【C+】Day69(2)

Class	C++
Date	@March 7, 2022
Material	
Series Number	
Summary	Avoid Ambiguity

【Ch14】Overloaded Operations and Conversions

14.9.2 Avoiding Ambiguous Conversions

If a class has one or more conversions, it is important to ensure that there is only one way to convert from the class type to the target type.

There are two ways that multiple conversion paths can occur. The first happens when two classes provide mutual conversions.

The second way to generate multiple conversion paths is to define multiple conversions from or to types that are themselves related by conversions.

Warning: Ordinarily, it is a bad idea to define classes with mutual conversions or to define conversions to or from two arithmetic types.

Argument Matching and Mutual Conversions

In the following example, we 've defined two ways to obtain an A from a B: either by using B's conversion operator or by using the A constructor that takes a B:

```
// usually a bad idea to have mutual conversions between two class types
struct B;
struct A {
  A() = default;
  A(const B&); // converts a B to an A
};

struct B {
  operator A() const; // also converts a B to an A
};

A f(const A&);
B b;
A a = f(b); // error ambiguous
```

Because there are two ways to obtain an A from a B, the compiler doesn’t know which conversion to run; the call to f is ambiguous.

If we want to make this call, we have to explicitly call the conversion operator or the constructor:

```
A a1 = f(b.operator A()); // ok: use B's conversino operator
A a2 = f(A(b)); // ok: use A's constructor
```