

Modern C++ for Computer Vision and Image Processing

Lecture 0: The basics

Ignacio Vizzo and Cyrill Stachniss

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				
12	8-Jul	[[No Lectures]]	Final Project	Final Examination Date	
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
- RAI
- Smart pointers
- Generic programming
- Template functions
- Template classes
- Static code generation
- lambdas

Course Philosophy



**Talk is cheap.
Show me the code.**

Linus Torvalds

What you will do in this course



Please stop me!



Why?

Why C++? Why Linux? Why?



Developer Survey Results 2018

- 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++

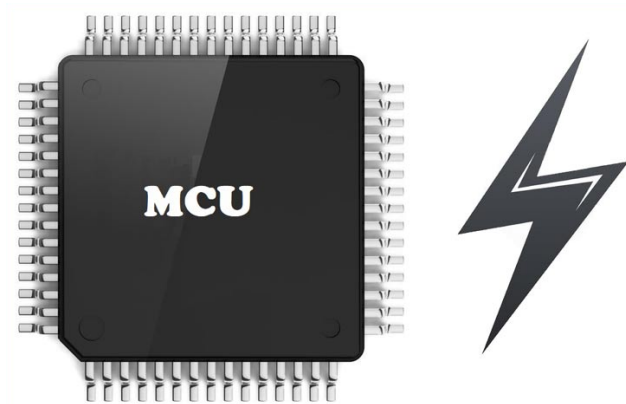


Image taken from <https://circuitdigest.com/>

Companies that use C++



Microsoft



amazon.com[®]



facebook[®]

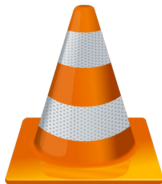
The following slides are adapted from Avery Wang

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

Browsers written in C++



Software written in C++



Games written in C++



CALL^{OF}DUTY.

**MASS
EFFECT**



HALO

C++ History: assembly

Benefits:

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

Why don't we always use assembly?

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:

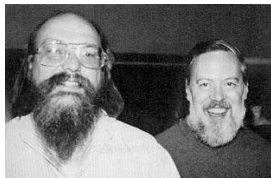
- Source code can be written in a more intuitive language
- An additional program can convert it into assembly [`compiler`]

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

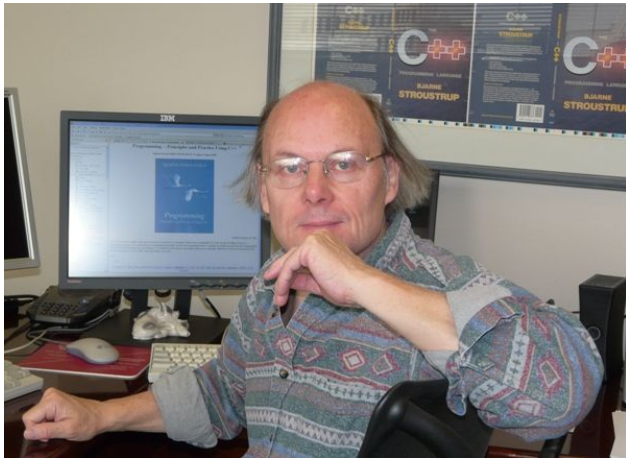
C was popular since it was simple.

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++

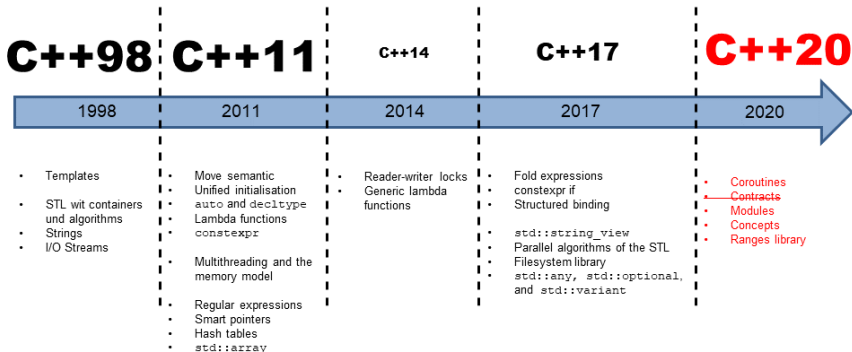
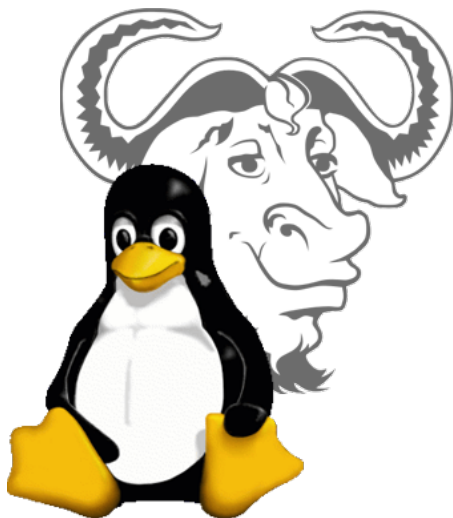


Image taken from <https://www.modernescpp.com/>

Design Philosophy of C++

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

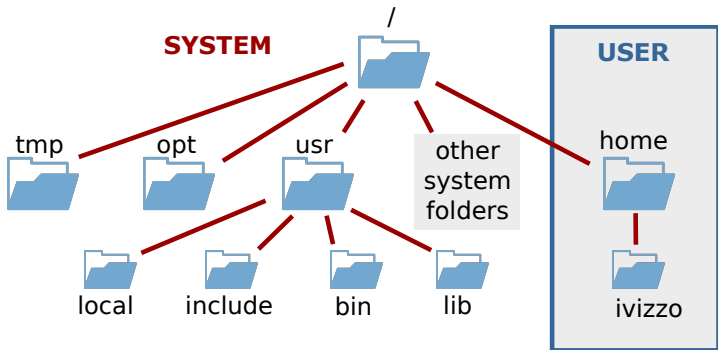


What is GNU/Linux?

- 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**

ubuntu 

Linux directory tree



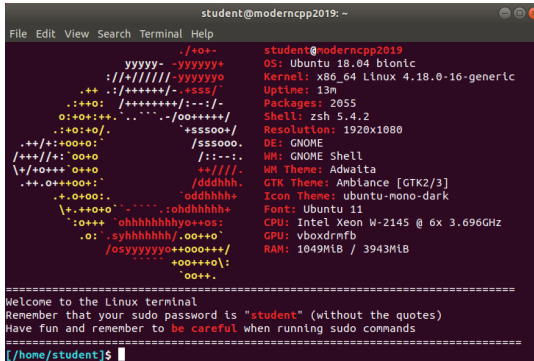
- Tree organization starting with root: `/`
- There are no volume letters, e.g. `C:`, `D:`
- User can only access his/her own folder

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



```
student@moderncpp2019: ~  
File Edit View Search Terminal Help  
      ./.+o+-  
      yyyyy- -yyyyyy+  
      ://+///// -yyyyyyo  
      .++ ://+++++/- .+sss/  
      .:++o: //+++++/:--:/-  
      o:+o+:++ .:~`.-/oo++++/  
      .:+:+o:/ . `+sssoo+/  
      .++/+:+oo+o: ` /sssooo.  
      /+++//+:`oo+o :/:--:..  
      \+/+o+++`o+o+ ++////.  
      .++.o+++oo+:` /dddhhh.  
      .+.o+oo: . `oddhhhh+  
      \+.++o+o``-` .:ohdhhhh+  
      :o+++ `ohhhhhhhhyo++os:  
      .o: `syhhhhhhh/.oo++o  
      /osyyyyyyo++ooo++/  
      .:++o++o\:  
      `oo++.  
=====
```

```
student@moderncpp2019  
OS: Ubuntu 18.04 bionic  
Kernel: x86_64 Linux 4.18.0-16-generic  
Uptime: 13m  
Packages: 2055  
Shell: zsh 5.4.2  
Resolution: 1920x1080  
DE: GNOME  
WM: GNOME Shell  
WM Theme: Adwaita  
GTK Theme: Ambiance [GTK2/3]  
Icon Theme: ubuntu-mono-dark  
Font: Ubuntu 11  
CPU: Intel Xeon W-2145 @ 6x 3.696GHz  
GPU: vboxdrmfb  
RAM: 1049MiB / 3943MiB  
=====
```

```
Welcome to the Linux terminal  
Remember that your sudo password is "student" (without the quotes)  
Have fun and remember to be careful when running sudo commands  
=====
```

```
[/home/student]$
```

- 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**ist 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`: absolute 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

```
1 [/home/student]$ cat --help
2 Usage: cat [OPTION]... [FILE]...
3 Concatenate FILE(s) to standard output.
4   -A, --show-all           equivalent to -vET
5   -b, --number-nonblank     number nonempty output lines
6
7 Examples:
8   cat f -   Output fs contents, then standard input.
9   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:

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

Files and folders

- **mkdir** [-p] <foldername> — **make directory**
Create a folder <foldername>
(with all parent folders [-p])
- **rm** [-r] <name> — **remove** [recursive]
Remove file or folder <name>
(With folder contents [-r])
- **cp** [-r] <source> <dest> — **copy**
Copy file or folder from <source> to <dest>
- **mv** <source> <dest> — **move**
Move file or folder from <source> to <dest>

Using placeholders

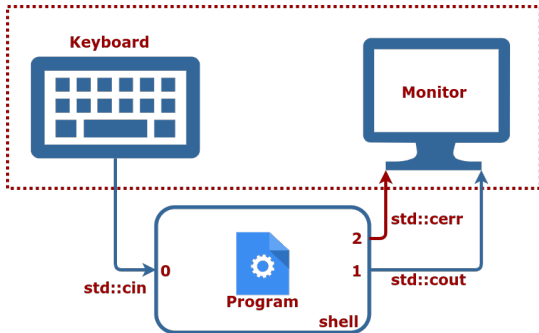
Placeholder	Meaning
*	Any set of characters
?	Any single character
[a-f]	Characters in [abcdef]
[^a-c]	Any character not in [abc]

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

```
1  [/home/student/Examples/placeholders]$ ls
2  u01.tex      v01.pdf      v01.tex
3  u02.tex      v02.pdf      v02.tex
4  u03.tex      v03.pdf      v03.tex
5
6  [/home/student/Examples/placeholders]$ ls *.pdf
7  v01.pdf      v02.pdf      v03.pdf
8
9  [/home/student/Examples/placeholders]$ ls u*
10 u01.tex      u02.tex      u03.tex
11
12 [/home/student/Examples/placeholders]$ ls ?01*
13 u01.tex      v01.pdf      v01.tex
14
15 [/home/student/Examples/placeholders]$ ls [uv]01*
16 u01.tex      v01.pdf      v01.tex
17
18 [/home/student/Examples/placeholders]$ ls u0[~12].tex
19 u03.tex
```

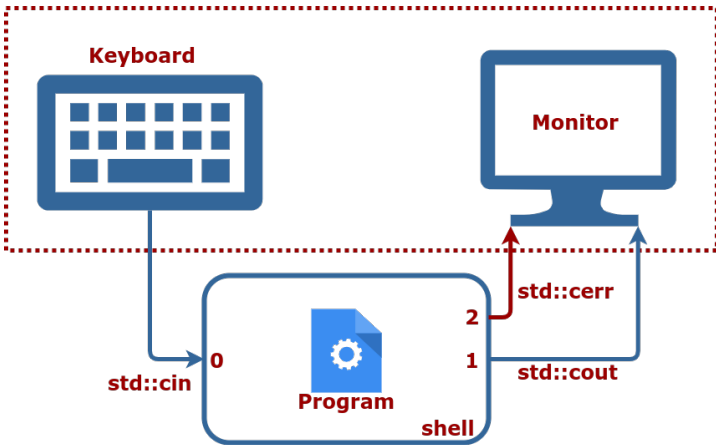
Standard input/output channels

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



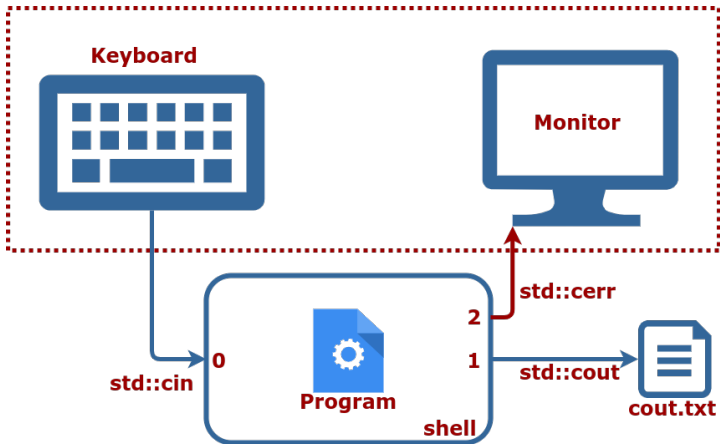
Standard input/output channels

\$ program



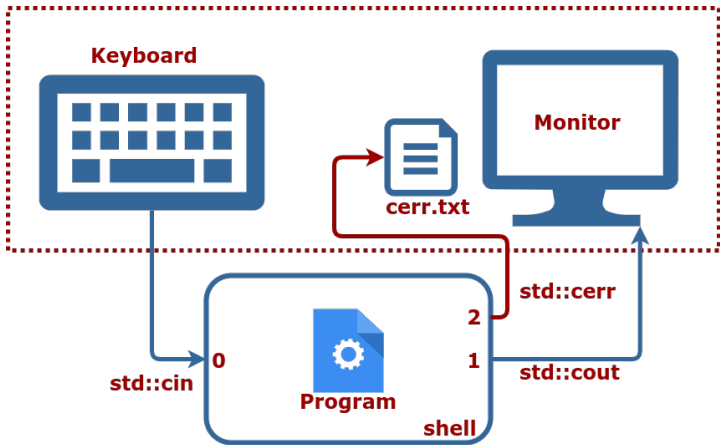
Redirecting `stdout`

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



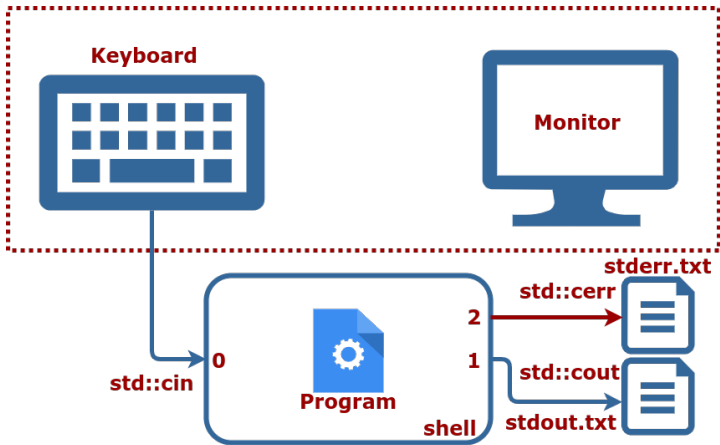
Redirecting `stderr`

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



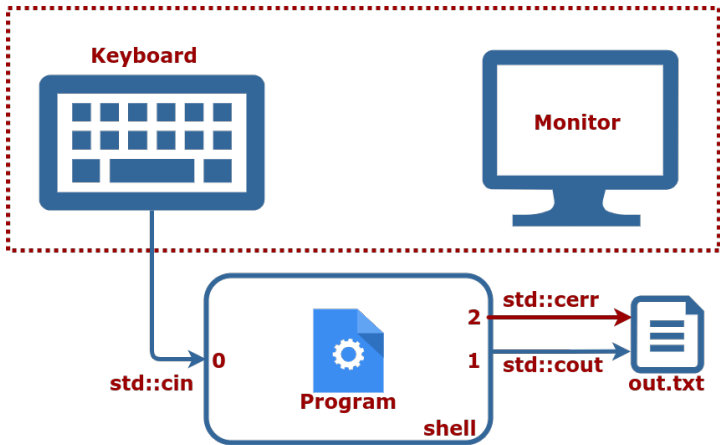
Redirect `stdout` and `stderr`

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



Redirect `stdout` and `stderr`

```
programm 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_8GbwZMM

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 

Command history

The shell saves the history of the last executed commands

- : go to the previous command
- : go to the next command
-  +  <query>: search in history
-  + : 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

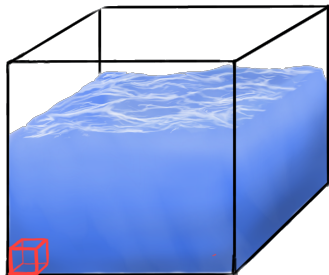
Bash tutorial



<https://youtu.be/oxuRxtrO2Ag>



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]

Hello World!

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

```
1 #include <iostream>
2
3 int main() {
4     // Is this your first C++ program?
5     std::cout << "Hello World!" << std::endl;
6     return 0;
7 }
```


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

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





Modern C++ Course (2018)


10 videos • 21,511 views • Last updated on May 15, 2018


Modern C++ for Image Processing lectures given by Igor in summer term 2018


 Cyril Stachniss [SUBSCRIBE](#)


1  **CPP-00 Modern C++: Course Introduction and Hello World (2018, Igor)**
Cyril Stachniss


2  **CPP-01 Modern C++: Variables, Basic Types, Control Structures (2018, Igor)**
Cyril Stachniss


3  **CPP-02 Modern C++: Compilation, Debugging, Functions, Header/Source, Libraries, CMake (2018, Igor)**
Cyril Stachniss


4  **CPP-03 Modern C++: Google Test, Namespaces, Classes (2018, Igor)**
Cyril Stachniss


5  **CPP-04 Modern C++: Move Semantics, Classes (2018, Igor)**
Cyril Stachniss

6  **CPP-05 Modern C++: Polymorphism, I/O, Stringstreams, CMake find (2018, Igor)**
Cyril Stachniss

7  **CPP-06 Modern C++: Static, Numbers, Arrays, Non-owning pointers, Classes (2018, Igor)**
Cyril Stachniss

8  **CPP-07 Modern C++: Pointers, const with pointers, Stack and Heap, Memory leaks (2018, Igor)**
Cyril Stachniss

9  **CPP-08 Modern C++: Smart/Unique/Shared ptrs, Associative con., Enumeration (2018, Igor)**
Cyril Stachniss

10  **CPP-09 Modern C++: Templates, Iterators, Exceptions, Program input parameters, OpenCV (2018, Igor)**
Cyril Stachniss

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

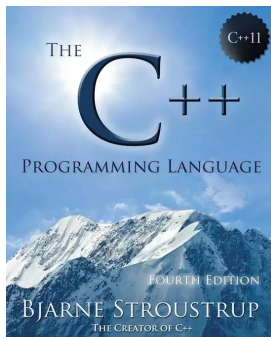
Suggested Video

"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

C++ reference

C++98, C++03, C++11, C++14, C++17, C++20

Compiler support

Freestanding implementations

Language

Basic concepts

C++ keywords

Preprocessor

Expressions

Declaration

Initialization

Functions

Statements

Classes

Templates

Exceptions

Headers

Named requirements

Feature test macros (C++20)

Language support library

Type support — traits (C++11)

Program utilities

Relational comparators (C++20)

numeric_limits — type_info

initializer_list (C++11)

Concepts library (C++20)

Diagnostics library

General utilities library

Smart pointers and allocators

Date and time

Function objects — hash (C++11)

String conversions (C++17)

Utility functions

pair — tuple (C++11)

optional (C++17) — any (C++17)

variant (C++17) — format (C++20)

Strings library

basic_string

basic_string_view (C++17)

Null-terminated strings:

byte — multibyte — wide

Containers library

array (C++11) — vector

map — unordered_map (C++11)

priority_queue — span (C++20)

Other containers:

sequence — associative

unordered associative — adaptors

Iterators library

Ranges library (C++20)

Algorithms library

Numerics library

Common math functions

Mathematical special functions (C++17)

Numeric algorithms

Pseudo-random number generation

Floating-point environment (C++11)

complex — valarray

Input/output library

Stream-based I/O

Synchronized output (C++20)

I/O manipulators

Localizations library

Regular expressions library (C++11)

basic_regex — algorithms

Atomic operations library (C++11)

atomic — atomic_flag

atomic_ref (C++20)

Thread support library (C++11)

Filesystem library (C++17)

Technical specifications

Standard library extensions (library fundamentals TS)

resource adaptor — invocation_type

Standard library extensions v2 (library fundamentals TS v2)

propagate_const — ostream_joiner — randint

observer_ptr — detection idiom

Standard library extensions v3 (library fundamentals TS v3)

scope_exit — scope_fail — scope_success — unique_resource

Concurrency library extensions (concurrency TS)

Concepts (concepts TS)

Ranges (ranges TS)

Transactional Memory (TM TS)

External Links — Non-ANSI/ISO Libraries — Index — std Symbol Index

<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/>