Aufgaben für das Fach Konzepte von Programmiersprachen (KPS-MD) im SS05

Inhaltsverzeichnis

Aufgabe 1: Rekursion und Iteration....2
Aufgabe 2: Kontextfreie (Typ2-) Grammatiken....5
Aufgabe 3: Zwei generische Einheiten in Java....6
Teilaufgabe 3.1 Eine Sammlung als verkettete Liste realisieren....7
Teilaufgabe 3.2 Eine Sammlung als Hash-Tabelle realsieren....8
Aufgabe 4: Eine generische Einheit (Schablone) in C++....9
Aufgabe 5: Ein selbstbezüglicher Satz....10

Aufgabe 1: Rekursion und Iteration

5 Punkte, Abgabe der Lösung: Do 14.04.05

Betrachten Sie die Datei `ReIterUnvollstaendig.java` (siehe unten). Fügen Sie am Anfang der Upros `iter01`, `iter02`, `reku03`, `reku04` und `reku05` je einen Anfangskommentar (**AKO**) ein, und zwar jeweils am Anfang des *Rumpfes*, d. h. nach der öffnenden geschweiften Klammer {). Dieser AKO soll die Frage „*Was macht dieses Unterprogramm?*“ möglichst einfach und verständlich beantworten (als simples Beispiel siehe das Upro `schritt01`). Hier ein paar Regeln dazu, wie man solche AKOs gestalten sollte:

Regel 1: Der AKO beginnt normalerweise mit einem Verb ("*Gibt ... aus*", "*Formatiert ...*" etc.).
Regel 2: Der AKO einer Funktion beinnt normalerweise mit "*Liefert ...*" (oder "*Gibt ... zurück*").
Regel 3: Im AKO kommt jeder *Parametername* mindestens einmal vor.
Regel 4: Der AKO beschreibt, *was* das Upro leistet, nicht *wie* es diese Leistung erbringt (ob durch Aufrufe weiterer Upros oder durch andere Befehle, ob iterativ oder rekursiv etc.).
Regel 5: Im AKO erwähnt man nichts, was schon davor ("in der ersten Zeile des Upros") steht. Z. B. schreibt man statt "Gibt den übergebenen `String`-Parameter namens `karl_heinz` aus und springt unmittelbar anschließend in die nächste Zeile" kürzer "Gibt `karl_heinz` und einen Zeilenwechsel aus".

Unterprogramme mit gleicher Nr (z. B. `iter01` und `reku01`, oder `reku03` und `iter03` etc.) sollen *exakt das Gleiche* leisten (aber auf verschiedene Weise: iterativ bzw. rekursiv). Ersetzen Sie die „unsinnigen“ Rümpfe der Upros `reku01`, `reku02`, `iter03`, `iter04` und `iter05` entsprechend.

```
1   // Datei ReIter01Unvollstaendig.java
2   ...
3   class ReIter01Unvollstaendig {
4      // ------------------------------------------------------------------------
5      static private void schritt01(char char01) {
6         // Gibt char01 aus.
7         AM.p(char01);
8      } // schritt01
9      // ------------------------------------------------------------ Aufgabe 01
10     static public void iter01(final int N) {
11        for (int i=1; i<=N; i++) {
12           schritt01('X');
13        }
14        AM.pln();
15     } // iter01
16     // ------------------------------------------------------------ Loesung 01
17     static public void reku01(final int N) {
18        // Ersetzen Sie diesen "unsinnigen" Rumpf
19        AM.pln("reku01, unsinniges Ergebnis!");
20     } // reku01
21     // ------------------------------------------------------------ Aufgabe 02
22     static public void iter02(final int N) {
23        int n = N;
24        while (n > 0) {
25           char ziff = (char) (n%2 + '0');
26           schritt01(ziff);
27           n = n/2;
28        }
29        AM.pln();
30     } // iter02;
31     // ------------------------------------------------------------ Loesung 02
32     static public void reku02(final int N) {
33        // Ersetzen Sie diesen "unsinnigen" Rumpf
34        AM.pln("reku02, unsinniges Ergebnis!");
```

```
35     } // reku02
36     // ------------------------------------------------------------ Aufgabe 03
37     static public void reku03(final int N) {
38        char ziff = (char) (N%2 + '0'); // "Rechteste" Binaerziffer
39        if        (N <  0) {
40           schritt01('-');
41           reku03(-N);
42        } else if (N == 0 || N == 1) {  // Hoechstwertige Ziffer
43           schritt01(ziff);
44           return;
45        } else {
46           reku03(N/2);
47           schritt01(ziff);
48        }
49     } // reku03
50     // ------------------------------------------------------------ Loesung 03
51     static public void iter03(final int N) {
52        // Ersetzen Sie diesen "unsinnigen" Rumpf
53        AM.p("iter03, unsinniges Ergebnis!");
54     } // iter03
55     // ------------------------------------------------------------ Aufgabe 04
56     static public String reku04(String s) {
57        if (s.length() <= 1) {
58           return s;
59        } else {
60           char   letzt     = s.charAt   (   s.length()-1);
61           String ohneLetzt = s.substring(0, s.length()-1);
62           return letzt + reku04(ohneLetzt);
63        }
64     } // reku04
65     // ------------------------------------------------------------ Loesung 04
66     static public String iter04(String s) {
67        // Ersetzen Sie diesen "unsinnigen" Rumpf
68        return "iter04, unsinniges Ergebnis!";
69     } // iter04
70     // ------------------------------------------------------------ Aufgabe 05
71     static public int reku05(int[] r1, int[] r2) {
72        return reku05a(r1, r2, 0);
73     }
74
75     static private int reku05a(int[] r1, int[] r2, int index) {
76        if (index >= r1.length || index >= r2.length) {
77           return r1.length - r2.length;
78        } else if (r1[index] != r2[index]) {
79           return r1[index] - r2[index];
80        } else {
81           return reku05a(r1, r2, index+1);
82        }
83     } // reku05a
84     // ------------------------------------------------------------ Loesung 05
85     static public int iter05(int[] r1, int[] r2) {
86        // Ersetzen Sie diesen "unsinnigen" Rumpf
87        return -999999999; // unsinniges Ergebnis
88     } // iter05
89     // ---------------------------------------------------------------------
90     static public void main(String[] _) {
91        ...
92     } // main
93     // ---------------------------------------------------------------------
94  } // class ReIter01Unvollstaendig
```

Die Datei `ReIter01Unvollstaendig.java` finden Sie am üblichen Ort. *Testen* Sie Ihre Lösung, indem Sie geeignete Befehle in die `main`-Methode schreiben und ausführen lassen.

Aufgabe 2: Kontextfreie (Typ2-) Grammatiken

5 Punkte, Abgabe der Lösung: Do 21.04.05

2.1. Geben Sie eine Grammatik für die folgende Sprache `Ahnen2` an:

```
{   die mutter von maria,                                        (*)
    der vater  von maria,
    die mutter von paul,
    der vater  von paul,
    die mutter der mutter von maria,
    die mutter des vaters von maria,
    der vater  der mutter von maria,
    der vater  des vaters von maria,
    die mutter der mutter von paul,                              (*)
    die mutter des vaters von paul,
    der vater  der mutter von paul,
    der vater  des vaters von paul,
    ...
    die mutter des vaters des vaters der mutter von maria,  (*)
    ...
    der vater des vaters des vaters der mutter von paul,
    ...
}
```

2.2. Geben Sie für die drei mit (*) gekennzeichneten Sätze *Ableitungen* aus Ihrer Grammatik an.

2.3. Geben Sie eine Grammatik für die folgende Sprache `Zahlen` an:

```
{
      null, ein, zwei, drei, vier, fuenf, sechs, sieben, acht, neun,
      zehn, elf, zwoelf, dreizehn, vierzehn,
      fuenfzehn, sechzehn, siebzehn, achtzehn, neunzehn,
      zwanzig, einundzwanzig, zweiundzwanzig, dreiundzwanzig, vierundzwanzig,
      ...
      fuenfundneunzig, ..., achtundneunzig, neunundneunzig,
      einhundert, einhundertundein, einhundertundzwei, ...
      ...
      einhundertundelf, einhundertundzwoelf, einhundertunddreizehn, ...
      ...
      einhundertundeinundzwanzig, einhundertundzweiundzwanzig, ...
      ...
      neunhundert, neunhundertundein, neunhundertundzwei, ...
      ...
      neunhundertundachtundneunzig, neunhundertundneunundneunzig
}
```

Zu dieser Sprache `Zahlen` sollen alle männlichen deutschen Zahlworte für die Zahlen von 0 bis 999 gehören, die z. B. auf einem Scheck vor dem Wort `Euro` stehen könnten (etwa so: `ein Euro` oder `einhundertundein Euro` etc.).

2.4. Geben Sie für die Worte der Zahlen 0, 17, 99, 123 und 999 Ableitungen aus Ihrer Grammatik an.

Tip: Verwenden Sie als *Zwischensymbole* systematische Namen wie `z000_009`, `z010_019`, `z000_099`, `z010_090` etc. und als Startsymbol `z000_999`. Damit ist die Grammatik leichter zu schreiben und zu lesen.

Aufgabe 3: Zwei generische Einheiten in Java

5 Punkte, Abgabe der Lösung: Do 28.04.05

Eine *Sammlung* ist ein Objekt, in dem man andere Objekte sammeln (hineintun, suchen, wieder entfernen) kann. Die Schnittstelle `SammelbarG` (*G* wie *generisch*) beschreibt, welche Objekte *sammelbar* sind:

```
1   interface SammelbarG<S extends Comparable<? super S>> {
2      S getSlue();
3      // Liefert den Schluessel von diesem Objekt.
4   } // interface SammelbarG
```

Ein Objekt ist sammelbar, wenn es einen Schlüssel eines Typs `S` enthält und eine Methode `getSlue`, die diesen Schlüssel liefert. `<S extends Comparable<? super S>>` bedeutet: Jeder Schlüssel des Typs `S` muss mindestens mit jedem anderen Schlüssel (oder sogar mit jedem Objekt einer Oberklasse von `S`) vergleichbar sein mit der Methode `compareTo` der Schnittstelle `Comparable`.

`SammelbarG`-Objekte kann man in Objekten sammeln, die die Schnittstelle `SammlungG` („G“ wie generisch“) implementieren:

```
5   // Datei SammlungG.java
6   /* ---------------------------------------------------------------------
7   In einem SammlungG-Objekt kann man SammelbarG-Objekte sammeln,
8   d.h. einfuegen, suchen und entfernen. Ausserdem kann man die
9   Anzahl der momentan in der Sammlung vorhandenen Komponenten
10  ermitteln (mit size) und all diese Komponenten ausgeben (mit
11  print).
12
13  Jedes SammelbarG-Objekt muss eine Methode namens getSlue ent-
14  halten, die den Schluessel des Objekts liefert. Dieser Schlues-
15  sel muss mit anderen Schluesseln (oder sogar mit allen Objekten
16  eines Obertyps des Schluesseltyps S) vergleichbar sein (mit
17  der Methode compareTo der Schnittstelle Comparable).
18
19  Diese Schnittstelle SammlungG legt NICHT fest:
20  1. Ob doppelte Schluessel erlaubt oder verboten sind. Jede
21     Implementierung sollte diesen Punkt deutlich dokumentieren.
22  2. Ob eine Sammlung eine feste Groesse hat (wie eine Reihung),
23     oder praktisch unbegrenzt gross ist (wie eine verkettete
24     Liste oder ein Baum).
25
26  Bekannte Implementierungen dieser Schnittstelle:
27  ArrayListUn_Sort_G, BinBaumA_G, HashTabA_G,
28  ArrayListSortiertG, BinBaumB_G, HashTabB_G.
29  VerketteteListeG.
30  --------------------------------------------------------------------- */
31  interface SammlungG
32     <  S extends Comparable<? super S>, //  Schluessel-Typ
33        K extends SammelbarG<S>          // Komponenten-Typ
34     >
35  {
36     public boolean fuegeEin(K kompoNeu);
37        // Versucht, die Komponente kompoNeu in diese SammlungG einzufuegen.
38        // Liefert true, wenn das gelingt. Liefert false, wenn das Einfuegen
39        // nicht gelingt (weil diese Sammlung eine feste Groesse hat und
40        // schon voll ist oder weil diese Sammlung doppelte Schluessel ver-
41        // bietet und sich schon ein Objekt mit gleichem Schluessel darin
42        // befindet). Wirft eine Ausnahme des Typs NullPointerException,
43        // wenn kompoNeu gleich null ist.
```



```
44     public K       suche   (S schluessel);
45        // Liefert null, wenn es in dieser Sammlung keine Objekte mit dem
46        // angegebenen Schluessel gibt. Liefert sonst eines dieser Objekte.
47     public boolean entferne(S schluessel);
48        // Liefert false, wenn es in dieser Sammlung keine Objekte mit dem
49        // angegebenen Schluessel gibt. Entfernt sonst eines dieser Objekte
50        // aus dieser Sammlung und liefert true.
51     public void    print   ();
52        // Gibt diese Sammlung in lesbarer Form zur Standardausgabe aus
53        // (auch dann, wenn die Sammlung leer ist).
54     public int     size    ();
55        // Liefert die Anzahl der Komponenten, die sich momentan in dieser
56        // Sammlung befinden.
57  } // interface SammlungG
58  // ---------------------------------------------------------------------
```

In einem `SammlungG`-Objekt kann man Komponenten eines Typs `K` sammeln, die die Schnittstelle `SammelbarG<S>` implementieren (Zeile 9). Dabei muss `S` ein geeigneter Schlüsseltyp sein (d. h. `S`-Objekte müssen miteinander vergleichbar sein mit der Methode `compareTo` der Schnittstelle `Comparable`, wie in Zeile 8 verlangt).

Es folgt ein Beispiel einer `SammelbarG`-Klasse. Objekte dieser Klasse kann man in geeigneten `SammlungG`-Objekten sammeln:

```
59  class KompoG implements SammelbarG<String> {
60     private String schluessel;
61     private String daten;
62     // ------------------------------------------------------------------
63     KompoG(String schluessel, String daten) {
64        this.schluessel = schluessel;
65        this.daten      = daten;
66     } // Konstruktor KompoG
67
68     static int lfdNr = 0;
69
70     KompoG() {
71        lfdNr++;
72        schluessel = "Schluessel "        + lfdNr;
73        daten      = "Daten Daten Daten " + lfdNr;
74     } // Standard-Konstruktor KompoG
75     // ------------------------------------------------------------------
76     public String getSlue() {return schluessel;}
77     // ------------------------------------------------------------------
78     public String toString() {
79        return "KompoG, Slue: " + schluessel + ", Daten: " + daten;
80     } // toString
81     // ------------------------------------------------------------------
82     // Eine "vernuenftige" equals-Methode ist noetig, damit beim Testen die
83     // Gleichheit zweier KompoG-Objekte (die nicht identisch sind) richtig
84     // erkann werden kann:
85     public boolean equals(KompoG k) {
86        return this.schluessel.equals(k.schluessel) &&
87               this.daten     .equals(k.daten);
88     } // equals
```

Jedes `KompoG`-Objekt hat einen Schlüssel des Typs `String` und eine Methode `getSlue`, die diesen Schlüssel liefert. Die Klasse `String` implementiert die Schnittstelle `Comparable<String>` (d. h. man kann jeden String mit jedem String vergleichen mit der Methode `compareTo` der Schnittstelle `Comparable`). Also implementiert die Klasse `KompoG` die Schnittstelle `SammelbarG<String>`.

Teilaufgabe 3.1 Eine Sammlung als verkettete Liste realisieren

Schreiben Sie eine generische Klasse `VerketteteListeG<S ..., K ....>` (die Auslassungen `"..."` müssen Sie durch geeignete Einschränkungen ersetzen). Dieses Klasse soll die Schnittstelle `SammlungG<S, K>` implementieren und die Sammlung als *verkettete Liste* realisieren. Doppelte Schlüssel sollen in einer solchen Sammlung erlaubt sein.

Jedes Objekt dieser Klasse soll zwei (Objekt-) Attribute namens `kompo` und `rest` enthalten, die auf eine Komponente der Sammlung bzw. auf das nächste `VerketteteListeG`-Objekt („den Rest der Liste“) zeigen (oder den Wert `null` haben).

Hinweis 1: Eine *leere Liste* soll (nicht etwa aus null Objekten sondern) aus *einem* `VerketteteListeG`-Objekt bestehen, bei dem `kompo` und `rest` beide den Wert `null` haben . Dieses Objekt wird manchmal als `Dummyobjekt` ("Strohmannobjekt") bezeichnet, weil es nur zur Verwaltung der Liste dient, aber nicht zu den "richtigen Listenobjekten" zählt. Wenn die Liste *nicht mehr leer* ist, sollte das Attribut `rest` des Dummyobjekts auf das *erste richtige Listenobjekt* zeigen (und das `rest`-Attribut dieses ersten Listenobjekts auf den *Rest der Liste*, d.h. auf das *zweite* Listenobjekt etc.). Das *letzte* Listenobjekt sollte man immer daran erkennen können, dass sein `rest`-Attribut den Wert `null` hat.

Das `kompo`-Attribut des Dummyobjekts wird nicht verwendet und sollte immer den Wert `null` behalten. Bei allen richtigen Listenobjekten sollte das `kompo`-Attribut immer einen Referenzwert *ungleich* `null` haben, der auf ein `SammelbarG`-Objekt zeigt.

Hinweis 2: Bevor Sie mit dem Entwerfen des Programms und dem Programmieren beginnen, sollten Sie unbedingt eine *leere Liste* als *Boje* darstellen und *mit Papier und Bleistift* in diese Liste 2 bis 3 Objekte *einfügen* (am einfachsten immer ganz *vorn*, unmittelbar hinter dem Dummyobjekt). Zeichnen Sie dann noch einmal ("sauber und vorzeigbar") *eine verkettete Liste*, die aus einem Dummyobjekt und mindestens *3 "richtigen Objekten"* besteht, in Bojendarstellung. Diese Bojendarstellung sollten Sie als "anschauliche Grundlage" für Ihre Programmierung verwenden und auch in den Übungen immer bei sich haben.

Hinweis 3: Was erscheint auf dem Bildschirm, wenn man mit der *print*-Methode Ihrer Klasse *VerketteteListe* eine *leere* Sammlung ausgibt? Ist diese Ausgabe für den Benutzer lesbar und verstehbar?

Testen Sie Ihre Lösung zuerst mit einer selbstgeschriebenen `main`-Methode und anschliessend mit den Testprogrammen `SammlungG_Tst01` bis `SammlungG_Tst03`.

Teilaufgabe 3.2 Eine Sammlung als Hash-Tabelle realsieren

Diese Aufgabe ist eine Variante der vorigen Aufgaben. Schreiben Sie eine Klasse `HashTabG<S, K>`, die die Schnittstelle `SammlungG<S, K>` implementiert und die Sammlung als *Hashtabelle* realisiert. Doppelte Schlüssel sollen in einer solchen Sammlung erlaubt sein.

Besonders *interessant* (und erstaunlicherweise auch *einfach*) wird Ihre Lösung, wenn Sie die Hashtabelle als `ArrayList`-Objekt mit Komponenten der Klasse `VerketteteListeG<S, K>` darin vereinbaren. Nur die Methode `print` "macht ein bisschen Probleme", für die es aber mehrere Lösungen gibt. Die Klasse `VerketteteListeG` haben Sie bereits als Lösung der vorigen Teilaufgabe entwickelt.

Hinweis 1: Falls beim Testen Ihrer Lösung eine `NullPointerException` auftritt, sollten Sie Ihre *Hashtabelle* (d.h. das `ArrayList`-Objekt mit Komponenten der Klasse `VerketteteListeG` darin) als *Boje* darstellen. Haben Sie wirklich die richtige Anzahl von Objekten der Klasse `VerketteteListeG` erzeugt (mit dem Befehl `new VerketteteListe`)?

Hinweis 2: Was erscheint auf dem Bildschirm, wenn man mit der `print`-Methode Ihrer Klasse `HashTabG` eine `leere` Sammlung ausgibt? Ist diese Ausgabe für den Benutzer lesbar und verstehbar?

Testen Sie Ihre Lösung für *diese Teilaufaufgabe* genau so wie Ihre Lösung für die vorigen Teilaufgaben (zuerst mit einer selbst-programmierten main-Methode und dann mit den Testprogrammen `SammlungG_Tst01` bis `SammlungG_Tst01`).

Aufgabe 4: Eine generische Einheit (Schablone) in C++

5 Punkte, Abgabe der Lösung: Do 12.05.05

Programmieren Sie in C++ eine Funktionsschablone namens `ganzToString` entsprechend der folgenden Deklaration:

```
1   template<class GanzTyp>
2   std::string ganzToString(GanzTyp g, int basis=10, int grupLen=3);
```

Die Schablone soll nur mit *Ganzzahltypen* instanziierbar sein. Versuche, sie mit anderen Typen zu instanziieren, sollen zur Compilezeit eine Fehlermeldung auslösen.

Jede Instanz dieser Schablone (d. h. jede Funktion `ganzToString`) soll ihren ersten Parameter (die Ganzzahl `g`) in einen "gut lesbaren String" umwandeln, etwa so:

```
3    short s01 = 12345;
4    short s02 = -12345;
5    short s03 = 255;
6    short s04 = -0;
7    int   i01 = 123456789;
8    int   i02 = -1000000000;
9
10   ganzToString(s01)        ist gleich "+12_345"
11   ganzToString(s02)        ist gleich "-12_345"
12   ganzToString(s03,  2)    ist gleich "+11_111_111"
13   ganzToString(s03,  2, 4) ist gleich "+1111_1111"
14   ganzToString(s03, 16)    ist gleich "+FF"
15   ganzToString(s04)        ist gleich "+0";
16   ganzToString(i01)        ist gleich "+123_456_789"
17   ganzToString(i01, 10, 4) ist gleich "+1_2345_6789"
18   ganzToString(i02)        ist gleich "-1_000_000_000"
```

Der Parameter `basis` gibt an, in welchem *Zahlensystem* die Zahl `g` dargestellt werden soll: im 2-er-System (binär) oder im 5-er-System (quintal) oder im 8-er-System (octal) oder im 16-er-System (hexadezimal) oder (ganz exotisch :-) im 10-er-System (dezimal) oder ... etc.

Als `basis` sind nur die Zahlen 2 bis 16 erlaubt. Falls `basis` einen anderen Wert hat, soll statt dessen `10` genommen werden.

"Gruppen von Ziffern" sollen durch je einen Unterstrich `'_'` voneinander getrennt werden. Wie viele Ziffern zu einer Gruppe gehören, wird durch den Parameter `grupLen` angegeben. Falls `grupLen` einen unplausiblen Wert hat (kleiner als `1` oder größer als `10`) soll stattdessen der Wert `3` genommen werden.

Der Ergebnisstring soll immer mit einem *Vorzeichen* beginnen (`'+'` bzw. `'-'`), mindestens *eine* Ziffer, aber *keine unnötigen führenden Nullen* enthalten.

Aufgabe 5: Ein selbstbezüglicher Satz

5 Punkte, Abgabe der Lösung: Do 19.05.05

Betrachten Sie den folgenden Satz, der etwas über sich selbst aussagt:

```
Dieser Satz enthält
genau n0 Mal die Ziffer 0,
genau n1 Mal die Ziffer 1,
genau n2 Mal die Ziffer 2,
genau n3 Mal die Ziffer 3,
genau n4 Mal die Ziffer 4,
genau n5 Mal die Ziffer 5,
genau n6 Mal die Ziffer 6,
genau n7 Mal die Ziffer 7,
genau n8 Mal die Ziffer 8 und
genau n9 Mal die Ziffer 9.
```

Sie sollen die Variablen `n0` bis `n9` so durch natürliche Zahlen ersetzen, dass der Satz zutrifft.

Schreiben Sie ein Programm (wahlweise in Java oder C/C++), welches geeignete natürliche Zahlen berechnet und ausgibt. Abzugeben ist das Programm und die ausgegebene Lösung.