1. Except for the overloaded function-call operator, **operator()**, an overloaded operator may not have default arguments.
2. When an overloaded operator is a member function, **this** is bound to the left-hand operand. Member operator functions have one less (explicit) parameter than the number of operands.
3. Ordinarily, the comma, address-of, logical AND, and logical OR operators should not be overloaded.
4. Symmetric operators—those that might convert either operand, such as the arithmetic, equality, relational, and bitwise operators—usually should be defined as ordinary nonmember functions.
5. Generally, **output operators** should print the content of the object, with minimal formatting. They should not print a newline.
6. **Input operators** must deal with the possibility that the input might fail; output operators generally don’t bother.
7. Input operators should decide what, if anything, to do about error recovery.
8. If a single logical definition for **<** exists, classes usually should define the **< operator**. However, if the class also has **==,** define **<** only if the definitions of **<** and **==** yield consistent results.
9. Assignment operators can be overloaded. Assignment operators, regardless of parameter type, must be defined as member functions.
10. If a class has a **subscript operator**, it usually should define two versions: one that returns a plain **reference** and the other that is const member and returns a **reference to const**.
11. The overloaded **arrow operator** ***must*** (unlike other overloaded operators) return either a pointer to a class type or an object of a class type that defines its own operator arrow.
12. A call signature corresponds to a function type.
13. different types can have the same call signature. [Example](Overloaded%20Operations%20and%20Conversions.cpp)
14. Conversion operators can be defined for any type (other than void) that can be a function return type.
15. A conversion function must be a member function, may not specify a return type, and must have an empty parameter list. The function usually should be **const**.
16. **explicit** Conversion Operators.
    1. If the conversion operator is explicit, we can still do the conversion. However, with one exception, we must do so explicitly through a cast.
    2. The exception is that the compiler will apply an explicit conversion to an expression used as a condition. That is, an **explicit** conversion will be used implicitly to convert an expression used as
       1. The condition of an **if**, **while**, or **do** statement
       2. The condition expression in a **for** statement header
       3. An operand to the logical **NOT** (**!**), **OR** (**||**), or **AND** (**&&**) operators
       4. the condition expression in a **conditional** (**? :**) **operator**
17. Conversion to **bool** is usually intended for use in conditions. As a result, **operator bool** ordinarily should be defined as **explicit**.
18. Ordinarily, it is a bad idea to define classes with mutual conversions or to define conversions to or from two arithmetic types.
19. When two user-defined conversions are used, the rank of the **standard conversion**, if any, preceding or following the conversion function is used to select the best match.
20. In a call to an overloaded function, the rank of an additional **standard conversion** (if any) matters only if the viable functions require the same **user-defined conversion**. If different **user-defined conversions** are needed, then the call is ambiguous.
21. Providing both **conversion functions** to an arithmetic type and **overloaded operators** for the same class type may lead to ambiguities between the overloaded operators and the **built-in operators**.