Greatest Common Divisor and Least Common Multiple, v3

Document #: WG21 N4061
Date: 2014-06-30
Revises: N3913, N3845
Project: JTC1.22.32 Programming Language C++
Reply to: Walter E. Brown <webrown.cpp@gmail.com>

Contents

1 **Introduction** 1
2 **Expository implementation** 2
3 **Proposed wording** 3
4 **Acknowledgments** 4
5 **Bibliography** 4
6 **Document history** 5

Abstract

This paper proposes two frequently-used classical numeric algorithms, **`gcd`** and **`lcm`**, for header **`<numeric>`**. The former calculates the greatest common divisor of two integer values, while the latter calculates their least common multiple. Both functions are already typically provided in behind-the-scenes support of the standard library's **`<ratio>`** and **`<chrono>`** headers.

> *Die ganze Zahl schuf der liebe Gott, alles Übrige ist Menschenwerk.*
> *(The integer is dear God's achievement; all else is work of mankind.)*
> — LEOPOLD KRONECKER (1823–1891)

> *It is now clear that the whole structure of number theory rests on a single foundation, namely the algorithm for finding the greatest common divisor of two numbers.*
> — PETER GUSTAV LEJEUNE DIRICHLET (1805–1859)

1 Introduction

1.1 Greatest common divisor

The *greatest common divisor* of two (or more) integers is also known as the greatest or highest common *factor*. It is defined as the largest of those positive factors[1] shared by (common to) each of the given integers. When all given integers are zero, the greatest common divisor is typically not defined. Algorithms for calculating the GCD have been known since at least the time of Euclid.[2]

Some version of a GCD algorithm is typically taught to schoolchildren when they learn fractions. However, the algorithm has considerably wider applicability. For example, Wikipedia states that GCD "is a key element of the RSA algorithm, a public-key encryption method widely used in electronic commerce."[3]

Note that the standard library's **`<ratio>`** header already explicitly requires GCD's use behind the scenes; see [ratio.ratio]:

[1] Using C++ notation, we would say that the **`int f`** is a factor of the **`int n`** if and only if **`n % f == 0`** is **`true`**.

[2] See http://en.wikipedia.org/wiki/Euclidean_algorithm as of 2013-12-27.

[3] *Loc. cit.*

> 2 The static data members **num** and **den** shall have the following values, where **gcd** represents the greatest common divisor of the absolute values of **N** and **D**:
>
> — **num** shall have the value **sign(N) * sign(D) * abs(N) / gcd**.
>
> — **den** shall have the value **abs(D) / gcd**.

Moreover, the SGI STL has for nearly two decades[4] supplied and used a **__gcd** template in its **<stl_algo.h>** header, treating it as a detail of its **rotate** algorithm implementation.

Because it has broader utility as well, we propose that a **constexpr**, two-argument[5] **gcd** function be added to the standard library. Since it is an integer-only algorithm, we initially proposed that **gcd** become part of **<cstdlib>**, as that is where the integer **abs** functions currently reside, but SG6 consensus favored **<numeric>**.

1.2 Least common multiple

The *least common multiple* of two (or more) integers is also known as the *lowest* or *smallest* common multiple. It is defined as the smallest positive integer that has each of the given integers as a factor. When manipulating fractions, the resulting value is often termed the least common *denominator*.

Computationally, the LCM is closely allied to the GCD. Although its applicability may be not quite as broad as is that of the latter, it is nonetheless already in behind-the-scenes use to support the standard library's **<chrono>** header; see [time.traits.specializations]:

> 1 . . . . [*Note:* This can be computed by forming a ratio of the greatest common divisor of **Period1::num** and **Period2::num** and the least common multiple of **Period1::den** and **Period2::den**. —*end note*]

We therefore propose that a **constexpr**, two-argument[5] **lcm** function accompany **gcd** and likewise become part of the same header, **<numeric>**.

2 Expository implementation

2.1 Exposition-only helpers

We use two helper templates in our sample code. Since **<cstdlib>** defines **abs()** for only **int**, **long**, and **long long** argument types, we formulate our own absolute value function in order to accommodate all integer types, including unsigned standard integer types as well as any signed and unsigned extended integer types. Note that our function is annotated **constexpr**.

```
template< class T >
constexpr auto  abs( T i ) -> enable_if_t< is_integral<T>{}(), T >
{ return i < T(0) ? -i : i; }
```

Second, we factor out the computation of the **common_type** of two integer types. This will allow us, via SFINAE, to restrict our desired functions' applicability to only integer types, as was done for a single type in computing the return type in our **abs** template above:

[4] At least since Version 2.03, dated 1997-09-09, available from https://www.sgi.com/tech/stl/download.html.

[5] Multiple-argument versions can be obtained via judicious combination of **std::accumulate** and the proposed two-argument form. It may be useful to consider an overload taking an **initializer_list**, however.

```
template< class M, class N = M >
using common_int_t = enable_if_t< is_integral<M>{}() and is_integral<N>{}()
                                , common_type_t<M,N>
                                >;
```

2.2 Greatest common divisor

We formulate our **gcd** function as a recursive one-liner so that it can qualify for **constexpr** treatment under C++11 rules:

```
template< class M, class N >
constexpr common_int_t<M,N>  gcd( M m, N n )
{ return n == 0 ? abs(m) : gcd(n, abs(m) % abs(n)); }
```

While this code exhibits a form of the classical Euclidean algorithm, other greatest common divisor algorithms, exhibiting a variety of performance characteristics, have been published.[6] As of this writing, it is unclear whether any of these is suitable for use in the context of a **constexpr** function. We have also been made aware[7] of additional greatest common divisor-related research that may lead to a future proposal for a more general algorithm in the standard library.

2.3 Least common multiple

We also formulate our **lcm** function as a one-liner so that it, too, can qualify for **constexpr** treatment under C++11 rules:

```
template< class M, class N >
constexpr common_int_t<M,N>  lcm( M m, N n )
{ return m == 0 or n == 0 ? 0 : (abs(m) / gcd(m,n)) * abs(n); }
```

3 Proposed wording[8]

3.1 Synopsis

Insert the following declarations into the synopsis in [numeric.ops.overview]:

```
namespace std {
  ...
  template< class M, class N >
  constexpr common_type_t<M,N> gcd(M m, N n);

  template< class M, class N >
  constexpr common_type_t<M,N> lcm(M m, N n);
}
```

[6] E.g., [Web95, Sed97, Sed01, Web05]. See also these papers' respective bibliographies for additional citations to relevant literature.

[7] Sean Parent: Reflector message [c++std-lib-ext-695], citing [Ste99].

[8] All proposed additions and ~~deletions~~ are relative to the post-Chicago Working Draft [N3797]. Editorial notes are displayed against a gray background.

3.2 New text

Append the following new sections to the end of [numeric.ops]:

26.7.7 Greatest common divisor [numeric.gcd]

```
template< class M, class N >
constexpr common_type_t<M,N> gcd(M m, N n);
```

1 *Requires:* |**m**| shall be representable as a value of type **M** and |**n**| shall be representable as a value of type **N**. [*Note:* These requirements ensure, for example, that $\text{gcd}(\text{m}, \text{m}) = |\text{m}|$ is representable as a value of type **M**. — *end note*]

2 *Remarks:* If either **M** or **N** is not an integer type, the program is ill-formed.

3 *Returns:* zero when **m** and **n** are both zero. Otherwise, returns the greatest common divisor of |**m**| and |**n**|,

4 *Throws:* Nothing.

26.7.8 Least common multiple [numeric.lcm]

```
template< class M, class N >
constexpr common_type_t<M,N> lcm(M m, N n);
```

1 *Requires:* |**m**| shall be representable as a value of type **M** and |**n**| shall be representable as a value of type **N**. The least common multiple of |**m**| and |**n**| shall be representable as a value of type **common_type_t<M,N>**.

2 *Remarks:* If either **M** or **N** is not an integer type, the program is ill-formed.

3 *Returns:* zero when either **m** or **n** is zero. Otherwise, returns the least common multiple of |**m**| and |**n**|.

4 *Throws:* Nothing.

3.3 Feature-testing macro

For the purposes of SG10, we recommend the macro name **__cpp_lib_gcd_lcm**.

4 Acknowledgments

Many thanks to the readers of early drafts of this paper for their thoughtful comments. Special thanks to Cassio Neri for his extra-careful proofreading and helpful suggestions, and to Michael Spertus for suggesting that algorithmic complexity be considered. Additional thanks to Cassio Neri for representing the author during the 2014 Rapperswil meeting.

5 Bibliography

[N3797] Stefanus Du Toit: "Working Draft, Standard for Programming Language C++." ISO/IEC JTC1/SC22/WG21 document N3797 (post-Chicago mailing), 2013-10-13. http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2013/n3797.pdf.

[Sed01] Sidi Mohammed Sedjelmaci: "On a Parallel Lehmer-Euclid GCD Algorithm." *Proceedings of the 2001 International Symposium on Symbolic and Algebraic Computation* (ISSAC '01) (2001): 303–308. 1-58113-417-7. http://lipn.univ-paris13.fr/~sedjelmaci/sedjelmaci.pdf.

[Sed97] Mohamed S. Sedjelmaci and Christian Lavault: "Improvements on the Accelerated Integer GCD Algorithm." *Information Processing Letters* 61.1 (1997): 31–36.

[Ste99] Alexander Stepanov: "Greatest Common Measure: the Last 2500 Years." Arthur Schoffstall Lecture in Computer Science and Computer Engineering, Rensselaer Polytechnic Institute, 1999. http://www.stepanovpapers.com/gcd.pdf.

[Web05] Kenneth Weber, Vilmar Trevisan, and Luiz Felipe Martins: "A Modular Integer GCD Algorithm." *Journal of Algorithms* 54.2 (2005): 152–167.

[Web95] Kenneth Weber: "The Accelerated Integer GCD Algorithm." *ACM Transactions on Mathematical Software* 21.1 (1995): 111–122.

6 Document history

Version	**Date**	**Changes**
1	2014-01-01	• Published as N3845 (pre-Issaquah).
2	2014-02-25	• Restored missing abs() calls in algorithm implementations. • Excised comment re standardizing our abs<> in future. • Required abs() result be representable in the argument's type. • Augmented the Acknowledgements. • Mentioned possible future proposal for generalization. • Edited proposed wording per SG6 guidance at Issaquah. • Published as N3913 (post-Issaquah).
3	2014-06-30	• Provided information re gcd in the SGI STL. • Augmented Bibliography of gcd algorithms. • Added to Acknowledgments. • Revised Proposed Wording per LEWG and LWG guidance at Rapperswil. • Fixed typos and made other editorial improvements. • Published as N4061 (post-Rapperswil).