10. 10. 2006

Vordiplom-Klausur zu "Informatik I und II"

Name: ..... **Matrikelnummer:** .....

Vorname: .....

Studiengang: .....

Semesterzahl: .....

E-Mail: .....

Ich bin mit dem Aushang meines Klausurergebnisses unter alleiniger Nennung der Matrikelnummer

einverstanden: ☐ Ja ☐ Nein

Bitte in Druckschrift ausfüllen.

Hinweise: (GENAU DURCHLESEN!)

- Neben Papier und Schreibutensilien sind keine weiteren Hilfsmittel erlaubt. Verwenden Sie keine roten Stifte und keine Bleistifte.
- Vergessen Sie nicht, Ihren Namen und die Matrikelnummer auf *jedes* Blatt zu schreiben. Blätter ohne diese Angaben werden nicht gewertet.
- Schreiben Sie Ihre Lösungen auf die Aufgabenblätter möglichst in die dafür vorgesehenen Felder. Sie können auch die Rückseiten verwenden. Weiteres Schreibpapier kann von den Betreuern angefordert werden. Benutzen Sie kein mitgebrachtes Papier.
- Bei Multiple-Choice-Fragen wird für jedes richtig gesetzte Kreuz die im Kopf der Aufgabe angegebene Teil-Punktzahl vergeben. Für jedes falsch gesetzte Kreuz wird eine entsprechende Anzahl von Punkten abgezogen. Nicht beantwortete Fragen werden nicht gewertet. Die Gesamtpunktzahl beträgt mindestens 0.
- Bitte schreiben Sie in Ihrem eigenen Interesse deutlich. Für nicht lesbare Lösungen können wir keine Punkte vergeben.
- Klausurblätter dürfen nicht voneinander getrennt werden.
- Werden mehrere unterschiedliche Lösungen für eine Aufgabe abgegeben, so wird die Aufgabe nicht gewertet.
- Im Fall von Täuschungsversuchen wird die Klausur sofort mit 0 Punkten bewertet. Eine Vorwarnung erfolgt nicht.

Aufgabe	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	$\sum$
max. Punkte	8	8	8	8	10	8	7	8	9	7	9	10	**100**
erreicht													

1	Name:	(8 x 1)
	Matrikel-Nr.:	Σ: .....

Quickies

a) Der Stack hat eine

- ☐ FIFO-Struktur
- ☑ LIFO-Struktur

b) Die Addition zweier 8 Bit Zahlen ergibt im Ergebnis maximal

- ☐ 8 Bit
- ☑ 9 Bit
- ☐ 10 Bit

c) Die Multiplikation zweier 16 Bit Zahlen in Betrag-und-Vorzeichen-Darstellung ergibt im Ergebnis maximal

- ☐ 30 Bit
- ☑ 31 Bit
- ☐ 32 Bit
- ☐ 33 Bit

d) Seien p und q boolsche Variablen. Welcher der folgenden Ausdrücke ist äquivalent zum Ausdruck $\neg(p \vee q)$

- ☐ $\neg p \vee \neg q$
- ☐ $p \vee q$
- ☑ $\neg p \wedge \neg q$
- ☐ $p \wedge q$

e) Ist folgende Aussage wahr oder falsch?
$f \cdot g \in O(f \cdot g) \Rightarrow f \cdot g \in O(f) \cdot O(g)$

- ☐ falsch
- ☑ wahr

f) Die Laufzeit $T(n)$ eines Algorithmus sei gegeben durch folgende Rekursionsgleichung:
$T(1) = c$
$T(n) = 27 \cdot T(\frac{n}{3}) + n^3$
Eine Schranke im O-Kalkül für die Laufzeit beträgt:

- ☐ $\Theta(\log n^3)$
- ☐ $\Theta(n^3)$
- ☑ $\Theta(n^3 \log n)$

g) Ist die folgende Definition wahr oder falsch?
Ein Rechenverfahren ist genau dann ein *total korrekter Algorithmus*, wenn es für jede Eingabe, die der Eingabespezifikation genügt, terminiert.

- ☑ falsch
- ☐ wahr

h) Einfache Datentypen werden in Python immer per call-by-reference übergeben

- ☑ falsch
- ☐ wahr

2	Name:	(4+4)
	Matrikel-Nr.:	Σ: .....

Zahlendarstellung

a) Vervollständigen Sie die Tabelle:

Dezimalzahl	Binärzahl	Oktalzahl	Hexadezimalzahl
113	1110001	161	71
170	10101010	252	AA

b) Stellen Sie folgende Dezimalzahlen als 8-Bit-Binärzahlen im Einer- und Zweierkomplement dar.

i) 111

Einerkomplement	Zweierkomplement
01101111	01101111

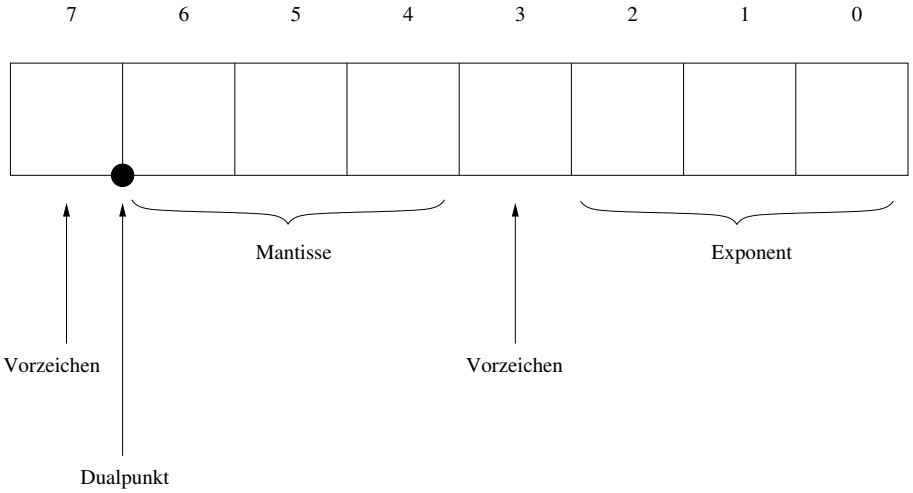
ii) -89

Einerkomplement	Zweierkomplement
10100110	10100111

3	Name:	(4+4)
	Matrikel-Nr.:	Σ: .....

Zahlendarstellung

Betrachten Sie die folgende Darstellung einer 8-Bit-Gleitkommazahl. Mantisse und Exponent sind in Betrag-und-Vorzeichen-Darstellung gespeichert. Die Basis für den Exponenten ist 2. Es existiert kein hidden Bit, d.h., das führende Bit wird nicht weggelassen. Anmerkung: Diese Darstellung entspricht nicht der IEEE-Darstellung für Gleitkommazahlen!



a) Bestimmen Sie den Dezimalwert der folgenden Bitstrings:

i) 01011011 Lösung: $\frac{5}{64} = 0,078125$ weil: $(\frac{1}{2} + \frac{1}{8}) \cdot 2^{-3} = \frac{5}{64}$

ii) 10010111 Lösung: -16 weil: $(-\frac{1}{8}) \cdot 2^{7} = -16$

b) Eine Zahl wird hier als normalisiert angenommen, wenn der Wert der Mantisse kleiner als Eins und größer oder gleich 0,5 ist.

Sind die beiden folgenden Zahlen normalisiert? Normalisieren Sie gegebenenfalls.

i) 01010111 Lösung: Ist schon normalisiert

ii) 00010001 Lösung: 01001001 weil: $00010001 = \frac{1}{8} \cdot 2^{1} = \frac{1}{4} = \frac{1}{2} \cdot 2^{-1} = 01001001$

4	Name:	(4+4)
	Matrikel-Nr.:	Σ: .....

Rekursion

Gegeben sei folgende Python-Funktion für ganzzahlige Eingaben:

```
def myst( n ):
    if n > 0:
        return n-1
    else:
        return myst( myst( n+2 ) )
```

a) Berechnen Sie das Ergebnis dieses Algorithmus für $n = -3$. Protokollieren Sie dabei die rekursiven Aufrufe der Prozedur. Vervollständigen Sie dazu die folgende Tabelle:

Lösung:

```
myst(-3) = myst( myst( -1 ) ) = myst( myst ( myst ( 1 ) ) )
                              = myst ( myst ( 0 ) )
                              = myst( myst (myst ( 2 ) ) )
                              = myst ( myst ( 1 ) )
                              = myst( 0 )
                              = myst ( myst ( 2 )  )
                              = myst ( 1 )
                              = 0
```

b) Geben Sie einen nicht-rekursiven Algorithmus für die Berechnung der Funktion `myst( n )` an.

Lösung:

```
def mystiter( n ):
    if n > 0:
        return n-1
    else:
        return 0
```

5	Name:	(3+7)
	Matrikel-Nr.:	Σ: .....

Assembler-Programmierung

Zur Erinnerung hier nochmals der Befehlssatz der Toy-Machine aus der Vorlesung:

OPCODE	DESCRIPTION	FORMAT	ASSEMBLERCODE
0	halt	-	exit
1	add	1	R[d] ← R[s] + R[t]
2	subtract	1	R[d] ← R[s] - R[t]
3	and	1	R[d] ← R[s] & R[t]
4	xor	1	R[d] ← R[s] ∧ R[t]
5	left shift	1	R[d] ← R[s] << R[t]
6	right shift	1	R[d] ← R[s] >> R[t]
7	load address	2	R[d] ← addr
8	load	2	R[d] ← mem[addr]
9	store	2	mem[addr] ← R[d]
A	load indirect	1	R[d] ← mem[R[t]]
B	store indirect	1	mem[R[t]] ← R[d]
C	branch zero	2	if (R[d] == 0) pc ← addr
D	branch positive	2	if (R[d] > 0) pc ← addr
E	jump register	-	pc ← R[d]
F	jump and link	2	R[d] ← pc; pc ← addr

Format 1:

opcode	dest d	source s	source t

Format 2:

opcode	dest d	addr

a) Welche Werte haben die Register 1, 2 und 3 nach Ausführung des folgenden Programms:

```
10: 7211
11: 7110
12: 2321
13: 0000
```

Lösung:

Register 1:	0010
Register 2:	0011
Register 3:	0001

b) Schreiben Sie ein Toy-Machine-Programm, das die Werte aus den Adressen `0A` und `0B` miteinander multipliziert und das Ergebnis in Adresse `0C` speichert. Es genügt die Angabe des Assemblercodes (Nicht Maschinencode!).

Lösung:

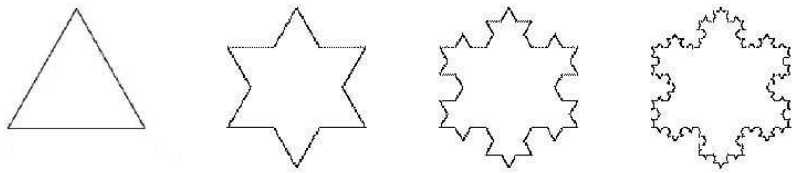
	Assembler-Code:
10:	$R[A] \leftarrow mem[0A]$
11:	$R[B] \leftarrow mem[0B]$
12:	$R[C] \leftarrow 00$
13:	$R[1] \leftarrow 01$
14:	$if R[A] == 0 : pc \leftarrow 18$
15:	$R[C] \leftarrow R[C] + R[B]$
16:	$R[A] \leftarrow R[A] - R[1]$
17:	$pc \leftarrow 14$
18:	$mem[0C] \leftarrow R[C]$
19:	$halt$
1A:	
1B:	
1C:	

6	Name:	(4+4)
	Matrikel-Nr.:	Σ: .....

Python-Programmierung

Die aus der Vorlesung und den Übungen bekannte Kochkurve ist ein rekursiv aus Dreiecken erzeugtes Fraktal, das nach folgender Vorschrift erzeugt wird:

Beginnend mit einem gleichseitigen Dreieck wird das mittlere Drittel jeder Dreiecksseite als Grundlinie für ein weiteres gleichseitiges Dreieck verwendet. Die Tiefe der Rekursion bestimmt das endgültige Aussehen der Kochkurve. Die unten abgebildeten Zeichnungen verdeutlichen den Entstehungsprozess einer Kochkurve mit Rekursionstiefe 3.



a) Leiten Sie aus der Entstehungsreihe der Kochkurve eine *rekursive* Formel für die Berechnung des Flächeninhalts der Kochkurve ab.
Der Flächeninhalt eines gleichseitigen Dreiecks mit Grundseite a beträgt $\frac{\sqrt{3}}{4}a^2$.

Lösung:

Bei der i-ten Iteration kommen $3 \cdot 4^{i-1}$ Dreiecke mit Seitenlänge $a \cdot (\frac{1}{3})^i$ hinzu. Deswegen:

$$f(n,a) = \begin{cases} \frac{\sqrt{3}}{4}a^2 \text{ falls n} = 0 \\ f(n-1,a) + 3 \cdot 4^{n-1} \cdot \frac{\sqrt{3}}{4}(\frac{a}{3^n})^2 \text{ sonst} \end{cases}$$

b) Schreiben Sie eine Python-Funktion `kochArea( n, a )`, die den Flächeninhalt einer Kochkurve mit Rekursionstiefe n und Grundseitenlänge a rekursiv berechnet.

Lösung:

```
def kochArea( n, a ):
    if n == 0:
        return 3**0.5*0.25*a**2
    else:
        return kochArea( n-1, a )+ 3*4**(n-1)*kochArea( 0, a / (3.0**n) )
```

7	Name:	(7 x 1)
	Matrikel-Nr.:	Σ: .....

Quickies

a) Jeder Knoten eines Baumes der Ordnung k muss genau k Kinder haben ☐ Richtig ☑ Falsch

b) Bei Wahl des Pivot-Elements mittels der 3-Median-Strategie hat Quicksort im schlechtesten Fall die Komplexität $\mathcal{O}(n \log n)$ ☐ Richtig ☑ Falsch

c) Beim Einfügen in einen B-Baum muss höchstens ein Knoten aufgespalten werden ☐ Richtig ☑ Falsch

d) Bubblesort ist ein stabiles Sortierverfahren. ☑ Richtig ☐ Falsch

e) Beim perfekten Hashing gibt es keine voneinander verschiedenen Schlüssel, die denselben Hashwert haben. ☑ Richtig ☐ Falsch

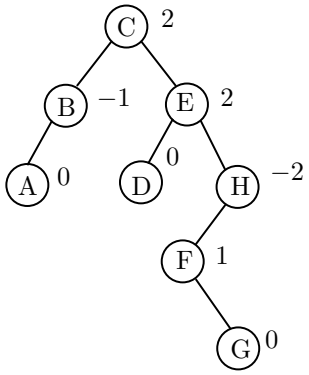
f) Nach dem Einfügen in eine AVL-Baum ändert sich immer die Balance aller Knoten im Baum. ☐ Richtig ☑ Falsch

g) Die Schleifeninvariante ist eine Bedingung, die in der Schleife nach jeder Anweisung gelten muss. ☐ Richtig ☑ Falsch

8	Name:	(4+4)
	Matrikel-Nr.:	Σ: .....

Suchbäume

Gegeben sei folgender binärer Suchbaum.



a) Formulieren Sie kurz, wie die Inorder-Traversierung rekursiv definiert ist und geben Sie die Schlüssel des Baumes in Inorder aus.

Pseudo-Code für Inorder-Traversierung:

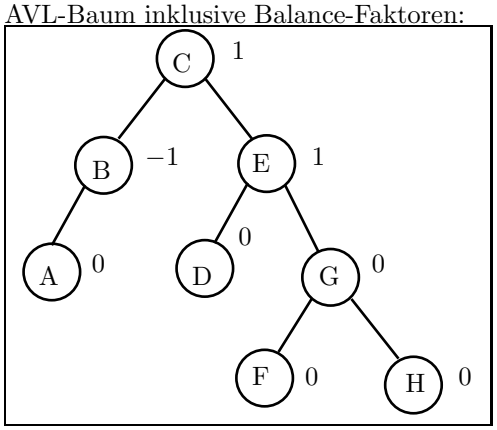
```
INORDER(node):
    if node == NULL:
       return
    INORDER(node.left)
    print node.value
    INORDER(node.right)
```

Ausgabe bei Inorder-Traversierung: A, B, C, D, E, F, G, H

b) Lässt man den Knoten mit dem Schlüssel G weg, so erfüllt der obige Baum die AVL-Eigenschaft. Nach dem Einfügen von G ist die AVL-Eigenschaft verletzt. Stellen Sie für diesen Baum die AVL-Eigenschaft wieder her. Schreiben Sie an jeden Knoten in obigem Baum den Balance-Faktor. Geben Sie den Namen der Rotation an, mit der der o.g. Baum in einen AVL-Baum umgewandelt werden kann. Geben Sie den Baum nach den Ausgleichsoperationen einschliesslich der Balancefaktoren an den Knoten an.

Name der Rotation: LR

AVL-Baum inklusive Balance-Faktoren:



9	Name:	(7+2)
	Matrikel-Nr.:	Σ: .....

Backtracking

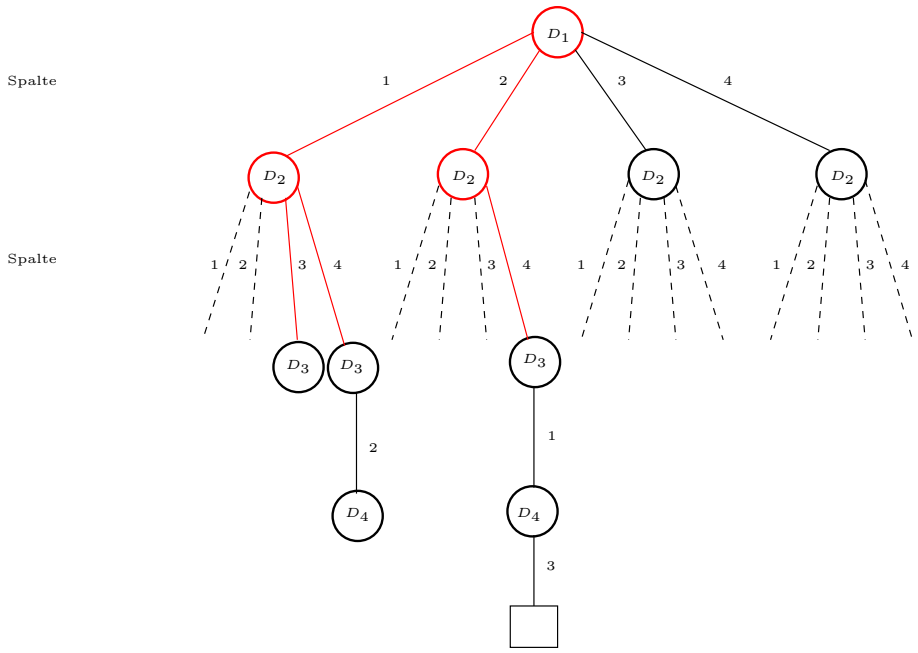
Betrachten Sie das aus der Vorlesung bekannte *Damenproblem*: Auf einem $n \times n$-Feld sind n Damen so zu platzieren, dass sie sich nicht gegenseitig bedrohen. Eine Dame bedroht alle Felder in der gleichen Zeile, Spalte und in den Diagonalen. Eine Dame in Feld (4,4) bedroht also alle Felder der Spalte 4, Zeile 4 und alle Felder $(4+i, 4+i)$ und $(4+i, 4-i)$ mit $i \in \mathbb{Z} \setminus \{0\}$.

Da in jeder Zeile höchstens eine Dame platziert werden kann, lässt sich eine Lösung als n-Tupel $(D_1, D_2, ..., D_n)$ ausdrücken, wobei D_i die Position (Spalte) der Dame in der i-ten Zeile angibt.

Wir beschränken uns in dieser Aufgabe auf $n = 4$. Der Aufruf `platziereDame(1)` der folgenden rekursiven Prozedur führt dann ein Backtracking durch:

```
platziereDame(i):                # i = Zeilennummer
    if i == 5:
        Stellung ausgeben
        HALT
    for j in range(1,5):
        D[i] = j                 # platziere i-te Dame in Spalte j
        if keine Bedrohung:
            platziereDame(i+1)
```

a) Markieren Sie in dem unten teilweise vorgegebenen Suchbaum den und nur den von dem Programm tatsächlich durchlaufenen Teil und ergänzen Sie weitere Rekursionstiefen so, dass der vollständige Entscheidungsbaum entsteht.



b) Wie oft wird überprüft, ob eine gegebene Platzierung eine „Bedrohung“ ist?

#Überprüfungen: 26

10	Name:	(3+4)
	Matrikel-Nr.:	Σ: .....

Hashing

Gegeben sei eine Hashtabelle der Länge 11 und die Hashfunktion $h(k) = k \mod 11$

a) Verwenden Sie *Chaining* für die Kollisionsauflösung. Fügen Sie in eine zu Beginn leere Hashtabelle Objekte mit den Schlüsseln 34, 40, 18, 27, 12 in dieser Reihenfolge ein. Geben Sie an, wie die Hashtabelle aussieht, nachdem alle Schlüssel eingefügt wurden.

0	1	2	3	4	5	6	7	8	9	10
	34				27		40			
	↓						↓			
	12						18			

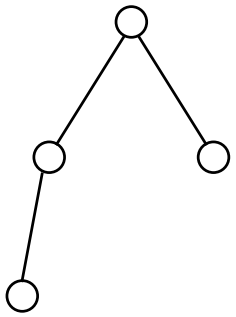
b) Verwenden Sie nun *Open Adressing* mit der Sondierungsfolge $s(i, k) = i + i^2$ für die Kollisionsauflösung. Fügen Sie in eine zu Beginn leere Hashtabelle Objekte mit den Schlüsseln 23, 12, 14, 25 in dieser Reihenfolge ein. Geben Sie an, wie die Hashtabelle nach diesen Operationen aussieht.

0	1	2	3	4	5	6	7	8	9	10
	23		12		14				25	

11	Name:	(4+5)
	Matrikel-Nr.:	Σ: .....

Binärbäume

Gegeben ist folgender binärer Suchbaum mit 4 Knoten.



a) Geben Sie an, mit welcher Wahrscheinlichkeit dieser Baum durch sukzessives Einfügen der Schlüssel aus der Menge $\{0, 1, 2, 3\}$ in den anfangs leeren Baum erzeugt wird, wenn jede Permutation der Schlüssel $0, 1, 2, 3$ als gleich wahrscheinlich vorausgesetzt wird.

Wahrscheinlichkeit: 1/8

b) Wieviele strukturell verschiedene Binärbäume mit vier Knoten gibt es? Geben Sie an, wie Sie zu Ihrer Lösung kommen. (z.B. Skizze, Berechnung,...).

Anzahl: 14

12	Name:	(7+3)
	Matrikel-Nr.:	Σ: .....

Schleifeninvarianten

Gegeben ist folgender Programmcode.

```
x = b
k = n
z = 1
while k != 0:
    if k % 2 != 0:
        z = z * x
    x = x * x
    k = k / 2
```

a) Geben Sie an, welche der folgenden Invarianten korrekt ist.

- ☑ $z \cdot x^k = b^n \ \wedge \ k \geq 0$
- ☐ $z \cdot b^k = x^n \ \wedge \ k \geq 0$
- ☐ $z = b^{n-k} \ \wedge \ k \geq 0$
- ☐ $z = x^{n-k} \ \wedge \ k \geq 0$

b) Geben Sie die Nachbedingung nach dem Ende der `while`-Schleife an.

Nachbedingung: $z = b^n$