Klausur Programmieren II

SS 96

3. Juli 1996

Name ______________________________

Vorname ______________________________

Matrikel-Nr. ______________________________

Aufgabe		Punkte	Bewertung
1		8	
2		10	
3	a)	3	
	b)	3	
	c)	3	
	d)	5	
4		6	
5		10	
6		10	
7		12	
Summe		**70**	

Aufgabe 1

Durch folgende Definitionen sei ein Binärbaum gegeben:

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

Schreiben Sie eine **rekursive Funktions-Prozedur** *summe*, die die Summe der Knoteninhalte (Record-Komponente *number*) zurückliefert.

```
PROCEDURE summe(root: Tree):CARDINAL;
(* ab hier ausfüllen *)
```

Aufgabe 2

Gegeben sei eine einfach verkettete Liste:

```
TYPE
        Liste = POINTER TO ListElement;
        ListElement = RECORD
                          next: Liste;
                          content: CARDINAL
                      END;
```

Schreiben Sie eine Prozedur, die alle Elemente mit `content = 0` verkettet; die gegebene Verkettung (Komponente *next*) soll dazu benutzt werden. Das letzte Element der Liste enthält in der Komponente `next` den Wert `NIL`.

```
PROCEDURE newList(VAR head: Liste);
(*ab hier auszufüllen: *)
```

Aufgabe 3

(*a*)) Modula-2 kennt Prozedur-Typen. Nennen Sie einen Aspekt, bei dem diese sehr sinnvoll eingesetzt werden können!

(*b*)) Modula-2 kennt *hidden types*. Nennen Sie einen Aspekt, bei dem diese sehr sinnvoll eingesetzt werden können!

Zur Übersetzung größerer Programme kann man mit *mmm* ein sog. *Makefile* erzeugen und mit dem Kommando *make* das ausführbare Programm generieren.

(*c*)) Nennen Sie <u>einen</u> grundsätzlichen Aspekt, der im *Makefile* festgehalten wird.

(*d*)) Nennen Sie zwei Aufgaben, die *make* automatisch erledigt.

Aufgabe 4

Gegeben sei ein nach *inorder*-sortierter Binärbaum:

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

Schreiben Sie eine Prozedur, die die Knotencontente **absteigend** sortiert ausgibt.

```
PROCEDURE printDesc(wurzel: Tree);
(*ab hier ausfuellen*)
```

Aufgabe 5

Gegeben sei folgendes Programm:

```
MODULE a5;
FROM InOut IMPORT WriteCard, WriteLn;
FROM Storage IMPORT ALLOCATE;


TYPE
        Liste = POINTER TO ListElement;
        ListElement = RECORD
                        next: Liste;
                        content: CARDINAL
                      END;
VAR
        liste,p: Liste;

CONST
        anz = 10;

PROCEDURE buildList(VAR head: Liste);
        VAR i: CARDINAL;
            p: Liste;
BEGIN
        head := NIL;
        FOR i:=1 TO anz DO
                NEW(p);
                p^.content := i;
                p^.next := head;
                head := p
        END
END buildList;

PROCEDURE printList(p:Liste);
BEGIN
        WHILE p <> NIL DO
                WriteCard(p^.content,6);
                p := p^.next;
        END;
        WriteLn;
END printList;

PROCEDURE whatever(VAR p: Liste; x: CARDINAL):Liste;
        VAR q: Liste;
BEGIN
        IF p=NIL THEN
                RETURN NIL
        ELSE
                IF p^.content = x THEN
                        q:=p;
                        p:=p^.next;
                        RETURN q
                ELSE
                        RETURN whatever(p^.next,x)
                END
        END
END whatever;
```

```
BEGIN
        buildList(liste);
        printList(liste);
        p := whatever(liste,5);
        WriteCard(p^.content,6); WriteLn;
        printList(liste);
END a5.
```

Geben Sie an, was dieses Programm ausgibt!

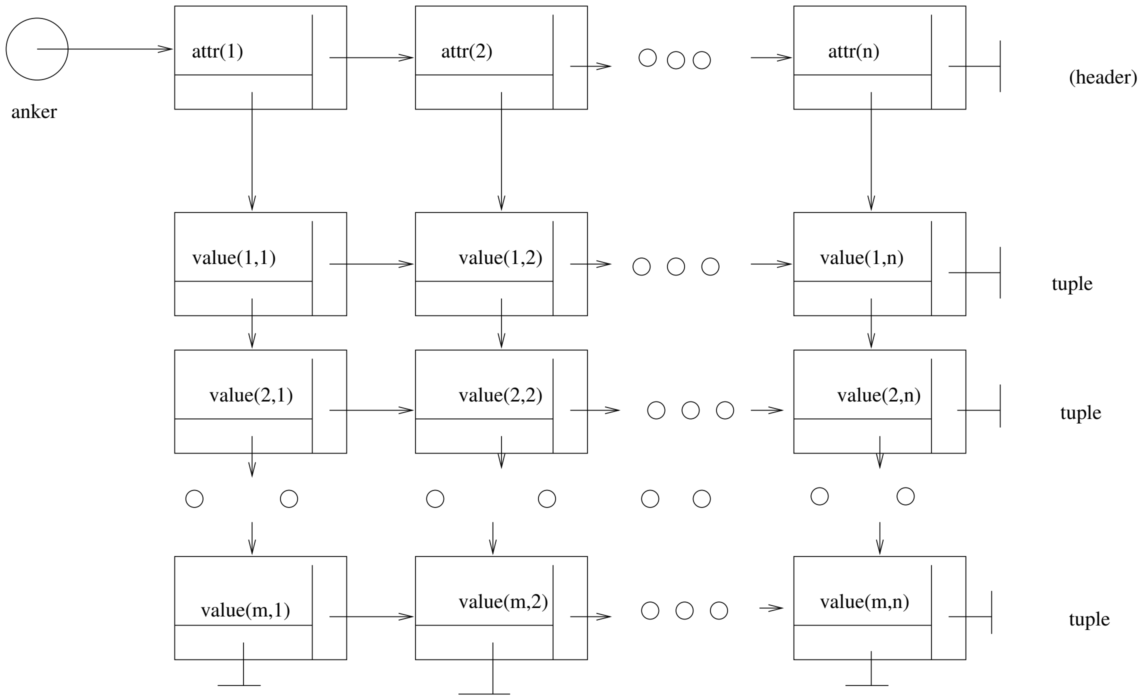
Aufgabe 6

Gegeben ist folgende Grammatik:

```
table = header { tuple }
header = attr { ":"  attr } "12C"
tuple = value { ":" value } "12C"
attr = string
value = string | number
string = {"A" "B" ... "Z" "a" "b" ... "z"}
number = { "0" "1" ... "9" }
```

Anm.: Die in Apostroph gesetzten Symbole sind Terminalsymbole.

Zu einem Programm, das mit der so definierten Grammatik arbeiten soll, sollen Sie nun eine Datenstruktur (Typ-Definitionen) entwickeln, die einen "Satz" dieser Sprache aufnehmen kann. Um die Sache zu erleichtern bzw. klarer zu definieren, sei folgendes Beispiel gegeben:



Der Doppelpunkt in der Definition wird also in der Datenstruktur nicht wiedergegeben, da die Trennung der Objekte *attr* bzw. *value* in der Listenstruktur durch Ablegen in einzelnen Listenelementen erkennbar ist. Die in der Grafik angegebenen Indizes (z.B. *value(1,2)*) sollen nicht als Array-Indizes verstanden werden.

→ nächste Seite

Geben Sie geeignete Typ-Definitionen an, so daß eine Variable *anker* vereinbart werden kann!

Aufgabe 7

Schreiben Sie eine Prozedur *disposeTree*, die als Parameter einen Zeigen auf einen Knoten eines Binärbaumes erhält und diesen wie auch alle Knoten des Teilbaumes, dessen Wurzel der gegebene Knoten ist, via *DISPOSE* freigibt.

Vereinbart sei:

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

```
PROCEDURE disposeTree(root: Tree);
(* ab hier ausfüllen *)
```