

---

**Demo-Klausur Informatik II**
(Fassung vom 19.3.2001, 11:00)

---

Die Bearbeitungszeit für die Klausur beträgt 60 Minuten; es gibt insgesamt 62 Punkte. Als Hilfsmittel ist lediglich der *Revised<sup>5</sup> Report on the Algorithmic Language Scheme* zugelassen.

Wir haben uns bemüht, die Aufgaben so zu stellen, daß sie in Themenbreite und Schwierigkeitsgrad denen der richtigen Klausur so gut wie möglich entsprechen. Das bedeutet insbesondere nicht, daß in der richtigen Klausur nur Themen vorkommen, die auch in der Demo-Klausur vorkommen. *Für die Ähnlichkeit dieser Klausur mit der richtigen Klausur gibt es somit keine Gewähr.*

1. [10 Punkte]

- (a) Was ist ein endrekursiver Prozeduraufruf?
- (b) Wie läßt sich von einem Prozeduraufruf in einem Programm bestimmen, ob er endrekursiv ist oder nicht? Geben Sie die genauen Regeln für die Bestimmung in Programmen mit `lambda`, Prozeduraufrufen und `if` an.
- (c) Unterstreichen Sie in den folgenden Prozeduren die endrekursiven Aufrufe der Prozeduren `map`, `fold-left`, `list-sum`, `sum f, g, h` und `i`:

```
(define map
  (lambda (proc list)
    (if (null? list)
        '()
        (cons (proc (car list))
              (map proc (cdr list))))))
```

```
(define fold-left
  (lambda (op unit list)
    (if (null? list)
        unit
        (fold-left op (op unit (car list)) (cdr list)))))
```

```
(define list-sum
  (lambda (list)
    (letrec
      ((sum (lambda (list)
              (if (null? list)
                  0
                  (+ (car list) (sum (cdr list))))))
      (sum list))))
```

```
(define f
  (lambda (x)
    (g (h x))))
```

```
(define g
  (lambda (x)
    (+ x (* x 2))))
```

```
(define h
  (lambda (x)
    (+ 1 (i x))))
```

```
(define i
  (lambda (x)
    x))
```

2. [5 Punkte] Was sind die wesentlichen Unterschiede zwischen Scheme und Assembler?
3. [5 Punkte] Was macht das folgende Assemblerprogramm? Beschreiben Sie den Zusammenhang zwischen den Inhalten der Register a, b und c vor Start des Programms und danach. Begründen Sie ihre Antwort.

```
(registers a b c)
(operations cons null? car cdr)
(controller
  (assign b (const ()))
  x
  (test (op null?) (reg a))
  (branch (label z))
  (assign c (op car) (reg a))
  (assign b (op cons) (reg c) (reg b))
  (assign a (op cdr) (reg a))
  (goto (label x))
  z)
```

4. [12 Punkte] Schreiben Sie ein Assemblerprogramm, das eine ganzzahlige Wurzelfunktion  $r$  berechnet. Dabei soll für eine natürliche Zahl  $n$  gelten:

$$r(n) = \lfloor \sqrt{n} \rfloor$$

( $\lfloor x \rfloor$  ist die größte natürliche Zahl  $f$  mit  $f \leq x$ .)

Das Assemblerprogramm soll die Eingabe in einem Register  $n$  akzeptieren und das Ergebnis in einem Register  $e$  abliefern.

Verwenden Sie zur Berechnung den Newton'schen Algorithmus zur Berechnung der Quadratwurzel. Berechnen Sie also die Folgenglieder der Folge  $(a_i)_i$  mit:

$$a_{i+1} = \frac{a_i + \frac{n}{a_i}}{2}$$

Verwenden Sie  $a_0 = n/2$  und berechnen Sie die Folgenglieder für  $i = 0, 1, 2, \dots$  bis zum Abbruchkriterium  $a_{i+2} = a_i$ . Als primitive Operationen können Sie Addition (also die Scheme-Prozedur  $+$ ), Subtraktion ( $-$ ), ganzzahlige Division (`quotient`) und Multiplikation ( $*$ ) sowie  $=$  benutzen.

5. [10 Punkte] Überführen Sie die folgenden Prozeduren in CPS:

```
(define (gcd a b)
  (if (= b 0)
      a
      (gcd b (remainder a b))))
```

```

(define (remainder n d)
  (if (< n d)
      n
      (remainder (- n d) d)))

(define list-sum
  (lambda (list)
    (if (null? list)
        0
        (+ (car list) (list-sum (cdr list))))))

(define map
  (lambda (proc list)
    (if (null? list)
        '()
        (cons (proc (car list))
                (map proc (cdr list))))))

(define fold-right
  (lambda (op unit list)
    (if (null? list)
        unit
        (op (car list)
            (fold-right op unit (cdr list))))))

```

6. [6 Punkte] Programmieren Sie in Java einen Visitor `LongestStringListVisitor` für Listen von Strings, der den längsten String einer Liste findet und zurückgibt!

Zur Erinnerung hier die Definition für Listen mit Visitors:

```

public abstract class List {
    public abstract Object visitFrom(ListVisitor visitor);
}

public class EmptyList extends List {
    public Object visitFrom(ListVisitor visitor) {
        return visitor.visitedEmptyList();
    }
}

public class ConsList extends List {
    private Object car;
    private List cdr;

    public ConsList(Object theCar, List theCdr) {
        car = theCar;
        cdr = theCdr;
    }
    public Object getCar() {
        return car;
    }
    public List getCdr() {
        return cdr;
    }
}

```

```

    }
    public Object visitFrom(ListVisitor visitor) {
        return visitor.visitedConsList(car, cdr);
    }
}

```

und hier das Interface des Visitors für Listen:

```

public interface ListVisitor {
    Object visitedEmptyList();
    Object visitedConsList(Object car, List cdr);
}

```

Die Länge eines Strings `s` erhalten Sie mit `s.length()`.

7. [4 Punkte] Geben Sie die Definition einer Java-Methode `maxArray` (nur die Methode, keine Klasse drumherum) an, die für ein Array aus Ganzzahlen das Maximum zurückgibt.

`maxArray([1,6,5]) → 6`

Die Länge eines Arrays `a` läßt sich mit `a.length` bestimmen.

8. [10 Punkte]

- (a) Wandeln Sie die folgenden Zahlen in das einfach genaue IEEE-Floating-Point-Format um. Geben Sie jeweils das Vorzeichen sowie Charakteristik und Mantisse als Dezimalzahlen an:

- 3.14159265
- $-7.25 \cdot 10^{-7}$
- $1/3 \cdot 10^{-4}$

- (b) Addieren Sie 3.14159265 und  $-7.25 \cdot 10^{-7}$  im einfach genauen IEEE-Floating-Point-Format!

- (c) Multiplizieren Sie  $-7.25 \cdot 10^{-7}$  und  $1/3 \cdot 10^{-4}$  im einfach genauen IEEE-Floating-Point-Format!