PRG1 (3): 数の表現

脇田建

2017.10.3

前回の小テスト

- 小テストの内容：ウェブサイトの講義資料等のところ
- 講評
- よくできなかった人は，よく復習しておいて下さい．

整数の表現

予告：次回小テスト(1)

- 100の二進表現は？
- 二進表現で10010000となる数nについて(n+4)の二進表現は？
- -3.toByte の二進表現は？
- -3.toByteの十六進表現は？
- -3.toShortの二進表現は？
- Intで表現可能な整数の区間は？

ビットとバイトとScalaの4種の整数型

Byte	Short	Int	Long
8 bits	16 bits	32 bits	64 bits
-2^7〜2^7-1	-2^{15}〜$2^{15}-1$	-2^{31}〜$2^{31}-1$	-2^{63}〜$2^{63}-1$
-128〜127	-32,768〜32,767	-2,147,483,648 〜 2,147,483,647	-9,223,372,036,854,775,808 〜 9,223,372,036,854,775,807
± 百	± 3万	± 21億	± 920京

計算機における整数の表現 (Byte)

	符号	2^6	2^5	2^4	2^3	2^2	2^1	2^0
val b0: Byte = 0	0	0	0	0	0	0	0	0
val b1: Byte = 1	0	0	0	0	0	0	0	1
val b2: Byte = 2	0	0	0	0	0	0	1	0
val b3: Byte = 3	0	0	0	0	0	0	1	1
val b4: Byte = 4	0	0	0	0	0	1	0	0
val b5: Byte = 5	0	0	0	0	0	1	0	1
val b6: Byte = 6	0	0	0	0	0	1	1	0
val b7: Byte = 7	0	0	0	0	0	1	1	1
val b8: Byte = 8	0	0	0	0	1	0	0	0

計算機における整数の表現 (Byte)

	符号	2^6	2^5	2^4	2^3	2^2	2^1	2^0
val b0: Byte = 119	0	1	1	1	0	1	1	1
val b1: Byte = 120	0	1	1	1	1	0	0	0
val b2: Byte = 121	0	1	1	1	1	0	0	1
val b3: Byte = 122	0	1	1	1	1	0	1	0
val b4: Byte = 123	0	1	1	1	1	0	1	1
val b5: Byte = 124	0	1	1	1	1	1	0	0
val b6: Byte = 125	0	1	1	1	1	1	0	1
val b7: Byte = 126	0	1	1	1	1	1	1	0
val b8: Byte = 127	0	1	1	1	1	1	1	1

計算機における負数の表現 (Byte)

	符号	2^6	2^5	2^4	2^3	2^2	2^1	2^0
val b3: Byte = 3	0	0	0	0	0	0	1	1
val b2: Byte = 2	0	0	0	0	0	0	1	0
val b1: Byte = 1	0	0	0	0	0	0	0	1
val b0: Byte = 0	0	0	0	0	0	0	0	0
val b0: Byte = -1	1	?	?	?	?	?	?	?
val b1: Byte = -2	1	?	?	?	?	?	?	?
val b2: Byte = -3	1	?	?	?	?	?	?	?
val b3: Byte = -4	1	?	?	?	?	?	?	?
val b8: Byte = -5	1	?	?	?	?	?	?	?

計算機における負数の表現 (Byte)

仮に符号ビットの前にもう1ビットあったとしよう

	謎	符号	2^6	2^5	2^4	2^3	2^2	2^1	2^0
val b3: Byte = 3	1	0	0	0	0	0	0	1	1
val b2: Byte = 2	1	0	0	0	0	0	0	1	0
val b1: Byte = 1	1	0	0	0	0	0	0	0	1
val b0: Byte = 0	1	0	0	0	0	0	0	0	0
val bm0: Byte = -1		1	?	?	?	?	?	?	?
val bm1: Byte = -2		1	?	?	?	?	?	?	?
val bm2: Byte = -3		1	?	?	?	?	?	?	?
val bm3: Byte = -4		1	?	?	?	?	?	?	?
val bm8: Byte = -5		1	?	?	?	?	?	?	?

計算機における負数の表現 (Byte)

★ 2の補数表現 ★

	謎	符号	2^6	2^5	2^4	2^3	2^2	2^1	2^0
val b3: Byte = 3	1	0	0	0	0	0	0	1	1
val b2: Byte = 2	1	0	0	0	0	0	0	1	0
val b1: Byte = 1	1	0	0	0	0	0	0	0	1
val b0: Byte = 0	1	0	0	0	0	0	0	0	0
val b0: Byte = -1	0	1	1	1	1	1	1	1	1
val bm1: Byte = -2	0	1	1	1	1	1	1	1	0
val bm2: Byte = -3	0	1	1	1	1	1	1	0	1
val bm3: Byte = -4	0	1	1	1	1	1	1	0	0
val bm5: Byte = -5	0	1	1	1	1	1	0	1	1

計算機における負数の表現 (Byte)

2の補数表現

	符号	2^6	2^5	2^4	2^3	2^2	2^1	2^0
val b5: Byte = 5	0	0	0	0	0	1	0	1
val bm5: Byte = -5	1	1	1	1	1	0	1	1

- 5と-5の表現をよくよく見比べてみよう.
  - 5の表現から-5の表現を簡単に得るには？

2の補数表現の利点

加算器のハードウェアの実装が単純

	符号	2^6	2^5	2^4	2^3	2^2	2^1	2^0
val b1: Byte = 1	0	0	0	0	0	0	0	1
val bm1: Byte = -1	1	1	1	1	1	1	1	1
b1 + bm1 = 1 + (-1) = 0	0	0	0	0	0	0	0	0
val b6: Byte = 6	0	0	0	0	0	1	1	0
val bm6: Byte = -6	1	1	1	1	1	0	1	0
b6 + bm6 = 6 + (-6) = 0	0	0	0	0	0	0	0	0
val b100: Byte = 100	0	1	1	0	0	1	0	0
val bm100: Byte = -100	1	0	0	1	1	1	0	0
b100 + bm100 = 0	0	0	0	0	0	0	0	0

2の補数表現で表現可能な整数

- Nビットの2の補数表現
  - 最大値：0111…1（0に続けて(N-1)個の1）$= 2^{(N-1)}-1$
  - 最小値：2の補数表現は2^N個の相異なる連続した整数を表現可能だから，植木算により

    $(2^{(N-1)}-1) - (2^N)+1 = (2^{(N-1)}-1) - (2 \cdot 2^{(N-1)})+1 = -2^{(N-1)}$

- ということで $[-2^{(N-1)}, 2^{(N-1)}-1]$ が表現可能な範囲

ビットとバイトとScalaの4種の整数型

Byte	Short	Int	Long
8 bits	16 bits	32 bits	64 bits
-2^{7}〜$2^{7}-1$	-2^{15}〜$2^{15}-1$	-2^{31}〜$2^{31}-1$	-2^{63}〜$2^{63}-1$
±百	±3万	±21億	±920京

```
scala> (Byte.MinValue, Byte.MaxValue)
res16: (Byte, Byte) = (-128,127)

scala> (Short.MinValue, Short.MaxValue)
res17: (Short, Short) = (-32768,32767)

scala> (Int.MinValue, Int.MaxValue)
res18: (Int, Int) = (-2147483648,2147483647)

scala> (Long.MinValue, Long.MaxValue)
res19: (Long, Long) = (-9223372036854775808,9223372036854775807)
```

予告：次回小テスト(2)

- Nビットを用いた2の補数表現における最大の数と最小の数は？

- ScalaにおいてShort型は何ビットか？

オーバーフローの観測 (powers)

2^n: Byte編	2^n: Short編
$2^0 = 1$ $2^1 = 2$ $2^2 = 4$ … 2^5 = 32 2^6 = 64	$2^0 = 1$ $2^1 = 2$ $2^2 = 4$ … 2^13 = 8192 2^14 = 16384
2^7 = -128 2^8 = 0	2^15 = -32768 2^16 = 0
2^n: Int編	2^n: Long編
$2^0 = 1$ $2^1 = 2$ $2^2 = 4$ … $2^{29} = 536870912$ $2^{30} = 1073741824$	$2^0 = 1$ $2^1 = 2$ $2^2 = 4$ … $2^{61} = 2305843009213693952$ $2^{62} = 4611686018427387904$
$2^{31} = -2147483648$ $2^{32} = 0$	$2^{63} = -9223372036854775808$ $2^{64} = 0$

2^n: Byte, 2^n: Short, 2^n: Int, 2^n: Long

- いろんな型について2nを計算したい．でも右のコードは阿呆っぽい．どうする？

- まずはアドホック多相 (Ad Hoc Polymorphism)

```scala
// 2^nをShort上で計算する関数
def powerOf2S(n: Int): Short = {
  def aux(i: Int, p: Short): Short = {
    println(f"2^${i} = ${p}\n")
    if (i == n) p
    else aux(i+1, (p+p).toShort)
  }
  println("\n2^n[Short]")
  aux(0, 1: Short)
}

// 2^nをInt上で計算する関数
def powerOf2I(n: Int): Int = {
  def aux(i: Int, p: Int): Int = {
    println(f"2^${i} = ${p}\n")
    if (i == n) p
    else aux(i+1, p+p)
  }
  println("\n2^n[Int]")
  aux(0, 1)
}

// 2^nをLong上で計算する関数
def powerOf2L(n: Int): Long = {
  def aux(i: Int, p: Long): Long = {
    println(f"2^${i} = ${p}\n")
    if (i == n) p
    else aux(i+1, p+p)
  }
  println("\n2^n[Long]")
  aux(0, 1L)
}
```

Ad Hoc多相な2の冪 (PowersB)

- オーバーローディング：同名だが引数の型が異なる関数定義群

- powerOf2(n: Int, one: Int): Int, powerOf2(n: Int, one: Long): Long, ...

二進法と十六進法

	符号	2^6	2^5	2^4	2^3	2^2	2^1	2^0
val b4: Byte = 123	0	1	1	1	1	0	1	1

- 二進表現は表記が長くてつらい．十進表現でN桁のものは二進表現では$(\log_2 10)N$桁 $\fallingdotseq 3.3N$桁を要する．
  - ※情報科学では$\log_2 M$を$\lg M$と表記することもある．
- つらいので二進表現と親和性の高い十六進表現が便利

二進法と十六進法

2^{11}	2^{10}	2^{9}	2^{8}	2^{7}	2^{6}	2^{5}	2^{4}	2^{3}	2^{2}	2^{1}	2^{0}
1	0	1	0	1	0	1	1	0	1	0	1

- 4M桁の二進表現 ⇒ 4桁の二進表現がM個並んだもの

二進				16進	二進				16進	二進				16進	二進				16進
0	0	0	0	0	0	1	0	0	4	1	0	0	0	8	1	1	0	0	C
0	0	0	1	1	0	1	0	1	5	1	0	0	1	9	1	1	0	1	D
0	0	1	0	2	0	1	1	0	6	1	0	1	0	A	1	1	1	0	E
0	0	1	1	3	0	1	1	1	7	1	0	1	1	B	1	1	1	1	F

Scalaにおける十六進表現

- 0x十六進表現

- 例: 0x2a = 16*2 + 10 = 42

```
scala> println(0x5,0xa,0x2a,0xff,0xffff)
(5,10,42,255,65535)

scala> println(f"${5}%h,${10}%h,${42}%h,${255}%h,${65535}%h")
5,a,2a,ff,ffff

scala> 0xfffffffb
res36: Int = -5

scala> println(f"${-5}%h")
fffffffb
```

おまけ：書式つき文字列

書式つき文字列 (cs1.LX0C)

- 普通の文字列: “This is a string”
- 書式つき文字列
  - f“…${式}…”
    - f“Int.MaxValue is ${**Int.MaxValue**}”
    - f“${5 > 3}”
  - f“…$変数名…”
    - val imv = Int.MaxValue

      f“Int.MaxValue is $**imv**”
  - 書式つき文字列： f“…${式}%書式…”

```scala
package cs1

import scala.math._

// run-main cs1.LX03C: 書式つき文字列の例

object LX03C {
  def fact(n: Int): Int = {
    if (n == 0) 1
    else n * fact(n - 1)
  }

  def main(arguments: Array[String]) {
    // ${...} の内側に書かれた Scala の式の評価結果が文字列に埋め込まれる
    println(f"fact(10) = ${fact(10)}")
    println(f"fact(10) は偶数か？ ${fact(10) % 2 == 0}")
    println(f"fact(10) は偶数か？ ${if (fact(10) % 2 == 0) "はい" else "いいえ"}")
    println(f"Int.MaxValue = ${Int.MaxValue}")

    // 変数の値を文字列に埋め込むときは { } を省略できる.
    val lmax = Long.MaxValue
    println(f"Long.MaxValue = $lmax")

    // %d, %x, %f, %gなどを使ってさらに細かく書式を制御できる
    //     %8x: 8桁の16進表記を埋める
    //
    p
    //
    //
    //
    p
    //
    //
  }
}
```

```
fact(10) = 3628800
fact(10) は偶数か？ true
fact(10) は偶数か？ はい
Int.MaxValue = 2147483647
Long.MaxValue = 9223372036854775807
100> 100,    100>       64,    100> 00000064
100の16進表現 = 64,   π = 3.14159,   e^30 = 1.06865e+13
```

%x　%1.5f　%g

詳しくは，Javaの Formatterクラスの解説

ビット演算

ビット演算とは

- 自然数の二進表現の各桁をそれぞれ独立したビットと見做して演算を施すこと。
- 例
  - ビットごとの論理積

    $3 \,\&\, 5 = 0011_{(2)} \,\&\, 0101_{(2)} = 0001_{(2)} = 1$
  - ビットごとの論理和

    $3 \mid 5 = 0011_{(2)} \mid 0101_{(2)} = 0111_{(2)} = 7$

ビット演算子の概念

bitwise operators

op	説明	例
&	ビットごとの論理積	$3 \& 5 = 0011_{(2)} \& 0101_{(2)} = 0001_{(2)} = 1$
|	ビットごとの論理和	$3 \mid 5 = 0011_{(2)} \mid 0101_{(2)} = 0111_{(2)} = 7$
^	ビットごとの排他的論理和 eXclusive OR (XOR)	$3 \wedge 5 = 0011_{(2)} \wedge 0101_{(2)} = 0111_{(2)} = 7$
~	ビットごとの反転	$\sim 5 = \sim 0101_{(2)} = 1010_{(2)} = 12$
<<	左シフト	$01010101_{(2)} << 3 = 10101000_{(2)}$
>>	算術右シフト	$01010101_{(2)} >> 3 = 00001010_{(2)}$ $10101010_{(2)} >> 3 = 11110101_{(2)}$
>>>	論理右シフト	$01010101_{(2)} >> 3 = 00001010_{(2)}$ $10101010_{(2)} >> 3 = 00010101_{(2)}$

Scalaが提供する二項ビット演算子

& | << >> >>>	Byte	Short	Int	Long
Byte	Int	Int	Int	Long
Short	Int	Int	Int	Long
Int	Int	Int	Int	Long
Long	Long	Long	Long	Long

Scalaが提供する二項算術演算子

+ - * /	Byte	Short	Int	Long
Byte	Int	Int	Int	Long
Short	Int	Int	Int	Long
Int	Int	Int	Int	Long
Long	Long	Long	Long	Long

Scalaが提供する単項ビット演算子

	~
Byte	Int
Short	Int
Int	Int
Long	Long

Scalaにビット演算子の型

- 単項演算子 (~)、二項演算子(& | ^ >> << <<<) の

いずれも、引数にLongが与えられたら結果の型はLong。そうでなければ結果はInt。特に Byte や Short 同士の演算結果も Byte や Short にならないことに注意。

整数型間の型変換

```
scala> ~0
res15: Int = -1

scala> ~0.toByte
res16: Int = -1

scala> (~0).toByte
res17: Byte = -1

scala> (~0).toByte.toInt
res18: Int = -1

scala> ~(0.toByte)
res19: Int = -1
```

注意：Byte型になってないよ。
正しくは次の例

amazon.co.jp

ハッカーのたのしみ—本物のプログラマはいかにして問題を解くか (単行本)

著者: ジュニア,ヘンリー・S. ウォーレン, Jr.,Henry S. Warren

(11)

10 点の新品/中古品を見る ¥3,444 より。

Henry S. Warren, Jr.: *Hacker's Delight*

ハッカーのたのしみ

本物のプログラマはいかにして問題を解く

ヘンリー・S・ウォーレン、ジュニア

滝沢 徹、鈴木 貢、赤池英夫、葛 毅、藤波順久、玉井

初版(2002)邦訳

森先生が学生のときに
愛読してたそうですよ。

Hacker's Delight

SECOND EDITION

HENRY S. WARREN, JR.

原著改訂第２版 (2012)

amazon.co.jp

ハッカーのたのしみ—本物のプログラマはいかにして問題を解くか (単行本)

著者: ジュニア,ヘンリー・S. ウォーレン, Jr.,Henry S. Warren

(11)

10 点の新品/中古品を見る ¥3,444 より。

Henry S. Warren, Jr.: Hacker's Delight

ハッカーのたのしみ

本物のプログラマはいかにして問題を解く

ヘンリー・S・ウォーレン、ジュニア

滝沢 徹、鈴木 貢、赤池英夫、葛 毅、藤波順久 訳

「著作権保護コンテンツ」

print(f"${8}" ${8}%x, ${-8}%x, ${math.pi:12.2f, ${1024*1024}%g")

原著改訂第２版 (2012)

Hacker's Delight

Second Edition

Warren, Jr.

Ryuhei Mori 12:40 PM

「ハッカーのたのしみ」2004年9月発売ですか。このときに買いました。とても好きな本でした。
そして原著では第二版が2012年に出ていることを今知りました。すごい読みたい。

そういや１人ビット演算好きそうな学生がいますよ。assignment0 の #29 算術シフトを使って絶対値を計算するという。

1

いくつかの基本的なハック

ハック	効果
x & (x - 1)	最右ビットを0にする 01011000 → 01010000
x & (x + 1)	(2ⁿ-1)の形式か判定する 結果が 0 なら YES
x & (- x)	最右の1のビットだけを分離 01011000 → 00001000
~x & (x + 1)	最右の0のビットだけを分離 10100111 → 00001000
((x | (x-1)) + 1) & x	最右の連続した1を0にする 01011000 → 01000000

浮動小数点数の表現

IEEE 754 規格

指数部11ビット(e: exponent)　　**仮数部52ビット(f: fractional, mantissa)**

符号(s)
1ビット

- $x = (-1)^s\, 2^{e-1023}\, (1.f)_2$
  仮数部のけち表現

- e in $[1, 2^{11}-2] = [1, 2046]$
  だから,

  $(e-1023)$ in $[-1022, 1023]$

- $1024 / \log 2 = 308.$ 桁

- ScalaのDouble型

IEEE 754の特殊な数表現

- e = 2047, f ≠ 0: 非数 (NaN: not a number)

  Scala では Double.NaN, Float.NaN

- e = 2047, f = 0: sに応じて +∞ あるいは -∞

- e = 0, f ≠ 0: $(-1)^s\ 2^{-1022}\ (0.f)_2$ **けち**をやめて高精度化

- e = 0, f = 0: sに応じて +0 あるいは -0

ScalaのDouble型

- MaxValue / MinValue
- MaxPositiveValue / MinPositiveValue
- PositiveInfinity (+∞) / NegativeInfinity (-∞)
- NaN

浮動小数点数の性質 (LX03D)

- 1.0: Double を2で順次割り続けて右のような出力を得るプログラムを作成しなさい．など

```
sbt:lx03> project lx03
[info] Set current project to
/prg1/)
sbt:lx03> runMain FPUnderflow
[info] Running FPUnderflow
1.0 * 2^-0 = 1.00000
1.0 * 2^-1 = 0.500000
1.0 * 2^-2 = 0.250000
1.0 * 2^-3 = 0.125000
1.0 * 2^-4 = 0.0625000
1.0 * 2^-5 = 0.0312500
1.0 * 2^-6 = 0.0156250
1.0 * 2^-7 = 0.00781250
1.0 * 2^-8 = 0.00390625
1.0 * 2^-9 = 0.00195313
1.0 * 2^-10 = 0.000976563
1.0 * 2^-11 = 0.000488281
1.0 * 2^-12 = 0.000244141
1.0 * 2^-13 = 0.000122070
1.0 * 2^-14 = 6.10352e-05
1.0 * 2^-15 = 3.05176e-05
1.0 * 2^-16 = 1.52588e-05
1.0 * 2^-17 = 7.62939e-06
```

- このプログラムで $2^{-n} = 0.00000$ となる最初の n の値はなにか．IEEE 754 規格の設計に照らして，n がこの値となることの説明をプログラムのコメントとして書きなさい．

- ヒント：n が 2^m とかけ離れる理由を考えよう．

FPOverflow

```
def main(arguments: Array[String]) {
  def addMany(delta: Float, n: Int, v: Float): Float =
    if (n == 0) v else addMany(delta, n-1, v + delta)

  println(f"0.01 * 100 = ${addMany(0.01f, 100, 0f)}")

  val N = 5000000
  val s0 = 0f

  val s1 = addMany(1f, N, s0)
  println(f"1.0 * $N = $s1\n")

  val s2 = addMany(1f, N, s1)
  println(f"1.0 * $N = $s2\n")

  val s3 = addMany(1f, N, s2)
  println(f"1.0 * $N = $s3\n")

  val s4 = addMany(1f, N, s3)
  println(f"1.0 * $N = $s4\n")

  val s5 = addMany(1f, N, s4)
  println(f"1.0 * $N = $s5\n")
}
```

```
sbt:lx03> project lx03
[info] Set current project to lx03
/prg1/)
sbt:lx03> runMain FPOverflow
[info] Running FPOverflow
0.01 * 100 = 0.99999934
1.0 * 5000000 = 5000000.0

1.0 * 5000000 = 1.0E7

1.0 * 5000000 = 1.5E7

1.0 * 5000000 = 1.6777216E7

1.0 * 5000000 = 1.6777216E7

[success] Total time: 0 s, completed Oct 3, 2017 12:11:14 PM
sbt:lx03>
```