Egész számábrázolás

Miért tanulunk egyáltalán számábrázolást?

Ha ezt és a következő az anyagrészt (egész- és lebegőpontos számábrázolás) nem tanulod meg, nem fogod érteni, hogy mi a különbség pl. a `long` és `int` típusok között, mikor és miért használjuk egyiket vagy másikat. Azt sem fogod érteni, hogy hogyan fordulhat az elő, hogy egy számot csak növelsz és növelsz, de a végén mégiscsak negatív lesz. A lebegőpontos számok esetében pedig egy csomó (megjósolható) kerekítési hibát fogsz kapni, és nem fogod érteni, hogy két számítás eredménye bár matematikailag azonos, „a gép mégis hibásan működik”. Nem hibás, csak így működik. És természetesen mindennek megvan a maga oka, ezt fogjuk most megtanulni.

És még valami: ez a rendszer általános programozási alap, szinte bármilyen programnyelvben találkozni fogsz ezekkel az elvekkel, szóval ez még csak nem is a Java „hibája”.

Az egész számok ábrázolása

A számítógép kettes számrendszerben "gondolkodik", a számokat is így tárolja.

A tízes számrendszer (decimális számrendszer – amit a hétköznapokban használunk) azért tízes, mert 10-féle számjegyet használunk: 0, 1, 2, 3, 4, 5, 6, 7, 8 és 9. A kettes számrendszerben (bináris számrendszer) pedig kétfélét: 0 és 1.

Adódik a kérdés, hogy hogyan írjuk le az 1-nél nagyobb számokat kettes számrendszerben?

Hogyan írunk le 9-nél nagyobb számokat a megszokott, tízes számrendszerünkben?

A 10 értékét (ami a 9-nél eggyel nagyobb) úgy írjuk le, hogy egy 1-est írunk le egy 0 elé. Az értéke ebből úgy számolódik ki, hogy az 1-esnek 10-es értéket tulajdonítunk, tízszerakkorát, mint akkor, ha az utolsó helyen lenne.

A logika ugyanez akkor is, ha kettes számrendszert használunk. Hogyan írjuk le az 1-nél eggyel nagyobb számot? Két számjeggyel. 1 és 0. Csak itt az első 1-esnek nem 10, hanem 2 értéket tulajdonítunk.

Ahogyan tízes számrendszerben az egyre inkább balra elhelyezkedő számjegyek értéke mindig tízszeresre változik, tehát ha azt írjuk le, hogy 3245, annak értéke 3*1000 + 2*100 + 4*10 + 5*1, ugyanígy számoljuk a kettes számrendszerbeli számokat, csak a 2-es alapszámmal: 1011 = 1*8+0*4+1*2+1*1 = 11. A szorzótényezők a 2 hatványai.

Néhány szám kettes számrendszerben a tízes számrendszerbeli megfelelőjével együtt:

0 = 0
1 = 1

2 = 10
3 = 11
4 = 100
5 = 101
6 = 110
7 = 111
8 = 1000
9 = 1001
10 = 1010
11 = 1011
12 = 1100
13 = 1101
14 = 1110
15 = 1111
... és így tovább.

Számok átváltása binárisból tízes számrendszerbe

Ha adott egy kettes számrendszerben megadott szám (pl. 01010110), akkor annak tízes számrendszerű számra váltása úgy történik, hogy jobbról balra haladva egyre nagyobb 2 hatvánnyal szorozzuk meg, majd az így kapott számokat összegezzük. Az utolsó számjegyet (példánkban épp 0 a 2^0-nal szorozzuk meg, vagyis 1-gyel. A szorzat eredménye 0. Az utolsó előtti jegyet (1) pedig 2^1-nel, ami 1*2 = 2. A hátulról a 3. számjegyet (1) pedig 2^2-nel, vagyis 4-gyel):

	2^7		2^6		2^5		2^4		2^3		2^2		2^1		2^0		
	128		64		32		16		8		4		2		1		
0	*	**1**	*	**0**	*	**1**	*	**0**	*	**1**	*	**1**	*	**0**	*		
	0	+	64	+	0	+	16	+	0	+	4	+	2	+	0	=	86

A kék hátterű a szám bináris számjegyeit (azaz bitet – ***bi***nary digi***t***) jelenti, a narancssárga hátterű a számjegyhez tartozó helyiértéket jelenti. Minden bitet összeszorzunk a helyiértékével, amiből a zöld hátterű számokat kapjuk. A zöld hátterű számokat összeadjuk, és így kapjuk meg az eredményt.

Matematikailag ugyanez:

$0*2^7 + 1*2^6 + 0*2^5 + 1*2^4 + 0*2^3 + 1*2^2 + 1*2^1 + 0*2^0 =$
$0*128 + 1*64 + 0*32 + 1*16 + 0*8 + 1*4 + 1*2 + 0*1 =$
$0 + 64 + 0 + 16 + 0 + 4 + 2 + 0 =$
86

Ez a számábrázolás az ún. *előjel nélküli* vagy *unsigned* ábrázolási mód.

Negatív számok

A fenti módszerrel alapesetben csak pozitív számokat tárolhatunk. Dönthetünk úgy, hogy a kombinációk felét negatív számok tárolására fordítjuk. A negatív számok tárolására az egyik megoldás (az ún. **kettes komplemens – two's complement**), amiben a legmagasabb helyiértékű bit (8 bit esetén a $2^7 = 128$) értékének az ellentettjét vesszük figyelembe:

-128, 64, 32, 16, 8, 4, 2, 1

	-2^7		2^6		2^5		2^4		2^3		2^2		2^1		2^0		
	-128		64		32		16		8		4		2		1		
1	*	**1**	*	**0**	*	**1**	*	**0**	*	**1**	*	**1**	*	**0**	*		
	-128	+	64	+	0	+	16	+	0	+	4	+	2	+	0	=	-42

Ennek a változtatásnak a következtében a legkisebb szám a -128 lesz (10000000), a legnagyobb pedig a 127 (01111111). (Csak pozitív számok tárolása esetén a tartomány 0…255).

Előny, hogy ebben az esetben ha ezt a nyolc bites értéket (-128 = 10000000) elkezdjük növelni, akkor egy idő után elérjük a -1-et (11111111), majd ha ehhez hozzáadunk egyet, akkor 9 biten (100000000) értéket kapnánk. Mi viszont nem rendelkezünk 9, csak 8 bittel, ezért a legelső 1-est el kell hagyjuk. (Ezt a jelenséget hívják *túlcsordulásnak*, angolul *overflow*). A maradék 8 bit a 00000000 lesz, ami a 0 értéke. Tehát a mi kis ábrázolásunkban a -1-et (11111111) eggyel megnövelve 0-t (00000000) kapunk, ami hasznos dolog.

Másik tulajdonság, hogy a +127 (01111111)-et eggyel megnövelve -128-at (10000000) kapunk. Ennek a neve is *túlcsordulás (overflow)*.

Ez a módszer az (egyik) *előjeles* azaz *signed* számábrázolási mód.

Ez a Java `byte` ábrázolása (-128..+127). A többi egész típus is hasonló elveken működik, csak több bittel.