# Kurs 1612 "Konzepte imperativer Programmierung"
# Kurs 1613 "Einführung in die imperative Programmierung"

Studientag am 22.01.2000 bzw. am 23.01.2000

Aufgaben

---

## Aufgabe 1

Schreiben Sie eine PASCAL-Prozedur `transponiereMatrix`, die als Parameter eine quadratische Matrix von `integer`-Werten erhält und diese Matrix transponiert, also die Zeilen und Spalten der Matrix vertauscht, wie im folgenden Beispiel gezeigt ist:

$$\begin{bmatrix} 15 & 8 & 1 & 24 & 17 \\ 16 & 14 & 7 & 5 & 23 \\ 22 & 20 & 13 & 6 & 4 \\ 3 & 21 & 19 & 12 & 10 \\ 9 & 2 & 25 & 18 & 11 \end{bmatrix} \rightarrow \begin{bmatrix} 15 & 16 & 22 & 3 & 9 \\ 8 & 14 & 20 & 21 & 2 \\ 1 & 7 & 13 & 19 & 25 \\ 24 & 5 & 6 & 12 & 18 \\ 17 & 23 & 4 & 10 & 11 \end{bmatrix}$$

Verwenden Sie dazu die folgenden Definitionen sowie den angegebenen Prozedurkopf von `transponiereMatrix` mit den vorgegebenen Deklarationen!

```
const
ZEILENSPALTENMAX = 5;

type
tZeileSpalte = 1..ZEILENSPALTENMAX;
tMatrix = array [tZeileSpalte, tZeileSpalte] of integer;

procedure transponiereMatrix (var ioMat : tMatrix);
{ transponiert die quadratische Matrix ioMat }

  var
  i,
  j : tZeileSpalte; { Zeilen- und Spaltenindex }
  { ...evtl. weitere Deklarationen }
```

# Kurs 1612 "Konzepte imperativer Programmierung"
# Kurs 1613 "Einführung in die imperative Programmierung"

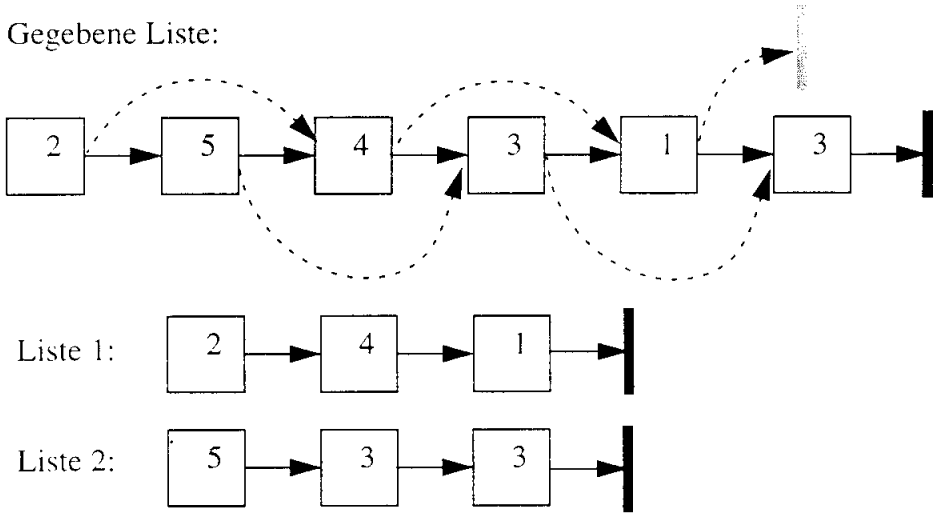
Studientag am 22.01.2000 bzw. am 23.01.2000

Aufgaben

---

## Aufgabe 2

Schreiben Sie eine Prozedur `ListenUnmerge`, die eine gegebene Liste von `integer`-Werten nur durch Änderung der Verkettung in zwei Listen aufteilt. Die erste Liste soll das erste, dritte, ..., Element der Eingabeliste enthalten, die zweite Liste das zweite, vierte, ... Element, wie im folgenden Beispiel grafisch gezeigt ist. (Die gestrichelten Pfeile deuten die Änderung der Verkettung an.) Sie können davon ausgehen, daß die Eingabeliste mindestens zwei Elemente enthält.




Verwenden Sie in Ihrer Prozedur die folgenden Typdefinitionen und den angegebenen Prozedurkopf.

```
type
tRefListe = ^tListe;
tListe = record
           info : integer;
           next : tRefListe
         end;

procedure ListenUnmerge (
                inListe : tRefListe;
             var outListe1, outListe2 : tRefListe);
{ teilt die aus mindestens 2 Elementen bestehende inListe nur
  durch Aendern der Verkettung auf outListe1 und outListe2 auf }
```

Die Parameter der Prozedur `ListenUnmerge` haben folgende Bedeutung:

| | |
|---|---|
| `inListe` | Zeiger auf die aufzuteilende Liste, |
| `outListe1` | Zeiger auf die Liste mit dem 1., 3., ... Element aus der aufzuteilenden Liste, |
| `outListe2` | Zeiger auf die Liste mit dem 2., 4., ... Element aus der aufzuteilenden Liste. |

## Aufgabe 3

Gegeben seien die folgenden Typdefinitionen eines binären Suchbaums und einer linearen Liste:

```
type
tRefBinBaum = ^tBinBaum;
tBinBaum =  record
              info : integer;
              links,
              rechts : tRefBinBaum
            end;
tRefListe = ^tListe;
tListe = record
           info : integer;
           next : tRefListe
         end;
```

Schreiben Sie eine PASCAL-Funktion `Baumpfad`, welche die gefundenen Werte auf dem Suchpfad von der Wurzel eines nicht-leeren Suchbaums zu einem gesuchten Wert in einer Liste ablegt und diese zurückgibt. Ist der gesuchte Wert nicht im Baum enthalten, so soll eine leere Liste (also ein Zeiger mit Wert **`nil`**) zurückgegeben werden. Durchlaufen Sie dazu in Ihrer Prozedur den Suchbaum **iterativ** anhand des Suchwertes `inSuchwert`, und fügen Sie während des Durchlaufs die gefundenen Werte hinten an die Liste an. Wird `inSuchwert` nicht gefunden, ist die aufgebaute Liste wieder zu löschen. Benutzen Sie den unten angegebenen Funktionskopf von `Baumpfad`!

```
function Baumpfad (
           inWurzel: tRefBinBaum;
           inSuchwert: integer) : tRefListe;
{ sucht einen Wert im Baum und gibt eine Liste aller Werte auf dem
  Pfad zum gesuchten Wert zurueck }
```

# Kurs 1612 "Konzepte imperativer Programmierung"
# Kurs 1613 "Einführung in die imperative Programmierung"

Studientag am 22.01.2000 bzw. am 23.01.2000

Aufgaben

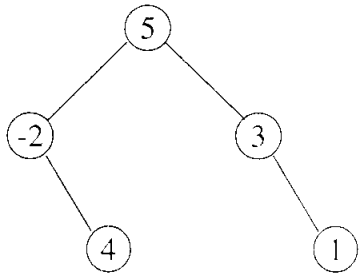
---

## Aufgabe 4

Gegeben sei die folgende Typdefinition eines Binärbaums in PASCAL:

```
type
tRefBinBaum = ^tBinBaum;
tBinBaum = record
             info : integer;
             links,
             rechts : tRefBinBaum
           end;
```

Gesucht ist eine PASCAL-Prozedur, die bei einem Baumdurchlauf in symmetrischer Reihenfolge (inorder) eine Folge von Zahlen ausgibt. Bei jedem Knotenbesuch wird eine Zahl ausgegeben, die gleich dem Maximum der bis dahin betrachteten Werte in den `info`-Komponenten des Binärbaums ist.
Für den Baum




ergibt sich z.B. die Ausgabe: -2 4 5 5 5. Ein Aufruf der Prozedur im Hauptprogramm geschieht durch

```
BinBaumInorderMax (Wurzel, Maximum);
```

wobei `Wurzel` ein Zeiger auf die Wurzel des zu durchlaufenden Binärbaums und `Maximum` eine mit `-maxint` initialisierte Variable ist.
Benutzen Sie den unten angegebenen Prozedurkopf von `BinBaumInorderMax` und überlegen Sie, ob eine rekursive oder iterative Lösung vorzuziehen ist. Begründen Sie ihre Auswahl.

```
procedure BinBaumInorderMax (
                inWurzel : tRefBinBaum;
            var ioMax : integer);
{ ioMax enthaelt das (Teilbaum-) Maximum, inWurzel den Zeiger
  auf die Wurzel des Binaerbaums }
```

# Achtung!

Die nachfolgenden Aufgaben sind nur für die Beleger des Kurses **1612 „Konzepte imperativer Programmierung"** relevant.

## Aufgabe 5

Das folgende PASCAL-Programm `GroessterExponent` berechnet für zwei positive ganze Zahlen `K` und `X` einen Exponenten h, so daß $K^h \leq X < K^{h+1}$ gilt:

```
program GroessterExponent;
(* berechnet fuer zwei integer-Zahlen K>1 und X>0 einen Exponenten
   h, so dass K^h <= X < K^(h+1) gilt *)

  const
  K = ..;
  X = ..;

  var
  y,
  h: integer;

{ Vorbedingung: P ≡ K>1 ∧ X≥1 }
begin
  y := K;
  h := 0;
  while y <= X do
  begin
    y := y * K;
    h := h + 1
  end
end.(* GroessterExponent *)
{ Nachbedingung: Q ≡ K^h ≤ X < K^(h+1) }
```

a) Wann ist ein Prädikat INV eine geeignete Invariante der **while**-Schleife von `GroessterExponent`? Geben Sie die Programmformel sowie eventuell zusätzliche Bedingungen an, die dies ausdrücken.

b) Geben Sie eine geeignete Invariante INV an.
(Tip: Welche Beziehung besteht zwischen `y`, `K` und `h`?)

c) Beweisen Sie nun die Invarianteneigenschaft des von Ihnen gefundenen Prädikats INV.

d) Zeigen Sie jetzt die partielle Korrektheit von `GroessterExponent` mit Hilfe der Hoare-Regeln.

---

## Aufgabe 6

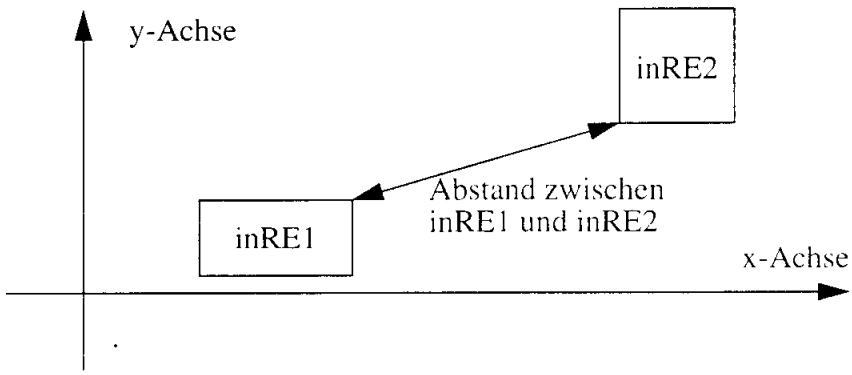
Beweisen Sie nun, daß das Programm `GroessterExponent` aus Aufgabe 5 terminiert, das heißt, finden Sie eine geeignete Funktion $\tau$ und zeigen Sie, daß $\tau$ die Eigenschaften einer Terminierungsfunktion für die **`while`**-Schleife besitzt.

## Aufgabe 7

Die Funktion `Abstand` soll den kleinsten Abstand zwischen zwei disjunkten, achsenparallelen Rechtecken in der euklidischen Ebene bestimmen:

```
function Abstand (inRE1, inRE2 : tRechteck) : real;
```

Beispiel:




Sowohl die Implementierung der Funktion als auch die genaue Deklaration des Datentyps `tRechteck` spielen hier keine Rolle.

Ihre Aufgabe ist es, Eingabeäquivalenzklassen zu bilden, anhand derer die Funktion getestet werden kann. Dazu können Sie ein Rechteck unverändert lassen, während Sie die Lage und Größe des zweiten Rechtecks variieren. Es reicht aus, wenn Sie nur die Fälle betrachten, in denen `inRE2` rechts von `inRE1` liegt.

## Aufgabe 8

Gegeben sei eine natürliche Zahl z. Zur Dezimaldarstellung von z, d.h. zu

$$z = \sum_{i=0}^{n} a_i \cdot 10^i \text{ mit } n \in \mathbb{N},\ a_0, \ldots, a_n \in \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\},$$

soll die Quersumme berechnet werden, also $\sum_{i=0}^{n} a_i$ .

(Die Darstellung von z kann sich durch die zusätzliche Angabe führender Nullen unterscheiden. Diese tragen aber nicht zur Veränderung des in der Quersumme berechneten Wertes bei.) Wir bieten als Lösung folgendes Programm `QuersummeF` an:

```
program QuersummeF (input, output);
{ soll die Quersumme einer nicht-negativen integer-Zahl berechnen }

  type
  tNatZahl = 0..maxint;

  var
  Quer,
  Rest : tNatZahl;

begin
  readln (Rest);
  Quer := 0;
  while Rest > 0 do
  begin
    Quer := Rest mod 10;
    Rest := Rest div 10
  end;
  writeln (Quer)
end. { QuersummeF }
```

a) Zeichnen Sie den kompakten Kontrollflußgraphen zum Programm `QuersummeF`.

b) Geben Sie Pfade im Kontrollflußgraphen für einen Zweigüberdeckungstest an. Geben Sie außerdem zu jedem Pfad den assoziierten Testfall und ein Testdatum an.

c) Wie unterscheidet sich der boundary interior-Pfadtest für das Programm `QuersummeF` vom Zweigüberdeckungstest aus Teil b) dieser Aufgabe? Geben Sie falls nötig weitere Testfälle und Testdaten an. Ist es möglich, daß auch dieses Testverfahren den Fehler im Programm nicht aufdeckt? Begründen Sie Ihre Antwort.

---

d) Zeichnen Sie den Kontrollflußgraphen des korrigierten Programms `QuersummeF` in Datenflußdarstellung.

e) Ermitteln Sie die *du*-Mengen. Welche Teilpfade ergeben sich für einen Test nach dem *all uses*-Kriterium? Geben Sie Pfade an, mit denen diese Teilpfade abgedeckt werden.