【C++】Day80

Class	C++
Date	@April 11, 2022
Material	
Series Number	
Summary	Class Template

【Ch16】Templates and Generic Programming

16.1.2 Class Templates

A class template is a buleprint for generating classes.

Defining a Class Template

We'll write a template `Blob`. Our template will provide shared access to the elements it holds.

Like function templates, class templates begin with the keyword template followed by a template parameter list.

```
template <typename T> class Blob {
public:
  // Rename T to be value_type
  typedef T value_type;
  typedef std::vector<T>::size_type size_type;

  // constructors
  Blob();
  Blob(std::initializer_list<T> il);

  // Number of elements in the Blob
  size_type size() const { return data->size(); }
  bool empty() const { return data->empty(); }

  // Add and remove elements
  void push_back(const T &t) { data->push_back(t); }
  void pop_back();

  // Element access
  T& back();
  T& operator(size_type i);
```

```
private:
  std::shared_ptr<vector<T>> data;
  // throws msg if data[i] isn't valid
  void check(size_type i, const std::string &msg) const;
};
```

Instantiating a Class Template

When we use a class template, we must supply extra information. We can now see that extra information is a list of explicit template arguments that are bound to the template's parameters.

For example, to define a type from our `Blob` template, we must provide the element type:

```
Blob<int> ia;
Blob<int> ia2 = (0, 1, 2, 3, 4};
```

When the compiler instantiates a class from our `Blob` template, it rewrites the `Blob` template, replacing each instance of the template parameter `T` by the given template argument, which in this case is `int`.

Note: Each instantiation of a class template constitutes an independent class. The type Blob<string> has no relationship to, or any special access to, the members of any other Blob type.

References to a Template Type in the Scope of the Template

Code in a class template doesn't use the name of an actual type as a template argument.
Instead, we often use the template's own parameters as the template arguments.

For example, our data members uses two template, `vector` and `shared_ptr`.
Whenever we use a template, we must supply template arguments.

```
std::shared_ptr<std::vector<T>> data;
```