Institut 186 für Computergraphik	**Algorithmen und Datenstrukturen 1**				VO-Prüfung 17. Oktober 1997 Gruppe B	
Kennz.†	**Matrikelnr.**	**Beil.**‡	1 (25)	2 (25)	3 (25)	4 (25)
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!**).

(25) Aufgabe 1 – Patientendatenbank

Die Patienten eines Ambulatoriums müssen bei ihrem Eintreffen in der Ordination sofort in einer Datenbank erfaßt werden. Es werden für jeden Patienten der Vorname (max. 40 Zeichen), der Nachname (max. 40 Zeichen) und die Sozialversicherungsnummer (10 Zeichen) gespeichert. Die Patienten sollen in der Reihenfolge ihres Eintreffens behandelt werden, daher muß es diese Datenbank erlauben, möglichst schnell den nächsten zu behandelnden Patienten zu finden. Desweiteren soll es möglich sein, zu jedem Zeitpunkt eine nach der Sozialversicherungsnummer sortierte Liste der gerade wartenden Patienten auszugeben.

(10) **a)** Entwerfen sie eine kombinierte Datenstruktur, die alle geforderten Operationen (Eintragen, nächster Patient, sortierte Ausgabe) *effizient* unterstützt. Welche Datenstrukturen sind notwending? Geben sie sowohl die Typdefinition als auch die Variablen an, die dazu nötig sind.

(10) **b)** Schreiben sie die Prozedur

```
PROCEDURE Insert(firstname, lastanme : STRING40; svnr : STRING10);
```

die einen neu angekommenen Patienten in Ihre Datenstruktur einfügt.

(5) **c)** Schreiben sie nun ebenfalls die Prozedur

```
PROCEDURE SortedOutput(...);
```

die eine nach der Sozialversicherungsnummer sortiere Liste der gerade wartenden Patienten ausgibt.

Um die Namen zu vergleichen, steht ihnen die Funktion

```
FUNCTION Before(svnr1,svnr2 : STRING10): BOOLEAN;
```

zur Verfügung, die genau dan `TRUE` zurückliefert, wenn `svnr1` vor `svnr2` in der Sortierung kommt.

(25) Aufgabe 2 – Handelsfirma

Eine große Handelsfirma hat alle österreichischen Städte und deren Straßenverbindungen in einem ungerichteten, gewichteten Graphen gespeichert, der durch eine Adjazenzliste gegeben ist. Für jede Verbindung zwischen zwei Städten ist bekannt, wie lange man mit dem Auto braucht, um sie zurückzulegen.

```
CONST AnzahlDerStaedte = ...;
TYPE  Pointer = ^Knoten;
      Knoten = RECORD
                  strasseNach: INTEGER;
                  zeit: REAL;  (* in Stunden *)
                  next: Pointer;
               END;
      StaedteGraph = ARRAY[1..AnzahlDerStaedte] of Pointer;
VAR   streets : StaedteGraph;
      erreichbar : ARRAY[1..AnzahlDerStaedte] of BOOLEAN;
```

Die Firma möchte nun herausfinden, welche Städte von ihren Vertretern innerhalb eines Arbeitstages (8 Stunden) erreichbar sind. Schreiben Sie eine *rekursive* Prozedur:

```
PROCEDURE BerechneErreichbar(...);
```

die im Feld `erreichbar` alle jene Städte mit `TRUE` markiert, die innerhalb von 8 Stunden erreichbar sind. Nehmen Sie dabei an, daß sich die Vertreter 15 Minuten in jeder Stadt aufhalten. Achten Sie außerdem darauf, daß die Vertreter am Ende des Arbeitstages (also innerbalb von 8 Stunden) wieder in die Ausgangsstadt (die Stadt mit der Nummer 1) zurückkehren müssen. Definieren Sie außerdem alle zusätzlich benötigten globalen Variablen, und zeigen Sie wie `BerechneErreichbar` aufgerufen werden muß.

Hinweis: Verwenden Sie Depth-First Search!

(25) Aufgabe 3 – Raum im Forschungszentrum Seibersdorf

In objekt-orientiertem Pascal soll eine Raumverwaltung für das Forschungszentrum Seibersdorf modelliert werden. Von jedem Raum ist bekannt, wie groß er ist (in Quadratmetern) und welchem Institut er zugeordnet ist (per Institutsnummer). Für die Verwaltung ist außerdem relevant, ob es sich um ein Büro, ein Labor oder einen Gang handelt.

Für jedes Büro ist außerdem noch bekannt, wieviele Personen es nutzen, und für Labors ist die Anzahl der Labor-Arbeitsplätze bekannt.

Die Verwaltung vom Forschungszentrum Seibersdorf möchte nun für jede Abteilung feststellen, wieviele Quadratmeter Raum für jede Person zu Verfügung stehen. Da jedoch Gänge nur wenig genutzt werden können, zählt die Gangfläche nur zu 10%. Desweiteren sind die Labors zumeist nur zu 50% benützt, daher zählt jeder Labor-Arbeitsplatz nur für eine halbe Person.

(18) **a)** Definieren Sie die Klassen `Room`, `Office`, `Laboratory` und `Hallway`. Implementieren Sie die Methoden `getDepartmentNumber`, `getUsableArea`, und `getUsingPersons`, die die Institutsnummer, die nutzbare Fläche und die Anzahl der benützenden Personen für jede Art von Raum berechnet. Überlegen Sie dabei, ob diese Methoden virtuell sein müssen. Konstruktoren und Destruktoren sind der Einfachheit halber nicht notwendig.

(7) **b)** Gegeben sei nun ein Array von Räumen des Forschungszentrums Seibersdorf:

```
CONST numberOfRooms = ...;
      numberOfDepartments = ...;
TYPE  RoomPtr = ^Room;

VAR   rooms : ARRAY[1..numberOfRooms] OF RoomPtr;
```

Schreiben Sie die Prozedur

```
PROCEDURE AreaPerPerson;
```

die die folgenden Felder korrekt mit Werten auffüllt. Sie können annehmen, daß die Feldelemte bereits mit den Werten 0.0 initialisiert sind.

```
VAR usableArea: ARRAY[1..numberOfDepartments] OF REAL;
    usingPersons: ARRAY[1..numberOfDepartments] OF REAL;
    m2perPerson: ARRAY[1..numberOfDepartments] OF REAL;
```

(25) Aufgabe 4 – Quicksort

Eine alternative Quicksort-Implementierung verwendet nicht die `Swap` Prozedur, sondern folgende Idee: wenn das Trennelement aus der Teilfoge herausgenommen wird, entsteht ein freies Feld an der ersten Stelle der Teilfolge. Dieses Feld kann dazu verwendet werden, um das erste Element aufzunehmen, das kleiner als das Trennelement ist. Es wird von rechts beginnend gesucht. Danach wird im rechten Teil des Feldes ein Platz frei, und dieser kann wiederum ein Element aufnehmen, das größer als das Trennelement ist, usw. Am Ende wird das Trennelement plaziert. Der Algorithmus sieht wie folgt aus:

```
PROCEDURE quicksort(l, r :INTEGER);
VAR i, j , x : INTEGER; loop : BOOLEAN;
BEGIN
    IF l < r THEN BEGIN
        i := l; j := r; loop := TRUE; x := f[i]; (* Trennelement rausnehmen *)
        WHILE loop DO BEGIN
            WHILE i < j AND x < f[j] DO Dec(j);        (* kleines E. suchen *)
            IF i < j THEN BEGIN
                f[i] := f[j]; Inc(i);              (* kleines E. nach links *)
                WHILE i < j AND f[i] < x DO Inc(i);    (* grosses E. suchen *)
                IF i < j THEN
                    f[j] := f[i]; Dec(j);         (* grosses E. nach rechts *)
                ELSE
                    f[j] := x; loop := FALSE;     (* Trennelement plazieren *)
            END ELSE
                f[i] := x; loop := FALSE;         (* Trennelement plazieren *)
        END;
        quicksort(l,i-1); quicksort(j+1,r);
    END;
END;
```

(15) **a)** Verwenden Sie nun diesen alternativen Quicksort, um fogendes Feld **aufsteigend** zu sortieren. **Geben Sie jeden Zwischenschritt an**, in der Tabelle ist ausreichend Platz. Achten Sie darauf, daß **in jeder Zeile nur Teilfolgen derselben Rekursionstiefe** stehen. **Kennzeichnen Sie deutlich die Grenzen der Folgen und die plazierten Trennelemente!** 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
59	**82**	**35**	**49**	**38**	**70**	**43**	**42**	**55**	**77**	**74**	**25**	**53**	**64**	**58**

(10) **b)** Ändern Sie nun den obigen Algorithmus so, daß er ***absteigend*** statt aufsteigend sortiert und das ***letzte*** Element der jeweiligen Teilfolge als Trennelement verwendet.