【C++】Day84

Class	C++
Date	@April 25, 2022
Material	
Series Number	
Summary	Template Specialization

【Ch16】Templates and Generic Programming

16.5 Template Specializations

It is not always possible to write a single template that is best suited for every possible template argument with which the template might be instantiated.

Our `compare` function is a good example of a function template for which the general definition is not appropriate for a particular type, namely, character poitners.

We can overload the compare function to handle character string literals:

```
// First version: can compare any two types
template <typename T> int compare(const T&, const T&);
// Second version: handle string literals
template <size_t N, size_t M>
int compare(const char (&) [N], const char (&) [M]);
```

However, the version of `compare` that has two nontype template parameters will be called only when we pass a string literal or an array.

If we call compare with character pointers, the first version of the template will be called:

```
const cahr *p1 = "hi", *p2 = "mom";
compare(p1, p2); // Calls the first template
compare("hi", "mom"); // Calls the template with two nontype parameters.
```

To handle character poitners, we can define a template specialization of the first version of compare.

Defining a Function Template Specialization

When we specialize a function template, we must supply arguments for every template parameter in the original template.

To indicate that we are specializing a template, we use the keyword template followed by an empty pair of angle brackets(`< >`). The empty brackets indicate that arguments will be supplied for all the template parameters of the original template:

```
// Special version of compare to handle pointers to character arrays
template <>
int compare(const char* const &p1, const char* const &p2) {
  return strcmp(p1, p2);
}
```

When we define a specialization, the function parameter types must match the corresponding types in a previously declared template.

Here we are specializing:

```
template <typename T>
int compare(const T&, const T&);
```

in which the function parameters are references to a `const` type.