# CS 247: Software Engineering Principles

## Special Member Functions

Readings: Eckel, Vol. 1
- Ch. 11 References and the Copy Constructor
- Ch. 12 Operator Overloading ( operator= )

# Special Member Functions

C++ member functions that are so important that the compiler will provide default versions if we don't provide them

- default constructor (generated iff we define *no* constructor)
- destructor
- copy constructor
- assignment ( operator= )
- move constructor
- move assignment

# Compiler-Generated Default Constructor

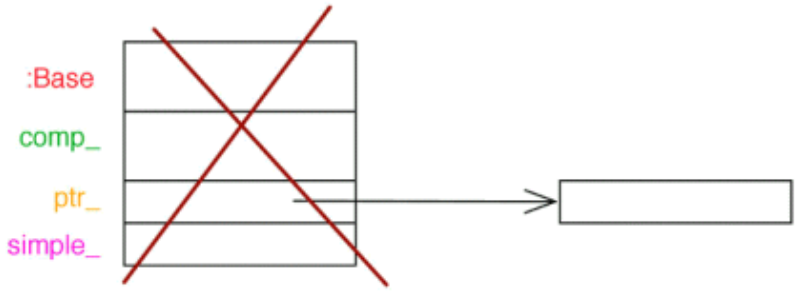
If we do not declare any constructor for our class, the compiler will generate a default constructor for us: based on memberwise initialization

- simple data members (built-in types): uninitialized
- pointer members: uninitialized
- member objects: initialized using members' default constructors
- inherited members: initialized using base class default constructor

# Compiler-Generated Destructor

If we do not declare a destructor for our class, the compiler will generate a destructor for us: based on memberwise destruction

- simple data members: deallocated
- pointer members: pointer deallocated (not deleted)
- member objects: cleaned up using members' destructors
- inherited members: cleaned up using base class's destructor



# Copy Constructor

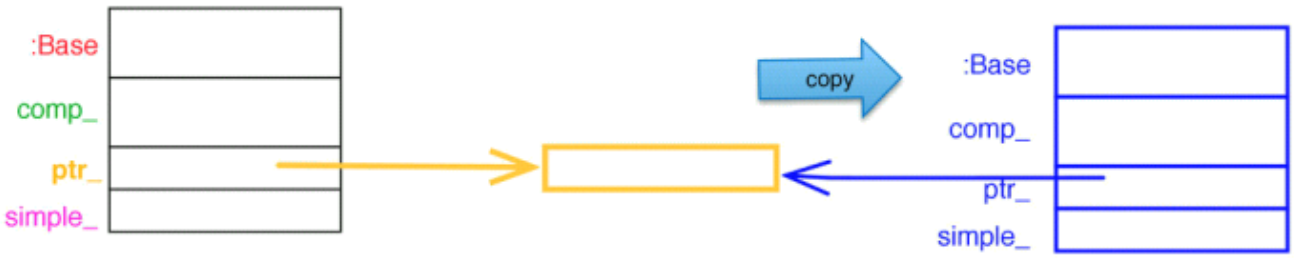
A copy constructor constructs a new object whose value is equal to an existing object.

- used by the compiler to copy objects of the ADT

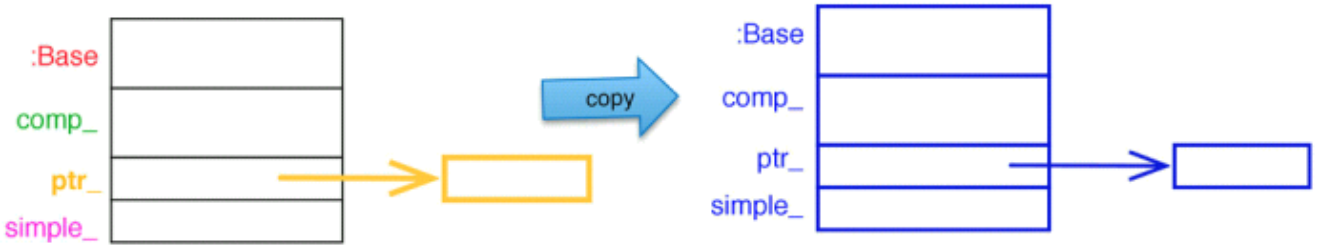
```
class Money;

Money operator+ (Money m, Money n);

int main() {
  Money m;
  Money n{m};
  Money p = m;
  p = p + n;
}
```

# Copying Objects with Pointers

Shallow copy copies the object and its pointers' addresses, so that the original and copied pointers refer to the same object.



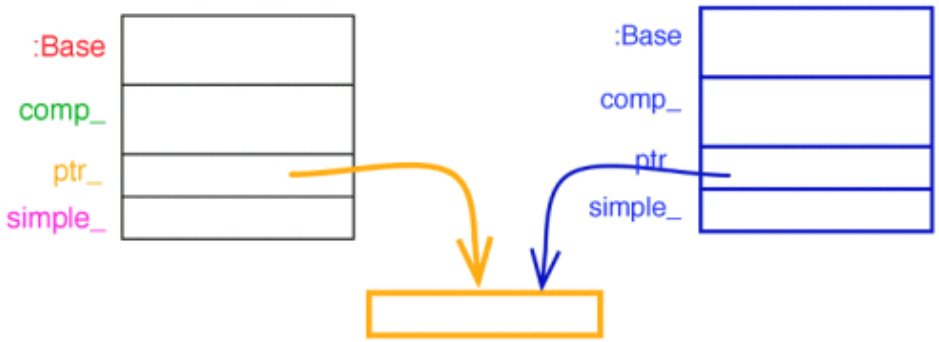

Deep copy copies the object and what its pointers point to, so that the pointer data members refer to distinct objects.



# Compiler-Generated Copy Constructor

If we do not provide a destructor, copy/move constructor, copy/move assignment, the compiler will generate a copy constructor for us: based on memberwise copy

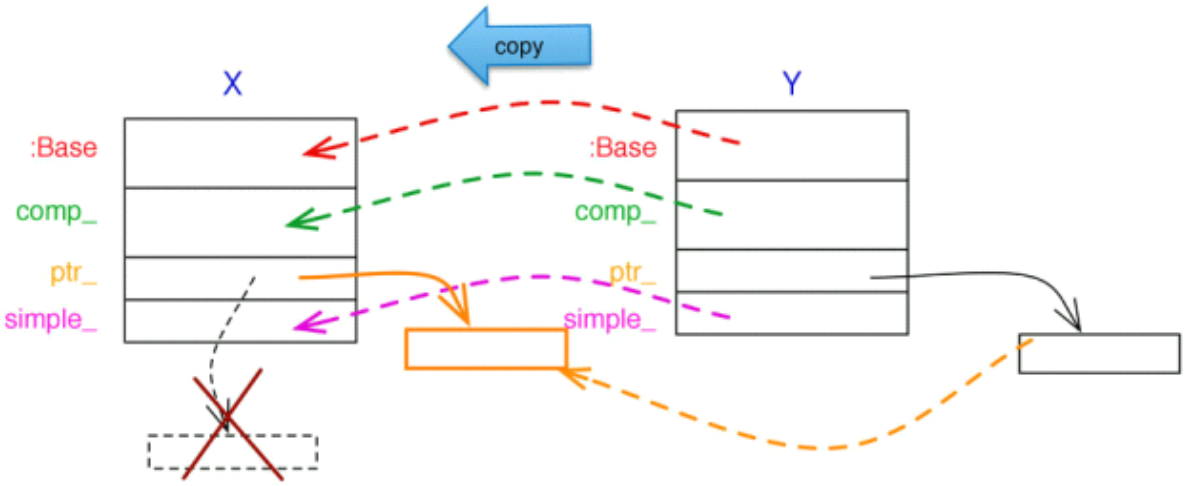
- simple data members: bitwise copy
- pointer members: bitwise copy
- member objects: copied using members' copy constructors
- inherited members: copied using base class's copy constructor



# Assignment Operator

Copy assignment is similar to the copy constructor, except that the destination of the copy already exists.

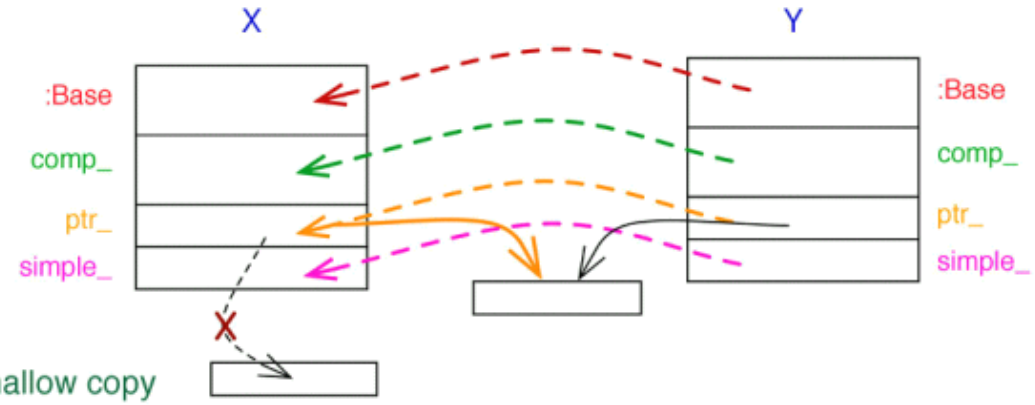
```
X = Y;   // deep copy
```



# Compiler-Generated Assignment Operator

If we do not provide a destructor, copy/move constructor, copy/move assignment, the compiler will create a copy `operator=` member function for us: based on memberwise assignment:

- simple data members: bitwise copy
- pointer members: bitwise copy
- member objects: uses members' assignment operators
- inherited members: uses base class's assignment operator



# Copy-Swap Idiom

Assignment operators may deal with pointer members by

- creating a new object of the same type with the copy constructor
- swapping the old values of the pointer members with the values in the newly created object
- letting the destructor take care of deleting the old members
- but at the cost of efficiency!!

```
MyClass& MyClass::operator= (const MyClass& m) {
   MyClass copy{m};      // 1. MyClass copy constructor
   C* temp;

   temp = copy.ptr_;       // 2. swap ptr data members
   copy.ptr_ = ptr_;
   ptr_ = temp;

   return *this;         // 3. Return reference to self
}
```

# Move Constructor

A move constructor constructs a new object whose value is equal to an existing object, but does not preserve the value of the existing object.

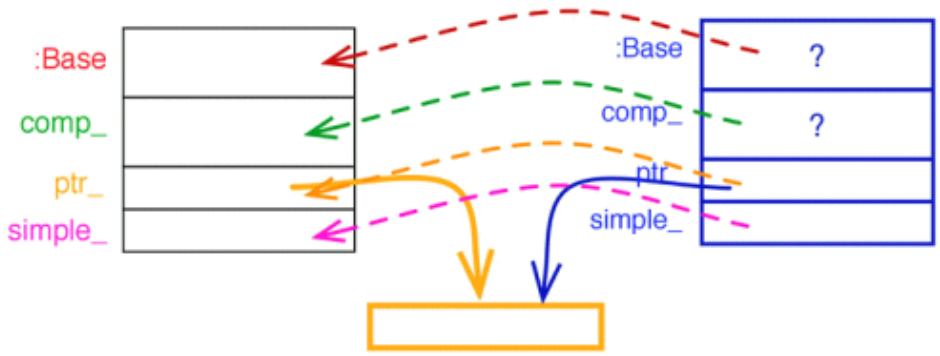
```
MyClass::MyClass (MyClass&& m) :
   Base{m}, simple_{m.simple}, comp_{m.comp_}, ptr_{m.ptr_} {

   m.ptr_ = nullptr;
}
```

Only requirement of moved-from object is that it be easy to delete and (copy) reassign.

# Compiler-Generated Move Constructor

If we do not provide a destructor, copy/move constructor, copy/move assignment, the compiler will generate a move constructor for us: based on memberwise move

- simple data members: bitwise copy
- pointer members: bitwise copy
- member objects: uses members' move/copy constructors
- inherited members: uses base class's move/copy constructor



# Move Assignment

Move assignment is similar to the move constructor, except that the destination of the move already exists.

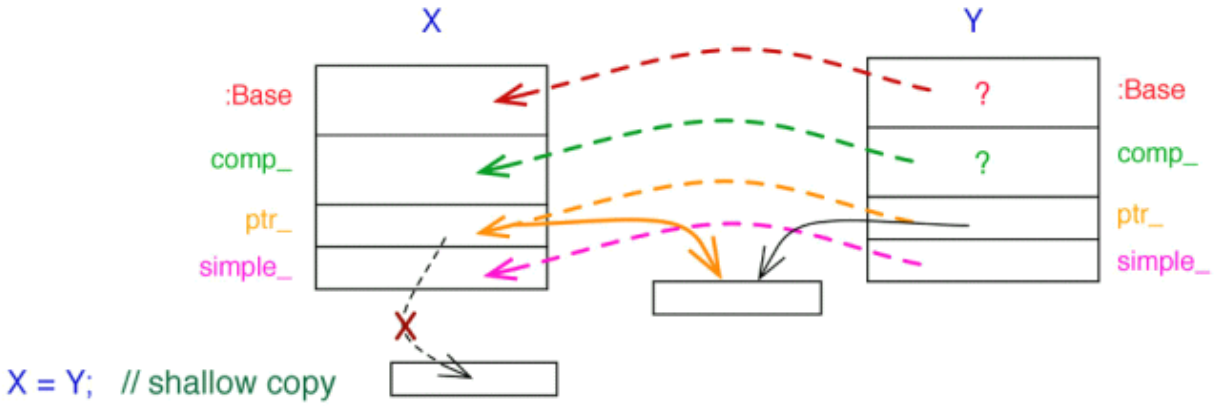
Only requirement of moved-from object is that it be easy to delete and (copy) reassign.

```
MyClass& MyClass::operator= (MyClass&& m) {
   Base::operator=(m);
   simple_ = m.simple_;
   swap( comp_, m.comp_ );
   swap( ptr_, m.ptr_ );

   return *this;
}
```

# Compiler-Generated Move Assignment

If we do not provide a destructor, copy/move constructor, copy/move assignment, the compiler will create a move `operator=` member function for us: based on memberwise move assignment:

- simple data members: bitwise copy
- pointer members: bitwise copy
- member objects: uses members' move/copy assignment
- inherited members: uses base class's move/copy assignment



http://accu.org/content/conf2014/Howard_Hinnant_Accu_2014.pdf

# Special Members

compiler implicitly declares

| user declares | default constructor | destructor | copy constructor | copy assignment | move constructor | move assignment |
|---|---|---|---|---|---|---|
| Nothing | defaulted | defaulted | defaulted | defaulted | defaulted | defaulted |
| Any constructor | not declared | defaulted | defaulted | defaulted | defaulted | defaulted |
| default constructor | user declared | defaulted | defaulted | defaulted | defaulted | defaulted |
| destructor | defaulted | user declared | defaulted | defaulted | not declared | not declared |
| copy constructor | not declared | defaulted | user declared | defaulted | not declared | not declared |
| copy assignment | defaulted | defaulted | defaulted | user declared | not declared | not declared |
| move constructor | not declared | defaulted | deleted | deleted | user declared | not declared |
| move assignment | defaulted | defaulted | deleted | deleted | not declared | user declared |

http://accu.org/content/conf2014/Howard_Hinnant_Accu_2014.pdf

# Special Members

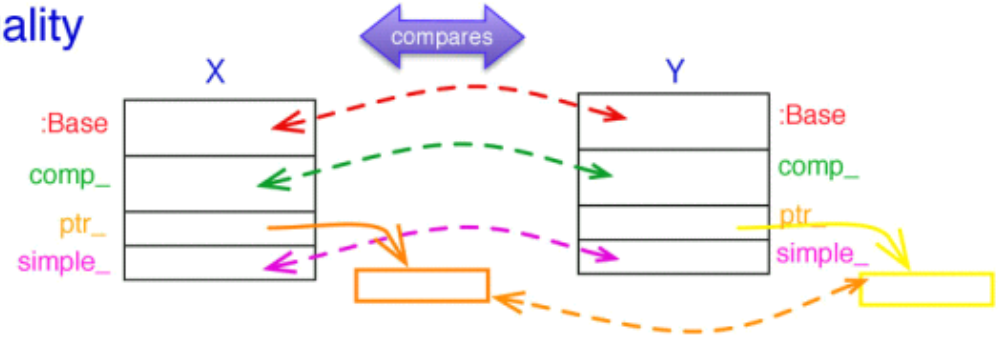
compiler implicitly declares

| user declares | default constructor | destructor | copy constructor | copy assignment | move constructor | move assignment |
|---|---|---|---|---|---|---|
| Nothing | defaulted | defaulted | defaulted | defaulted | defaulted | defaulted |
| Any constructor | not declared | defaulted | defaulted | defaulted | defaulted | defaulted |
| default constructor | user declared | defaulted | defaulted | defaulted | defaulted | defaulted |
| destructor | defaulted | user declared | defaulted | defaulted | not declared | not declared |
| copy constructor | not declared | defaulted | user declared | defaulted | not declared | not declared |
| copy assignment | defaulted | defaulted | defaulted | user declared | not declared | not declared |
| move constructor | not declared | defaulted | deleted | deleted | user declared | not declared |
| move assignment | defaulted | defaulted | deleted | deleted | not declared | user declared |

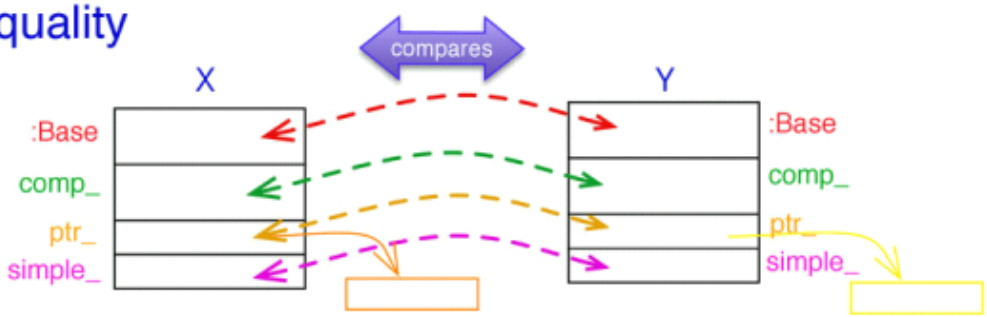
# Equality

A copied/assigned object should be equal ( `operator==` ) to the original.

## Deep equality




## Shallow equality




The compiler does NOT generate a default version of the equality operator -- we are on our own.

# Equality

```
bool operator== ( const MyClass &m, const MyClass &m2 )
{
  if ( !( Base::operator==(m, m2) ) ) return false;

  if ( m.simple_ != m2.simple_ ) return false;

  if ( ! (m.comp_ == m2.comp_) ) return false;

  if  ( !m.p_  && !m2.p_  ) return true;

  if  ( !m.p_ || !m2.p_ ) return false;

  return *m.p_ == *m2.p_ ;
}
```

# Take Aways

## Recognition

- C++ special member functions (6 of them):
  if one needs to be hand-crafted, likely all of them do

## Comprehension

- Best practices for ADT design
- Entity vs. Value-based ADTs
- Rules for compiler-default special member functions (default constructor, destructor, copy constructor, copy assignment)

## Application

- Operator overloading
- Const function arguments and member functions
- ADT design (entity vs. value-based design, immutable ADTs, hidden implementation)
- User-defined constructors, destructor, copy constructor, copy assignment