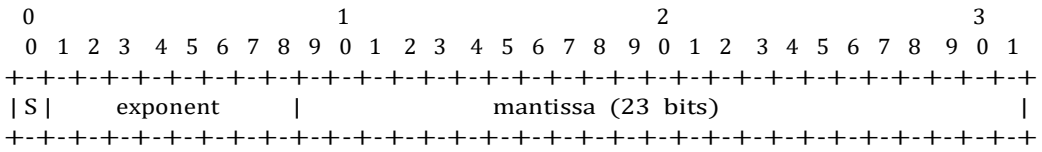
ICS 2022 Problem Sheet #5

Problem 5.1: *IEEE 754 floating point numbers*

IEEE 754 floating point numbers (single precision) use the following format (the numbers on the top of the box indicate bit positions, the fields in the box indicate what the various bits mean).



The encoding starts with a sign bit, followed by the biased exponent (8 bits), followed by the mantissa (23 bits). For single-precision floating-point numbers, the exponents in the range of -126to +127 are biased by adding 127 to get a value in the range 1 to 254 (0 and 255 have special meanings).

a) The standard acceleration of gravity is roughly 9.80665 meters per square seconds. Explain step by step (in your own words) how the decimal fraction 9.80665 is converted into a single precision floating point number.

(1) Determinate the sign bit

9.80665 is positive, hence S = 0

(2) Convert the integer part 9 to binary

9 mod 2 = 1	1
4 mod 2 = 0	01
2 mod 2 = 0	001
1 mod 2 = 1	1001

(3) Convert the fractional part .80665 to binary

0.80665 * 2 = 1.6133	1
0.6133 * 2 = 1.2266	11
0.2266 * 2 = 0.4532	110
0.4532 * 2 = 0.9064	1100
0.9064 * 2 = 1.8128	11001
0.8128 * 2 = 1.6256	110011
0.6256 * 2 = 1.2512	1100111
0.2523 * 2 = 0.5024	11001110
0.5024 * 2 = 1.0048	110011101
0.0048 * 2 = 0.0096	1100111010
0.0096 * 2 = 0.0192	11001110100
0.0192 * 2 = 0.0384	110011101000
0.0384 * 2 = 0.0768	1100111010000
0.0768 * 2 = 0.1536	11001110100000
0.1536 * 2 = 0.3072	110011101000000
0.3072 * 2 = 0.6144	1100111010000000
0.6144 * 2 = 1.2288	11001110100000001
0.2288 * 2 = 0.4576	110011101000000010
0.4576 * 2 = 0.9152	1100111010000000100
0.9152 * 2 = 1.8304	11001110100000001001

[continues]

(4) Binary representation of 9.80665

1001.11001110100000001001 . . . . .

(5) 1001. 11001110100000001001 . . . . . = 1.00111001110100000001001 x 2^3

(6) Biasing the exponent (using the single precision constant 127)

3 + 127 = 130

(7) Conversion of the biased exponent 130 to binary

130 mod 2 = 0	0
65 mod 2 = 1	10
32 mod 2 = 0	010
16 mod 2 = 0	0010
8 mod 2 = 0	00010
4 mod 2 = 0	000010
2 mod 2 = 0	0000010
1 mod 2 = 1	10000010

(8) Drop leading bit of the mantissa (if 1) and encode the bits

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| S |      exponent       |                 mantissa (23 bits)                    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|0| 1 0 0 0 0 0 1 0 |0 0 1 1 1 0 0 1 1 1 0 1 0 0 0 0 0 0 0 1 0 0 1|
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

(9) Representation of the 32 bits in hexadecimal

0100 0001 0001 1100 1110 1000 0000 1001

4 1 1 c e 8 0 9

0x411CE809

b) What is the decimal fraction that is actually stored in the single precision floating point number and what is the absolute error?

Decimal fraction: 1010. 11001110100000001001 = 9.845833

⟹ 9.845833 - 9.80665 = 0.039183 (the absolute error)

Problem 5.2: *unicode and utf-8 encoding*

The content of a file containing UTF-8 Unicode encoded text is given by the following sequence of bytes in hexadecimal notation:

f0 9f 94 a8 20 e2 88 a7 20 f0 9f 8e 93 0a

a) Write each byte in binary notation.

f0 = 11110000, 9f = 10011111, 94 = 10010100, a8 = 10101000, 20 = 00100000,

e2 = 11100010, 88 = 10001000, a7 =10100111, 20 = 00100000, f0 = 11110000,

9f = 10011111, 8e = 10001110, 93 = 10010011, 0a = 00001010

b) Identify the unicode code points of the characters. What is the text stored in the file?

11110000 10011111 10010100 10101000 = 0xF09F94A8 = U+07D49 = '🔨'

00100000 = 0x20 = ' '

11100010 10001000 10100111 = 0xE288A7 = U+2227 = '^'

00100000 = 0x20 = ' '

11110000 10011111 10001110 10010011 = 0xF09F8E93 = U+07CE4 = '🎓'

00001010 = 0x0A = '\n'

⟹ 🔨 ^ 🎓 (\n)

c) Which line end convention is used? What are other popular line end conventions?

The line end convention used is LF: Line Feed, U+000A (UTF-8 in hex: 0A) (\n)
Other popular line end conventions are :

- LF: Line Feed, U+000A (UTF-8 in hex: 0A) (\n)
- VT: Vertical Tab, U+000B (UTF-8 in hex: 0B) (\x0b)
- FF: Form Feed, U+000C (UTF-8 in hex: 0C) (\x0c)
- CR: Carriage Return, U+000D (UTF-8 in hex: 0D) (\r)
- CR+LF: CR (U+000D) followed by LF (U+000A) (UTF-8 in hex: 0D 0A) (\r\n)
- NEL: Next Line, U+0085 (UTF-8 in hex: C2 85) (\x85)
- LS: Line Separator, U+2028 (UTF-8 in hex: E2 80 A8) (\u2028)
- PS: Paragraph Separator, U+2029 (UTF-8 in hex: E2 80 A9) (\u2029)

The most popular ones are LF, CR+LF (Windows and DOS), and CR (old pre-OSX Mac systems, mostly).

Problem 5.3: *large Schröder numbers (haskell)*

Implement a function **s :: Integral a => a -> a** to compute the Schröder number. Write unit test cases using the HUnit test framework.

```
-- | Returns the sum of the products inside the equation
sp :: Integral a => a -> a -> a
sp 0 n = s 0 * s n
sp n k = sp (n-1) (k+1) + s(k)*s(n)

-- | Returns the large Schroder number
s :: Integral a => a -> a
s 0 = 1
s n = s (n-1) + sp (n-1) 0

sTests = TestList [ s 0 ~?= 1
                  , s 1 ~?= 2
                  , s 2 ~?= 6
                  , s 3 ~?= 22
                  ]

main = runTestTT sTests
```