# Fakultät Elektrotechnik/Informatik

# Klausur Einführung in die Informatik I für Elektrotechniker 16. Juli 2003

**Name:** .....

**Matr.-Nr.** .....

Bearbeitungszeit: 120 Minuten

**Bewertung**
(bitte offenlassen :–)

| **Aufgabe** | Punkte | Erreichte Punkte |
|---|---|---|
| 1 | 4 | |
| 2 | 7 | |
| 3 | 4 | |
| 4 | 4 | |
| 5 | 7 | |
| 6 | 10 | |
| 7 | 8 | |
| Summe | 44 | |

Spielregeln (**Jetzt lesen!**):

- Benutzen Sie für die Lösung der Aufgaben **nur** das mit diesem Deckblatt ausgeteilte Papier. Lösungen, die auf anderem Papier geschrieben werden, können **nicht** bewertet werden. Schreiben Sie ihre Lösung auch auf die Rückseiten der Blätter; benötigen Sie für eine Lösung mehr als ein Blatt, finden Sie am Ende der Klausur Leerblätter. Zusätzliches Papier können Sie von den Tutoren bekommen.
- Tragen Sie jetzt (vor Beginn der eigentlichen Bearbeitungszeit !!!) auf *allen* Blättern ihren Namen und ihre Matrikelnummer ein.
- Schreiben Sie deutlich! Unleserliche oder zweideutige Lösungen können nicht gewertet werden.
- Schreiben Sie *nicht* mit Bleistift und *nicht* mit rotem oder grünem Stift (das sind die Farben für die Korrektur).
- Lesen Sie die Aufgaben jeweils bis zum Ende durch; oft gibt es hilfreiche Hinweise!
- Kommentare kosten Zeit; kommentieren Sie ihr Programm nur da, wo der Code alleine nicht verständlich wäre.
- Wir weisen noch einmal darauf hin, daß die Benutzung von Taschenrechnern und anderen elektronischen Hilfsmitteln nicht gestattet ist.

Viel Erfolg!

• **AUFGABE 1 (4 Punkte) Theorie.**

1. (2 Punkte) Erläutern Sie die Begriffe *Attribut*, *Variable* und *Parameter*. Wie stehen diese zueinander in Beziehung?

2. (1 Punkt) Was ist ein *Register* und wozu dient es?

3. (1 Punkt) Ein gegebenes Programm `p` hat als Eingabe eine natürliche Zahl $n$. Bei $n = 2$ läuft `p` doppelt so lange wie bei $n = 1$. Welche der folgenden Aussagen ist richtig?

   ◯ Das Programm hat *logarithmischen* Aufwand.

   ◯ Das Programm hat *linearen* Aufwand.

   ◯ Das Programm hat *exponentiellen* Aufwand.

   ◯ Die Information ist für die Bestimmung der Aufwandsklasse nicht ausreichend.

   Begründen Sie Ihre Antwort!

## • AUFGABE 2 (7 Punkte) JAVA.

1. (2 Punkte) Geben Sie den Aufwand der Methoden `f` und `g` in Anhängigkeit der Parameter `n` bzw. `k` in $\mathcal{O}$-Notation an.

```
void f(int n) {
  int a = 2 * n;
  int b = 0;
  do {
    b = b + g(a);
    a--;
  } while(a > 0);
}

void g(int k) {
  int l = 0;
  int m = 1;
  while (l < k) {
    m = m * k;
    l = l + 1;
  }
  l = m;
  return l;
}
```

2. (2 Punkte) Ersetzen Sie folgendes Code-Fragment durch äquivalenten Java-Code ohne `switch`-Anweisung.

```
int x = ...;
int y = 0 ;
switch (x) {
  case 1:
    y = 0; break;
  case 2:
    y = 1;
  default:
    y--; break;
}
```

3. (3 Punkte) Welche Fehler enthält folgendes Java-Code-Fragment? Geben Sie jeweils die Zeilennummer an und beschreiben Sie den Fehler. Folgefehler werden ignoriert.

```
 1  class Foo {
 2    int value;
 3
 4    Foo(int value) {
 5      this.value = value;
 6    }
 7
 8    int g(String diff) {
 9      value = (value + diff) / 2;
10      return value;
11    }
12
13  }
14
15  class Bar {
16    Foo foo = new Foo();
17    Bar bar;
18
19    Bar() {
20      bar = this;
21      int bletch = this.bar.f();
22    }
23
24    int f() {
25      return foo.g("42") + Foo(1).g("43");
26    }
27  }
```

*Diese Seite wurde absichtlich freigelassen.*

• **AUFGABE 3 (4 Punkte) Zahlensysteme.**

1. (2 Punkte) Wandeln Sie die reelle Zahl 0.85 in eine Dualzahl mit 4 Nachkommastellen um und wandeln Sie diese dann wieder in eine reelle Zahl um. Geben Sie den dabei auftretenden Fehler an.

   Lassen Sie den Lösungsweg erkennen.

2. (2 Punkte) Führen Sie folgende Berechnungen unter Verwendung der 2-Komplementdarstellung auf einem imaginären 4-Bit-Rechner aus.

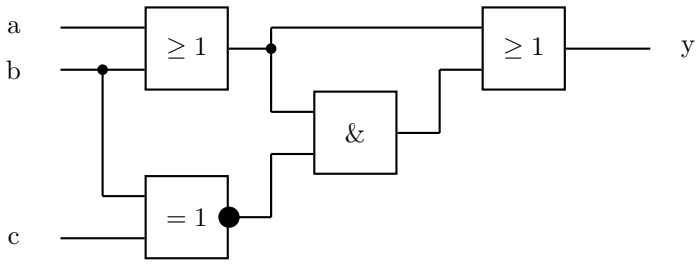
   Lassen Sie den Lösungsweg erkennen. Wo findet ein *Over-* oder *Underflow* statt? Woran erkennt man das Auftreten eines *Over-* bzw. *Underflows*?

   - $-3 - 6$
   - $5 + 5$
   - $7 - 1$

*Diese Seite wurde absichtlich freigelassen.*

• **AUFGABE 4 (4 Punkte) Schaltungen.**

1. (2 Punkte) Stellen Sie die Wertetabelle für folgende Schaltung auf:




2. (2 Punkte) Geben Sie eine einfachere Schaltung an, die das gleiche berechnet.

*Diese Seite wurde absichtlich freigelassen.*

• **AUFGABE 5 (7 Punkte) Gauß-Reihe.** Der Mathematiker *Gauß* entdeckte als Schüler im Mathematikunterricht folgende Gesetzmäßigkeit:

$$\sum_{i=1}^{n} i = \frac{n \cdot (n+1)}{2}$$

1. (3 Punkte) Schreiben Sie eine Methode
   - `int sum(int n)`, welche die Summe der natürlichen Zahlen von 1 bis $n$ berechnet. Für `int`-Zahlen $n \leq 0$ soll das Ergebnis 0 sein.

     Berechnen Sie die Summe durch Addition der einzelnen Zahlen, *nicht* durch die vereinfachte Formel von Gauß!
2. (3 Punkte) Schreiben Sie eine Methode
   - `boolean checkGauss(int a, int b)`, welche prüft, ob die Gleichung von Gauß für alle $n$ mit $a \leq n \leq b$ gilt.
3. (1 Punkt) Welches Ergebnis liefert die von Ihnen geschriebene Methode
   - beim Aufruf `checkGauss(-2, -1)`?
   - beim Aufruf `checkGauss(5, 3)`?

*Diese Seite wurde absichtlich freigelassen.*

**• AUFGABE 6 (10 Punkte) Boxer.**

Für statistische Untersuchungen über Boxer soll ein Java-Programm geschrieben werden. Dazu wurden die relevanten Attribute der Testpersonen in folgender Klasse festgehalten:

```
class Boxer {
  String name ;
  int iq ;        // Intelligenzquotient
  float gewicht ; // in Kilogramm

  // weiteres siehe unten
}
```

*Hinweis:* Für jede Teilaufgabe können Sie die Methoden aus vorangehenden Teilaufgaben verwenden. Dies gilt auch, wenn Sie keine eigene Lösung angegeben haben.

1. (1 Punkt) Erweitern Sie die Klasse `Boxer` um eine Methode
   - `boolean inGewichtsklasse(float von, float bis)`, welche feststellt ob der Boxer in die angegebene Gewichtsklasse gehört. Dies ist der Fall, wenn sein Gewicht größer als `von`, aber nicht größer als `bis` ist.
2. (3 Punkte) Schreiben Sie eine Methode
   - `Boxer[ ] gewichtsklasse(Boxer[ ] A, float von, float bis)`, welche aus dem angegebenen Array von Boxern alle diejenigen auswählt, die in die angegebene Gewichtsklasse gehören.
3. (2 Punkte) Schreiben Sie eine Methode
   - `int durchschnittsIq(Boxer[ ] A)`, welche den durchschnittlichen Intelligenzquotienten in einem nichtleeren Array von Boxern berechnet.

     *Hinweis:* Gesucht ist der *arithmetische* Mittelwert, also die Summe der einzelnen Intelligenzquotienten, geteilt durch die Anzahl der Boxer.
4. (2 Punkte) Erweitern Sie die Klasse `Boxer` um eine Methode
   - `boolean überlegen(Boxer other)`, welche feststellt, ob ein Boxer einem anderen überlegen ist.

     Es gilt: Ein Boxer sei überlegen, wenn er einen höheren Intelligenzquotienten besitzt. Bei gleicher Intelligenz entscheidet das höhere Gewicht.
5. (2 Punkte) Nun soll getestet werden, ob das in der vorigen Teilaufgabe definierte Kriterium für Überlegenheit zutreffend ist. Dazu wurden Ranglisten von Boxern ermittelt und als Arrays gespeichert.

   Schreiben Sie eine Methode
   - `boolean testRangliste(Boxer[ ] R)`, welche überprüft, ob der erste Boxer der angegebenen Rangliste `R` dem zweiten `überlegen` ist, der zweite dem dritten und so weiter.

*Diese Seite wurde absichtlich freigelassen.*

**• AUFGABE 7 (8 Punkte) Schiffe versenken.**

Das Spiel *Schiffe versenken* wird auf einem rechteckigen Spielbrett gespielt. Dieses kann man in Java durch eine Matrix von `short`-Werten repräsentieren:

- der Wert `0` steht für Wasser.
- ein positiver Wert $n > 0$ steht für das Schiff mit der Nummer $n$.

Für diese Aufgabe sollen nur Schiffe betrachtet werden, die genau 1 Feld groß sind. Außerdem nehmen wir an, daß jede Schiffsnummer auf dem Spielbrett nur einmal vorkommt.

Beispiel für ein Spielbrett mit $7 \times 5$ Feldern und 3 Schiffen:

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 3 | 0 | 0 | 0 |
| 2 | 0 | 0 | 0 | 0 | 0 | 4 | 0 |
| 3 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 4 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

*Hinweis:* Sie dürfen jede der zu programmierenden Methoden für die übrigen Teilaufgaben verwenden. Dies gilt auch, wenn Sie selbst keine Lösung dafür angegeben haben.

1. (1 Punkt) Schreiben Sie eine Methode
   - `short[ ][ ] anfang(int spalten, int zeilen)`, welche ein neues Spielbrett der angegebenen Größe liefert. Das Spielbrett soll im Anfangszustand sein; d.h., alle Felder sind mit Wasser belegt.
2. (2 Punkte) Schreiben Sie eine Methode
   - `boolean testSchiff(short[ ][ ] spielbrett, int spalte, int zeile)`, welche `true` liefert, falls sich auf dem `spielbrett` auf dem angegebenen Feld (`spalte`, `zeile`) ein Schiff befindet. Ist dort Wasser, ist das Ergebnis `false`.

     Im Fall von ungültigen Werten für `spalte` oder `zeile` (außerhalb des Spielbrettes) soll ebenfalls das Ergebnis `false` geliefert werden.
3. (2 Punkte) Schreiben Sie eine Methode
   - `short nächstesSchiff(short[ ][ ] spielbrett)`, welche die nächste freie Schiffsnummer auf dem `spielbrett` berechnet. Dabei gilt:
     - für ein leeres Spielbrett soll das Ergebnis 1 sein.
     - wenn $n$ die höchste auf dem Spielbrett vorkommende Schiffsnummer ist, dann soll das Ergebnis $n + 1$ sein.

4. (2 Punkte) Schreiben Sie eine Methode

   – `boolean frei(short[ ][ ] spielbrett, int spalte, int zeile)`, welche `true` liefert, falls auf das Feld (`spalte`, `zeile`) auf dem `spielbrett` ein Schiff gesetzt werden darf.

   Dabei gilt folgende Regel: *Schiffe dürfen nicht auf schon besetzte Felder und nicht direkt nebeneinander oder übereinander gesetzt werden.*

   Die im Beispiel grau gefärbten Felder dürfen also nicht besetzt werden:

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 3 | 0 | 0 | 0 |
| 2 | 0 | 0 | 0 | 0 | 0 | 4 | 0 |
| 3 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 4 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

5. (1 Punkt) Geben Sie ein Stück Java-Code an, welches das abgebildete Spielbrett erzeugt.

*Diese Seite wurde absichtlich freigelassen.*

*Diese Seite wurde absichtlich freigelassen.*

**Name:** .....

**Matr.-Nr.** .....