Demoklausur Allgemeine Informatik II

Dies ist eine Sammlung von 8 Aufgaben, die der Klausur am 9. Juli nahekommen.

Verzeichnis der Aufgaben

Aufgabe 1 (Rekursion)

Die Folge von Mallow $\{x_n\}$ für $n \in \mathbb{N}$ ist folgendermaßen definiert:

$$
\begin{aligned}
x_1 &= 1 \\
x_2 &= 1 \\
x_n &= x_{(x_{n-2})} + x_{(n-x_{(n-2)})}
\end{aligned}
$$

(a) Welchen Wert hat x_3?

(b) Schreiben Sie eine rekursive Prozedur, die x_n für ein gegebenes n berechnet. Dabei sollten Sie sicherstellen, daß n im zulässigen Bereich liegt.

Aufgabe 2 (Aufbau dynamischer Datenstrukturen)

Gegeben sei folgende dynamische Datenstruktur und dazu passender Konstruktor:

```
TYPE
   Pedigree = POINTER TO PedigreeRec;
   PedigreeRec =
      RECORD
         name: ARRAY 80 OF CHAR;
         father, mother: Pedigree;
      END;

PROCEDURE Create(VAR pedigree: Pedigree;
                 name: ARRAY OF CHAR;
                 father, mother: Pedigree);
BEGIN
   NEW(pedigree); COPY(name, pedigree.name);
   pedigree.father := father; pedigree.mother := mother;
END Create;
```

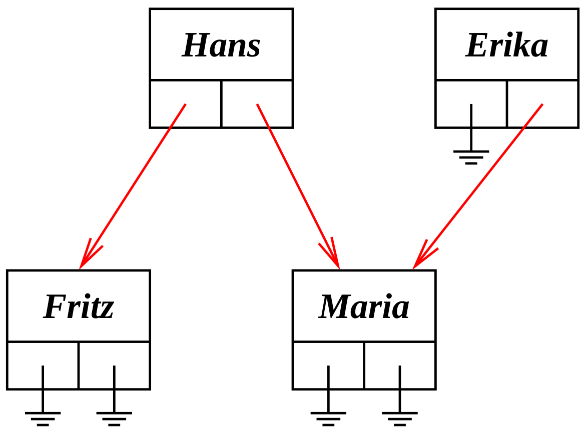
Zeichen Sie bei den Teilaufgaben (a) und (b) jeweils auf, wie Datenstruktur nach der Ausführung aller Anweisungen aussieht. Die Teilaufgaben sind separat (und nicht etwa aufeinander aufbauend) zu betrachten. Für alle Teilaufgaben gelten ferner folgende Variablenvereinbarungen:

```
VAR erika, fritz, hans, maria: Pedigree;
```

(a)
```
Create(hans, "Hans", NIL, NIL);
Create(maria, "Maria", NIL, NIL);
Create(fritz, "Fritz", hans, maria);
```

(b)
```
Create(maria, "Maria", NIL, NIL);
Create(erika, "Erika", NIL, maria);
Create(fritz, "Fritz", NIL, maria);
Create(hans, "Hans", NIL, erika);
```

(c) Schreiben Sie die Anweisungen auf, die zu folgender Datenstruktur führen:



Aufgabe 3 (Hash-Funktionen)

(a) Kreuzen Sie an, welche Eigenschaften für eine Hash-Funktion wichtig sind und auch sinnvollerweise erwartet werden können:

Linearität	☐
Differenzierbarkeit	☐
Effiziente Berechenbarkeit	☐
Nicht-Determinismus	☐
Bijektivität	☐
Injektivität	☐
Surjektivität	☐

(b) Gegeben sei folgende Datenstruktur für ein deutsches Auto-Kennzeichen:

```
TYPE
   Kennzeichen =
      RECORD
         kreis: ARRAY 4 OF CHAR; (* 1..3 Grossbuchstaben *)
         gruppe: ARRAY 3 OF CHAR; (* 1..2 Grossbuchstaben *)
         nummer: INTEGER; (* 1..9999 *)
      END;
```

Schreiben Sie eine geeignete Hash-Funktion in Oberon, die ein Kennzeichen in einen Hash-Wert aus dem Bereich $[0, 1023]$ abbildet.

Aufgabe 4 (Auswahl einer geeigneten dynamischen Datenstruktur)

Für Datensätze, die aus einer ganzen Zahl (dient als eindeutiger Schlüssel) und einem Text (mit weniger als 64 Zeichen) bestehen, wird eine Datenstruktur gesucht, die die folgenden wichtigen Operationen möglichst effizient umsetzt:

- Einfügen eines Datensatzes,
- Ausgabe aller Datensätze (aufsteigend sortiert nach dem Schlüssel),
- Finden von Datensätzen, deren Schlüssel in ein vorgegebenes Intervall von möglichen Schlüsselwerten fällt.

Es ist dabei davon auszugehen, daß neue Datensätze nicht in sortierter Form eintreffen, sondern in ihrer angenommenen Zufälligkeit unabhängig voneinander sind.

(a) Welche (in der Vorlesung vorgestellte) dynamische Datenstruktur ist für diese Problemstellung zu empfehlen? Geben Sie dazu bitte eine kurze Begründung.

(b) Geben Sie bitte die Typ-Deklarationen in Oberon an, um die in der Teilaufgabe (a) vorgeschlagene Datenstruktur umzusetzen.

(c) Schreiben Sie für Ihre Datenstruktur aus Teilaufgabe (b) eine Prozedur **Find**, die zwei Schlüssel erhält, die ein geschlossenes Intervall bilden und — sofern möglich — einen beliebigen Datensatz aus der Datenstruktur mit einem Schlüssel aus dem genannten Bereich zurückliefert. Abhängig vom Erfolg soll ein entsprechender **BOOLEAN**-Wert zurückgegeben werden.

Aufgabe 5 (Rekursiver Algorithmus mit linearen Listen)

Gegeben sei folgendes Oberon-Modul:

```
MODULE Fun;
   IMPORT Write;
   TYPE
      Number = POINTER TO NumberRec;
      NumberRec =
         RECORD
            i: INTEGER; flag: BOOLEAN; next: Number;
         END;

   PROCEDURE HaveFun(numbers: Number; i: INTEGER);
      VAR number: Number;

      PROCEDURE MoreFun(number: Number; j: INTEGER);
      BEGIN
         IF number # NIL THEN
            IF j = 0 THEN
               number.flag := FALSE; j := i;
            END;
            IF (i = 2) & number.flag THEN
               Write.Int(number.i, 1); Write.Ln;
            END;
            MoreFun(number.next, j-1);
         END;
      END MoreFun;

   BEGIN (* HaveFun *)
      NEW(number); number.i := i; number.flag := TRUE;
      number.next := numbers; numbers := number;
      IF i > 1 THEN
         MoreFun(number, i); HaveFun(number, i-1);
      END;
   END HaveFun;

BEGIN HaveFun(NIL, 5);
END Fun.
```

Hinweis: Es handelt sich dabei um eine etwas obskure Fassung eines sehr bekannten Algorithmus aus der Mathematik. Alle Zahlen, die ausgegeben werden, haben eine wichtige gemeinsame Eigenschaft.

(a) Was gibt das Programm aus?

(b) Sei n der zweite Parameter von *HaveFun* im Initialisierungsteil (im obigen Beispiel ist das 5). Beschreiben Sie kurz, was das Programm im generellen Fall für ein beliebiges positives ganzahliges n ausgibt.

(c) Geben Sie die Komplexität des Programmes in Abhängigkeit von n an.

Aufgabe 6 (Sortierte Binärbäume)

(a) Fügen Sie die nachstehenden Comicfiguren in der angegebenen Reihenfolge in einen sortierten Binärbaum ein (sortiert nach Name) und zeichnen Sie diesen Baum:
Garfield, Donald, Alf, Mickey, E.T., Asterix, Obelix, Tick, Trick, Track, Baloo

(b) Traversieren Sie den in Teilaufgabe (a) erstellten Baum in **preorder, inorder** und **postorder** und schreiben Sie die Ausgabe auf.

(c) Ist der Baum der Höhe nach ausgeglichen? Ist er nach Gewicht ausgeglichen?

(d) Entfernen Sie **Donald** aus dem Baum und zeichnen Sie den Baum erneut.

Aufgabe 7 (Modulkonzept in Oberon)

Folgende Quelltexte in Oberon seien gegeben:

<table>
<tr><th>Dateiname</th><th>Programmtext</th></tr>
<tr><td><code>A.od</code></td><td><pre>
DEFINITION A;
   TYPE
      T1 = POINTER TO T1Rec;
      T1Rec = RECORD a: INTEGER END;
   PROCEDURE P(t1: T1);
END A.
</pre></td></tr>
<tr><td><code>A.om</code></td><td><pre>
MODULE A;
   TYPE
      T1 = POINTER TO T1Rec;
      T1Rec = RECORD a: INTEGER; b: INTEGER END;
   PROCEDURE P(t1: T1);
   BEGIN (* ... *) END P;
END A.
</pre></td></tr>
<tr><td><code>B.od</code></td><td><pre>
DEFINITION B;
   IMPORT A;
   TYPE
      T2 = POINTER TO T2Rec;
      T2Rec = RECORD (A.T1Rec) c: INTEGER END;
   PROCEDURE Q(t2: T2);
END B.
</pre></td></tr>
<tr><td><code>B.om</code></td><td><pre>
MODULE B;
   IMPORT A;
   TYPE
      T2 = POINTER TO T2Rec;
      T2Rec = RECORD (A.T1Rec)
         c: INTEGER; d: INTEGER END;
   PROCEDURE Q(t2: T2);
   BEGIN (* ... *) END Q;
END B.
</pre></td></tr>
</table>

(a) Geben Sie eine Reihenfolge an, in der die genannten Quellen übersetzt werden können, ohne daß es zu Fehlermeldungen kommt.

(b) Gibt es auch noch eine weitere Reihenfolge? Falls ja, geben Sie sie bitte an.

Es sei folgender Programmtext eines weiteren Moduls gegeben:

```
IMPORT A, B;

PROCEDURE X;
   VAR
      t2: B.T2;
BEGIN
   NEW(t2); A.P(t2); B.Q(t2);
END X;
```

(c) Auf welche Felder von dem RECORD hinter *t2* kann die Prozedur *X* zugreifen?

(d) Auf welche Felder von dem RECORD hinter *t2* kann die Prozedur *A.P* zugreifen?

(e) Auf welche Felder von dem RECORD hinter *t2* kann die Prozedur *B.Q* zugreifen?

Aufgabe 8 (Typerweiterungen)

Gegeben seien folgende Typdefinitionen:

```
TYPE
   T1 = POINTER TO T1Rec; T1Rec = RECORD t1: INTEGER END;
   T2 = POINTER TO T2Rec; T2Rec = RECORD (T1Rec) t2: INTEGER END;
   T3 = POINTER TO T3Rec; T3Rec = RECORD (T2Rec) t3: INTEGER END;
   T4 = POINTER TO T4Rec; T4Rec = RECORD (T2Rec) t4: INTEGER END;
   T5 = POINTER TO T5Rec; T5Rec = RECORD (T1Rec) t5: INTEGER END;
```

(a) Zeichnen Sie ein Diagramm, das die Typ-Hierarchie widerspiegelt.

(b) Bitte kreuzen Sie jeweils das Zutreffende an:

Aussage	Korrekt	Falsch
T3 ist eine Erweiterung von *T1*		
T4 ist eine Erweiterung von *T2*		
T4 ist eine Erweiterung von *T3*		
T5 ist eine direkte Erweiterung von *T1*		

Hinzu kommen die folgenden Deklarationen:

```
VAR t1: T1; t2: T2; t3: T3; t4: T4; t5: T5;

PROCEDURE Assign(VAR ptr: T2; t2: T2);
BEGIN
   ptr := t2;
END Assign;
```

Geben Sie bei folgenden Teilaufgaben jeweils an, ob der jeweilige Programmtext vom Compiler akzeptiert wird und ob er zur Laufzeit fehlerfrei durchläuft. Begründen Sie Ihre Angaben jeweils durch die Angaben der statischen und dynamischen Typen von *ptr* und *t2* innerhalb von *Assign*.

(c) `NEW(t1); NEW(t5); Assign(t1, t5);`

(d) `NEW(t3); NEW(t4); Assign(t3, t4);`

(e) `NEW(t2); NEW(t4); Assign(t2, t4);`