【C++】Day40(1)

Class	C++
Date	@January 18, 2022
Material	
Series Number	
Summary	

【Ch10】Generic Algorithms

10.3.4 Binding Arguments

Lambda expressions are most useful for simple operations that we do not need to use in more than one or two places.

If we need to do the same operation in many places, we should usually define a function rather than writing the same lambda expression multiple times. Similarly, if an operation requires many statements, it is ordinarily better to use a function.

However, it is not so easy to write a function to replace a lambda that captures local variables. For example, the lambda that we used in the call to `find_if` compared a string with a given size. We can easily write a function to do the same work:

```
bool check_size(const string &s, string::size_type sz) {
  return s.size() >= sz;
}
```

However, we cannot use this function as an argument to `find_if`. As we've seen, `find_if` takes a unary predicate, so the callable passed to `find_if` must take a single argument. In order to use `check_size` in place of that lambda, we have to figure out how to pass an argument to the `sz` parameter.

The library bind Function

We can solve the problem of passing a size argument to `check_size` by using a new library function named `bind`, which is defined in the `functional` header.

The `bind` function can be thought of as a general-purpose function adaptor. It takes a callable object and generates a new callable that “adapts” the parameter list of the original object.

The general form of a call to bind is:

```
auto newCallable = bind(callable, arg_list);
```

The arguments in `arg_list` may include names of the form _n, where n is an integer. These arguments are “placeholders” representing the parameters of `newCallable`. They stand “in place of” the arguments that will be passed to `newCallable`.

The number n is the position of the parameter in the generated callable: `_1` is the first parameter in the `newCallable`, `_2` is the second, and so forth.

Binding the sz Parameter of check_size

As a simple example, we’ll use bind to generate an object that calls `check_size` with a fixed value for its size parameter as follows:

```
//check6 is a callable object that takes one argument of type string
//and calls check_size on its given string and the value 6
auto check6 = bind(check_size, std::placeholders::_1, 6);
```

This call to bind has only one placeholder, which means that `check6` takes a single argument. The placeholder appears first in `arg_list`, which means that the parameter in check6 corresponds to the first parameter of `check_size.` That parameter is a `const string&`. Thus, a call to check6 must pass an argument of type string, which check6 will pass as the first argument to `check_size`.

Using placeholders Name

The `_n` names are defined in a namespace named placeholders. That namespace is itself defined inside the `std` namespace.

To use the _n as the placeholder, we can use using or explicit state the namespace:

```
using std::placeholders::_1;
```

Arguments to Bind

We can use find to fix the value of a parameter. More generally, we can use bind to bind or rearrange the parameters in the given callable.

For example, assuming f is a callable object that has five parameters, the following call to bind:

```
//g is a callable object that takes two arguments
auto g = bind(f, a, b, _2, c, _1);
```

The arguments to g are bound positionally to the placeholders.

If we call

```
g(_1, _2);
```

This call to bind maps to

```
f(a, b, _2, c, _1);
```

Calling g calls f using g's arguments for the placeholders along with the bound arguments a, b, and c.

For example, calling g(X, Y) calls:

```
f(a, b, Y, c, X);
```

Exercise

Exercise 10.22: Rewrite the program to count words of size 6 or less using functions in place of the lambdas.

See 10_22.cpp for code

Exercise 10.23: How many arguments does `bind` take?

The arguments of bind depends on the callable we pass in. It should be the same as the callable.

Exercise 10.24: Use `bind` and `check_size` to find the first element in a `vector` of `int`s that has a value greater than the length of a specified `string` value.

See 10_24.cpp for code