Document number: P3605R0
Project: Programming Language C++
Audience: SG6 (Numerics Study Group), LEWG, LWG

Nikita Sakharin, PJSC Sberbank <nikitasa1997@gmail.com>

Date: 2025-02-04

isqrt: A function to calculate integer square root of nonnegative integers

Contents

1. Abstract

This paper proposes to add an `isqrt` function (template) to calculate the integer square root of a nonnegative integer. Mathematically defined as:

$$\operatorname{isqrt}\left(n\right) = \left\lfloor\sqrt{n}\right\rfloor = \max\left\{k \in N : k^2 \le n\right\}, \text{for } n \in N, \text{where } N = \left\{0, 1, 2, 3, \ldots\right\}.$$

`isqrt` of a nonnegative integer `n` is the greatest integer whose square is less than or equal to `n`.

2. Motivation

We will use the following notation:

- **Standard** is the N5001 Working Draft;
- `uintmax_t` is the type `std::uintmax_t` from header `<cstdint>`;
- `sqrt` is the function `std::sqrt` from header `<cmath>`; and
- `double` type is assumed to be the `binary64` type defined in the **ISO/IEC 60559:2020** (**IEEE 754-2019**) standard.

2.1. A common number-theoretic algorithm

The integer square root[1] is a useful number-theoretic primitive. For example, it is commonly applied in:

- **Primality test** and **Integer factorization** algorithms, such as **Trial division** and **Fermat's method**[2];
- **Cryptography** algorithms, such as block entanglement (non-linear transformation)[3];
- **Sqrt-decomposition** method[2]; and
- **Block Merge Sort** algorithm[4].

This algorithm is also pedagogically important. For example, it is published as the “Sqrt(x)”[5] problem on **LeetCode**.

2.2. Expressions `uintmax_t(sqrt(n))` and `isqrt(n)` give different results

There are numerous popular questions about integer square root in **C++** on **StackOverflow**. For example:

- “Fastest way to get the integer part of sqrt(n)?”[6]
- “Looking for an efficient integer square root algorithm for ARM Thumb2”[7]
- “How can you easily calculate the square root of an unsigned long long in C?”[8]
- “Determining if square root is an integer”[9]

Answers to the above questions often recommend a naive solution such as `uintmax_t(sqrt(n))` (or equivalent). At first glance, this may seem correct, because for “small” numbers, that expression actually gives the same result as the `isqrt(n)` function defined above. But for “large” numbers, these two expressions could give different results, even when the value of `n` is exactly representable in the floating-point type used for `sqrt` calculation. Here we define a number to be “large” when it is greater than $2^{\texttt{digits}}$, where `digits` is the number of mantissa bits of the type. Therefore, how “large” the number must be to cause a different result depends on the type used for calculation.

Consider, for example, the `double` type. Per section **§29.7.1 [cmath.syn]** of the **Standard**, this type is used for calculation whenever the argument of `sqrt` has an integer type. For the `double` type, the number of mantissa bits is `52`, so the value of `n` from the example is greater than 2^{52}. Let us take the number:

$$n = 67108865^2 - 1 = 4503599761588224 = 2^{52} + 2^{27},$$

which is exactly representable as a `double`. The square root of this number is:

$$\sqrt{n} = \sqrt{4503599761588224} = 67108864.9999999925494195\ldots$$

According to the definition given above, the `isqrt(n)` call must discard the fractional part of the square root:

$$\text{isqrt}\left(n\right) = \left\lfloor\sqrt{n}\right\rfloor = \left\lfloor\sqrt{4503599761588224}\right\rfloor = \left\lfloor 67108864.9999999925494195\ldots\right\rfloor = 67108864.$$

As an irrational number, $\sqrt{n}$ cannot be represented exactly as a `double`. Therefore, the call `sqrt(n)` returns the nearest (to the “correct”) `double`-representable value. The two adjacent values of `double` type between which $\sqrt{n}$ is enclosed are:

$$67108865 - 2^{-26} = 67108864.99999998509883880615234375 < \sqrt{n} < 67108865.$$

These two values are shown in the table below, along with their representation in the `double` type. The rightmost column shows the absolute error of the `double` approximations to $\sqrt{n}$:

Value	Representation in `double` type			$\left\lvert \text{value} - \sqrt{n} \right\rvert$
	sign	exponent	mantissa	
`67108864.99999998509883880615234375`	`0`	`10000011001`	`0000000000000000000000000011111111111111111111111111`	**7.450580**7079461285×10^{-9}
`67108865.0`	`0`	`10000011001`	`0000000000000000000000000100000000000000000000000000`	**7.450580**4859015277×10^{-9}

Since the absolute error is smaller for the value `67108865.0`, this number is closer to the exact value of the square root, so:

$$\text{sqrt}\left(n\right) = \text{sqrt}\left(4503599761588224\right) = 67108865.$$

Therefore:

$$\text{uintmax\_t}\left(\text{sqrt}\left(n\right)\right) = \text{uintmax\_t}\left(\text{sqrt}\left(4503599761588224\right)\right) = \text{uintmax\_t}\left(67108865\right) = 67108865.$$

Thus:

$$\exists n = i^2 - 1 \text{ for some } i \in N\text{: uintmax\_t}\left(\text{sqrt}\left(n\right)\right) \neq \text{isqrt}\left(n\right).$$

The following table shows the least four values of the integer `n` for which the expressions `uintmax_t(sqrt(n))` and `isqrt(n)` give different results.

`i`	`n = i² - 1`	`uintmax_t(sqrt(n)) = i`	`isqrt(n) = i - 1`
67108865	4503599761588224	67108865	67108864
67108866	4503599895805955	67108866	67108865
67108867	4503600030023688	67108867	67108866
67108868	4503600164241423	67108868	67108867

Finally, if the `long double` type were implemented as an 80-bit floating-point type, then using it would solve the problem for 64-bit integers. However, if the `int128_t` type were added to the **Standard**, then the problem would arise again even for such an 80-bit `long double`. Therefore, the problem cannot be solved simply by using a wider floating-point type.

2.3. Prior Art

Several programming languages have a function (or class method) for calculating the integer square root:

- In **Java**, the `BigInteger` class has the `sqrt()` method[10];
- In **Python**, the `math` module has the `isqrt()` function[11];
- In **Ruby**, the `Integer` class has the `sqrt()` method[12]; and
- In **Rust**, primitive integer types have the `isqrt()` method[13].

3. Design Considerations

3.1 A new overload of `sqrt` cannot be added

Section **§29.7.1 [cmath.syn]** of the **Standard** defines overloads of the `sqrt` for an argument of integer type. Therefore, an overload of the `sqrt` function with an argument and a return value of integer type cannot be added.

3.2 The new function should be named `isqrt`

The **ISO/IEC 10967-2:2001** standard defines an integer square root function named `isqrt`, and we follow this standard's guidance.

3.3 `isqrt` function template can be instantiated for both `signed` and `unsigned` integer types

Mathematically, the integer square root function is defined for only non-negative integers. However, if the function template could not be instantiated for `signed` integers, it would be necessary to cast the argument to an `unsigned` type, even where it is known (e.g., by construction) that the `signed` integer is non-negative. Therefore, the function template should be able to be instantiated for each integer type except `bool`.

3.4 The header to which the function should be added

The header `<cmath>` provides the standard mathematical function `sqrt`. It seems most reasonable (and least surprising to users) to provide the `isqrt` function in the same header.

4. Questions for WG21

WG21 is asked to consider (and preferably affirm) the following questions:

1. Should `isqrt` become part of header `<cmath>` so as to mirror `sqrt`'s location?
2. Should `isqrt`'s algorithmic *complexity* be specified?

5. Proposed Wording

Based on N5001, assuming that WG21 affirms each of the above questions:

5.1 Header `<version>` synopsis

Add to section **§17.3.2 Header `<version>` synopsis [version.syn]** the following:

```
#define __cpp_lib_isqrt yyyymmL // also in <cmath>
```

5.2 Header `<cmath>` synopsis

Add to section **§29.7.1 Header `<cmath>` synopsis [cmath.syn]** the following:

```
// 29.7.7, integer square root
template<class T>
  constexpr T isqrt(T n) noexcept;
```

5.3 Integer square root

Add section **§29.7.7 Integer square root [c.math.isqrt]** consisting of the following:

```
template<class T>
  constexpr T isqrt(T n) noexcept;
1    Mandates: T is a integer type other than cv bool.
2    Preconditions: n is non-negative.
3    Returns: ⌊√n⌋, which is the largest integer whose square is less than or equal to n.
4    Complexity: log(log(n)).
```

6. Implementation Experience

Heron's method (a special case of **Newton's method**) for integers is discussed, for example, in the book "Hacker's Delight"[1]. For the initial estimate, the value $2^{\left\lceil \frac{\log_2(n)}{2} \right\rceil}$ is used, which is the least integer power of two that is greater than or equal to $\sqrt{n}$. Reference implementation:

```
template<class T>
constexpr T isqrt(const T n) noexcept {
    if (n <= T{1})
        return n;

    T i_current{0}, i_next{T(T{1} << ((std::bit_width(T(n - 1)) + 1) >> 1))};
    do {
        i_current = i_next;
        i_next = T((i_current + n / i_current) >> 1);
    } while (i_next < i_current);

    return i_current;
}
```

7. Acknowledgements

Many thanks to Antony Polukhin for assistance in preparation of this paper.

Sincere thanks to Walter E. Brown for being so kind to give valuable advice and corrections to this proposal.

8. References

Hacker's Delight:

1. Henry S. Warren. 2012. Hacker's Delight (2nd. ed.). Addison-Wesley Professional.

Algorithm applications:

2. Antti Laaksonen. 2018. Guide to Competitive Programming: Learning and Improving Algorithms Through Contests (1st. ed.). Springer Publishing Company, Incorporated.
3. Boris S Verkhovsky. 2014. Integer Algorithms in Cryptology and Information Assurance. WORLD SCIENTIFIC.
4. Pok-Son Kim and Arne Kutzner. 2008. Ratio based stable in-place merging. In Proceedings of the 5th international conference on Theory and applications of models of computation (TAMC'08). Springer-Verlag, Berlin, Heidelberg, 246–257.

LeetCode:

5. Sqrt(x)

StackOverflow:

6. Fastest way to get the integer part of sqrt(n)?
7. Looking for an efficient integer square root algorithm for ARM Thumb2
8. How can you easily calculate the square root of an unsigned long long in C?
9. Determining if square root is an integer

Prior Art:

7. BigInteger.sqrt
8. math.isqrt
9. Integer.sqrt
10. i32.isqrt