# How to master C++

Florian Richoux

March 13, 2014

UNIVERSITÉ DE NANTES

## Licence CC BY-NC-SA 4.0

## Why C++?

## Why C++?

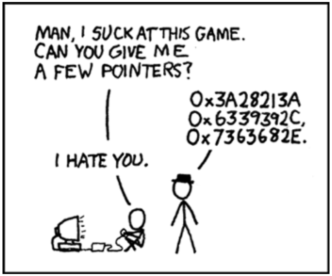
## Why C++?

CC BY 2.0

http://www.lextrait.com/vincent/implementations.html

## Outline

- Quick recalls about virtual
- Object copy
- Memory management
- Extra


MAN, I SUCK AT THIS GAME. CAN YOU GIVE ME A FEW POINTERS?
0x3A28213A 0x6339392C, 0x7363682E.
I HATE YOU.

Randall Munroe, CC BY-NC 2.0

http://xkcd.com/138/

## Some (virtual) recalls

## Some quick recalls

```
#include <iostream>
using namespace std;

struct A {
  void f() { cout << "Class␣A" << endl; }
};

struct B: A {
  void f() { cout << "Class␣B" << endl; }
};

int main() {
  A *a = new B;
  a->f();
  delete a; // ?
}
```

## Some quick recalls

```
#include <iostream>
using namespace std;

struct A {
  void f() { cout << "Class A" << endl; }
};

struct B: A {
  void f() { cout << "Class B" << endl; }
};

int main() {
  A *a = new B;
  a->f();
  delete a; // ?
}
```

**Output**

```
Class A
```

## Some quick recalls

```cpp
#include <iostream>
using namespace std;

struct A {
  virtual void f() { cout << "Class␣A" << endl; }
};

struct B: A {
  void f() { cout << "Class␣B" << endl; }
};

int main() {
  A *a = new B;
  a->f();
  delete a; // ?
}
```

## Some quick recalls

```
#include <iostream>
using namespace std;

struct A {
  virtual void f() { cout << "Class␣A" << endl; }
};

struct B: A {
  void f() { cout << "Class␣B" << endl; }
};

int main() {
  A *a = new B;
  a->f();
  delete a; // ?
}
```

**Output**

```
Class B
```

## Some quick recalls

```
class Base
{
  ...
};

class Derived : public Base
{
  ~Derived()
  {
    // Do some important cleanup
  }
}
```

```
Base *b = new Derived();
// use b
delete b; // Here's the problem: (usually) call ~Base()
```

http://stackoverflow.com/questions/461203/when-to-use-virtual-destructors

## Some quick recalls

```
class Base
{
  public: virtual ~Base() { }
};

class Derived : public Base
{
  ~Derived()
  {
    // Do some important cleanup
  }
}
```

```
Base *b = new Derived();
// use b
delete b; // call ~Derived()
```

http://stackoverflow.com/questions/461203/when-to-use-virtual-destructors

## Some quick recalls

```
struct A { virtual ~A() { } };
struct B : A { };

struct C { };
struct D : C { };

int main() {
  B b;
  A* ap = &b;
  A& ar = b;
  cout << "ap: " << typeid(*ap).name() << endl;
  cout << "ar: " << typeid(ar).name() << endl;

  D d;
  C* cp = &d;
  C& cr = d;
  cout << "cp: " << typeid(*cp).name() << endl;
  cout << "cr: " << typeid(cr).name() << endl;
}
```

## Some quick recalls

```
struct A { virtual ~A() { } };
struct B : A { };

struct C { };
struct D : C { };

int main() {
  B b;
  A* ap = &b;
  A& ar = b;
  cout << "ap: " << typeid(*ap).name() << endl;
  cout << "ar: " << typeid(ar).name() << endl;

  D d;
  C* cp = &d;
  C& cr = d;
  cout << "cp: " << typeid(*cp).name() << endl;
  cout << "cr: " << typeid(cr).name() << endl;
}
```

**Output**

```
ap: B
ar: B
cp: C
cr: C
```

## Some quick recalls

```
struct A { virtual ~A() { } };
struct B : A { };

struct C { };
struct D : C { };

int main() {
  B b;
  A* ap = &b;
  A& ar = b;
  cout << "ap: " << typeid(*ap).name() << endl;
  cout << "ar: " << typeid(ar).name() << endl;

  D d;
  C* cp = &d;
  C& cr = d;
  cout << "cp: " << typeid(*cp).name() << endl;
  cout << "cr: " << typeid(cr).name() << endl;
}
```

Output

```
ap: B
ar: B
cp: C
cr: C
```

http://publib.boulder.ibm.com/infocenter/comphelp/v8v101/index.jsp?topic=%2Fcom.ibm.xlcpp8a.doc%2Flanguage%2Fref%2Fthe_typeid_operator.htm

## Object copy

## Copy constructor and copy assignment operator

```cpp
class Person
{
  std::string name_;
  int age_;

  public:
  Person(std::string name, int age)
    : name_(name), age_(age) { }
};

int main()
{
  Person a("Bjarne␣Stroustrup", 63);
  Person b(a);    // What happens here?
  b = a;          // And here?
}
```

## Copy constructor and copy assignment operator

```
// 1. copy constructor
Person(const Person& that)
  : name_(that.name_), age_(that.age_) { }

// 2. copy assignment operator
Person& operator=(const Person& that)
{
  name_ = that.name_;
  age_ = that.age_;
  return *this;
}
```

**Signature**

```
           Classname( const Classname& ) // copy ctor
Classname& operator=( const Classname& ) // assignment op
```

## Copy constructor and copy assignment operator

```
int main()
{
  Person a("Bjarne␣Stroustrup", 63);
  Person b(a);     // Call the copy ctor
  b = a;           // Call the copy assignment operator
  Person c = a;    // ?
}
```

## Copy constructor and copy assignment operator

```
int main()
{
  Person a("Bjarne␣Stroustrup", 63);
  Person b(a);      // Call the copy ctor
  b = a;            // Call the copy assignment operator
  Person c = a;     // Call the copy ctor
                    // (almost equivalent to Person c(a))
}
```

**More about initializations**

http://herbsutter.com/2013/05/09/gotw-1-solution/

## In which situations is the C++ copy constructor called?

```
MyClass a;
MyClass b(a);                // copy constructor

////////////////

void foo(MyClass x);
foo(a);                      // copy constructor
                             // (but can be moved in C++11)
                             // A simple thing to avoid this?

////////////////

MyClass foo ()
{
  MyClass temp;
  ...
  return temp;               // copy constructor
                             // (but usually RVO applies)
}
```

http://stackoverflow.com/questions/21206359/in-which-situations-is-the-c-copy-constructor-called

# In which situations is the C++ copy constructor called?

```
MyClass a;          // constructor
MyClass b;          // constructor
a = b;              // copy assignment op
b = MyClass(a);     // copy ctor + copy assignment op
```

```
MyClass *a = new MyClass(); // constructor
MyClass *b;                 // nothing is called
b = a;                      // still nothing is called
b = new MyClass(*a);        // copy constructor
```

http://stackoverflow.com/questions/21206359/in-which-situations-is-the-c-copy-constructor-called

## Why writing a copy ctor and a copy assignment operator?

**Question**

Why do we need to (sometimes) write them?

## Why writing a copy ctor and a copy assignment operator?

**Question**

Why do we need to (sometimes) write them?

**Reformulated question**

**When** do we need to write them?

## Why writing a copy ctor and a copy assignment operator?

Question

Why do we need to (sometimes) write them?

Reformulated question

**When** do we need to write them?

Answer

Each time you have a class managing resources (like manipulating memory, pointers)!

## Why writing a copy ctor and a copy assignment operator?

```
class A {
  public:
    A() {i = new int;}
    int *i;
};

A a;
A b = a;
// same story with just b = a
std::cout << a.i << std::endl << b.i << std::endl;
```

## Why writing a copy ctor and a copy assignment operator?

```
class A {
  public:
    A() {i = new int;}
    int *i;
};

A a;
A b = a;
// same story with just b = a
std::cout << a.i << std::endl << b.i << std::endl;
```

**Output**

```
0x3A28213A
0x3A28213A
```

## Why writing a copy ctor and a copy assignment operator?

```
class A {
  public:
    A() {i = new int;}  // ctor

    A(const A& other) { // copy ctor
      i = new int;
      *i = *(other.i);
    }
    int *i;
};

A a;
A b = a;
// same story with just b = a
std::cout << a.i << std::endl << b.i << std::endl;
```

## Why writing a copy ctor and a copy assignment operator?

```
class A {
  public:
    A() {i = new int;}  // ctor

    A(const A& other) { // copy ctor
      i = new int;
      *i = *(other.i);
    }
    int *i;
};

A a;
A b = a;
// same story with just b = a
std::cout << a.i << std::endl << b.i << std::endl;
```

**Output**

```
0x3A28213A
0x6339392C
```

## The copy-and-swap idiom

I explained you:

- **What** copy ctor and copy assignment operator are.
- **When** they are called.
- **Why** it is important to (sometimes) write them.

But I did not explain yet **how** to implement them properly.

**Good implementation**

Apply the copy-and-swap idiom.

## The copy-and-swap idiom

```
class MyClass {
  public:
    MyClass(std::size_t size = 0) // ctor
      : size(size),
        array(size ? new int[size] : nullptr)
      {}

    MyClass(const MyClass& other) // copy ctor
      : size(other.size),
        array(size ? new int[size] : nullptr)
      { std::copy(other.array, other.array + size, array); }

  private:
    std::size_t size;
    int *array;
};
```

http://stackoverflow.com/questions/3279543/what-is-the-copy-and-swap-idiom/

# The copy-and-swap idiom

```
class MyClass {
  ...
  public:
    MyClass& operator=(const MyClass& other) // copy asgmt op
    {
      if (this != &other)
      {
        // put in the new data...
        std::size_t newSize = other.size;
        int *newArray = newSize ? new int[newSize] : nullptr;
        std::copy(other.array, other.array + size, newArray);


        // ...and get rid of the old data
        delete [] array;
        size = newSize;
        array = newArray;
      }

      return *this;
    }
};
```

## The copy-and-swap idiom

```
class MyClass {
  ...
  public:
    MyClass& operator=(const MyClass& other) // copy asgmt op
    {
      if (this != &other) // often useless
      {
        // put in the new data...
        std::size_t newSize = other.size;
        int *newArray = newSize ? new int[newSize] : nullptr;
        std::copy(other.array, other.array + size, newArray);
        // (these 3 lines are code duplication)

        // ...and get rid of the old data
        delete [] array;
        size = newSize;
        array = newArray;
      }

      return *this;
    }
};
```

## The copy-and-swap idiom

```
class MyClass {
  ...
  public:
    void swap(MyClass& other)
    {
      std::swap(this->size, other.size);
      std::swap(this->array, other.array);
    }

    MyClass& operator=(MyClass other) // no reference!
    {
      swap(other);
      return *this;
    }
};
```

http://stackoverflow.com/questions/3279543/what-is-the-copy-and-swap-idiom/

## Memory management

## The Rule of Three

### Rule of 3

If your class needs any of

- a destructor,
- or a copy constructor,
- or a copy assignment operator.

defined explicitly, then it is likely to need **all three of them**.

### Put in other words

If your class manages resources, you need to explicitly define:

- a destructor,
- a copy constructor,
- and a copy assignment operator.

## The Rule of Three

**These three are linked**

What do a copy assignment operator?

- It copies a new state (copy ctor),
- and it deletes the old state (destructor).

http://stackoverflow.com/questions/4172722/what-is-the-rule-of-three

## The Rule of Three

The rule “A delete for each new” is not sufficient!

```
class A
{
  public:
    A(int i) : array_(i ? new int[i] : nullptr) { }
    ~A() { delete[] array_; }
  private:
    int *array_;
};

A *a1 = new A(42);
A *a2 = new A(24);
...
(*a1) = (*a2);
...
delete a1;
delete a2;
```

## The Rule of Three

The rule "A delete for each new" is not sufficient!

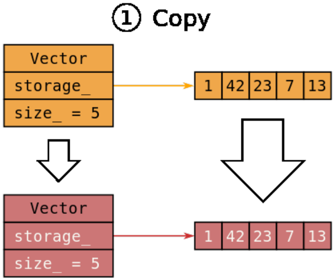
```
class A
{
  public:
    A(int i) : array_(i ? new int[i] : nullptr) { }
    ~A() { delete[] array_; }
  private:
    int *array_;
};

A *a1 = new A(42);
A *a2 = new A(24);
...
(*a1) = (*a2); // Memory leak!
...            // We have lost original a1's array_
delete a1;
delete a2; // Undefined behavior!
```

## C++11 and move semantics

```
class Vector {
  int *storage_;
  size_t  size_;

  public:

  // ctor
  Vector(size_t numElements)
    : storage_(new int[numElements]),
      size_(numElements)
  { }

  // dtor
  ~Vector() { delete[] storage_; }
};
```

http://kholdstare.github.io/technical/2013/11/23/moves-demystified.html

# C++11 and move semantics



http://kholdstare.github.io/technical/2013/11/23/moves-demystified.html

## C++11 and move semantics

```
Vector c = a + b;
```

```
??? operator+ (Vector const & a, Vector const & b);
```

http://kholdstare.github.io/technical/2013/11/23/moves-demystified.html

## C++11 and move semantics

```
Vector c = a + b;
```

```
??? operator+ (Vector const & a, Vector const & b);
```

- Return by value seems bad.

http://kholdstare.github.io/technical/2013/11/23/moves-demystified.html

## C++11 and move semantics

```cpp
Vector c = a + b;
```

```cpp
??? operator+ (Vector const & a, Vector const & b);
```

- Return by value seems bad.
- Return a pointer is bad too: you must make disallocation somewhere, and can't chained + operations (like a+b+c).

http://kholdstare.github.io/technical/2013/11/23/moves-demystified.html

## C++11 and move semantics

```
Vector c = a + b;
```

```
??? operator+ (Vector const & a, Vector const & b);
```

- Return by value seems bad.
- Return a pointer is bad too: you must make disallocation somewhere, and can't chained + operations (like a+b+c).
- Return a reference seems not a good idea either.

http://kholdstare.github.io/technical/2013/11/23/moves-demystified.html

## C++11 and move semantics

You need to

**move!**

## C++11 and move semantics

### Is returning by value really bad?

```
Vector operator+ (Vector const& a, Vector const& b)
{
  // create result of same size
  assert(a.size() == b.size());
  Vector result(a.size());

  // compute addition
  std::transform(
    a.begin(), a.end(),    // input 1
    b.begin(),             // input 2
    result.begin(),        // result
    std::plus<int>()    // binary operation
  );

  return result; // RVO usually applies
}
```

http://kholdstare.github.io/technical/2013/11/23/moves-demystified.html

## C++11 and move semantics

Yes, but...

Reason #1

```
std::string f(bool cond = false) {
  std::string first("first");
  std::string second("second");

  return cond ? first : second; // return under condition:
                                // usually no RVO
}
```

Reason #2

RVO applies when one transfers a value **out of a scope**.
What if we need to transfer **into a scope**?

http://kholdstare.github.io/technical/2013/11/23/moves-demystified.html
http://en.wikipedia.org/wiki/Return_value_optimization

## C++11 and move semantics

### Transferring value into a scope

```
Ray computeRay()
{
  Vector origin;
  Vector direction;

  ...

  return Ray(
    origin, // COPY!
    direction // COPY!
  ); // certainly RVO
}
```

http://kholdstare.github.io/technical/2013/11/23/moves-demystified.html

## lvalue Vs rvalue

**lvalue**

c = a + b;

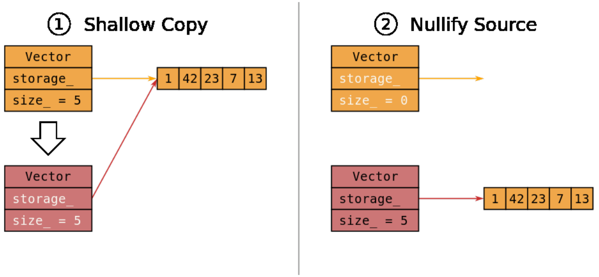
**rvalue**

c = a + b;
Must be a temporary, non-named value.

http://stackoverflow.com/questions/3601602/what-are-rvalues-lvalues-xvalues-glvalues-and-prvalues

## C++11 and move semantics

```
Vector::Vector(Vector&& other)
// shallow copy
  : storage_(other.storage_),
    size_(other.size_)
{
  // nullify source
  other.storage_ = nullptr;
  other.size_ = 0;
}
```



http://kholdstare.github.io/technical/2013/11/23/moves-demystified.html

## C++11 and move semantics

### Transferring value into a scope

```
Ray computeRay()
{
  Vector origin;
  Vector direction;

  ...

  return Ray(
    std::move(origin),     // moved!
    std::move(direction)   // moved!
  ); // certainly RVO
}
```

http://kholdstare.github.io/technical/2013/11/23/moves-demystified.html

## The Rule of Four and a Half

### When a class manipulates resources

Rule of 4.5

=

Rule of 3

+

define the move ctor
(+ define a move assignment operator?)

http://stackoverflow.com/questions/4782757/rule-of-three-becomes-rule-of-five-with-c11

http://stackoverflow.com/questions/3279543/what-is-the-copy-and-swap-idiom

## RAII and Smart pointers

RAII

**R**esource **A**cquisition **I**s **I**nitialization: release resource automatically.

Some applications

- Files,
- Network sockets,
- Mutex,
- **Memory**.

**Smart pointers** *std::unique_ptr* and *std::shared_ptr*.

## Rule of Zero

**Rule of 0**

Using smart pointers (and RAII principle) to manage resources, **no need** to explicitly declare dtor, copy ctor, etc.

http://flamingdangerzone.com/cxx11/2012/08/15/rule-of-zero.html

## Rule of Zero

**Rule of 0**

Using smart pointers (and RAII principle) to manage resources, **no need** to explicitly declare dtor, copy ctor, etc.

You can

rest!

http://flamingdangerzone.com/cxx11/2012/08/15/rule-of-zero.html

## Extra

## I did not talk about

- const (http://duramecho.com/ComputerInformation/WhyHowCppConst.html)
- Exceptions (and C++11 noexecpt)
- operator+= and operator+ (and operator++ and stuff)
- C++ cast
- Functors
- C++11 features like:
  - auto,
  - lambda,
  - decltype
- C++14
- C++17

## Safety

### Asserts

Use **assert**. Unable them with the -DNDEBUG compile option.

### Valgrind

valgrind –leak-check=full –show-reachable=yes ./your_program

### Warnings

Try to solve them!

## Read!

### Books

- **Efficient C++** by Scott Meyers (C++11/14 update soon!)
- **Exceptionnal C++** by Herb Sutter (C++11/14 update soon!)

### Blog

Herb Sutter's "Guru of the Week"
`http://herbsutter.com/category/c/gotw/`

### Twitter

@isocpp
@cppstack

## Use!

**Boost library**

http://www.boost.org/

**<algorithm>**

Gotta use 'em all!
http://www.cplusplus.com/reference/algorithm/

## Fonctional C++

**John Carmack's blog**

http://www.altdevblogaday.com/2012/04/26/functional-programming-in-c/

**Modern Functional Programming in C++**

http://zao.se/~zao/boostcon/10/2010_presentations/thu/funccpp.pdf

**C++17: I See a Monad in Your Future!**

http://bartoszmilewski.com/2014/02/26/c17-i-see-a-monad-in-your-future/

## Template Metaprogramming

### Books

- **Modern C++ Design** by Andrei Alexandrescu.
- **C++ Template Metaprogramming** by Dave Abrahams and Aleksey Gurtovoy.
- **C++ Templates: The Complete Guide** by David Vandevoorde and Nicolai Josuttis (second edition planned for 2015).

### A nice intro

`http://www.codeproject.com/Articles/3743/A-gentle-introduction-to-Template-Metaprogramming`

## SOLID

- **S**ingle Responsibility: One reason to exist, one reason to change
- **O**pen Closed Principle: Open for extension, closed for modification
- **L**iskov Substitution Principle: An object should be semantically replaceable for it's base class/interface
- **I**nterface Segregation Principle: Don't force a client to depend on an interface it doesn't need to know about
- **D**ependency Inversion Principle: Depend on abstractions, not concrete detail or implementations

http://stackoverflow.com/questions/1423597/solid-principles

http://en.wikipedia.org/wiki/SOLID_%28object-oriented_design%29

## Repository and comments

svn, git, mercurial, ...

**Ultimate combo**

GitHub + Travis
(http://docs.travis-ci.com/user/getting-started/)

**Comments**

Comment your code with doxygen

## Repository and comments

svn, git, mercurial, ...

**Ultimate combo**

GitHub + Travis
(`http://docs.travis-ci.com/user/getting-started/`)

**Comments**

Comment your code with doxygen **in English!**

## To make progress

Code!

## To make progress

Code!

Teach!

## C++ hiring questions 1/3

- How many ways are there to initialize a primitive data type in C++ and what are they?
- Why should you declare a destructor as virtual?
- What does it mean that C++ supports overloading?
- What are examples of overloading in C++?
- What is name mangling in C++ and why is it used?
- What is an abstract base class?
- What is RTTI?
- How can you access a variable that is "hidden" by another variable of the same name?
- What is a namespace and how is it used.
- What are the differences between a class and a struct in C++, and how does this compare to C?
- What are templates? How are they used?
- What is a copy constructor and when is it used, especially in comparison to the equal operator.
- What is the difference between a "shallow" and a "deep" copy?
- What is the const operator and how is it used?

## C++ hiring questions 2/3

- What are the differences between passing by reference, passing by value, and passing by pointer in C++?
- When is it and when is it not a good idea to return a value by reference in C++?
- What is the difference between a variable created on the stack and one created on the heap?
- How do you free memory allocated dynamically for an array? What are the implications of just using delete?
- What is multiple inheritance? When should it be used?
- What is a pure virtual function?
- What does the keyword mutable do?
- What does the keyword volatile do?
- What is the STL?
- What is a Vector?
- What is contained in the <algorithms> header?

## C++ hiring questions 3/3

- What is the difference between #include <iostream.h> and #include <iostream>?
- What's the difference between "++i" and "i++"?
- What is short circuit evaluation? How can it be used? Why can is be dangerous?
- What is the ',' operator?
- What is the only ternary operator? How is it used?
- What is the use of a const member function and how can it be used?
- How is try/catch used in C++?
- Why should you never throw an exception in a destructor?
- What is the explicit keyword?
- What is the proper way to perform a cast in C++?
- What does inline do?

## Thanks!