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:

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

<table>
<thead>
<tr>
<th>Week</th>
<th>Date</th>
<th>Lecture</th>
<th>Homework</th>
<th>Recommended Deadline</th>
<th>Official Deadline</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>8-Apr</td>
<td>[[ No Lectures ]]</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>0</td>
<td>15-Apr</td>
<td>Course Introduction, Organization, Hello world</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>1</td>
<td>22-Apr</td>
<td>C++ Tools</td>
<td>Homework 1</td>
<td>3-May</td>
<td>10-May</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Part II: The C++ core language</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>29-Apr</td>
<td>C++ Basic syntax</td>
<td>Homework 2</td>
<td>10-May</td>
<td>17-May</td>
</tr>
<tr>
<td>3</td>
<td>6-May</td>
<td>C++ Functions</td>
<td>Homework 3</td>
<td>17-May</td>
<td>24-May</td>
</tr>
<tr>
<td>4</td>
<td>13-May</td>
<td>C++ Containers</td>
<td>Homework 4</td>
<td>24-May</td>
<td>31-May</td>
</tr>
<tr>
<td>5</td>
<td>20-May</td>
<td>C++ STL Library</td>
<td>Homework 5</td>
<td>31-May</td>
<td>7-Jun</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Part III: Modern C++</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>27-May</td>
<td>Classes</td>
<td>Homework 6</td>
<td>7-Jun</td>
<td>14-Jun</td>
</tr>
<tr>
<td>7</td>
<td>3-Jun</td>
<td>OOP</td>
<td>Homework 7</td>
<td>14-Jun</td>
<td>21-Jun</td>
</tr>
<tr>
<td>8</td>
<td>10-Jun</td>
<td>Memory Management</td>
<td>Homework 8</td>
<td>21-Jun</td>
<td>28-Jun</td>
</tr>
<tr>
<td>9</td>
<td>17-Jun</td>
<td>Generics Programing</td>
<td>Homework 9</td>
<td>28-Jun</td>
<td>5-Jul</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Part IV: Final Project &quot;Place recognition using Bag of Visual Words in C++&quot;</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>24-Jun</td>
<td>Bag of Visual Words</td>
<td>Final Project</td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>1-Jul</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>8-Jul</td>
<td>[[ No Lectures ]]</td>
<td>Final Project</td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>15-Jul</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
# 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
- Typing
- 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?

- 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

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++

Google  Microsoft  Apple

Intel  IBM  Amazon.com

Adobe  Facebook

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

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> >&std::__ostream_insert<char, std::char_traits<char> >, char const*, long)
    xor    eax, eax
    pop    rcx
    ret

_GLOBAL__sub_I_example.cpp:
    # @_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
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++

- 1998: Templates, STL, wt containers, and types, Strings, I/O Streams
- 2011: Move semantic, Unified initialisation, auto and decaltype, Lambda functions, constexpr
- 2014: Multithreading and the memory model, Regular expressions, Smart pointers, Hash tables, std::array
- 2017: Reader-writer locks, Generic lambda functions
- 2020: Fold expressions, constexpr if, Structured binding, std::string_view, Parallel algorithms of the STL, Filesystem library, std::any, std::optional, and std::variant

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**
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`: print working directory
- `cd <dir>`: change directory to `<dir>`
- `ls <dir>`: list 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>` — manual
  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.
   - `--show-all` equivalent to `-vET`
   - `--number-nonblank` number nonempty output lines

Examples:
4. `cat f -` Output fs contents, then standard input.
5. `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  $ 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

<table>
<thead>
<tr>
<th>Placeholder</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>*</td>
<td>Any set of characters</td>
</tr>
<tr>
<td>?</td>
<td>Any single character</td>
</tr>
<tr>
<td>[a-f]</td>
<td>Characters in [abcdef]</td>
</tr>
<tr>
<td>[^a-c]</td>
<td>Any character not in [abc]</td>
</tr>
</tbody>
</table>

Can be used with most of terminal commands: `ls`, `rm`, `mv` etc.
```bash
[/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: Standard input: channel 0

- Two output channels:
  - stdout: Standard output: channel 1
  - stderr: Standard error output: channel 2
Standard input/output channels

$ program
Redirecting `stdout`

```bash
$ 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**

```bash
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

- \( \uparrow \): go to the previous command
- \( \downarrow \): go to the next command
- \( \text{Ctrl} + \text{R} \) <query>: search in history
- \( ! + 10 \): execute the 10th command
- \texttt{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]

Most icons are from Paper Icon Set: [https://snwh.org/paper](https://snwh.org/paper)
Hello World!

Simple C++ program that prints Hello World!

```
#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++:

```cpp
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]**

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

```c
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

```cpp
#include <iostream>

int main() {
    int some_number;
    std::cout << "please input any number" << std::endl;
    std::cin >> some_number;
    std::cout << "number = " << some_number << std::endl;
    std::cerr << "boring error message" << std::endl;
    return 0;
}
```
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:

```bash
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

## C++ reference

### Concepts library (C++20)

- Diagnostic 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_flag (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/