Greatest Common Divisor and Least Common Multiple

Document #:	WG21 N3845
Date:	2014-01-01
Revises:	None
Project:	JTC1.22.32 Programming Language C++
Reply to:	Walter E. Brown <`webrown.cpp@gmail.com`>

Contents

Abstract

This paper proposes two frequently-used classical numeric algorithms, **`gcd`** and **`lcm`**, for header **`<cstdlib>`**. 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.
(Integers are dear God's achievement; all else is work of mankind.)
— LEOPOLD KRONECKER

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 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**.

Because it has broader utility as well, we propose that a **constexpr**, two-argument[4] **gcd** function be added to the standard library. Since it is an integer-only algorithm, we propose that **gcd** become part of **<cstdlib>**, as that is where the integer **abs** functions currently reside.

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 is 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 we propose that a **constexpr**, two-argument[4] **lcm** function accompany **gcd** and likewise become part of **<cstdlib>**.

2 Implementation

2.1 Helpers

We use two helper templates in our sample code. Since **<cstdlib>** defines **abs()** for only **int**, **long**, and **long long** argument types, we first formulate an overload to accommodate all remaining integer types, including unsigned standard integer types and any signed and unsigned extended integer types. Note that our function is marked **constexpr**; if necessary, we will propose that the existing **abs** functions also be thusly declared.

```
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:

```
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>
                                >;
```

[4]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.

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 auto  gcd( M m, N n ) -> common_int_t<M,N>
{
  using CT = common_int_t<M,N>;
  return n == 0 ? clib::abs<CT>(m) : gcd<CT,CT>(n, m % n);
}
```

While this code exhibits a form of the classical Euclidean algorithm, other greatest common divisor algorithms, exhibiting different performance characteristics, have been published.[5] As of this writing, it is unclear whether any of these is suitable for use in the context of a **constexpr** function.

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 auto  lcm( M m, N n ) -> common_int_t<M,N>
{ return abs((m / gcd(m,n)) * n); }
```

3 Proposed wording[6]

3.1 New text

Insert the following consecutively-numbered paragraphs so as to follow [c.math]/7, and renumber the displaced original paragraphs 8–11 as 11-14:

8 C++ further adds function templates **gcd** and **lcm** to **<cstdlib>**.

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

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

9 *Requires:* **M** and **N** shall be integer types. **m** and **n** shall not both be zero.

10 *Returns:* the greatest common divisor of |**m**| and |**n**|, and the least common multiple of |**m**| and |**n**|, respectively.

3.2 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.

[5] E.g., [Web95, Sed97, Web05].

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

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.

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

[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.