【Effective CPP】Day4

Book	Effective C++
Author	
Summary	
Date	@2022/05/11

【Ch2】Constructors, Destructors, and Assignment Operators

Item 5: Know what functions C++ silently writes and calls

If we don't declare our own versions of a copy constructor, a copy assignment operator, and a destructor, the compiler will declare them for us.

All these functions will be `public` and `inline`.

If we write:

```
class Empty {};
```

It's essentially the same as if we'd written this:

```
class Empty {
public:
  Empty() {};
  Empty(const Empty& rhs) {}
  Empty& operator=(const Empty& rhs) {}
  ~Empty() {}
};
```

These functions are generated only if they are needed. The following code will cause each function to be generated.

```
Empty e1; // Default constructor
Empty e2(e1); // Copy constructor
e2 = e1; // Copy assignment operator
```

As for the copy constructor and the copy assignment operator, the compiler-generated versions simply copy each non-static data member of the source object over to the target object.

For example, consider a `NamedObject` template that allows us to associate names with objects of type T:

```
template <typename T>
class NamedObject {
public:
  NamedObject(const std::string& name, const T& value);
  NamedObject(const char* name, const T& value);

private:
  std::string nameValue;
  T objectValue;
};
```

Because a constructor is declared in `NamedObject`, compilers won't generate a default constructor.

`NamedObject` declares neither copy constructor nor copy assignment operator, so compilers will generate those functions(if they are needed).

```
NamedObject<int> no1("Smallest Prime Number", 2);
NamedObject<int> no2(no1);
```

The copy constructor generated by compilers must initialize `no2.nameValue` and `no2.objectValue` using `no1.nameValue` and `no1.objectValue`.

The type of `nameValue` is string, and the standard string type has a copy constructor, so `no2.nameValue` will be initialized by calling the string copy constructor with no1.nameValue as its argument.

On the other hand, the type of `NamedObject<int>::objectValue` is int, and int is a built-in type, so `no2.objectValue` will be initialized by copying the bits in `no1.objectValue`.

Things to Remember

Compilers may implicitly generate a class's default constructor, copy constructor, copy assignment operator, and destructor.