Klausur zur

Allgemeinen Informatik II

21. Juli 2001

Prof. Dr. H. Neumann
Dr. K. Murmann

Vorname: ______________________ Name: ______________________

Studienfach: ______________________ Semester: ______________________

Matrikel-Nr.: ______________________

Hinweise

- Lösungen bitte in die Aufgabenblätter schreiben.
- Bereits während der Klausur den Namen auf jedes Lösungsblatt schreiben.
- Zur Lösung der 9 Aufgaben stehen Ihnen 3 Stunden (180 Minuten) Bearbeitungszeit zur Verfügung. Insgesamt können Sie 100 Punkte erreichen.
- Erlaubte Hilfsmittel: Skript, Vorlesungsmitschrift und ein Modula-2-Buch.

  Elektronische Hilfsmittel sind nicht erlaubt!

Aufgabe	1	2	3	4	5	6	7	8	9	$\sum$
erreichbare Punkte	10	10	10	10	10	10	16	10	14	100
erreichte Punkte										
Korrekteur										

1. Aufgabe 10

Antworten Sie auf die folgenden Behauptung einfach mit **richtig** oder **falsch** (je 1 Punkt pro Frage) :

A. Bei Divide-and-Conquer Verfahren wird die Datenmenge jeweils in separate Teilmengen aufgeteilt und anschliessend jede Teilmenge durch rekursiven Aufruf der Methode bearbeitet.

B. Bei Verwendung eines Wächterknotens als Listenende kann eine einfach verkettete Liste auch von hinten nach vorne durchlaufen werden.

C. Zur Implementierung einer Queue (FIFO) wird stets eine dynamische Datenstruktur benötigt.

D. Ein Abstrakter Datentyp ist nur im Definitionsteil eines Modula-2 Programms definiert (und daher abstrakt !).

E. Jeder Binärbaum der Höhe h hat genau 2^h Blätter.

F. Binäre Bäume lassen sich mithilfe eines Stacks (zur Zwischenspeicherung der Resultate) nach Schlüsselelementen durchsuchen.

G. Für einen arithmetischen Ausdruck in Präfix-Notation müssen Klammern als Stukturierungsmittel verwendet werden.

H. Der Sortieraufwand kann für eine feste Datenmenge von der aktuellen Ordnung der Sortierschlüssel abhängen.

I. In einem binären Suchbaum in dem n Zahlen vom Typ `CARDINAL` gespeichert sind, lässt sich das Minimum dieser Zahlen stets mit dem Aufwand $O(log(n))$ berechnen.

J. In einem `IMPLEMENTATION MODULE` sind die Definitionen aus dem `DEFINITION MODULE` gleichen Namens automatisch bekannt, und müßen nicht explizit mittels `IMPORT` sichtbar gemacht werden.

Lösung:

```
A. richtig    B. falsch
C. falsch     D. falsch
E. falsch     F. richtig
G. falsch     H. richtig
I. falsch     J. richtig
```

2. Aufgabe 10

Beantworten Sie die folgenden Fragen mit **ja** oder **nein** ohne Begründung (je 2 Punkte pro Frage) :

A. $f(n) = \log(n^2)$. Gilt $f \in O(log(n))$?

B. $f(n) = n \sin(n) + 1$. Gilt $f \in O(1)$?

C. $f(n) = \sum_{k=1}^{n} k$. Gilt $f \in O(n)$?

D. $f(n) = 10^6 n^6 + 10^{-7} n^7 + 10^{-8} n^8$. Dann ist $f \in O(n^8)$.

E. $f(n) = 10^6 n^{-6} + 10^{-7} n^{-7} + 10^{-8} n^{-8}$. Dann ist $f \in O(n^{-8})$.

Lösung:

A. **ja**

B. **nein**

C. **nein**

D. **ja**

E. **nein**

3. Aufgabe 10

Schreiben Sie eine **iterative** Funktionsprozedur

```
PROCEDURE Threshold(li : LIST; theta : CARDINAL) : BOOLEAN;
```

die den Wert `TRUE` zurückgibt, falls eines der `data`-Felder in der Liste `li` einen Wert $\geq$ `theta` hat. In allen anderen Fällen soll die Funktionsprozedur `Threshold` den Wert `FALSE` liefern (insbesondere auch dann, wenn `li` leer ist).

Lösung:

```
PROCEDURE Threshold(li : LIST; theta : CARDINAL) : BOOLEAN;
BEGIN
   WHILE li <> NIL DO
      IF li^.data >= theta THEN
         RETURN TRUE;
      END;
      li := li^.next;
   END;
   RETURN FALSE;
END Treshold;
```

4. Aufgabe 10

A. Schreiben Sie eine **rekursive** Funktionsprozedur 5

```
PROCEDURE Multiply(a, b : CARDINAL) : CARDINAL;
```

die die Multiplikation von `a` und `b` mittels wiederholter Addition implementiert.

B. Geben Sie eine Handsimulation der rekursiven Funktionsaufrufe für die Multiplikation $5 * 2$ an. 4

C. Werden die Multiplikationen $5 * 2$ und $2 * 5$ mit identischem Aufwand berechnet ? 1

Lösung:

A.

```
PROCEDURE Multiply(a, b : CARDINAL) : CARDINAL;
BEGIN
   IF b = 0 THEN
      RETURN 0;
   ELSE
      RETURN a + Multiply(a, b - 1);
   END;
END Multiply;
```

B.

```
Multiply(5,2) = 5 + Multiply(5,1)
              = 5 + 5 + Mulitply(5,0)
              = 5 + 5 + 0
              = 10
```

C. Nein. Für $5 * 2$ werden 2 Additionen benötigt, für $2 * 5$ dagegen 5.

5. Aufgabe 10

Gegeben sei die folgende Definition eines Prozedurtyps für eine Funktionsprozedur, die eine Eigenschaft einer `CARDINAL`-Zahl prüfen soll :

```
TYPE  PREDICATE = PROCEDURE(CARDINAL) : BOOLEAN;
```

A. Schreiben Sie eine **rekursive** Funktionsprozedur 8

```
PROCEDURE Select(li : LIST; p : PREDICATE) : LIST;
```

die eine neue Liste zurückgibt, die diejenigen Zahlen aus `li` enthält, für die `p` den Wert `TRUE` liefert. Dabei soll `li` unverändert bleiben.

B. Das folgende Programmfragment extrahiert mithilfe der Funktionsprozedur `Select` eine neue Liste `li2`, die genau die ungeraden Zahlen der gegebenen Liste `li1` enthält : 2

```
VAR li1, li2 : LIST;
BEGIN
  li1 := ... (* Konstruktion der Liste li1 *)
  li2 := Select(li1, UngeradeZahl);
  ...
END
```

Schreiben Sie eine Prozedur `UngeradeZahl` !

Lösung:

A. 1. Variante (mit Zerstörung der alten Liste `li`):

```
PROCEDURE Select(li : LIST; p : PREDICATE) : LIST;
BEGIN
   IF li = NIL THEN
      RETURN NIL;
   ELSIF p(li^.data) THEN
      li^.next := Select(li^.next, p);
      RETURN li;
   ELSE
      RETURN Select(li^.next, p);
   END;
END Select;
```

2. Variante (ohne Zerstörung der alten Liste `li`):

```
PROCEDURE Select(li : LIST; p : PREDICATE) : LIST;
   VAR
      node : LINK;
BEGIN
   IF li = NIL THEN
      RETURN NIL;
   ELSIF p(li^.data) THEN
      NEW(node);
      node^.data := li^.data;
      node^.next := Select(li^.next, p);
      RETURN node;
   ELSE
      RETURN Select(li^.next, p);
   END;
END Select;
```

B.

```
PROCEDURE UngeradeZahl(a : CARDINAL) : BOOLEAN;
BEGIN
   RETURN ODD(a); (* oder RETURN a MOD 2 = 1; *)
END UngeradeZahl;
```

6. Aufgabe 10

A. Ein Baum sei durch das folgende Programmfragment definiert : 5

```
VAR t0, t1, t2, t3, t4 : TREE;
BEGIN
  t4 := MakeTreeNode(5, NIL, NIL);
  t3 := MakeTreeNode(17, NIL, NIL);
  t2 := MakeTreeNode(2, NIL, NIL);
  t1 := MakeTreeNode(11, t2, t3);
  t0 := MakeTreeNode(8, t1, t4);
```

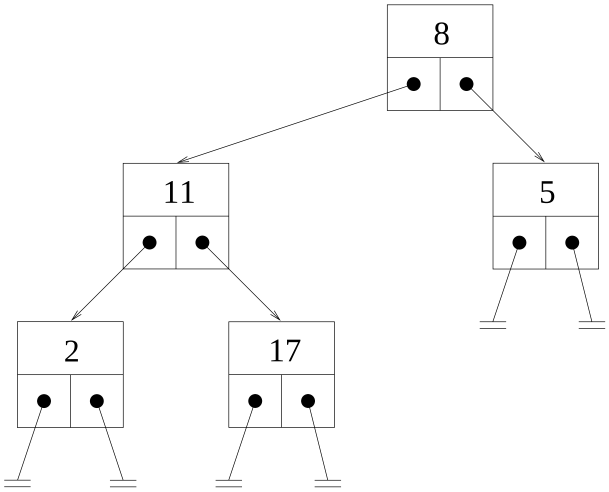
Zeichnen Sie die Struktur des Baumes mit der Wurzel `t0`.

B. Schreiben Sie eine Funktionsprozedur 5

```
PROCEDURE Blaetter(t : TREE) : CARDINAL;
```

die die Anzahl der Blätter des Baums `t` zurückgibt.

Lösung:



A.

B.

```
PROCEDURE Blaetter(t : TREE) : CARDINAL;
BEGIN
   IF t = NIL THEN
      RETURN 0;
   ELSIF (t^.left = NIL) & (t^.right = NIL) THEN
      RETURN 1;
   ELSE
      RETURN Blaetter(t^.left) + Blaetter(t^.right);
   END;
END Blaetter;
```

7. Aufgabe 16

Gegeben sei eine Datenmenge in einem Feld. Diese Datenmenge soll mithilfe des Sortieralgorithmus `BubbleSort` in **aufsteigender** Reihenfolge sortiert werden. Dieser Algorithmus ist gegeben:

```
TYPE FELD = ARRAY [1..8] OF CARDINAL;

PROCEDURE BubbleSort(VAR arr : FELD);
VAR
  i, j, x: CARDINAL;
BEGIN
   FOR i := 2 TO 8 DO
       FOR j := 8 TO i BY -1 DO
           IF arr[j-1] > arr[j] THEN
              x        := arr[j-1];
              arr[j-1] := arr[j];
              arr[j]   := x
           END
       END
   END
END BubbleSort;
```

A. Führen Sie für das folgende Feld von 8 Elementen 10

5	6	20	11	7	13	4	18

eine Handsimulation des Sortieralgorithmus durch. Geben Sie den Inhalt des Feldes am Ende eines jeden Durchlaufs der äußeren `FOR`-Schleife an (in Tabellenform).

B. Erweitern Sie die Prozedur `BubbleSort` so, dass beliebige Prozeduren zum Vergleich der Sortierschlüssel als Parameter übergeben werden können. 6

Schreiben Sie eine Vergleichsprozedur `Decreasing` bei deren Verwendung die Sortierschlüssel im Feld `arr` absteigend sortiert werden.

Lösung:

A.

4	5	6	20	11	7	13	18

4	5	6	7	20	11	13	18

4	5	6	7	11	20	13	18

4	5	6	7	11	13	20	18

4	5	6	7	11	13	18	20

4	5	6	7	11	13	18	20

4	5	6	7	11	13	18	20

B.

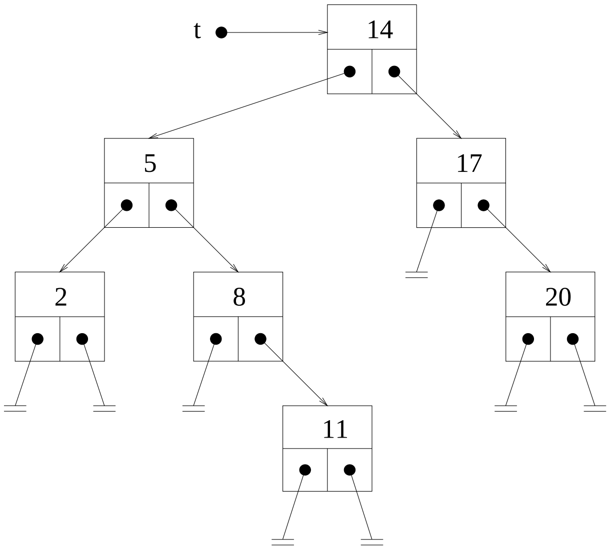
```
TYPE FELD = ARRAY [1..8] OF CARDINAL;
     CompareFunc = PROCEDURE(CARDINAL, CARDINAL) : INTEGER;

PROCEDURE BubbleSort(VAR arr : FELD; compare : CompareFunc);
VAR
  i, j, x: CARDINAL;
BEGIN
   FOR i := 2 TO 8 DO
       FOR j := 8 TO i BY -1 DO
           IF compare(arr[j-1], arr[j]) > 0 THEN
              x        := arr[j-1];
              arr[j-1] := arr[j];
              arr[j]   := x
           END
       END
   END
END BubbleSort;

PROCEDURE Decreasing(a, b : CARDINAL) : INTEGER;
BEGIN
   RETURN INTEGER(b) - INTEGER(a);
END Decreasing;
```

8. Aufgabe 10

Gegeben sei folgender **geordneter** Binärbaum `t` :



Dieser Baum soll mittels der Funktionsprozedur `TL` in eine lineare Liste `li` überführt werden.

```
PROCEDURE TL(t : TREE; li : LIST) : LIST;
BEGIN
  IF t = NIL THEN
    RETURN li
  ELSE
    RETURN TL(t^.right,  MakeListNode(t^.data, TL(t^.left, li)))
  END
END TL;
```

A. Geben Sie die lineare Liste an, die aus dem Aufruf 9

```
TL(t, NIL)
```

resultiert.

B. Wie nennt man die Art der Baum-Traversierung in `TL` ? 1

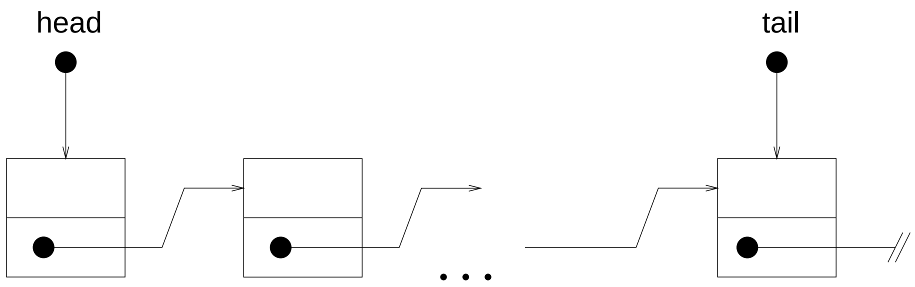
Lösung:

A. 20 17 14 11 8 5 2

B. Inorder (absteigend)

9. Aufgabe 14

In einem Computernetzwerk werden Druckaufträge nach dem *first-in first-out/FIFO*-Prinzip in einer Warteschlange abgearbeitet.
Die Warteschlange ist als Liste organisiert (siehe Abbildung). Hierbei werden neue Aufträge am Ende der Liste `tail` eingefügt und Aufträge aus der Warteschlange vom Kopf der Liste `head` entfernt.



Schreiben Sie die folgenden Prozeduren zur Verwaltung einer Warteschlange nach dem *FIFO*-Prinzip:

A. `PROCEDURE InsertItem(tail : LIST, info : CARDINAL) : LIST` 7

   Funktionalität: Ein neues Listenelement wird an das Ende der Liste (hierauf zeigt `tail`) angehängt. Hierzu soll Speicher für das neue Listenelement allokiert werden und das `data`-Feld durch den Wert des Prozedurparameters `info` belegt werden. Als Resultat soll `InsertItem` einen Zeiger auf das aktualisierte Listenende liefern.

B. `PROCEDURE DeleteItem(head : LIST) : LIST` 7

   Funktionalität: Das Listenelement am Kopf der Liste (hierauf zeigt `head`) soll aus der Liste entfernt werden. Dabei soll auch der Speicherplatz frei gegeben werden! Als Resultat soll `DeleteItem` einen Zeiger auf den aktualisierten Listenkopf liefern.

Lösung:

A.
```
PROCEDURE InsertItem(tail : LIST, info : CARDINAL) : LIST;
   VAR
      p : LIST;
BEGIN
   NEW(p);
   p^.data := info;
   p^.next := NIL;
   IF tail <> NIL THEN
      tail^.next := p;
   END;
   RETURN p;
END InsertItem;
```

B.
```
PROCEDURE DeleteItem(head : LIST) : LIST;
   VAR
      p : LIST;
BEGIN
   IF head <> NIL THEN
      p := head^.next;
      DISPOSE(head);
      RETURN p;
   ELSE
      RETURN NIL;
   END;
END DeleteItem;
```