Philipps-Universität Marburg
Fachbereich Mathematik und Informatik
Arbeitsgruppe Datenbanksysteme

Wintersemester 2013/2014
Montag, 11. November 2013

Prof. Dr. Bernhard Seeger
Dipl.-Inf. Marc Seidemann
Johannes Drönner, M.Sc.

Übungen zur Vorlesung

Praktische Informatik I

Blatt 4

Abgabe: Montag, 18.11.2013,
bis **spätestens** 10:00 Uhr
über die ILIAS-Plattform

Hinweise:

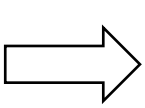
- Vergessen Sie nicht Ihren **Programmcode** zu **kommentieren**!
- **Testen** Sie die von Ihnen implementierten Methoden durch Aufrufe **in der `main`-Methode**!

Aufgabe 4.1: Zahlensysteme (6+3+3P)

In dieser Aufgabe sollen Sie Methoden zu Umwandlung einer Dezimalzahl in andere Zahlensysteme implementieren.

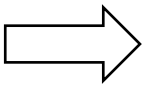
a) In der Vorlesung haben Sie das sog. Horner-Schema zur Umwandlung einer Dezimalzahl in ihre Binärdarstellung kennen gelernt. Sie sollen nun eine Methode `static void printBinary(long number)` implementieren, welche die binäre Darstellung der übergebenen Zahl `number` auf der Konsole ausgibt. Dafür können Sie den Befehl `System.out.print` benutzen, der analog zum schon bekannten Befehl `System.out.println` einen Text auf der Konsole ausgibt, im Gegensatz dazu aber keine neue Zeile beginnt, sondern die Ausgaben direkt hintereinander auf die Konsole schreibt.

```
System.out.println("a");
System.out.println("b");
```



```
>a
>b
```

```
System.out.print("a");
System.out.print("b");
```



```
>ab
```

Im Horner-Schema wird die Binärdarstellung von rechts nach links berechnet, die Ausgabe auf der Konsole erfolgt hingegen von links nach rechts. Um die Zahl dennoch in der richtigen Reihenfolge auszugeben, soll `printBinary` die Darstellung **rekursiv** berechnen!

b) Das Horner-Schema ist nicht nur für die Umwandlung in das Binärsystem mit Basis 2, sondern in Zahlensysteme mit beliebiger Basis geeignet. Implementieren Sie analog zu Aufgabenteil a) eine Methode `static void printNumberSystem(long number, int basis)`, die für eine Dezimalzahl `number` deren Repräsentation in einem Zahlensystem mit einer beliebigen Basis zwischen 2 und 10 auf der Konsole ausgibt. Testen Sie Ihre Implementierung in der main-Methode.

c) Erweitern Sie Ihre Implementierung in Aufgabenteil b), so dass sie auch mit der Hexadezimaldarstellung umgehen kann. Hexadezimalzahlen bestehen 16 Ziffern, wobei die Ziffer **A** der Dezimalzahl 10, **B** der Dezimalzahl 11 etc. enspricht. Implementieren Sie zunächst eine Methode `static void printCipher(int cipher)`, die für eine Zahl zwischen 0 und 15 ihre Zifferndarstellung auf der Konsole ausgibt (bspw. Soll für den Wert 3 „3“ , für den Wert 14 „E“ auf der Konsole ausgegeben). Verwenden Sie diese Methode anschließend in Ihrer Implementierung von `printNumberSystem`!

Aufgabe 4.2: Boolesche Ausdrücke **(4+2+2P)**

In der Vorlesung haben Sie die booleschen Operatoren kennengelernt. Die Semantik der Operatoren wurde durch Funktionstabellen (auch **Wahrheitstabellen** genannt) definiert (Skript Folie 171). Mit Hilfe dieser Wahrheitstabellen kann man auch Äquivalenzen zwischen booleschen Ausdrücken zeigen.

a) Zeigen Sie mit Hilfe einer Wahrheitstabelle folgende Äquivalenz:

$$(X \textbf{ AND } (\textbf{NOT } Y)\ ) \textbf{ OR } (\ \textbf{NOT}(X) \textbf{ AND } Y\ ) \Leftrightarrow X \textbf{ XOR } Y$$

Hinweis: Die Wahrheitstabelle für den Ausdruck auf der linken Seite kann man bestimmen, indem man ihn in Teilausdrücke zerlegt und zunächst für diese die Wahrheitstabelle bestimmt (z.B. für NOT Y). Anschließend kann man die gewonnen Teilergebnisse verwenden, um daraus die Wahrheitstabelle für den gesamten Ausdruck zu bestimmen.

X	**Y**	NOT **Y**	**X** AND (NOT **Y**)	NOT **X**	(NOT **X**) AND **Y**	…	**X** XOR **Y**
true	true	false	false				false
true	false	true	true				true
false	true	false	…				true
false	false	true	…				false

b) Im Rahmen von Aufgabenblatt 2 haben Sie eine Methode `tageImJahr` implementiert, die mit Hilfe mehrerer bedingter Anweisungen die Anzahl der Tage eines Jahres bestimmt (unter Berücksichtigung der Schaltjahre). In dieser Aufgabe soll die Berechnung mit Hilfe einer einzigen bedingten Anweisung erfolgen:

```
static int tageImJahr(int jahr) {
      if (…)
            return 366;
      else
            return 365;
}
```

Implementieren Sie diese Methode und vervollständigen Sie die Bedingung in der obigen IF-Anweisung.

c) In dieser Aufgabe sollen Sie zwei neue boolesche Operatoren definieren und implementieren: < und >. Diese können Sie arithmetisch definieren, in dem Sie folgende Zuordnung machen: true := 1, false := 0.
Implementieren Sie zwei Methoden `static boolean lower(boolean a, boolean b)` bzw. `static boolean greater(boolean a, boolean b)`, die zurückgibt ob `a` < `b` bzw. `a` > `b` gilt.

Aufgabe 4.3: Bitoperationen **(3+4+4+5P)**

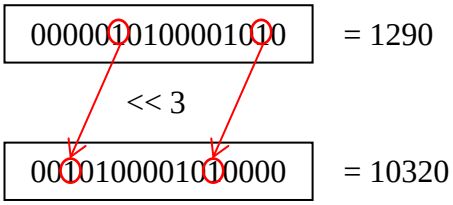
In der letzten Aufgabe haben wir die Operatoren NOT, AND und OR als logische Operatoren auf booleschen Werten benutzt. Diese Operatoren können jedoch auch auf Bitfolgen angewendet werden (siehe Skript Folie 189). Die jeweilige boolesche Operation wird dabei für jedes Bit einzeln ausgewertet.

Folgendes Beispiel wendet den Booleschen Operator OR auf zwei Bitfolgen an:

	1000110111011010
OR	0010110100001010
	1010110111011010

Neben den booleschen Operatoren gibt es noch weitere Operatoren auf Bitfolgen, die sog. Shift-Operatoren. Diese Operatoren „schieben“ die Bitfolge förmlich nach links bzw. rechts aus ihrem Bereich (der eine feste Größe hat: byte = 8bit, short = 16bit, …). Dabei wird von rechts bzw. links üblicherweise mit „0“ aufgefüllt.

Folgendes Beispiel schiebt die angegebene Bitfolge um 3 Positionen nach links und füllt dabei rechts mit Nullen auf:



Die Zahl 10320 entspricht gerade der Zahl $1290 * 2^3$. Allgemein erhöht ein Bitshift um *n* Positionen nach links die entsprechende Zahl um 2^n, ein Bitshift nach rechts erniedrigt sie analog um den gleichen Betrag.

a) Zunächst sollen Sie eine Methode `static boolean checkIfBitIsSet(int bits, int bitPosition)` implementieren, die genau dann **true** zurückliefert, wenn in der Binärdarstellung von `bits` das Bit an Position `bitPosition` auf 1 gesetzt ist.

Für diese Aufgabe sollen Sie nur Shift-Operatoren und logische Operatoren benutzen. Beachten Sie bei Shift-Operatoren die Binärdarstellung von ganzen Zahlen!

Testen die die implementierte Methode (mindestens) mit den folgenden Beispielen:

Ganzzahl `bits`	Binärdarstellung (`short`) `bits`	Position `bitPosition`	Ausgabe
1	00000000 00000001	1	true
2	00000000 00000010	1	false

b) In dieser Aufgabe sollen Sie eine Methode `static int countBitsSetToOne(int bits)` implementieren, die für die Binärdarstellung einer Ganzzahl `bits` die Summe aller auf 1 gesetzten Bits zurückgibt.

Hinweis: Verwenden Sie die in Aufgabenteil a) implementierte Methode, um zu überprüfen ob ein Bit auf 1 gesetzt ist!

Testen Sie die implementierte Methode (mindestens) mit den folgenden Beispielen:

Ganzzahl `bits`	Binärdarstellung (`short`) `bits`	Summe der Bits
1	00000000 00000001	1
3	00000000 00000011	2
17	00000000 00010001	2
21845	01010101 01010101	8

c) In der vorherigen Aufgabe wurde die Summe aller auf 1 gesetzter Bits gezählt. Für diese Aufgabe sollen sie nun eine Methode `static int countLongestSequenceOfBitsSetToOne(int bits)` implementieren, welche, die Länge der längsten Folge auf 1 gesetzter Bits, in der Binärdarstellung einer Ganzzahl `bits`, ermittelt.

Testen Sie die implementierte Methode (mindestens) mit den folgenden Beispielen:

Ganzzahl `bits`	Binärdarstellung (`short`) `bits`	Max. Folge auf 1 gesetzter Bits
1	00000000 00000001	1
3	00000000 00000011	2
17	00000000 00010001	1
3309	00001100 11101101	3

d) Implementieren Sie eine Methode `static boolean containsBitPattern(int bits, int bitPattern)`, die für eine Ganzzahl `bits` überprüft, ob in deren Binärdarstellung eine bestimmte Folge von Bits auftritt. Die Folge ist durch die Parametervariable `bitPattern` gegeben, wobei alle Bits bis zum höchstwertigsten gesetzten Bit in `bitPattern` berücksichtigt werden. Z. B. ist durch die Zahl 5 das Muster 101 gegeben.

1) Zunächst sollen Sie eine Methode `static int findHighestBit(int bits)` implementieren, die für eine Ganzzahl `bits` die Position des höchstwertigsten, auf 1 gesetzten, Bit bestimmt.

   Testen Sie die implementierte Methode (mindestens) mit den folgenden Beispielen:

Ganzzahl `bits`	Binärdarstellung (`short`) `bits`	Höchstwertigstes Bit
1	00000000 00000001	1
3	00000000 00000011	2
17	00000000 00010001	5
3309	00001100 11101101	12

2) Sie können nun die Methode `findHighestBit` benutzen, um das höchstwertigste Bit der Binärdarstellung von `bitPattern` zu bestimmen und so gleichzeitig die Länge des gesuchten Bitmusters herauszufinden.
   Implementieren Sie nun eine Methode `static int createBitmask(int highestBit)`, die eine Ganzzahl erzeugt, in deren Binärdarstellung jedes Bit mit einer Position <= `highestBit` auf 1 gesetzt ist.
   Z.B. würde für die Zahl 5 (Binärdarstellung 101) die Zahl 7 (Binärdarstellung 111) erzeugt.

Tipp: Beachten Sie auch hier die Binärdarstellung von ganzen Zahlen und nutzen Sie Shift-Operationen!

3) Nutzen Sie nun die, in den vorherigen Aufgabenteilen, implementierten Methoden sowie logische Operationen und eine `while`-Schleife zur Implementierung der Methode `containsBitPattern`.

Testen die die implementierte Methode (mindestens) mit den folgenden Beispielen:

Binärdarstellung `bits` (`short`)	`bits` (Wert)	Binärdarstellung `bitPattern`	`bits` enthält `bitPattern`
00000000 00000001	1	1	true
00000000 00000001	1	11	false
00000000 10011101	157	1001	true
00001001 10011101	2461	110	true