GCD and LCM

Daryle Walker

Copyright © 2001 -2002 Daryle Walker

Table of Contents

This manual is also available in printer friendly PDF format.

Greatest Common Divisor and Least Common Multiple

Introduction

The class and function templates in <boost/math/common_factor.hpp> provide run-time and compile-time evaluation of the greatest common divisor (GCD) or least common multiple (LCM) of two integers. These facilities are useful for many numeric-oriented generic programming problems.

Synopsis

```
namespace boost
{
namespace math
{

template < typename IntegerType >
   class gcd_evaluator;
template < typename IntegerType >
   class lcm_evaluator;

template < typename IntegerType >
   IntegerType  gcd( IntegerType const &a, IntegerType const &b );
template < typename IntegerType >
   IntegerType  lcm( IntegerType const &a, IntegerType const &b );

template < unsigned long Value1, unsigned long Value2 >
   struct static_gcd;
template < unsigned long Value1, unsigned long Value2 >
   struct static_lcm;

}
}
```

GCD Function Object

Header: <boost/math/common_factor_rt.hpp>

```
template < typename IntegerType >
class boost::math::gcd_evaluator
{
public:
   // Types
   typedef IntegerType  result_type;
   typedef IntegerType  first_argument_type;
   typedef IntegerType  second_argument_type;

   // Function object interface
   result_type  operator ()( first_argument_type const &a,
   second_argument_type const &b ) const;
};
```

The boost::math::gcd_evaluator class template defines a function object class to return the greatest common divisor of two integers. The template is parameterized by a single type, called IntegerType here. This type should be a numeric type that represents integers. The result of the function object is always nonnegative, even if either of the operator arguments is negative.

This function object class template is used in the corresponding version of the GCD function template. If a numeric type wants to customize evaluations of its greatest common divisors, then the type should specialize on the gcd_evaluator class template.

LCM Function Object

Header: <boost/math/common_factor_rt.hpp>

```
template < typename IntegerType >
class boost::math::lcm_evaluator
{
public:
   // Types
   typedef IntegerType  result_type;
   typedef IntegerType  first_argument_type;
   typedef IntegerType  second_argument_type;

   // Function object interface
   result_type  operator ()( first_argument_type const &a,
   second_argument_type const &b ) const;
};
```

The boost::math::lcm_evaluator class template defines a function object class to return the least common multiple of two integers. The template is parameterized by a single type, called IntegerType here. This type should be a numeric type that represents integers. The result of the function object is always nonnegative, even if either of the operator arguments is negative. If the least common multiple is beyond the range of the integer type, the results are undefined.

This function object class template is used in the corresponding version of the LCM function template. If a numeric type wants to customize evaluations of its least common multiples, then the type should specialize on the lcm_evaluator class template.

Run-time GCD & LCM Determination

Header: <boost/math/common_factor_rt.hpp>

```
template < typename IntegerType >
IntegerType  boost::math::gcd( IntegerType const &a, IntegerType const &b );

template < typename IntegerType >
IntegerType  boost::math::lcm( IntegerType const &a, IntegerType const &b );
```

The boost::math::gcd function template returns the greatest common (nonnegative) divisor of the two integers passed to it. The boost::math::lcm function template returns the least common (nonnegative) multiple of the two integers passed to it. The function templates are parameterized on the function arguments' IntegerType, which is also the return type. Internally, these function templates use an object of the corresponding version of the gcd_evaluator and lcm_evaluator class templates, respectively.

Compile time GCD and LCM determination

Header: <boost/math/common_factor_ct.hpp>

```
template < unsigned long Value1, unsigned long Value2 >
struct boost::math::static_gcd
{
   static unsigned long const  value = implementation_defined;
};

template < unsigned long Value1, unsigned long Value2 >
struct boost::math::static_lcm
{
   static unsigned long const  value = implementation_defined;
};
```

The boost::math::static_gcd and boost::math::static_lcm class templates take two value-based template parameters of the unsigned long type and have a single static constant data member, value, of that same type. The value of that member is the greatest common factor or least common multiple, respectively, of the template arguments. A compile-time error will occur if the least common multiple is beyond the range of an unsigned long.

Example

```cpp
#include <boost/math/common_factor.hpp>
#include <algorithm>
#include <iterator>


int main()
{
   using std::cout;
   using std::endl;

   cout << "The GCD and LCM of 6 and 15 are "
   << boost::math::gcd(6, 15) << " and "
   << boost::math::lcm(6, 15) << ", respectively."
   << endl;

   cout << "The GCD and LCM of 8 and 9 are "
   << boost::math::static_gcd<8, 9>::value
   << " and "
   << boost::math::static_lcm<8, 9>::value
   << ", respectively." << endl;

   int  a[] = { 4, 5, 6 }, b[] = { 7, 8, 9 }, c[3];
   std::transform( a, a + 3, b, c, boost::math::gcd_evaluator<int>() );
   std::copy( c, c + 3, std::ostream_iterator<int>(cout, " ") );
}
```

Header <boost/math/common_factor.hpp>

This header simply includes the headers <boost/math/common_factor_ct.hpp> and <boost/math/common_factor_rt.hpp>.

Note this is a legacy header: it used to contain the actual implementation, but the compile-time and run-time facilities were moved to separate headers (since they were independent of each other).

Demonstration Program

The program common_factor_test.cpp is a demonstration of the results from instantiating various examples of the run-time GCD and LCM function templates and the compile-time GCD and LCM class templates. (The run-time GCD and LCM class templates are tested indirectly through the run-time function templates.)

Rationale

The greatest common divisor and least common multiple functions are greatly used in some numeric contexts, including some of the other Boost libraries. Centralizing these functions to one header improves code factoring and eases maintainence.

History

- 17 Dec 2005: Converted documentation to Quickbook Format.

- 2 Jul 2002: Compile-time and run-time items separated to new headers.

- 7 Nov 2001: Initial version

Credits

The author of the Boost compilation of GCD and LCM computations is Daryle Walker. The code was prompted by existing code hiding in the implementations of Paul Moore's rational library and Steve Cleary's pool library. The code had updates by Helmut Zeisel.