Modern C++ for Computer Vision
Lecture 0: Introduction to C++

Ignacio Vizzo, Rodrigo Marcuzzi, Cyrill Stachniss

Course Organization

- **Lectures:** Tuesdays 10:00
  - **In Person** at the lecture hall at Nußalle 15.
- **Hands-on Tutorials:** Thursdays 14:00
  - **In Person** at the computer lab at Nußalle 15.
  - Also offline Tutorials.
  - Not all the Tutorials are provided by me.
- **Discord:** Fastest channel to communicate.

Course structure

The course is split in **two parts**:

1. **Learning the basics**
   - Lectures : Consists of 10 lectures.
   - Homeworks: Consists of 9 **hands-on** homeworks.
2. **Working on a project**
   - Plan and code **place-recognition pipeline**
   - Groups of 2 people

Workload

- **180 h** per semester (Workload)
- **60 h** per semester (Lectures)
- **16 weeks** per semester

Doing some math:

$$\left(\frac{180-60}{16}\right) \approx 8 \left[\frac{h}{week}\right]$$

What you will learn in course

- How to work in GNU/Linux
- How to write software in modern `C++`
- Core software development techniques
- How to work with images using `OpenCV`
- How to implement **Bag of Visual Words**

How is the course structured?

- **Part I:** C++ basics tools.
- **Part II:** The C++ core language.
- **Part III:** Modern C++.
- **Part IV:** Final project.

#	Week	Lecture	Tutorial	Homework	Topic	Homework Deadline
Part I: C++ tools						
-	4-Oct	[[No Lectures]]				
0	11-Oct	Course Introduction, Organization, Hello world	Setup GNU/Linux	-	-	-
1	18-Oct	Build System	How to submit homework // git	Homework 1	Build and tools	**31-Oct**
Part II: The C++ core language						
2	25-Oct	C++ Basic syntax	clang-tools and how to fight the hw bot	Homework 2	C++ programs	**7-Nov**
3	1-Nov	C++ Functions	gdb hands on	Homework 3	browser	**14-Nov**
4	8-Nov	C++ STL Library	hands on lab	Homework 4	stl	**21-Nov**
5	15-Nov	Filesystem + BoW Introduction	OpenCV	Homework 5	SIFT I/O	**28-Nov**
Part III: Modern C++						
6	22-Nov	Classes	hands on lab	Homework 6	igg_image	**5-Dec**
7	29-Nov	OOP	hands on lab	Homework 7	dict	**12-Dec**
8	6-Dec	Memory Managment	hands on lab	Homework 8	strattegy/memory	**19-Dec**
9	13-Dec	Generics Programing	Templates // pybind11	Homework 9	histograms/templates	**23-Dec**
Part IV: Final Project "Place recognition using Bag of Visual Words in C++"						
10	20-Dec	Bag of Visual Words	-			
11	10-Jan	[[No Lectures]]		Final Project	Final Project	**28 -Jan**
12	17-Jan					
13	24-Jan					

Course Content

Tools

- GNU/Linux **[Tutorial]**
  - Filesystem
  - Terminal
  - standard input/output
- Text Editor
  - Configuring
  - Terminal
  - Compile
  - Debug
- Build systems
  - headers/sources
  - Libraries
  - Compilation flags
  - CMake
  - 3rd party libraries
- Git **[Tutorial]**
- Homework submissions
- Gdb **[Tutorial]**
- Web-based tools
  - Quick Bench
  - Compiler Explorer
  - Cpp insights
  - Cppreference.com
- Clang-tools **[Tutorial]**
  - Clang-format
  - Clang-tidy
  - Clangd
  - Cppcheck
- Google test **[tutorial]**
- OpenCV **[tutorial]**

Core C++

- C++ basic syntax
- The “main” function
- #include statements
- Variables
- Control structures (if, for, while)
- I/O streams
- Input parameters

- Built-in types
- Operators
- Scopes
- Functions
- C++ strings
- Pass by value / Pass by reference

- Namespaces
- Containers
- std::tuple
- Iterators
- try/catch
- enum classes

- STL library
- STL Algorithms
- Function overloading
- Operator overloading
- String streams
- filesystem

Modern C++

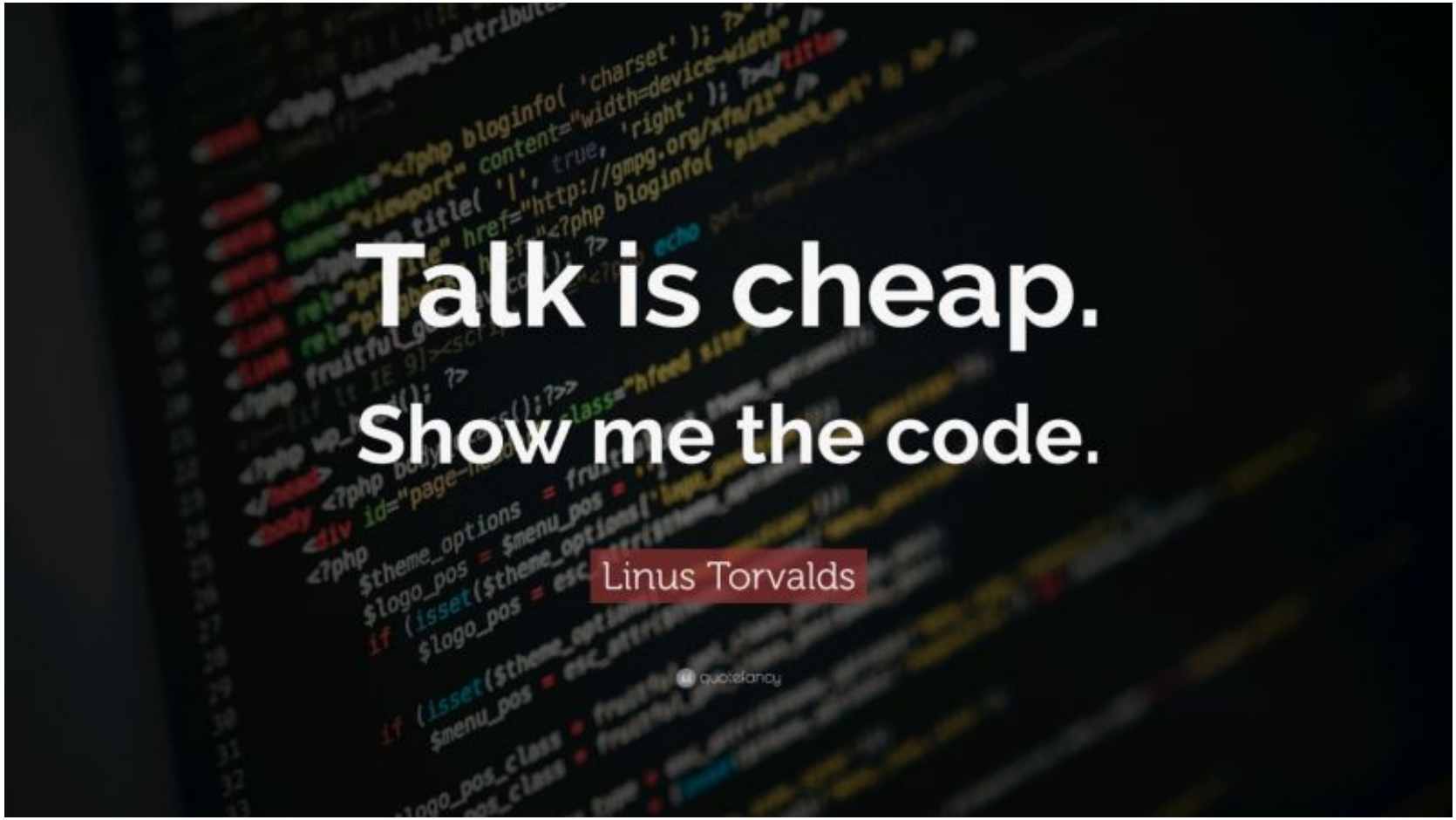
- Classes introduction
- Const correctness
- typedef/using
- static variables /methods
- Move Semantics
- Special Functions
- Singleton Pattern

- Inheritance
- Function Overriding
- Abstract classes
- Interfaces
- Strategy Pattern
- Polymorphism
- Typecasting

- Memory management
- Stack vs Heap
- Pointers
- new/delete
- **this** pointer
- Memory issues
- RAII
- Smart pointers

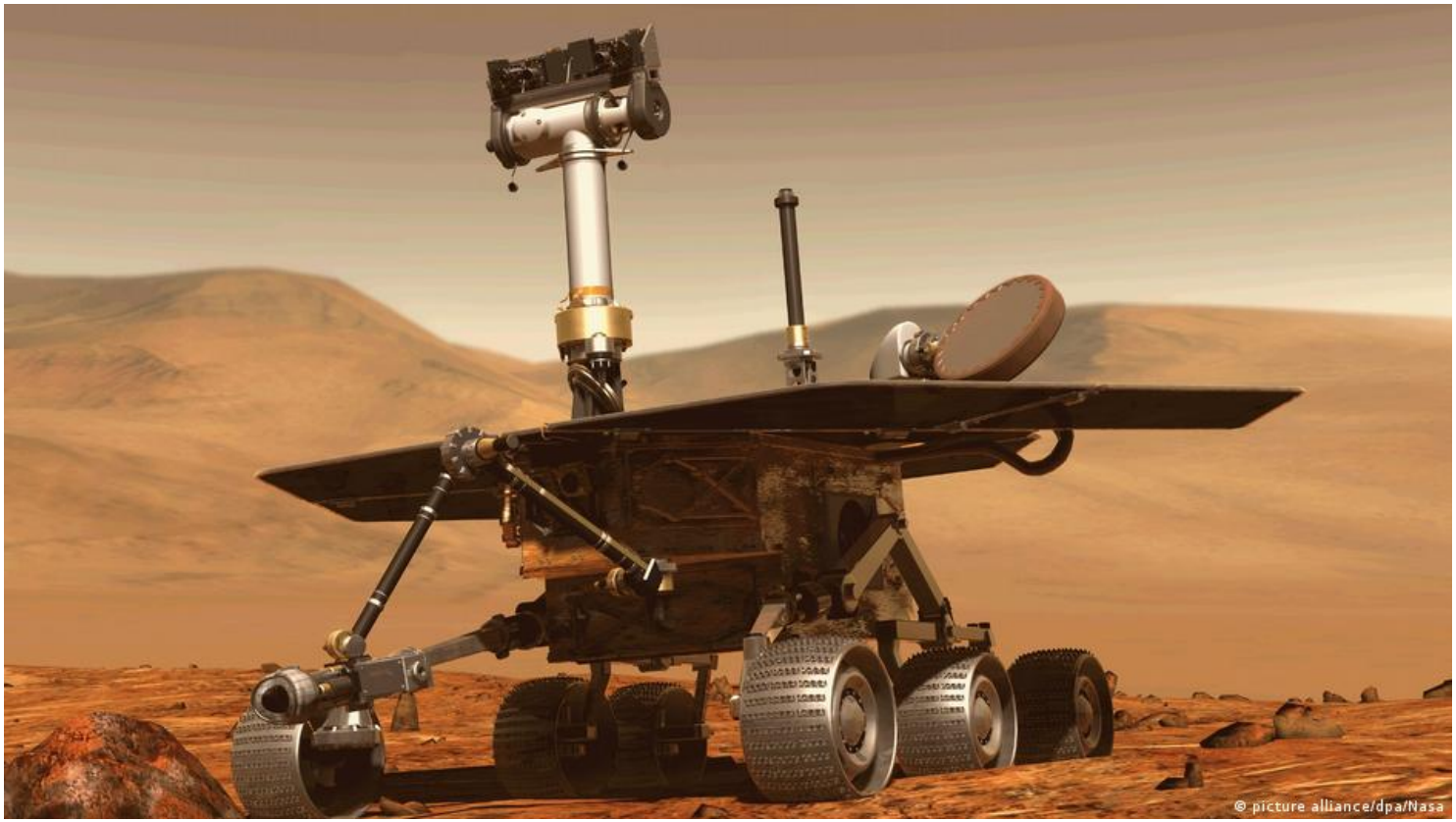
- Generic programming
- Template functions
- Template classes
- Static code generation
- lambdas

Course Philosophy



What you will do in this course

Real Robots



https://www.dw.com/en/nasa-bids-farewell-to-historic-mars-rover-opportunity/a-47509863

Efficiency

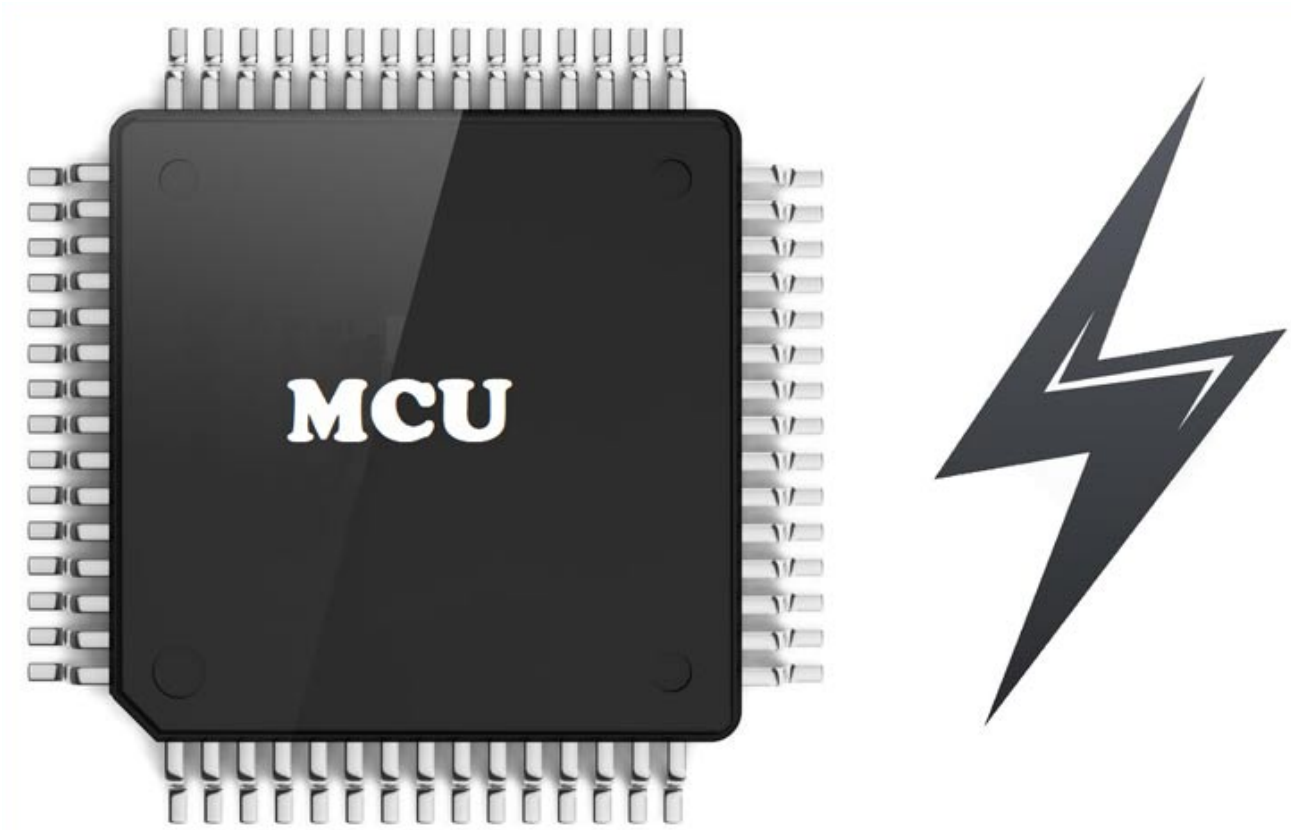
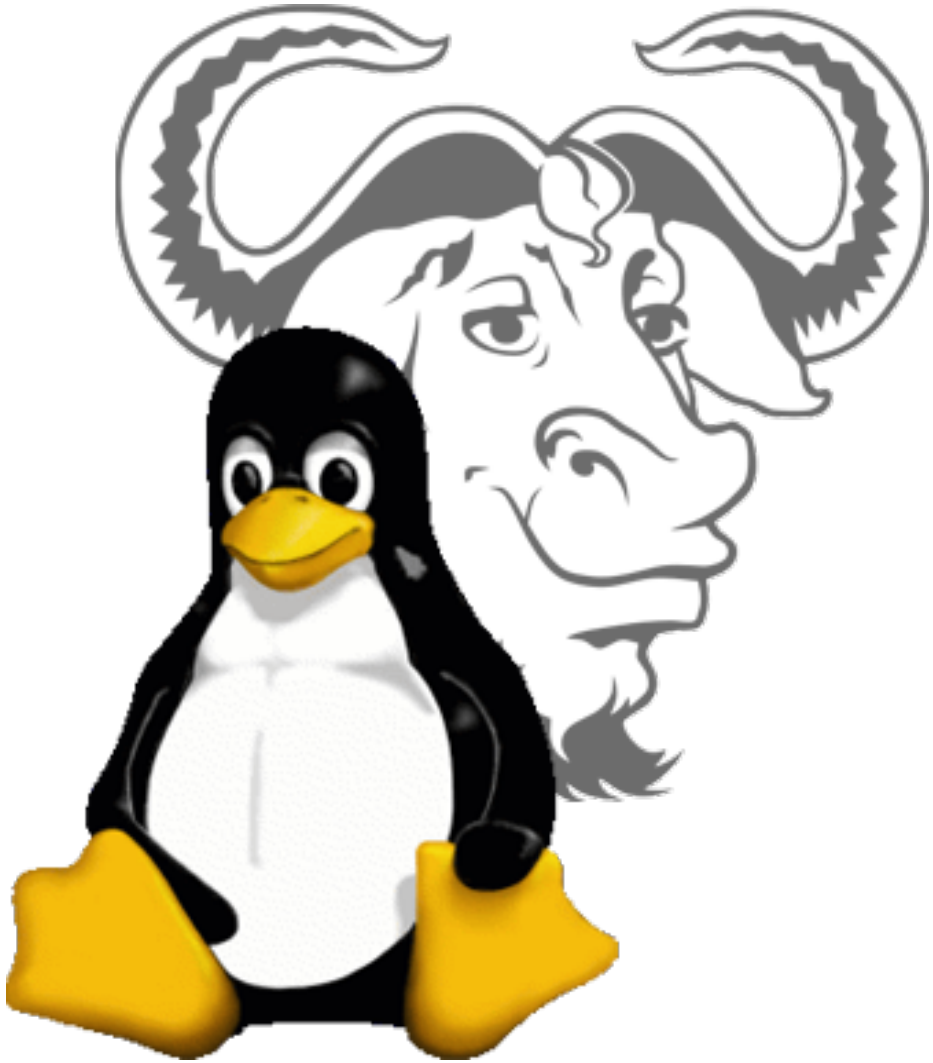


Image taken from https://circuitdigest.com/

ROS

Safety

We use GNU/Linux



Icon taken from Wikipedia

Is C++ a Popular Language?

PYPL Index (Worldwide)

Aug 2021	Change	Programming language	Share	Trends

PYPL Index (US)

Aug 2021	Programming language	Share
1		
2		
3		
4		
5		
6		
7		
8		
9		
10		
11		
12		
13		
14		
15		
16		
17		
18		
19		

Aug 2021	Au 202
1	1
2	3
3	2
4	4
5	5
6	6
7	7
8	9
9	1
10	1
11	1
12	1
13	4
14	8
15	1
16	1
17	1
18	1
19	3
20	1

Aug 2021
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15

PYPL Index (Germany)

Aug 2021	Programming language	Share
1	Python	31.1 %
2	Java	13.9 %
3	C#	10.47 %
4	JavaScript	8.82 %
5	PHP	4.78 %
6	C/C++	3.4 %
7	R	3.37 %
8	Rust	3.3 %
9	TypeScript	2.34 %
10	Matlab	2.34 %
11	Abap	2.33 %
12	Kotlin	2.32 %
13	VBA	2.02 %
14	Swift	1.79 %

C++ History: assembly

Benefits:

- Unbelievably simple instructions
- **Extremely** fast (when well-written)
- Complete control over your program

Why don't we always use assembly?

The following slides are adapted from Avery Wang

C++ History: assembly

```
main:                                   # @main
        push    rax
        mov     edi, offset std::cout
        mov     esi, offset .L.str
        mov     edx, 13
        call    std::basic_ostream<char, std::
    char_traits<char> >& std::__ostream_insert<char, std
    ::char_traits<char> >(std::basic_ostream<char, std::
    char_traits<char> >&, char const*, long)
        xor     eax, eax
        pop     rcx
        ret
_GLOBAL__sub_I_example.cpp:             #
    @_GLOBAL__sub_I_example.cpp
        push    rax
        mov     edi, offset std::__ioinit
        call    std::ios_base::Init::Init() [complete
    object constructor]
        mov     edi, offset std::ios_base::Init::~Init
    () [complete object destructor]
        mov     esi, offset std::__ioinit
        mov     edx, offset __dso_handle
        pop     rax
        jmp     __cxa_atexit            # TAILCALL
.L.str:
        .asciz  "Hello, world\n"
```

C++ History: assembly

Drawbacks:

- A lot of code to do simple tasks
- Hard to understand
- Extremely unportable

C++ History: Invention of C

Problem:

- Computers only understand `assembly` language.

Idea:

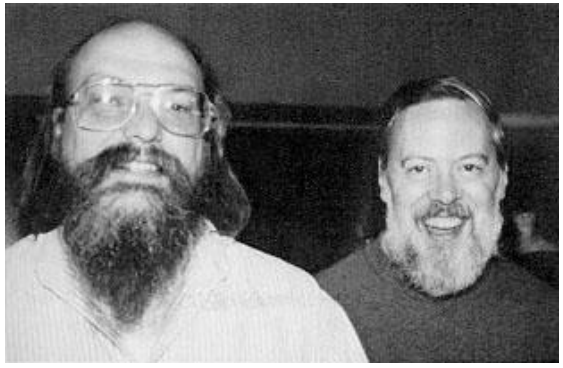
- Source code can be written in a more intuitive language
- An additional program can convert it into assembly [`compiler`]

C++ History: Invention of C

T&R created **C** in 1972, to much praise.

C made it easy to write code that was

- Fast
- Simple
- Cross-platform



Ken Thompson and Dennis Ritchie, creators of the C language.

C++ History: Invention of C

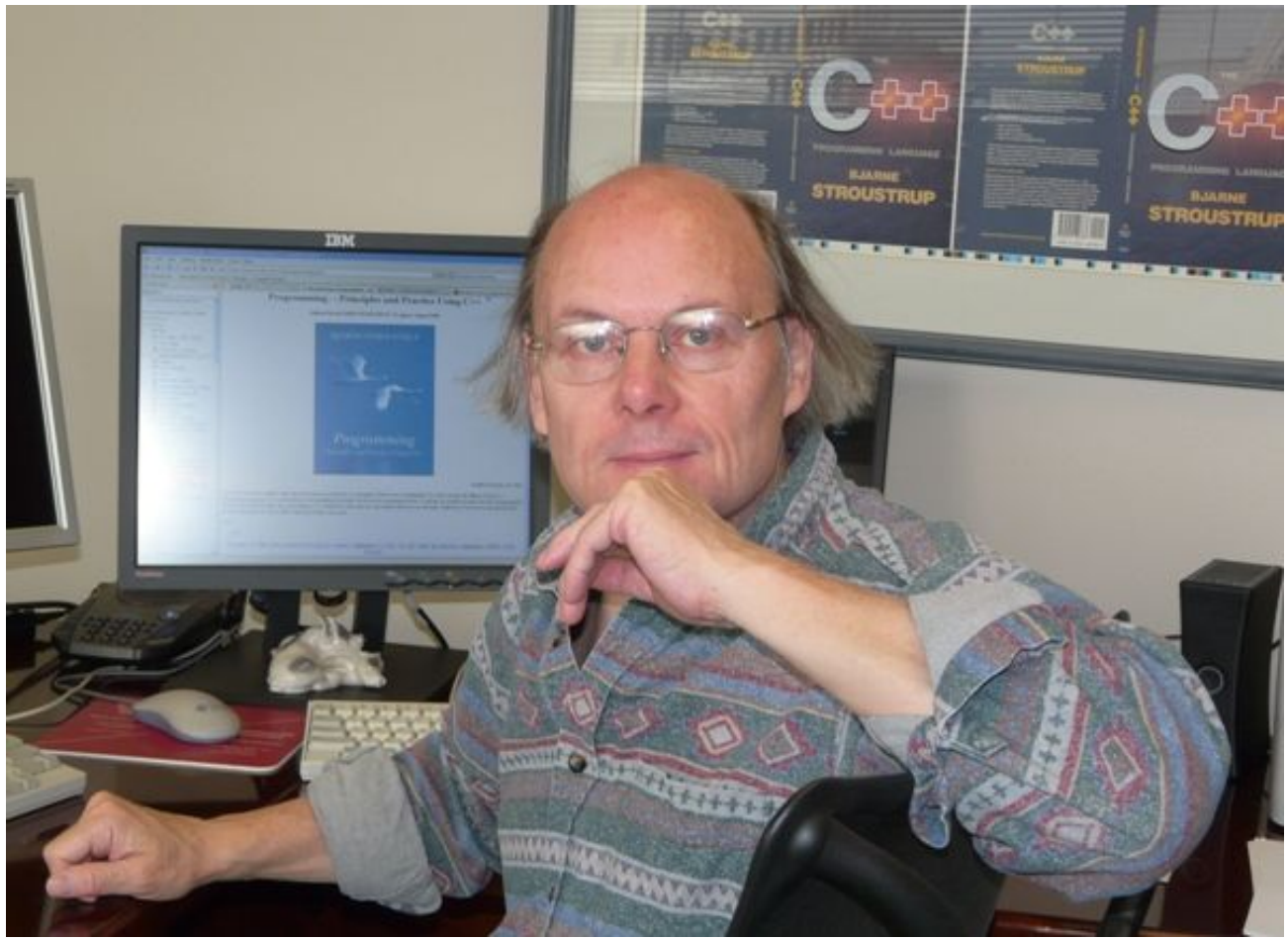
C was popular since it was simple.

This was also its weakness:

- No `objects` or `classes`.
- Difficult to write code that worked `generically`.
- Tedious when writing `large` programs.

C++ History: Welcome to C++

In 1983, the first vestiges of C++ were created by Bjarne Stroustrup.



C++ History: Welcome to C++

He wanted a language that was:

- Fast
- Simple to Use
- Cross-platform
- Had high level features

Evolution of C++

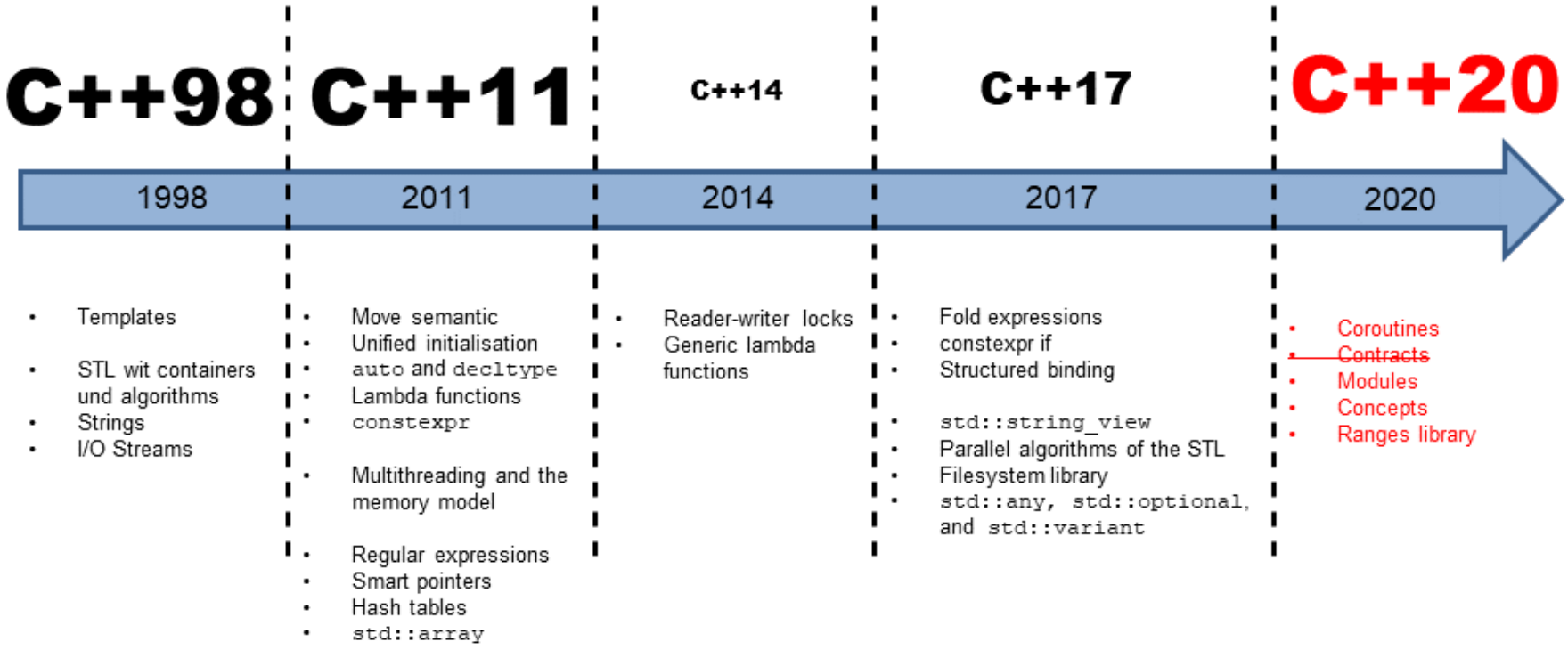


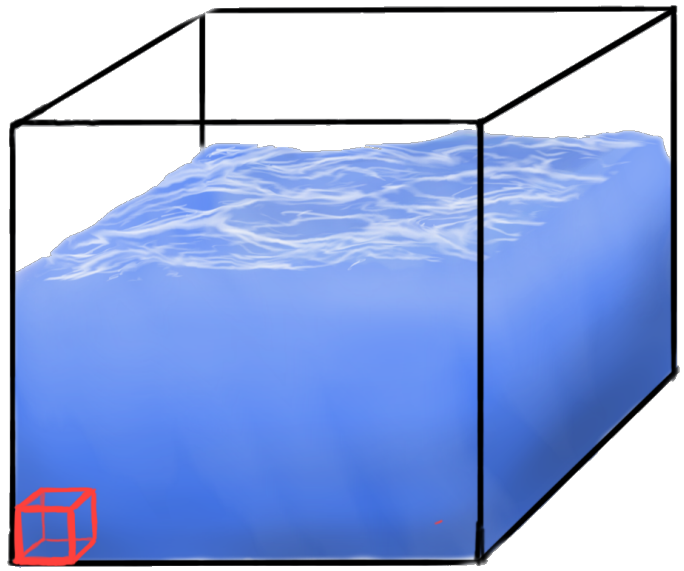
Image taken from https://www.modernescpp.com/

Design Philosophy of C++

- Multi-paradigm
- Express ideas and intent directly in code.
- Safety
- Efficiency
- Abstraction

Icon taken from Wikipedia

We won't teach you everything about C++



Within C++, there is a much smaller and cleaner language struggling to get out.

-Bjarne Stroustrup

Where to write C++ code

There are two options here:

- Use a C++**IDE**
  -  CLion
  -  Qt Creator
  -  Eclipse
- Use a **modern text editor** `[recommended]`
  -  Visual Studio Code `[my preference]`
  -  Sublime Text 3
  -  Atom
  -  VIM `[steep learning curve]`
  -  Emacs `[steep learning curve]`

Most icons are from Paper Icon Set: `https://snwh.org/paper`

Hello World!

Simple C++ program that prints `Hello World!`

```cpp
#include <iostream>

int main() {
  // Is this your first C++ program?
  std::cout << "Hello World!" << std::endl;
  return 0;
}
```

Comments and any whitespace: completely ignored

- A comment is text:
  - On one line that follows `//`
  - Between `/*` and `*/`
- All of these are valid C++:

```
int main() {return 0;}  // Ignored comment.
```

```
int main()

{          return 0;
}
```

```
int main() {
  return /* Ignored comment */ 0;
}
```

Good code style is important

> Programs are meant to be read by humans and only incidentally for computers to execute.
>
> -Donald Knuth

- Use `clang_format` to format your code
- use `cpplint` to check the style
- Following a style guide will save you time and make the code more readable
- We use **Google Code Style Sheet**
- Naming and style recommendations will be marked by `GOOGLE-STYLE` tag in slides

https://google.github.io/styleguide/cppguide.html

Everything starts with main

- **Every** C++ program starts with `main`
- `main` is a function that returns an error code
- Error code `0` means `OK`
- Error code can be any number in `[1, 255]`

```
1 int main() {
2   return 0;  // Program finished without errors.
3 }
```

```
1 int main() {
2   return 1;  // Program finished with error code 1.
3 }
```

#include directive

Two variants:

- `#include <file>` — system include files
- `#include "file"` — local include files

Copies the content of `file` into the current file

```
1 #include "some_file.hpp"
2 // We can use contents of file "some_file.hpp" now.
3 int main() { return 0; }
```

I/O streams for simple input and output

- Handle `stdin`, `stdout` and `stderr`:
  - `std::cin` — maps to `stdin`
  - `std::cout` — maps to `stdout`
  - `std::cerr` — maps to `stderr`
- `#include <iostream>` to use I/O streams
- Part of C++ standard library

```
1 #include <iostream>
2 int main() {
3   int some_number;
4   std::cout << "please input any number" << std::endl;
5   std::cin >> some_number;
6   std::cout << "number = " << some_number << std::endl;
7   std::cerr << "boring error message" << std::endl;
8   return 0;
9 }
```

Compile and run Hello World!

- We understand **text**
- Computer understands **machine code**
- **Compilation** is translation from text to machine code
- **Compilers** we can use on Linux:
  - Clang [*] [used in examples]
  - GCC

Compile and **run** Hello World example:

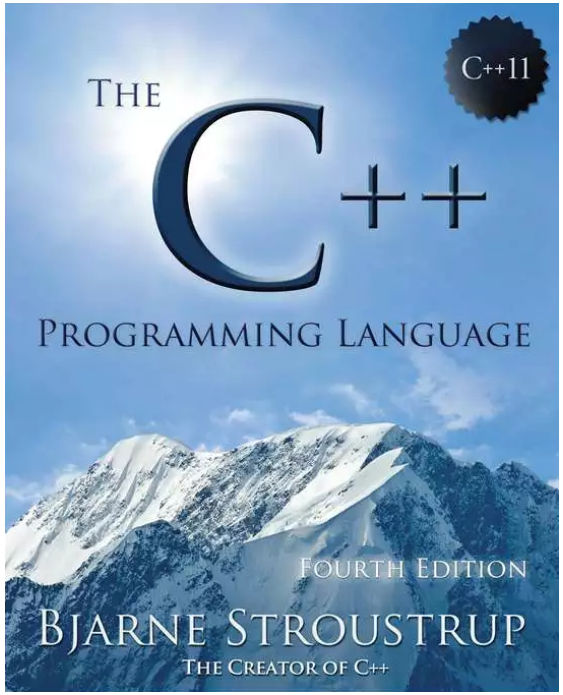
```
1 c++ -std=c++11 -o hello_world hello_world.cpp
2 ./hello_world
```

Suggested Video

”You Should Learn to Program” by Christian Genco at TEDxSMU

https://youtu.be/xfBWk4nw440

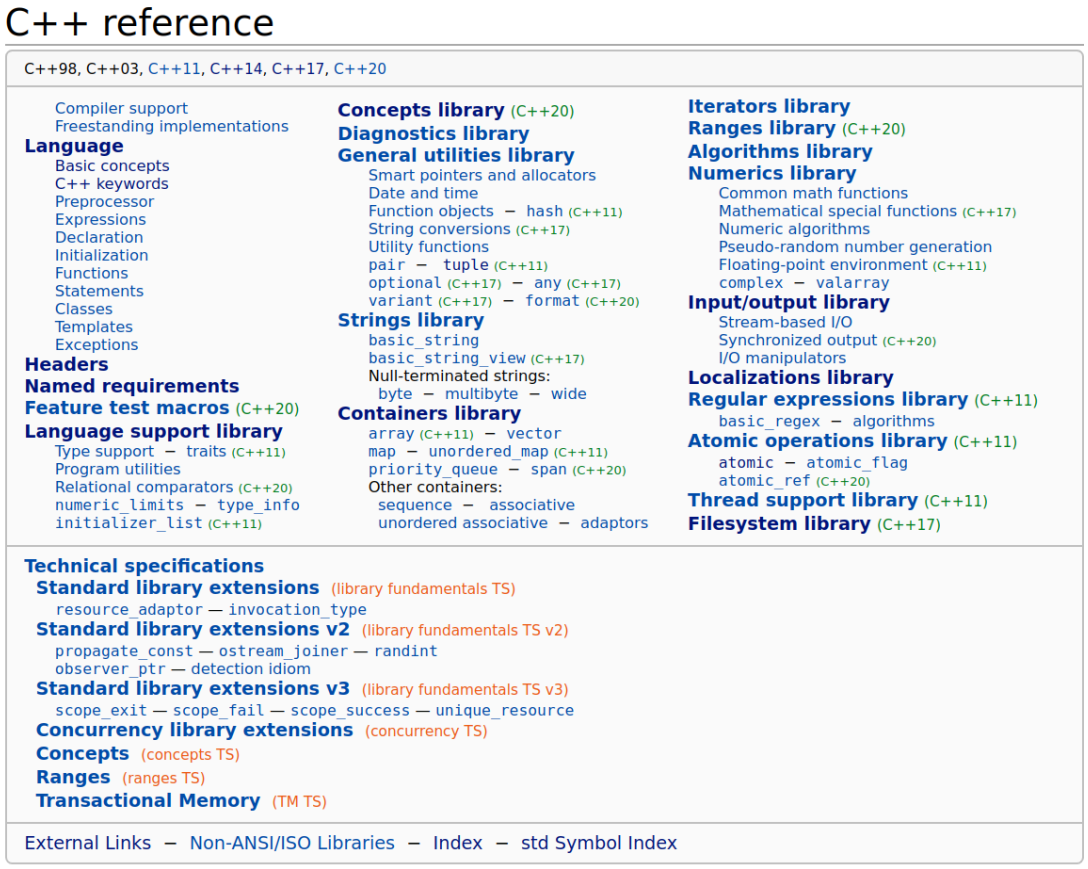
C++ Programming Language



- **Website:**

  http://www.stroustrup.com/4th.html

Best reference



https://en.cppreference.com/w/cpp

References

- **C++ Reference:**
  https://en.cppreference.com/w/cpp
- **Cpp Core Guidelines:**
  https://github.com/isocpp/CppCoreGuidelines
- **Google Code Styleguide:**
  https://google.github.io/styleguide/cppguide.html
- **C++ Tutorial:**
  http://www.cplusplus.com/doc/tutorial/