Klausur zu Programmieren II

SS 95

15. Juli 1995

Name ______________________________

Vorname ______________________________

Matrikel-Nr. ______________________________

Aufgabe		**Punkte**	**Bewertung**
1	a)	3	
	b)	3	
	c)	3	
	d)	3	
2	a)	2	
	b)	2	
	c)	4	
3		10	
4		6	
5		6	
6	a1)	3	
	a2)	3	
	b1)	6	
	b2)	6	
7	a)	8	
	b1)	2	
	b2)	6	
8	a)	4	
	b)	6	
Summe		**86**	

☞ Bearbeitungszeit: 180 Min.

Aufgabe 1

a) Was versteht man unter **Information Hiding** (Idee, Vorteil)?

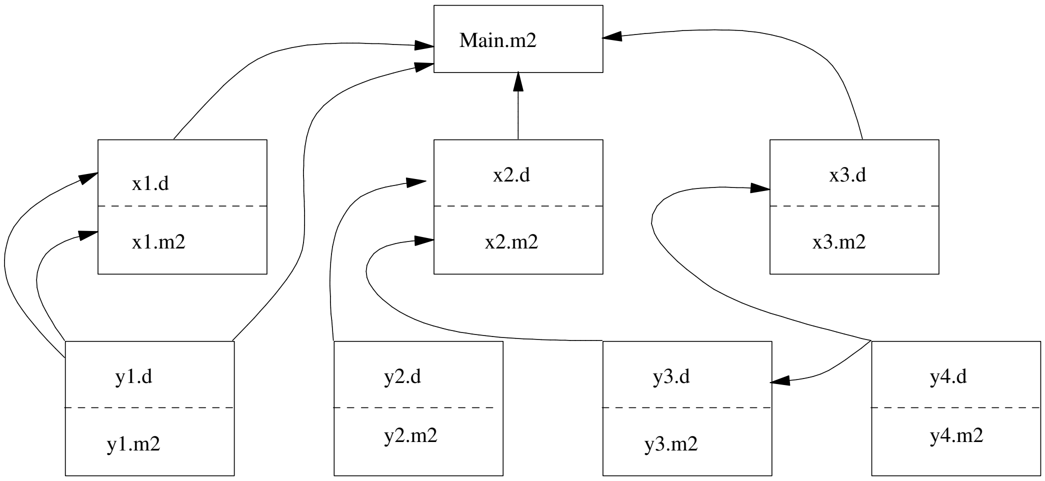
b) Beschreiben Sie kurz die Idee der **Abstrakten Datentypen** und zeichnen Sie eine Realisierungsmöglichkeit in Modula-2 auf!

c) Was versteht man in der UNIX-Welt unter **Filter**?

d) Nennen Sie wenigstens einen plausiblen Grund für die Trennung der Ausgabe eines Programms unter UNIX in **stdin** und **stderr**!

Aufgabe 2

Gegeben sei folgende Programmstruktur:



Anm.: Die Pfeile symbolisieren die IMPORT-Beziehung!

a) Geben Sie eine mögliche Übersetzungsreihenfolge an!

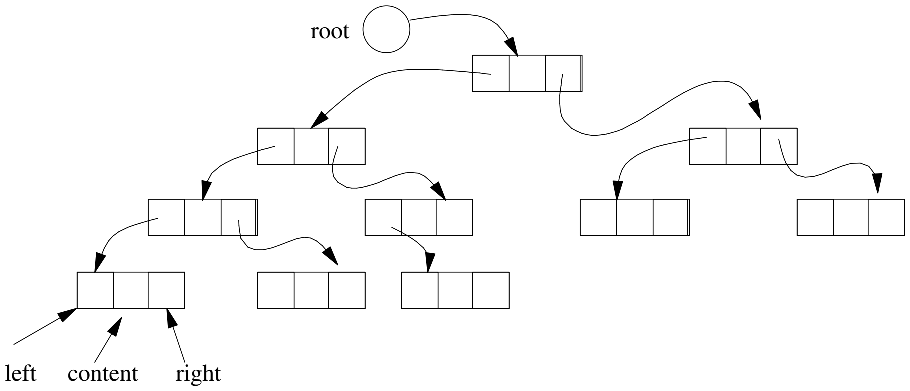
b) Was enthält das sogenannte *makefile* (nur das Prinzipielle, nicht den syntaktischen Inhalt)?

c) Welchen Vorteil bietet die Verwendung von *make*?

Aufgabe 3

Ein nach inorder sortierter Binär-Baum sei wie folgt vereinbart:

```
TYPE
    Tree = POINTER TO Node;
    Node = RECORD
             content: CARDINAL;
             left,right: Tree;
           END;
```



Die Höhe eines Knotens ist die Länge des Weges von der Wurzel zu diesem Knoten (die Wurzel hat also die Höhe 0).

Aufgabe: Schreiben Sie eine rekursive Prozedur

```
PROCEDURE levelPrint(root: Tree; level:CARDINAL)
```

die den Inhalt (*content*) aller Knoten mit Höhe = *level* absteigend nach *stdout* ausgibt (sie können davon ausgehen, daß jeder Pointer einen definierten Wert hat)

Aufgabe 4

Für einen Rechner mit Dateiblockgröße 1kByte will jemand eine B-Baum-Verwaltung realisieren. Die abzulegenden Daten entstammen folgender Typ-Vereinbarung:

```
TYPE
    DataRec = RECORD
                  key: CARDINAL;
                  name: ARRAY[0..19] OF CHAR;
                  married: BOOLEAN
          END
```

Bestimmen Sie für diesen "Jemand" den Typ **k** des B-Baumes - sollten Sie dazu Annahmen zugrundelegen, geben Sie diese unbedingt an!

Aufgabe 5

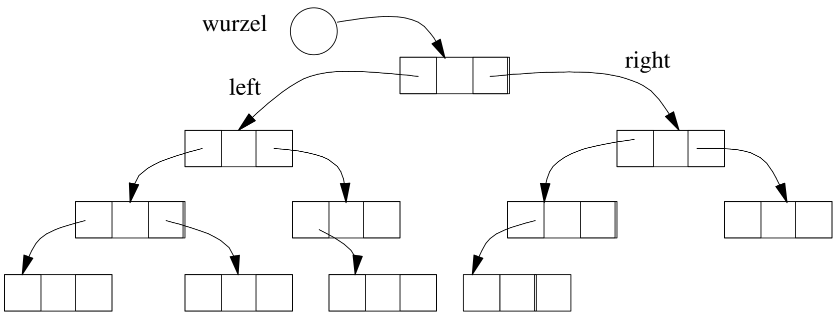
Gegeben sei folgender Auszug aus einem Modul:

```
TYPE
    Tree = POINTER TO NODE;
    Node = RECORD
        content: CARDINAL;
        left, right: Tree
           END;

    Direction = (l,r);
CONST   MaxLevel = 3;

PROCEDURE oha(   VAR root: Tree;
        dir: Direction;
        level: CARDINAL );
BEGIN
    IF root = NIL THEN RETURN END;
    IF (level < MaxLevel)
         & (root^.left <> NIL)
         & (root^.right <> NIL) THEN
       IF dir = l THEN
        oha(root^.right,r,level+1)
       ELSE
        oha(root^.left,l,level+1)
       END
    ELSE
       IF dir = l THEN
        root := root^.right
       ELSE
        root := root^.left
       END
    END
END oha;
```

Für folgenden Baum wird die Prozedur mit `oha(wurzel,l,1) aufgerufen:`



(Alle "Pointer-Kästchen", von denen kein Pfeil weggeht, sind mit dem NIL-Pointer gesetzt)

Was bewirkt dieser Aufruf:

- □ Er verändert am Baum NICHTS (Begründung)!

- □ Er verändert die Struktur des Baumes (Bitte in Skizze eintragen WIE! Was ist in der Prozedur dafür verantwortlich?)

Aufgabe 6

Gegeben sei folgende Typ-Vereinbarung:

```
TYPE Element = ARRAY [0..MAX] OF CHAR
```

a) Geben Sie eine dynamische Datenstruktur an für die Repräsentation einer Menge von Elementen vom Typ `Element`, und zwar

a1) einmal in grafischer Darstellung für die konkrete Menge M = {"matthias", "werner", "peter", "andreas", "franz"}

a2) und zum anderen als Typ-Definition(en) in Modula-2

b) Geben Sie eine dynamische Datenstruktur für die Repräsentation einer Menge von Mengen von Elementen vom Typ `Element`, so daß jedes elementare Element (vom Typ `Element`) nur einmal gespeichert wird, und zwar

b1) einmal in grafischer Darstellung für die konkrete Menge

```
K = { {"franz", "inge", "ralf", "ute", "joerg"},
      {"werner", "nicole"},
      {"matthias", "ralf", "ute"},
      {"joerg", "nicole"}
    }
```

b2) und zum anderen als Typ-Definition(en) in Modula-2

Aufgabe 7

Irgend jemand möchte ein Programm, das die Gewichtsausgeglichenheit eines sortierten Binärbaumes prüfen soll. **Sie** sollen ihm dazu eine Hilfsprozedur liefern, die zu einem Baum, gegeben durch einen Zeiger auf dessen Wurzel, die Zahl der Knoten (incl. Wurzel) liefert. Da er ein Fan der Rekursion ist, verlangt er von Ihnen auch die **Verwendung der Rekursion**.

Folgende Typvereinbarungen legt er zugrunde:

```
TYPE
    Tree = POINTER TO Node;
    Node = RECORD
             left,right: Tree
           END;
```

a) Er will zunächst eine Funktionsprozedur mit folgender Vereinbarung:

```
PROCEDURE count1(r: Tree): CARDINAL;
```

b) Aus welchem Grund auch immer will er von Ihnen das gleiche nocheinmal, allerdings nicht als Funktionsprozedur (sondern als "normale" Prozedur), sie dürfen aber auch keine globalen Variablen verwenden.

b1) Geben Sie den Prozedurkopf für `count2`!

b2) Formulieren Sie die Prozedur aus!

Aufgabe 8

Gegeben sei folgendes Programm:

```
MODULE Passage;
    IMPORT InOut;
    IMPORT Storage;
    IMPORT SYSTEM;

    TYPE NodePointer = POINTER TO Node;
        Node = RECORD
                data: CHAR;
                next: NodePointer;
                  END;

    VAR currentPtr: NodePointer;

    PROCEDURE change(tempPtr: NodePointer);
    BEGIN
        tempPtr^.data := "C";
        tempPtr := tempPtr^.next;
        tempPtr^.data := "D";
    END change;
BEGIN
    Storage.ALLOCATE(currentPtr,SYSTEM.TSIZE(Node));
    currentPtr^.data := "A";
    Storage.ALLOCATE(currentPtr^.next,SYSTEM.TSIZE(Node));
    currentPtr^.next^.data := "B";
    InOut.Write(currentPtr^.data);
    InOut.Write(currentPtr^.next^.data);

    change(currentPtr);

    InOut.Write(currentPtr^.data);
    InOut.Write(currentPtr^.next^.data);
END Passage.
```

a) Was gibt dieses Programm aus?

b) Was verändert sich in der Ausgabe und im Programmverhalten, wenn die Prozedur *change* wie folgt verändert wird:

```
PROCEDURE change(VAR tempPtr: NodePointer)
```