Fakultät Elektrotechnik/Informatik

Klausur Einführung in die Informatik II für Elektrotechniker (Version A)

18. Juli 2002

Name: .....

Matr.-Nr. .....

Bearbeitungszeit: 120 Minuten

Bewertung
(bitte offenlassen :–)

Aufgabe	Punkte	Erreichte Punkte
1	6	
2	7	
3	5	
4	7	
5	10	
6	9	
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.
- Bitte schreiben Sie 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!
- Wir weisen noch einmal darauf hin, daß die Benutzung von Taschenrechnern und anderen elektronischen Hilfsmitteln nicht gestattet ist.

Viel Erfolg!

• AUFGABE 1 (6 Punkte) Theorie.

1. (2 Punkte) Was versteht man unter *balancierten* Bäumen und wofür sind sie wichtig?

2. (2 Punkte) Erstellen Sie einen balancierten Binärbaum, bei dem durch Postorder-Traversierung die Zeichenkette $T\ R\ A\ V\ E\ R\ S\ I\ E\ R\ U\ N\ G$ (ohne Leerzeichen) ausgegeben wird.

3. (2 Punkte) Klassen, Arrays und (verkettete) Listen erlauben jeweils die Aggregation (das Zusammenfassen) von mehreren Datenelementen. Je nach Eigenschaften (z.B. Art, Anzahl wechselnd) der zu speichernden Datenelemente ist jeweils eine dieser Datenstrukturen besser geeignet. Unter welchen Bedingungen würden Sie sich jeweils für welche der genannten Datenstrukturen entscheiden?

• **AUFGABE 2 (7 Punkte) JAVA.**

1. (2 Punkte) Was versteht man unter *Referenzen*? Welche beiden Arten von „Gleichheit“ gibt es deshalb für Objekte?

2. (1 Punkt) Wozu dient das Schlüsselwort *public* in Java?

3. (1 Punkt) Was sind *packages* und wozu dienen sie?

4. (3 Punkte) Wo sind die Fehler im folgenden *JAVA*-Code? Beschreiben Sie die Fehler und geben Sie die Zeilenzahlen des Auftretens an. (Folgefehler werden wie üblich ignoriert.)

```
 1 /* Datei Foo.java */
 2 package mypackages;
 3
 4 public class Foo {
 5     private int value;
 6
 7     Foo(int value) {
 8         this.value = value;
 9     }
10
11     int getValue() {
12         return value;
13     }
14 }
15
16 class Bar extends Foo {
17     public String store;
18     protected Bar next;
19
20     Bar(String value) {
21         super();
22         this.store = value;
23         this.next = null;
24     }
25
26     void doSomething(int counter) {
27         Foo f = new Foo(counter);
28         store = store + counter;
29         next = f;
30     }
31 }
32
33 // ------------------------------------------------------
34
35 /* Datei Ext.java */
36 package mypackages.ext;
37
38 class Ext {
39     int doMore(int val) {
40         Foo f = new Foo(val);
41         for (int i = 1; i <= f.getValue(); i++) {
42             (new Ext()).doMore(val - i);
43         }
44     }
45 }
```

• AUFGABE 3 (5 Punkte) Numerik.

Schreiben Sie eine *JAVA*-Methode

```
double cos(double x)
```

welche den Cosinus einer Gleitpunktzahl berechnet.

Verwenden Sie dabei nicht die vordefinierten Funktionen aus der *Java*-Klasse `Math`, insbesondere nicht `Math.cos`, sondern berechnen Sie die Cosinusfunktion durch Approximation der Taylor-Reihe

$$\cos(x) \;=\; \sum_{k=0}^{\infty} (-1)^k \frac{x^{2k}}{(2k)!}$$

Beenden Sie jeweils den Approximationsprozeß, wenn eine Genauigkeit von 10 Stellen nach dem Komma erreicht ist. Formulieren Sie diese Bedingung mit Hilfe einer zu erstellenden Hilfsfunktion `boolean close( double x, double y, double eps )`.

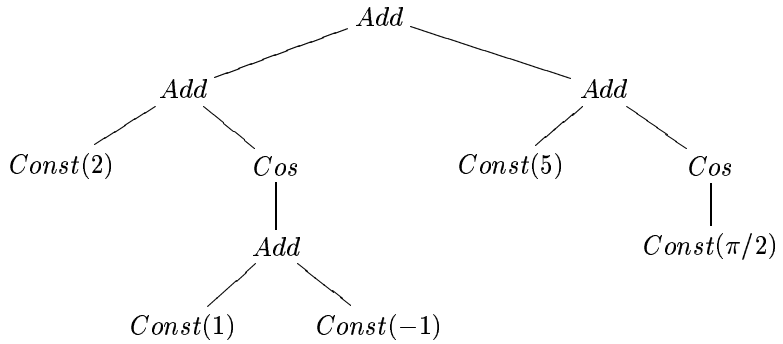
• AUFGABE 4 (7 Punkte) Expression-Trees.

Arithmetische Ausdrücke lassen sich durch Bäume der Klasse

```
abstract class Expr {
    double eval();
}
```

repräsentieren, die eine Methode `eval()` zur Auswertung des Ausdrucks bereitstellt. Davon abgeleitete Unterklassen wie `Const`, `Add` und Cos implementieren diese Methode jeweils für die konkrete Art des arithmetischen Ausdrucks.

Beispiel: Der folgende Baum repräsentiert den Ausdruck $(2 + \cos(1 + (-1))) + \left(5 + \cos(\frac{\pi}{2})\right) = (2+1)+(5+0) = 8$. `eval()`, angewendet auf den Wurzelknoten (`Add`) würde also 8 liefern.



Schreiben Sie die folgenden Klassen, die jeweils von `Expr` erben und keinen (`Const`), zwei (`Add`) bzw. einen (`Cos`) Unterbaum vom Typ `Expr` enthalten. Die Klassen sollen jeweils einen geeigneten Konstruktor sowie die Methode `eval` implementieren.

1. (2 Punkte) Schreiben Sie eine Klasse `Const`, die von `Expr` erbt und eine Konstante vom Typ `double` repräsentiert.

   Der Wert der Konstanten wird geeignet als Attribut gespeichert, im Konstruktor gesetzt und bei `eval` zurückgeliefert. Da Konstanten die Blätter eines Expression-Trees sind, gibt es hier keine Unterbäume.

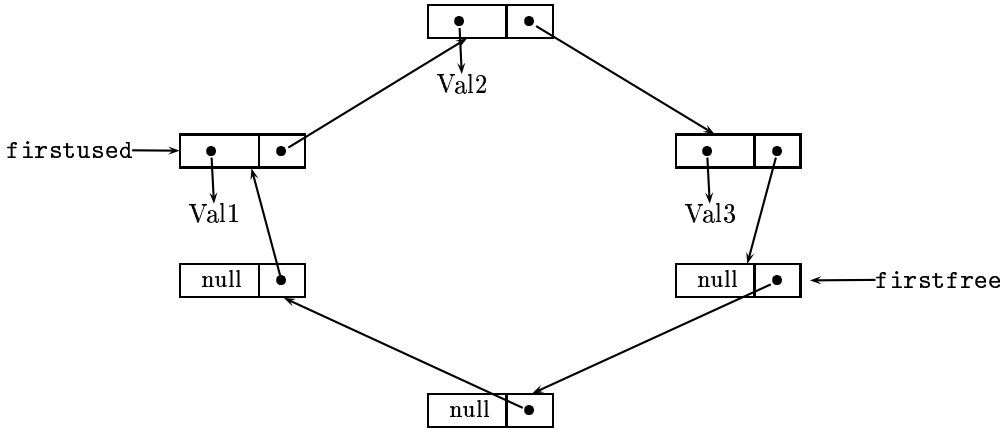
2. (3 Punkte) Schreiben Sie ferner eine von `Expr` abgeleitete Klasse `Add`, welche die Summe zweier Summanden darstellt.

   Sie enthält folglich zwei (Unter-)Ausdrücke als Attribute, welche gleichzeitig die beiden Unterbäume darstellen.

3. (2 Punkte) Leiten Sie von `Expr` ebenfalls eine Klasse `Cos` ab, welche den Cosinus eines Unterausdrucks berechnen kann. Dazu können Sie in der Methode `eval()` die in *Java* vordefinierte Methode `Math.cos` verwenden.

• AUFGABE 5 (10 Punkte) RingBuffer.

Ein `RingBuffer` ist eine Verwaltungsstruktur, über die ein Zugriff auf das zuerst eingefügte belegte Element (`firstused`, das als nächstes zu liefernde Datenobjekt) und auf das erste leere Element (`firstfree`) möglich ist. Weiterhin erlauben sie wie bei Listen den Zugriff auf das jeweils nächste Element. Dabei ist das letzte Element mit dem ersten verbunden, um einen Ring zu bilden. Die Größe des Ringpuffers, d.h. die Anzahl der speicherbaren Datenobjekte, ist fest und wird bei der Erzeugung des Puffers angegeben.



Wenn ein neues Datenobjekt eingefügt werden soll, wird es in dem ersten leeren Ringelement eingetragen und der Verweis auf das erste leere Element eins weiter gesetzt. Gibt es kein leeres Ringelement mehr, d.h. falls der Puffer voll belegt ist, wird das älteste Datenelement überschrieben, d.h. gelöscht. Wird ein Datenelement ausgelesen und damit gleichzeitig entfernt, muß der Zeiger auf das erste belegte Ringelement (nach Auslesen das Datenobjektes) um ein Element weitergesetzt werden. Beim Auslesen wird stets – wie bei einer normalen Warteschlange (Queue) – das älteste, im Puffer enthaltene Datenobjekt zurückgegeben.

1. (1 Punkt) Schreiben Sie eine Klasse `RingElem`, welche ein Element des Ringes verwaltet. Dazu sind Verweise auf den eventuell gespeicherten Dateninhalt vom Typ `Object` sowie das nächste Ringelement erforderlich.

   Ihre Klasse soll einen geeigneten Konstruktor zum Setzen der Attribute bereitstellen.

2. (9 Punkte) Schreiben Sie ferner eine Klasse `RingBuffer` zur Verwaltung eines Ringpuffers. Es werden Verweise auf das erste Element der Warteschlange (`firstused`) sowie auf das erste freie Element (`firstfree`) benötigt.

   Die Klasse `RingBuffer` soll die folgenden Methoden bereitstellen:

   - Der Konstruktor `RingBuffer(int size)` soll einen Ringpuffer erzeugen, der `size` Elemente speichern kann.
   - `void enqueue(Object value)`, welche ein neues Objekt im Ring speichert. Falls keine freien, belegbaren Ringelemente mehr existieren, wird einfach das nächste (älteste) Datenobjekt in der Warteschlange überschrieben. (Der nächste Aufruf von `dequeue` soll dann nicht das neu eingefügte, sondern das nächstälteste Datenobjekt liefern.)

     Beachten Sie, daß Sie den Fall eines vollen Ringes (alle Ringelemente mit Daten belegt) vom dem eines leeren Ringes (alle Ringelemente enthalten keine Daten) geeignet unterscheiden müssen.
   - `Object dequeue()`, welche das nächste Datenobjekt in der Warteschlange liefert. Das von ihm im Ringpuffer belegte Ringelement wird durch Weitersetzen von `firstused` für die weitere Verwendung freigegeben. Falls sich keine Daten im Ringpuffer befinden, wird `null` zurückgegeben.

• AUFGABE 6 (9 Punkte) Sortierung von Kundenlisten.

In einer Firma werden Listen von Kunden mit ihren jeweiligen Umsätzen verwaltet. Da zu Werbezwecken vor allem die Kunden mit großem Umsatz angesprochen werden sollen, muß die vorhandene Kundenliste nach Umsätzen absteigend sortiert werden, wobei bei gleichem Umsatz die Elemente nach Kundennamen (aufsteigend) sortiert sein sollen.

1. (3 Punkte) Erstellen Sie eine Klasse `Kunde`, welche den Namen eines Kunden (als `String`) sowie dessen Umsatz verwaltet. Die Attribute sollen von außen nicht zugreifbar sein und im Konstruktor gesetzt werden.

   Die Klasse `Kunde` soll Methoden `getName` und `getUmsatz` zum Lesen von Kundennamen und Umsatz sowie eine geeignete Methode `public String toString()` zur Ausgabe bereitstellen.

   Ferner soll eine Methode `boolean vergleicheMit(Kunde k)` in der Klasse `Kunde` vorhanden sein, welche `true` liefert, wenn der eigene Umsatz größer als der vom Kunden `k` ist, oder im Falle des gleichen Umsatzes der eigene Name kleiner als der andere ist.

   Hinweis: Zum Vergleichen von Strings können Sie die Methode `int String.compareTo(String otherString)` verwenden. Diese Methode liefert einen Wert kleiner/gleich/größer als Null, sofern der erste String kleiner/gleich/größer als der zweite ist.

2. (3 Punkte) Schreiben Sie eine Methode

   ```
   void insert(Kunde k, List kl)
   ```

   welche einen Kunden in eine bereits sortierte (Teil-)Liste von Kunden entsprechend der Ordnung an der richtigen Stelle einfügt. Die Liste soll also nach Umsätzen absteigend und bei gleichen Umsatz nach Kundennamen aufsteigend geordnet sein. Der abstrakte Datentyp `List` ist dabei bereits vorgegeben (siehe unten) und braucht von Ihnen nicht implementiert zu werden.

   Beispiel: `insert(` (20, “Schmidt“), (9.999, “Meier“) → (20, “Huber“) → (1, “Mueller“)
   = (9.999, “Meier“) → (20, “Huber“) → (20, “Schmidt“) → (1, “Mueller“)

3. (3 Punkte) Schreiben Sie eine Methode

   ```
   List sort(List kl)
   ```

   welche eine Liste `kl` von Kunden gemäß des oben dargestellten Sortiertkriteriums sortiert zurückgibt. Die als Parameter übergebene Liste darf dabei beliebig verändert werden.

   Beispiel: `sort(` (20, “Huber“) → (9.999, “Meier“) → (1, “Mueller“) → (20, “Schmidt“) `)`
   = (9.999, “Meier“) → (20, “Huber“) → (20, “Schmidt“) → (1, “Mueller“)

Hinweis: Bei der Bearbeitung dieser Aufgabe können Sie voraussetzen, daß Ihnen die folgende Klasse `List` zur Verfügung steht.

```
class List {
    List();                  // leere Liste
    void insert(Object o);   // Einfuegen vor aktueller Position
    Object currentValue();   // aktuelles Element
    void remove();           // aktuelles Element entfernen
    void reset();            // zum Listenanfang
    void advance();          // zum naechsten Element
    boolean finished();      // Listenende erreicht?
    boolean isempty();       // Liste leer?
}
```