```
> restart:
```

Aufgabe 3

Iteration versus Rekursion (mit option remember)

Schreiben Sie zwei Prozeduren zur effizienten Berechnung des n-ten Fibonacci-Polynoms (Leonardo Fibonacci). Die Fibonacci-Polynome sind durch die Rekursion

$$F_0 = 0, F_1 = 1, F_n = x F_{n-1} + F_{n-2}$$

definiert. Programmieren Sie die Rekursion

a) mittels "for loop" und "double assignment" (Doppelzuweisung, allgemein Mehrfachzuweisung)

Hinweis: Expandieren Sie die Polynome in jedem Rekursionsschritt (expand), sowohl in a) als auch in b).

```
> fib_it := proc(n::integer, polynom::boolean := true)::integer;
    local f_n_minus_one, f_n_minus_two, f_n, i, x;
    description "Fibonacci iterativ. Wenn polynom auf true steht,
  so wird ein Fibonacci-Polynom ausgewertet, ansonsten die
  Fibonacci-Zahl.";

    if n < 0 then return(FAIL) end if;

    f_n_minus_one, f_n_minus_two, f_n := 0, 1, 1;

    if n = 0 then
      return f_n_minus_one
    elif n = 1 then
      return f_n_minus_two
    end if;

    if polynom then x:=x; else x:=1; end if;

    for i from 2 by 1 to n do
      f_n_minus_two := f_n_minus_one;
      f_n_minus_one := f_n;
      f_n := expand(x * f_n_minus_one + f_n_minus_two);
    end do;

    f_n
```

```
  end proc;
```

```
fib_it := proc(n::integer, polynom::boolean := true)::integer;
    local f_n_minus_one, f_n_minus_two, f_n, i, x;
    description
    "Fibonacci iterativ. Wenn polynom auf true steht, so wird ein Fibonacci-Polynom ausgewertet,
    ansonsten die Fibonacci-Zahl.";
    if n < 0 then return FAIL end if;
    f_n_minus_one, f_n_minus_two, f_n := 0, 1, 1;
    if n = 0 then return f_n_minus_one elif n = 1 then return f_n_minus_two end if;
    if polynom then x := x else x := 1 end if;
    for i from 2 to n do
        f_n_minus_two := f_n_minus_one;
        f_n_minus_one := f_n;
        f_n := expand(x*f_n_minus_one + f_n_minus_two)
    end do;
    f_n
end proc
```

(1)

b) mittels rekursiver Prozedur mit "option remember".

```
> fib_rek := proc(n::integer, polynom::boolean := true)::integer;
    option remember;
    description "Fibonacci rekursiv. Wenn polynom auf true steht,
  so wird ein Fibonacci-Polynom ausgewertet, ansonsten die
  Fibonacci-Zahl.";
    if polynom then
      if n < 2 then n else expand(x * fib_rek(n-1) + fib_rek(n-2))
  end if;
    else
      if n < 2 then n else fib_rek(n-1, false) + fib_rek(n-2,
  false) end if;
    end if;
  end proc;
```

```
fib_rek := proc(n::integer, polynom::boolean := true)::integer;
    option remember;
    description
    "Fibonacci rekursiv. Wenn polynom auf true steht, so wird ein Fibonacci-Polynom ausgewertet,
```

(2)

```
    ansonsten die Fibonacci-Zahl.";
    if polynom then
        if n < 2 then n else expand(x*fib_rek(n - 1) + fib_rek(n - 2)) end if
    else
        if n < 2 then n else fib_rek(n - 1, false) + fib_rek(n - 2, false) end if
    end if
end proc
```

c) Vergleichen Sie die CPU-Zeiten für die Berechnungsmethoden in a) und b) bei $n = 3000$. Erklären Sie das beobachtete Laufzeitverhalten.

```
> n := 3000;
  time(fib_it(n));
  time(fib_rek(n));
```

$$n := 3000$$

$$1.281$$

$$3.000 \qquad \textbf{(3)}$$

Antwort: Die Laufzeitunterschiede verhalten sich ähnlich wie in anderen Programmiersprachen, da die rekursive Funktion meistens die einfachere und auch die mit weniger Aufwand programmierbare ist. Allerdings wird der Stack durch die vielen rekursiven Aufrufe vollgeschrieben und muss dann in einer Art Baummodell abgearbeitet werden. Dies ist extrem langsam und daher ist die iterative Variante, bei der nur meistens nur der letzte Eintrag gespeichert werden muss viel schneller, als wenn immer bis auf die Abbruchbedingung zurückgegangen werden muss.

d) Für $x = 1$ definiert die obige Rekursion die Fibonacci-Zahlen. Modifizieren Sie die Parameterlisten der Prozeduren in a) und b) so, dass sowohl Fibonacci-Polynome als auch Fibonacci-Zahlen berechnet werden können.

```
> for i from 0 by 1 to 5 do
    fib_it(i, false);
    fib_rek(i, false);
  end do;
```

$$0$$

$$0$$

$$1$$

$$1$$

$$1$$

$$1$$

$$2$$

2

3

3

5

5 **(4)**