-/Ausgabe

Quell-Programm

E/A-Anweisungen

```
READ(5,*) n,m
WRITE(6,*) x,y
PRINT *, x,y
OPEN
CLOSE
REWIND
INQUIRE
BACKSPACE
ENDFILE
...
FORMAT
```

logische Dateinummer (LUN)

Geräte-/Dateizuordnung
Quelltext $\leftrightarrow$ Betriebssystem
(explizit oder implizit)

LUN	–	phys. Datei
Fortran IV (IBM/OS)		
1	–	LKL
2	–	LKS
3	–	Drucker
Fortran 77 ff.		
5	–	`stdin`
6	–	`stdout`
0	–	`stderr`
1-4,7-?	–	beliebig

Peripherie

E/A-Geräte

Tastatur
Bildschirm
Drucker
Plotter
Harddisk (Datei)
CD
...

Programmstart:
`stdin` $\leftarrow$ Tastatur
`stdout` $\rightarrow$ Terminal
`stderr` $\rightarrow$ Terminal

Fortran-Ein-/Ausgabe

Verschiedene Arten des Zugriffs auf Dateien

- sequentiell – Datensätze entsprechend ihrer physischen Reihenfolge verarbeitet (Standardannahme).
- direkt – Lese-/Schreibzugriff auf beliebigen (k-ten) Datensatz aus einer Gesamtheit (erfordert spezielle Vereinbarung mit `OPEN`-Anweisung).
- Grundsätzlich pro `Read`- oder `WRITE`-Anweisung ein Datensatz; abweichend davon durch Formatsteuerung auch mehrere Datensätze mit einer Anweisung oder Fortsetzung eines Datensatzes mit nachfolgender `READ/WRITE`-Anweisung.

Datenformatierung

- `FORMATTED` – interne Daten werden durch explizite Formatangabe oder standardmäßig in (für Menschen lesbare) Zeichenketten umgewandelt.
- `UNFORMATTED`/`BINARY` – Daten werden in ihrer internen Darstellung übertragen (ohne Konvertierung, also auch ohne Rundungsfehler).

Fortran-Ein-/Ausgabe

READ/WRITE-Anweisungen (FORMATTED)

$$\left\{\begin{array}{l}\texttt{READ}\\ \texttt{WRITE}\end{array}\right\}\texttt{(}\textit{control-specs}\texttt{) [ioliste]}$$

---

control-specs (Auswahl):

`[UNIT=]nr`	logische Dateinummer, die einer Datei zugeordnet ist
`,[FMT=]fspec`	Marke einer Formatanweisung / * / Format-String
`[,END=marke]`	bei `READ`: Sprungziel bei Datei-Ende (EOF)
`[,ERR=marke]`	Sprungziel bei E/A-Fehler (Zugriffsrechte, Datenfehler)
`[,IOSTAT=ivar]`	Zustandsvariable (`ivar=0` nach fehlerfreier E/A)
`[,REC=iausdr]`	bei Direktzugriff: Nummer des Datensatzes

`ioliste`	Liste von Variablen (bei `WRITE` auch Ausdrücke)

Kurzformen für stdin/stdout (evtl. veraltet):

```
READ fspec, ioliste    → UNIT=5
PRINT fspec, ioliste   → UNIT=6
```

READ/WRITE-Anweisungen (UNFORMATTED)

$$\left\{\begin{array}{l}\texttt{READ}\\ \texttt{WRITE}\end{array}\right\}\texttt{(}\textit{control-specs}\texttt{) [ioliste]}$$

---

control-specs (Auswahl):

`[UNIT=]nr`	logische Dateinummer, die einer Datei zugeordnet ist
`,[FMT=]fspec`	
`[,END=marke]`	bei `READ`: Sprungziel bei Datei-Ende (EOF)
`[,ERR=marke]`	Sprungziel bei E/A-Fehler (Zugriffsrechte, Datenfehler)
`[,IOSTAT=ivar]`	Zustandsvariable (`ivar=0` nach fehlerfreier E/A)
`[,REC=iausdr]`	bei Direktzugriff: Nummer des Datensatzes

`ioliste`	Liste von Variablen (bei `WRITE` auch Ausdrücke)

Kurzformen für stdin/stdout (evtl. veraltet):

```
READ fspec, ioliste    → UNIT=5
PRINT fspec, ioliste   → UNIT=6
```

READ/WRITE-Anweisungen

$$\left\{\begin{array}{l}\texttt{READ}\\ \texttt{WRITE}\end{array}\right\}\texttt{(}\textit{control-specs}\texttt{) [ioliste]}$$

---

control-specs (Auswahl):

`[UNIT=]nr`	logische Dateinummer, die einer Datei zugeordnet ist
`[,[FMT=]fspec]`	Marke einer Formatanweisung / * / Format-String
`[,END=marke]`	bei `READ`: Sprungziel bei Datei-Ende (EOF)
`[,ERR=marke]`	Sprungziel bei E/A-Fehler (Zugriffsrechte, Datenfehler)
`[,IOSTAT=ivar]`	Zustandsvariable (`ivar=0` nach fehlerfreier E/A)
`[,REC=iausdr]`	bei Direktzugriff: Nummer des Datensatzes

`ioliste`	Liste von Variablen (bei `WRITE` auch Ausdrücke)

Kurzformen für stdin/stdout (evtl. veraltet):

```
READ fspec, ioliste    → UNIT=5
PRINT fspec, ioliste   → UNIT=6
```

Fortran-Ein-/Ausgabe

Beispiele

```
   WRITE(*,*) 'Bitte n eingeben:'
   READ(*,*) n
1  WRITE(*,'(A,$)') 'und jetzt m:'
   READ(*,*,ERR=2,END=3) m
   IF (n .GT. m) THEN
     WRITE(*,5) '>'
   ELSE IF (n .LT. m) THEN
     WRITE(*,5) '<'
   ELSE
     WRITE(*,5) '='
   ENDIF
5  FORMAT('n ',A1,' m')
   GOTO 1
2  WRITE(*,*) 'Fehler!'
   GOTO 1
3  CONTINUE
```

Terminal

-

Fortran-Ein-/Ausgabe

Beispiele

```
   WRITE(6,*) 'Bitte n eingeben:'
   READ(5,*) n
1  WRITE(*,'(A,$)') 'und jetzt m:'
   READ(*,*,ERR=2,END=3) m
   IF (n .GT. m) THEN
     WRITE(*,5) '>'
   ELSE IF (n .LT. m) THEN
     WRITE(*,5) '<'
   ELSE
     WRITE(*,5) '='
   ENDIF
5  FORMAT('n ',A1,' m')
   GOTO 1
2  WRITE(*,*) 'Fehler!'
   GOTO 1
3  CONTINUE
```

Terminal

-

Fortran-Ein-/Ausgabe

Beispiele

```
    WRITE(6,*) 'Bitte n eingeben:'
    READ(5,*) n
1   WRITE(*,'(A,$)') 'und jetzt m:'
    READ(*,*,ERR=2,END=3) m
    IF (n .GT. m) THEN
      WRITE(*,5) '>'
    ELSE IF (n .LT. m) THEN
      WRITE(*,5) '<'
    ELSE
      WRITE(*,5) '='
    ENDIF
5   FORMAT('n ',A1,' m')
    GOTO 1
2   WRITE(*,*) 'Fehler!'
    GOTO 1
3   CONTINUE
```

Terminal

```
Bitte n eingeben:
-
```

Fortran-Ein-/Ausgabe

Beispiele

```
    WRITE(6,*) 'Bitte n eingeben:'
    READ(5,*) n
1   WRITE(*,'(A,$)') 'und jetzt m:'
    READ(*,*,ERR=2,END=3) m
    IF (n .GT. m) THEN
      WRITE(*,5) '>'
    ELSE IF (n .LT. m) THEN
      WRITE(*,5) '<'
    ELSE
      WRITE(*,5) '='
    ENDIF
5   FORMAT('n ',A1,' m')
    GOTO 1
2   WRITE(*,*) 'Fehler!'
    GOTO 1
3   CONTINUE
```

Terminal

```
Bitte n eingeben:
123
und jetzt m:
=
```

Fortran-Ein-/Ausgabe

Beispiele

```
    WRITE(6,*) 'Bitte n eingeben:'
    READ(5,*) n
1   WRITE(*,'(A,$)') 'und jetzt m:'
    READ(*,*,ERR=2,END=3) m
    IF (n .GT. m) THEN
      WRITE(*,5) '>'
    ELSE IF (n .LT. m) THEN
      WRITE(*,5) '<'
    ELSE
      WRITE(*,5) '='
    ENDIF
5   FORMAT('n ',A1,' m')
    GOTO 1
2   WRITE(*,*) 'Fehler!'
    GOTO 1
3   CONTINUE
```

Terminal

```
Bitte n eingeben:
123
und jetzt m:_
```

Fortran-Ein-/Ausgabe

Beispiele

```
    WRITE(6,*) 'Bitte n eingeben:'
    READ(5,*) n
1   WRITE(*,'(A,$)') 'und jetzt m:'
    READ(*,*,ERR=2,END=3) m
    IF (n .GT. m) THEN
      WRITE(*,5) '>'
    ELSE IF (n .LT. m) THEN
      WRITE(*,5) '<'
    ELSE
      WRITE(*,5) '='
    ENDIF
5   FORMAT('n ',A1,' m')
    GOTO 1
2   WRITE(*,*) 'Fehler!'
    GOTO 1
3   CONTINUE
```

Terminal

```
Bitte n eingeben:
123
und jetzt m:45
-
```

Fortran-Ein-/Ausgabe

Beispiele

```
    WRITE(6,*) 'Bitte n eingeben:'
    READ(5,*) n
1   WRITE(*,'(A,$)') 'und jetzt m:'
    READ(*,*,ERR=2,END=3) m
    IF (n .GT. m) THEN
      WRITE(*,5) '>'
    ELSE IF (n .LT. m) THEN
      WRITE(*,5) '<'
    ELSE
      WRITE(*,5) '='
    ENDIF
5   FORMAT('n ',A1,' m')
    GOTO 1
2   WRITE(*,*) 'Fehler!'
    GOTO 1
3   CONTINUE
```

Terminal

```
Bitte n eingeben:
123
und jetzt m:45
n > m
-
```

Fortran-Ein-/Ausgabe

Beispiele

```
    WRITE(6,*) 'Bitte n eingeben:'
    READ(5,*) n
1   WRITE(*,'(A,$)') 'und jetzt m:'
    READ(*,*,ERR=2,END=3) m
    IF (n .GT. m) THEN
      WRITE(*,5) '>'
    ELSE IF (n .LT. m) THEN
      WRITE(*,5) '<'
    ELSE
      WRITE(*,5) '='
    ENDIF
5   FORMAT('n ',A1,' m')
    GOTO 1
2   WRITE(*,*) 'Fehler!'
    GOTO 1
3   CONTINUE
```

Terminal

```
Bitte n eingeben:
123
und jetzt m:45
n > m
-
```

Fortran-Ein-/Ausgabe

Beispiele

```
   WRITE(6,*) 'Bitte n eingeben:'
   READ(5,*) n
1  WRITE(*,'(A,$)') 'und jetzt m:'
   READ(*,*,ERR=2,END=3) m
   IF (n .GT. m) THEN
     WRITE(*,5) '>'
   ELSE IF (n .LT. m) THEN
     WRITE(*,5) '<'
   ELSE
     WRITE(*,5) '='
   ENDIF
5  FORMAT('n ',A1,' m')
   GOTO 1
2  WRITE(*,*) 'Fehler!'
   GOTO 1
3  CONTINUE
```

Terminal

```
Bitte n eingeben:
123
und jetzt m:45
n > m
und jetzt m:_
```

Fortran-Ein-/Ausgabe

Beispiele

```
    WRITE(6,*) 'Bitte n eingeben:'
    READ(5,*) n
1   WRITE(*,'(A,$)') 'und jetzt m:'
    READ(*,*,ERR=2,END=3) m
    IF (n .GT. m) THEN
      WRITE(*,5) '>'
    ELSE IF (n .LT. m) THEN
      WRITE(*,5) '<'
    ELSE
      WRITE(*,5) '='
    ENDIF
5   FORMAT('n ',A1,' m')
    GOTO 1
2   WRITE(*,*) 'Fehler!'
    GOTO 1
3   CONTINUE
```

Terminal

```
Bitte n eingeben:
123
und jetzt m:45
n > m
und jetzt m:xc
-
```

Fortran-Ein-/Ausgabe

Beispiele

```
    WRITE(6,*) 'Bitte n eingeben:'
    READ(5,*) n
1   WRITE(*,'(A,$)') 'und jetzt m:'
    READ(*,*,ERR=2,END=3) m
    IF (n .GT. m) THEN
      WRITE(*,5) '>'
    ELSE IF (n .LT. m) THEN
      WRITE(*,5) '<'
    ELSE
      WRITE(*,5) '='
    ENDIF
5   FORMAT('n ',A1,' m')
    GOTO 1
2   WRITE(*,*) 'Fehler!'
    GOTO 1
3   CONTINUE
```

Terminal

```
Bitte n eingeben:
123
und jetzt m:45
n > m
und jetzt m:xc
Fehler!
-
```

Fortran-Ein-/Ausgabe

Beispiele

```
    WRITE(6,*) 'Bitte n eingeben:'
    READ(5,*) n
1   WRITE(*,'(A,$)') 'und jetzt m:'
    READ(*,*,ERR=2,END=3) m
    IF (n .GT. m) THEN
      WRITE(*,5) '>'
    ELSE IF (n .LT. m) THEN
      WRITE(*,5) '<'
    ELSE
      WRITE(*,5) '='
    ENDIF
5   FORMAT('n ',A1,' m')
    GOTO 1
2   WRITE(*,*) 'Fehler!'
    GOTO 1
3   CONTINUE
```

Terminal

```
Bitte n eingeben:
123
und jetzt m:45
n > m
und jetzt m:xc
Fehler!
und jetzt m:_
```

Fortran-Ein-/Ausgabe

Beispiele

```
    WRITE(6,*) 'Bitte n eingeben:'
    READ(5,*) n
1   WRITE(*,'(A,$)') 'und jetzt m:'
    READ(*,*,ERR=2,END=3) m
    IF (n .GT. m) THEN
      WRITE(*,5) '>'
    ELSE IF (n .LT. m) THEN
      WRITE(*,5) '<'
    ELSE
      WRITE(*,5) '='
    ENDIF
5   FORMAT('n ',A1,' m')
    GOTO 1
2   WRITE(*,*) 'Fehler!'
    GOTO 1
3   CONTINUE
```

Terminal

```
Bitte n eingeben:
123
und jetzt m:45
n > m
und jetzt m:xc
Fehler!
und jetzt m:678
-
```

Fortran-Ein-/Ausgabe

Beispiele

```
    WRITE(6,*) 'Bitte n eingeben:'
    READ(5,*) n
1   WRITE(*,'(A,$)') 'und jetzt m:'
    READ(*,*,ERR=2,END=3) m
    IF (n .GT. m) THEN
      WRITE(*,5) '>'
    ELSE IF (n .LT. m) THEN
      WRITE(*,5) '<'
    ELSE
      WRITE(*,5) '='
    ENDIF
5   FORMAT('n ',A1,' m')
    GOTO 1
2   WRITE(*,*) 'Fehler!'
    GOTO 1
3   CONTINUE
```

Terminal

```
Bitte n eingeben:
123
und jetzt m:45
n > m
und jetzt m:xc
Fehler!
und jetzt m:678
n < m
-
```

Fortran-Ein-/Ausgabe

Beispiele

```
    WRITE(6,*) 'Bitte n eingeben:'
    READ(5,*) n
1   WRITE(*,'(A,$)') 'und jetzt m:'
    READ(*,*,ERR=2,END=3) m
    IF (n .GT. m) THEN
      WRITE(*,5) '>'
    ELSE IF (n .LT. m) THEN
      WRITE(*,5) '<'
    ELSE
      WRITE(*,5) '='
    ENDIF
5   FORMAT('n ',A1,' m')
    GOTO 1
2   WRITE(*,*) 'Fehler!'
    GOTO 1
3   CONTINUE
```

Terminal

```
Bitte n eingeben:
123
und jetzt m:45
n > m
und jetzt m:xc
Fehler!
und jetzt m:678
n < m
_
```

Fortran-Ein-/Ausgabe

Beispiele

```
   WRITE(6,*) 'Bitte n eingeben:'
   READ(5,*) n
1  WRITE(*,'(A,$)') 'und jetzt m:'
   READ(*,*,ERR=2,END=3) m
   IF (n .GT. m) THEN
     WRITE(*,5) '>'
   ELSE IF (n .LT. m) THEN
     WRITE(*,5) '<'
   ELSE
     WRITE(*,5) '='
   ENDIF
5  FORMAT('n ',A1,' m')
   GOTO 1
2  WRITE(*,*) 'Fehler!'
   GOTO 1
3  CONTINUE
```

Terminal

```
Bitte n eingeben:
123
und jetzt m:45
n > m
und jetzt m:xc
Fehler!
und jetzt m:678
n < m
und jetzt m:_
```

Fortran-Ein-/Ausgabe

Beispiele

```
    WRITE(6,*) 'Bitte n eingeben:'
    READ(5,*) n
1   WRITE(*,'(A,$)') 'und jetzt m:'
    READ(*,*,ERR=2,END=3) m
    IF (n .GT. m) THEN
      WRITE(*,5) '>'
    ELSE IF (n .LT. m) THEN
      WRITE(*,5) '<'
    ELSE
      WRITE(*,5) '='
    ENDIF
5   FORMAT('n ',A1,' m')
    GOTO 1
2   WRITE(*,*) 'Fehler!'
    GOTO 1
3   CONTINUE
```

Terminal

```
Bitte n eingeben:
123
und jetzt m:45
n > m
und jetzt m:xc
Fehler!
und jetzt m:678
n < m
und jetzt m:^D  (EOF)
_
```

Fortran-Ein-/Ausgabe

Beispiele

```
    WRITE(6,*) 'Bitte n eingeben:'
    READ(5,*) n
1   WRITE(*,'(A,$)') 'und jetzt m:'
    READ(*,*,ERR=2,END=3) m
    IF (n .GT. m) THEN
      WRITE(*,5) '>'
    ELSE IF (n .LT. m) THEN
      WRITE(*,5) '<'
    ELSE
      WRITE(*,5) '='
    ENDIF
5   FORMAT('n ',A1,' m')
    GOTO 1
2   WRITE(*,*) 'Fehler!'
    GOTO 1
3   CONTINUE
```

Terminal

```
Bitte n eingeben:
123
und jetzt m:45
n > m
und jetzt m:xc
Fehler!
und jetzt m:678
n < m
und jetzt m:^D  (EOF)
_
```

Formatsteuerung

- Formatspezifikation als zweite Angabe bei `READ( )` oder `WRITE( )`:
  - `*` – Standardformat entsprechend dem Typ der Variablen
  - `marke` – Verweis auf Format-Anweisung `marke  FORMAT(fliste)`
  - `'(fliste)'` – direkte Angabe der Formatliste in `READ` bzw. `WRITE`
- `fliste` enthält (durch Komma getrennte) Formatelemente,
  - je ein Datenformatelement pro Variable bzw. Feldelement (für komplexe Variable 2 Elemente), z. B. `I,F,E,D,G,L,A,B,O,Z`
  - dazwischen beliebig Steuerelemente (ohne direkten Bezug zu angegebenen Variablen), z. B. `X,T,H,'...',P,/,$,:,B,S`
- Typische Angabe eines Formatelements `nFw.d`, wobei:
  `n` Wiederholungsfaktor, `w` Gesamtzeichenzahl, `d` ggf. Zusatzangabe wie 'davon Anzahl Dezimalziffern' – alles ganzzahlige Konstanten!
- Gruppierung durch Klammerung, z.B. `FORMAT(1X,5(I2,':',F6.2))`

Formatsteuerung

Beachte … (als Motivation;-)

Formatierung dient „lediglich“ der optischen Verschönerung des laufenden Programms.

Wozu?

Die Qualität eines Programms können nur Experten erkennen.
Chaotische Ausgaben erkennt jeder als schlechten Programmierstil, also auch der Chef …

Gute Verpackung erhöht den Preis.

`n` Wiederholungsfaktor, `w` Gesamtzeichenzahl, `d` ggf. Zusatzangabe wie 'davon Anzahl Dezimalziffern' – alles ganzzahlige Konstanten!

- Gruppierung durch Klammerung, z.B. `FORMAT(1X,5(I2,':',F6.2))`

Fortran-Ein-/Ausgabe

Formatsteuerung — Datenformate

Format	für Ausgabe von ...	Bsp.	Wert	Ausgabe
`Iw[.m]`	integer, *w* Zeichen, rechtsbündig	`I5`	-45	`␣␣-45`
	mindestens *m* Ziffern	`I2.2`	8	`08`
`Fw.d`	real, Festkomma, *d* Kommastellen	`F5.2`	1.2345	`␣␣1.23`
`Ew.d[Ee]`	real, Gleitkomma,	`E9.2`	-1.0	`-0.10E+01`
	d Mantissenziffern,	`E9.2`	0.01	`␣0.10E-01`
	e Ziffern für Exponent	`E9.2E3`	$1.1 \cdot 10^{99}$	`0.11E+100`
`Dw.d`	wie E-Format (ohne e)	`D9.2`	0.01	`␣0.10D-01`
		`D9.2`	$1.1 \cdot 10^{99}$	`␣0.11+100`
`Lw`	logical	`L3`	$(2 > 1)$	`␣␣T`
`A[w]`	character (mit `w` rechtsbündig!)	`A3`	'Texte'	`Tex`
`Gw.d[Ee]`	generisches Format, angepasst	`G9.2`	0.02	`␣0.20E-01`
	an Typ und Wert der Variablen	`G9.2`	2.0	`␣␣2.0␣␣␣␣`
`Zw[.m]`	integer,real,logical $\rightarrow$ hexadezimal	`Z9.8`	20	`␣00000014`
	mindestens *m* Hexadezimalziffern	`Z9.8`	1.5	`␣3FC00000`
		`Z9.8`	$(2 > 1)$	`␣00000001`
`Ow[.m]`	dasselbe in Oktalziffern	`O12.11`	20	`␣00000000024`
`Bw[.m]`	dasselbe in Binärziffern	`B32.32`	20	`0...010100`

Fortran-Ein-/Ausgabe

Formatsteuerung — Datenformate

Format	für			Ausgabe
`Iw[.m]`	int			␣␣-45
	mi			08
`Fw.d`	rea			␣␣1.23
`Ew.d[Ee]`				-0.10E+01
	d			␣0.10E-01
	e			0.11E+100
`Dw.d`	wie			␣0.10D-01
				␣0.11+100
`Lw`	log			␣␣T
`A[w]`	cha			Tex
`Gw.d[Ee]`	ger			␣0.20E-01
	an Typ und Wert der Variablen	`G9.2`	2.0	␣␣2.0␣␣␣␣
`Zw[.m]`	integer,real,logical → hexadezimal	`Z9.8`	20	␣00000014
	mindestens *m* Hexadezimalziffern	`Z9.8`	1.5	␣3FC00000
		`Z9.8`	(2 > 1)	␣00000001
`Ow[.m]`	dasselbe in Oktalziffern	`O12.11`	20	␣00000000024
`Bw[.m]`	dasselbe in Binärziffern	`B32.32`	20	0..010100

Spezialvarianten des E-Formats

Bsp.: $x = 0.2345 \cdot 10^n, \quad n = -2, \ldots, 4$

n		`E9.1`	`EN9.1`	`ES9.1`
−2	→	␣␣0.2E-02	␣␣2.3E-03	␣␣2.3E-03
−1	→	␣␣0.2E-01	␣23.4E-03	␣␣2.3E-02
0	→	␣␣0.2E+00	234.5E-03	␣␣2.3E-01
1	→	␣␣0.2E+01	␣␣2.3E+00	␣␣2.3E+00
2	→	␣␣0.2E+02	␣23.4E+00	␣␣2.3E+01
3	→	␣␣0.2E+03	234.5E+00	␣␣2.3E+02
4	→	␣␣0.2E+04	␣␣2.3E+03	␣␣2.3E+03
			techn.	scient.

Fortran-Ein-/Ausgabe

Formatsteuerung — Datenformate

Format	für			Ausgabe
`Iw[.m]`	int			␣␣-45
	mi			08
`Fw.d`	rea			␣␣1.23
`Ew.d[Ee]`				-0.10E+01
	d			␣0.10E-01
	e			0.11E+100
`Dw.d`	wie			␣0.10D-01
				␣0.11+100
`Lw`	log			␣␣T
`A[w]`	cha			Tex
`Gw.d[Ee]`	ger			␣0.20E-01
	an Typ und Wert der Variablen	G9.2	2.0	␣␣2.0␣␣␣␣
`Zw[.m]`	integer,real,logical → hexadezimal	Z9.8	20	␣00000014
	mindestens *m* Hexadezimalziffern	Z9.8	1.5	␣3FC00000
		Z9.8	(2 > 1)	␣00000001
`Ow[.m]`	dasselbe in Oktalziffern	O12.11	20	␣00000000024
`Bw[.m]`	dasselbe in Binärziffern	B32.32	20	0..010100

Spezialvarianten des E-Formats

Bsp.: $x = -0.2345 \cdot 10^n$, $n = -2, \ldots, 4$

n		E9.1	EN9.1	ES9.1
−2	→	␣-0.2E-02	␣-2.3E-03	␣-2.3E-03
−1	→	␣-0.2E-01	-23.4E-03	␣-2.3E-02
0	→	␣␣0.2E+00	*********	␣-2.3E-01
1	→	␣-0.2E+01	␣-2.3E+00	␣-2.3E+00
2	→	␣-0.2E+02	-23.4E+00	␣-2.3E+01
3	→	␣-0.2E+03	*********	␣-2.3E+02
4	→	␣-0.2E+04	␣-2.3E+03	␣-2.3E+03
			techn.	scient.

Fortran-Ein-/Ausgabe

Formatsteuerung — Datenformate

Format	für Ausgabe von ...	Bsp.	Wert	Ausgabe
`Iw[.m]`	integer, *w* Zeichen, rechtsbündig	`I5`	-45	`␣␣-45`
	mindestens *m* Ziffern	`I2.2`	8	`08`
`Fw.d`	real, Festkomma, *d* Kommastellen	`F5.2`	1.2345	`␣␣1.23`
`Ew.d[Ee]`	real, Gleitkomma,	`E9.2`	-1.0	`-0.10E+01`
	d Mantissenziffern,	`E9.2`	0.01	`␣0.10E-01`
	e Ziffern für Exponent	`E9.2E3`	$1.1 \cdot 10^{99}$	`0.11E+100`
`Dw.d`	wie E-Format (ohne e)	`D9.2`	0.01	`␣0.10D-01`
		`D9.2`	$1.1 \cdot 10^{99}$	`␣0.11+100`
`Lw`	logical	`L3`	$(2 > 1)$	`␣␣T`
`A[w]`	character (mit `w` rechtsbündig!)	`A3`	'Texte'	`Tex`
`Gw.d[Ee]`	generisches Format, angepasst	`G9.2`	0.02	`␣0.20E-01`
	an Typ und Wert der Variablen	`G9.2`	2.0	`␣␣2.0␣␣␣␣`
`Zw[.m]`	integer,real,logical $\rightarrow$ hexadezimal	`Z9.8`	20	`␣00000014`
	mindestens *m* Hexadezimalziffern	`Z9.8`	1.5	`␣3FC00000`
		`Z9.8`	$(2 > 1)$	`␣00000001`
`Ow[.m]`	dasselbe in Oktalziffern	`O12.11`	20	`␣00000000024`
`Bw[.m]`	dasselbe in Binärziffern	`B32.32`	20	`0...010100`

Fortran-Ein-/Ausgabe

Formatsteuerung — Datenformate

Format	für Ausgabe von ...	Bsp.	Wert	Ausgabe
`Iw[.m]`	integ mind	`I5`	-45	`␣␣-45`
		`I2.2`	8	`08`
`Fw.d`	real,	`F5.2`	1.2345	`␣␣1.23`
`Ew.d[Ee]`	real, *d* Ma *e* Zif	`E9.2`	-1.0	`-0.10E+01`
		`E9.2`	0.01	`␣0.10E-01`
		`E9.2E3`	$1.1 \cdot 10^{99}$	`0.11E+100`
`Dw.d`	wie E	`D9.2`	0.01	`␣0.10D-01`
		`D9.2`	$1.1 \cdot 10^{99}$	`␣0.11+100`
`Lw`	logic	`L3`	$(2 > 1)$	`␣␣T`
`A[w]`	chara	`A3`	'Texte'	`Tex`
`Gw.d[Ee]`	generisches Format, angepasst	`G9.2`	0.02	`␣0.20E-01`
	an Typ und Wert der Variablen	`G9.2`	2.0	`␣␣2.0␣␣␣`
`Zw[.m]`	integer,real,logical → hexadezimal	`Z9.8`	20	`␣00000014`
	mindestens *m* Hexadezimalziffern	`Z9.8`	1.5	`␣3FC00000`
		`Z9.8`	$(2 > 1)$	`␣00000001`
`Ow[.m]`	dasselbe in Oktalziffern	`O12.11`	20	`␣00000000024`
`Bw[.m]`	dasselbe in Binärziffern	`B32.32`	20	`0..010100`

Beispiele zum Format `G9.2`

$x =$		Ausgabe
0.002	→	`␣0.20E-02`
0.02	→	`␣0.20E-01`
0.2	→	`␣0.20␣␣␣`
2.0	→	`␣␣2.0␣␣␣`
20.0	→	`␣␣20.␣␣␣`
200.0	→	`␣0.20E+03`
2000.0	→	`␣0.20E+04`

Fortran-Ein-/Ausgabe

Formatsteuerung — Datenformate

Format	für Ausgabe von ...	Bsp.	Wert	Ausgabe
`Iw[.m]`	integ mind	`I5`	-45	␣␣-45
		`I2.2`	8	08
`Fw.d`	real,	`F5.2`	1.2345	␣␣1.23
`Ew.d[Ee]`	real, *d* Ma *e* Zif	`E9.2`	-1.0	-0.10E+01
		`E9.2`	0.01	␣0.10E-01
		`E9.2E3`	$1.1 \cdot 10^{99}$	0.11E+100
`Dw.d`	wie E	`D9.2`	0.01	␣0.10D-01
		`D9.2`	$1.1 \cdot 10^{99}$	␣0.11+100
`Lw`	logic	`L3`	$(2 > 1)$	␣␣T
`A[w]`	chara	`A3`	'Texte'	Tex
`Gw.d[Ee]`	generisches Format, angepasst	`G9.2`	0.02	␣0.20E-01
		`2`	2.0	␣␣2.0␣␣␣
`Zw[.m]`		`8`	20	␣00000014
		`8`	1.5	␣3FC00000
		`8`	$(2 > 1)$	␣00000001
`Ow[.m]`		`.11`	20	␣00000000024
`Bw[.m]`		`.32`	20	0..010100

Beispiele zum Format `G9.2`

$x =$		Ausgabe
0.002	→	␣0.20E-02
0.02	→	␣0.20E-01
0.2	→	␣0.20␣␣␣
2.0	→	␣␣2.0␣␣␣
20.0	→	␣␣20.␣␣␣
200.0	→	␣0.20E+03
2000.0	→	␣0.20E+04

Format `Gw.d` wird interpretiert als
`Ew.d` für $x < 0.1$ bzw. $x \geq 10^d$,
`F(w-4).(d-k),4X` für $10^{k-1} \leq x < 10^k$, $(k = 0, 1, \ldots, d)$,
bzw. als `Iw`, `Lw`, `Aw` für entsprechende Datentypen

Fortran-Ein-/Ausgabe

Formatsteuerung — Datenformate

Format	für Ausgabe von ...	Bsp.	Wert	Ausgabe
`Iw[.m]`	integer, *w* Zeichen, rechtsbündig	`I5`	-45	`␣␣-45`
	mindestens *m* Ziffern	`I2.2`	8	`08`
`Fw.d`	real, Festkomma, *d* Kommastellen	`F5.2`	1.2345	`␣␣1.23`
`Ew.d[Ee]`	real, Gleitkomma,	`E9.2`	-1.0	`-0.10E+01`
	d Mantissenziffern,	`E9.2`	0.01	`␣0.10E-01`
	e Ziffern für Exponent	`E9.2E3`	$1.1 \cdot 10^{99}$	`0.11E+100`
`Dw.d`	wie E-Format (ohne e)	`D9.2`	0.01	`␣0.10D-01`
		`D9.2`	$1.1 \cdot 10^{99}$	`␣0.11+100`
`Lw`	logical	`L3`	$(2 > 1)$	`␣␣T`
`A[w]`	character (mit `w` rechtsbündig!)	`A3`	'Texte'	`Tex`
`Gw.d[Ee]`	generisches Format, angepasst	`G9.2`	0.02	`␣0.20E-01`
	an Typ und Wert der Variablen	`G9.2`	2.0	`␣␣2.0␣␣␣␣`
`Zw[.m]`	integer,real,logical $\rightarrow$ hexadezimal	`Z9.8`	20	`␣00000014`
	mindestens *m* Hexadezimalziffern	`Z9.8`	1.5	`␣3FC00000`
		`Z9.8`	$(2 > 1)$	`␣00000001`
`Ow[.m]`	dasselbe in Oktalziffern	`O12.11`	20	`␣00000000024`
`Bw[.m]`	dasselbe in Binärziffern	`B32.32`	20	`0...010100`

Fortran-Ein-/Ausgabe

Formatsteuerung — Steuerungselemente

Format	Wirkung
`'...'`	angegebene Zeichenkette ausgeben
`nH`$\underbrace{\texttt{.....}}_{n\text{ Zeichen}}$	Zeichenkette (Hollerith-Konstante) ausgeben
`Tn`	Sprung in der Zeile zur Spaltenposition n (absolut)
`TRn`	Sprung in der Zeile um n Spalten nach rechts (relativ)
`TLn`	Sprung in der Zeile um n Spalten nach links (relativ)
`nX`	n Leerzeichen ausgeben (bzw. n Zeichen überspringen bei Eingabe)
`/`	neuen Datensatz (neue Zeile) beginnen (auch: `...,/,...` → `.../...`)
`nP`	Skalierungsfaktor 10^n für alle nachfolgenden `E`- und `F`-Formate (z. B. mit `1P E9.2` statt ␣`-.10E+01` Ausgabe `-1.00E+00`) bei `E`-Format wird Exponent angepasst, bei `F`-Format Zahlenwert skaliert
`SP`	erzwingt Ausgabe des Vorzeichens `'+'` für positive Zahlen,
`SS`	positive Zahlen ohne Vorzeichen (Standard).
`BN`	Blanks (␣) bei Eingabe ignorieren,
`BZ`	␣ als `0` interpretieren.
`:`	(vorzeitiges) Ende, falls `ioliste` abgearbeitet, sonst noch Steuerelemente
`$`	am Ende der Formatliste, keine Ausgabe von Newline

Fortran-Ein-/Ausgabe

Formatsteuerung — Steuerungselemente

Format	Wirkung
`'...'`	angegebene Zeichenkette ausgeben
`nH.....` (*n* Zeichen)	Zeichenkette (Hollerith-Konstante) ausgeben
`Tn`	Sprung in der Zeile zur Spaltenposition *n* (absolut)
`TRn`	s (relativ)
`TLn`	(relativ)
`nX`	erspringen bei Eingabe)
`/`	n: `...,/,...` → `.../...`)
`nP`	E- und F-Formate
	abe `-1.00E+00`)
	-Format Zahlenwert skaliert
`SP`	ositive Zahlen,
`SS`	
`BN`	
`BZ`	
`:`	(vorzeitiges) Ende, falls `ioliste` abgearbeitet, sonst noch Steuerelemente
`$`	am Ende der Formatliste, keine Ausgabe von Newline

Beispiel zur Anwendung des `:`-Formats

```
      integer h,m,s
      ...
      write(*,10) h,m,s
      write(*,10) h,m
10    Format(1X,I2.2,':',I2.2,':',I2.2)
```

Ausgabe ohne `:`-Format

```
  16:01:05
  16:01:        ←
```

Fortran-Ein-/Ausgabe

Formatsteuerung — Steuerungselemente

Format	Wirkung
`'...'`	angegebene Zeichenkette ausgeben
`nH.....` (*n* Zeichen)	Zeichenkette (Hollerith-Konstante) ausgeben
`Tn`	Sprung in der Zeile zur Spaltenposition *n* (absolut)
`TRn`	s (relativ)
`TLn`	(relativ)
`nX`	erspringen bei Eingabe)
`/`	n: `...,/,...` → `.../...`)
`nP`	E- und F-Formate
	abe `-1.00E+00`)
	-Format Zahlenwert skaliert
`SP`	ositive Zahlen,
`SS`	
`BN`	
`BZ`	
`:`	(vorzeitiges) Ende, falls `ioliste` abgearbeitet, sonst noch Steuerelemente
`$`	am Ende der Formatliste, keine Ausgabe von Newline

Beispiel zur Anwendung des `:`-Formats

```
      integer h,m,s
      ...
      write(*,10) h,m,s
      write(*,10) h,m
10    Format(1X,I2.2,:,':',I2.2,:,':',I2.2)
```

Ausgabe mit `:`-Format

```
  16:01:05
  16:01
```

←

Regeln zur Abarbeitung der Formatliste

- Jedem Wert aus `ioliste` wird ein Datenformat aus `fliste` zugeordnet.
- Ende der `ioliste`:
  alle verbleibenden Steuerelemente aus `fliste` bis zum nächsten Datenformatelement werden noch abgearbeitet
  (ein `:` als Steuerelement stoppt diesen Prozess)
- Ende der `fliste`:
  Newline und Wiederholung ab letzter Formatgruppe von `fliste`.
  Formatgruppe = geklammerte Liste von Formatelementen
- Beispiel:

```
      WRITE(*,20) I, (A(I,J),J=1,N)
20    FORMAT(' Zeile',I3 / (1X, 8E10.2) )
```

Formatgruppe

Regeln zur Abarbeitung der Formatliste

- Jedem Wert aus `ioliste` wird ein Datenformat aus `fliste` zugeordnet.
- Ende der `ioliste`:
  alle verbleibenden Steuer[illegible] bis zum nächsten
  Datenformatelement wer[illegible]
  (ein `:` als Steuerelement[illegible]
- Ende der `fliste`:
  Newline und Wiederholu[illegible]
  Formatgruppe = geklam[illegible]
- Beispiel:

```
      WRITE(*,20) I, (A(I,J),J=1,N)
20    FORMAT(' Zeile',I3 / (1X, 8E10.2) )
```

Formatgruppe

Ausgabe z.B. für `I=1, N=20`:

␣Zeile␣␣1
␣ $a_{1,1}$ ... ... $a_{1,8}$
␣ $a_{1,9}$ ... ... $a_{1,16}$
␣ $a_{1,17}$ ... $a_{1,20}$

Variable Formatstrings / Interne Files

- Angaben in den Formatelementen müssen ganzzahlige Konstante sein. Aber als Formatspezifikation darf auch eine Zeichenkettenvariable `fliste` angegeben werden, deren Inhalt kann zur Laufzeit angepasst werden:

```
character*10 fliste /'(1X,3I4)'/
fliste(5:5)=CHAR(ICHAR('0')+n) !  ersetze 3 durch Wert von n
write(*,fliste) (M(i),i=1,n)
```

  (so funktioniert das nur mit $1 \leq n \leq 9$).

- Auch anstelle von `[UNIT=]nr` kann bei `READ` oder `WRITE` eine Zeichenkettenvariable stehen, die anstelle einer Datei den mit `WRITE` geschriebenen oder mit `READ` zu lesenden Text enthält. (internal file)

- `READ (str,fmt) ...`
  `WRITE(str,fmt) ...`
  Ist `str` ein Character-Array, dann entspricht jedes Feldelement einem Datensatz (bzw. einer Zeile).

Fortran-Ein-/Ausgabe

Beispiel für „interne Files“

```
      CHARACTER*80 zeile
      CHARACTER*10 fmt
      READ (*,'(A80)') zeile
      k=INDEX(zeile,'N=')
      IF (k .GT. 0) THEN
        k=k+2
        m=INDEX(zeile(k:),',')
        WRITE(fmt,10) m-1
10      FORMAT('(I',I2,')')
        READ(zeile(k:k+m-2),fmt) n
      ELSE
        n=0
      ENDIF
```

Eingabezeile:

```
 Dim. ist N=512, weitere Angaben
```

Fortran-Ein-/Ausgabe

Beispiel für „interne Files“

```
      CHARACTER*80 zeile
      CHARACTER*10 fmt
      READ (*,'(A80)') zeile
      k=INDEX(zeile,'N=')
      IF (k .GT. 0) THEN
        k=k+2
        m=INDEX(zeile(k:),',')
        WRITE(fmt,10) m-1
10      FORMAT('(I',I2,')')
        READ(zeile(k:k+m-2),fmt) n
      ELSE
        n=0
      ENDIF
```

Eingabezeile:

␣Dim.␣ist␣N=512,␣weitere␣Angaben

$k = 11$

Fortran-Ein-/Ausgabe

Beispiel für „interne Files“

```
      CHARACTER*80 zeile
      CHARACTER*10 fmt
      READ (*,'(A80)') zeile
      k=INDEX(zeile,'N=')
      IF (k .GT. 0) THEN
        k=k+2
        m=INDEX(zeile(k:),',')
        WRITE(fmt,10) m-1
10      FORMAT('(I',I2,')')
        READ(zeile(k:k+m-2),fmt) n
      ELSE
        n=0
      ENDIF
```

Eingabezeile: ↓

␣Dim.␣ist␣N=512,␣weitere␣Angaben

$k = 11$

$k = 13$

Fortran-Ein-/Ausgabe

Beispiel für „interne Files“

```
      CHARACTER*80 zeile
      CHARACTER*10 fmt
      READ (*,'(A80)') zeile
      k=INDEX(zeile,'N=')
      IF (k .GT. 0) THEN
        k=k+2
        m=INDEX(zeile(k:),',')
        WRITE(fmt,10) m-1
10      FORMAT('(I',I2,')')
        READ(zeile(k:k+m-2),fmt) n
      ELSE
        n=0
      ENDIF
```

Eingabezeile: ↓

␣Dim.␣ist␣N=512,␣weitere␣Angaben

$k = 11$

$k = 13$

$m = 4$

Fortran-Ein-/Ausgabe

Beispiel für „interne Files“

```
      CHARACTER*80 zeile
      CHARACTER*10 fmt
      READ (*,'(A80)') zeile
      k=INDEX(zeile,'N=')
      IF (k .GT. 0) THEN
        k=k+2
        m=INDEX(zeile(k:),',')
        WRITE(fmt,10) m-1
10      FORMAT('(I',I2,')')
        READ(zeile(k:k+m-2),fmt) n
      ELSE
        n=0
      ENDIF
```

Eingabezeile: ↓

␣Dim.␣ist␣N=512,␣weitere␣Angaben

$k = 11$

$k = 13$

$m = 4$

fmt ← (I 3)

Fortran-Ein-/Ausgabe

Beispiel für „interne Files“

```
      CHARACTER*80 zeile
      CHARACTER*10 fmt                    Eingabezeile: ↓
      READ (*,'(A80)') zeile              ␣Dim.␣ist␣N=512,␣weitere␣Angaben
      k=INDEX(zeile,'N=')                 k = 11
      IF (k .GT. 0) THEN
        k=k+2                             k = 13
        m=INDEX(zeile(k:),',')            m = 4
        WRITE(fmt,10) m-1                 fmt ← (I 3)
10      FORMAT('(I',I2,')')
        READ(zeile(k:k+m-2),fmt) n        zeile(13:15) → n = 512
      ELSE
        n=0
      ENDIF
```

Namelist-gesteuerte E/A

- vermutlich selten genutzt; sinnvoll, wenn Daten aus Dateien eingelesen werden (nicht interaktiv)
- Beliebige Variable werden zu einer Namengruppe zusammengefasst (`NAMELIST`-Anweisung)

```
REAL F(6)
NAMELIST /daten/ n1,n2,m,F
```

- E/A erfolgt ohne `ioliste` und ohne Formatspezifikation (es gelten typspezifische Standards)
- Ein- und Ausgabe erfolgt in der Form `variablenname = wert` für alle Variablen einer Namengruppe (bei Eingabe müssen nicht alle Variablen angegeben sein).
- Typische Anwendung: Standardvorgaben des Programms anzeigen und optional ausgewählte Werte ändern.

```
DATA n1 /3/, n2/6/, m/1/, F/6*0.0/
WRITE(*,[NML=]daten)
READ (*,[NML=]daten)
```

Namelist-gesteuerte E/A

- vermutlich selten genutzt; sinnvoll, wenn Daten aus Dateien eingelesen werden (nicht interaktiv)
- Beliebige Variable werden zu einer Namengruppe zusammengefasst (`NAMELIST`-Anweisung)

```
REAL F(6)
NAMELIST /daten/ n1,n2,m,F
```

- E/A erfolgt ohne `ioliste` und ohne Formatspezifikation (es gelten typspezifische Star
- Ein- `ame = wert` für alle Variablen eine Variablen angegeben sein).
- Typi mms anzeigen und optional ausg

Namelist-Ausgabe

```
&DATEN
 N1=      3,
 N2=      6,
 M=       1,
 F= 6*0.000000  ,
 /
```

```
DATA n1 /3/, n2/6/, m/1/, F/6*0.0/
WRITE(*,[NML=]daten)
READ (*,[NML=]daten)
```

Namelist-gesteuerte E/A

- vermutlich selten genutzt; sinnvoll, wenn Daten aus Dateien eingelesen werden (nicht interaktiv)
- Beliebige Variable werden zu einer Namengruppe zusammengefasst (`NAMELIST`-Anweisung)

```
REAL F(6)
NAMELIST /daten/ n1,n2,m,F
```

- E/A erfolgt ohne `ioliste` und ohne Formatspezifikation (es gelten typspezifische Star
- Ein- `ame = wert` für alle Variablen eine Variablen angegeben sein).
- Typi mms anzeigen und optional ausg

```
DATA n1 /3/, n2/6/, m/1
WRITE(*,[NML=]daten)
READ (*,[NML=]daten)
```

Namelist-Ausgabe

```
&DATEN
 N1=      3,
 N2=      6,
 M=       1,
 F= 6*0.000000  ,
 /
```

Namelist-Eingabe

```
&daten m=0, n2=-1, F(4:6)=3.,2.,1. /
```

Namelist-gesteuerte E/A

- vermutlich selten genutzt; sinnvoll, wenn Daten aus Dateien eingelesen werden (nicht interaktiv)
- Beliebige Variable werden zu einer Namengruppe zusammengefasst (`NAMELIST`-Anweisung)

```
REAL F(6)
NAMELIST /daten/ n1,n2,m,F
```

- E/A erfolgt ohne `ioliste` und ohne Formatspezifikation (es gelten typspezifische Star
- Ein- `ame = wert` für alle Variablen eine Variablen angegeben sein).
- Typi mms anzeigen und optional ausg

```
DATA n1 /3/, n2/6/, m/1
WRITE(*,[NML=]daten)
READ (*,[NML=]daten)
```

Namelist-Ausgabe (nach Eingabe)

```
&DATEN
 N1=      3,
 N2=     -1,
 M=       0,
 F= 3*0.000000   , 3.000000   , 2.000000   , 1.000000   ,
 /
```

Namelist-Eingabe

```
&daten m=0, n2=-1, F(4:6)=3.,2.,1. /
```

Fortran-Ein-/Ausgabe

Zuordnung zwischen Dateien und UNIT=nr

- Jeder in READ oder WRITE benutzten logischen Dateinummer (UNIT) muss eine (geöffnete) Datei zugewiesen sein; stdin, stdout und stderr sind bei Programmstart bereits geöffnet und den Dateinummern 5, 6 bzw. 0 zugeordnet.
- Mittels Fileumlenkung können stdin und/oder stdout auch mit beliebigen (Text-)Dateien verknüpft werden (anstelle des interaktiven Terminals):

  ```
  % ./myexec < eingabe.txt > ausgabe.txt    oder
  % cat eingabe.txt | ./myexec | tee ausgabe.txt
  ```

- Bei Dateinummern, die (noch) keiner Datei zugeordnet sind, wird beim ersten Zugriff (READ oder WRITE) ein vom Compiler generierten Dateiname zugewiesen (z. B. FTN.01 oder fort.8 je nach System und Compiler), diese Datei wird ggf. durch WRITE neu angelegt oder meldet bei READ einen EOF-Fehler.
- Die explizite Zuordnung eines Dateinamens zu einer Dateinummer (Öffnung einer Datei) erfolgt mittels OPEN-Anweisung, jede so geöffnete Datei sollte vor dem Ende des Programms auch explizit wieder geschlossen werden (CLOSE-Anweisung).
- Alle noch offenen Dateien werden bei Programm-Ende automatisch geschlossen (so gut es eben geht).

Fortran-Ein-/Ausgabe

OPEN ([UNIT=]nr, keyword=value, ...)

keyword	value	Bemerkungen
FILE=	filename	Character-Ausdruck, Vorsicht mit Leerzeichen!
		fehlt FILE, so wird eine temporäre Datei angelegt
ACCESS=	zugriffsart	Character 'SEQUENTIAL'/'DIRECT'
RECL=	byteprosatz	notwendig für DIRECT, sonst max. Länge der Datensätze
FORM=	format	Char. 'FORMATTED'/'UNFORMATTED'/'BINARY'
		Standard: seq.→FORMATTED, dir.→UNFORMATTED
STATUS=	'UNKNOWN'	Standard (OLD oder NEW automatisch erkannt)
	'OLD'	Datei muss existieren, wird nicht überschrieben
	'REPLACE'	Datei wird überschrieben, falls vorhanden
	'NEW'	Datei darf nicht existieren, wird neu angelegt
	'SCRATCH'	temporäre Datei (nur solange das Programm läuft)
ACTION=	zugriff	'READ','WRITE','READWRITE' (vorgesehene Verwendung)
	sharing	'DENYBOTH','DENYREAD','DENYWRITE','DENYNONE'
		beides kombiniert z.B. ACTION='READWRITE,DENYWRITE'
u.v.a.m.	(wenige Details können versionsabhängig abweichen)	

Fortran-Ein-/Ausgabe

CLOSE ([UNIT=]nr, keyword=value, ...)

keyword	value	Bemerkungen
STATUS=	'KEEP'	nach dem Schließen ... bleibt Datei auf Platte erhalten
	'DELETE'	... wird Datei gelöscht (immer für 'SCRATCH')
ERR=	marke	Sprungziel bei Fehler
IOSTAT=	istat	Integer-Variable istat erhält Statusinformation zurück

Bem.: ERR=marke und IOSTAT=istat gibt es bei allen E/A-Anweisungen, insbes. auch bei OPEN (z.B. 'OLD'-Datei nicht vorhanden oder 'NEW'-Datei existiert bereits oder Dateinummer wurde bereits für eine andere, noch geöffnete Datei verwendet)

Um solche Fehlermeldungen zu vermeiden, nutzt man die INQUIRE-Anweisung, um rechtzeitig Informationen über Dateien und logische Dateinummern abzufragen.

Fortran-Ein-/Ausgabe

`INQUIRE (` { `[UNIT=]nr` | `FILE=name` } `, keyword=var, ...)`

Abfrage von Informationen über die der log. Dateinummer `nr` zugeordnete Datei **oder** über die Datei namens `name` (Dateiname als Zeichenkettenausdruck)

keyword	var-Typ	Bemerkungen
`NAME=`	character	Dateiname bei `inquire-by-unit`
`NUM=`	integer	Unit-Nummer bei `inquire-by-file`
`EXIST=`	logical	Existenz der Datei bzw. Unit
`OPENED=`	logical	Ist Datei bzw. Unit bereits geöffnet?
`FORM=`	character	`'FORMATTED' / 'UNFORMATTED' / 'UNDEFINED'`
`ACCESS=`	character	`'SEQUENTIAL' / 'DIRECT' / 'UNDEFINED'`
`FORMATTED=`	character	zulässig für Datei? $\rightarrow$ `'YES' / 'NO' / 'UNKNOWN'`
analoge Abfragen für `UNFORMATTED= / SEQUENTIAL= / DIRECT=`		
`NAMED=`	logical	`.FALSE.` bei `SCRATCH` oder ungeöffneter Datei
u.v.a.m.		

Beispiel für INQUIRE: Dateinummer ermitteln

```
INTEGER FUNCTION newLUN()
LOGICAL exist, open
INTEGER lun, nr
lun=-1
DO nr=8,99
  INQUIRE(UNIT=nr,EXIST=exist,OPENED=open)
  IF (exist .AND. .NOT. open) THEN
    lun=nr
    EXIT
  ENDIF
ENDDO
newLUN=lun
RETURN
END FUNCTION newLUN
```

Beispiel für INQUIRE: Dateinummer ermitteln

```
INTEGER FUNCTION newLUN()
LOGICAL exist, open
INTEGER lun, nr
lun=-1
DO nr=8,99
  INQUIRE(UNIT=nr,EXIST=exist,OPENED=open)
  IF (exist .AND. .NOT. open) THEN
    lun=nr
    EXIT
  ENDIF
ENDDO
newLUN=lun
RETURN
END FUNCTION newLUN
```

Aufruf, nicht verwendete Dateinummer bestimmen:

```
...
nr=newLUN()
IF (nr .GT. 0) THEN
  OPEN(nr,FILE=dateiname,STATUS='UNKNOWN')
ELSE
  WRITE(0,*) 'Alles besetzt!'
ENDIF
```

Fortran-Ein-/Ausgabe

Beispiel für INQUIRE: Dateinamen prüfen

```
      ...
1     READ (*,'(A)') dateiname
      L=Len_Trim(dateiname)
      INQUIRE(FILE=dateiname(1:L),EXIST=exist)
      IF (exist) THEN
        WRITE(*,10)
10      FORMAT(' Datei existiert schon. Ueberschreiben? ',$)
        READ(*,'(A1)') antwort
        IF ( INDEX('yYjJ',antwort) .GT. 0 ) THEN
          stat='REPLACE'
        ELSE
          GOTO 1
        ENDIF
      ELSE
        stat='NEW'
      ENDIF
      nr=newLUN()
      OPEN(nr,dateiname(1:L),STATUS=stat,ERR=999)
      WRITE(nr) datenfeld
```