Processing Decimal Values

Dietmar Kühl
Processing Decimal Data

ACCU 2021
2021-03-13

Dietmar Kühl
Developer
dkuhl@bloomberg.net

TechAtBloomberg.com
© 2021 Bloomberg Finance L.P. All rights reserved.
Objective: Correct Decimal Processing

• Round-Trip: Decimal input comes back as the exact same value on output.

• Correct Basic Maths:
  • In case of too many digits, at least they are correctly rounded.
  • Otherwise, addition, subtraction, and multiplication are exact.
  • Division, fractional power, etc. generally can’t be exact.
The number of digits grows…

… for addition to use an extra digit per addition.

… for multiplication to use the sum of the factors’ digits.

The number of digits in fixed size representations is limited.

For exact computations, the number of digits needs to be controlled.
The Problem

0.1
The Problem (double)

0.1

0.1

0.1000000000000000055511151231257827021181583404541015625
The Problem (float)

0.1

0.0999999940395355224609375
The Problem (float)

0.1

1.10011001100110011001100b-4
The Problem

• Integer values do **not** have a problem.

• For fractional values, a floating point representation is used:
  • Due to available hardware, *binary* floating points are used.
  • A binary representation **cannot** represent all decimal values exactly.
  • The problem is masked by looking as if things work.
The Problem (float)

Let's use this value instead:

0.0999999940395355224609375

Computer! 0.1

I got: 0.1
Expectations

- Basic arithmetic operations work correctly.

- Nothing really esoteric, just some simple expressions:

  
  0.3 + 0.6 == 0.9  
  0.4 - 0.3  == 0.1  
  0.3 * 3    == 0.9

- Sadly: none of the above is true for float or double.
The Representation

• Values get decomposed into three components:
  • The values of the components depend on the used base.
  • The *sign* of the value: + or -
  • An integer used as *exponent* for the base to scale the value.
  • An integer to represent the unscaled value (called *significand*).
The Representation

\[ \pm 0 \ldots 0 d_i d_{i-1} \ldots d_0 . d_{-1} \ldots d_{-j} 0 \ldots 0 \]

\[ d_i \in [0, \text{base}) \]
The Representation

\[ (-1)^{\text{sign}} \times \text{base}^{\text{exponent}} \times \sum_{i \in [0,\infty)} d_i \times \text{base}^i \]
The Representation

\((-1)^{\text{sign}} \times \text{base}^{\text{exponent}} \times \sum_{i \in [0, \#\text{digits})} d_i \times \text{base}^i\)
The Representation

$$(-1)^{sign} \times base^{exponent} \times significand$$
The Representation: Special Case Integer

\((-1)^{sign} \times significand\)

Computers don’t really use that: using two’s complement makes things a bit simpler.
The Representation: Special Case Unsigned Integer

significand
Encoding Decimal Values

• Use the closest representable values.

• Minimizes errors on computations.

• Allows round-trip of decimal values (subject to reasonable constraints):
  • The decimal value can be restored from the binary representation.
  • Assuming not too many digits are used and the value is in range.
  • Trailing zeros can’t be recovered.
Encoding Decimal Values: 0.1

Digit Value:
- $0 \times 0.5$
- $0 \times 0.25$
- $0 \times 0.125$
- $1 \times 0.0625$
- $1 \times 0.03125$
- $0 \times 0.015625$
- $0 \times 0.0078125$
- $1 \times 0.00390625$

Remaining value:
- $0.1$
- $0.1$
- $0.1$
- $0.0375$
- $0.00625$
- $0.00625$
- $0.00625$
- $0.00234375$
Encoding Decimal Values

• Two related papers:
  
  
  • “How to Print Floating-Point Numbers Accurately”, Steele/White, https://dl.acm.org/doi/pdf/10.1145/93548.93559
  In particular Dragon 4 for general printing.

• Better performance algorithms for Printing: Grisu and Ryu.
Dragon Algorithms Idea (Recovering Decimal Value)

• Determine the decimal value closest to the encoded binary value.

• To do so, produce leading digits and track the size of the remaining error:
  • Once the error becomes bigger than the remaining value, stop!
  • Implication: the binary value correctly represents a decimal value.
Round-Trip

• Represent decimal value as binary FP and restore decimal value

• Assumes the decimal is in the range the binary value can cover

• There are a limited number of decimal digits:
  • float: 6, double: 15

• Trailing, fractional zeros are lost (numeric value is the same, though)
Why Can Float Only Round-Trip 6 Digits?

- Float uses 24 bits for the significand:
  - 10 bits can represent 1024 values, 3 decimal digits.
  - 4 bits can easily represent one decimal digit.
- Problem: the values are not evenly distributed.
- Example problem:
  - Identical representation for 9.536745e-07 and 9.536746e-07 (0x35800002)
Base 10: Exact Representation of Decimal Value

- Subtraction, addition, multiplication can produce exact values.
- Comparison and formatting readily produce correct results.
- Decimal rounding can be done correctly.
Decimal Representations: String

- The usual representations when processing text.

- BCD (Binary Coded Decimal) packs the data more tightly:
  - 4 bits per digit (or sign or, possibly, decimal point).

- Problems:
  - Variable size or a relatively small range of value.
  - Computations are relatively slow.
Decimal Fixed Point: Scale by a Fixed Power of 10

- Representation is just a signed integer: decimal point implicit in the type.
- Advantage: Operations are very fast - just integer operations.
- Disadvantage: the scale needs to be known and fixed.
- FixedPoint<N> + FixedPoint<N> => FixedPoint<N>
- FixedPoint<N> * FixedPoint<M> => FixedPoint<N + M>
Decimal FixedPoint: Different Scaling Factors

- FixedPoint\(<N> + FixedPoint\(<M> \Rightarrow FixedPoint\text{<max}(N, M)>

- Not quite as fast: requires a multiplication by $10^{\text{abs}(N - M)}$.

- Often a suitable, constant scaling factor isn’t known.

- Make it more flexible: don’t encode the scaling in the type!
  - Use scaling factor from context: becomes more fragile.
  - Idea: store the scaling factor together with the value!
Decimal Floating Point: Scale by Variable Power of 10

- Representation is similar to binary floating point.
- The representation is not normalized:
  - Equal values may have multiple representations: cohorts.
  - Allows representation of number of trailing zeros.
  - Although equal these may display differently, e.g., 0.1 and 0.10.
- Standardized by IEEE 754 (2008)
Decimal Floating Point

- IEEE 754 DFP use ~54 bits for the significand, ~9 bits for the exponent, and 1 bit sign.

- Scaling for addition may require division by a power of 10:
  - Fixed set of divisors needed: use precomputed values with multiplication.
  - Idea: instead of division by $10^n$, multiply by $2^k \cdot 10^{-n}$.

- Typical use cases often sum values with the same scale.

- The flexibility has some cost.
Choose n such that the error doesn’t matter.
Decimal Floating Point: bdldfp Implementation

• With C++, DFPs can be represented as a suitable class.
• There is an open source implementation as part of BDE.
  • bdldfp::Decimal64 (and bdldfp::Decimal32).
• Implemented using Intel’s open source C implementation.
  • https://github.com/bloomberg/bde/tree/master/groups/bdl/bdldfp
Conversions From Binary To Decimal Floating-Point

• Value preserving: fast, but doesn’t restore encoded decimal values.

• Decimal value restoring: slow, but get back the original value.

• Which one to use depends on the context.
Decimal vs. Binary: What to Use

• For exact computation, e.g., in finance: decimal.

  • The transport can be binary if necessary, e.g., for Excel plug-ins.

• Any estimate, simulation, etc.: binary.

  • Exposing a decimal does allow control over the result.

• The value of fractional powers, e.g. interest rates, can’t be represented exactly.
Nature Primed Us Well

• We have 8 fingers: we should use these to count!
Nature Primed Us Well

• We have 8 fingers: we should use these to count!

• Sadly, someone had the bad idea to also use the thumbs…

TechAtBloomberg.com
Thank you!
Questions?
References

• BDE: https://github.com/bloomberg/bde/tree/master/groups/bdl/bdldfp

• IEEE 754 analyzer: https://babbage.cs.qc.cuny.edu/IEEE-754/

• Printing Floating Points: https://dl.acm.org/doi/pdf/10.1145/93548.93559

• Reading Floating Points: https://dl.acm.org/doi/pdf/10.1145/93542.93557