【Effective CPP】Day4(2)

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

【Ch2】Constructors, Destructors, and Copy Assignment Operators

Item 6: Explicitly disallow the use of compiler-generated functions we do not want

We can declare the copy constructor and copy assignment operator explicitly as private, and not to define them in order to prevent copying.

This technique is so well established and is used in C++'s iostreams library.

Applying the trick to `HomeForSal`e class:

```
class HomeForSale {
public:
  ...
private:
  ...
  HomeForSale(const HomeForSale&);
  HomeForSale& operator=(const HomeForSale&);
};
```

If we try to use the copy constructor and copy assignment operator in a member or a friend function, the linker will complain.

It is possible to move the link-time error up to compile time.

We can declare the copy constructor and copy assignment operator private not in `HomeForSale` itself ,but in a base class specifically designed to prevent copying. The base class is simply itself:

```
class Uncopyable {
protected:
  Uncopyable() {} // Allow construction and destruction of derived objects
  ~Uncopyable() {}

private:
  Uncopyable(const Uncopyable&);
  Uncopyable& operator=(const Uncopyable&);
};
```

```
 class HomeForSale : private Uncopyable {};
```

Things to Remember

To disallow functionality automatically provided by compilers, declare the corresponding member functions private and give no implementations. Using a base class like `Uncopyable` is one way to do this.