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 (0 parameters; generated iff we define no constructor)
- destructor
- copy constructor
- copy 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.

```cpp
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 default constructor or copy/move constructor or 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

Copy Assignment Operator

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

```cpp
X = Y; // deep copy
```
If we do not provide a copy/move constructor or 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

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.

```cpp
MyClass::MyClass (MyClass&& m) :
    Base{ std::move(m) },
    comp_{ std::move(m.comp_) },
    simple_{ m.simple },
    ptr_{ m.ptr_ } {
    m.ptr_ = nullptr;
}
```

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

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!!

```cpp
// friend that swaps contents of m1 and m2
// using std::swap from <algorithm>
void swap( MyClass & m1, MyClass &m2 ) {
    Base &b_m1 = static_cast<Base&>(m1);
    Base &b_m2 = static_cast<Base&>(m2);
    swap( b_m1, b_m2 );
    std::swap( m1.comp_, m2.comp_ );
    std::swap( m1.ptr_, m2.ptr_ );
    std::swap( m1.simple_, m2.simple_ );
}
```

```cpp
MyClass& MyClass::operator= (const MyClass& m) {
    MyClass temp{m};
    swap( *this, temp );
    return *this;
}
```
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.

```cpp
MyClass& MyClass::operator= (MyClass&& m) {
    MyClass temp(0);
    swap( temp, m ); // temp steals m’s contents
    swap( *this, temp ); // put what used to be in m into this
    return *this;
}
```

Improved Assignment

We can improve the code by passing the argument by value and unifying the copy assignment and move assignment operators into a single member method. Compiler will call this assignment operator even with an rvalue.

```cpp
MyClass::MyClass( const MyClass & other) : MyClass(0) {
    copy( *this, other );
}

MyClass::MyClass( MyClass && other ) : MyClass(0) {
    swap( *this, other );
}

MyClass& MyClass::operator= (MyClass m) {
    swap( *this, m );
    return *this;
}
```

Compiler-Generated Move Assignment

If we do not provide a copy/move constructor or copy/move assignment, the compiler will create a move assignment operator 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

Special Members

The compiler implicitly declares:

- Nothing
- Any constructor
- Copy constructor
- Copy assignment
- Move constructor
- Move assignment

User declares:

- Copy constructor
- Copy assignment
- Move constructor
- Move assignment

X = Y;  // shallow copy
### Equality

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

#### Deep equality

![Deep equality diagram]

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

#### Shallow equality

![Shallow equality diagram]

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

### 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