

IO und Formatangaben

Sonderübung

Luise Zieger & Lea Happel

PROG Sonderübung

January 15, 2019

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

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!'
```

Übungsaufgabe

In einer Datei steht in der ersten Zeile, wie viele Zeilen noch folgen und wie viele Zeichen (Spalten) diese jeweils enthalten. Danach ist der Text aber senkrecht statt waagrecht geschrieben, Leerzeichen sind mit `_` dargestellt.

Um dies zu lesen, öffne zuerst die Datei und lies die erste Zeile. Allokiere dann eine Matrix mit Einträgen vom Typ `Character` in entsprechender Größe.

Lies nun den Text zeilenweise in eine Matrix ein und schreibe ihn dann spaltenweise auf den Bildschirm und in eine extra Datei.

ZUSATZ: Schreibe eine Funktion, die eine entsprechende Textdatei erstellt. Den Text kannst du zeilenweise eingeben lassen. Beachte jedoch, dass in der Datei zwischen jedem Zeichen ein Leerzeichen stehen muss.

```
4 6
H O R N U E
A _ T ! F H
L F R _ _ T
L O A A G S
```


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,m
```

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

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

```
DO
```

```
    READ(31,*,IOSTAT=ioread) m
```

```
    IF (m /= 0) EXIT
```

```
END DO
```

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) STOP  
DO i=1,100  
    WRITE (31,*) i  
END DO
```

Vergleich von Funktionsergebnissen

f_n bezeichne die n -te Fibonaccizahl. Der Goldene Schnitt ergibt sich als Grenzwert von $\lim_{n \rightarrow \infty} \frac{f_{n+1}}{f_n}$.

Berechne eine Näherung des Goldenen Schnitts, indem du die ersten hundert Quotienten in einer Schleife berechnest.

Speichere dabei jedes Zwischenergebniss, d.h. jeden der hundert Quotienten, in einer Datei ab.

Vergleiche deine Ergebnisse mit denen deines Nachbarn, um ihre Richtigkeit zu überprüfen. Schreibe dazu eine Funktion, die zuerst beide Dateien öffnet und dann in einer Schleife jeweils eine Zahl aus jeder Datei einliest und diese vergleicht. Stimmen die Zahlen immer überein, so soll die Funktion wahr zurückgeben. Stimmen die Zahlen nicht überein oder ist eine Datei länger als die andere, so soll die Funktion falsch zurückgeben.