Klausur zu Programmieren II

SS 92

27.06.92

Name _____

Vorname _____

Matrikel-# _____

Aufgabe	Punkte	
1	18	
2		
a	4	
b	4	
c	4	
d	4	
3		
a	6	
b	8	
4		
a	8	
b	8	
5	18	
6		
a	9	
b	9	
Σ	100	

Aufgabe 1

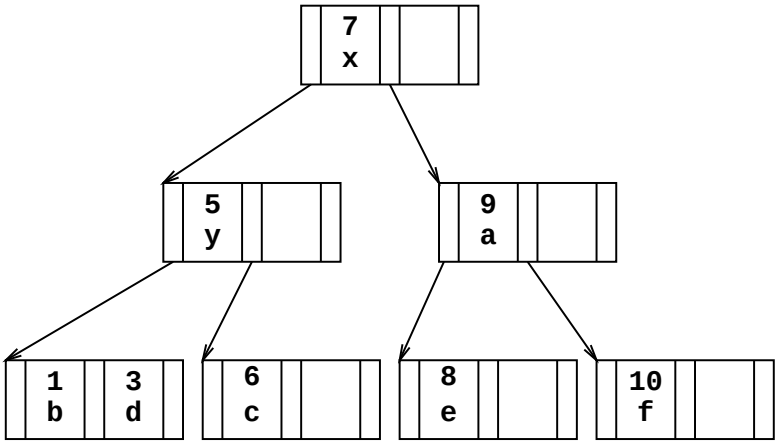
Gegeben ist folgende Schnittstelle:

```
DEFINITION MODULE Zaehler;

   TYPE Zaehler;

   PROCEDURE Init(VAR zaehler: Zaehler; inkrement: CARDINAL);
      (* Initialisierung eines neuen Zaehlers,
         der zu Beginn den Wert 0 besitzt
      *)

   PROCEDURE Wert(zaehler: Zaehler) : CARDINAL;
      (* liefert den aktuellen Wert des Zaehlers zurueck
         und erhoeht dann den Wert des Zaehlers um den Inkrement-Wert,
         der bei Init mitgegeben worden ist
      *)

   PROCEDURE Wegwerfen(VAR zaehler: Zaehler);
      (* Elimination eines nicht mehr benoetigten Zaehlers *)

END Zaehler.
```

Schreiben Sie ein zugehöriges Implementation-Modul, das sich gemäß der Spezifikation in der Schnittstelle verhält.

Aufgabe 2

Gegeben sei der folgende B-Baum mit $k = 1$, wobei der Schlüssel vom Typ **INTEGER** und die Satzinformation vom Typ **CHAR** ist. Die folgenden Teilaufgaben beziehen sich **jeweils** auf folgende Ausgangssituation:



Zeichnen Sie bei jeder der folgenden Teilaufgaben den Baum im Endzustand. Sollte die angegebene Operation zu Zwischenschritten führen (Splitten oder Zusammenfassen), so sind diese Schritte kurz zu skizzieren.

(*a*) Hinzufügen von [12, z].
(*b*) Hinzufügen von [2, t].
(*c*) Löschen des Datensatzes mit dem Schlüssel 5.
(*d*) Löschen des Datensatzes mit dem Schlüssel 9.

☞ **Wichtig:** gehen Sie bei jeder Teilaufgabe immer von der oben dargestellten Ausgangssituation aus. Die Operationen sind also keinesfalls hintereinander auszuführen!

Aufgabe 3

Gegeben sei eine bereits teilweise gefüllte Hash-Tabelle:

```
CONST
   hashtabGroesse = 11;
TYPE
   HashIndex = [0..hashtabGroesse-1];
   Name = ARRAY [0..15] OF CHAR;
   Eintrag =
      RECORD
         name: Name;
         alter: CARDINAL;
      END;
VAR
   hashtab: ARRAY HashIndex OF Eintrag;
```

0	Kurt, 28
1	
2	Alexander, 22
3	
4	
5	
6	
7	
8	
9	
10	Ulrike, 23

(*a*) Wenn folgende Sätze mit den angegebenen Hash-Werten hintereinander eingetragen werden sollten, wie würde die Tabelle anschließend bei quadratischer Sondierung aussehen?

Anton, 33	2
Franz, 21	3
Anne, 27	10
Elisabeth, 18	2

(*b*) Schreiben Sie eine Hash-Funktion, die einen Namen in einen Hash-Wert abbildet.

☞ Sie sollten davon ausgehen, daß Namen nur aus Buchstaben bestehen, wobei Klein- und Großbuchstaben gleich behandelt werden dürfen, d.h. bei "Anton" und "anton" sollte der gleiche Hash-Wert zurückgeliefert werden.

☞ Bei Permutationen sollten (in der Regel) unterschiedliche Hash-Werte geliefert werden, d.h. "axel" und "axle" sollten unterschiedliche Hash-Werte haben.

☞ Die Hash-Funktion sollte auch dann noch gut streuen, wenn man die Größe der Hash-Tabelle auf hohe Werte setzt.

Aufgabe 4

Sie sind im Besitz recht vieler Bücher und leihen Ihre guten Stücke auch gerne aus. Beim Entwurf einer Datenstruktur zur Repräsentierung Ihrer Bücher und Ausleiher sind folgende Punkte zu berücksichtigen:

- Sie benötigen eine Liste aller Bücher, die Sie besitzen. Neben Autor und Titel haben Sie für jedes Buch auch eine eindeutige Signatur vergeben, die aus einer natürlichen Zahl besteht. Um mit der Signatur als Schlüssel rasch das zugehörige Buch ermitteln zu können, beschließen Sie, als Organisation einen sortierten binären Baum zu verwenden.
- Alle Personen, die mindestens ein Buch von Ihnen ausgeliehen haben, sind in einer nach dem Namen sortierten linearen Liste zu verwalten.
- Zu jedem Buch gehört der Hinweis, ob und ggf. von wem das Buch ausgeliehen wurde.
- Für jeden Ausleiher benötigen Sie eine Liste der von ihm ausgeliehenen Bücher.

(*a*) Schreiben Sie in Modula-2 alle notwendigen Typ- und Variablendeklarationen für die obige Datenstruktur auf.

(*b*) Angenommen, Sie haben folgende Bücher und Ausleiher. Stellen Sie graphisch dar, wie Ihre Datenstruktur bei diesem Beispiel aussehen könnte. Der Graph sollte aus Kästchen für die Records und Pfeilen für die Zeiger bestehen.

Bücher	Signatur	Autor	Titel
	1368	Niklaus Wirth	Programming in Modula-2
	1821	James Gosling	The NeWS Book
	2418	Martin Reiser	Programming in Oberon
	1245	Martin Reiser	The Oberon System
	1523	Alfred Aho	Principles of Compiler Design

Ausleiher	Name	ausgeliehene Bücher
	Roswitha	2418
	Andreas	1245, 1523
	Martin	1821

☞ Bei der Beschriftung der Kästchen für die Records reicht die Angabe des Schlüssels.

☞ Wenn von einem Record mehrere Zeiger ausgehen, sollten sie unterscheidbar sein. Die Zeiger sollten also entweder ebenfalls beschriftet sein oder sich durch Farben oder die Art der Linie (z.B. durchgehend, gestrichelt oder gepunktet) unterscheiden. Bei der Verwendung von Farben oder unterschiedlichen Linien sollte eine kleine Legende mit angegeben werden.

Aufgabe 5

Gegeben sei folgende Datenstruktur:

```
TYPE
   Schluessel = INTEGER;
   Info = ARRAY [0..19] OF CHAR;
   Knoten = POINTER TO KnotenRec;
   KnotenRec =
      RECORD
         schluessel: Schluessel;
         info: Info;
         links, rechts: Knoten; (* sortierter Binaerbaum *)
         naechster: Knoten; (* sortierte lineare Liste aller Knoten *)
      END;
   Baum = POINTER TO BaumRec;
   BaumRec =
      RECORD
         wurzel: Knoten; (* Wurzel des sortierten Binaerbaums *)
         liste: Knoten; (* Zeiger auf das 1. Element der linearen Liste *)
      END;
```

Angenommen, der Baum wird zuerst Schritt für Schritt gefüllt und erst später wird die sortierte lineare Liste benötigt, um alle Elemente in sortierter Reihenfolge zu durchlaufen. Dann ist es angebracht, die lineare Liste erst dann zu erzeugen, wenn alle Knoten eingetragen sind.

Implementieren Sie folgende Prozedur, die für einen gegebenen sortierten Binärbaum die sortierte lineare Liste erzeugt, d.h. für jeden Knoten die Komponente *naechster* setzt und *baum^.liste* auf das erste Element der Liste zeigen läßt:

```
PROCEDURE BestimmeSortierteListe(baum: Baum);
```

Aufgabe 6

Gegeben sei folgende lineare Liste:

```
TYPE
   Liste = POINTER TO ListeRec;
   ListeRec =
      RECORD
         anfang: Element; (* zeigt auf das 1. Element der Liste *)
         ende: Element; (* zeigt auf das letzte Element der Liste *)
         (* wenn die Liste leer ist, sind anfang und ende = NIL *)
      END;
   Info = ARRAY [0..19] OF CHAR;
   Element = POINTER TO ElementRec;
   ElementRec =
      RECORD
         info: Info;
         naechster: Element; (* Zeiger auf das naechste Element *)
            (* wenn es das letzte Element ist, dann ist naechster = NIL *)
      END;
```

(*a*) Schreiben Sie eine Prozedur, die ein neues Element an das Ende der Liste hängt. Bedenken Sie dabei auch den Fall, daß die Liste leer sein könnte, d.h. *liste* ist zwar ungleich **NIL**, aber *liste^.anfang* und *liste^.ende* sind gleich **NIL**.

```
PROCEDURE Anhaengen(liste: Liste; info: Info);
   (* lege ein neues Element an mit der gegebenen info-Komponente
      und haenge es an das Ende der Liste 'liste'
   *)
```

(*b*) Schreiben Sie eine Prozedur, die alle *info*-Komponenten einer Liste ausgibt. Verwenden Sie dazu geeignete Prozeduren aus *InOut*. Jede *info*-Komponente sollte dabei auf einer eigenen Zeile erscheinen.

```
PROCEDURE Ausgabe(liste: Liste);
```

☞ Solange Sie Prozeduren aus *Storage* oder *InOut* verwenden, brauchen Sie nicht die zugehörigen **IMPORT**-Anweisungen mit anzugeben.