Programmierkurs Ada 95

WS 03/04
Montags: $17^{25} - 18^{00}$ (18^{15})
im 47.02

Übungen gesondert in Gruppen

- Erlernen einer Programmiersprache
- Illustration der Konzepte
- Programmieren im Kleinen (WS)
  und etwas größer (SS 04)
- Einige Richtlinien
- Konkrete Beispiele

Erwerb des Scheins durch Mitarbeit.

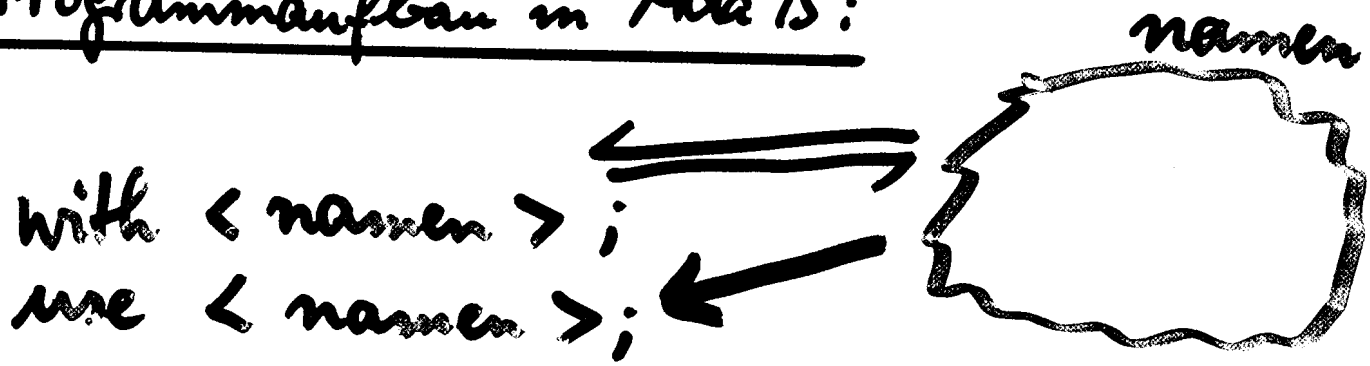
1. Programmaufbau in Ada 95:

Umgebung

```
PROCEDURE <name> IS
<Deklarationsteil>;
BEGIN
   <Anweisungsteil>
END <name>
```

Das Programm wird in eine Umgebung eingebettet, in der alle Sprachelemente "bekannt" sind.

1. Programmaufbau in Ada 95:




```
with <namen>;
use <namen>;

PROCEDURE <name> IS

<Deklarationsteil>;

BEGIN
   <Anweisungsteil>
END <name>
```

Will man anderweitig definierte Strukturen benutzen, so bindet man sie mit "with" ein.
Will man die Namen von Prozeduren usw. so benutzen, ohne auf deren Herkunft hinweisen zu müssen, so fügt man "use ..." hinzu.

Fast immer braucht man eine Standard-Ein- und -Ausgabe für Text:

```
WITH ...TEXT_IO;
USE  ...TEXT_IO;
```

Hierdurch werden die Ein-/Ausgabe-anweisungen GET, PUT, ... bereit gestellt und die Darstellung der elementaren Daten festgelegt.

Im Deklarationsteil werden die Variablen, die Typen, die weiteren Prozeduren und Funktionen vereinbart.

Der Anweisungsteil enthält die Folge der (meist ineinander geschachtelten) auszuführenden Anweisungen.

Prinzipien für Programme

- Trifft man während der Ausführung auf irgendeine Stelle im Programm, so müssen alle dort verwendeten Bezeichner ("Namen") bekannt sein, also zuvor deklariert worden sein.
- Programme besitzen fast immer "Klammerstrukturen"; meist sind sie durch Ineinanderschachteln hierarchisch aufgebaut.
- Programme müssen (von den Erstellern) lesbar und verständlich geschrieben werden und sich an Richtlinien halten. Die Erläuterungen, ergänzende Dokumente, Testdaten, Lösungswege, Ergebnisse usw. gehören stets zu einem Programm dazu.

Beispiel

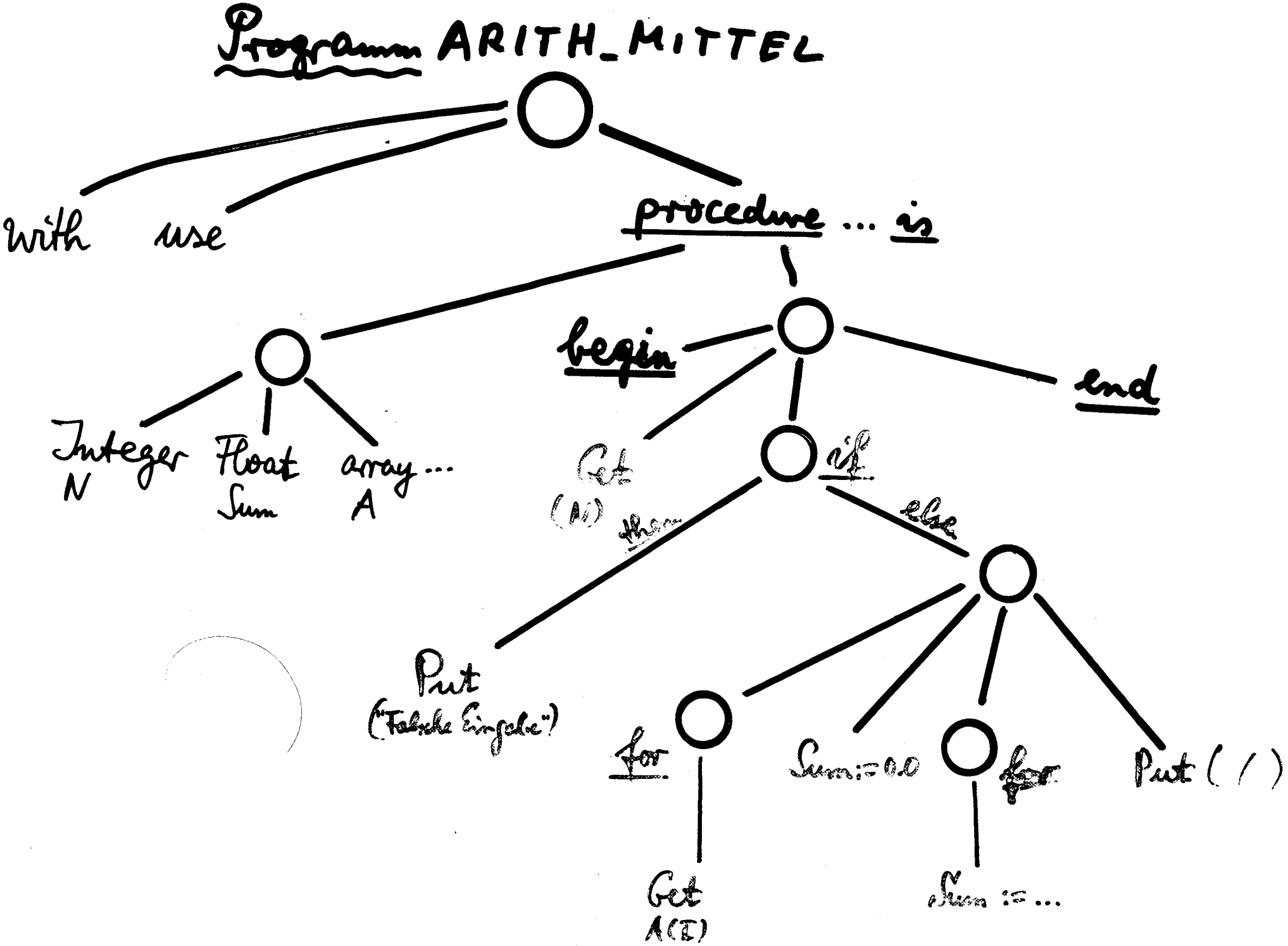
```
with Ada.Integer_Text_IO;
use Ada.Integer_Text_IO;
with Ada.Float_Text_IO;
use Ada.Float_Text_IO;
with Text_IO; use Text_IO;

procedure ARITH_MITTEL is
   N: Integer; Sum: Float;
   A: array (1..2000) of Float;

begin  Get (N);
  if (N < 1) or (N > 2000) then
                Put ("Falsche Eingabe");

  else
     for I in 1..N loop
              Get (A(I)); end loop;

     Sum := 0.0;
     for I in 1..N loop
        Sum := Sum + A(I); end loop;
     Put (Sum / FLOAT(N));
  end if;
end ARITH_MITTEL;
```


Programm ARITH_MITTEL
with
use
procedure ... is
begin
end
Integer
N
Float
Sum
array ...
A
Get
(N)
if
then
else
Put
("Falsche Eingabe")
for
Get
A(I)
Sum:=0.0
for
Sum := ...
Put (/)

2. Darstellungen der elementaren Wertebereiche und der Bezeichner

Bezeichner

≙ Folge von Buchstaben, Ziffern und '_', beginnend mit einem Buchstaben.

Beispiele

A B73

HEUTE EINS_UND_EINS_IST_2

ARITH_MITTEL

W23_ODER_27

Beachte in Ada 95:

Groß- und Kleinbuchstaben werden nicht unterschieden bei Bezeichnern!

Ganze Zahlen

{0,1,2,3,4,5,6,7,8,9}

(a) Folge von Ziffern, evtl. mit Vorzeichen + oder - am Anfang.
[Es dürfen innerhalb der Ziffenfolge Unterstriche '_' zur Lesbarkeit verwendet werden.]

Beispiele -23 62 7134 013
1_241_319 1_0_2_001342

(b) Das "Paket TEXT_IO" lässt auch zu, dass Zahlen binär, hexadesimal oder zu einer anderen (positiven) Basis $b \geq 2$ eingegeben werden. 2#1101# (Basis 2, Zahl ≙ 13)
16#F3# (Basis 16, Zahl ≙ 243)

(c) Man kann einen Exponenten hinzufügen:
1E3 ($= 1 \cdot 10^3 = 1000$), 217E4 ($= 2170000$)

"Reelle" Zahlen

Solche Zahlen müssen einen '.' haben sowie mindestens eine Ziffer rechts und links. Ein Exponententeil

E <ganze Zahl> darf folgen.

Beispiele:

3.7 -16.2314 12.0

0.173 E5 (≙ 17300)

41.9 E-3 (≙ 0,0419)

Zeichen (character)

In Apostroph einschließen: 'A', '+', ',',

aber: '''' für '

Zeichfolgen ("Zeichenketten")

In Anführungsstriche einschließen:

"Heute" "A" "Einführung␣in␣"

Soll ein Zeichenraum sein

Zeichen " hierbei verdoppeln:

"""Danke""", sagte er."

Kommentare einfügen

..... -- .....

Programm — Kommentar bis zum Zeilenende !

Beispiele :

```
A: Character;
X,Y: Float; N,M: Integer;

X := 23.1 + 2.1 E-2;
Y := X * (-4.5E1 + 5.0*(X-7.1));
N := 0;          M :=(N + 10 ) * 2#11#;
N := N+1;
A := 'A';    A := ';';
A := '''';
```

3. Ein - Ausgabe

Standard: füge TEXT_IO hinzu

Normale Schreibweise:
(sei C: character)

TEXT_IO.PUT(C)

Schreibt ein Zeichen in die Ausgabe

dies entfällt, falls use-Klausel USE TEXT_IO am Anfang verwendet wurde

TEXT_IO.PUT("Hallo")
TEXT_IO.PUT_LINE("HEUTE")
TEXT_IO.NEW_LINE
Ada.FLOAT_TEXT_IO.PUT(7.123)
Ada.INTEGER_TEXT_IO.PUT(746)

PUT(N,4) 4 stellige Ausgabe (Standard: 8-stellig)
PUT(123.37, h, v, e)
Stellen des Exponenten
Stellen vor "." Stellen nach "."

4. Anweisungen

siehe Grundvorlesung.

Aber: ; ist zugleich das Ende jeder Anweisung und jeder Deklaration!

X := <Ausdruck>

↑ Variablenname

↖ Typ muss gleich dem von X sein!

;;

... ; ...

IF THEN ELSE END IF

WHILE LOOP ENDLOOP

FOR IN LOOP ENDLOOP

FOR IN REVERSE LOOP ENDLOOP

Spiegeln eines Textes der Länge N

```
WITH Ada.INTEGER_TEXT_IO;
USE Ada.INTEGER_TEXT_IO;
WITH TEXT_IO; USE TEXT_IO;

PROCEDURE SPIEGELN IS
N: INTEGER;
T: ARRAY(1..5000) OF CHARACTER;
BEGIN
  GET(N);
  IF (N<1) OR (N>5000)
  THEN PUT("Falsche Eingabe");
  ELSE
    FOR I IN 1..N LOOP
        GET(T(I)); END LOOP;
    FOR I IN REVERSE 1..N LOOP
        PUT(T(I)); END LOOP;
  END IF;
END SPIEGELN;
```