CS205 - C/C++ Programming Lab Assignment 7

In this lab, you are asked to implement a template to determine whether a given type is incrementable (i.e. **both** ++*obj* and *obj++* compiles) at compile time using C++'s ~~in~~famous template metaprogramming technique.

We assume that if the type is incrementable, both *operator++* are public.

Your template should be able to be called by the following code:

std::cout << *is_incrementable*<int>() << *std::endl*;

and *is_incrementable*<T>() returns a *bool*.

For example,

std::cout << *std::boolalpha* << *is_incrementable*<int>() << *std::endl*;

std::cout << *is_incrementable*<std::string>() << *std::endl*;

should prints *true* and *false* respectively.

To implement this, we need two templates, one for incrementable types (the primary template), another for non-incrementable ones. The specialization will be based on SFINAE. Thus, we are going to write code that attempts to increment T in the first specialization. If T is indeed incrementable, the code will be valid, and the specialization will be instantiated. The first template is the one that inherit from *std::true_type*.

On the other hand, if T is non-incrementable, then the first specialization won't be valid. In this case SFINAE says that an invalid instantiation doesn't stop compilation. It is just discarded, and falling back to the second template, the one inheriting from *std::false_type*.

To call *operator++*, we need to obtain a reference of type T. The standard library has *std::declval*<T>. Note that we need to substitute T& in this case, because *std::declval*<T> returns a rvalue, which is non-incrementable whatsoever, and *std::declval*<T&> returns a lvalue.

All the documentations you may need:

- https://en.cppreference.com/w/cpp/language/template_parameters
- https://en.cppreference.com/w/cpp/language/template_specialization
- https://en.cppreference.com/w/cpp/language/partial_specialization
- https://en.cppreference.com/w/cpp/language/sfinae
- https://en.cppreference.com/w/cpp/types/integral_constant
- https://en.cppreference.com/w/cpp/utility/declval
- https://en.cppreference.com/w/cpp/language/decltype
- https://en.cppreference.com/w/cpp/types/void_t
- https://en.cppreference.com/w/cpp/language/reference

Note 1: Please submit your answer in a file named "**template.hpp**", and do **not** include any main function in the header.

Note 2: This assignment will be compiled at C++17 language level.

Hint: use *std::true_type* to represent true and *std::false_type* otherwise.