64-041 Übung Rechnerstrukturen und Betriebssysteme

Aufgabenblatt 5 Ausgabe: 13.11., Abgabe: 20.11. 24:00

Gruppe	
Name(n)	Matrikelnummer(n)

Aufgabe 5.1 (Punkte 10+10+10)

Arithmetische Operationen mit Gleitkommazahlen: Gegeben seien zwei IEEE 754 Zahlen A und B (32-bit). Von den 23-bit der Mantisse sind hier nur die oberen 8-bit angegeben, alle anderen Stellen sind 0. Die Zahlen sind folgendermaßen formatiert: *s eeee eeee mmmm mmmm*.

$A = 1\,1000\,0001\,1110\,0000$ und
$B = 0\,1000\,0000\,1100\,0000$

Berechnen Sie ohne Umwandlung in das Dezimalsystem[1] die folgenden Ausdrücke. Alle Ergebnisse sollen wieder als IEEE 754 Zahlen (wie oben) dargestellt werden. Geben Sie dabei immer auch die einzelnen Rechenschritte an.

(a) $A + B$

(b) $A - B$

(c) $A \cdot B$

Aufgabe 5.2 (Punkte 5+7+8)

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`.

[1] Allerdings kann es helfen Kontrollrechnungen dezimal durchzuführen.

(a) 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?

(b) Überlegen Sie sich ein Verfahren zum Vergleich von zwei Gleitkommazahlen, das nicht die absolute (wie oben), sondern eine relative Abweichung berücksichtigt.

(c) Hat auch dieses Verfahren Nachteile? ... und wenn ja, wie könnte man sie beheben?

Aufgabe 5.3 (Punkte 10+10+5)

ULP: Um zu untersuchen, wie sich Rundungsfehler bei der Gleitkomma-Arithmetik auswirken, betrachten wir folgendes Programm, das in einer Schleife immer wieder den Wert `0,1`, der sich in binärer Gleitkommadarstellung ja nicht exakt darstellen lässt, aufaddiert.

```
1  // Test floating point arithmetic...
2  public class aufg05_3 {
3    public static void main( String args[] ) {
4      // calculate the sum of 1E9 unprecise numbers
5      int n = 0;
6      float sum;
7      float limit = 1.0E8f;
8      for( sum = 0; sum < limit; sum += 0.1 ) {
9        n++;
10 //     if ((n % 100000) == 0) {
11 //       System.out.println("n="+ n + " sum="+ sum + " target="+ (n*0.1));
12 //     }
13     }
14     System.out.println("sum is " + sum + " compared to " + (n*0.1));
15   }
16 }
```

(a) Beschreiben Sie kurz, was das Programm tut. Was erwarten Sie (ungefähr) als Ausgabewert des Programms? Wie viele Iterationen sollten am Ende in der `for`-Schleife durchlaufen worden sein?

(b) Was passiert tatsächlich? Warum?
Tipp: Es kann helfen, die auskommentierten Codezeilen 10 ... 12 wieder zu aktivieren.

(c) Schreiben Sie das Programm so um, dass es den ursprünglich angedachten Zweck erfüllt.

Aufgabe 5.4 (Punkte 10+10+5)

Zeichensatzcodierung: Entschlüsseln Sie mit Hilfe der Vorlesungsunterlagen den folgenden hexadezimal codierten Text.

```
41 75 66 67 61 62 65 20 35 2E 34 20 69 73 74 20 65 69 6E 20 55 54 46 2D
38 20 54 65 78 74 0A 09 55 6D 6C 61 75 74 65 3A 20 C3 A4 20 73 74 61 74
74 20 61 65 0A 09 53 79 6D 62 6F 6C 65 3A 20 E2 87 92 2C 20 E2 80 B0 20
E2 80 A6 0A
```

(a) Um was für eine Zeichensatzcodierung[2] handelt es sich?

(b) Wie sieht der Text aus? Notieren Sie dazu die Textdarstellung (mit Steuerzeichen).

(c) Was verrät Ihnen der Text über den Rechner mit dem er erstellt worden ist?

[2] Dabei soll natürlich ein sinnvoller Text herauskommen!