Lehrstuhl für Praktische Informatik IV
Holger Füßler
A5, 6, Raum B 219
68131 Mannheim
Telefon: (0621) 181–2605
Email: fuessler@informatik.uni-mannheim.de

Robert Schiele
B6, 29, Raum C0.04
68131 Mannheim
Telefon: (0621) 181–2214
Email: rschiele@uni-mannheim.de

UNIVERSITÄT MANNHEIM

Praktische Informatik I — 10. Übungsblatt
Wintersemester 2005/2006 — Abgabe: 18. Januar 2006

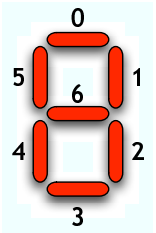
Allgemeine Bearbeitungshinweise

Verwenden Sie für die Programmieraufgaben die in den Aufgaben angegebenen Klassennamen und geben Sie den Klassen-Quellcode sowohl auf Papier als auch per e-mail bei Ihrem Tutor ab. Falls nicht anders gefordert, lagern Sie sinnvoll Funktionalität in Methoden aus. Verwenden Sie Javadoc-Kommentare zur Dokumentation.
Aufgaben, bei denen die Punktzahl mit einem $*$ gekennzeichnet ist, zählen als Zusatzaufgaben zum Punktekonto, d.h. sie werden für die Berechnung der Zulassungsvoraussetzung im Zähler gewertet, aber nicht im Nenner.

Aufgabe 1 — 9 Punkte

Aufgabe 1 a) — 4 Punkte

Zur Darstellung der Ziffern $0 - -9$ wird in elektronischen Geräten oft eine sogenannte Sieben-Segment-Anzeige verwendet (siehe Bild).



Gegeben sei eine Zahl zwischen 0 und 9, die mit 4 Bits $B_0, \ldots, B_3$ dargestellt ist, also

	B_3	B_2	B_1	B_0
0	false	false	false	false
1	false	false	false	true
⋮	⋮	⋮	⋮	⋮
9	true	false	false	true

Stellen Sie nun für jedes der sieben Segmente eine Logik-Formel F_0 bis F_6 auf, die aus den vier Eingangs-Bits entscheidet, ob das Segment leuchten soll (true) oder nicht (false).

Aufgabe 1 b) — 5 Punkte

Laden Sie die Klasse `pi1.blatt9.SevenSegment` von der Übungsseite herunter. Als Startklasse geladen, gibt sie die Ziffern $0-9$ in „Sieben-Segment-Facon“ aus. Ferner bietet sie die Methode `void sevenSegOut(boolean vec[])`, die einen boolean Vektor der Länge 7 erwartet und diesen entsprechend ausgibt.
Schreiben Sie nun eine Klasse `pi1.blatt9.SegTester`, deren Methode `boolean[] map(int i)` die Zuordnung der Zahl i auf den Segment-Vektor berechnet. Benutzen Sie hierfür die oben hergeleiteten Logik-Funktionen. Schreiben Sie eine entsprechende main-Methode zum Testen.
Hinweis: Sie Können die SevenSegment-static-Methode `boolean getNthBit(int number, int n)`, die den Bool'schen Wert des n-ten Bits von number zurückgibt.

Aufgabe 2 — 11 Punkte

Aufgabe 2 a) — 7 Punkte

Implementieren Sie in Java ein einfaches Programm, welches automatisch die Lösung für das Rätsel mit den fünf Häusern vom ersten Übungsblatt findet.
Sie können hierfür zur Unterstützung die Java-Library verwenden, die Sie auf der Übungshomepagefinden. Mit dieser Library können Sie in einer von `logelei.Loesung` abgeleiteten Klasse mit den Methoden `permutierePersonen()`, `permutiereFarben()`, `permutiereGetraenke()`, `permutiereGerichte()` und `permutiereTiere()` automatisch alle Permutationen der entsprechenden Elemente auf die fünf Häuser aufzählen. Schreiben Sie also zum Beispiel eine Schleife `while (permutierePersonen()) { ...}`, so erhalten Sie innerhalb dieser `while`-Schleife in jedem Schleifendurchgang jeweils eine Permutation. Die `while`-Schleife terminiert automatisch, wenn alle Permutationen aufgezählt sind. Mit den Methoden `bei()`, `neben()` und `nach()` können Sie prüfen, ob zwei Eigenschaften im selben Haus, in benachbarten Häusern oder in benachbarten Häusern in bestimmter Reihenfolge auftreten. Die einzelnen Elemente können dabei mit `Person.Finne`, `Getraenk.Milch`, usw., die einzelnen Häuser mit ihrer Hausnummer(1–5) angesprochen werden.
Implementieren Sie den Algorithmus einfach, indem Sie alle Permutationen in ineinander verschachtelten Schleifen aufzählen und dann alle Bedingungen abprüfen. Haben Sie eine gültige Lösung gefunden, so geben Sie diese einfach durch Ausgabe des Objekts auf dem Bildschirm aus.

Aufgabe 2 b) — 4 Punkte

Geben Sie die Laufzeitkomplexität Ihrer Lösung an. Welchen offensichtlichen Nachteil hat diese einfache Lösung? Mit welcher Technik können Sie diesen Nachteil beseitigen oder zumindest abmildern? Verbessern Sie Ihre Lösung unter diesem Aspekt. Ändert

diese Technik etwas an der Laufzeitkomplexität?

Aufgabe 3 — 10 Punkte

Aufgabe 3 a) — 6 Punkte

In der Vorlesung wurde die Rechenregel (7) des O-Kalküls formal bewiesen. (Siehe Folie 5a-29) Beweisen Sie analog die Rechenregeln (4) und (6) von Folie 5a-28.

Aufgabe 3 b) — 4 Punkte

Zeigen Sie ebenso formal, dass gilt: $n \log^2 n \in O(n^2)$.

Aufgabe 4 — $*$ 10 Punkte

Unter der Primfaktorzerlegung einer Ganzzahl $\nu > 0$ versteht man die Aufspaltung dieser Zahl in p von 1 verschiedene Primfaktoren (also Faktoren, die Primzahlen sind) $\pi_1, \ldots, \pi_p$, so dass gilt

$$\nu = \pi_1 \cdot \pi_2 \cdot \ldots \cdot \pi_{p-1} \cdot \pi_p$$

Zum Beispiel wäre die Primfaktorzerlegung der Zahl $210 = 2 \cdot 3 \cdot 5 \cdot 7$, wohingegen 23 die "Zerlegung" 23 hat, da es sich bereits um eine Primzahl handelt. Zur Erinnerung: eine Primzahl ist eine Zahl, die ganzzahlig außer durch sich selbst nur durch 1 teilbar ist.

Aufgabe 4 a) — 4 Punkte

Schreiben Sie eine **rekursive** Java-Methode `void zerlege(...)` in einer Klasse `PrimZerleger`, die die Primfaktoren einer als Argument übergebenen positiven Ganzzahl in aufsteigender Größe ausgibt.
Hinweise:

- Nutzen Sie den Zusammenhang aus, dass nach dem Teilen einer Zahl durch einen ihrer Primfaktoren wieder eine Zahl entsteht, auf die man eine Primfaktorzerlegung anwenden kann.
- Bestimmen Sie einzelne Primfaktoren durch geschicktes "Ausprobieren".
- Genau dann, wenn eine Zahl n eine Zahl p ganzzahlig teilt, ist der Rest der Ganzzahldivision von $p \mod n = 0$.

Aufgabe 4 b) — 0,5 Punkte

Bei welchen Zahlen r bekommen Sie die größte Rekursionstiefe im Vergleich mit ähnlich großen Zahlen und wie groß ist diese in Abhängigkeit von r? Begründen Sie ohne Beweis.

Aufgabe 4 c) 0,5 Punkte

Bei welchen Zahlen v (wieder im Vergleich mit ähnlich großen Zahlen) sind die meisten Faktor-Tests zu erwarten und wie viele sind es? Begründen Sie ohne Beweis.

Aufgabe 4 d) 3 Punkte

Ein berühmtes und sehr altes Verfahren zur Primzahlbestimmung ist das „Sieb des Eratosthenes“ (siehe `http://de.wikipedia.org/wiki/Sieb_des_Eratosthenes`). Schreiben Sie eine Implementierung `EraPrimTester` des Interfaces `PrimTester`, deren Methode `isPrime(long p)` die Überprüfung vornimmt, ob eine Zahl p eine Primzahl ist. Die Methode soll nach dem Prinzip des „Sieb des Eratosthenes“ vorgehen. Beachten Sie bitte, dass die vollst. Initialisierung des Siebes im Konstruktor durchgeführt werden soll. Sie dürfen hier annehmen, dass p höchstens 10000 groß wird (Trotzdem soll die Obergrenze zum Zeitpunkt der Objektinstanziierung festgelegt werden können).

```
1 package pi1.blatt9;

  public interface PrimTester {
      public boolean isPrime(long p);
  }
```

Aufgabe 4 e) 2 Punkte

Gegeben sei die folgende Klasse `SimplePrimTester`, die einen sehr einfachen Primzahltest implementiert:

```
package pi1.blatt9;

public class SimplePrimTester implements PrimTester {
    public boolean isPrime(long p){
5       if (p == 1)
            return true;
        for (int n = 2; n < Math.sqrt(p) + 1; n++) {
            if (p % n == 0)
                return false;
10      }
        return true;
    }
}
```

Geben Sie für folgende Algorithmen zum Test von p auf die Primzahleigenschaft die Laufzeiten und den Hauptspeicherbedarf in O-Notation an und diskutieren sie kurz, wann sie gut geeignet sind.

1. Sieb des Eratosthenes, inkl. Initialisierung
2. Sieb des Eratosthenes, bis P bereits initialisiert und $p \leq P$
3. SimplePrimTester, beliebige p