The Journal of Open Source Software

# Boost.Decimal: A C++14 Library for Decimal Floating Point Arithmetic

**Matt Borland** [1] **and Christopher Kormanyos**[2]

**1** The C++ Alliance, United States **2** Independent Researcher, Germany

**DOI:** 10.21105/joss.10345

**Software**

- Review
- Repository
- Archive

**Editor:** Patrick Diehl

**Reviewers:**

- @conradsnicta
- @Becheler

**Submitted:** 25 March 2026
**Published:** 12 August 2026



## Summary

Boost.Decimal is a header-only, C++14 implementation of IEEE 754 decimal floating-point arithmetic types (`decimal32_t`, `decimal64_t` and `decimal128_t`), integrating seamlessly with the C++ Standard Template Library and other Boost Libraries (Boost Authors, 2025). The library is available at https://github.com/boostorg/decimal, and in the Boost distribution with most standard package managers starting from version 1.91.

## Statement of need

In the 2008 revision of IEEE 754 ("IEEE Standard for Floating-Point Arithmetic," 2008), decimal floating-point arithmetic was standardized. These types are critical in applications where decimal numbers must be represented exactly, such as financial software (Cowlishaw, 2003). Binary floating-point representations cannot precisely store common decimal values like 0.3, leading to subtle rounding errors that accumulate over repeated calculations (Goldberg, 1991). Prior to Boost.Decimal, no cross-platform C++ library provided IEEE 754 decimal arithmetic or the associated types.

## State of the field

The two main implementations in this space are produced by IBM (IBM Corporation, 2010) and Intel (Intel Corporation, 2023), with the latter representing the industry standard. These libraries have two shortcomings that Boost.Decimal aims to address. First, both libraries are fundamentally C libraries. This can make integration into modern C++ applications awkward, and they lack support for C++-specific functionality provided in the Standard Library. Second, neither library is cross-platform; for instance, neither library supports ARM architectures. Boost.Decimal addresses both of these shortcomings.

## Software design

Boost.Decimal's design philosophy emphasizes ease of use, portability, and performance. To make the library easy to consume, it requires only C++14 and is header-only with zero external dependencies. Users can simply clone the repository, add `#include <boost/decimal.hpp>`, and begin using the library, even with older toolchains such as Clang 6. Portability is a primary design concern, so we test the library on all three major operating system families (Linux, Windows, and macOS) across a variety of architectures including x86-32, x86-64, ARM32, ARM64, S390X, and PPC64LE. Our continuous integration pipeline ensures consistent numerical results across all supported platforms. Performance is also a priority, both in absolute terms and relative to existing libraries. Extensive benchmarking demonstrates that Boost.Decimal outperforms

both the Intel and IBM libraries in many operations across a variety of platforms. Benchmark results and methodology are available in the repository documentation.

Figure 1 is a minimal usage example showing the representation difference between decimal and binary floating-point arithmetic, and Figure 2 shows the output that the program produces.

```cpp
#include <boost/decimal.hpp>    // This includes the entire decimal library
#include <iostream>
#include <iomanip>

int main()
{
    // The literals are in their own namespace, like std::literals
    using namespace boost::decimal::literals;

    // First we show the result of 0.1 + 0.2 using regular doubles
    std::cout << std::fixed << std::setprecision(17)
              << "Using doubles:\n"
              << "0.1 + 0.2 = " << 0.1 + 0.2 << "\n\n";

    // Construct the two decimal values
    // We construct using the literals defined by the library
    constexpr boost::decimal::decimal64_t a {0.1_DD};
    constexpr boost::decimal::decimal64_t b {0.2_DD};

    // Now we display the result of the same calculation using decimal64_t
    std::cout << "Using decimal64_t:\n"
              << "0.1_DD + 0.2_DD = " << a + b << std::endl;

    return 0;
}
```

**Figure 1:** A complete program that evaluates `0.1 + 0.2` with both `double` and `decimal64_t`.

```
Using doubles:
0.1 + 0.2 = 0.30000000000000004

Using decimal64_t:
0.1_DD + 0.2_DD = 0.3000000000000000
```

**Figure 2:** Output of the program in Figure 1. The binary result carries a representation error, while the decimal result is exact.

## Research Impact Statement

Boost.Decimal enables reproducible numerical research in domains where decimal precision is essential. The library's cross-platform consistency ensures that computational results are identical regardless of hardware architecture, a critical property for scientific reproducibility. Quantitative finance research frequently involves calculations on currency values, interest rates, and transaction amounts that are inherently decimal. Binary floating-point approximations introduce error as early in the computing pipeline as parsing of values, for example, stock prices. Boost.Decimal allows researchers to eliminate these artifacts and produce results that match real-world financial systems.

## AI usage disclosure

No generative AI tools were used in the development of this software, the writing of this manuscript, or the preparation of supporting materials.

## Acknowledgements

We thank the C++ Alliance for sponsoring the development of this work. The library underwent peer review in January and October 2025 by domain experts before acceptance into the Boost library collection. We thank all of these reviewers for their time and contributions. We also thank John Maddock for managing both of these reviews.

## References

Boost Authors. (2025). Boost C++ libraries version. In *GitHub repository*. GitHub. https://github.com/boostorg/boost

Cowlishaw, M. F. (2003). Decimal floating-point: Algorism for computers. *Proceedings of the 16th IEEE Symposium on Computer Arithmetic (ARITH-16)*, 104–111. https://doi.org/10.1109/ARITH.2003.1207666

Goldberg, D. (1991). What every computer scientist should know about floating-point arithmetic. *ACM Computing Surveys*, *23*(1), 5–48. https://doi.org/10.1145/103162.103163

IBM Corporation. (2010). Decimal floating point C library. In *GitHub repository*. GitHub. https://github.com/libdfp/libdfp

IEEE standard for floating-point arithmetic. (2008). *IEEE Std 754-2008*, 1–70. https://doi.org/10.1109/IEEESTD.2008.4610935

Intel Corporation. (2023). Intel(R) decimal floating-point math library v2.3. In *Netlib repository*. netlib. https://www.netlib.org/misc/intel/