64-041 Übung Rechnerstrukturen

Aufgabenblatt 4 Ausgabe: 18.11., Abgabe: 25.11. 12:00

Gruppe	
Name(n)	Matrikelnummer(n)

Aufgabe 4.1 (Punkte 10)

Größenvergleich von Gleitkommazahlen: Für den Vergleich von Gleitkommazahlen bietet Java alle sechs Vergleichsoperatoren:

```
a == b    a != b    a > b    a >= b    a < b    a <= b
```

Aufgrund der unvermeidlichen Rundungsfehler bei Gleitkommarechnung ist jedoch Vorsicht bei Verwendung dieser Operatoren geboten. Zum Beispiel liefert

```
double a = 0.1;
double b = 0.3;
System.out.println( (3*a) == b );
```

den Wert `false`.

Ein naheliegender Ansatz ist daher, zwei Zahlen als „gleich“ anzusehen, wenn der Absolutwert ihrer Differenz kleiner als eine (vom Benutzer) vorgegebene Konstante ist:

```
final double eps = 1.0E-12;

if (Math.abs( a - b ) <= eps) { // Zahlen fast gleich
  ...
}
```

Welchen offensichtlichen Nachteil hat dieses Verfahren? Skizzieren Sie außerdem, wie man ihn beheben könnte.

Aufgabe 4.2 (Punkte 8+2+5)

UTF-8: Die ISO-8859-1 Kodierung benutzt 8 Bit für jedes enthaltene Zeichen. Die direkte Kodierung der basic-multilingual Plane von Unicode (Java Datentyp `char`) verwendet pro Zeichen 16 Bit, während die UTF-8 Kodierung Vielfache von 8 Bit benutzt.

(a) Wir betrachten einen deutschsprachigen Text mit insgesamt 500 000 Zeichen. Wir nehmen die folgenden Wahrscheinlichkeiten für die Umlaute an: **Ä/ä** 0,56%, **Ö/ö** 0,287%, **Ü/ü** 0,616% und **ß** 0,308%. Andere Sonderzeichen kommen nicht vor.

Wie viele Bytes belegt dieser Text bei Kodierung nach ISO-8859-1, in direkter Unicode-Kodierung und in UTF-8?

(b) Wir betrachten einen chinesischen Text mit insgesamt 500 000 Schriftzeichen. Im Unicode-Standard sind für die CJK-Symbole (chinesisch, japanisch, koreanisch) die Bereiche von `U+3400` bis `U+4BDF` und `U+4E00` bis `U+9FCF` reserviert.

Wie viele Symbole sind das?

(c) Wie viele Bytes belegt der chinesische Text bei direkter Unicode-Kodierung und bei Kodierung als UTF-8?

Aufgabe 4.3 (Punkte 5+5+5+5)

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

(a) $y = 18 * x$

(b) $y = 14 * x$

(c) $y = -56 * x$

(d) $y = 62 * (x + 2)$

Aufgabe 4.4 (Punkte 5+5+10+10+15)

Logische- und Shift-Operationen: Realisieren Sie, die folgenden Funktionen als *straightline*-Code in Java, das heißt ohne Schleifen oder If-Else Abfragen, bzw. ternärer 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) `bitXnor(x,y)` Diese Funktion soll die XNOR-Verknüpfung (Äquivalenz) realisieren: $x_i \equiv y_i$. Als Operatoren dürfen nur | und ~ (OR, Negation) benutzt werden.

(c) `getByte(x,n)` Diese Funktion soll das, durch `n` angegebene Byte ($0 \leqslant n \leqslant 3$) 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: $0 \leqslant n \leqslant 31$.

(e) `abs(x)` Der Absolutwert (Betrag) von `x`. Welchen Wert liefert ihre Funktion für den Eingabewert -2^{31}? Beschreiben Sie, wie Ihre Lösung funktioniert.

Aufgabe 4.5 (Punkte 10)

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

Beschreiben Sie durch logische- und Schiebe-Operationen, wie aus den drei Eingabezeichen `a1...a3`, die vier 6-bit Ausgangswerte `b1...b4` 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[] {
  'A', 'B', ... 'Z',
  'a', 'b', ... 'z',
  '0', '1', ... '9', '+', '/' };

String base64Output =
  base64table[ b1 ] +
  base64table[ b2 ] +
  base64table[ b3 ] +
  base64table[ b4 ];
...
```