

# AGSO C++ Intro

# Christopher Rabotin

christopher.rabotin@ { gmail.com  
colorado.edu

github.com/ChristopherRabotin<sup>1</sup>  
linkedin.com/in/chrisrabotin

November 16, 2016

<sup>1</sup>[github.com/ChristopherRabotin/asgo-cpp/](https://github.com/ChristopherRabotin/asgo-cpp/)

# Outline

C++ Intro

Chris Rabotin

Concepts

OOP

C → C++

Adv. OOP

Useful libraries

STL

BLAS

Qt

El futuro!

Programming is about practice, practice and more practice.

## Concepts

Object oriented programming

Transition from C to C++

Advanced OOP

## Useful libraries

The Standard Template Library

The Basic Linear Algebra Subprograms

Qt

## Where fun and future meet

# Outline

C++ Intro

Chris Rabotin

Concepts

OOP

C  $\rightarrow$  C++

Adv. OOP

Useful libraries

STL

BLAS

Qt

El futuro!

## Concepts

Object oriented programming

Transition from C to C++

Advanced OOP

## Useful librairies

The Standard Template Library

The Basic Linear Algebra Subprograms

Qt

Where fun and future meet

# Object Oriented Programming?

Recommended reading:

[https://en.wikipedia.org/wiki/Object-oriented\\_programming](https://en.wikipedia.org/wiki/Object-oriented_programming)

- ▶ Very common paradigm of programming which uses special data structures called objects
- ▶ Enables abstraction and separation of concern
- ▶ Instances share properties (attributes and methods)
  - ▶ Attributes define the specificity of an instance
  - ▶ Methods define how all objects of a given type (generally) behave

## Example

Let Alice and Bob be two humans. We can respectively set the name attribute to ``Alice`` and ``Bob``. However, both instances behave identically for the `walk()` method.

# Object Oriented Programming?

Recommended reading:

[https://en.wikipedia.org/wiki/Object-oriented\\_programming](https://en.wikipedia.org/wiki/Object-oriented_programming)

- ▶ Very common paradigm of programming which uses special data structures called objects
- ▶ Enables abstraction and separation of concern
- ▶ Instances share properties (attributes and methods)
  - ▶ Attributes define the specificity of an instance
  - ▶ Methods define how all objects of a given type (generally) behave

## Example

Let Alice and Bob be two humans. We can respectively set the name attribute to ``Alice`` and ``Bob``. However, both instances behave identically for the `walk()` method.

# Outline

C++ Intro

Chris Rabotin

Concepts

OOP

C  $\rightarrow$  C++

Adv. OOP

Useful libraries

STL

BLAS

Qt

El futuro!

## Concepts

Object oriented programming

Transition from C to C++

Advanced OOP

## Useful librairies

The Standard Template Library

The Basic Linear Algebra Subprograms

Qt

## Where fun and future meet

# In practice, what's an object?

"Well... THAT escalated quickly." - Ron Burgundy

We'll define a Person object which can walk() a provided distance.

- ▶ In C,
  - ▶ Use structs to bundle attributes
  - ▶ But methods are independently defined, cf. **Struct example**<sup>2</sup>
- ▶ In C++, we'll use an object:
  - ▶ **OOP example 1**<sup>3</sup>
  - ▶ **OOP example 2**<sup>4</sup>
- ▶ Constructor allows to set the attributes of an object, including default values
- ▶ Properties defined either as private or public<sup>5</sup>

---

<sup>2</sup>Notice the walk() function requires a pointer to a struct.

<sup>3</sup>Any issues here?

<sup>4</sup>Identify attributes, methods, constructor, initialization and scope.

<sup>5</sup>Or protected. More on that later.

Concepts

OOP

C → C++

Adv. OOP

Useful libraries

STL

BLAS

Qt

El futuro!

# In practice, what's an object?

"Well... THAT escalated quickly." - Ron Burgundy

We'll define a Person object which can walk( ) a provided distance.

- ▶ In C,
  - ▶ Use structs to bundle attributes
  - ▶ But methods are independently defined, cf. **Struct example**<sup>2</sup>
- ▶ In C++, we'll use an object:
  - ▶ **OOP example 1**<sup>3</sup>
  - ▶ **OOP example 2**<sup>4</sup>
- ▶ Constructor allows to set the attributes of an object, including default values
- ▶ Properties defined either as private or public<sup>5</sup>

---

<sup>2</sup>Notice the walk( ) function requires a pointer to a struct.

<sup>3</sup>Any issues here?

<sup>4</sup>Identify attributes, methods, constructor, initialization and scope.

<sup>5</sup>Or protected. More on that later.

Concepts

OOP

C → C++

Adv. OOP

Useful libraries

STL

BLAS

Qt

El futuro!



# Take home problem 1

What did I say about practice?

- ▶ Create a 3D vector object<sup>6</sup>.
- ▶ Attributes:  $x$ ,  $y$ ,  $z$  as double
- ▶ Methods:
  - ▶ Equals: returns whether two vectors are equal
  - ▶ PrettyPrint: prints to the console as  $[1, -5, 6.1]$
- ▶ Implementation example<sup>7</sup>

---

<sup>6</sup>Don't use `std::vector`.

<sup>7</sup>Try it yourself before the link.

# Namespaces

Thank you Mario! But our princess is in another castle!

- ▶ Add some organization to your code
- ▶ Easily avoid naming conflicts
- ▶ Enable header class definitions
- ▶ Compare for yourself: [ns2.cpp diff 1](#)
- ▶ Avoid: `using namespace blah;`<sup>8</sup>
- ▶ Instead, specify the function(s) to extract: [ns2.cpp diff 2](#)
- ▶ A class defines a new namespace
- ▶ Allows for forward declaration, i.e. deferring function implementation
  - ▶ [New ns3.hpp header file](#)
  - ▶ [ns3.cpp diff](#)

More info [here](#).

---

<sup>8</sup>Unless you can guarantee no conflict at all.

# Namespaces

Thank you Mario! But our princess is in another castle!

- ▶ Add some organization to your code
- ▶ Easily avoid naming conflicts
- ▶ Enable header class definitions
- ▶ Compare for yourself: [ns2.cpp diff 1](#)
- ▶ Avoid: `using namespace blah;`<sup>8</sup>
- ▶ Instead, specify the function(s) to extract: [ns2.cpp diff 2](#)
- ▶ A class defines a new namespace
- ▶ Allows for forward declaration, i.e. deferring function implementation
  - ▶ [New ns3.hpp header file](#)
  - ▶ [ns3.cpp diff](#)

More info [here](#).

---

<sup>8</sup>Unless you can guarantee no conflict at all.

# Namespaces

Thank you Mario! But our princess is in another castle!

- ▶ Add some organization to your code
- ▶ Easily avoid naming conflicts
- ▶ Enable header class definitions
- ▶ Compare for yourself: [ns2.cpp diff 1](#)
- ▶ Avoid: `using namespace blah;`<sup>8</sup>
- ▶ Instead, specify the function(s) to extract: [ns2.cpp diff 2](#)
- ▶ A class defines a new namespace
- ▶ Allows for forward declaration, i.e. deferring function implementation
  - ▶ [New ns3.hpp header file](#)
  - ▶ [ns3.cpp diff](#)

More info [here](#).

---

<sup>8</sup>Unless you can guarantee no conflict at all.

# Class instantiation

Don't be like your annoying roommate: clean up your mess.

## ▶ Two kinds of memory:

### ▶ Stack:

- ▶ Local variables
- ▶ Cleared upon return
- ▶ May overflow (hence "stack overflow")

```
Person Alice("AliceStack");
```

### ▶ Heap

- ▶ Everything else
- ▶ Much bigger than the stack
- ▶ You are responsible for the mess<sup>9</sup>

```
Person *Alice = new Person("AliceHeap");
```

## ▶ Examples:

- ▶ What should we expect here: `init1.cpp`
- ▶ How about now? `init1 diff`

<sup>9</sup>cf. [https://en.wikipedia.org/wiki/Memory\\_leak](https://en.wikipedia.org/wiki/Memory_leak)

# Class instantiation

Don't be like your annoying roommate: clean up your mess.

- ▶ Two kinds of memory:

- ▶ Stack:

- ▶ Local variables
    - ▶ Cleared upon return
    - ▶ May overflow (hence "stack overflow")

```
Person Alice("AliceStack");
```

- ▶ Heap

- ▶ Everything else
    - ▶ Much bigger than the stack
    - ▶ You are responsible for the mess<sup>9</sup>

```
Person *Alice = new Person("AliceHeap");
```

- ▶ Examples:

- ▶ What should we expect here: `init1.cpp`
  - ▶ How about now? `init1 diff`

Concepts

OOP

C → C++

Adv. OOP

Useful libraries

STL

BLAS

Qt

El futuro!

<sup>9</sup>cf. [https://en.wikipedia.org/wiki/Memory\\_leak](https://en.wikipedia.org/wiki/Memory_leak)

# Take home problem 2

I hope you kept your code from problem 1

- ▶ Split up your previous class in a header file and an implementation file
- ▶ Create two vectors, one must be defined on the heap and the other on the stack
- ▶ Initialize one of them in a function, and the other in the `main`
- ▶ Print out whether they are equal or not in the `main`
- ▶ Implementation examples: `pb2.hpp` `pb2.cpp`

# Outline

C++ Intro

Chris Rabotin

Concepts

OOP

C  $\rightarrow$  C++

Adv. OOP

Useful libraries

STL

BLAS

Qt

El futuro!

## Concepts

Object oriented programming

Transition from C to C++

Advanced OOP

## Useful librairies

The Standard Template Library

The Basic Linear Algebra Subprograms

Qt

Where fun and future meet



# Class inheritance

Mommy?

- ▶ Object attributes are a has-a relationship
  - ▶ e.g. a Person has-a name and walking\_distance
- ▶ Class inheritance:
  - ▶ Define a is-a relation.
  - ▶ Somewhat “classes of objects”:
    - ▶ Share common attributes and methods between objects
    - ▶ Each object instance shares their class’ properties and the parent’s properties
  - ▶ Examples:
    - ▶ a Person is-a great ape
    - ▶ a Gorilla is-a great ape
- ▶ inherit1.cpp
- ▶ inherit2.cpp
- ▶ Things to remember:
  - ▶ Constructors and destructors must be redefined
  - ▶ The virtual keyword allows a function to be redefined
  - ▶ The protected keyword allows a property to be used solely by the subclasses

Concepts

OOP

C → C++

Adv. OOP

Useful libraries

STL

BLAS

Qt

El futuro!

# Class inheritance

Mommy?

- ▶ Object attributes are a has-a relationship
  - ▶ e.g. a Person has-a name and walking\_distance
- ▶ Class inheritance:
  - ▶ Define a is-a relation.
  - ▶ Somewhat “classes of objects”:
    - ▶ Share common attributes and methods between objects
    - ▶ Each object instance shares their class’ properties and the parent’s properties
  - ▶ Examples:
    - ▶ a Person is-a great ape
    - ▶ a Gorilla is-a great ape
- ▶ inherit1.cpp
- ▶ inherit2.cpp
- ▶ Things to remember:
  - ▶ Constructors and destructors must be redefined
  - ▶ The virtual keyword allows a function to be redefined
  - ▶ The protected keyword allows a property to be used solely by the subclasses

# Class inheritance

Mommy?

- ▶ Object attributes are a has-a relationship
  - ▶ e.g. a Person has-a name and walking\_distance
- ▶ Class inheritance:
  - ▶ Define a is-a relation.
  - ▶ Somewhat “classes of objects”:
    - ▶ Share common attributes and methods between objects
    - ▶ Each object instance shares their class' properties and the parent's properties
  - ▶ Examples:
    - ▶ a Person is-a great ape
    - ▶ a Gorilla is-a great ape
- ▶ inherit1.cpp
- ▶ inherit2.cpp
- ▶ Things to remember:
  - ▶ Constructors and destructors must be redefined
  - ▶ The virtual keyword allows a function to be redefined
  - ▶ The protected keyword allows a property to be used solely by the subclasses

# Operator overloading

Remember how you can define operators in algebraic structures in abstract algebra? Same, same, only really useful.<sup>10</sup>

- ▶ Enables one to use the basic C++ operators for any object
  - ▶ `+` `-` `*` `/` `>>` `<<` etc.
- ▶ Example: [advoop.hpp](#) [advoop.cpp](#)
- ▶ More information: [StackOverflow](#) and [Wikibooks](#)

---

<sup>10</sup>No offense to [ring](#) lovers out there.

# Templating

“Started from the bottom now we’re here” - Drake.

**This is a complicated topic. Don't expect to understand this immediately.**

- ▶ Defines genericity between objects
- ▶ Extract common behavior of unrelated objects
- ▶ Very powerful design pattern when mastered
- ▶ Examples:
  - ▶ Apply a given operation to many different classes:
    - ▶ Example 1
    - ▶ Example 2
  - ▶ Re-arranging items of an array in a specific way
- ▶ More info on Wikiversity [here](#) and [here](#)

# Templating

"Started from the bottom now we're here" - Drake.

**This is a complicated topic. Don't expect to understand this immediately.**

- ▶ Defines genericity between objects
- ▶ Extract common behavior of unrelated objects
- ▶ Very powerful design pattern when mastered
- ▶ Examples:
  - ▶ Apply a given operation to many different classes:
    - ▶ Example 1
    - ▶ Example 2
  - ▶ Re-arranging items of an array in a specific way
- ▶ More info on Wikiversity [here](#) and [here](#)

# Templating

"Started from the bottom now we're here" - Drake.

**This is a complicated topic. Don't expect to understand this immediately.**

- ▶ Defines genericity between objects
- ▶ Extract common behavior of unrelated objects
- ▶ Very powerful design pattern when mastered
- ▶ Examples:
  - ▶ Apply a given operation to many different classes:
    - ▶ [Example 1](#)
    - ▶ [Example 2](#)
  - ▶ Re-arranging items of an array in a specific way
- ▶ More info on Wikiversity [here](#) and [here](#)

# Outline

C++ Intro

Chris Rabotin

Concepts

OOP

C  $\rightarrow$  C++

Adv. OOP

Useful libraries

STL

BLAS

Qt

El futuro!

## Concepts

Object oriented programming

Transition from C to C++

Advanced OOP

## Useful librairies

The Standard Template Library

The Basic Linear Algebra Subprograms

Qt

Where fun and future meet



# The Standard Template Library

STL is awesome and you should use it.

- ▶ Collection of templated data structures<sup>11</sup>
- ▶ In aerospace, you'll probably mostly use `std::vector`
  - ▶ [stl.cpp](#)
- ▶ Several versions exist, and some are no longer maintained, so check which one you have
- ▶ More information: [Wikiversity](#) and [this STL tutorial](#)

---

<sup>11</sup>My torture of you guys was justified, [right?](#)

# Take home problem 3

Let the fun begin!

- ▶ Use STL's vector to create a 3D vector of double
- ▶ Overload the == operator to return whether two vectors are identical
- ▶ Implementation example: [pb3.cpp](#)

# Outline

C++ Intro

Chris Rabotin

Concepts

OOP

C  $\rightarrow$  C++

Adv. OOP

Useful libraries

STL

BLAS

Qt

El futuro!

## Concepts

Object oriented programming

Transition from C to C++

Advanced OOP

## Useful librairies

The Standard Template Library

The Basic Linear Algebra Subprograms

Qt

Where fun and future meet

# BLAS intro of intro

A real drag to use, but well worth it

- ▶ BLAS is a very high performance linear algebra library<sup>12</sup>
- ▶ Isn't distributed by default, i.e. needs installation<sup>13</sup>
- ▶ Requires the `-lblas` flag when compiling<sup>14</sup>
- ▶ Documentation seems spares at best, tutorials barely existant
- ▶ BLAS is split in three levels:
  1. Vector operations `blas1.cpp`
  2. Matrix to vector operations `blas2.cpp`
  3. Matrix to matrix operations
- ▶ BLAS does not solve linear algebra problems. For this, use LAPACK.
- ▶ **Official documentation**

<sup>12</sup>AFAIK, a large chunk is written in assembly, another in C and Fortran.

<sup>13</sup>Search for `blas-devel` in your package manager.

<sup>14</sup>e.g. `g++ blas1.cpp -lblas`

# Outline

C++ Intro

Chris Rabotin

Concepts

OOP

C  $\rightarrow$  C++

Adv. OOP

Useful libraries

STL

BLAS

Qt

El futuro!

## Concepts

Object oriented programming

Transition from C to C++

Advanced OOP

## Useful librairies

The Standard Template Library

The Basic Linear Algebra Subprograms

Qt

Where fun and future meet

# Qt

Qt is actually a lot of fun.

- ▶ Cross platform
  - ▶ Templates and various helpers
  - ▶ Beautiful and easy to build user interfaces
  - ▶ Works on (some) embedded devices
  - ▶ **Official website**
- ▶ Full disclosure: I have not used Qt5, only Qt4. Back in 2013, the official documentation had examples that would not compile<sup>15</sup>.

# Qt

Qt is actually a lot of fun.

- ▶ Cross platform
  - ▶ Templates and various helpers
  - ▶ Beautiful and easy to build user interfaces
  - ▶ Works on (some) embedded devices
  - ▶ **Official website**
- 
- ▶ Full disclosure: I have not used Qt5, only Qt4. Back in 2013, the official documentation had examples that would not compile<sup>15</sup>.

---

<sup>15</sup>cf. [this stack overflow question](#)

# Not a C++ fan?

You're not alone!

- ▶ C++ has many of disadvantages:
  - ▶ No pointer safety
  - ▶ Slow to code
  - ▶ Shared library linking (i.e. not easily portable)
  - ▶ Cryptic compiling errors
  - ▶ Debugging is terribly annoying
  - ▶ Sickening verbosity
  - ▶ Multicore capabilities are annoying
- ▶ But it's still widely used because of...
  - ▶ Heritage systems
  - ▶ Optimization possibilities in C and assembly<sup>16</sup>
  - ▶ Managers who are tired of playing catch-up with the ever evolving world of software engineering



# Not a C++ fan?

You're not alone!

- ▶ C++ has many of disadvantages:
  - ▶ No pointer safety
  - ▶ Slow to code
  - ▶ Shared library linking (i.e. not easily portable)
  - ▶ Cryptic compiling errors
  - ▶ Debugging is terribly annoying
  - ▶ Sickening verbosity
  - ▶ Multicore capabilities are annoying
- ▶ But it's still widely used because of...
  - ▶ Heritage systems
  - ▶ Optimization possibilities in C and assembly<sup>16</sup>
  - ▶ Managers who are tired of playing catch-up with the ever evolving world of software engineering

# Python

Bringing awesomeness to people since 1991

## ► Advantages

- Plenty of libraries (cf. **PyPI**)
- Trivial to learn
- As readable as novel<sup>17</sup>

## ► Disadvantages

- Interpreted language (i.e. slow to execute and mostly runtime errors)
- Multicore is a drag as well (cf. **gevent**)
- Two different on-going releases: Python 2.x and 3.x

## ► Official website

---

<sup>17</sup> Usually. (code by Yours truly)

# Go (golang)

Fun, and with a huge community

- ▶ By Google
- ▶ Advantages
  - ▶ Compiled
  - ▶ Simple syntax
  - ▶ Optimized for concurrency
  - ▶ Garbage collected (no need to free memory)
  - ▶ No useless characters (bye-bye semi-colons!)
  - ▶ Cross platform and statically linked
- ▶ Disadvantages
  - ▶ Quite new language, i.e. a lot of boilerplate code
  - ▶ Garbage collected (the world freezes momentarily)
  - ▶ Workspace setup is a drag
- ▶ Official website
- ▶ Learn online: [Go By Example](#)

# Rust

Quite possibly a C++ killer

- ▶ Led by Mozilla
- ▶ Advantages
  - ▶ Ridiculous speeds
  - ▶ Outstanding memory management and safety (variables are immutable by default)
  - ▶ Optimized for concurrency
  - ▶ Runs on bare ARM processors (i.e. can be used for hard real time applications)<sup>18</sup>
- ▶ Disadvantages
  - ▶ Too recent to be stable
  - ▶ Some functional programming paradigms are confusing
- ▶ Official website
- ▶ Learn online: [Rust By Example](#)

<sup>18</sup>via Zinc (experimental)