Modern C++ for Computer Vision and Image Processing

Lecture 0: The basics

Ignacio Vizzo and Cyrill Stachniss

UNIVERSITÄT BONN

Course Organization

- **Lectures:** Wednesday 16:00 (CEST)
  - Held at Youtube live-stream on the course channel.
  - Questions via Youtube channel during the lecture.
- **Tutorials:** Friday 15:00 (CEST)
  - Also offline Tutorials.
  - Also ”on-demand” Tutorials.
  - Not all the Tutorials are provided by me.
- **Discord:** Fastest channel to discuss.

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 **inverse image search**
   - 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 Linux
- How to write software with modern `C++`
- Core software development techniques
- How to work with images using `OpenCV`
- How to implement **inverse image search**

Check out **Google Image Search** for example: https://images.google.com/

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	Date	Lecture	Homework	Recommended Deadline	Official Deadline
Part I: C++ tools					
-	8-Apr	[[No Lectures]]	-	-	**-**
0	15-Apr	Course Introduction, Organization, Hello world	-	-	-
1	22-Apr	C++ Tools	Homework 1	3-May	**10-May**
Part II: The C++ core language					
2	29-Apr	C++ Basic syntax	Homework 2	10-May	**17-May**
3	6-May	C++ Functions	Homework 3	17-May	**24-May**
4	13-May	C++ Containers	Homework 4	24-May	**31-May**
5	20-May	C++ STL Library	Homework 5	31-May	**7-Jun**
Part III: Modern C++					
6	27-May	Classes	Homework 6	7-Jun	**14-Jun**
7	3-Jun	OOP	Homework 7	14-Jun	**21-Jun**
8	10-Jun	Memory Managment	Homework 8	21-Jun	**28-Jun**
9	17-Jun	Generics Programing	Homework 9	28-Jun	**5-Jul**
Part IV: Final Project "Place recognition using Bag of Visual Words in C++"					
10	24-Jun	Bag of Visual Words			
11	1-Jul	[[No Lectures]]	Final Project	**Final Examination Date**	
12	8-Jul				
13	15-Jul				

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

Please stop me!

Why?

Why C++? Why Linux? Why?

- Over 50 000 developers surveyed
- Nearly half of them use Linux
- C++ is the most used systems language (4.5 million users in 2015)
- C++ 11 is a modern language
- All companies want C++ in our field

Stack Overflow survey: https://insights.stackoverflow.com/survey/2018/

CLion survey: https://blog.jetbrains.com/clion/2015/07/infographics-cpp-facts-before-clion/

Why C++

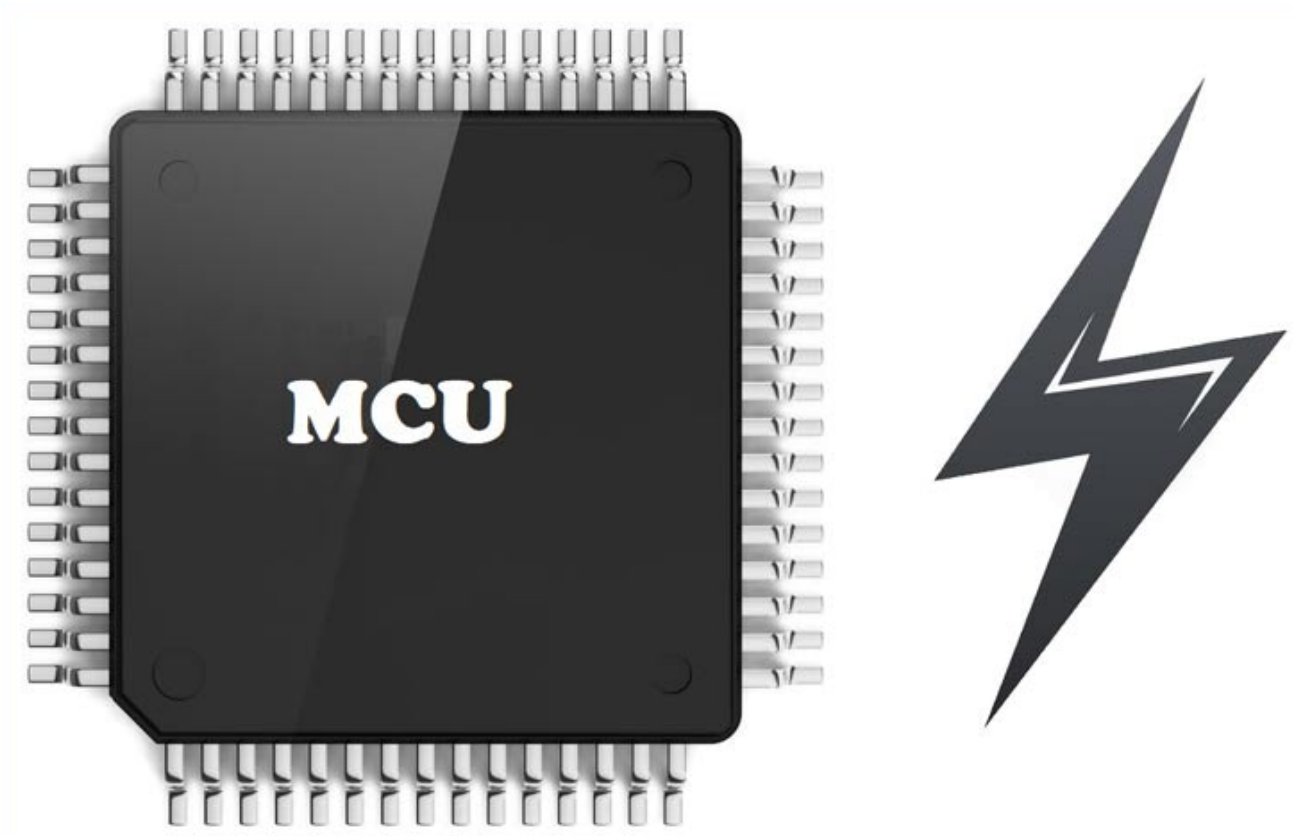
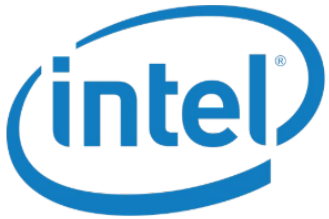


Image taken from https://circuitdigest.com/

Companies that use C++



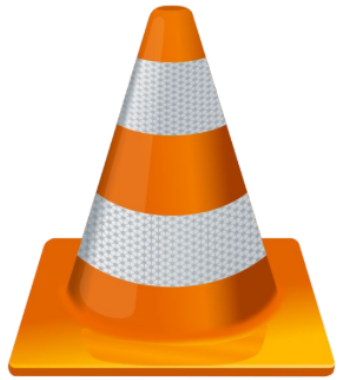
The following slides are adapted from Avery Wang

More info at `http://www.stroustrup.com/applications.html`

Browsers written in C++

Slides adapted from Avery Wang

Software written in C++



Games written in C++

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

```
1  main:                                   # @main
2          push    rax
3          mov     edi, offset std::cout
4          mov     esi, offset .L.str
5          mov     edx, 13
6          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)
7          xor     eax, eax
8          pop     rcx
9          ret
10 _GLOBAL__sub_I_example.cpp:             #
       @_GLOBAL__sub_I_example.cpp
11         push    rax
12         mov     edi, offset std::__ioinit
13         call    std::ios_base::Init::Init() [complete
       object constructor]
14         mov     edi, offset std::ios_base::Init::~Init
       () [complete object destructor]
15         mov     esi, offset std::__ioinit
16         mov     edx, offset __dso_handle
17         pop     rax
18         jmp     __cxa_atexit            # TAILCALL
19 .L.str:
20         .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:

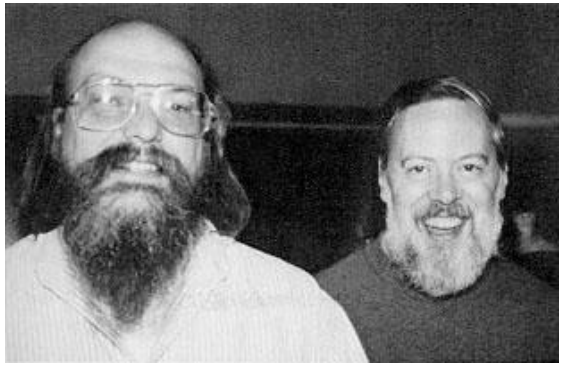
- Source code can be written in a more intuitive language
- An additional program can convert it into assembly [`compiler`]

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

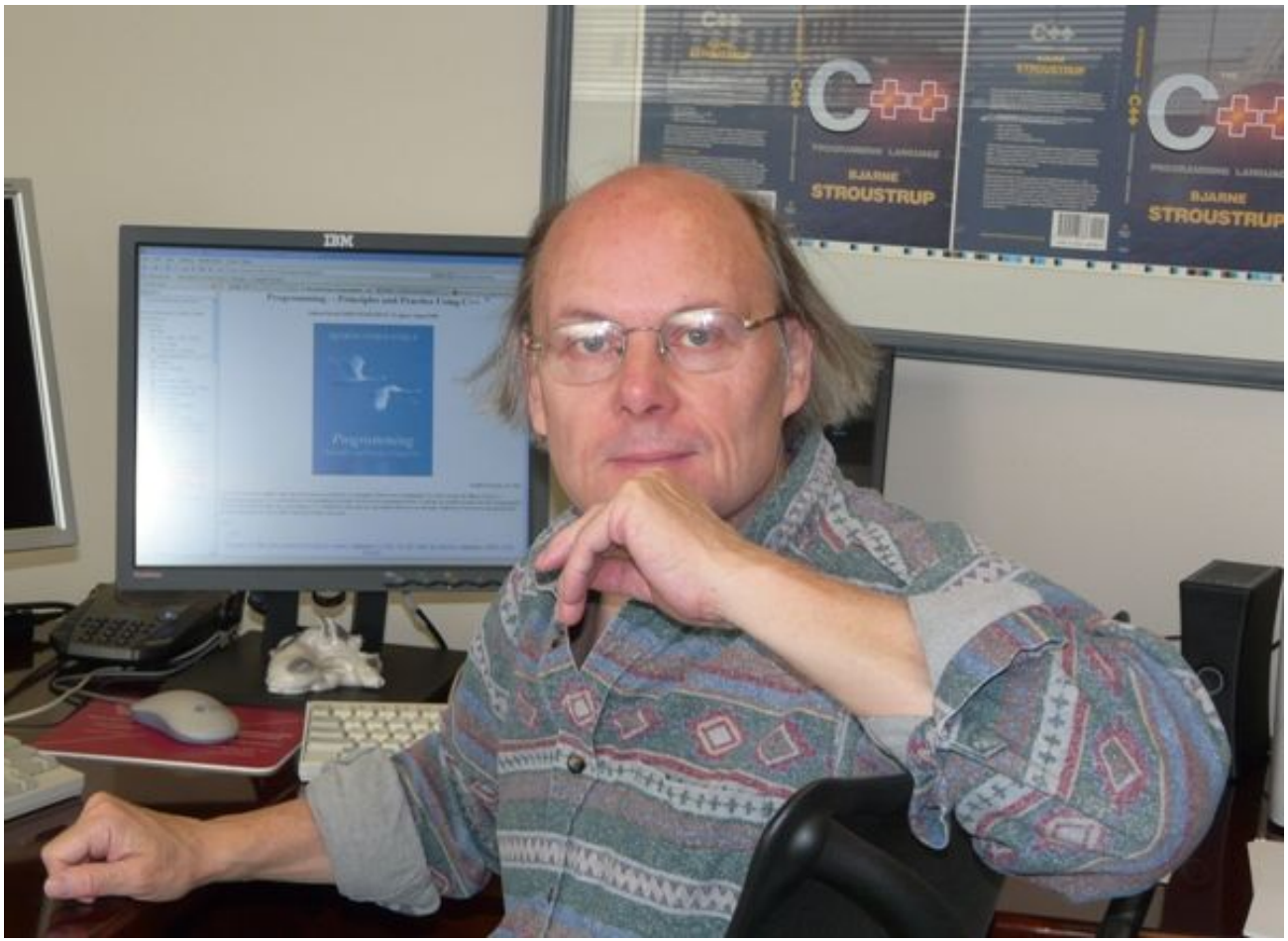
C was popular since it was simple.

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++

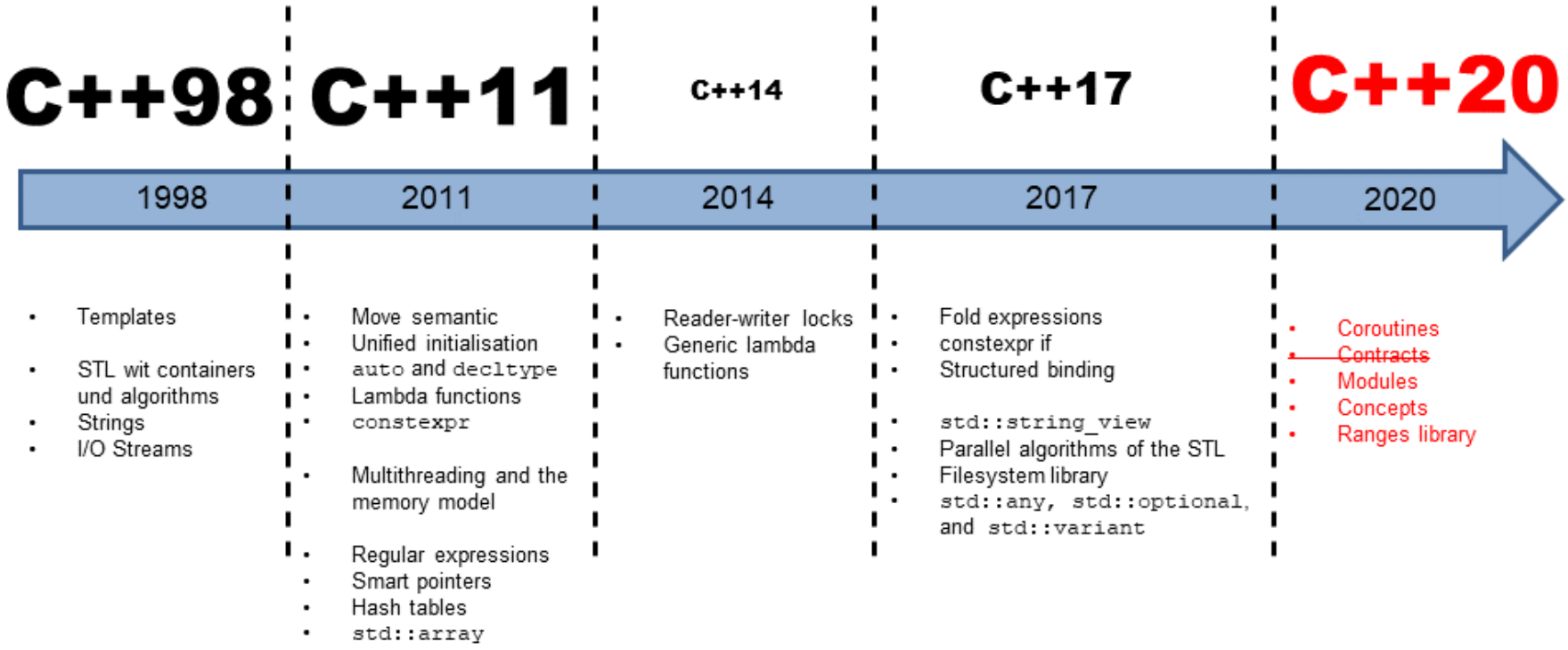
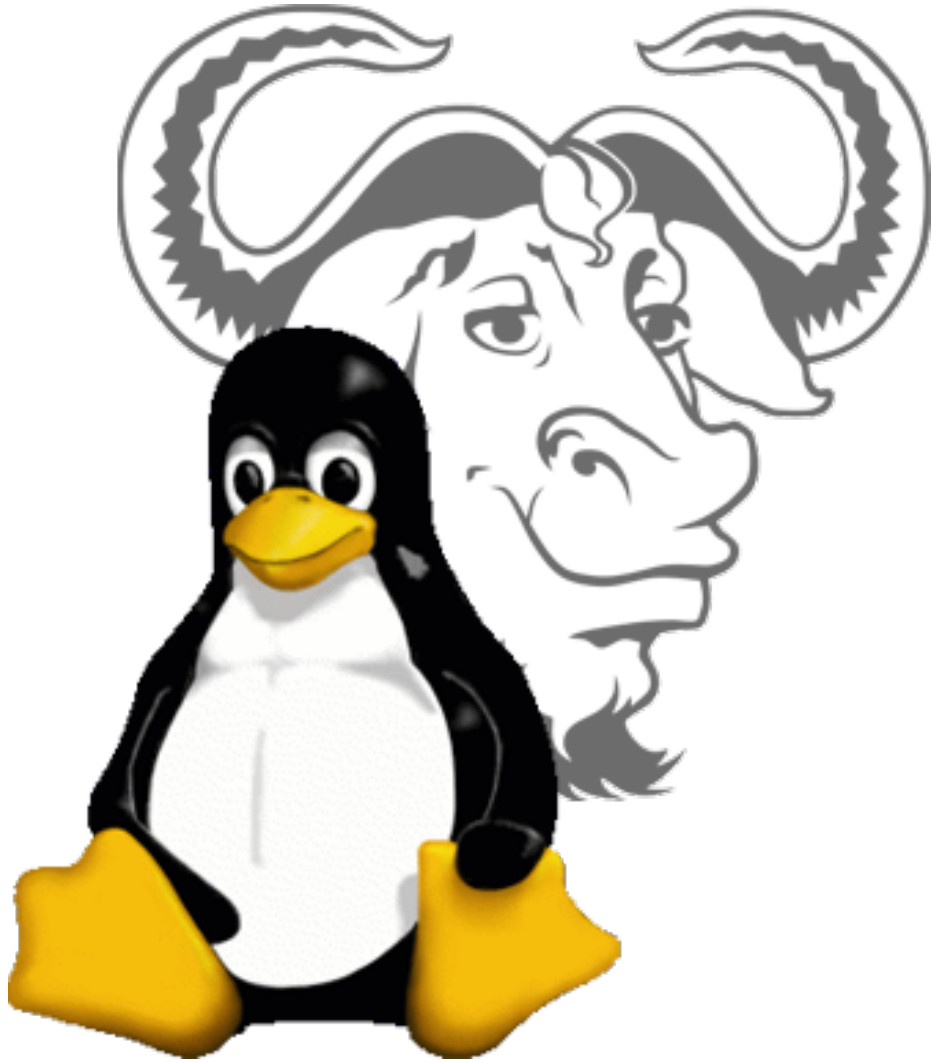


Image taken from https://www.modernescpp.com/

Design Philosophy of C++

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

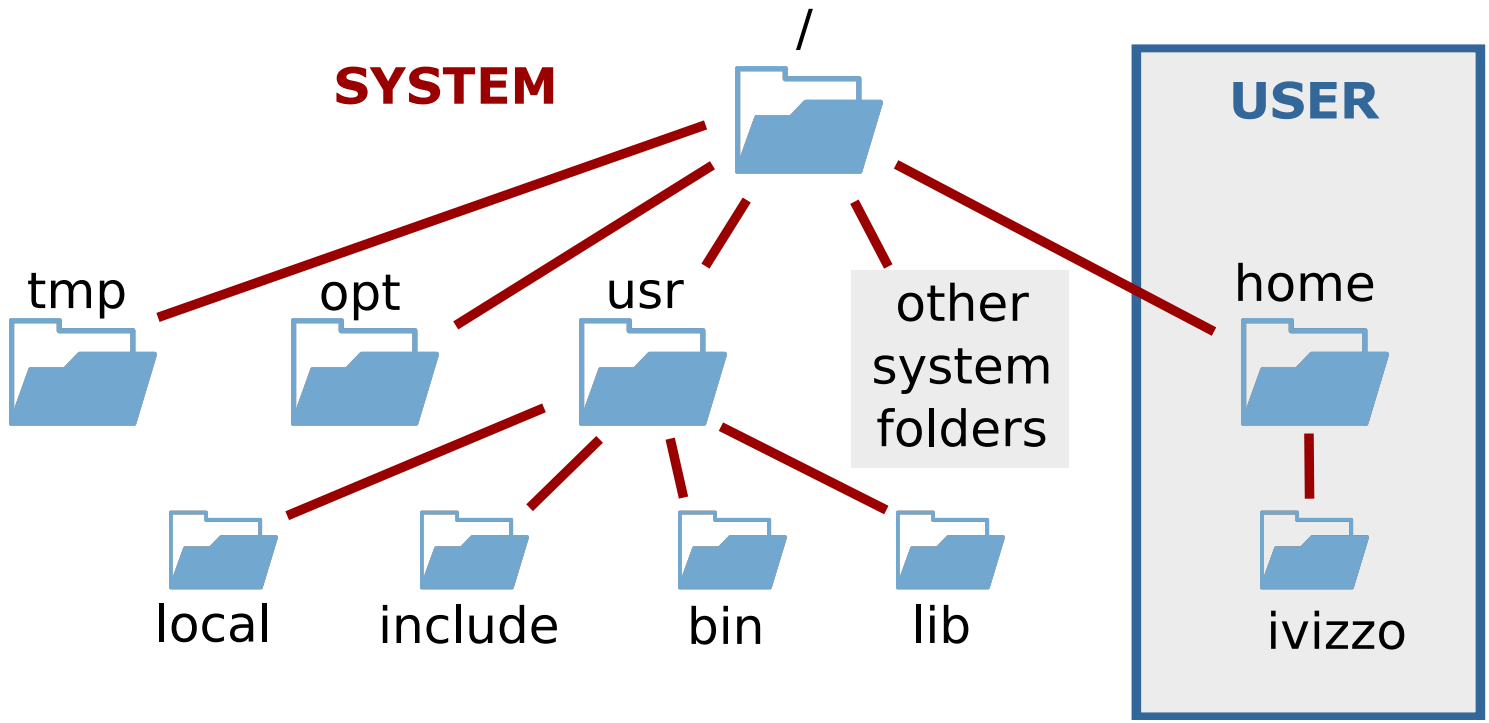


Icon taken from Wikipedia

What is GNU/Linux?

- Linux is a free **Unix-like OS**
- Linux kernel implemented by Linus Torvalds
- **Extremely popular:** Android, ChromeOS, servers, supercomputers, etc.
- Many **Linux distributions** available
- Use any distribution if you have preference
- Examples will be given in **Ubuntu**

Linux directory tree



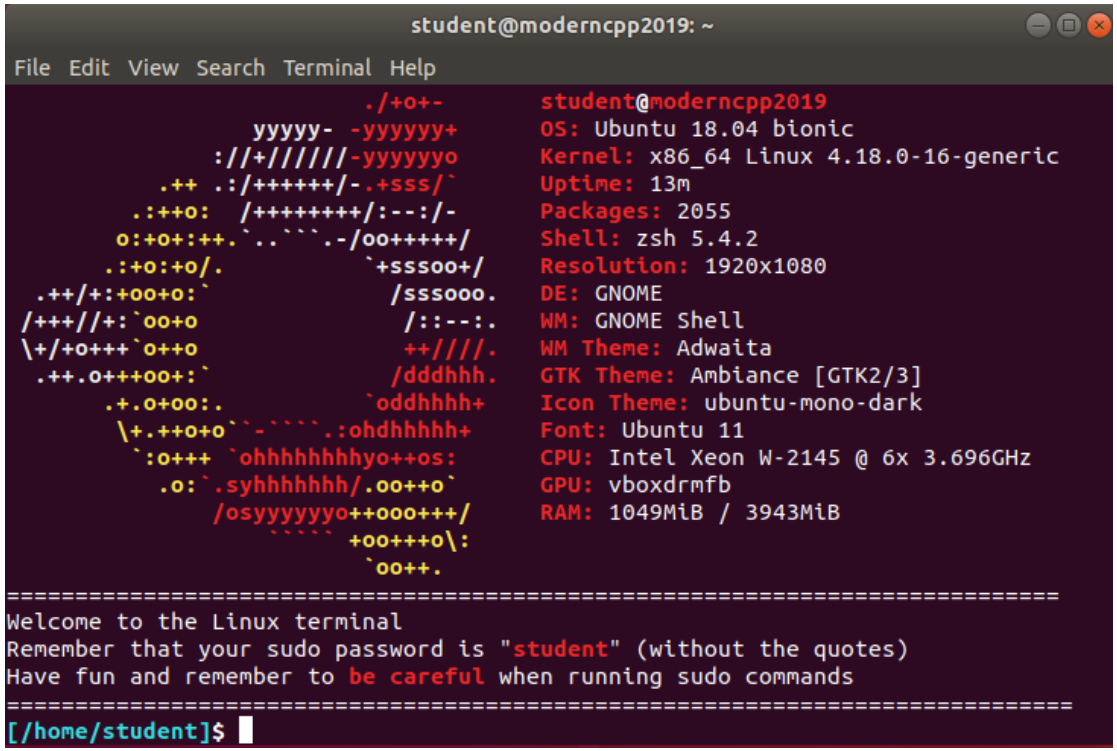
- Tree organization starting with root: `/`
- There are no volume letters, e.g. `C:`, `D:`
- User can only access his/her own folder

Understanding files and folders

- Folders end with `/` e.g. `/path/folder/`
- Everything else is files, e.g. `/path/file`
- Absolute paths start with `/`
  while all other paths are relative:
  - `/home/ivizzo/folder/` — **absolute** path to a folder
  - `/home/ivizzo/file.cpp` — **absolute** path to a file
  - `folder/file` — **relative** path to a file
- Paths are case sensitive:
  `filename` is different from `FileName`
- Extension is part of a name:
  `filename.cpp` is different from `filename.png`

Linux terminal

- Press Ctrl + Alt + T to open terminal



- Most tasks can be done faster from the terminal than from the GUI

Navigating tree from terminal

- Terminal is always in some folder
- `pwd`: **p**rint **w**orking **d**irectory
- `cd <dir>`: **c**hange **d**irectory to `<dir>`
- `ls <dir>`: **l**i**s**t contents of a directory

- Special folders:
  - `/` — root folder
  - `~` — home folder
  - `.` — current folder
  - `..` — parent folder

Structure of Linux commands

Typical structure

```
${PATH}/command [ options ] [ parameters ]
```

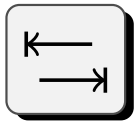
- `${PATH}/command`: obsolute or relative path to the program binary
- `[options]`: program-specific options
  e.g. `-h`, or `--help`
- `[parameters]`: program-specific parameters
  e.g. input files, etc.

Use help with Linux programs

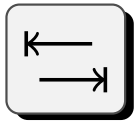
- `man <command>` — **man**ual
  exhaustive manual on program usage
- `command -h/--help`
  usually shorter help message

```
[/home/student]$ cat --help
Usage: cat [OPTION]... [FILE]...
Concatenate FILE(s) to standard output.
  -A, --show-all            equivalent to -vET
  -b, --number-nonblank     number nonempty output lines

Examples:
  cat f -  Output fs contents, then standard input.
  cat      Copy standard input to standard output.
```

Using command completion

Pressing  while typing:

- completes name of a file, folder or program
- "beeps" if current text does not match any file or folder uniquely

Pressing  **twice** shows all potential matches

Example:

```
[/home/student]$ cd D [TAB] [TAB]
Desktop/    Documents/  Downloads/
```

Files and folders

- `mkdir [-p] <foldername>` — **m**a**k**e **dir**ectory
  Create a folder `<foldername>`
  (with all parent folders `[-p]`)
- `rm [-r] <name>` — **r**e**m**ove [recursive]
  Remove file or folder `<name>`
  (With folder contents `[-r]`)
- `cp [-r] <source> <dest>` — **c**o**p**y
  Copy file or folder from `<source>` to `<dest>`
- `mv <source> <dest>` — **m**o**v**e
  Move file or folder from `<source>` to `<dest>`

Using placeholders

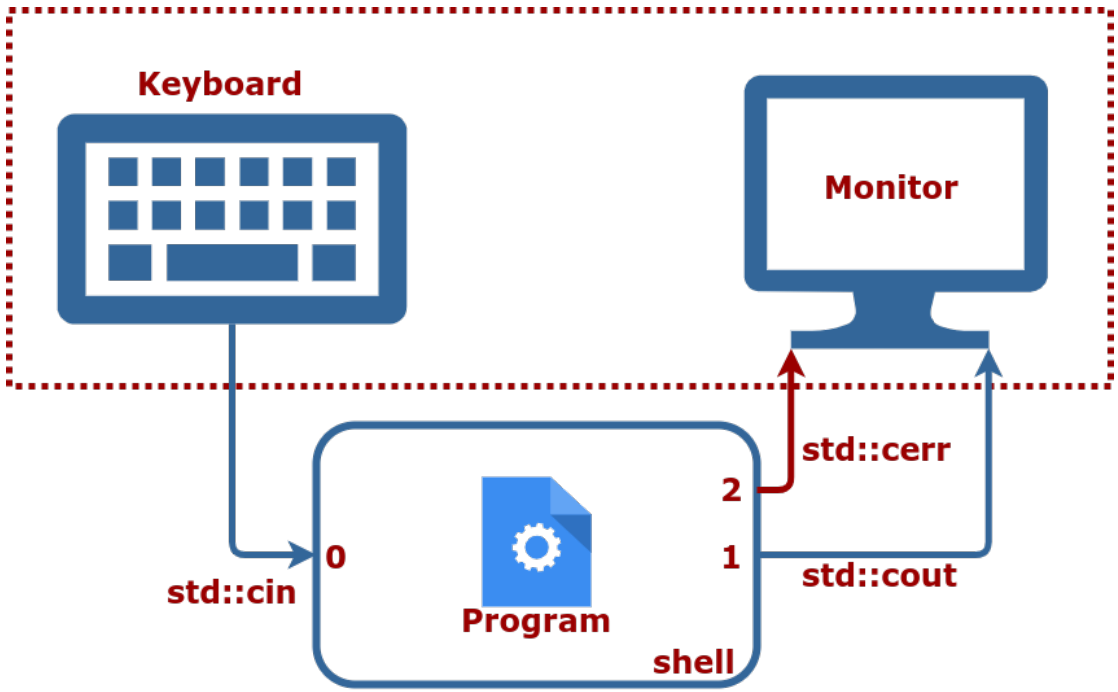
Placeholder	Meaning
`*`	Any set of characters
`?`	Any single character
`[a-f]`	Characters in `[abcdef]`
`[^a-c]`	Any character **not** in `[abc]`

Can be used with most of terminal commands: `ls`, `rm`, `mv` etc.

```
[/home/student/Examples/placeholders]$ ls
u01.tex    v01.pdf     v01.tex
u02.tex    v02.pdf     v02.tex
u03.tex    v03.pdf     v03.tex

[/home/student/Examples/placeholders]$ ls *.pdf
v01.pdf   v02.pdf   v03.pdf

[/home/student/Examples/placeholders]$ ls u*
u01.tex   u02.tex   u03.tex

[/home/student/Examples/placeholders]$ ls ?01*
u01.tex    v01.pdf   v01.tex

[/home/student/Examples/placeholders]$ ls [uv]01*
u01.tex    v01.pdf   v01.tex

[/home/student/Examples/placeholders]$ ls u0[^12].tex
u03.tex
```

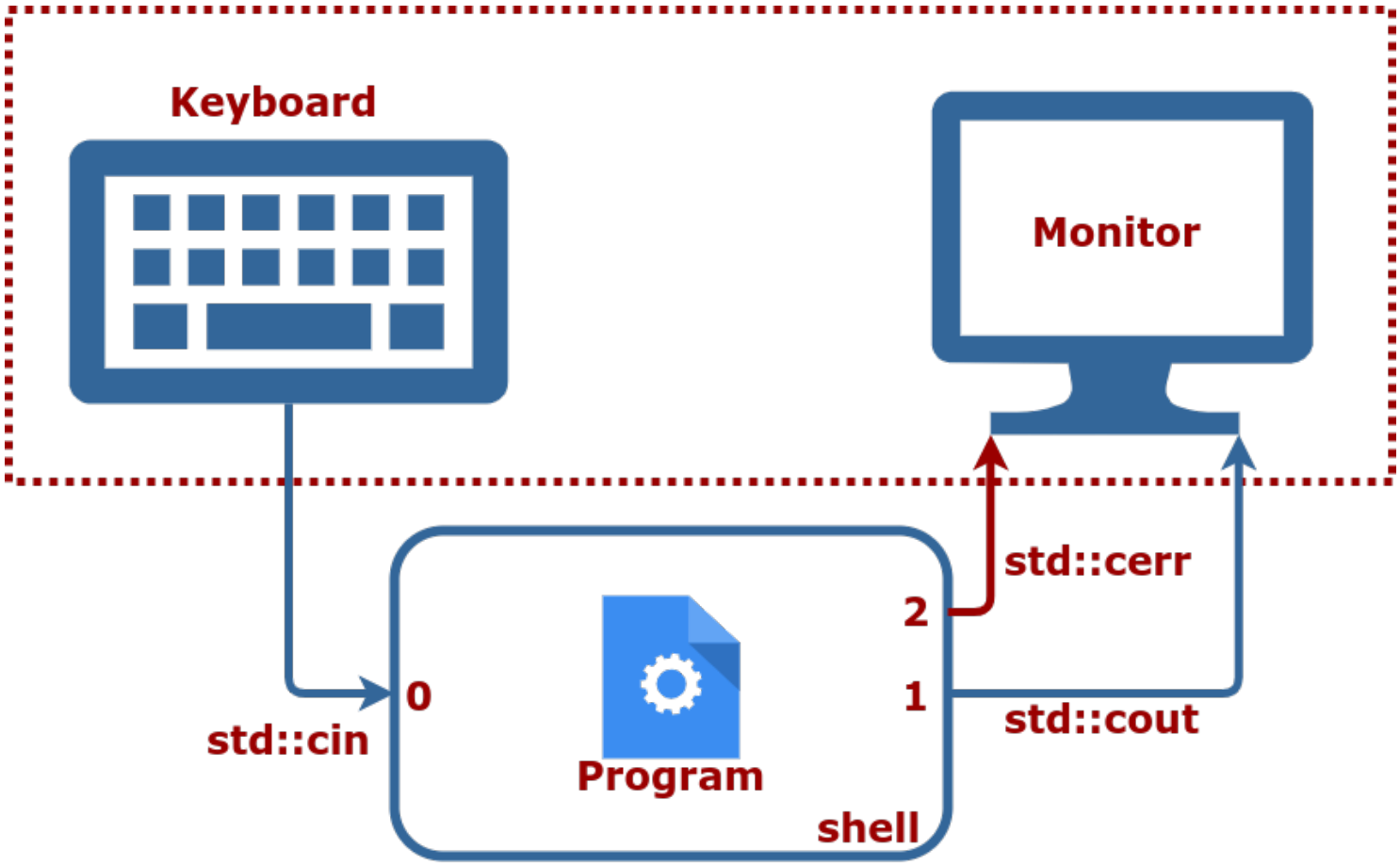
Standard input/output channels

- Single input channel:
  - `stdin`: **St**an**d**ard **in**put: channel 0
- Two output channels:
  - `stdout`: **St**an**d**ard **out**put: channel 1
  - `stderr`: **St**an**d**ard **err**or output: channel 2



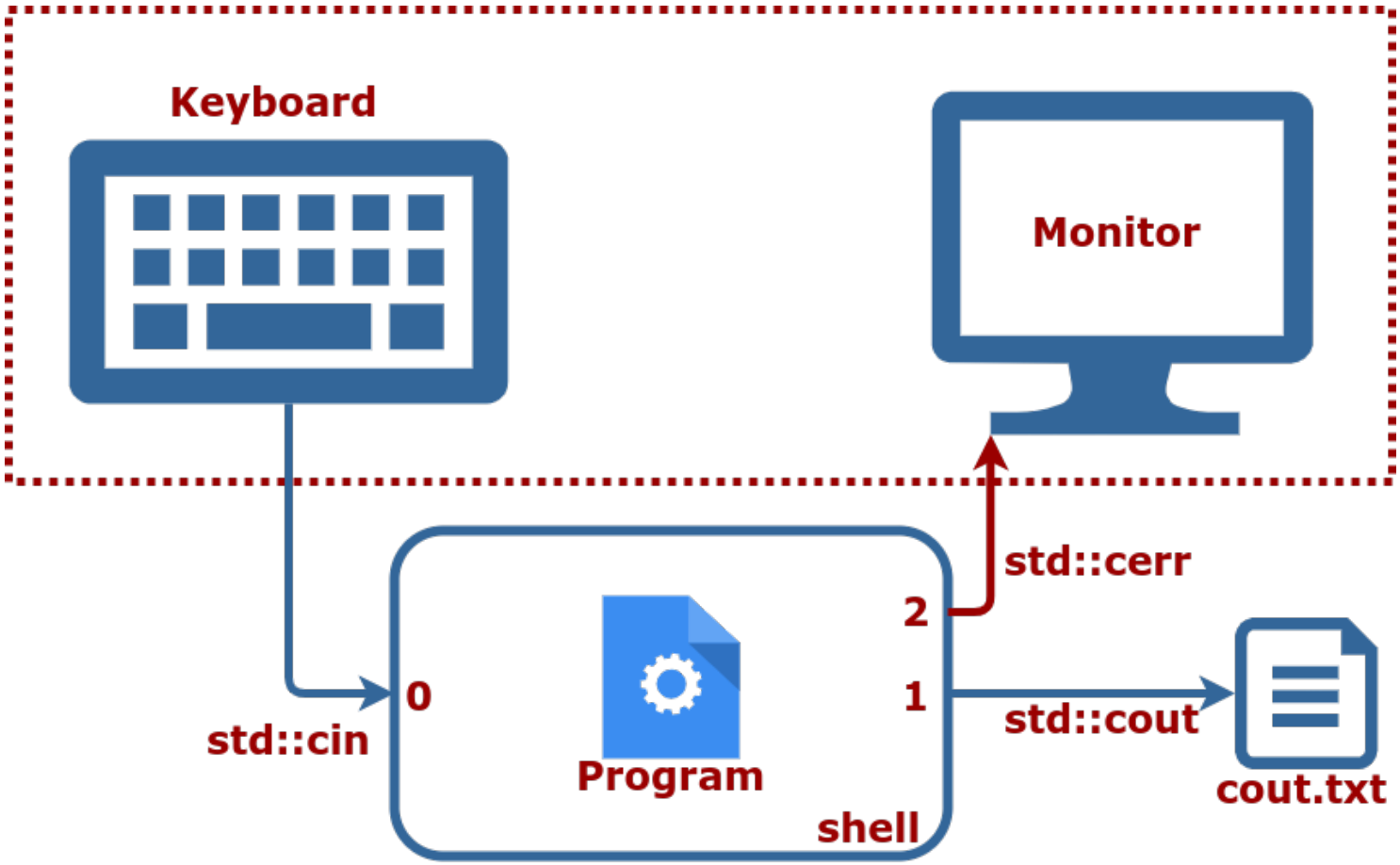
Standard input/output channels

```
$ program
```



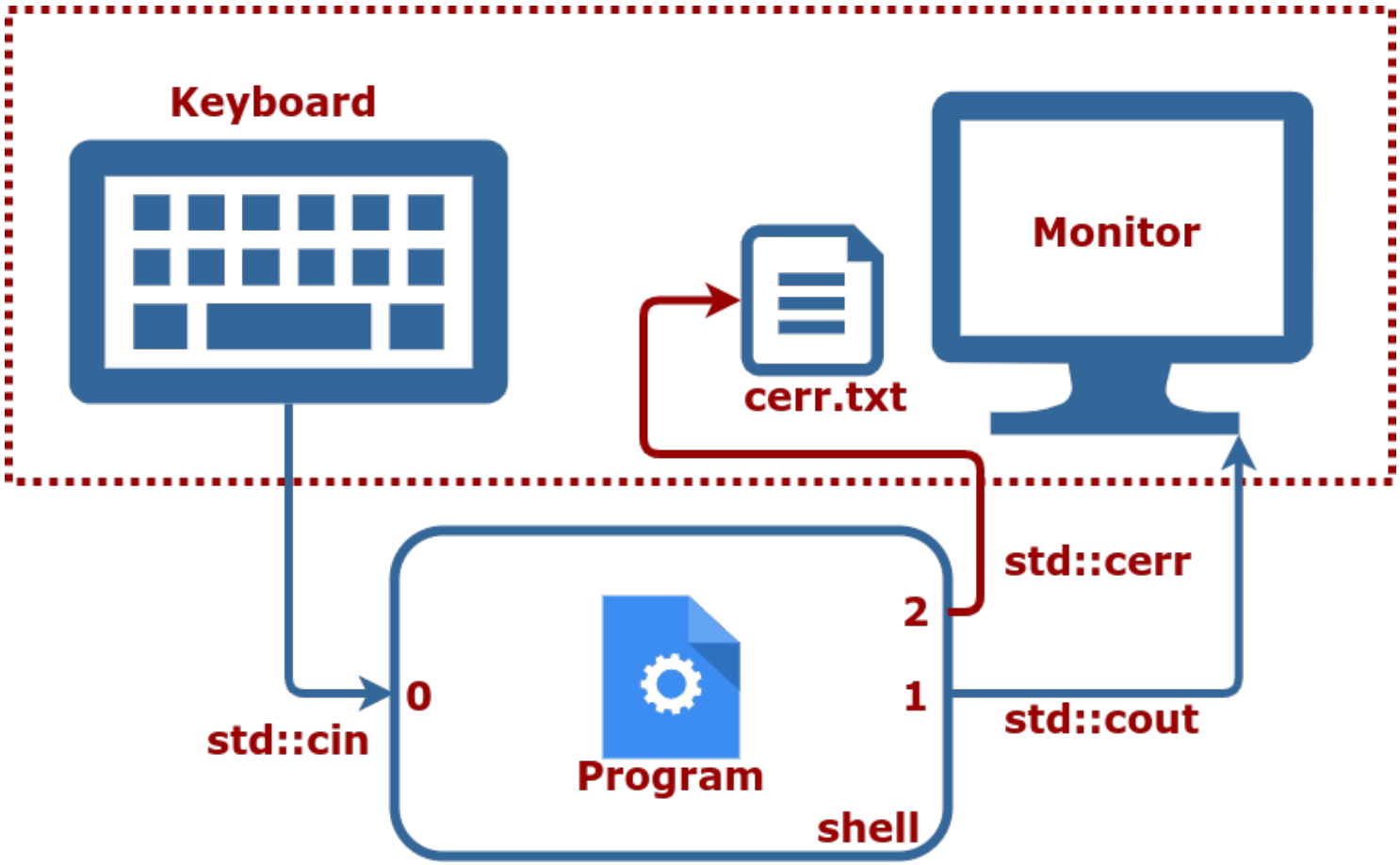
Redirecting stdout

```
$ program 1>cout.txt
```



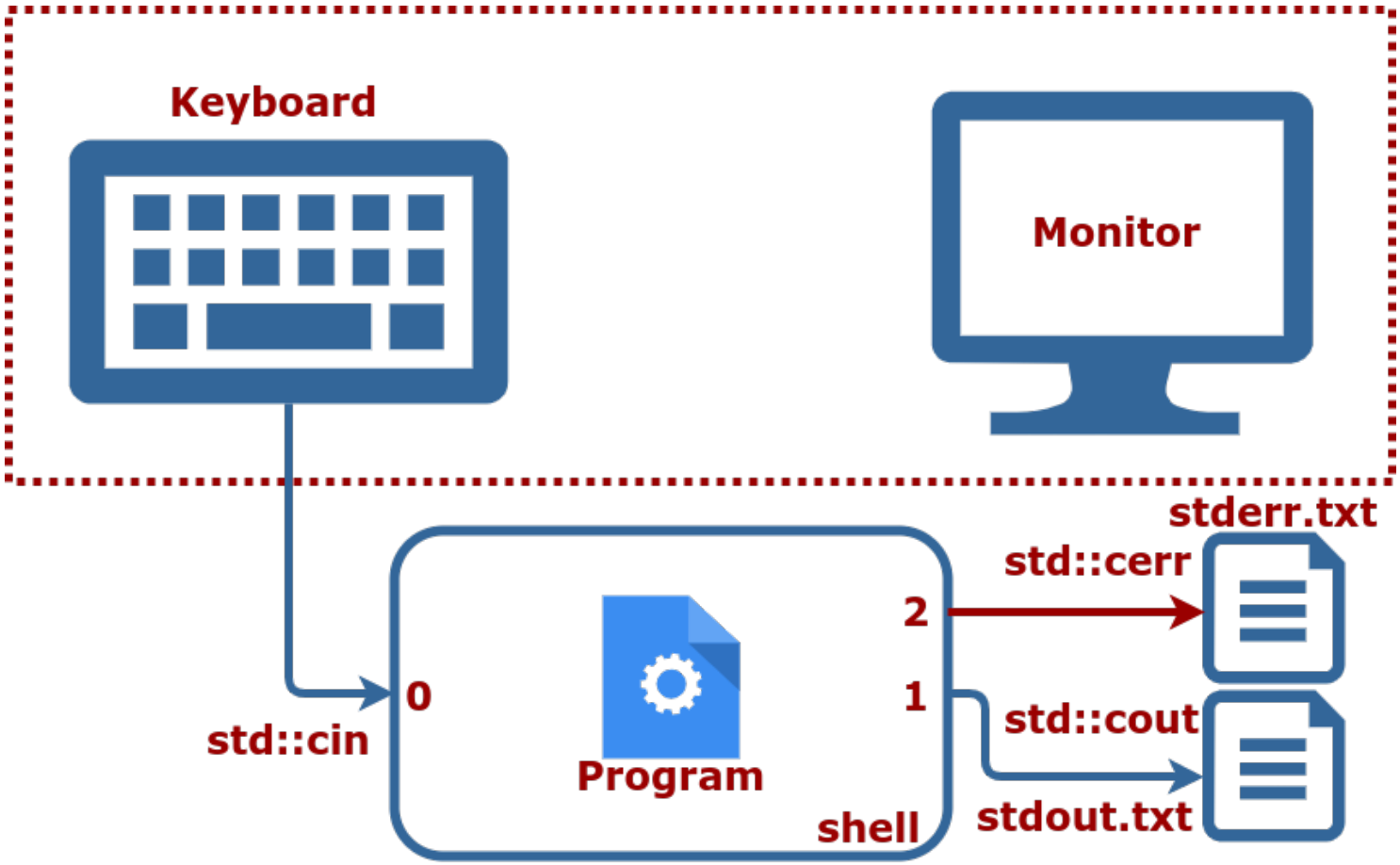
Redirecting stderr

```
$ program 2>cerr.txt
```



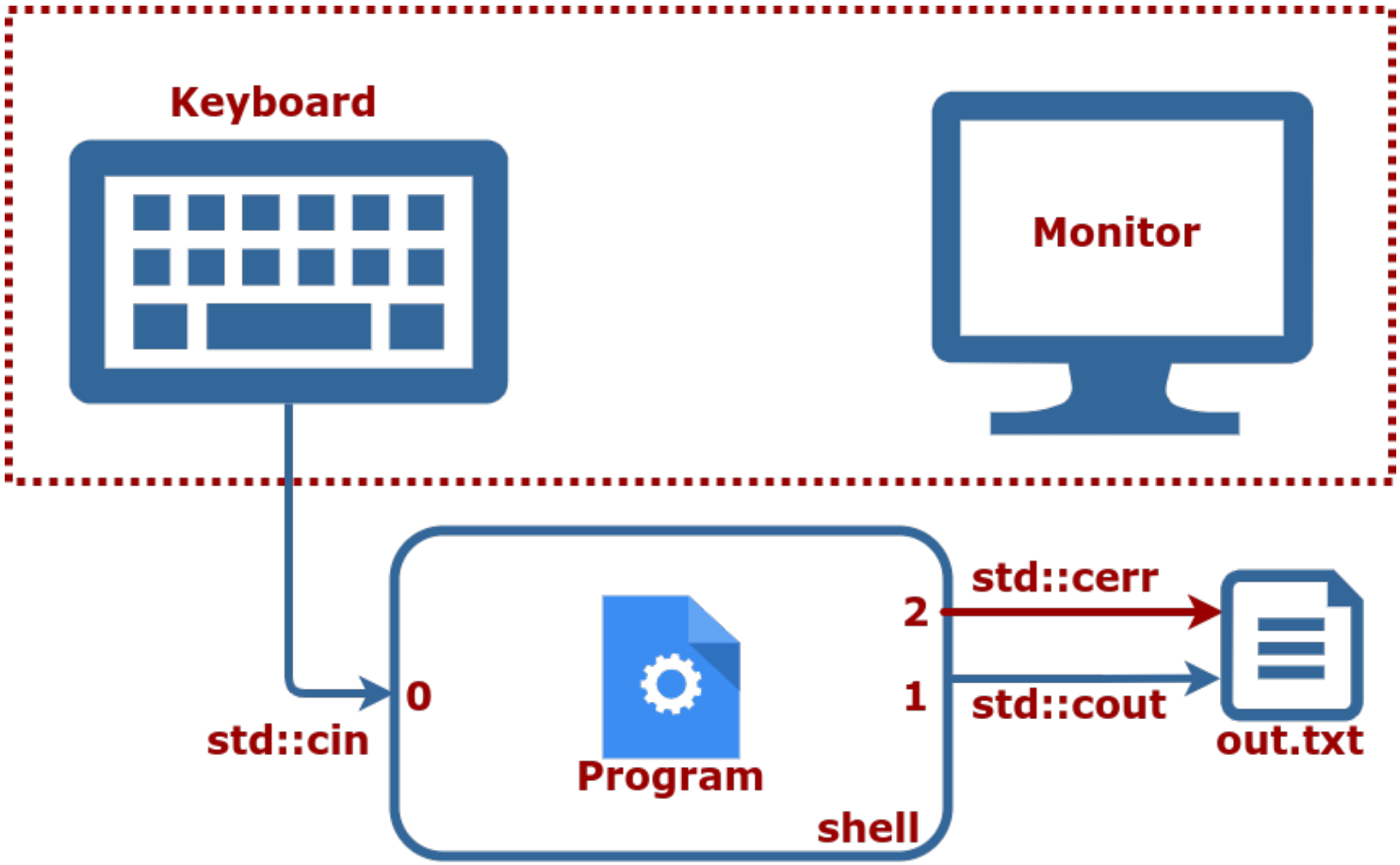
Redirect stdout and stderr

```
$ program 1>stdout.txt 2>stderr.txt
```



Redirect `stdout` and `stderr`

```
progamm 1>out.txt 2>&1
```



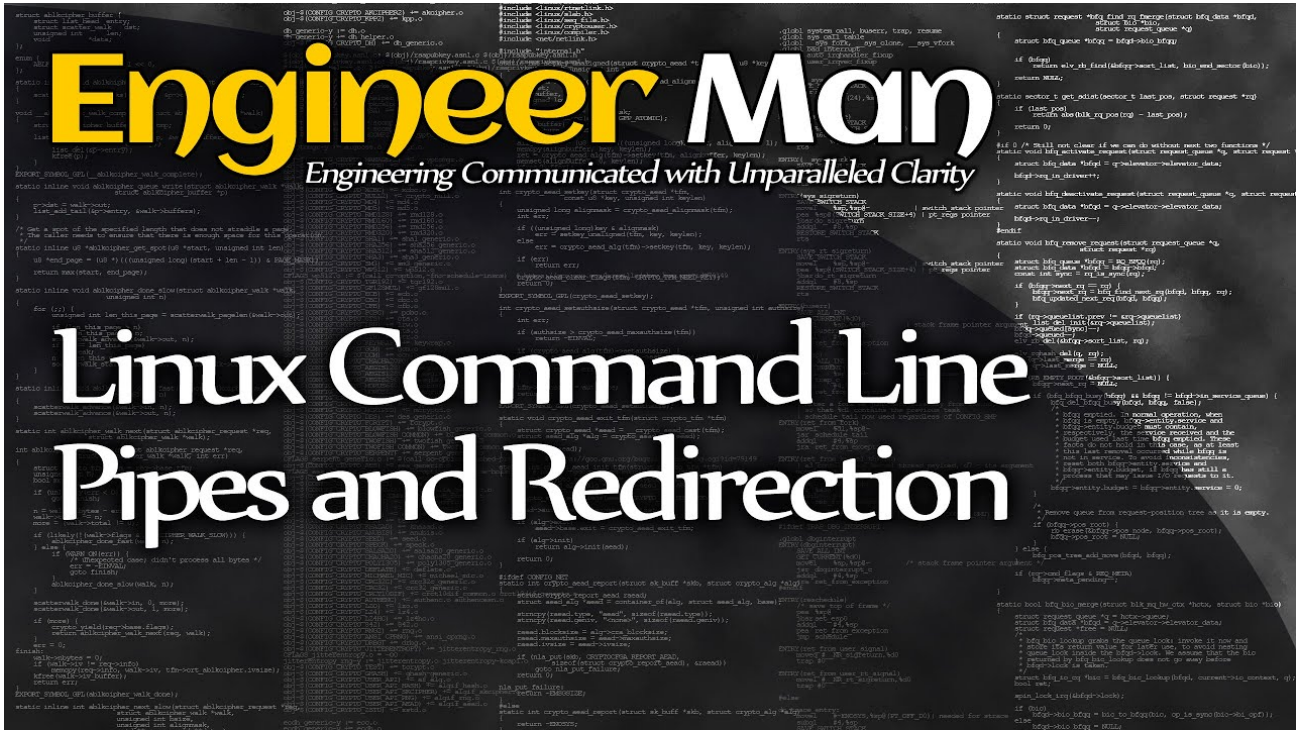
Working with files

- `more/less/cat <filename>`
  Print the contents of the file
  Most of the time using `cat` if enough
- `find <in-folder> -name <filename>`
  Search for file `<filename>` in folder `<in-folder>`, allows wildcards
- `locate <filename>`
  Search for file `<filename>` in the entire system!
  just remember to `sudo updatedb` often
- `grep <what> <where>`
  Search for a string `<what>` in a file `<where>`
- `ag <what> <where>`
  Search for a string `<what>` in a dir `<where>`

Chaining commands

- `command1; command2; command3`
  Calls commands one after another
- `command1 && command2 && command3`
  Same as above but fails if any of the commands returns an error code
- `command1 | command2 | command3`
  Pipe `stdout` of `command1` to `stdin` of `command2` and `stdout` of `command2` to `stdin` of `command3`
- Piping commonly used with `grep`:
  `ls | grep smth` look for `smth` in output of `ls`

Linux Command Line Pipes and Redirection



https://youtu.be/mV_8GbzwZMM

Canceling commands

- `CTRL + C`
  Cancel currently running command
- `kill -9 <pid>`
  Kill the process with id `pid`
- `killall <pname>`
  Kill all processes with name `pname`
- `htop` (`top`)
  - Shows an overview of running processes
  - Allows to kill processes by pressing k

Command history

The shell saves the history of the last executed commands

- ↑: go to the previous command
- ↓: go to the next command
- Ctrl + R `<query>`: search in history
- ! + 10: execute the 10th command
- `history`: show history

Installing software

Most of the software is available in the system repository. To install a program in Ubuntu type this into terminal:

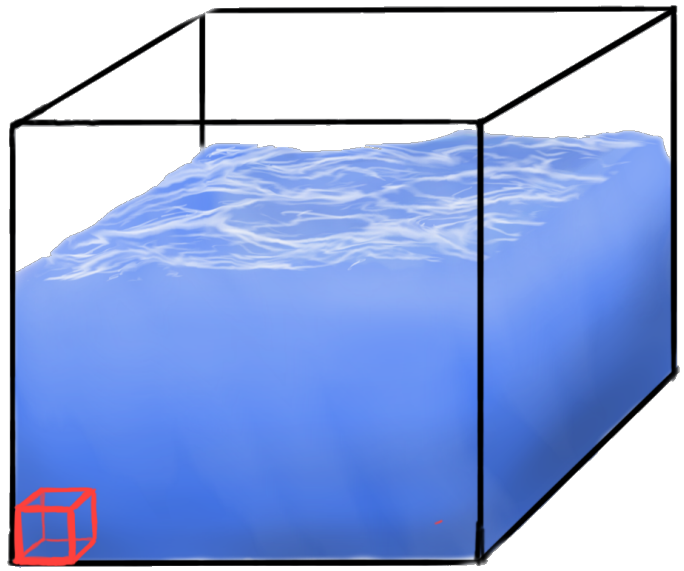
- `sudo apt update` to update information about available packages
- `sudo apt install <program>` to install the program that you want
- Use `apt search <program>` to find all packages that provide `<program>`
- Same for any library, just with `lib` prefix

Bash tutorial

https://youtu.be/oxuRxtrO2Ag

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++:

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

```
1 int main()
2
3 {          return 0;
4 }
```

```
1 int main() {
2   return /* Ignored comment */ 0;
3 }
```

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 }
```

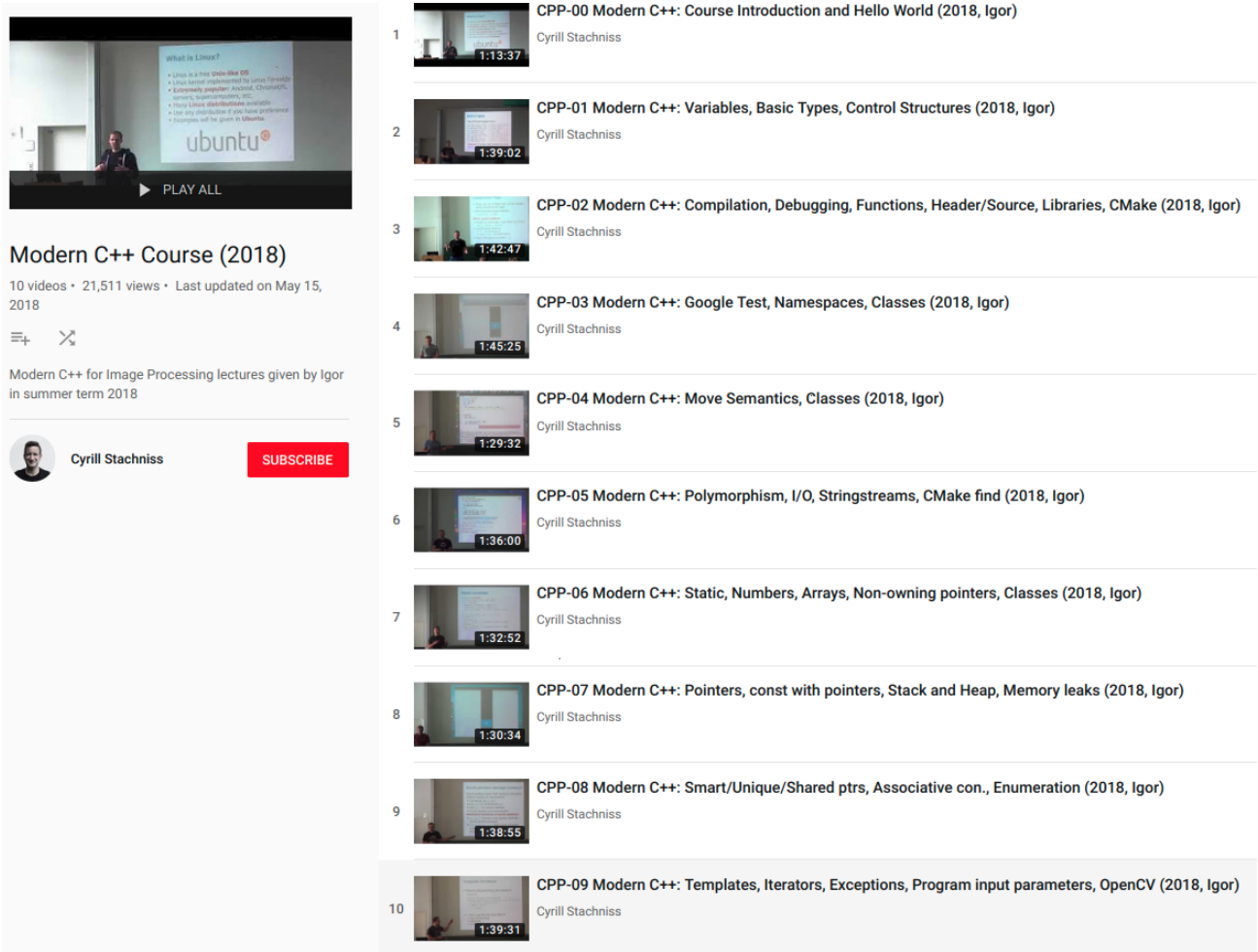
Compile and run Hello World!

- 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
```

Credits to Igor the great



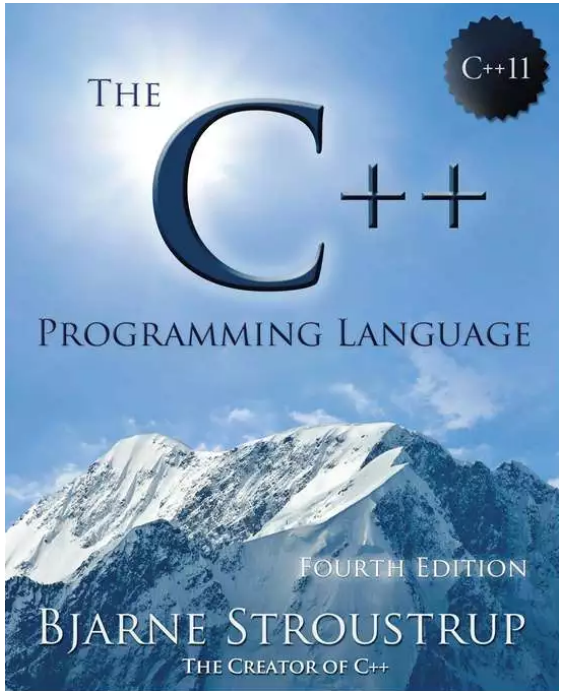
https://bit.ly/2JmIqGs [shortened]

Suggested Video

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

https://youtu.be/xfBWk4nw440

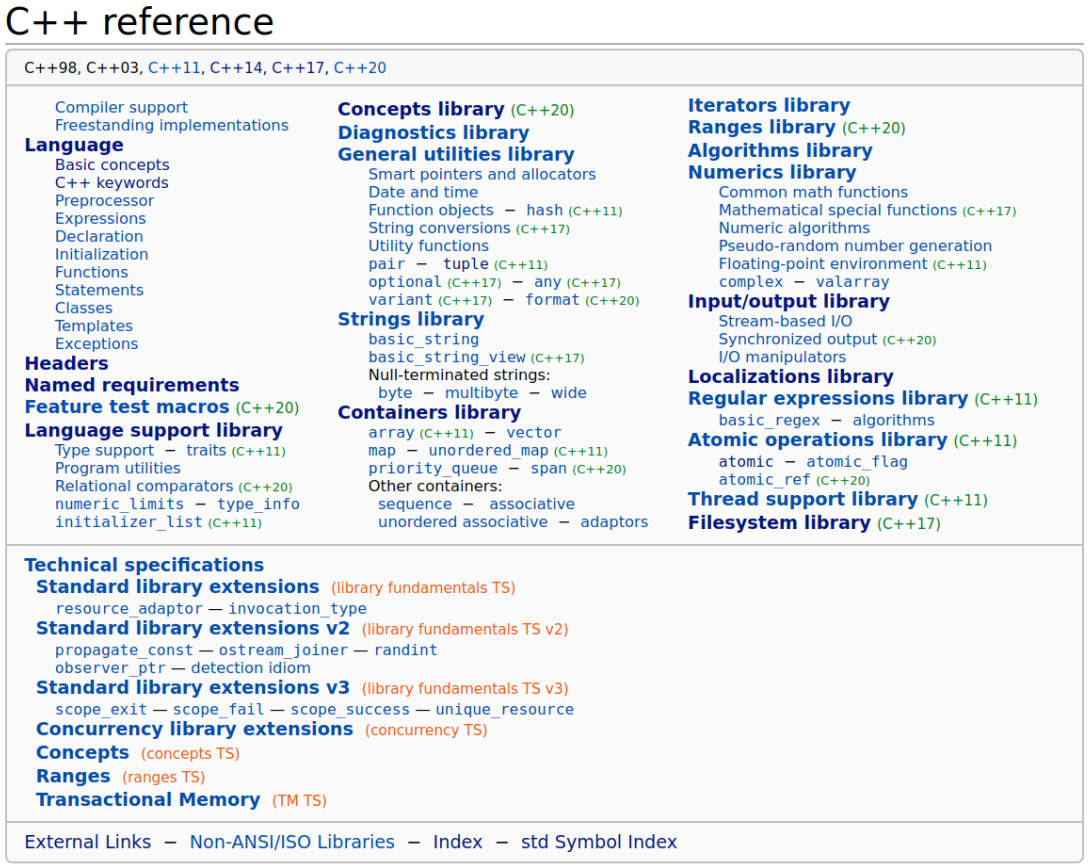
C++ Programming Language



- **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/