Name: Sungho Ahn
Student ID: 862026328

CS133 Lab 1 – Number representation

Objectives

- Understand how numbers, especially floating point numbers, are represented and processed in the processor.
- Manipulate floating point numbers at the bit level.
- Identify the differences between single-precision and double-precision floating point numbers.
- Observe the loss of precision and errors in floating point numbers.

Prerequisites

- Java development environment and Java IDE on your machine.
- Review the IEEE standard floating-point representations

Further Readings

- Steve Hollasch, IEEE Standard 754 Floating Point Numbers, Dec-2-2015, Access online http://steve.hollasch.net/cgindex/coding/ieeefloat.html

Instructions

1. Create a Java project with a single main class.
2. Try the following two lines and write down the output below.
   a. `System.out.println(Float.floatToIntBits(123.5f));`
      Output: 1123483648
   b. `System.out.println(Float.intBitsToFloat(0x42F70000));`
      Output: 123.5
3. Convert the following decimal real number to a binary real number:
   Decimal: 125.75
   Binary: 1111101.11
4. Review the IEEE single-precision floating point standard. What is the number of bits reserved for the sign, exponent, and fraction?
   Sign: 1
   Exponent: 8
   Fraction: 23
5. To represent this number in the IEEE standard single-precision format, we need to normalize that number first.

Normalized representation: $1.1111010 \times 10^{110}$

Mantissa: 1.1111010

Exponent: 110

6. Put the above number in the standard format by adjusting the mantissa and exponent:
   Sign bit: 0
   Adjusted exponent: 1000 0101
   Adjusted mantissa (fraction): 1111 0111 0000 0000 0000 000
7. Provide two methods to verify your answer. (Hint 1: use the two methods in Step 2 above.) (Hint 2: You can use the system calculator in your OS.)

① We can solve out backwards to verify that answer is correct.
   ↳ convert 8-bits exponent to decimal and subtract by 127.
   ↳ Denormalize the mantissa.
   ↳ Convert the result binary to decimal for verification.

② We can use the decimal-to-binary calculator and vice versa.

8. Repeat the above steps for the decimal number -0.2 (-ve number). What do you notice?

Decimal : −0.2

Binary : −0.0011001100110011 0011

Normalized : $-1.1001100110011001 \times 10^{-011}$

Exponent : −011

Sign bit : 1

Adjusted exponent : 0111 1100

Adjusted Mantissa : 1001 1001 1001 1001 1001 101

⇒ 1 0111 1100 1001 1001 1001 1001 1001 101

9. According to your understanding of the IEEE floating-point representation, how do we obtain the smallest +ve value using the method `Float.intBitsToFloat`? Is this a normalized or non-normalized number? Explain your answer.

Smallest +number = 0 0000001 00000000000000000000000
= (0080 0000)16
= 2^{-126}
∴ ≅ 1.17549×10^{-38}

Thus, we can find smallest +ve value by:

```
System.out.println(Float.intBitsToFloat(0x00800000));
```

10. What is the largest +ve value that can be represented in a single-precision floating point number? Please explain how to obtain this value using the method `Float.intBitsToFloat`.

largest number = 0 11111110 11111111111111111111111
= $(7F7F\ FFFF)_{16}$
= $2^{127} \times (2 - 2^{-23})$
≅ 3.40282×10^{38}

Then, we can find the largest +ve value by:

```
System.out.println(Float.intBitsToFloat(0x7F7FFFFF));
```

11. In the following code snippet, what is the smallest value for the variable y that will cause the condition to evaluate to true? Please explain how the answer is obtained using the floating-point standard representation. Smallest y = 0.0000001f

```
float x = 1.0f;
float y =  …;
if ((x+y) > x)
      System.out.println("not-equal "+(x+y));
```

12. Run the following code snippet and report the output.

```
float z = 1.0f / 0.0f;
System.out.println(z);
```

Output: Infinity

13. What is the bit representation of the value of z in the code snippet above?

0 1111 1111 0000 0000 0000 0000 0000 000

14. Repeat parts 12&13 for the following expressions.

Expression	Output	Bit representation
w = 0.0f / 0.0f	NaN	0 1111 1111 111 111 111 111 111 111
z - z	NaN	0 1111 1111 111 111 111 111 111 111
z * z	Infinity	0 1111 1111 0000 0000 0000 0000 0000 000
z / z	NaN	0 1111 1111 111 111 111 111 111 111
z * 0.0f	NaN	0 1111 1111 111 111 111 111 111 111
z * w	NaN	0 1111 1111 111 111 111 111 111 111
0.3f – 0.3f	0.0	0 0000 0000 0000 0000 0000 0000 0000 000
0.3f – 0.2f – 0.1f	7.4505806 E-9	0 0110 0100 0000 0000 0000 0000 0000 000

15. Consider the following code snippet. Is there an assignment to x1 and x2 that causes the program to print "Case 4?" If yes, provide this assignment. If no, explain why not.

```
float x1 = …, x2 = …;
if (x1 < x2)
  System.out.println("Case 1");
```

If either x1 or x2 is NaN, then case 4 will be printed

```
else if (x1 > x2)
  System.out.println("Case 2");
else if (x1 == x2)
  System.out.println("Case 3");
else
  System.out.println("Case 4");
```

16. Will your answer in part 15 change if the type of x1 and x2 was `int`? Why or why not?

Yes, integer type must have a number variable.
Unlike float, int type cannot have NaN as its content.