Operationen div und mod für Ganzzahlen

1. Eine Divisionsoperation für rationale Zahlen

Für *rationale Zahlen* gibt es nur **eine** Divisionsoperation. Um sie von anderen Divisionsoperationen zu unterscheiden, bezeichnen wir sie hier mit dem Namen `divr` (statt wie üblich mit dem Operator `/`).

Für die Operation `divr` gilt z.B.

`8,0 divr 4,0` ist gleich `2,0`
`7,0 divr 2,0` ist gleich `3,5`
`7,5 divr 2,0` ist gleich `3,75`
`1,0 divr 3,0` ist gleich $0{,}33\overline{3}$ (sprich: null Komma drei Periode)

2. Mehrere Divisionsoperationen für ganze Zahlen

Als *Ganzzahl-Divisionsoperation* bezeichnen wir eine Divisionsoperation, die man nur auf Ganzzahlen anwenden darf und die immer *eine Ganzzahl als Ergebnis liefert*. Anders als bei rationalen Zahlen gibt es aber nicht nur *eine* solche Divisionsoperation, sondern mehrere. Jede Ganzzahl-Divisionsoperation `divg` muss (für alle Ganzzahlen `g1` und `g2`) die folgenden beiden Bedingungen erfüllen:

Bedingung 1: Wenn das Ergebnis einer rationalen Division `g1 divr g2` eine Ganzzahl `g3` ist, dann muss auch die Ganzzahl-Division `g1 divg g2` dasselbe Ergebnis `g3` liefern.

Beispiel-01: Für jede Ganzzahl-Divisionsoperation `divg` muss gelten:
`12 divg 3` ist gleich `4`.

Bedingung 2: Wenn das Ergebnis einer rationalen Division `g1 divr g2` zwischen zwei Ganzzahlen `g3` und `g3+1` liegt (d.h. größer als `g3` und kleiner als `g3+1` ist), dann muss das Ergebnis der Ganzzahl-Division `g1 divg g2` eine dieser beiden Zahlen sein, entweder `g3` oder `g3+1`.

Beispiel-02: Für jede Ganzzahl-Divisionsoperation `divg` muss gelten:
`5 divg 3` ist gleich `1` oder gleich `2`.

Bei der Definition einer Ganzzahl-Divisionsoperation erlaubt uns die **Bedingung 2** in bestimmten Zweifelsfällen eine Wahl zwischen zwei Alternativen. Da es unendlich viele solcher Zweifelsfälle gibt, gibt es (zumindest theoretisch) unendlich viele Ganzzahl-Divisionsoperationen. Praktisch wichtig sind davon aber weniger als ein halbes Dutzend. In diesem Abschnitt sollen davon 4 mit den Namen `divRR`, `divLL`, `divRL` und `divLR` bezeichnet und erläutert werden. Im nächsten Abschnitt wird dann noch eine fünfte Operation (`divP1`) behandelt.

Anmerkung: Leider gibt es keine verbreiteten Standardnamen für diese Operationen, obwohl ihre Unterscheidung beim Programmieren manchmal wichtig ist.

In den vier Namen `divRR`, `divLL`, `divRL` und `divLR` soll der Buchstabe `R` bedeuten, dass in Zweifelsfällen die nächste weiter **r**echts liegende (die nächste größere) Ganzzahl geliefert wird.

Der Buchstabe `L` bedeutet entsprechend, dass in Zweifelsfällen die nächste weiter **l**inks liegende (die nächste kleinere) Ganzzahl geliefert wird.

In den vier Namen `divRR`, `divLL`, `divRL`, `divLR` betrifft der *erste* Großbuchstabe die Fälle, in denen das rationale Divisionsergebnis *negativ* ist. Der *zweite* Großbuchstabe betrifft die Fälle, in denen das rationale Divisionsergebnis *positiv* ist. Mit anderen Worten und Zeichen:

`divRR` rundet immer nach rechts	`-> 0 ->`
`divLL` rundet immer nach links	`<- 0 <-`
`divRL` rundet hin zur 0	`-> 0 <-`
`divLR` rundet weg von der 0	`<- 0 ->`

Beispiel-03: Ein paar Ergebnisse der 4 Ganzzahl-Divisionsoperationen

a	**b**	**a divr b**	**a divRR b**	**a divLL b**	**a divRL b**	**a divLR b**
+11	+4	+2,75	+3	+2	+2	+3
+11	-4	-2,75	-2	-3	-2	-3
-11	+4	-2,75	-2	-3	-2	-3
-11	-4	+2,75	+3	+2	+2	+3

Aufgabe-01: Füllen Sie die leeren Felder der folgenden Tabelle aus:

a	**b**	**a divr b**	**a divRR b**	**a divLL b**	**a divRL b**	**a divLR b**
+17	+4					
+17	-4					
-17	+4					
-17	-4					

3. Was versteht man unter dem Rest einer Ganzzahl-Division?

Wenn man mit rationalen Zahlen rechnet, ist die Operation *Multiplikation* (`mult`) die exakte Umkehrung der Operation *Division* (`divr`), denn für beliebige rationale Zahlen `a` und `b` gilt:

`(a divr b) mult b` ist gleich `a`

Beispiel-04: `(10,0 divr 4,0) mult 4,0` ist gleich `10,0`

Wenn man mit Ganzzahlen und einer Ganzzahl-Divisionsoperation rechnet, ist die Sache nicht ganz so einfach.

Beispiel-05: `(10 divLL 4) mult 4` ist gleich `8` (denn `10 divLL 4` ist gleich `2`, siehe oben).

Diese Rechnung beginnt mit `10`, endet aber nicht mit `10`, sondern bloß mit `8`.
Die Differenz `10 - 8` gleich `2` bezeichnet man als *den Rest* der Division `10 divLL 4`.

Ausführlicher kann man das auch so schreiben:

`Rest-von[10 divLL 4]` **ist gleich** `10 - (10 divLL 4) mult 4` **ist gleich** `2`

Diese (unübliche und ziemlich umständliche) Notation soll deutlich machen:

Der Rest einer Division hängt (nicht nur von den bearbeiteten Zahlen, sondern) auch davon ab, welche Ganzzahl-Divisionsoperation man verwendet.

Beispiel-06: Der Rest hängt von der Divisionsoperation ab:

`Rest-von[+11 divRR +4]` ist gleich `11 - (+11 divRR +4) mult +4` ist gleich `-1`
`Rest-von[+11 divLL +4]` ist gleich `11 - (+11 divLL +4) mult +4` ist gleich `+3`

`Rest-von[+11 divRL -4]` ist gleich `11 - (+11 divRL -4) mult -4` ist gleich `+3`
`Rest-von[+11 divLR -4]` ist gleich `11 - (+11 divLR -4) mult -4` ist gleich `-1`

Zu jeder Ganzzahl-Divisionsoperation gibt es also eine zugehörige *Rest-Operation* (die auch als *Modulo-Operation* oder kurz mit `mod` bezeichnet wird). Die Restoperationen, die zu den Divisionsoperationen `divRR`, `divLL`, `divRL`, `divLR` gehören, bezeichnen wir hier (natürlich) mit `modRR`, `modLL`, `modRL` und `modLR`. Diese Rest-Operationen sind wie folgt definiert:

```
modRR(a, b) = a - (a divRR b) mult b
modLL(a, b) = a - (a divLL b) mult b
modRL(a, b) = a - (a divRL b) mult b
modLR(a, b) = a - (a divLR b) mult b
```

Beispiel-07: Ein paar Ergebnisse von Divisions- und von Rest-Operationen

a	**b**	**a divr b**	**a divRR b**	**a modRR b = a - (a divRR b) mult b**
+10	+4	+2,5	+3	+10 - (+3 mult +4) = -2
+10	-4	-2,5	-2	+10 - (-2 mult -4) = +2
-10	+4	-2,5	-2	-10 - (-2 mult +4) = -2
-10	-4	+2,5	+3	-10 - (+3 mult -4) = +2

a	**b**	**a divr b**	**a divLL b**	**a modLL b = a - (a divLL b) mult b**
+10	+4	+2,5	+2	+10 - (+2 mult +4) = +2
+10	-4	-2,5	-3	+10 - (-3 mult -4) = -2
-10	+4	-2,5	-3	-10 - (-3 mult +4) = +2
-10	-4	+2,5	+2	-10 - (+2 mult -4) = -2

a	**b**	**a divr b**	**a divRL b**	**a modRL b = a - (a divRL b) mult b**
+10	+4	+2,5	+2	+10 - (+2 mult +4) = +2
+10	-4	-2,5	-2	+10 - (-2 mult -4) = +2
-10	+4	-2,5	-2	-10 - (-2 mult +4) = -2
-10	-4	+2,5	+2	-10 - (+2 mult -4) = -2

a	**b**	**a divr b**	**a divLR b**	**a modLR b = a - (a divLR b) mult b**
+10	+4	2,5	+3	+10 - (+3 mult +4) = -2
+10	-4	-2,5	-3	+10 - (-3 mult -4) = -2
-10	+4	-2,5	-3	-10 - (-3 mult +4) = +2
-10	-4	+2,5	+3	-10 - (+3 mult -4) = +2

Aufgabe-02: Füllen Sie die leeren Felder der folgenden Tabelle aus:

a	**b**	**a modRR b = a-(a divRR b)*b**	**a modLL b = a-(a divLL b)*b**	**a modRL b = a-(a divRL b)*b**	**a modLR b = a-(a divLR b)*b**
+17	+4				
+17	-4				
-17	+4				
-17	-4				

Aufgabe-03: Angenommen, Sie sind der Präsident eines Kaninchenzüchter-Vereins. Der Verein hat 5 Mitglieder und ist zur Zeit (bei einem anderen Verein) mit `17` Kaninchen verschuldet (anders ausgedrückt: Ihr Verein besitzt zur Zeit `-17` Kaninchen). Als Präsident sollen sie die Rückzahlung der Vereinsschuld (aus den privaten Beständen der einzelnen Mitglieder) organisieren. Dazu versuchen Sie, die Schuld von `-17` Kaninchen möglichst gerecht auf die 5 Mitglieder zu verteilen, indem Sie den Quotienten `-17 div 5` berechnen. Aber welche Divisionsoperation sollen Sie anstelle von `div` verwenden? Die Operation `divr` für rationale Zahlen ist prinzipiell ausgeschlossen (Tiere dezimal zu zerstückeln kommt für einen Kaninchenzüchter nicht in Frage). Aber welche Ganzzahl-Divisionsoperation sollten Sie verwenden? Begründen Sie Ihre Wahl.

Aufgabe-04 (für fortgeschrittene Java-Programmierer): Schreiben Sie die oben beschriebenen 4 Ganzzahl-Divisionsoperationen und die 4 zugehörigen Rest-Operationen als Java-Methoden. Natürlich sollen diese Methoden `divRR`, `divLL`, ... und `modRR`, `modLL`, ... heißen. Die Methode `divRR` sollte etwa so aussehen:

```
static public int divRR(int dend, int dor) {
   ...
}
```

4. Noch ein div-mod-Paar

Es gibt noch eine weitere Ganzzahl-Divisionsoperation, die eine beachtliche Fan-Gemeinde hat (z.B. die Autoren der Programmiersprache Maple). Leider hat auch diese Operation noch keinen verbreiteten Namen. Wir nennen sie hier `divP1` (und die zugehörige Rest-Operation `modP1`).

Beschreibung: Die Operation `divP1` lässt nie einen negativen Rest übrig.

Das `P` im Namen der Operation soll daran erinnern, dass der Rest häufig positiv ist.

Mit den oben beschriebenen Funktionen `divRR` und `divLL` kann man `divP1` wie folgt programmieren:

```
static public int divP1(int dend, int dor) {
   // Eine Ganzzahldivision, die nie einen negativen Rest uebrig laesst.
   // Version 1:
   // divP1(+7, +2) ist +3, modP1(+7, +2) ist +1
   // divP1(+7, -2) ist -3, modP1(+7, -2) ist +1
   // divP1(-7, +2) ist -4, modP1(-7, +2) ist +1
   // divP1(-7, -2) ist +4, modP1(-7, -2) ist +1

   if (dor<0) {
      return divRR(dend, dor);
   } else {
      return divLL(dend, dor);
   }
}
```

Die zugehörige Mod-Funktion `modP1` kann man dann etwa so schreiben:

```
static public int modP1(int dend, int dor) {
   // Die zur Divisionsfunktion divP1 passende Modulo-Funktion.
   // Liefert immer ein nicht-negatives Ergebnis.

   return dend - divP1(dend, dor) * dor;
}
```

5. Und was hat das mit Java zu tun?

In Java sind die beiden Ganzzahl-Operationen `divRL` ("rundet hin zur 0") und `modRL` vordefiniert, allerdings unter den kürzeren Namen `/` und `%`. Wenn man andere Ganzzahl-Operationen benötigt (z.B. `divRR` oder `modLL` etc.), muss man sie selbst programmieren.

Aufgabe-05: Programmieren Sie eine Variante `divP2` von `divP1`, die (anstelle von `divRR` und `divLL`) die Methoden `divLR` und `divRL` aufruft.

Anmerkung zur Programmiersprache Ada:

In Ada gibt es zwei Rest-Operationen: **rem** (entspricht der Operation **modRL**) und **mod** (entspricht der Operation **modLL**). Die Ada-Ganzzahldivision `/` "passt" zu **rem**. Zu **mod** gibt es in Ada keine passende Division.