- Solution

In this lesson, we will look at the solution to using assertations to ensure that the algorithm only accepts natural numbers.

WE'LL COVER THE FOLLOWING

Example

```
// staticAssertGcd.cpp

#include <iostream>
#include <type_traits>

template<typename T>
T gcd(T a, T b){
  static_assert(std::is_integral<T>::value, "T should be an integral type!");
  if( b == 0 ){ return a; }
  else{
    return gcd(b, a % b);
  }
}

int main(){

  std::cout << std::endl;

  std::cout << gcd(3.5, 4.0)<< std::endl; // should be gcd(3, 4)
  std::cout << gcd("100", "10") << std::endl; // should be gcd(100, 10)

  std::cout << std::endl;

}
```

Explanation

The `static_assert` operator and the predicate `std::is_integral<T>::value` enable us to check at compile time whether or not `T` is an integral type. A

predicate always returns a `boolean` value.

The compilation will not fail by accident since the modulo operator is not defined for a `double` value and a C string, but the compilation fails since the assertion in line 8 will not hold true. Now, we get an exact error message rather than a cryptic output of a failed template instantiation.

> The rule is quite simple: **If a compilation must fail, we will get an unambiguous error message. **

For further information, see static_assert.

The power of `static_assert` can be evaluated at compile time. We have the new type-traits library in C++11. The powerful type-traits library empowers you to check, compare, and change types at compile time. We will discuss this concept in the next section.