



Aufgabenblatt 6 Ausgabe: 22.11., Abgabe: 29.11. 24:00

Gruppe	
Name(n)	Matrikelnummer(n)

Aufgabe 6.1 (Punkte 5+5+5+5)

Shift-Operationen statt Multiplikation: Ersetzen Sie die arithmetischen Ausdrücke *möglichst effizient* durch eine Folge von Operationen: $\ll$, $+$, $-$. Nehmen Sie für die Variablen x und y den Datentyp `int` (32-bit Zweierkomplementzahl) an.

- (a) $y = 20 \cdot x$
- (b) $y = -56 \cdot x$
- (c) $y = 124 \cdot x$
- (d) $y = 60 \cdot (x + 2)$

Aufgabe 6.2 (Punkte 5+5+10+10)

Logische- und Shift-Operationen: Realisieren Sie die folgenden Funktionen als *straightline*-Code in Java, das heißt ohne Schleifen, If-Else Abfragen bzw. ternären Operator `.. ? .. : ...`. Außerdem dürfen nur einige der logischen und arithmetischen Operatoren benutzt werden:

`! ~ & ^ | + << >> >>>`

Alle Eingabeparameter und Rückgabewerte sind jeweils (32-bit) Integerwerte.

- (a) `bitNand(x,y)` Diese Funktion soll das bitweise NAND liefern: $\overline{x_i \wedge y_i}$. Als Operatoren dürfen nur `|` und `~` (OR, Negation) benutzt werden.
- (b) `bitXor(x,y)` Diese Funktion soll die XOR-Verknüpfung (Antivalenz) realisieren: $x_i \neq y_i$. Als Operatoren dürfen nur `|` und `~` (OR, Negation) benutzt werden.
- (c) `getBytes(x,n)` Diese Funktion soll das durch n angegebene Byte ($0 \leq n \leq 3$) aus dem Wert x extrahieren.
- (d) `rotateLeft(x,n)` Die Funktion soll den in Java nicht vorhandenen Rotate-Left Operator für x nachbilden. Für das zweite Argument n gilt: $0 \leq n \leq 31$.

Aufgabe 6.3 (Punkte 10)

Base-64 Codierung: Wie in der Vorlesung skizziert, werden bei der Base-64 Codierung jeweils drei 8-bit Eingangswerte durch vier 6-bit Ausgangswerte ersetzt, die dann zur Datenübertragung in (7-bit) ASCII-Zeichen codiert werden.

Beschreiben Sie durch Logische- und Schiebe-Operationen, wie aus den drei Eingabezeichen $a_1 \dots a_3$, die vier 6-bit Ausgangswerte $b_1 \dots b_4$ berechnet werden. Vervollständigen Sie dazu die Ausdrücke b.. im nachfolgenden Java-Code.

```
int a1, a2, a3;                                // drei Zeichen, Wertebereich je 0..255

int b1 = ?
int b2 = ?
int b3 = ?
int b4 = ?

char[] base64table = new char[] { // Tabelle ersetzt bi durch Zeichen
    'A', 'B', ... 'Z',             // nicht vollständig ...
    'a', 'b', ... 'z',
    '0', '1', ... '9', '+', '/' };

String base64out =                            // 4 Zeichen String als Ausgabe
    base64table[ b1 ] +
    base64table[ b2 ] +
    base64table[ b3 ] +
    base64table[ b4 ];
...
```

Aufgabe 6.4 (Punkte 5+15)

Codierung: Für eine Winkelcodierscheibe mit 15° Grad Auflösung soll ein einschrüttiger zyklischer Binärcode entwickelt werden.

(a) Wie viele Codewörter hat der Code?

(b) Entwickeln Sie einen Code mit dem rekursiven Verfahren aus der Vorlesung.

Aufgabe 6.5 (Punkte 20)

Optimale Codierung: Die folgenden 12 Symbole a_i sind mit ihren Wahrscheinlichkeiten $p(a_i)$ in der Tabelle angegeben:

a_i	a	b	c	d	e	f	g	h	i	j	k	l
$p(a_i)$	0,06	0,12	0,03	0,05	0,3	0,02	0,05	0,1	0,02	0,03	0,1	0,12

Bilden Sie einen Huffman-Baum und geben sie die zugehörige Symbolcodierung an.