

IO und Formatangaben

Sonderübung

Dana Liebscher

PR10

16.01.2024

1. Input und Output
 - Öffnen von Dateien
 - Schließen von Dateien
 - Besonderheiten von Scratch-Dateien
 - Lesen und Schreiben in Dateien
2. Attribute zum Einlesen/Schreiben
 - Attribute beim Einlesen
 - Attribute beim Schreiben
3. Übungsaufgabe
4. Formatangaben

Aufgabe und Nutzen von IO

Aufgaben:

- Verwaltung von Dateien
- Datentransfer zwischen Dateien und internem (Haupt-)Speicher

Nutzen:

- Speichern von Ergebnissen über den Programmablauf hinaus
- vereinfacht das Einlesen großer Datenmengen (d.h. man muss nicht alles selbst tippen)

Öffnen von Dateien

```
OPEN([UNIT=]u [,IOSTAT=iostat] [,ERR=err][,FILE=file]  
[,STATUS=status][,ACTION=action])
```

- wähle immer eine Unitnummer **> 30**, um keine bereits intern vergebenen Nummern zu nutzen
- Bei File sollte entweder eine Charaktervariable stehen oder Dateiname in ' '
- STATUS (OLD/NEW/REPLACE/SCRATCH/ **UNKNOWN**) und ACTION (READ/WRITE/**READWRITE**) dienen dazu, euch vor euch selbst zu schützen \Rightarrow ungewolltes Überspeichern/ Verändern von Dateien verhindern
- wenn ihr eine IOSTAT-Variable habt, dann fragt sie auch ab!

Beispiel

```
INTEGER:: iopen  
DO  
    OPEN (UNIT=31,IOSTAT=iopen, FILE='Beispiel.dat', &  
          & STATUS='new', ACTION='write')  
    IF (iopen /= 0) EXIT  
END DO
```

Schließen von Dateien

```
CLOSE([UNIT=]u [,IOSTAT=iostat] [,ERR=err][,FILE=file][,STATUS=status])
```

- Meistens verwenden wir bei CLOSE nur die Unitnummer und keine Attribute

Beispiel

```
CLOSE(UNIT=31)
```

```
CLOSE(31)
```

Besonderheiten von Scratch-Dateien

Scratch-Dateien sind (unbenannte) temporäre Dateien, die bei der OPEN-Anweisung erzeugt werden und bei der Ausführung von CLOSE/ beim Beenden des Programms gelöscht werden.

- Nützlich z. B. zum Speichern von Nebenrechnungen/Zwischenergebnissen
- Beim Öffnen von Scratch-Dateien darf kein Name (FILE-Attribut) vergeben werden
- Beim Schließen darf nicht STATUS='keep' verwendet werden

Beispiel

```
INTEGER:: iopen
```

```
OPEN(UNIT=32,IOSTAT=iopen, STATUS='scratch')
```

Lesen und Schreiben in Dateien

Bei READ(*,*) und WRITE(*,*) steht der Eintrag vor dem Komma für **'Woher?'** und der danach für **'Was?'**.

Ausführlich: READ(**UNIT=*,FMT=***) bzw. WRITE(**UNIT=*,FMT=***)

* steht für Standard, d.h. beim Lesen die Tastatur, beim Schreiben der Bildschirm

Um in eine Datei zu schreiben/ aus einer Datei zu lesen, ersetzt ihr das * bei Unit durch die entsprechende Unitnummer.

Wenn nicht anders spezifiziert, geschieht lesen/schreiben zeilenweise!

Beispiel

```
INTEGER:: iopen
```

```
OPEN(UNIT=31,IOSTAT=iopen, FILE='Beispiel.dat', STATUS='new', ACTION='write')
```

```
IF( iopen /= 0) STOP
```

```
WRITE(31,*) 'Hallo'
```

```
WRITE(31,*) 'Welt!'
```

Attribute beim Einlesen

```
READ([UNIT=]u,[FMT=]f [,IOSTAT=iostat] [,ERR=err][,ADVANCE=ad])
```

- IOSTAT/ ERR werden meist dazu genutzt, das Dateiende zu identifizieren (daher: man liest in einer Schleife ein, bis es einen Einlesefehler gibt)
- bei IOSTAT bedeutet: iostat = 0 erfolgreich eingelesen, iostat < 0 Datensatz zu Ende, iostat > 0 anderer Fehler
- mit ADVANCE ('NO'/'YES') lässt sich regeln, ob nach dem Einlesen die Zeile gewechselt werden soll ('YES') oder nicht ('NO')
- **Bei der Verwendung von ADVANCE='NO' müssen Formatangaben verwendet werden!**
- Default-Einstellung ist ADVANCE='YES'

Beispiel

```
INTEGER :: iopen, ioread, i
INTEGER, DIMENSION(20) :: m

OPEN(UNIT=31, IOSTAT=iopen, FILE='Beispiel.dat', &
     & STATUS='old', ACTION='read')
IF (iopen /= 0) THEN
    WRITE(*,*) 'Fehler beim Öffnen der Datei'
ELSE
    DO
        READ(31,*, IOSTAT=ioread) m
        IF (ioread /= 0) EXIT
    END DO
END IF
CLOSE(31)
```

Attribute beim Schreiben

```
WRITE([UNIT=]u,[FMT=]f [,IOSTAT=iostat] [,ERR=err][,ADVANCE=ad])
```

- mit ADVANCE ('NO'/'YES') lässt sich regeln, ob nach dem Schreiben die Zeile gewechselt werden soll ('YES') oder nicht ('NO')
- **Bei der Verwendung von ADVANCE='NO' müssen Formatangaben verwendet werden!**
- Default-Einstellung ist ADVANCE='YES'

Beispiel

```
INTEGER:: iopen,ioread,i

OPEN(UNIT=31,IOSTAT=iopen, FILE='Beispiel.dat', STATUS='new', ACTION='write')
IF( iopen == 0) THEN
    DO i=1, 100
        WRITE (31,*) i
    END DO
END IF
```

Übungsaufgabe

Schreibe ein Programm in welchem die Dateien *io_text_01.txt* und *io_text_02.txt* jeweils Zeilenweise eingelesen und ins Terminal ausgegeben werden.

Außerdem sollen die Zahlen 1 bis 10 zeilenweise in eine (neu angelegte) Datei geschrieben werden.

Formatangaben

Wir wollen unsere Ausgaben hübsch formatieren.

```
WRITE(UNIT=*,FMT=*)
```

Dazu ersetzen wir das zweite * durch einen Format**string** (" nicht vergessen):

```
TYPE(student) :: studi  
REAL :: diameter
```

```
WRITE (UNIT = *, FMT = '(A,X,A,X,I2,X,A)') TRIM(studi%vorname), &  
& 'ist', student%alter, 'Jahre alt.'
```

```
WRITE(*,'(A,X,F6.2,X,A)') 'Der Durchmesser beträgt', diameter, 'cm.'
```

Formatangaben

```
INTEGER :: length2
```

```
REAL :: length1
```

```
CHARACTER(15) :: formatstring
```

```
formatstring = '(A,X,F5.3)'
```

```
WRITE(*,FMT=TRIM(formatstring)) 'Länge in km:', length1
```

```
formatstring(6:9) = 'l5 _ _'
```

```
WRITE(*,FMT=TRIM(formatstring)) 'Länge in m:', length2
```

Formatangaben

Wollt ihr zum Beispiel mehrere Ausgaben in einer Zeile vornehmen, benutzt ihr `ADVANCE = 'NO'`. Dazu sind Formatangaben nötig.

Beispiel

```
TYPE(student), DIMENSION(10) :: studis

DO i = 1, 10
    WRITE(*,'(A, A, I2, A)', ADVANCE = 'NO') studis(i)%vorname, &
        & '(', studis(i)%alter, '), '
END DO

WRITE(*,*)
```

Taskmanager Teil 3

Um nicht alle Daten in unserem Datenbank-ähnlichen Programm (Verwaltung von Wichteln und ihren Aufgaben) selbst über das Terminal ein- und auszugeben, sollen folgende Subroutinen geschrieben werden:

- Eine Subroutine (ähnlich zu GET in Teil 1 und 2), welche eine Datenbank für Wichtel (Vektor des Types *Wichtel*) allokiert und die Daten aus der Datei *wichtel.txt* ausliest.
- Eine Subroutine (ähnlich zu PUT in Teil 1 und 2), welche eine Wichtel-Datenbank kommentiert in eine Datei *wichtel.md* ausgibt. Nutzt dafür Formatangaben.
 - Zusatz: Formatiert die Ausgabe als Tabelle. Eine Markdown-Tabelle hat folgende Form:

```
| Spalte 1 | Spalte 2 |
| ----- | ----- |
| Eintrag  | Eintrag  |
```

Es sollen analoge Subroutinen für Aufgaben geschrieben werden: Auslesen aus der Datei *aufgaben.txt*, ausgeben in die Datei *aufgaben.md*.