1. When we use copy initialization, we are asking the compiler to copy the right-hand operand into the object being created, converting that operand if necessary. Example
2. When we use direct initialization, we are asking the compiler to use ordinary function matching to select the ctor that best matches the arguments we provide.
3. Assignment operators ordinarily should return a reference to their left-hand operand.
4. In a dtor, the function body is executed first and then the members are destroyed. Members are destroyed in reverse order from the order in which they were initialized.
5. If a class need a dtor, it almost surely also needs the copy-assignment operator and a copy constructor.
6. The Copy-Control Members May Be synthesized as Deleted
   1. The synthesized dtor is defined as deleted if the class has a member whose owen dtor is deleted or is inaccessible
   2. The synthesized copy ctor is defined as deleted if the class has a member whose own copy ctor is deleted or inaccessible. It is also deleted if the class has a member with a deleted or inaccessible dtor.
   3. The synthesized copy-assignment operator is defined as deleted if a member has a deleted or inaccessible copy-assignment operator, or if the class has a **const** or **reference** member.
   4. The synthesized default ctor is defined as deleted if the class has a member with a deleted or inaccessible dtor; or has a reference member that does not have an in-class initializer; or has a const member whose type dose not explicitly define a default constructor and that member dose not have an in-class initializer.
7. It is crucially important for assignment operators to work correctly, even when an object is assigned to itself. A good way to do so is to copy the right-hand operand before destroying the left-hand operand.
8. Assignment operators that use copy and swap are automatically exception safe and correctly handle self-assignment.
9. The copy-assignment operator often dose the same work as is needed in the copy ctor and dtor. In such cases, the common work should be put in **private** utility functions.
10. The library containers, **string**, and **shared\_ptr** classes support move as well as copy. The **IO** and **unique\_ptr** classes can be moved but not copied.
11. **Rvalue** **reference** refer to objects that are about to be destroyed. Hence, we can “steal” state from an obj bound to an **rvalue** **reference**.
12. A variable is an lvalue; we cannot directly bind an rvalue reference to a variable even if that variable was defined as an rvalue reference type.
13. We can destroy a moved-from object and can assign a new value to it, but we cannot use the value of a moved-from object.
14. **Move ctor and move assignment operators** that cannot throw exceptions should be marked as **noexcept**;
15. After a **move** operation, the “moved-from” object must remain a valid, destructible object but users may make no assumptions about its value.
16. The compiler synthesizes the move ctor and move assignment only if a class does not define any of its own copy-control members and only if all the data members can be moved constructed and move assigned, respectively.
17. Classes that define a **move constructor** or **move-assignment operator** must also define their own **copy operations**. Otherwise, those members are deleted by default.
18. If a class has a usable copy ctor and no move ctor, objects will be “moved” by the copy ctor. Similarly, for the copy-assignment operator and move-assignment.
19. Outside of class implementation code such as **move constructors** **move-assignment operators**, use std::move only when you are certain that you need to do a move and that the move is guaranteed to be safe
20. Overloaded functions that distinguish between moving and copying a parameter typically have one version that takes a **const T&** and one that takes a **T&&**
21. A function can be both **const** and **reference qualified**. In such cases, the **reference qualifier** must follow the **const qualifier**. [Example](Copy%20Control.cpp)
22. If a member function has a **reference qualifier**, all the versions of that member with the same parameter list must have **reference qualifiers**.