Numeric Limits

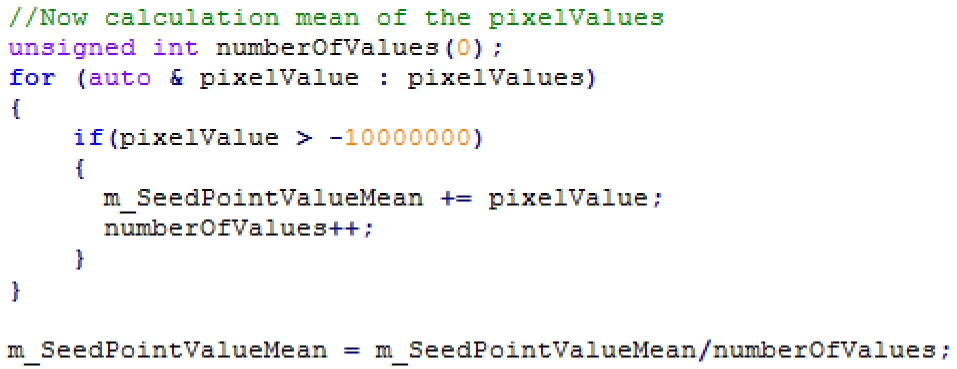
BugSquashing Seminar 11.11.15

Tobias Norajitra

GERMAN
CANCER RESEARCH CENTER
IN THE HELMHOLTZ ASSOCIATION

Motivation (1/2)

- Found in MITK:

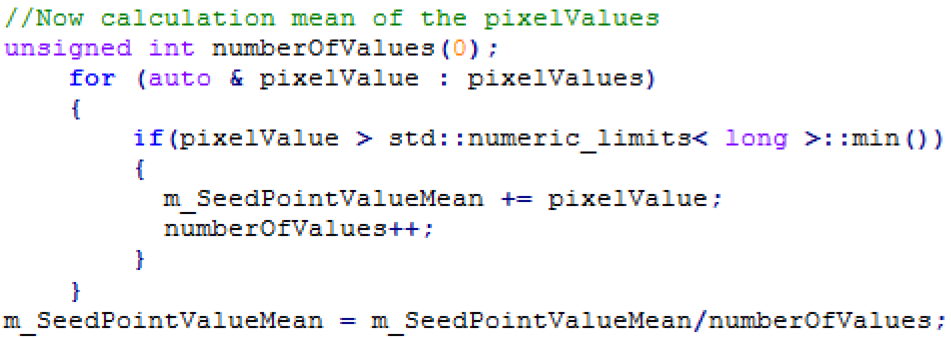


```
//Now calculation mean of the pixelValues
unsigned int numberOfValues(0);
for (auto & pixelValue : pixelValues)
{
    if(pixelValue > -10000000)
    {
      m_SeedPointValueMean += pixelValue;
      numberOfValues++;
    }
}

m_SeedPointValueMean = m_SeedPointValueMean/numberOfValues;
```

- Arbitrary value chosen as lower limit

Motivation (2/2)

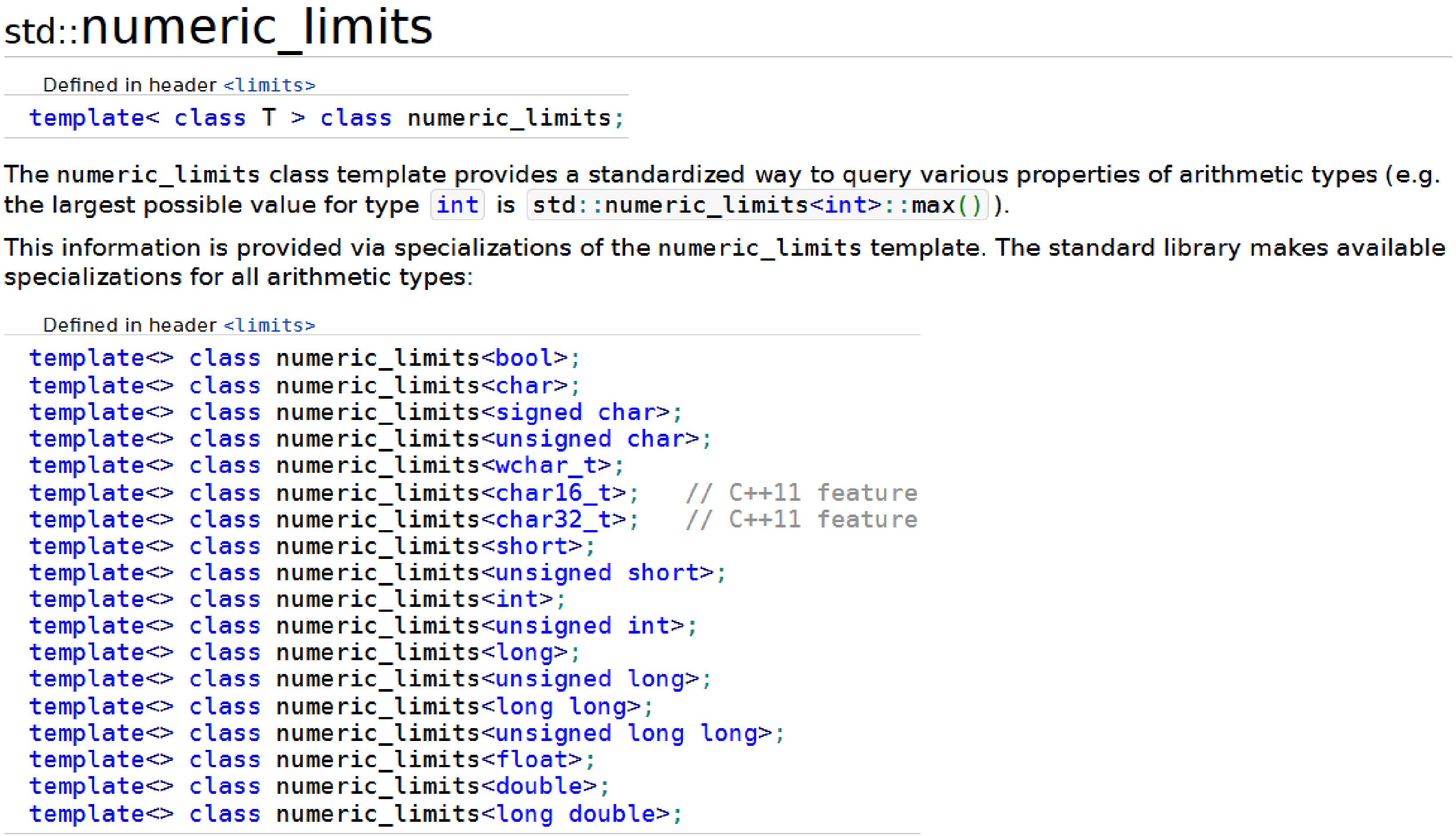
- Instead, make use of more well-defined limits:



```
//Now calculation mean of the pixelValues
unsigned int numberOfValues(0);
    for (auto & pixelValue : pixelValues)
    {
        if(pixelValue > std::numeric_limits< long >::min())
        {
          m_SeedPointValueMean += pixelValue;
          numberOfValues++;
        }
    }
m_SeedPointValueMean = m_SeedPointValueMean/numberOfValues;
```

- Besides: although the implicit type conversions caused no problems in this particular case, it is risky

std::numeric_limits (1/4)



std::numeric_limits

Defined in header `<limits>`

```
template< class T > class numeric_limits;
```

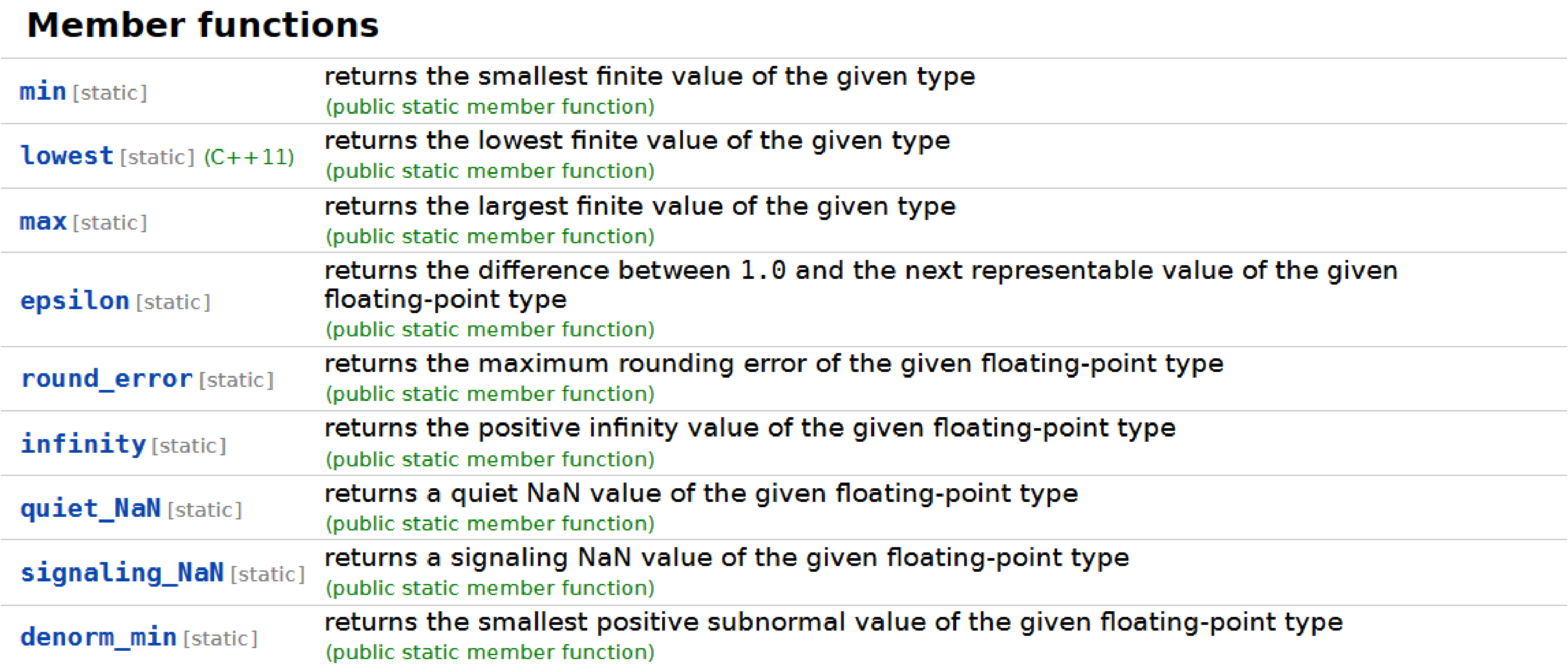
The `numeric_limits` class template provides a standardized way to query various properties of arithmetic types (e.g. the largest possible value for type `int` is `std::numeric_limits<int>::max()`).

This information is provided via specializations of the `numeric_limits` template. The standard library makes available specializations for all arithmetic types:

Defined in header `<limits>`

```
template<> class numeric_limits<bool>;
template<> class numeric_limits<char>;
template<> class numeric_limits<signed char>;
template<> class numeric_limits<unsigned char>;
template<> class numeric_limits<wchar_t>;
template<> class numeric_limits<char16_t>;   // C++11 feature
template<> class numeric_limits<char32_t>;   // C++11 feature
template<> class numeric_limits<short>;
template<> class numeric_limits<unsigned short>;
template<> class numeric_limits<int>;
template<> class numeric_limits<unsigned int>;
template<> class numeric_limits<long>;
template<> class numeric_limits<unsigned long>;
template<> class numeric_limits<long long>;
template<> class numeric_limits<unsigned long long>;
template<> class numeric_limits<float>;
template<> class numeric_limits<double>;
template<> class numeric_limits<long double>;
```

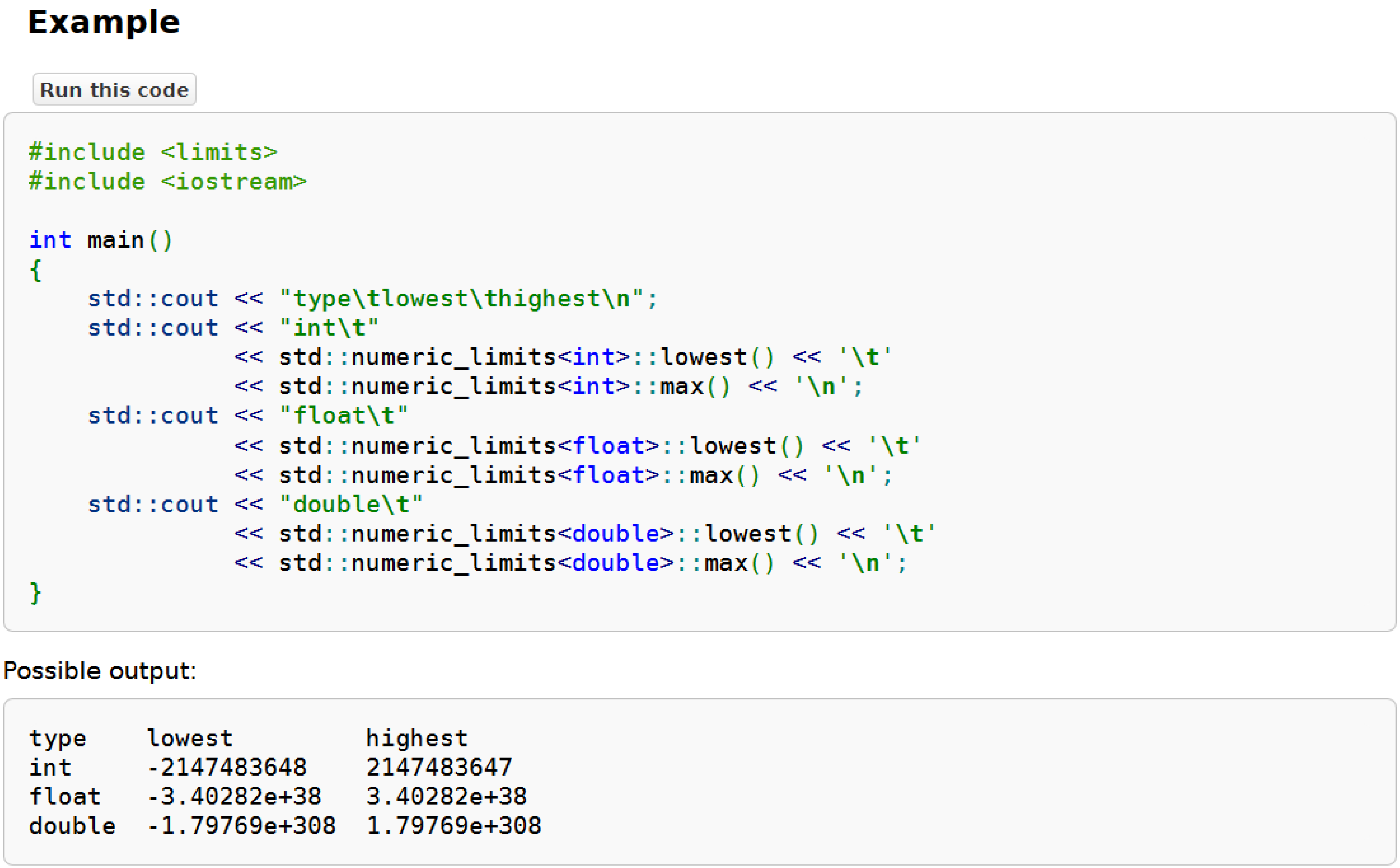
std::numeric_limits (2/4)



Member functions

min [static]	returns the smallest finite value of the given type (public static member function)
lowest [static] (C++11)	returns the lowest finite value of the given type (public static member function)
max [static]	returns the largest finite value of the given type (public static member function)
epsilon [static]	returns the difference between 1.0 and the next representable value of the given floating-point type (public static member function)
round_error [static]	returns the maximum rounding error of the given floating-point type (public static member function)
infinity [static]	returns the positive infinity value of the given floating-point type (public static member function)
quiet_NaN [static]	returns a quiet NaN value of the given floating-point type (public static member function)
signaling_NaN [static]	returns a signaling NaN value of the given floating-point type (public static member function)
denorm_min [static]	returns the smallest positive subnormal value of the given floating-point type (public static member function)

std::numeric_limits (3/4)



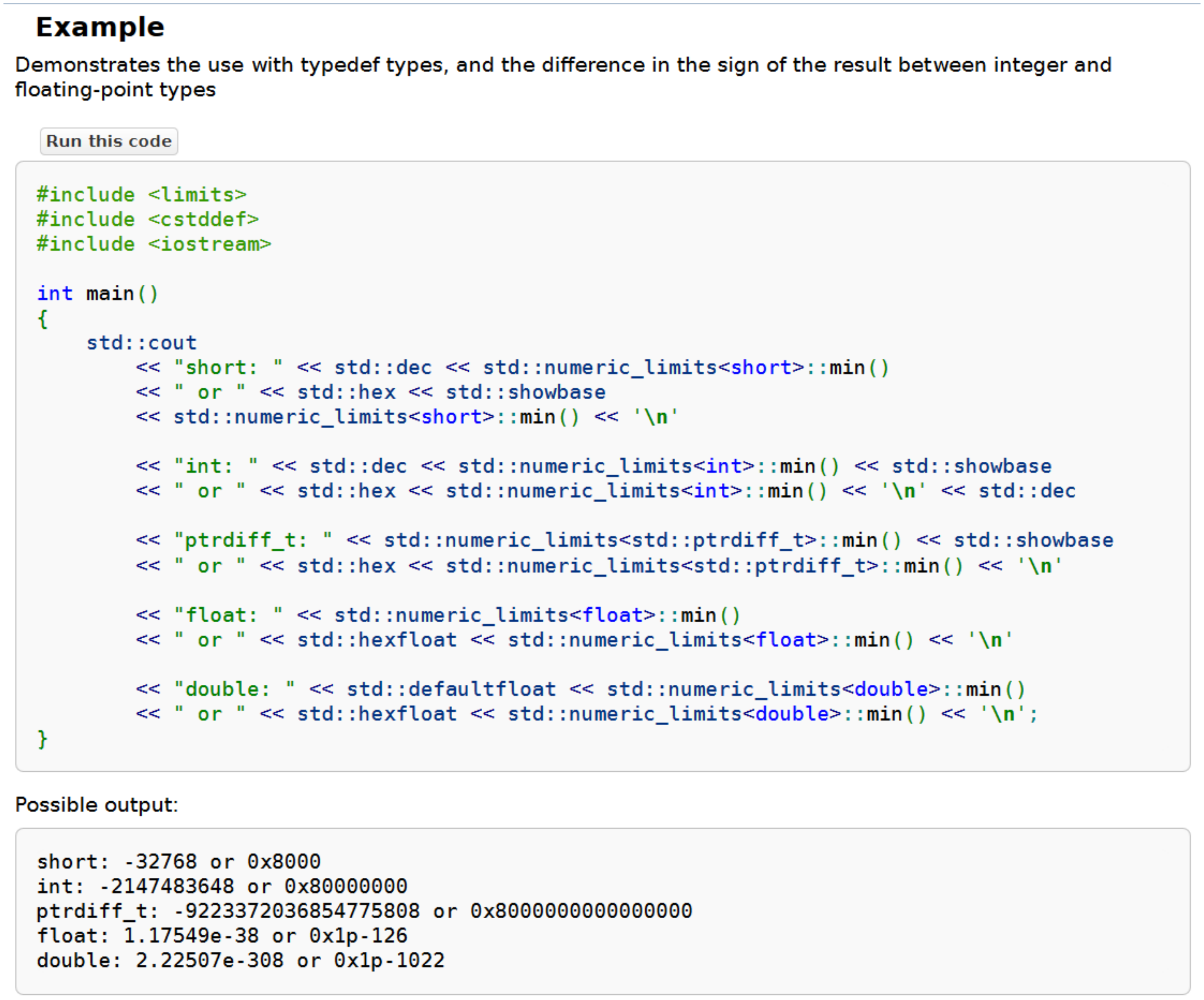
Example

```
#include <limits>
#include <iostream>

int main()
{
    std::cout << "type\tlowest\thighest\n";
    std::cout << "int\t"
              << std::numeric_limits<int>::lowest() << '\t'
              << std::numeric_limits<int>::max() << '\n';
    std::cout << "float\t"
              << std::numeric_limits<float>::lowest() << '\t'
              << std::numeric_limits<float>::max() << '\n';
    std::cout << "double\t"
              << std::numeric_limits<double>::lowest() << '\t'
              << std::numeric_limits<double>::max() << '\n';
}
```

Possible output:

```
type    lowest         highest
int     -2147483648    2147483647
float   -3.40282e+38   3.40282e+38
double  -1.79769e+308  1.79769e+308
```

std::numeric_limits (4/4)



Example

Demonstrates the use with typedef types, and the difference in the sign of the result between integer and floating-point types

Run this code

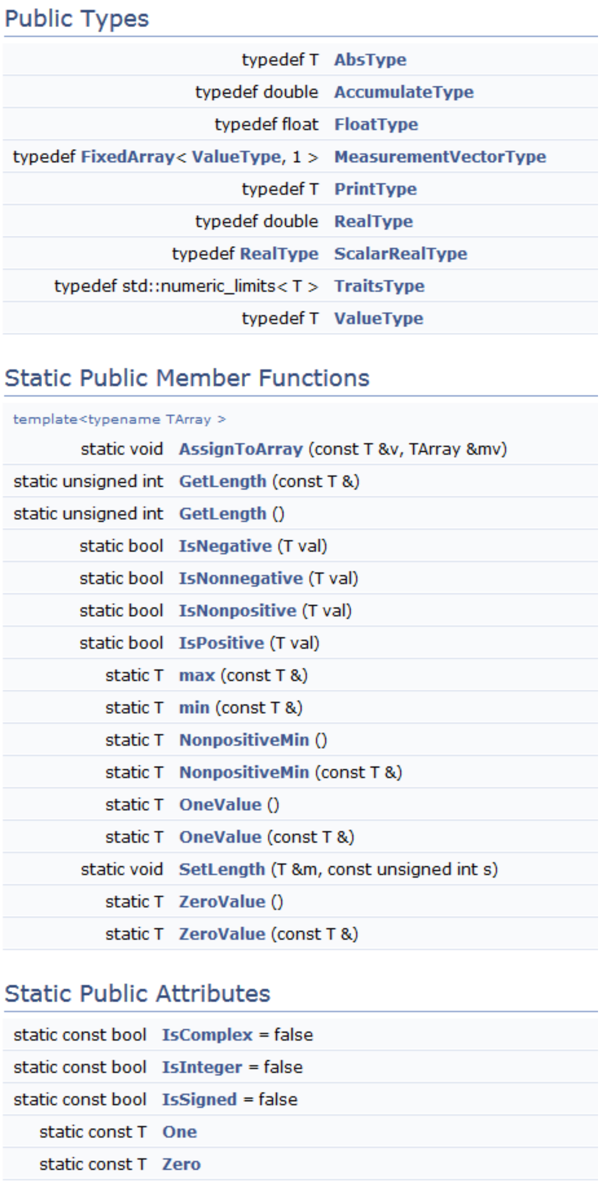
```
#include <limits>
#include <cstddef>
#include <iostream>

int main()
{
    std::cout
        << "short: " << std::dec << std::numeric_limits<short>::min()
        << " or " << std::hex << std::showbase
        << std::numeric_limits<short>::min() << '\n'

        << "int: " << std::dec << std::numeric_limits<int>::min() << std::showbase
        << " or " << std::hex << std::numeric_limits<int>::min() << '\n' << std::dec

        << "ptrdiff_t: " << std::numeric_limits<std::ptrdiff_t>::min() << std::showbase
        << " or " << std::hex << std::numeric_limits<std::ptrdiff_t>::min() << '\n'

        << "float: " << std::numeric_limits<float>::min()
        << " or " << std::hexfloat << std::numeric_limits<float>::min() << '\n'

        << "double: " << std::defaultfloat << std::numeric_limits<double>::min()
        << " or " << std::hexfloat << std::numeric_limits<double>::min() << '\n';
}
```

Possible output:

```
short: -32768 or 0x8000
int: -2147483648 or 0x80000000
ptrdiff_t: -9223372036854775808 or 0x8000000000000000
float: 1.17549e-38 or 0x1p-126
double: 2.22507e-308 or 0x1p-1022
```

itk::NumericTraits

- itk::NumericTraits<T> : an extension of std::NumericLimits<T>



Public Types

typedef T	**AbsType**
typedef double	**AccumulateType**
typedef float	**FloatType**
typedef **FixedArray**< **ValueType**, 1 >	**MeasurementVectorType**
typedef T	**PrintType**
typedef double	**RealType**
typedef **RealType**	**ScalarRealType**
typedef std::numeric_limits< T >	**TraitsType**
typedef T	**ValueType**

Static Public Member Functions

template<typename TArray >	
static void	**AssignToArray** (const T &v, TArray &mv)
static unsigned int	**GetLength** (const T &)
static unsigned int	**GetLength** ()
static bool	**IsNegative** (T val)
static bool	**IsNonnegative** (T val)
static bool	**IsNonpositive** (T val)
static bool	**IsPositive** (T val)
static T	**max** (const T &)
static T	**min** (const T &)
static T	**NonpositiveMin** ()
static T	**NonpositiveMin** (const T &)
static T	**OneValue** ()
static T	**OneValue** (const T &)
static void	**SetLength** (T &m, const unsigned int s)
static T	**ZeroValue** ()
static T	**ZeroValue** (const T &)

Static Public Attributes

static const bool	**IsComplex** = false
static const bool	**IsInteger** = false
static const bool	**IsSigned** = false
static const T	**One**
static const T	**Zero**

Questions?