

Vordiplomprüfung¹/Scheinklausur² Algorithmik 2 — September 2004

Angaben zur Person (Bitte in Druckschrift ausfüllen!):

Name, Vorname:

Geburtsdatum:

Matrikelnummer:

Studienfach (Bitte ankreuzen):

Informatik

Wirtschafts-
informatik

Linguistische
Informatik

Computational
Engineering

Mathematik

Techno-
mathematik

Sonstiges (Bitte angeben):

Nicht von der Kandidatin bzw. vom Kandidaten auszufüllen !!!

Punktzahlen:

Aufgabe	1	2	3	4	5	6	7	8	9	Σ
Max. Punkte	10	10	15	15	10	18	12	15	15	120
Erreichte Punkte										

Note:

¹Für Studierende der Informatik, Wirtschaftsinformatik

²Für Studierende der Linguistischen Informatik

Organisatorische Hinweise

Die folgenden Hinweise bitte aufmerksam lesen und die Kenntnisnahme durch Unterschrift bestätigen!

- Bitte legen Sie Ihren Studentenausweis und einen Lichtbildausweis zur Personenkontrolle bereit!
- Die angegebene Punkteverteilung gilt unter Vorbehalt.
- Hilfsmittel (außer Schreibmaterial) sind **nicht** zugelassen.
- Fragen zu den Prüfungsaufgaben werden grundsätzlich **nicht** beantwortet!
- Wenn Sie die Prüfung aus gesundheitlichen Gründen abbrechen müssen, so muss Ihre Prüfungsunfähigkeit durch eine Untersuchung in der Universitätsklinik nachgewiesen werden. Melden Sie sich in jedem Fall bei der Aufsicht und lassen Sie sich das entsprechende Formular aushändigen.
- Die Bearbeitungszeit beträgt 120 Minuten.
- **Überprüfen Sie die Prüfungsaufgaben auf Vollständigkeit (19 Seiten inklusive Deckblatt) und einwandfreies Druckbild!** Vergessen Sie nicht, auf dem Deckblatt die Angaben zur Person einzutragen!
- Verwenden Sie für die Darstellung der Wahrheitswerte **#t** und **#f**, für die leere Liste ' (). Darüber hinaus können Sie alle im *Revised⁵ Scheme Report* definierten Funktionen und Spezialformen benutzen.

Ausnahmen werden explizit in der jeweiligen Aufgabenstellung gestattet!

Erklärung

Durch meine Unterschrift bestätige ich den Empfang der vollständigen Klausurunterlagen und die Kenntnisnahme der obigen Informationen.

Erlangen, 21. September 2004

.....
(Unterschrift)

Ich bin damit einverstanden, dass mein Prüfungsergebnis unter Angabe der Matrikelnummer anonymisiert veröffentlicht wird:

ja: nein:

Erlangen, 21. September 2004

.....
(Unterschrift)

SCHEME-Programmierung

Aufgabe 1

(10 Punkte)

Stellen Sie sich vor, Sie starten für jede der folgenden Teilaufgaben den SCHEME-Interpreter erneut und geben die folgenden Ausdrücke ein. Geben Sie für jede Teilaufgabe das Ergebnis der Auswertung des **letzten** Ausdrucks an.

a)

```
(define (dbp x y z)
  (string-append (begin (set! x y) (string-append "x=" x)) ", "
    (begin (set! y z) (string-append "y=" y)) ", "
    (begin (set! z x) (string-append "z=" z))))
(dbp "3" "2" "1")
```

Lösungsvorschlag:

```
"x=2, y=1, z=2"
```

3 Punkte

b)

```
(define (f x)
  (cond ((= x 0) 3)
        ((= x 3) (* 3 (f (- x 3))))
        (else (lambda (x) (f (- x 3))))))
((f (f 3)) (f 0))
=> 3
```

3 Punkte

c)

```
(define (interleave l1 l2)
  (let hlp ((x (reverse l1)) (y l2))
    (cond ((or (null? x) (null? y) (null? (cdr y))) '())
          (else (cons (cons (car x) (car y))
                      (hlp (cdr x) (cddr y))))))
(interleave '(ganz ist diese) '(aufgabe klausur doch wirklich einfach leicht))
=> ((diese . aufgabe) (ist . doch) (ganz . einfach))
```

4 Punkte

Aufgabe 2

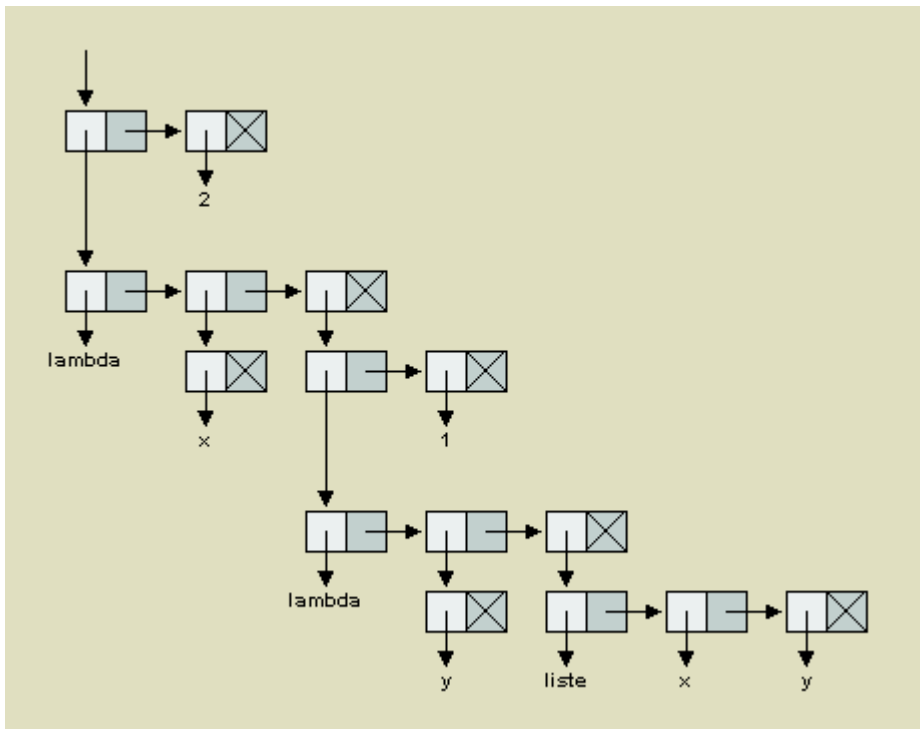
(10 Punkte)

Gegeben sei die folgende Liste:

```
((lambda (x) ((lambda (y) (list x y)) 1)) 2)
```

- a) Geben Sie eine entsprechende Kästchen-Zeiger-Darstellung für diese Liste an!
 (Achtung: Die Liste **nicht** vorher auswerten!)

Lösungsvorschlag:



7 Punkte

- b) Geben Sie das Ergebnis von (apply + (eval liste)) an, wobei liste die obige Liste sein soll.

Lösungsvorschlag:

3

3 Punkte

Aufgabe 3

(15 Punkte)

- a) Erläutern sie allgemein die Begriffe „rekursiv“, „endrekursiv“ und „iterativ“ und beziehen Sie dabei das Verarbeitungsmodell des SCHEME-Interpreters mit ein! Welchen Vorteil haben endrekursive Funktionen vor allgemein rekursiven?

Lösungsvorschlag:

Restrekursive Funktionen zeichnen sich durch fehlende „Einschachtelung“ der Funktion beim rekursiven Aufruf aus. Sie werden vom SCHEME-Interpreter wie iterative behandelt, d. h. sie haben konstanten Speicherplatzbedarf bei steigender Argumentgröße, während allgemein rekursive linearen Speicherplatzbedarf haben. Die Zeitkomplexität ist allerdings in beiden Fällen linear. Der Variablenstack kann bei jedem Rekursionsaufruf überschrieben werden, weil die Variablen in Akkumulatoren gespeichert sind.

5 Punkte

- b) Wie kann man allgemein rekursive in endrekursive Funktionen umwandeln?

Lösungsvorschlag:

Die Transformation rekursiv \rightarrow restrekursiv geschieht durch Einschachteln einer iterativen Hilfsfunktion; im einzelnen:

- Hinzufügen einer (oder mehrerer) Akkumulatorvariablen zu deren Parametern
- Rekursive Berechnungen ausschließlich in den Akkumulatorvariablen
- Rückgabe der Akkumulatorvariablen im Terminierungsfall
- Formulieren eines endrekursiven Rekursionsaufrufs

5 Punkte

- c) Die folgende Funktion `fakultaet-rek` ist rekursiv (warum?); geben sie dafür eine iterative Variante `fakultaet-iter` an! Achten Sie dabei auch auf die Stelligkeit von `fakultaet-iter`!

```
(define (fakultaet-rek n)
  (cond ((zero? n) 1)
        (else (* n (fakultaet-rek (- n 1))))))
```

Lösungsvorschlag:

```
(define (fakultaet-iter n)
  (let iter ((x n) (acc 1))
    (cond ((zero? x) acc)
          (else (iter (- x 1) (* x acc))))))
```

5 Punkte

Aufgabe 4

(15 Punkte)

- a) Schreiben Sie in SCHEME eine Funktion `left`, die als Parameter eine Liste `liste` und einen Integer $n > 0$ nimmt und als Ergebnis die Teilliste mit den ersten n Elementen aus `liste` zurück liefert.

Schreiben Sie weiterhin eine Funktion `right`, die als Parameter ebenfalls eine Liste `liste` und einen Integer $n > 0$ nimmt, als Ergebnis jedoch die Teilliste mit den letzten n Elementen aus `liste` zurück liefert.

Schreiben Sie schließlich mit Hilfe von `left` und `right` eine Funktion `mid`, die als Parameter eine Liste `liste` und zwei Integers l und r mit $0 < l \leq r$ nimmt und als Ergebnis die Teilliste beginnend mit dem l -ten bis einschließlich zum r -ten Element aus `liste` zurück liefert.

Anmerkung: Die Funktion `(list-tail liste n)`, die die Liste `liste` ohne die ersten n Elemente zurück liefert, ist Bestandteil des R⁵RS und sollte bei der Lösung der Aufgabe verwendet werden.

```
(define (left liste n)
```

```
)
```

```
(define (right liste n)
```

```
)
```

```
(define mid liste l r)
```

```
)
```

Lösungsvorschlag:

```
(define (left liste n)
  (reverse (list-tail (reverse liste) (- (length liste) n))))
(define (right liste n) (list-tail liste (- (length liste) n)))
(define (mid liste l r) (right (left liste r) (- r l -1)))
```

- b) Schreiben Sie mit Hilfe der Funktionen aus Teilaufgabe a) eine Funktion `swap`, die als Parameter eine Liste von Integers `liste` und zwei Integers `l` und `r` mit $0 < l < r$ nimmt und als Ergebnis die Liste zurück liefert, die sich ergibt, wenn man das `l`-te und das `r`-te Element der Liste `liste` vertauscht. Achten Sie dabei auch auf eine angemessene Fehlerabfrage für die Indizes!

```
(define (swap liste l r)
```

```
)
```

Lösungsvorschlag:

```
; Hilfsfunktionen
(define (nth liste el) (list-ref liste (- el 1)))
(define (disp . args)
  (for-each (lambda (arg) (display arg) (display " ")) args) (newline))

(define (swap liste l r)
  (let ((len (length liste)))
    (if (or (< l 1) (< r 1) (> l len) (> r len))
        (disp "Error: Illegale Parameter" l "und" r)
        (append (left liste (- l 1)) (list (nth liste r))
                  (mid liste (+ l 1) (- r 1)) (list (nth liste l))
                  (right liste (- (length liste) r))))))
```

6 Punkte

Aufgabe 5

(10 Punkte)

Gegeben seien die folgenden PROLOG-Regeln.

```
mystery([], 0).  
mystery(_|T, N) :- mystery(T, M), N is M + 1.
```

- a) Erläutern Sie kurz die Arbeitsweise dieser Regeln. Welche Listenfunktion lässt sich damit realisieren?

Lösungsvorschlag:

```
mystery([], 0). % Terminierung: [] hat die Länge 0  
% Rekursion: Die Länge einer Liste ist die Länge des Tails + 1  
mystery(_|T, N) :- mystery(T, M), N is M + 1.
```

Es handelt sich also um die Bestimmung der Länge einer Liste.

3 Punkte

- b) Wie lässt sich diese Funktion in SCHEME formulieren?

Lösungsvorschlag:

```
(define (laenge liste)  
  (cond ((null? liste) 0)  
        (else (+ 1 (laenge (cdr liste))))))
```

3 Punkte

c) Welche der folgenden PROLOG-Anfragen sind mit einer der obigen PROLOG-Regeln unifizierbar? Begründen Sie jeweils Ihre Antwort.

i) `mystery(L, 0)`.

Lösungsvorschlag:

Ja, mit der ersten Regel und der Belegung $L = []$.

1 Punkt

ii) `mystery([], 1)`.

Lösungsvorschlag:

Nein, weil für die erste Regel die Konstante 0 sein müsste und für die zweite die Liste nicht leer sein dürfte.

1 Punkt

iii) `mystery([a, b], L)`.

Lösungsvorschlag:

Ja, mit der zweiten Regel und den Belegungen:

$T = [b]$, $L = N = 2$, $M = 1$

1 Punkt

iv) `mystery([a], 2)`.

Lösungsvorschlag:

Nein, weil für die erste Regel die Liste leer und für die zweite der zweite Parameter 1 sein müsste.

1 Punkt

Aufgabe 6

(18 Punkte)

Ein *Funktionsverwalter* sei ein Objekt, das eine Funktion, nennen wir sie f , verwaltet. So kann über den Funktionsverwalter die Funktion f aufgerufen werden. Alle Aufrufe von f über den Funktionsverwalter werden protokolliert. Der Funktionswarter verfügt über geeignete Datenstrukturen, nämlich einen *Aufrufzähler* und eine *Protokollliste*, um die Anzahl, Argumente und Ergebnisse der Aufrufe von f repräsentieren zu können.

Sei nun fm-1 ein Funktionsverwalter-Objekt und f-1 die von fm-1 verwaltete Funktion. Folgende Operationen stellt fm-1 zur Verfügung:

- (fm-1 'call arg1 arg2 ... argn):
 - a) Erhöhen des Aufrufzählers um 1.
 - b) Falls die Parameterkombination (arg1 arg2 ... argn) in der Protokollliste enthalten ist: Rückgabe des gespeicherten Funktionswerts des Aufrufs von f-1 ;
sonst:
 - Aufruf der vom Objekt verwalteten Funktion f-1 mit den Argumenten arg1 , arg2 , ..., argn
 - Erweitern der Protokollliste um ein Paar aus der Liste der Aufrufparameter von f-1 und dem Rückgabewert von f-1
 - Rückgabe des Funktionswerts des Aufrufs von f-1
- (fm-1 'number-of-calls): Rückgabe der Anzahl der Aufrufe von f-1
- (fm-1 'calls): Rückgabe der Protokollliste der f-1 -Aufrufe (dies soll eine Assoziationsliste mit Paaren der Form (call . val) sein)
- (fm-1 'reset): Rücksetzen des Aufrufzählers und der Protokollliste

Implementieren Sie in SCHEME Funktionsverwalter als Objekte, die die o. g. Operationen realisieren, und die mit dem Aufruf (`make-function-manager` ($\langle \text{function-object} \rangle$)) erzeugt werden können.

Anmerkung: Die Funktion `assoc` ist Bestandteil des R⁵RS und sollte für die Lösung der Aufgabe benützt werden.

Verwenden Sie die in der objektorientierten Programmierung wichtigen Konzepte des *information hiding* und *message passing*!

Beispiele:

```
(define mult (lambda (x y) (* x y)))           ==> mult
(define mult-m (make-function-manager mult))    ==> mult-m
(mult-m 'call 2 4)                             ==> 8
(mult-m 'call 3 3)                             ==> 9
(mult-m 'number-of-calls)                       ==> 2
(mult-m 'calls)                                ==> (((3 3) . 9) ((2 4) . 8))
(mult-m 'reset)                                ==> #<unspecified>
(mult-m 'calls)                                ==> ()
```

```
(define (make-function-manager fobj)
```

```
)
```

Lösungsvorschlag:

```
(define (make-function-manager fobj)
  (let ((number-of-calls 0) (calls '()))
    (define (function-manager m . args)
      (cond ((eq? m 'number-of-calls) number-of-calls)
            ((eq? m 'calls) calls)
            ((eq? m 'reset) (set! number-of-calls 0) (set! calls '()))
            ((eq? m 'call) (set! number-of-calls (+ number-of-calls 1))
                          (let ((res (assoc args calls)))
                            (cond (res (display "taking memoized value: ") (cdr res))
                                  (else (let ((res (apply fobj args)))
                                          (set! calls (cons (cons args res) calls)) res))))))
            (else (error "wrong message"))))
    function-manager))
```

Aufgabe 7

(12 Punkte)

- a) Schreiben Sie in SCHEME eine Funktion `teile`, die als Parameter eine (möglicherweise verschachtelte) Liste `l` von Integers erwartet. Als Ergebnis soll die Liste zurückgeliefert werden, in der alle Teillisten von `l` (ohne die Atome in `l`) enthalten sind.

Beispiele:

```
(teile '(1 2 3 4))           ==> ()           ; nur Atome enthalten
(teile '((1 2) 3 4))        ==> (1 2)        ; nur eine Teilliste enthalten
; Auch eingeschachtelte Teile müssen gefunden werden!
(teile '((1 (2)) ((3) 4)))  ==> ((1 (2)) (2) ((3) 4) (3))
```

```
(define (teile l)
```

```
)
```

Lösungsvorschlag:

```
(define (teile l)
  (cond ((or (null? l) (number? l)) '())
        (else (let* ((1st (car l)) (last (append (teile 1st) (teile (cdr l)))))
                  (cond ((and (list? 1st) (not (null? 1st))) (cons 1st last))
                        (else last))))))
```

8 Punkte

- b) Schreiben Sie unter Verwendung der in a) definierten Funktion `teile` eine Funktion `teile-pretty-print`, die als Parameter eine Liste `liste` erwartet und die Teillisten von `liste` **durchnummeriert** etwa in der folgenden Form ausgibt:

```
(teile-pretty-print '((1 (2)) ((3) 4))) ==> Teilliste Nr. 1: (1 (2))
                                           Teilliste Nr. 2: (2)
                                           Teilliste Nr. 3: ((3) 4)
                                           Teilliste Nr. 4: (3)
```

```
(define (teile-pretty-print liste)
```

```
)
```

Lösungsvorschlag:

```
(define (teile-pretty-print liste)
  (define (disp . x) (for-each display x))
  (define (prt x i) (disp "Teilliste Nr. " i ": " x) (newline))
  (let hlp ((t (teile liste)) (ct 1))
    (if (not (null? t)) (prt (car t) ct) (hlp (cdr t) (+ ct 1)))))
```

4 Punkte

Aufgabe 8

(15 Punkte)

- a) Erläutern Sie das Substitutionsmodell der Auswertung **kurz!**

Lösungsvorschlag:

Nach dem Substitutionsmodell (allgemeine Auswertungsregel) wertet der Interpreter zuerst den Operator aus, ersetzt die formalen Parameter im Funktionsrumpf durch die aktuellen Parameter und wendet das Ergebnis auf die Operanden an (je nachdem, ob normale oder applikative Auswertung vorliegt, werden diese dabei zuerst ausgewertet oder nicht). Dabei wird *textuelle Substitution* verwendet.

5 Punkte

- b) Nennen Sie drei wesentliche Konzepte der Objektorientierten Programmierung und erläutern Sie diese **kurz!**

Lösungsvorschlag:

Vgl. Aufgabe 6! Denkbar wären z. B.:

- Objekte mit innerem Zustand
- Datenkapselung: Zugriff nur über definierte Schnittstellen
- Verschicken von Nachrichten an andere Objekte
- Vererbung via Klassenhierarchie

6 Punkte

- c) Erläutern Sie das Umgebungsmodell der Auswertung **kurz** und setzen Sie es in Beziehung zur Objektorientierten Programmierung!

Lösungsvorschlag:

Das Umgebungsmodell wird benötigt, sobald Seiteneffekte und innere Zustände erklärt werden müssen: Hier reicht die textuelle Substitution nicht mehr aus. Innere Zustände werden auch wesentlich für die Modellierung von Objekten benötigt (Klassenvariablen!).

4 Punkte

Aufgabe 9

(15 Punkte)

a) Nennen Sie 3 verschiedene Arten der Abstraktion und beschreiben Sie diese **kurz!**

Lösungsvorschlag:

denkbar wären z. B.:

- Prozedurale Abstraktion: Algorithmische Dekomposition
- Datenabstraktion: Benutzerdefinierte komplexe Datentypen
- Kontrollabstraktion: Einfrieren von Berechnungszuständen
- Syntaktische Abstraktion: Makros

6 Punkte

- b) Nennen Sie 3 verschiedene Parameterübergabemechanismen und erklären Sie diese **kurz!**

Lösungsvorschlag:

- *call by value* (+ kurze Erklärung)
- *call by name* (+ kurze Erklärung)
- *call by reference* (+ kurze Erklärung)

6 Punkte

- c) Welcher Parameterübergabemechanismus wird in SCHEME standardmäßig verwendet, welcher bei Makros?

Lösungsvorschlag:

Standardmäßig wird *call by value* verwendet, die Ausdrücke werden also vor dem Einsetzen ausgewertet. Lediglich bei Makros wird *call by name* verwendet, die Parameter werden also wirklich rein textuell substituiert.

3 Punkte