C++ course: 1. Introduction

2020 by Oleksiy Grechnyev

Why C++?

Often one can hear a question:
Why study outdated C++ when there are so many **REAL** languages:
Java, Javascript, Python ... ?

Why C++?

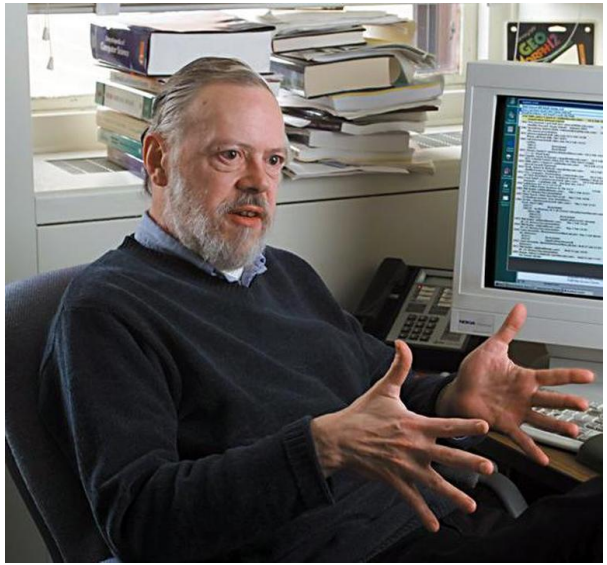
Often one can hear a question:
Why study outdated C++ when there are so many **REAL** languages:
Java, Javascript, Python ... ?

What language is ... written in?

- Operating Systems: Windows, Linux, MacOS, Android, iOS, ...
- Drivers, Game Engines, Blender, Maya ...
- Language Runtimes: Java, Python, Javascript (Google V8)
- Numerical libraries: OpenCV, numpy/scipy, tensorflow, pytorch, ...
- Matlab core routines
- Audio/Video codecs, signal processing

In e.g. Python : "Don't use **for** loops, don't use lists, use numpy/opencv/tensorflow/..."
C++ : No big difference

History: C and C++



Dennis Ritchie, the man behind C (1972) and Unix

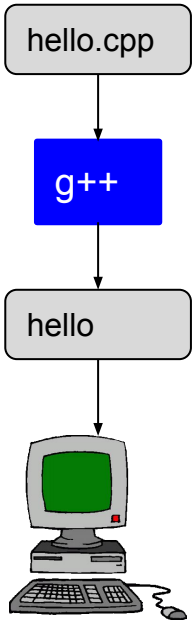
C is an *imperative* (procedural) language

Bjarne Stroustrup, the man who created C++ (1983)

C++ is an *object-oriented* language

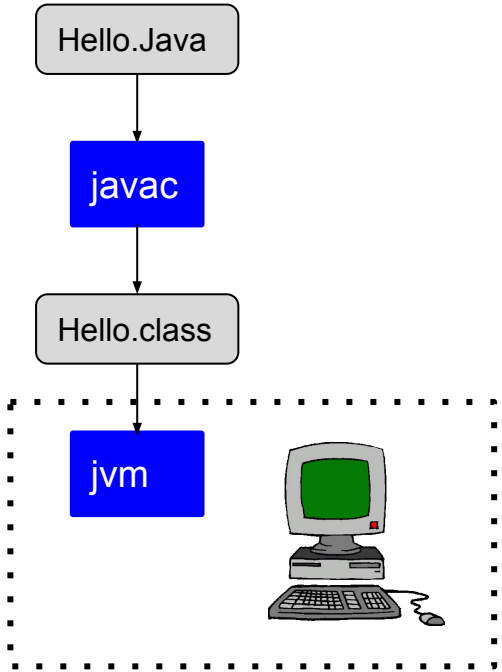
Compiler vs Interpreter

Compile to machine code



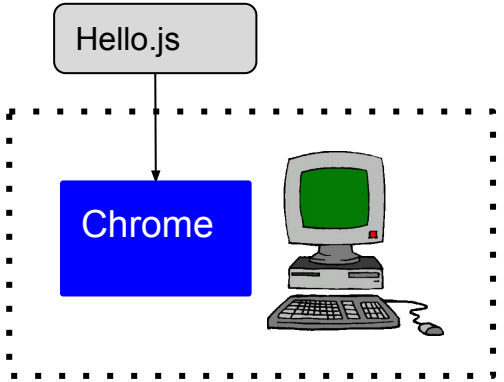
C, C++, Go, Fortran, Assembler, Pascal, Ada ...

Compile to bytecode



Java, Scala, Python, C#, ...

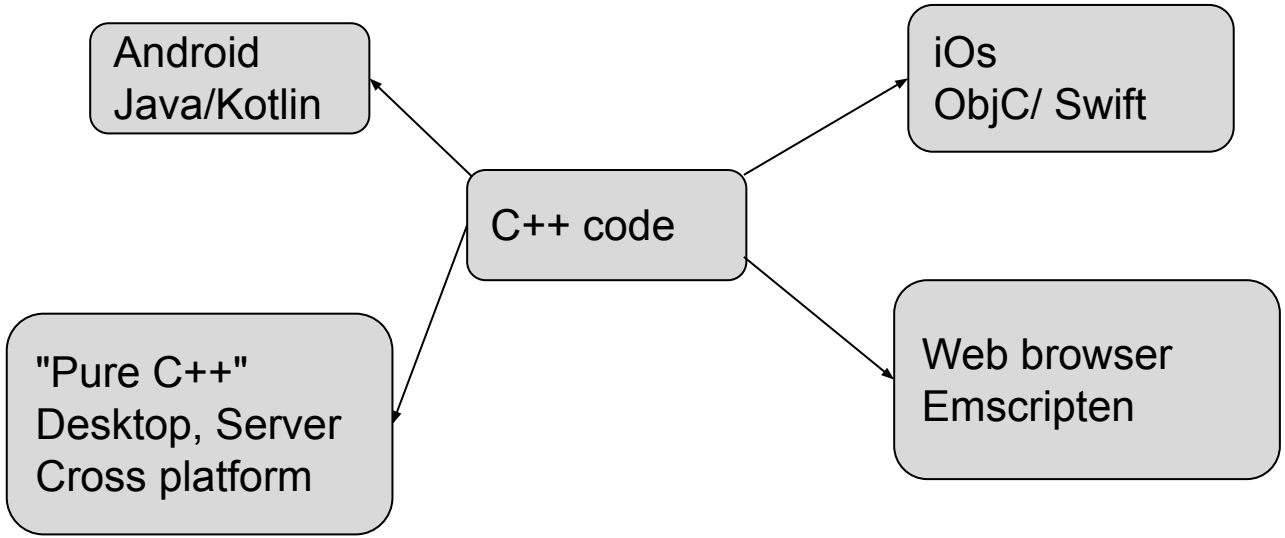
Interpret or Just In Time (JIT) compile



JavaScript, Ruby, Matlab, PHP, Basic, Logo ...

Why C++ ?

- Compiles to binary! Creates executable files and libraries (.a, .so/.dll)
- Can use directly any C libraries and OS API
- Fast, yet high level (compared to C)
- Portable
- Usable everywhere
- Easily combined with other languages (Java, Python, JS, ...)



C++ compilers

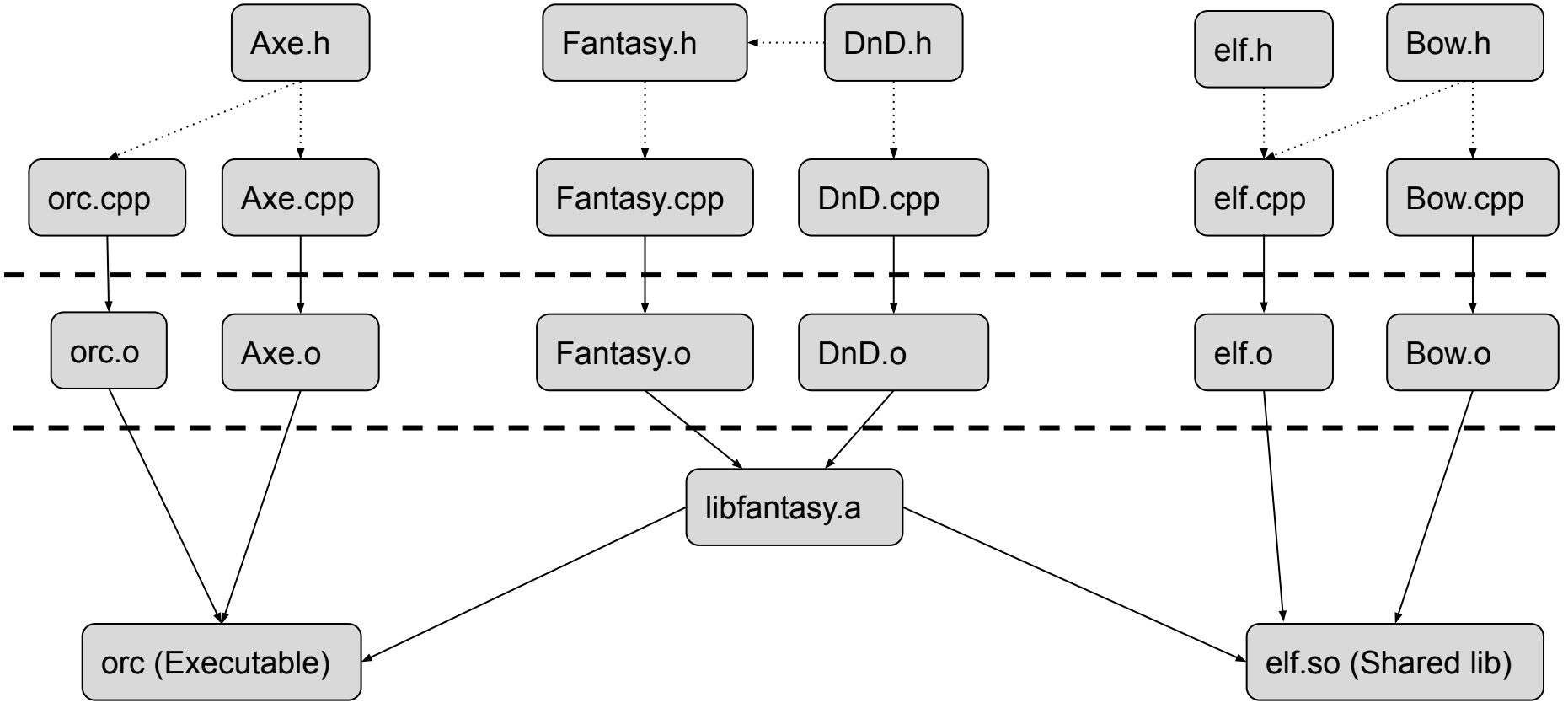
- **gcc** Linux, Windows (MinGW, MSYS, Cygwin)
- **clang/llvm** MacOS, iOS, Android, Linux, Windows
- **Microsoft CL** Windows only
- All other compilers are basically dead ...

How to choose your compiler:

- Does it have C++ 17 support?
- How many gigabytes of junk will it install ?
- What package manager are you going to use?

What to install on Windows?
I recommend MinGW for msys2 (Package manager + Lots of packages !)
http://www.msys2.org/
https://stackoverflow.com/questions/30069830/how-to-install-mingw-w64-and-msys2

Build process : The code is built from *.cpp and *.h (header) files



Build automation systems and IDEs

Build automation systems (Build tools)

- **CMake** ! WE USE THIS !
- make, nmake, ninja (low-level)
- qmake, Bazel
- Some horrible stuff from Google !

IDEs

- CLion (Commercial only, slow)
- Qt Creator
- Code.Blocks
- KDevelop
- ...

Debuggers : GDB / LLDB with or without IDE
For beginners : I strongly suggest NO IDE until you run your first 10-20 C++ programs!
Warning ! Otherwise you get VERY CONFUSED !

Example 1_1: Hello world

The course repo https://github.com/agrechnev/cpp-course-2020

```
#include <iostream>

int main(){
        std::cout << "Carthago delenda est." << std::endl;
        return 0;
}
```

To compile a .cpp file

```
g++ -o hello hello.cpp
```

To build examples with CMake

```
mkdir build
cd build
cmake ..
cmake --build .
```

Simplest cmake projects

My first CMake project (**CMakeLists.txt**), it actually works !

```
add_executable(hello hello.cpp)
```

My second CMake project

```
# This is a comment
cmake_minimum_required(VERSION 3.1)

project(hello)

set(CMAKE_CXX_STANDARD 14)

set(SRCS
#    somefile.h somefile.cpp
     hello.cpp
)

add_executable(${PROJECT_NAME} ${SRCS})
```

C++ memory management

Suppose **a** is some object ...

b = a; // What does it mean?

C++ memory management : To copy or not to copy?

Suppose **a** is some object ...
b = a; // What does it mean?

Java, Python, Javascript ... : Object **b** is now a *reference* to **a**, no copying!
C++ : a is *copied* into b (by default, actually depends on object type !)

Java etc.: Easy to create reference, hard to copy.
Automatic memory management for children (Garbage collection !).

C++ : Easy to copy, harder to avoid copying (references, pointers, **shared_ptr**, ...)
Also **std::move()** and **std::swap()** . Works with containers (**std::vector** ...) at least !
No garbage collection ! Danger of **memory leaks** !
User has *complete control* over memory. And *responsibility* ! C++ is for *grownups*!

Behavior for different C++ classes:
std::vector, **std::string**, **std::map**, ... : copy *all* data (*deep* copy)
std::shared_ptr, **cv::Mat** : Create a reference (or a *very shallow* copy)

Example 1_2: C++ at a glance

Include several headers

```
#include <iostream>
#include <vector>
```

Command line arguments

```
int main(int argc, char **argv){
    using namespace std; // Now we can use cout without std:: !
    cout << "argc = " << argc << endl << endl;
    for (int i = 0; i < argc; ++i)
        cout << "argv[" << i << "] = " << argv[i] << endl;
```

char ** - pointer to pointers (aka array of pointers) to **char** (1 byte)
char * - C-string, 0-terminated (not used much in C++ actually, except as "string literals")
argv[i] - ith array element, a C-string (using pointer as array !)
int - signed integer type (usually 4 bytes = 32 bit)
argv[0] - Program name, array indices start at ZERO in C++! ALWAYS !

Defining a class

Class name, *access modifiers* **public**, **private**, **protected**

```
class Warrior {
public:  // Public stuff goes here
```

Constructor (always written as "Ctor" by the pros)

```
Warrior(const std::string &name, const std::string &weapon, int age)   :
            name(name),  // Init class field 'name' with parameter 'name'
            weapon(weapon),
            age(age) {
        std::cout << "Constructor : " << str() << std::endl;
    }
```

const std::string & - Constant reference (immutable data !) of type **std::string** (C++ string)
Such way to pass argument avoids *copying* ! Otherwise the string would be *copied* !
name(name) - initialize class field **name** with constructor parameter **name**
str() - class method (see below !)

Class Fields

```
private: // Private stuff goes here
    std::string name;
    std::string weapon;
    int age;
}; // Note the semicolon ';' here, unlike in Java, in C++ it is important !
```

Warrior::str() is *declared* inside class and *defined* (or "implemented") outside of it

```
std::string str() const;     // Method declaration
... }; ...
std::string Warrior::str() const{ // Method definition
    return "name = " + name + ", weapon = " + weapon + ", age = " +
std::to_string(age);
}
```

Getter : returns a private class field

```
const std::string &getName() const {
        return name; // Note: Returns reference, only OK for getters !
    }
```

Let us create a *container* (**std::vector**) of **Warrior** objects

```
vector<Warrior> warriors{
     {"Brianna", "Lightsaber", 17},   // {..} converted to Warrior
     {"Sita", "Spear", 15},           // {} is std::initializer_list
     {"Jean Grey", "Telekinesis", 25},
     Warrior("Ashe", "Zodiac Spear", 19)   // Explicit constructor
};
```

NB: **std::vector** is a *template*
std::vector<Warrior> = vector of Warrior objects

Let us add a few more **Warrior** objects !

```
warriors.emplace_back("Jaheira", "Club+5", 110);  // Construct in-place. Best !
Warrior wz("Zoe Maya Castillo", "Fists", 20);   // Local variable (stack)
warriors.push_back(wz); // OOPS! A copy operation!
warriors.push_back(Warrior("Casca", "Sword", 24));  // move operation
```

emplace_back - create a new object (via constructor) directly inside container !
push_back - add existing object to the container (copy or move)
Warrior wz("Zoe Maya Castillo", "Fists", 20); - Create a local Warrior variable (in stack)

How can we print all elements of a std:::vector ?

Range **for** loop
Use **const Warrior &** to avoid copying

```
for (const Warrior & w: warriors)     // Or "const auto & w"
    cout << w.str() << endl;
```

Traditional **for** loop
Indexing starts with ZERO
++i means "increase variable **i** by one", very C/C++ style !

```
for (int i = 0; i < warriors.size() ; ++i)
        cout << warriors[i].str() << endl;
```

Iterator **for** loop.
Iterators are *sort of* like pointers. Note the arrow operator "**->**". C-pointer syntax!
Ugly! Use iterators for algorithms like **std::sort** instead !
cbegin() = 1st element, **cend**() = *after* last element, "c" = const (cannot modify data)

```
for (auto it = cbegin(warriors); it != cend(warriors); ++it)
    cout << it->str() << endl;
```

Library of the day: Boost

Boost is a library basically developed by C++ language creators
as standard library extension + experiments

Much of Boost has by now migrated to C++ standard. For example:

- thread, atomic, chrono, functional (C++ 11)
- shared_ptr, unique_ptr, ... (C++ 11)
- filesystem, optional, any (C++ 17)

Other Boost components

- Boost.Asio TCP/UDP client/server
- Boost.Log Logging
- Boost.Test Unit test library

Do it yourself ! Try different Boost components.

Example 1_3 : Boost.Format

How do you do (formatted) output in C++? C++ style

cout << 2*2 << endl; // Many people hate it !

Or C style ?

printf("%d\n", 2*2); // Does not work with objects or C++ streams !

Try Boost.Format ! First don't forget the header ...

```
#include <boost/format.hpp>
```

... and then you simply use it with cout.

```
// boost::format is a powerful formatting tool, similar to printf()
std::cout << boost::format{"%1%.%2%.%3%"} % 12 % 5 % 2014 << std::endl;
// Also works with C-strings !
std::cout << boost::format{"I ate %1% %2% today !"} % 8 % "cakes" <<
std::endl;
// Or with classes (here : std::string)
int n(50);
std::string s = "cookies";
std::cout << boost::format{"I will eat %1% %2% tomorrow !"} % n % s <<
std::endl;
```

Course resources

Course repo

https://github.com/agrechnev/cpp-course-2020

OLD (2017) course repo:

https://github.com/agrechnev/cpp-course

Do it yourself !

Read and run examples 1_1, 1_2, 1_3 !
Look into CMakeLists.txt (cmake project files).

References.

Books:
1. S.B. Lippman, J. Lajoie, B.E. Moo, *C++ primer* (2012).
2. Scottt Meyers, *Effective Modern C++* (2014).
3. N.M. Josutttis, *The C++ Standard Library* (2014).
4. Bjarne Stroustrup, *Programming: Principles and Practice Using C++* (2014).

Read only books/tutorials on C++ 11 or later ! C++ 98 = EVIL !!!
Nowadays it is probably best to stick to C++ 17.

Other resources:
1. http://en.cppreference.com/w/
2. http://www.cplusplus.com/
3. https://stackoverflow.com/
4. https://www.google.com
5. https://stackoverflow.com/questions/388242/the-definitive-c-book-guide-and-list

Thank you for your attention !

title

text