Institut 186 für Computergraphik	**Algorithmen und Datenstrukturen 1**				VO-Prüfung 14. Jänner 1998 Gruppe A	
Kennz.†	**Matrikelnr.**	**Beil.**‡	1 (30)	2 (15)	3 (25)	4 (30)
Nachname, Vorname						Σ (100)

† Geben Sie jene Kennzahl an, auf die das Zeugnis ausgestellt werden soll.
‡ Geben Sie an, wieviele Zusatzbl. Sie abgeben (**jedes mit Name & Matr.-Nr. beschriftet!**).

(30) Aufgabe 1 – Graphen

Eine Adjazenzmatrix soll mittels eines Quadtrees speichersparend implementiert werden, da ja nur sehr wenige Kanten wirklich vorhanden sind. Hierzu werden im Quadtree statt der Farben (Weiß, Schwarz), `BOOLEAN`s gespeichert, die angeben ob die entsprechende Kante vorhanden ist oder nicht. Sie können davon ausgehen, daß der Graph $n = 2^k$ Knoten enthält ($k > 0$).

(10) **a)** Entwerfen Sie eine geeignete Datenstruktur `GraphType` zum Speichern einer Adjazenzmatrix als Quadtree.

(20) **b)** Eine Adjazenzmatrix definiert genau dann einen *ungerichteten* Graphen wenn $a_{i,j} = a_{j,i}$ ist. Schreiben Sie nun eine Funktion

```
istUngerichtet(graph : GraphType) : BOOLEAN
```

die genau dann `TRUE` zurückliefert, wenn der übergebene Graph ungerichtet ist. Achten Sie darauf daß Ihr Algorithmus (er kann auch aus mehreren Funktionen bestehen) *effizient* und *rekursiv* ist!

Hinweis: Sie sollen die Quadtree-Datenstruktur für Ihren Algorithmus ausnützen. D.h. Sie müssen einen Algorithmus entwerfen der möglichst große Blöcke (idealerweise die entsprechenden Blöcke des Quadtrees) der Adjazenzmatrix auf einmal auf die angegebene Symmetrie prüft.

(15) Aufgabe 2 – Quicksort

Gegeben ist die folgende, unsortierte Zahlenfolge:

```
30 41 18 25 19 35 22 21 28 39 37 13 27 32 29
```

Sortieren Sie diese Zahlenfolge *aufsteigend* mit **Quicksort**. Verwenden Sie den Algorithmus so, wie er im Skriptum steht. **Geben Sie jeden Zwischensritt an**, in der Tabelle ist ausreichend Platz. Achten Sie darauf, daß **in jeder Zeile nur Teilfolgen derselben Rekursionstiefe** stehen. *Kennzeichene Sie deutlich die Grenzen der Teilfolgen!*

Als Hilfe enthält die erste Zeile der Tabelle die Indices der Elemente.

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
30	**41**	**18**	**25**	**19**	**35**	**22**	**21**	**28**	**39**	**37**	**13**	**27**	**32**	**29**

(25) Aufgabe 3 – Suchbaum

(10) **a)** Entfernen Sie aus dem folgenden binären Suchbaum das Element mit dem Schlüssel **25**, dann das Element mit dem Schlüssel **15**. **Holen Sie die Ersatzknoten immer aus dem linken Teilbaum!** Zeichen Sie den Baum nach dem Löschen von jdem der beiden Schlüsseln (also zwei mal)!

(15) **b)** Gegeben ist die folgende Definitionen für einen binären Suchbaum:

```
TYPE
  KnotenPtr = POINTER TO Knoten;
  Baum = OBJECT
           pKnoten : KnotenPtr;
           FUNCTION isNil : BOOLEAN; { TRUE, wenn pKnoten = NIL }
           FUNCTION getInfo : INTEGER; { gibt pKnoten^.info }
         END
  Knoten = RECORD
             info : INTEGER;
             links : Baum;   { kleinere Elemente }
             rechts : Baum;  { groessere Elemente }
           END;
```

Schreiben Sie eine *effiziente, rekursive* Prozedur

```
Baum.druckeBereich(von, bis : INTEGER);
```

die alle Knoten des baumes, die sich im übergebenen Bereich befinden (`von <= info <= bis`), in *absteigender* Reihenfolge ausgibt.

Bemerkung: Effizient heißt, daß Teilbäume, die sicher keinen Schlüssel im angegebenen Bereich enthalten, nicht untersucht werden.

(30) Aufgabe 4 – Objektorientierter Stack

In objektorientiertem Turbo Pascal sollen Stacks von `INTEGER` Zahlen als Objekte verwirklicht werden.

Die Basisklasse `Stack` gibt das Klassen-Interface vor: Mit der Methode `isEmpty : BOOLEAN` soll erfragt werden können ob der Stack leer ist, mit der Methode `push(value : INTEGER);` soll ein Wert auf den Stack gelegt werden können, mit der Methode `pop : INTEGER` soll der oberste vom Stack entfernt und zurückgeliefert werden.

Die Subklasse `ArrayStack` verwaltet den Stack als Array von Elementen. Sie können davon ausgehen das maximal `CONST maxElements = ...;` Elemente in einem Stack Platz finden müssen.

Die Subklasse `ListStack` verwaltet verwaltet den Stack als einfach verkettete Liste von Elementen.

Entwerfen und implementieren Sie die Klassen `Stack, ArrayStack` und `ListStack`, und deren Methoden `isEmpty` und `pop`. Konstruktoren und Destruktoren müssen Sie nicht berücksichtigen. Auf Fehlerbehandlung können sie ebenfalls verzichten.

Schreiben Sie außerdem eine Methode `printAndClearStack`, die nacheinander die Elemente aus einem Stack entfernt und sie ausgibt.