

Vorname	Nachname	Matrikel-Nr
---------	----------	-------------

Aufgabe 1 (15 Punkte): Betrachten Sie die folgende (rekursive!) Funktion:

```

1  static int reku(int n, int anz) {
2      if (n <= 1) {
3          return anz;
4      } else if (n % 2 == 0) {
5          return reku(n/2, anz+1);
6      } else {
7          return reku(3*n+1, anz+1);
8      } // if
9  } // reku

```

Berechnen Sie die Werte der drei Ausdrücke (der Funktionsaufrufe) **reku(3, 0)**, **reku(40, 8)** und **reku(7, 0)**, indem Sie die Funktion **reku** entsprechend mit Papier und Bleistift ausführen.

Aufgabe 2 (20 Punkte): Schreiben Sie eine rekursive Prozedur **gibTeilerAus** entsprechend der folgenden Beschreibung:

```

1  static void gibTeilerAus(int zahl, int ab) {
2      // Verlaesst sich darauf, dass die Parameter zahl und ab groesser
3      // als 1 sind. Gibt alle Teiler von zahl, die groesser oder gleich ab
4      // sind ("alle Teiler ab ab"), aufsteigend sortiert zum Bildschirm aus.
5      //
6      // Beispiel: Der Aufruf gibTeilerAus(12, 4) gibt folgende Zeile aus:
7      // 4 6 12
8      // Beispiel: Der Aufruf gibTeilerAus(12, 2) gibt folgende Zeile aus:
9      // 2 3 4 6 12
10     // Beispiel: Der Aufruf gibTeilerAus(17, 2) gibt folgende Zeile aus:
11     // 17
12     // Beispiel: Der Aufruf gibTeilerAus(17, 18) gibt eine leere Zeile aus.
13     //
14     // Beispiel: Der Aufruf gibTeilerAus(32, 2) gibt folgende Zeile aus:
15     // 2 4 8 16 32
16     ...
17 } // gibTeilerAus

```

Ihre Lösung darf **keine Schleife** enthalten, sondern muss "rein **rekursiv**" funktionieren.

Aufgabe 3 (15 Punkte): Betrachten Sie die folgende **Grammatik für Ausdrücke**:

R01: Aus1 -> Aus1 '+' Aus2	R04: Aus2 -> Aus2 '*' Aus3	R07: Aus3 -> 'x'
R02: Aus1 -> Aus1 '-' Aus2	R05: Aus2 -> Aus2 '/' Aus3	R08: Aus3 -> 'y'
R03: Aus1 -> Aus2	R06: Aus2 -> Aus3	R09: Aus3 -> 'z'
		R10: Aus3 -> '(' Aus1 ')'

Dabei sind **Aus1**, **Aus2** und **Aus3** Zwischensymbole und alle anderen Symbole Endsymbole, **Aus1** ist das Startsymbol. Geben Sie einen **Syntaxbaum** für das Wort **(x + y) * z** an.

Achtung: Als Lösung dieser Aufgabe sollen Sie einen **Syntaxbaum** angeben und **nicht** eine **Ableitung**.**Aufgabe 4** (20 Punkte): In einigen Programmiersprachen gibt es einen Datentyp, zu dem die Ganzzahlen zwischen 0 und 255 gehören. Geben Sie eine Grammatik an, aus der man die 256 dezimalen Ganzzahlen zwischen 0 und 255 (ohne unnötige führende Nullen) ableiten kann. Das Startsymbol dieser Grammatik sollte **Byte** heissen und es empfiehlt sich, weitere Zwischensymbole wie **Ziff0bis9**, **Ziff1bis9** und **Ziff1bis2** zu verwenden.**Aufgabe 5** (20 Punkte): Stellen Sie sich vor, dass Sie ein Softwaresystem entwickeln, in dem auch vier verschiedene **Sammlungen** (von Objekten mit Schlüsseln) eine wichtige Rolle spielen werden. Sie wissen, dass die vier Sammlungen (wenn das System fertig ist und läuft) wie folgt benutzt werden:**Sammlung S1:** Die Größe dieser Sammlung wird stark schwanken, vermutlich zwischen 1000 und 100 000 Komponenten. Es müssen oft neue Komponenten eingefügt und später wieder entfernt werden. Gesucht werden muss in dieser Sammlung fast nie, aber häufig müssen alle Komponenten nach ihren Schlüsseln sortiert ausgegeben werden. Welche Sammlungsform würden Sie für S1 wählen? Nennen Sie die Sammlungsform und begründen Sie Ihre Wahl **kurz** (maximal etwa 5 Zeilen).**Sammlung S2:** Diese Sammlung soll etwa 100 000 Komponenten aufnehmen. Es wird nur sehr selten nötig sein, neue Komponenten in diese Sammlung einzufügen oder vorhandene Komponenten zu entfernen und es müssen nie alle Komponenten ausgegeben werden. In dieser Sammlung wird aber sehr häufig gesucht werden (nach einer Komponenten anhand ihres Schlüssels). Welche Sammlungsform würden Sie für S2 wählen? Nennen Sie die Sammlungsform und begründen Sie Ihre Wahl **kurz** (maximal etwa 5 Zeilen).**Sammlung S3:** Diese Sammlung soll ebenfalls etwa 100 000 Komponenten aufnehmen und muss nur sehr selten verändert werden. Jede Komponente wird eine Kunden-Nr als Schlüssel und die Anschrift und ähnliche Stammdaten des Kunden enthalten. Die Kunden-Nrn sind Zahlen zwischen 1 und etwa 100 000, von denen fast alle (ca. 95 %) mit einem Kunden belegt und nur sehr wenige (ca. 5 %) noch frei verfügbar sind (z.B. für Neukunden). In dieser Sammlung muss fast nur gesucht werden (nämlich zu einer Kunden-Nr die Anschrift des Kunden). Welche Sammlungsform würden Sie für S3 wählen? Nennen Sie die Sammlungsform und begründen Sie Ihre Wahl **kurz** (maximal etwa 5 Zeilen).**Sammlung S4:** Diese Sammlung soll nur etwa 10 bis 20 Komponenten aufnehmen. Wie oft die einzelnen Grundoperationen (einfügen, suche, entfernen) ausgeführt werden müssen, ist zur Zeit leider noch nicht vorherzusagen. Welche Sammlungsform würden Sie für S4 wählen? Nennen Sie die Sammlungsform und begründen Sie Ihre Wahl **kurz** (maximal etwa 5 Zeilen).**Aufgabe 6** (10 Punkte):6.1. Welche **Tiefe** muss ein binärer Baum **mindestens** haben, damit er 10 000 Knoten enthalten kann?6.2. Welche **Tiefe** kann ein binärer Baum mit 10 000 Knoten **höchstens** haben?6.3. Was muss für die Knoten K_i eines binären Baumes und die Schlüssel S_i dieser Knoten gelten, damit der Baum **sortiert** ist? Vervollständigen Sie den Satz: "Ein Baum ist sortiert wenn ...!". Sie dürfen voraussetzen, dass kein Schlüssel mehr als einmal im Baum vorkommt.6.4. Geben Sie zu jeder der folgenden Zahlen eine entsprechende **normalisierte** Zahl an:

- A: 1234.56 E 5
- B: 1234.56 E-5
- C: 0.00123 E 3
- D: 0.00123 E-3
- E: 1.2345 E 17

Lösung 1 (15 Punkte):

```

1 reku( 3, 0): 3 10 5 16 8 4 2 1 --> 7
2 reku(40, 8): 40 20 10 5 16 8 4 2 1 --> 16
3 reku( 7, 0): 7 22 11 34 17 52 26 13 40 20 10 5 16 8 4 2 1 --> 16

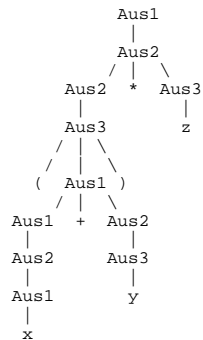
```

Lösung 2 (20 Punkte):

```

1 static void gibTeilerAus(int zahl, int ab) {
2     // Verlaesst sich darauf, dass die Parameter zahl und ab groesser
3     // als 1 sind. Gibt alle Teiler von zahl, die groesser oder gleich ab
4     // sind ("alle Teiler ab ab"), zum Bildschirm aus.
5
6     if (ab > zahl) {
7         System.out.println();
8         return;
9     } else if (zahl % ab == 0) {
10        System.out.print(ab + " ");
11    }
12    gibTeilerAus(zahl, ab+1);
13 } // gibTeilerAus

```

Lösung 3 (15 Punkte): Ein Syntaxbaum für das Wort $(x + y) * z$:**Lösung 4** (20 Punkte): Eine Grammatik für die 256 Dezimalzahlen von 0 bis 255 (ohne unnötige führende Nullen):

```

R01: Byte    -> Ziff0Bis9
R02: Byte    -> Ziff1Bis9 Ziff0Bis9
R03: Byte    -> 1      Ziff0Bis9 Ziff0Bis9
R04: Byte    -> 2      Ziff0bis4 Ziff0bis9
R05: Byte    -> 2      5      Ziff0bis5

```

```

R06: Ziff0Bis9 -> '0'
R07: Ziff0Bis9 -> Ziff1Bis9

```

```

R08: Ziff1Bis9 -> '1'
R09: Ziff1Bis9 -> '2'
R10: Ziff1Bis9 -> '3'
R11: Ziff1Bis9 -> '4'
R12: Ziff1Bis9 -> '5'
R13: Ziff1Bis9 -> '6'
R14: Ziff1Bis9 -> '7'
R15: Ziff1Bis9 -> '8'
R16: Ziff1Bis9 -> '9'

```

```

R17: Ziff0Bis5 -> Ziff0bis4
R18: Ziff0Bis5 -> '5'

```

```

R19: Ziff0Bis4 -> '0'
R20: Ziff0Bis4 -> '1'
R21: Ziff0Bis4 -> '2'
R22: Ziff0Bis4 -> '3'
R23: Ziff0Bis4 -> '4'

```

Lösung 5 (20 Punkte):

5.1. **Sammlung S1: Baum.** Ein Baum unterstützt das sortierte Ausgeben aller Komponenten gut und erlaubt relativ schnelles Einfügen, Suchen und Entfernen.

5.2. **Sammlung S2: Hash-Tabelle.** Diese Wahl ist hier möglich, weil nicht auf alle Komponenten in sortierter Reihenfolge zugegriffen werden muss. Eine gute Hash-Tabelle kann schneller sein als ein Baum.

5.3. **Sammlung S3:** Eine **Reihung** mit etwa 100 000 Komponenten. Die Kunden-Nrn werden als Indizes benützt. Etwa 5 % Komponenten der Reihung werden den Wert null haben, aber das ist eine tolerable "Speicherverschwendung". Eine solche Reihung ist die schnellste Sammlung überhaupt.

5.4. **Sammlung S4:** Z.B. eine **unsortierte Reihung** oder ein **Vektor**. Bei einer so kleinen Sammlung ist die Organisationsform unwichtig, weil die Vor- und Nachteile der verschiedenen Formen erst bei größeren Sammlungen zum Tragen kommen.

Lösung 6 (10 Punkte):

6.1. Mindestens die Tiefe **14** (für 10 000 Knoten).

6.2. Höchstens die Tiefe **10 000** (entartet zu einer Liste).

6.3. Ein Baum ist sortiert, wenn für **jeden** Knoten K_i mit Schlüssel S_i gilt:
Alle Schlüssel im **linken** Unterbaum von K_i sind **kleiner** als S_i und
alle Schlüssel im **rechten** Unterbaum von K_i sind **größer** als S_i .

6.4. Geben Sie zu jeder der folgenden Zahlen eine entsprechende **normalisierte** Zahl an:

A: 1234.56 E 5	1.23456 E 8
B: 1234.56 E-5	1.23456 E-2
C: 0.00123 E 3	1.23 E 0
D: 0.00123 E-3	1.23 E-6
E: 1.2345 E 17	1.2345 E 17 (war schon normalisiert)