Elemente der MAPLE–Programmiersprache

A) Wiederholte Ausführung von MAPLE–Anweisungen (do–statement)

Durch eine **do**–Anweisung werden Anweisungen wiederholt ausgeführt.

(1)

[**for** i] [**from** a] [**by** s] [**to** b] [**while** Bedingung]
do [Folge von MAPLE–Anweisungen (durch ; getrennt)]
end do;

Hierbei durchläuft der **Laufindex** i (ein Name) die Zahlen beginnend mit dem **Startwert** a mit einer **Schrittweite** s, solange i den **Endwert** b nicht übersteigt bzw. die angegebene Bedingung erfüllt ist. In jedem Schritt wird die Folge von MAPLE–Anweisungen ausgeführt. Hierbei sind a, s und b numerische Zahlen. Die in eckige Klammern eingeschlossenen Anweisungen können weggelassen werden, es muß allerdings sichergestellt werden, daß die do–Anweisung abbricht. Sind a oder s nicht angegeben, so werden diese automatisch auf 1 gesetzt.

Eine Bedingung im Sinne von MAPLE ist ein sog. **Boole'scher Ausdruck**, das ist ein MAPLE–Ausdruck, der von der MAPLE–Prozedur "evalb" zu "true" (wahr) oder "false" (falsch) ausgewertet wird.

Beispiele:
- **a)** $x = y$, $x <= y$, $x < y$, $x <> y$ $(x \neq y)$, $x >= y$, $x > y$
- **b)** MAPLE–Prozeduren, die "true/false" als Ausgabe haben
- **c)** Logische Verknüpfungen von Boole'schen Ausdrücken durch "and" (und), "or" (oder), "not" (nicht)

B) Bedingte Ausführung von MAPLE–Anweisungen (if–statement)

Durch eine **if**–Anweisung werden Anweisungen nur dann ausgeführt, wenn eine angegebene Bedingung erfüllt ist. Bedingungen sind unter A) erklärt.

(2)

if Bedingung_1 **then** Anweisung_1
[**elif** Bedingung_2 **then** Anweisung_2]
[**elif** Bedingung_3 **then** Anweisung_3]
[**elif** Bedingung_4 **then** Anweisung_4]
$\vdots$
[**elif** Bedingung_n **then** Anweisung_n]
[**else** Anweisung_{n+1}]
end if;

C) Allgemeine Form einer MAPLE–Prozedur

(3)

name := proc([P]) [**local** L;]
    Folge von MAPLE–Anweisungen (durch ; getrennt)
end proc;

Hierbei ist P eine Folge von Variablen. Für jede einzelne Variable x kann durch

x::Datentyp

ein spezieller Datentyp festgelegt werden. L ist eine Folge von lokalen Variablen, das sind Variable, die nur innerhalb der Prozedur verändert werden. [...] bedeutet, daß die Angabe auch weggelassen werden kann.

P kann auch die leere Folge sein, dann entfällt aber die Möglichkeit der Typ–Deklaration der einzelnen Argumente. In diesem Falle kann man – wie schon von den Prozeduren in Pfeil–Notation bekannt – mit

args[i] auf das i–te eingegebene Argument und mit

nargs auf die Anzahl der eingegebenen Argumente

zurückgreifen.

Eine Prozedur wird durch name(A); (A: Argument–Folge) aufgerufen. Ausgegeben wird das Ergebnis der letzten Anweisung innerhalb der Prozedur.

D) Der euklidische Algorithmus

Ein Beispiel: $a = 42$, $b = 5$

Bilde den Rest r bei Division von a durch b : $r = 2$

Teile $b = 5$ durch $r = 2$. Der Rest ist 1. Teile nun r durch 1. Der Rest ist 0, also $1 = \mathrm{ggT}(42, 5)$. Für einen Algorithmus ist es (aus Speicherplatzgründen) günstiger, a und b immer durch die neuen Werte zu ersetzen und den Rest bei Division von a durch b zu bilden. Dies führt zu dem folgenden Pseudo–Code für den EA:

(4)

Input: $a > 0$, $b > 0$
$r \longleftarrow \mathrm{Rest}(a, b)$
while $r \neq 0$ do
    $a \longleftarrow b$
    $b \longleftarrow r$
    $r \longleftarrow \mathrm{Rest}(a, b)$
Output: b (ggT)

Als MAPLE–Prozedur:

Der euklidische Algorithmus in $\mathbb{N}$

(5)

```
ggt := proc(x :: posint, y :: posint) local a, b, r;
       a := x;
       b := y;
       r := irem(a, b);
       while r <> 0 do
             a := b;
             b := r;
             r := irem(a, b);
       end do;
       b
       end proc;
```

Es ist zu beachten, daß in (5) die Variablen x und y **formale Parameter** sind, die innerhalb der Prozedur **nicht** verändert werden dürfen. Daher werden die Hilfsgrößen a und b eingeführt.

Wir wollen diese Prozedur jetzt auf ganze Zahlen ausweiten. Dazu beachten wir (2.9b) und behandeln den Fall $y = 0$ durch eine **if**–Abfrage gesondert.

Der euklidische Algorithmus in $\mathbb{Z}$

(6)

```
ggt := proc(x :: integer, y :: integer)
       local a, b, r;
       a := abs(x);
       b := abs(y);
             if b = 0 then a
             else
                   r := irem(a, b);
                   while r <> 0 do
                         a := b;
                         b := r;
                         r := irem(a, b)
                   end do;
                   b
             end if
       end proc;
```