Hochschule
Bonn-Rhein-Sieg
University of Applied Sciences

Programmierung 1
WiSe 25/26
Moritz Balg
Ausgabe: 20.10.2025
Bearbeiten bis: 26.10.2025

Übungsblatt 3

- Bitoperationen und Fließkommadarstellung -

Aufgabe 1: Interne Repräsentation von Zahlen

Überlegen Sie sich zunächst, wie Sie die Umrechnung **möglichst einfach** durchführen können.

1. Geben Sie die folgenden natürlichen Zahlen jeweils in Dual-, Dezimal-, Oktal- und Hexadezimaldarstellung an:

a) 110001101011_2
$6153_8; 3179_{10}; C6B_{16}$

b) 3712_8
$11111001010_2; 1994_{10}; 7CA_{16}$

c) 9675_{10}
$10010111001011_2; 22713_8; 25CB_{16}$

d) FAB_{16}
$111110101011_2; 7653_8; 4011_{10}$

2. Ermitteln Sie für die folgenden ganzen Zahlen jeweils die (kürzestmögliche) Zweikomplementdarstellung:

a) -5_{10}
$-5_{10} = -8_{10} + 3_{10} = 1000^*_2 + 11_2 = 1011^*_2$

b) -300_{10}
$-300_{10} = -512_{10} + 212_{10} = 1000000000^*_2 + 11010100_2 = 1011010100^*_2$

c) Was kennzeichnet die Bitfolge einer kürzestmöglichen Darstellung?
Die Resultate beginnen jeweils mit 10 – denn sonst wäre die Darstellung nicht kürzestmöglich.

3. Ermitteln Sie für die folgenden ganzen Zahlen (in Zweikomplementdarstellung) jeweils die Dezimaldarstellung mit Vorzeichen (der hochgestellte * soll kennzeichnen, dass dies eine Zweierkomplementdarstellung ist):

a) 1000001100^*_2
$1000001100^*_2 = 1000000000^*_2 + 1100_2 = -512_{10} + 12_{10} = -500_{10}$

b) 1111111111111010011^*_2
$1111111111111010011^*_2 = 1010011^*_2 = 1000000^*_2 + 10011_2 = -64_{10} + 19_{10} = -45_{10}$

4. Die ganzzahligen numerischen Datentypen nutzen für die interne Repräsentation negativer Zahlen die Zweierkomplementdarstellung. Welche sind demnach der kleinste und der größte Wert, die eine Variable vom Typ `byte` annehmen kann?
$10000000^*_2 = 10000000^*_2 + 0 = -128_{10}$
$01111111_2 = 127_{10}$

Aufgabe 2: Lotto Spielen mit Java

Die Anzahl an Möglichkeiten, beim Lottospiel 6 aus 49 zu gewinnen, ist $\frac{49*48*47*46*45*44}{1*2*3*4*5*6}$. Berechnen Sie dies in Java und auf ihrem Taschenrechner. Was stellen Sie fest – und warum?

Mögliche Lösung:

```
public class Programm2 {
  public static void main(String[] args) {
    System.out.println("Datentyp int");
    System.out.println("Zähler: " + (49*48*47*46*45*44));
    System.out.println("Nenner: " + (1*2*3*4*5*6));
    System.out.println("Ergebnis: " + (49*48*47*46*45*44)/(1*2*3*4*5*6));

    System.out.println("Datentyp long");
    System.out.println("Zähler: " + (49l*48*47*46*45*44));
    System.out.println("Nenner: " + (1l*2*3*4*5*6));
    System.out.println("Ergebnis: " + (49l*48*47*46*45*44)/(1*2*3*4*5*6));
  }
}
```

Aufgabe 3: 👾 BitExtraktion

Schreiben Sie ein Java-Programm `BitExtraktion`, das folgendermaßen funktioniert. In der Kommandozeile werden 3 ganzzahlige Werte `x`, `y` und `a` in dieser Reihenfolge übergeben. Dabei gilt: $0 \leq x \leq 31, 0 \leq y \leq 31, x \neq y$ und $0 \leq a < 2^{31}$.

Die Aufgabe des Programms ist es, das `x`-te und `y`-te Bit des Wertes `a` zu extrahieren (d.h. den jeweiligen Wert 0 oder 1 zu bestimmen) und die Summe dieser beiden Bitwerte auf dem Bildschirm auszugeben.

Beispiele:

- `java BitExtraktion 0 1 3` erzeugt die Ausgabe 2 (weil der Wert $a = 3_{10} = 0...011_2$ ist und die Summe der letzten beiden Bits an Position x = 0 und y = 1 der Wert 2 = 1 + 1 ist)
- `java BitExtraktion 0 2 3` erzeugt die Ausgabe 1 (weil der Wert $a = 3_{10} = 0...011_2$ ist und die Summe des Bits an Position x = 0(= 1) und an Position y = 2(= 0) gleich 1 ist)
- `java BitExtraktion 4 5 31` erzeugt die Ausgabe 1
- `java BitExtraktion 4 5 63` erzeugt die Ausgabe 2
- `java BitExtraktion 4 5 64` erzeugt die Ausgabe 0
- `java BitExtraktion 17 27 123456789` erzeugt die Ausgabe 1

Es ist sicherlich für Sie hilfreich, wenn Sie sich zuerst einige der Beispielwerte von `a` explizit auf einem Blatt Papier in Bitdarstellung notieren und daran die geforderten Extraktionen und die nachfolgende Summenbildung zu den Ergebniswerten nachvollziehen (siehe Erläuterungen der ersten Beispiele). Überlegen Sie sich anschließend, wie angewandten Operationen ihres Vorgehens auf dem Blatt Papier jeweils in Java umgesetzt werden können. Testen Sie Ihr entwickeltes Programm mit allen Beispielwerten sowie mit zusätzlichen eigenen Tests.

Aufgabe 4: RGB Farben

Eine Möglichkeit zur Kodierung von Farbwerten ist die RGB-Kodierung, in der eine Farbe als eine Mischung aus den Grundfarben Rot, Grün und Blau dargestellt wird. Normiert man den Anteil einer Grundfarbe auf den Wertebereich 0-255, so lässt sich ein solcher Farbanteil als Ganzzahl mit einem Byte darstellen. Eine Farbe lässt sich als Kombination dieser drei Grundfarben dann mit 3 Bytes darstellen. Eine kompakte Darstellung (und Speicherung) eines solchen Farbwertes ist die Nutzung eines int-Wertes, in dem die Bits 0-7 den Blauanteil darstellen, Bit 8-15 den Grünanteil und Bit 16-23 den Rotanteil.

Beispiel: Die Zahl `0x0ba1e2` ● stellt das Cyan der HBRS dar.

Zu jeder Farbe existiert eine Komplementärfarbe. Diese liegt auf dem Farbkreis gegenüber. Für unseren Cyanton wäre dies ein Orange `0xf45e1d` ●.

Zu einer Farbe C lässt sich die Komplementärfarbe C' wie folgt bestimmen:

Sei:
$C = (R, G, B), R, G, B \in [0, 255]$
$C' = (R', G', B') = (255 - R, 255 - G, 255 - B)$

Schreiben Sie ein Programm, welches zu einer gegebenen RGB-Farbe deren Komplementärfarbe bestimmt.

Info

Machen Sie sich klar, was dies genau bedeutet. Stellen Sie die beiden Farben/Zahlen zunächst als Binärzahlen dar und vergleichen Sie diese.

```
/**
 * Umwandlung eines RGB-Wertes in dessen Komplementärfarbe
 * @author Moritz Balg
 * @version 1.0
```

```java
 */
public class RGBInvertieren {
  public static void main(String[] args) {
    // Variante 1
    int rgb = 0x0ba1e2; // HBRS Cyan
    int r, g, b;
    b = 255 - (rgb & 0xff);
    g = 255 - ((rgb >> 8) & 0xff);
    r = 255 - ((rgb >> 16) & 0xff);
    int c1 = r;
    c1 = (c1 << 8) | g;
    c1 = (c1 << 8) | b;
    System.out.println(Integer.toHexString(c1));

    // Variante 2
    int c2 = ~rgb & 0xFFFFFF;
    System.out.println(Integer.toHexString(c2));
  }
}
```

Aufgabe 5: 👾 Prüfbit

Bei der menschlichen Eingabe von Zahlen kann es zu Falscheingaben kommen, zum Beispiel durch einen Tippfehler oder einen Zahlendreher. Einige Verfahren zur automatischen Erkennung solcher Fehler beruhen darauf, dass man zur eigentlichen Nutzinformation eine zusätzliche Prüfziffer anhängt, die sich aus den vorhergehenden Zahlen ergibt. Nachfolgend wird ein sehr einfaches Verfahren dazu beschrieben (Paritätsbit). Schreiben Sie ein Java-Programm ParityBit, das Folgendes leistet. In einer int Variablen wert soll ein (beliebiger) Wert enthalten sein, der aus genau 4 Bit Nutzinformation in den Bits 1-4 besteht und einem zusätzlichen Bit als Prüfziffer im niederwertigsten Bit 0. Der Wert des Prüfbits muss die Quersumme der Bits der Nutzinformation modulo 2 sein, damit die Nutzinformationen als korrekt gewertet wird. Oder anders ausgedrückt: ist die Quersumme der Bits der Nutzinformation eine gerade Zahl, so muss das Prüfbit 0 sein, ansonsten 1.

Beispiele:

- 11000_2 ist ein korrekter Wert, weil Summe 1+1+0+0=2 ergibt und dies demzufolge eine gerade Zahl ist, was dem Prüfbit 0 entspricht.
- 11101_2 ist ein korrekter Wert, weil Summe 1+1+1+0=3 ergibt und dies demzufolge eine ungerade Zahl ist, was dem Prüfbit 1 entspricht.
- 11111_2 ist ein inkorrekter Wert, weil Summe 1+1+1+1=4 ergibt und dies demzufolge eine gerade Zahl ist, aber das Prüfbit ist 1.

Schreiben Sie ihr Java-Programm basierend auf folgender Deklaration: `int wert = 0x17;` (wir haben derzeit noch keine Möglichkeiten zur Dateneingabe) und geben Sie folgendes aus, jeweils in einer Zeile und in der angegeben Reihenfolge:

- das Resultat der Überprüfung der Nutzinformation (Summe der 4 Bits) mit der Prüfziffer als Wahrheitswert (also entweder true oder false, je nachdem ob die Prüfziffer korrekt war oder nicht).
- die berechnete Summe (eine Zahl größer gleich 0)
- der Wert des Paritätsbits (0 oder 1).

Beispiel: Für den Wert $11000_2 = 24_{10}$ wäre die Ausgabe:

```
true
2
0
```

Ihr Programm soll natürlich nicht nur diesen Testwert bearbeiten können, sondern auch jeden anderen zulässigen Wert!

Info

- Nutzen Sie u.a. geeignete Bitoperationen.
- Wie kann man überprüfen, ob das 0. Bit eines int-Wertes 1 ist oder 0?
- Wenn Sie den Wert eines beliebigen Boolschen Ausdrucks ausgeben, so wird dieser Wert berechnet und entweder true oder false ausgegeben, je nachdem, ob der Wert des Ausdrucks wahr oder falsch war. Zum Beispiel produziert System.out.println(i > 5); die Ausgabe true oder false, abhängig vom Wert der Variablen i.

Aufgabe 6: Festkommadarstellung

Angenommen, Sie haben zur Darstellung von reellen Zahlen nur 6 Binärziffern zur Verfügung. In einer Festkommadarstellung (keine Gleitkommadarstellung!) werden davon 3 Vorkommabinärziffern und 3 Nachkommabinärziffern definiert, also kein Vorzeichen und kein Exponent. Oder anders ausgedrückt: jede Zahl hat das Aussehen `xxx,yyy`, wobei `xxx` die binären Vorzeichenziffern und `yyy` die binären Nachkommaziffern sind. Was ist:

1. die kleinste exakt darstellbare Zahl größer Null
kleinste Zahl: $000,001_2 = 0,125_{10}$

2. die größte exakt darstellbare Zahl
größte Zahl: $111,111_2 = 7,875_{10}$

Geben Sie jeweils die Darstellung mit Binärziffern und den jeweiligen Wert im Dezimalsystem an.

Aufgabe 7: IEEE-754 zur Darstellung von Fließkommazahlen

Wandeln Sie die Zahl 3,5 in eine 32 Bit IEEE-754 Fließkommadarstellung um. Geben Sie dabei jeden Zwischenschritt im Umwandlungsprozess an. Die Darstellung des Endresultats enthält also 32 Bit.

1. Darstellung als Dualzahl: $11,1_2$
2. Normalisieren der Mantisse: $1,11_2$
3. Anpassen des Exponenten: Exponent ist 1
4. Darstellung in Exzess-127: $127_{10} + 1_{10} = 128_{10} = 10000000_2$
5. Gesamtdarstellung: 0 10000000 11000...0

Aufgabe 8: Distributivgesetz und der Datentyp Float

Schreiben Sie ein Java-Programm, das für die Float-Werte $x = 2304f, y = -4096f$ und $z = 4096.001953125f$ die Berechnungen $x * (y + z)$ und $x * y + x * z$ ausführt. Wie erklären Sie sich das Ergebnis? Wie lautet das exakte Ergebnis?

$x = 2^{11} + 2^8, y = -(2^{12}), z = 2^{12} + 2^{-9}$

$\Rightarrow y + z = (-2^{12}) + (2^{12} + 2^{-9}) = -2^{12} + 2^{12} + 2^{-9} = 2^{-9}$

$\Rightarrow x * (y + z) = (2^{11} + 2^8) * (2^{-9}) = 2^2 + 2^{-1} = 4,5$

$\Rightarrow x * y = (2^{11} + 2^8) * (-(2^{12})) = -(2^{23} + 2^{30})$

$\Rightarrow x * z = (2^{11} + 2^8) * (2^{12} + 2^{-9}) = 2^{23} + 2^{20} + 2^2 + 2^{-1})$

Die Zahl $x * y$ ist nicht mehr darstellbar. Der untere Teil wird bei der Bestimmung der Mantisse abgeschnitten, sodass die 2^{-1} verloren gehen.

Info

- Stellen Sie die Werte für x, y und z zuerst als Zweierpotenzen dar. Beispiel zu 2304f: $2304 = 2^{11} + 2^{8}$.
- Führen Sie die Berechnungen anschließend Schritt für Schritt per Hand aus (ohne Taschenrechner!), indem Sie die Einzeloperation auf den Zweierpotenzwerten berechnen und das (Zwischen-)Ergebnis auch wieder über Zweierpotenzen ausdrücken. Berechnen Sie jedes Zwischenergebnis einzeln!

  Beispiel: $y + z = (-212) + (212 + 2-9) = 2-9$.
- Führen Sie die Berechnungen per Hand einmal mathematisch exakt aus. Was ist das Ergebnis? Vergleichen Sie dies mit der Ausgabe ihres Programms.
- Führen Sie anschließend die Berechnung unter der Restriktion aus, dass das IEEE-Format zu 32 Bit auf eine Mantissenlänge von 23 Bit beschränkt ist. Überlegen Sie sich, wo das in ihrer Berechnung eine Rolle spielt und was dies für Auswirkungen während der Berechnung hat. Vergleichen Sie den darüber erhaltenen Wert wiederum mit der Ausgabe ihres Programms.

Aufgabe 9: 👾 Geradengleichung

Durch zwei unterschiedliche Punkte $P_1 = (x_1, y_1)$ und $P_2 = (x_2, y_2) \in \mathbb{R}^2$ der Ebene wird eindeutig eine Gerade $f(x) = a * x + b$ durch diese Punkte definiert. Zur Erinnerung: die Parameter a und b der Geraden ergeben sich eindeutig aus den Punkten (siehe Schulmathematik).

Schreiben Sie ein Java-Programm Geradengleichung, das für zwei beliebige verschiedene Punkte $P_1 = (x_1, y_1)$ und $P_2 = (x_2, y_2)$ die Parameter der dadurch bestimmten Geraden berechnet. Es liegen also für die Koordinaten der beiden Punkte insgesamt vier reelle Werte vor, die in der Kommandozeile in der Form $x_1 y_1 x_2 y_2$ dem Programm übergegen werden.

Beispiel:

```
java Geradengleichung 1 2 4 3
a=0.3333333333333333 b=1.6666666666666667
```

⚠ Achtung

- Die beiden Angaben sind durch genau ein Leerzeichen getrennt.
- Je nach Systemeinstallung kann es statt des Dezimalspunkts auch zu einem Dezimalkomma in der Darstellung reeller Werte kommen.

Aufgabe 10: `java.lang.math`

Schauen Sie sich die Funktionalität an, die die Klasse Math laut der Java API-Beschreibung in Form von Methoden anbietet. Allgemeine Anmerkung: Sie finden diese Beschreibung grundlegender Java-Klassen, wenn Sie auf die Java-Seite bei Oracle gehen (https://docs.oracle.com/javase/22/), dort auf den Link API Documentation klicken (unter Specifications zu finden), dort bei den Modulen auf java.base klicken und zuletzt auf java.lang klicken. Durch den Klick auf einen der Links unter Class Summary erhalten Sie detaillierte Informationen zu dieser Klasse.

Aufgabe 11: `java.lang.String`

Schauen Sie sich die Funktionalität an, die die Klasse String laut der Java API-Beschreibung in Form von Methoden anbietet. Nach welchen Arten von Funktionalitäten könnte man die Methoden klassifizieren?