Contents

[Basic Object-oriented Programming 1](#_Toc141722663)

[Best practice 1](#_Toc141722664)

[value initialization 1](#_Toc141722665)

[List initialization 2](#_Toc141722666)

[A reminder about default parameters 2](#_Toc141722667)

[use = default. 2](#_Toc141722668)

[Member initializer lists 2](#_Toc141722669)

[RAII (Resource Acquisition Is Initialization) 2](#_Toc141722670)

[this pointer 2](#_Toc141722671)

[Key insight 3](#_Toc141722672)

[Static Function 3](#_Toc141722673)

[Conversion constructor and conversion operator 3](#_Toc141722674)

[Friend Function: 4](#_Toc141722675)

# Basic Object-oriented Programming

Class (and struct) definitions are like a blueprint -- they describe what the resulting object will look like, but they do not actually create the object. To actually create an object of the class, a variable of that class type must be defined:

## Best practice

Declare public members first, protected members next, and private members last. This spotlights the public interface and de-emphasizes implementation details.

## value initialization

Fraction f1 {}; // value initialization of Fraction f1

Fraction f2(); // forward declaration of function f2

## List initialization

Fraction fiveThirds{ 5, 3 }; // List initialization, calls Fraction(int, int)

Fraction threeQuarters(3, 4); // Direct initialization, also calls Fraction(int, int)

## A reminder about default parameters

The rules around defining and calling functions that have default parameters (described in lesson [8.13 -- Default arguments](https://www.learncpp.com/cpp-tutorial/default-arguments/)) apply to constructors too. To recap, when defining a function with default parameters, all default parameters must follow any non-default parameters, i.e. there cannot be non-defaulted parameters after a defaulted parameter.

## use = default.

If you have constructors in your class and need a default constructor that does nothing (e.g. because all your members are initialized using non-static member initialization), use = default.

## Member initializer lists

int value1 = 1; // copy initialization

double value2(2.2); // direct initialization

char value3 {'c'}; // uniform initialization

Use member initializer lists to initialize your class member variables instead of assignment.

If you have multiple constructors that have the same functionality, use delegating constructors to avoid duplicate code.

RAII (Resource Acquisition Is Initialization) is a programming technique whereby resource use is tied to the lifetime of objects with automatic duration (e.g. non-dynamically allocated objects). In C++, RAII is implemented via classes with constructors and destructors.

## this pointer

void setID(int id) { m\_id = id; }

is converted by the compiler into:

void setID(Simple\* const this, int id) { this->m\_id = id; }

When the compiler compiles a non-static member function, it implicitly adds a new parameter to the function named “this”. The **this pointer** is a hidden const pointer that holds the address of the object the member function was called on.

## Key insight

The **rule of five** says that if the copy constructor, copy assignment, move constructor, move assignment, or destructor are defined or deleted, then each of those functions should be defined or deleted.

Static Function

could be called on object as well as there is no harm in it.

Non-static function can be called within static function but there must be an object specification.

Static void Mystatic()

{

Base baseObj;

baseObj.simpleFunct(); // OK no error

simpleFunct() // there is an error here.

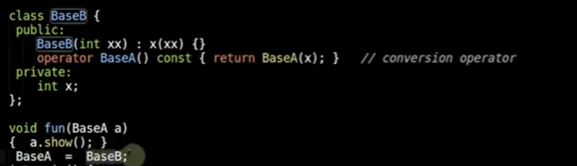
}

Base base;

Base. Mystatic() // OK no error

Conversion constructor and conversion operator

use precisely.



R value cannot be bound to reference.

Void myFunct(int& a)

{

}

myFunct(10); // Error

# Friend Function:

if you don’t have a getter and setter to class

then you can introduce the friend function for a class without messing the class

We Must Return Reference In Copy Assignment Operator

**Best practice**

# Use N as the name of an int non-type template parameter.