- 3.3.2, 17.5.2, 23.5.2.1
- rules for argument passing
    i. use pass-by-value for small objects
    ii. use pass-by-const-reference to pass large values that you don't need to modify
    iii. return a result as a return value rather than modifying an object through an argument
    iv. use rvalue references to implement move and forwarding
    v. pass a point if "no object" is a valid alternative (use nullptr for "no object")
    vi. use pass-by-reference only if you have to

# 3 Copy and Move

- by default the meaning of copy is memberwise copy
- always consider the semantics of copy for your class carefully
- abstract types should rarely use memberwise copy

## Copying Containers

- for containers, prefer copying each element individually
- if a container is a resource handle, i.e. responsible for an object accessed through a pointer, prefer to not copy the pointer (possibly create a new heap object that is a copy of the original)

## Moving Containers

- after moving from an object the object should be in a state that allows the destructor to be run

# 17 Construction, Cleanup, Copy, and Move

## Move

- for resource handles, move operations tend to be significantly simpler and more efficient than copy
- move operations should not throw as they do not acquire resources, they just transfer already acquired resources
- move for built-in/primitive types is generally just copy

- if a class has a reference member it probably needs copy operations
- if a class is a resource handle it probably needs copy and move
- for copy, copy every element that needs to be copied (beware defaults)
- copy operations should provide equivalence and independence
- prefer move semantics and copy-on-write to shallow copy
- if a class is used as a base class, protect against slicing; §17.5.1.4.
- if a class needs a copy operation or a destructor, it probably needs a constructor, a destructor, a copy assignment, and a copy constructor
- if a class has a pointer member, it probably needs a destructor and non-default copy operations
- if a class is a resource handle, it needs a constructor, a destructor, and non-default copy operations
- if a default constructor, assignment, or destructor is appropriate, let the compiler generate it (don’t rewrite it yourself)
- be explicit about your invariants; use constructors to establish them and assignments to maintain them
- make sure that copy assignments are safe for self-assignment
- when adding a new member to a class, check to see if there are user-defined constructors that need to be updated to initialize the member

# 23 Templates

- if a class template should be copyable, give it a non-template copy constructor and a non-template copy assignment
- if a class template should be movable, give it a non-template move constructor and a non-template move assignment