

Praktische Übung zu DO und IF

Sonderübung

Luise Zieger & Nadine Schärmann

PROG

27. November 2019

1. Wiederholung
2. Debuggen und Testen
3. Aufgaben
4. Vorteile von Funktionen

Wiederholung

CPU_TIME

```
REAL :: start, end  
  
CALL CPU_TIME(start)  
! do something  
CALL CPU_TIME(end)  
  
WRITE(*,*) 'Gesamtzeit: ', end - start
```

MOD(a,b)

Wenn a und b beide positiv sind, gibt dieser Befehl den Rest bei der Division von a durch b zurück. Nützlich ist dies z. B. beim Testen von Teilbarkeiten (z.B.

$\text{MOD}(a,4)=0$ bedeutet „Ist a durch 4 teilbar?“).

Haben a und b unterschiedliche Vorzeichen, arbeiten beide Befehle unterschiedlich. Die gute Nachricht: Oft braucht man nur die Reste positiver Zahlen und es ist daher egal, was ihr verwendet.

Quersumme

Schreibe ein Programm, das die Quersumme einer natürlichen Zahl berechnet. Lies dazu zuerst solange eine Zahl ein, bis eine Zahl ≥ 0 eingegeben wird. Berechne dann mithilfe einer Schleife die Quersumme dieser Zahl und gib diese am Ende aus.

Zusatz: Berechne in der Schleife zusätzlich die umgedrehte Zahl (d.h. die umgedrehte Zahl von 1234 ist 4321) und gib diese ebenfalls am Ende aus.

Zahlenrätsel

100 Scheffel Weizen wurden unter einer gewissen Anzahl von Personen restlos aufgeteilt. Dabei erhielt jeder Mann 3 Scheffel, jede Frau 2 Scheffel und jedes Kind einen halben Scheffel. Berechne alle Möglichkeiten und suche die „realistischsten“ heraus.

Warum kompiliert das schon wieder nicht?!

Gerade am Anfang sind Fehlermeldungen manchmal eher verwirrend als hilfreich. Wie kommt man trotzdem zurecht und was sind „typische“ Fehler?

- Arbeite die Fehlermeldung von **oben nach unten** ab!
- Kontrolliere an der gekennzeichneten Stelle die Syntax der Verzweigungen/Schleifen (typischer Fehler: „then“ bei Verzweigung vergessen)
- Prüfe, ob alle Variablen deklariert sind
- Prüfe, ob du „==“ und „=“ nicht verwechselt hast
- Bei Zählschleifen: Prüfe, dass du die Zählvariable nicht in der Schleife verändert hast
- Google die Fehlermeldung, frage deine Tutoren oder Kommilitonen

Testen

Auch wenn das Programm kompiliert hat, macht es noch nicht immer das, was es soll.

- Teste zuerst mit Werten, bei denen du das Ergebnis kennst/ leicht nachrechnen kannst
- Probiere, alle möglichen Fälle bei den Tests abzudecken

Was tun, wenn das Programm nicht das Richtige tut?

- Prüfe, ob du jeder Variable vor ihrer ersten Benutzung einen Wert zugewiesen hast.
- Lasse dir **Zwischenergebnisse** ausgeben!

Was kann man bei Schleifen tun, um nicht von zu vielen Zwischenausgaben überfordert zu werden?

- Ändere die Schleife fürs Testen in eine Zählschleife.
- Füge eine leere READ-Anweisung ein. Der nächste Schleifendurchgang wird erst dann ausgeführt, wenn du „Enter“ drückst.

Beliebig oft einlesen

Schreibe ein Programm, das beliebig viele einzelne Zeichen einliest (maximal 20). Bei der Eingabe eines Punktes soll das Einlesen gestoppt werden und das gesamte Wort und dessen Länge ausgegeben werden.

Achte besonders darauf, dass dein Programm auch eine korrekte Ausgabe liefert, wenn gar kein Buchstabe eingegeben wurde.

Erinnerung:

```
CHARACTER(len = 12) :: Name, Vorname, Nachname
```

```
Vorname = 'Tom'
```

```
Nachname = "Schulz"
```

```
Name = Vorname // Nachname
```

```
Name = TRIM(Vorname) // ' ' // TRIM(Nachname)
```

```
Name(2:2) = "i"
```

```
! Name = Tom
```

```
! Name = Tom Schulz
```

```
! Name = Tim Schulz
```

Collatz-Vermutung

Betrachte folgende Funktion:

$$f : \mathbb{N} \rightarrow \mathbb{N}$$
$$f(n) = \begin{cases} 3n + 1 & n \text{ ungerade} \\ n/2 & n \text{ gerade} \end{cases}$$

Die Collatz-Vermutung besagt, dass unabhängig von der Zahl mit der man anfängt, irgendwann 1 herauskommt. Zum Beispiel ergibt sich beim Start mit 19 folgende Folge:

19, 58, 29, 88, 44, 22, 11, 34, 17, 52, 26, 13, 40, 20, 10, 5, 16, 8, 4, 2, 1

Schreibe ein Programm, das eine Zahl einliest und für diese berechnet, wie viele Schritte benötigt werden, bis sich 1 ergibt.

Zusatz: Berechne, bei welcher Zahl zwischen 1 und 100 sich die längste Kette ergibt.

Vorteile von Funktionen

- Auslagern von Code \Rightarrow übersichtlicher
- Vermeiden von Copy-und-Paste-Programmierung \Rightarrow besserer Stil
- Ihr könnt jederzeit auf eure Funktion zugreifen
- Komplexere Programme möglich