Institut 186 für Computergraphik	**Algorithmen und Datenstrukturen 1**			Beispieltest 27. Juni 1997 Gruppe B
Matrikelnr.	**Beil.**†	1 (9)	2 (21)	3 (20)
NACHNAME, Vorname				Σ (50)

† Geben Sie an, wieviele Zusatzbl. Sie abgeben (**jedes mit Name & Matr.-Nr. beschriftet!**).

(9) Aufgabe 1 – Diverses

a) Gegeben ist der Landkartenfärbealgorithmus aus dem 7. Beispiel. Ein Programmierer kommt auf die Idee, einen binären Suchbaum zu verwenden, um jeweils das Land mit den meisten ungefärbten Nachbarn zu finden. Ist das besser oder schlechter als die im Beispiel vorgeschlagene Datenstruktur?

b) Gegeben ist das WATERWORLD Programm aus dem 9. Beispiel. Geben Sie zu jeder der 6 Fähigkeiten eines Lebewesens (simuliert werden, vermehren, dargestellt werden, eigelesen werden, eingegeben werden, auf File geschrieben werden) an, ob die jeweilige Methode der Klasse `LEBEWESEN` virtuell oder nicht virtuell ist (Keyword: `VIRTUAL`).

c) Gegeben ist die Bibliotheksverwaltung aus dem 5. Beispiel. Es soll zusätzlich zu den angegebenen Operationen auch noch möglich sein einen nach ISBN Nummern sortierten Katalog der Bücher zu erstellen. Ist die gewählte Datenstruktur (Hashtable) für diese Operation geeignet? Welche zusätzliche Datenstruktur würden Sie für diese Operation vorschlagen?

Achtung: Begründung sie alle Ihre Antworen! Ein richtiges Resultat oder eine richtige Antwort wird **nicht gewertet**, wenn aus Ihrer Antwort nicht hervorgeht, wie Sie zu dem Ergebnis oder dem Schluß gekommen sind.

(21) Aufgabe 2 – Tabellenkalkulation (6. Beispiel)

Eine einfache Tabellenkalkulation, die nur das Addieren von zwei referenzierten Feldern, und das Eintragen eines direkten Zahlenwertes erlaubt, sei gegeben. Die dazu notwendigen Datenstrukturen sehen folgendermaßen aus:

```
TYPE
  ReferenzPtr = ^Referenz;
  Referenz = RECORD
               zeile, spalte: INTEGER;
               next: ReferenzPtr;
             END;
  Cell = RECORD
           wert: REAL;
           operanden: ReferenzPtr;      (* Backpointers *)
           forward: ReferenzPtr;        (* Forwardpointers *)
         END;
 VAR tabelle: ARRAY [1..MAXZEILE,1..MAXSPALTE] OF Cell;
```

Bei einer nicht berechneten Zelle ist `operanden` gleich `NIL`, bei einer berechneten Zelle ist `operanden` eine Liste mit zwei Elementen, die auf die beiden Operanden verweisen. `forward` ist eine Liste mit Referenzen auf alle abhängigen Zellen. Sie können annehmen, daß in den Abhängigkeiten keine Zyklen vorkommen. Sowohl `operands` als auch `forward` sind schon korrekt gesetzt.
Ein alternativer Algorithmus zum Neuberechnen aller von einer veränderten Zelle abhängigen Zelle funktioniert folgendermaßen: zuerst berechnet man den neuen Wert der veränderten Zelle, danach berechnet man die neuen Werte aller direkt abhängigen Zellen, danach berechnet man die neuen Werte aller einfach indirekt abhängigen Zellen, usw.
Schreiben sie eine *rekursive* Funktion

```
FUNCTION berechneebene(ebene,zeile,spalte: INTEGER) : BOOLEAN;
```

die alle von der angegebenen Zelle abhängigen Zellen, neu berechnet, wobei `ebene` den Grad der Indirektion angibt, d.h. wenn `ebene = 0` ist, wird nur der Wert der Zelle neu berechnet, wenn `ebene = 1` ist wird der Wert aller direkt abhängigen Zellen neu berechnet, wenn `ebene = 2` werden nur alle einfach indirekt abhängigen Zellen neu berechnet, usw.. Der Rückgabewert der Funktion soll genau dann `TRUE` sein, wenn mindestens eine Neuberechnung durchgeführt wurde. Um Mehrfachberechnungen zu vermeiden, verwenden Sie folgendes Feld:

```
VAR besucht: ARRAY [1..MAXZEILE,1..MAXSPALTE] OF BOOLEAN;
```

von dem Sie annehmen können, daß alle Elemente mit `FALSE` initialisiert sind. Schreiben Sie nun die *effiziente* Prozedur

```
PROCEDURE berechne(zeile,spalte: INTEGER);
```

die `berechneebene` verwendet um den alternativen Algorithmus vollständig zu implementieren.

(20) Aufgabe 3 – Sparse Matrizen als Quadtree (1. Beispiel)

In einem Programm werden spärlich besetzte Matrizen als Quadtrees repräsentiert. Hierzu werden folgende Datenstrukturen verwendet:

```
QuadTree = ^Knoten;
Knoten =    OBJECT
              FUNCTION istZwischen : BOOLEAN; VIRTUAL;
              FUNCTION plus(otherTree: QuadTree) : QuadTree; VIRTUAL;
              FUNCTION subKnoten(i : 1..4) : QuadTree; VIRTUAL;
              FUNCTION getWert : FLOAT; VIRTUAL;
            END;
Blatt  =    OBJECT(Knoten)
              wert: REAL;
              CONSTRUCTOR Init(neuer_wert: REAL);
              FUNCTION istZwischen: BOOLEAN;  (* immer FALSE *)
              FUNCTION subKnoten(i : 1..4) : QuadTree; (* immer NIL *)
              FUNCTION getWert : FLOAT; (* returns wert *)
            END;
Zwischen =  OBJECT(Knoten)
              subKnotenTabelle: ARRAY [1..4] OF QuadTree;
              CONSTRUCTOR Init(neue_subKnoten: ARRAY[1..4] OF QuadTree);
              FUNCTION istZwischen: BOOLEAN;  (* immer TRUE *)
              FUNCTION subKnoten(i : 1..4) : QuadTree;
                  (* accessor method for subKnotenTabelle[1..4] *)
              FUNCTION getWert : FLOAT; (* error when called *)
            END;
```

Implementieren Sie die beiden Methoden die notwendig sind, um zwei als Quadtree repräsentierte Matrizen zu addieren:

```
FUNCTION Zwischen.plus(otherTree: QuadTree) : QuadTree;
FUNCTION Blatt.plus(otherTree: QuadTree) : QuadTree;
```

Hierbei ist zu beachten, daß die resultierende Matrix neu angelegt und als Funktionswert zurückgeliefert werden soll. Desweiteren soll beachtet werden, daß vier Knoten zusammegefasst werden, sobald sie den gleichen Wert haben. Vorsicht, das kann mehrere Ebenen direkt hintereinander betreffen! Sie können davon ausgehen, daß die übergebenen QuadTrees valid, d.h. ungleich NIL sind.