Aufgabenblatt 4

Ausgabe 15/11/2010, Abgabe bis 22/11/2010 12:00

Name(n):
Matrikelnummer(n):
Übungsgruppe:

Aufgabe 4.1 Größenvergleich von Gleitkommazahlen (5+15 Punkte)

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`.

a) Es ist naheliegend, 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
  ...
}
```

Was ist der Nachteil bei diesem Vorgehen?

b) Überlegen Sie sich ein Verfahren zum Vergleich von zwei Gleitkommazahlen, das deren relative Abweichung berücksichtigt und daher sowohl für große als auch kleine Zahlen (nahe Null) funktioniert. Beschreiben Sie Ihre Idee im Detail, möglichst als Java-Code.

Aufgabe 4.2 UTF-8 (9+6 Punkte)

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 1 000 000 Zeichen. Wir nehmen die folgenden Wahrscheinlichkeiten für die Umlaute (jeweils Groß- und Kleinbuchstaben zusammen) an: **ä** 0.54%, **ö** 0.30%, **ü** 0.65% und **ß** 0.37%. 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 1 000 000 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 Bytes belegt dieser Text bei direkter Unicode-Kodierung und bei Kodierung in UTF-8?

Aufgabe 4.3 Shift-Operationen statt Multiplikation (10+5 Punkte)

Geben Sie eine Folge von Java-Operationen an, mit denen man die folgenden Berechnungen effizient durch Shifts und Addition ersetzen könnte. Nehmen Sie für die Variablen `x` und `y` den Datentyp `int` (32-bit Zweierkomplementzahl) an.

a) $y = 20 * x$

b) $y = 31 * (x + 3)$

Aufgabe 4.4 Logische- und Shift-Operationen (5+5+10+20 Punkte)

Versuchen Sie, die folgenden Operationen als *straightline*-Code in Java zu realisieren, das heißt ohne Schleifen oder If-Else Abfragen. Außerdem dürfen nur die folgenden logischen und arithmetischen Operatoren benutzt werden:

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

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

a) `bitNor(x,y)`

Diese Funktion soll das bitweise NOR von `x` und `y` liefern. Als Operatoren dürfen nur `&` und `~` (AND, Negation) benutzt werden.

b) `bitXor(x,y)`

Diese Funktion soll die XOR-Verknüpfung von `x` und `y` liefern. Als Operatoren dürfen nur `&` und `~` (AND, Negation) benutzt werden.

c) `getByte(x,n)`

Diese Funktion soll das angegebene Byte ($0 \leq$ `n` ≤ 3) aus dem Wert `x` extrahieren.

d) `abs(x)`

Der Absolutwert (Betrag) von `x`. Keine If-Abfragen, nur logische und Shift-Operationen verwenden. Welchen Wert liefert ihre Funktion für den Eingabewert -2^{31}? Beschreiben Sie, wie Ihre Lösung funktioniert.

Aufgabe 4.5 Base-64 Kodierung (10 Punkte)

Die Base-64 Kodierung übersetzt im ersten Schritt jeweils drei 8-bit Eingangswerte in vier 6-bit Ausgangswerte, die dann im zweiten Schritt durch Zeichen aus dem ASCII-Zeichensatz kodiert werden.

Schreiben Sie eine Folge von logischen und Shift-Operationen, die ausgehend von drei Eingabezeichen **a1**, **a2**, **a3** die vier 6-bit Ausgangswerte **b1**, **b2**, **b3**, **b4** berechnet.

```
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 ];
...
```

Vervollständigen Sie die Zuweisungen an b1 bis b4 im obigen Java-Code.