2 – Introduction to Linux and Software Development

Robotics and Computer Vision (BPC-PRP)

Course supervisor:

Ing. Adam Ligocki, Ph.D.

Slides:

Ing. Tomáš Horeličan

Ing. Jakub Minařík
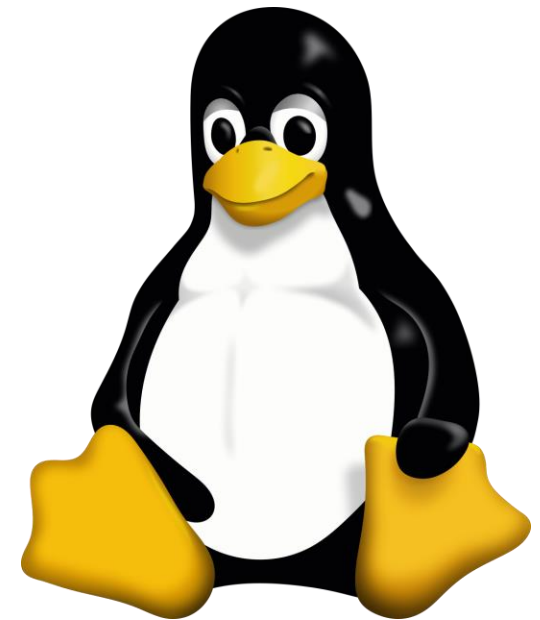
Brno University of Technology
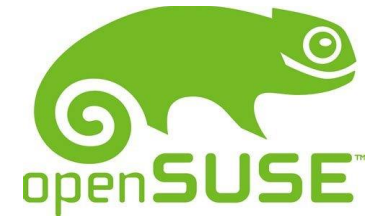
2024

Robotics and AI

- Open source and free

- Security, stability and reliability

- Flexibility and customizability

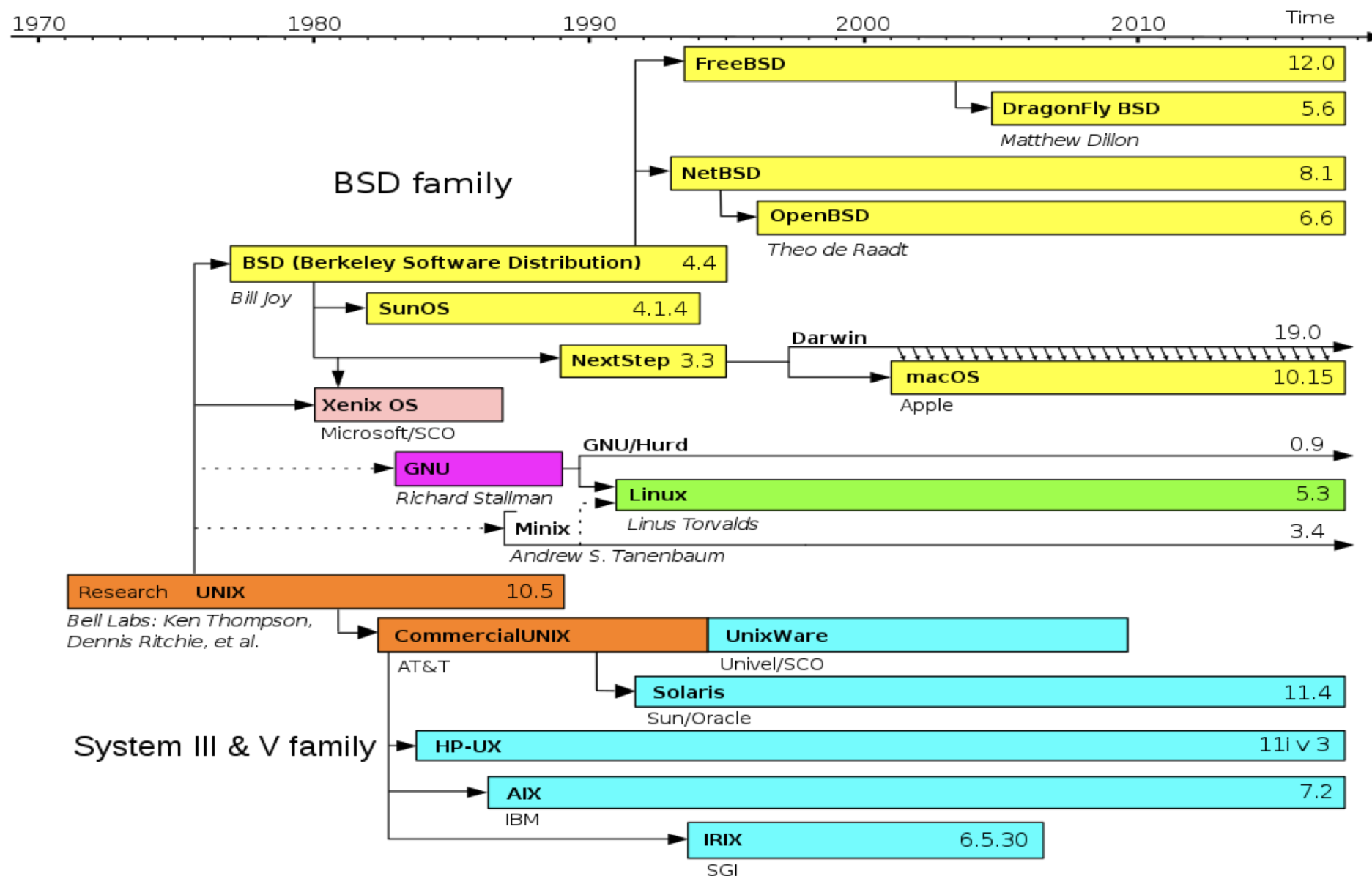- Web servers, supercomputers, routers, cars, smart devices etc.

What is linux?

- Linus Torvalds (Linux), Richard Stallman (GNU)

- OS kernel – GNU/Linux

- Multiple distributions, versions, variants...

Concepts

"Everything is a file (descriptor)" -> all I/O operations handled as simple streams of bytes.

Using common tools to operate on different things -> a single read() function instead of "15" different versions.

File mode flags:
- -   Regular files
- *d*   Directory
- *c*   Character device file (peripheral devices, busses, etc.)
- *b*   Block device file (usually hard disks)
- *s*   Local socket file (inter-processes communication)
- *p*   Named pipe (similar to local socket)
- *l*   Symbolic link (links to files)

File permission flags:
- No permission: -   (0)
- Can be read: r   (4)
- Can be modified: w   (2)
- Can be executed: x   (1)

user|group|others

$\overbrace{\text{rwx}}\ \overbrace{\text{rw-}}\ \overbrace{\text{r-x}}$ ($\underbrace{421}_{7}$, $\underbrace{420}_{6}$, $\underbrace{401}_{5}$)

Special flags:
- The setuid (user) or setgid (group) bit, not found in others (implies that x is set):   s
- Same as s, but x is not set (rare on regular files, useless on directories):   S
- The sticky bit, found only in others (implies that x is set):   t
- Same as t, but x is not set (rare on regular files):   T

```
user@user-MS-7A59:~$ ls -lahF /dev/
total 4.0K
drwxr-xr-x  20 root root          5.9K Feb 12 19:10 ./
drwxr-xr-x  27 root root          4.0K Aug 23 20:20 ../
crw-r--r--   1 root root      10, 235 Feb 12 18:51 autofs
drwxr-xr-x   2 root root          1.5K Feb 12 19:10 block/
crw-------   1 root root      10, 234 Feb 12 18:51 btrfs-control
drwxr-xr-x   3 root root            60 Feb 12 18:51 bus/
lrwxrwxrwx   1 root root             3 Feb 12 18:51 cdrom -> sr0
lrwxrwxrwx   1 root root             3 Feb 12 18:51 cdrw -> sr0
drwxr-xr-x   2 root root          5.3K Feb 12 19:10 char/
crw-------   1 root root       5,   1 Feb 12 18:51 console
lrwxrwxrwx   1 root root            11 Feb 12 18:51 core -> /proc/kcore
drwxr-xr-x   6 root root           140 Feb 12 18:51 cpu/
crw-------   1 root root      10,  60 Feb 12 18:51 cpu_dma_latency
crw-------   1 root root      10, 203 Feb 12 18:51 cuse
drwxr-xr-x   8 root root           160 Feb 12 18:51 disk/
drwxr-xr-x   3 root root           140 Feb 12 18:51 dri/
lrwxrwxrwx   1 root root             3 Feb 12 18:51 dvd -> sr0
lrwxrwxrwx   1 root root             3 Feb 12 18:51 dvdrw -> sr0
crw-------   1 root root      89,   0 Feb 12 18:51 i2c-0
crw-------   1 root root     239,   0 Feb 12 18:51 nvme0
brw-rw----   1 root disk    259,   0 Feb 12 18:51 nvme0n1
brw-rw----   1 root disk    259,   1 Feb 12 18:51 nvme0n1p1
brw-rw----   1 root disk      8,   0 Feb 12 18:51 sda
lrwxrwxrwx   1 root root            15 Feb 12 18:51 stderr -> /proc/self/fd/2
lrwxrwxrwx   1 root root            15 Feb 12 18:51 stdin -> /proc/self/fd/0
lrwxrwxrwx   1 root root            15 Feb 12 18:51 stdout -> /proc/self/fd/1
crw-rw-rw-   1 root tty      5,   0 Feb 12 18:51 tty
```

The beginning of the file system always starts with the "*root*" directory, marked with a backslash. → /
- Not to be confused with the root user directory. → /root

You can think of it as C: in Windows (if you're using just one physical drive).
- Linux doesn't have drive letters, all additional drives are located (mounted) in /media.
- You can also mount any directory to a specific drive partition.

```
user@user-MS-7A59:~$ tree -L 1 /
/
├── bin
├── boot
├── cdrom
├── dev
├── etc
├── home
├── initrd.img -> boot/initrd.img-4.15.0-140-generic
├── initrd.img.old -> boot/initrd.img-4.15.0-128-generic
├── lib
├── lib32
├── lib64
├── lost+found
├── media
├── opt
├── proc
├── root
├── run
├── sbin
├── snap
├── srv
├── swapfile
├── sys
├── tmp
├── usr
├── var
├── vmlinuz -> boot/vmlinuz-4.15.0-140-generic
└── vmlinuz.old -> boot/vmlinuz-4.15.0-128-generic

22 directories, 5 files
```

- /home -> all users' personal directories
- /root -> home directory for the root user (admin)
- /dev –> files (the byte streams) for physical devices (TTYs, USBs, keyboards, mice, drives, joysticks, ...)
- /media –> automatic mount point for new drives (the contents)
- /tmp -> temporary data (deleted after reboot)
- /etc –> all system configuration files
- /bin, /sbin -> executable binaries and scripts (s stands for sudo)
- /boot -> all files required for system boot (DO NOT MESS WITH THIS!)
- /lib -> library files (similar to *.dll files in windows)
- /opt -> some software gets installed here
- /proc, /sys -> hardware-specific files (similar to /dev)
- /usr, /var, …

TTY (TeleTYpewriter) → used for interacting with the OS.

- Early days: electromechanical device (prints on paper).
- Later: video terminal (prints on screen).
- Modern: terminal emulator/pseudo terminal (prints on a GUI window).

A command line interface for controlling Linux (just like the *PowerShell* (or *cmd*) window in Windows).

Opening a screen terminal: **CTRL + ALT + FX**, where **X** is a number <3,7>
- **F1** – login screen, **F2** – desktop environment
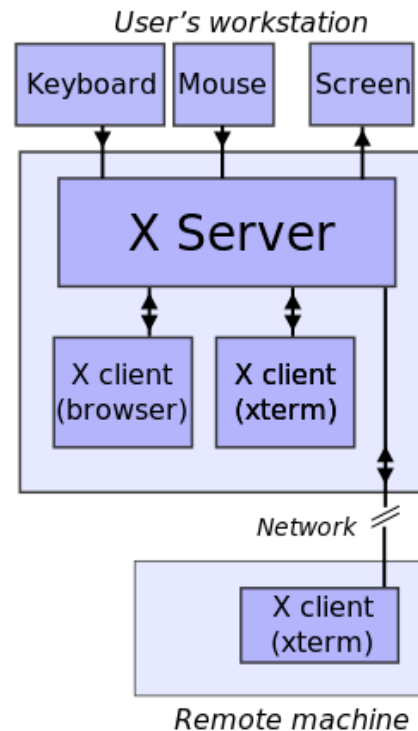
Opening a terminal emulator: **CTRL + ALT + T**

```
/dev/tty0
/dev/ttyS0
/dev/pts/0
/dev/ttyUSB0
```

Running programs (clients) communicate with X Server to visualize graphical outputs and to interact with the user.

X Server only defines the interface and is extended by window mangers, graphical interfaces (KDE, GNOME, Xfce, …) , and applications.

User's workstation

Keyboard | Mouse | Screen

X Server

X client (browser) | X client (xterm)

Network

X client (xterm)

Remote machine

user

graphical interface

Examples: KDE Plasma, Aqua, GNOME Shell

Examples: X11, Wayland, Quartz

display server | window manager

Examples: awesome, Compiz, OpenBox

Examples: X.Org Server, Weston, KWin, Mutter, Quarz Compositor, SurfaceFlinger

kernel

Examples: Linux kernel, FreeBSD kernel, XNU kernel

hardware

A language used to interact with and control Linux (in Windows this is the *PowerShell*).

Uses the terminal for interactive input and output:
- *shell* (Bourne shell) -> default on all machines
- *bash* (Bourne-Again shell) -> most common default
- *zsh* (Z shell) -> more modern, mostly backwards-compatible with bash

Also used to write scripts that can be executed when necessary.

SHELL Commands

ls -> (list) prints all files in the current working directory

cd -> (change directory) changes the working directory
- cd ~ -> into the home directory
- cd / -> into the root directory
- cd .. -> one level above the current directory
- cd . -> current directory (no change)
- cd some_folder -> into a local sub-directory
- cd /home/username/some_folder -> into a specific directory

pwd → (print working directory) prints the current location

mkdir → (make directory) creates a directory of a given name

cp → (copy) copies a given file to another location
- cp ./folder_a/file.txt ./folder_b/file.txt
- cp -r ./folder_a/ ./folder_b/

mv → (move) moves a file from one location to another (also changes the filename)
- mv ./folder_a/file.txt ./folder_b/file.txt
- mv file.txt log.txt

rm → (remove) removes a given file
- rm not_so_cool_file.txt
- rm -r not_so_cool_directory

chmod → (change mode) changes permissions of the given file
- chmod 444 read_only.txt → r--r--r--
- chmod +x executable_for_all → adds x for all groups
- chmod -x not_executable → removes x from all groups

SHELL Commands

- **ssh** - opens an ssh connection

- **touch** - creates a new empty file

- **ln** - creates a link

- **df** - file system usage

- **kill** - kills the process of given PID

- **pgrep** – finds the process PID from name

- **pkill** – kills the process of given name

- **mc** - midnight commander

- **nano** - the n00b text editor

- **vim (or vi)** - the l33t text editor

- **ping** - tests network connection

- **ps** - display current processes

- **cat** - prints contents of a given file

- **grep** – finds a given pattern in a given file

Each command always has a useful help section or a full manual:

- **ls --help** - prints a relatively short help with available flags and use-cases for the ls command.

- **man ls** - prints a full manual for the ls command.

こ# sudo

This command allows you to run commands with the superuser security privileges.

To use this command, you have to be a member of sudo group.
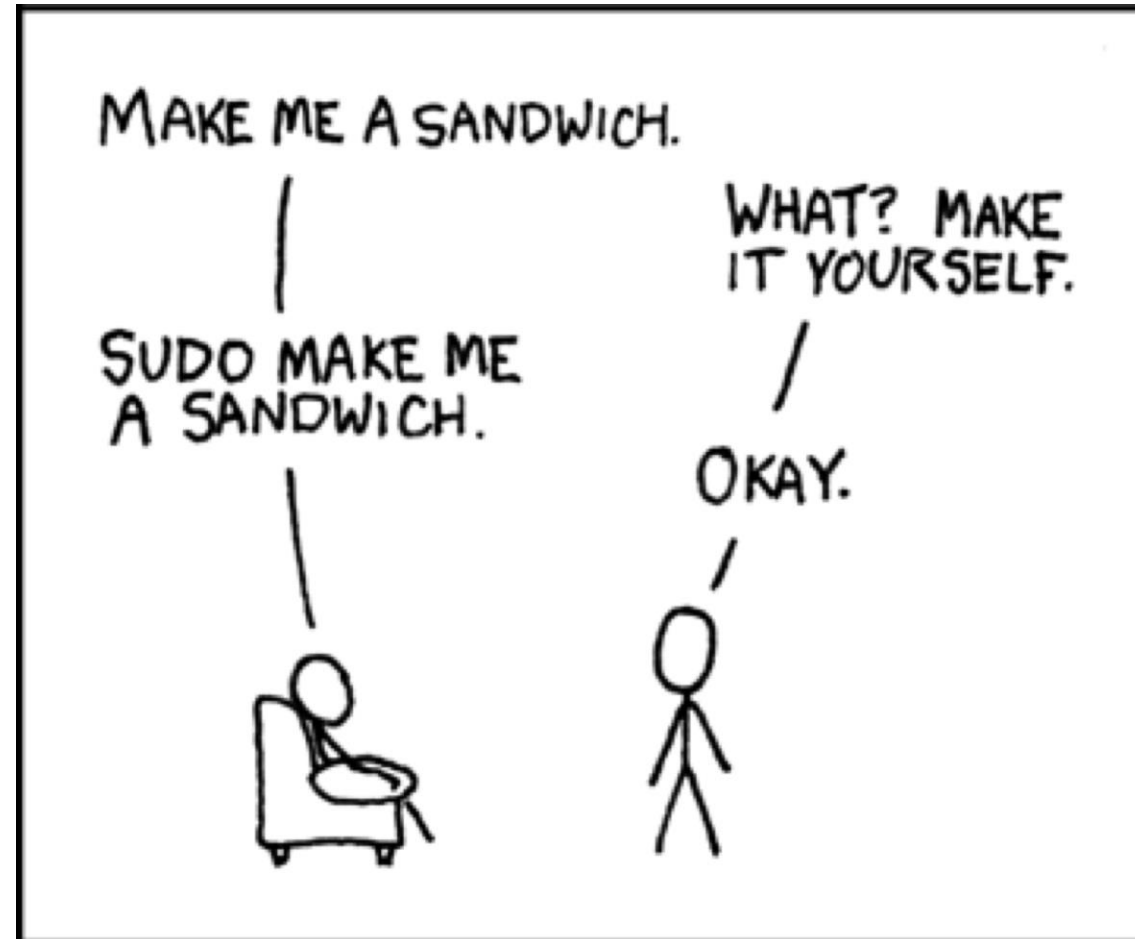
The Unix philosophy:
- Write programs that do one thing and do it well.
- Write programs to work together.
- Write programs to handle text streams, because that is a universal interface.

We can redirect or chain programs like this:
- cat some_file.txt → outputs the contents to screen

- cat some_file.txt > new_file.txt → redirects the output to a different file

- cat some_file.txt | grep hello → pipes the output to grep, grep searches for hello in the output and prints result to screen
  - grep hello some_file.txt → this is better if only searching in files

Package Managers are programs, that download software form online repositories and install them into your OS.

- sudo apt update
- sudo apt upgrade
- sudo apt install cool_package

- sudo snap install cool_package
- sudo flatpak install cool_package

https://packages.ubuntu.com

Remote

| Packages | Metadata |

Repository

Package Manager

OS

Local
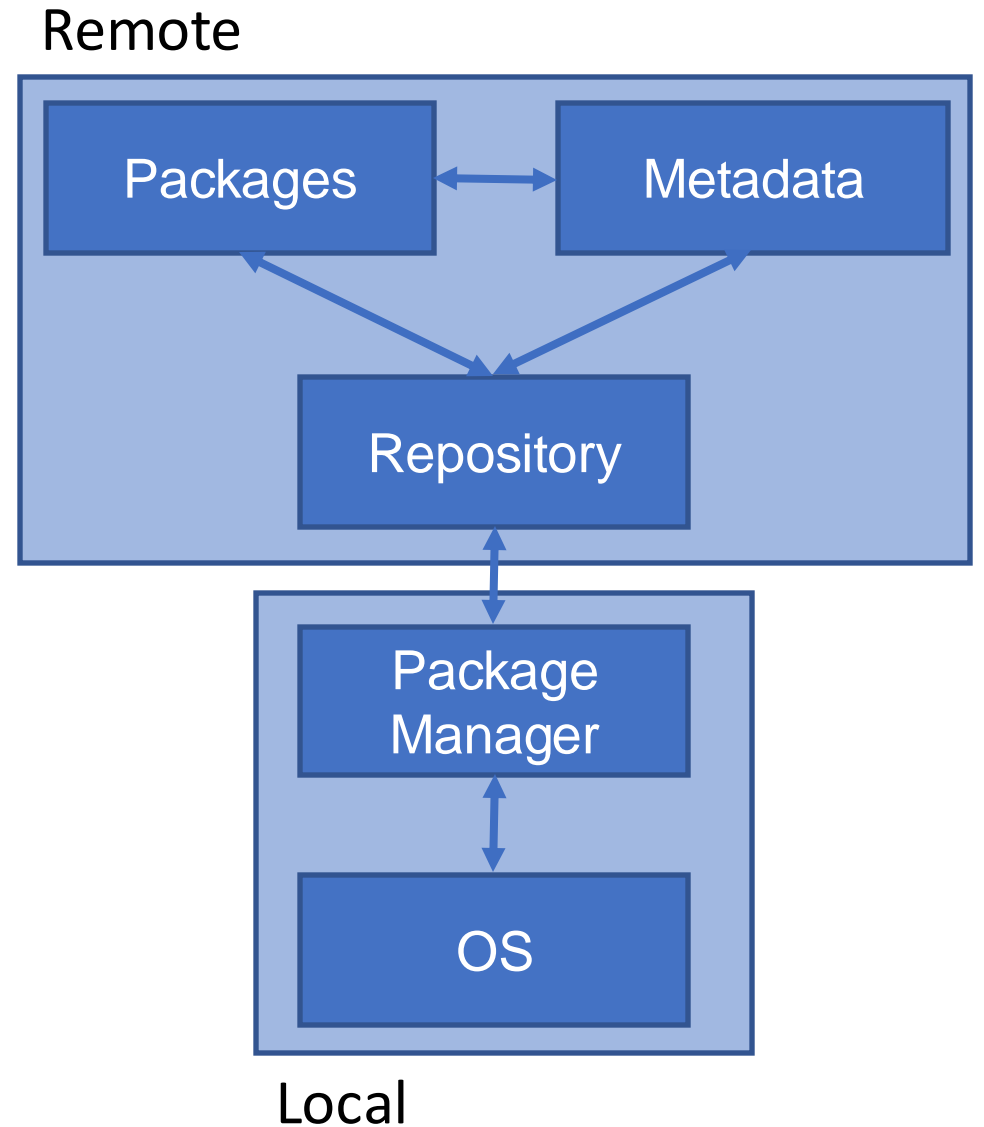
Linux (Ubuntu) installation

- Only one system on PC
- Dual boot with other OS e.g. Windows
  - Different disk
  - On different partition
  - Risk of losing data of both system when instaling (or updating)
- Virtual machine
  - No risk, but could be slower
- WSL2 – Windows subsystem for linux
  - Some restrictions, e.g. I/O

Download an image from the official Ubuntu web page (22.04 LTS)

Create a bootable usb:
- https://www.balena.io/etcher/
- https://unetbootin.github.io
- https://rufus.ie/en/
- *dd* command

Be aware of:
- Dual boot setup
- Swap area
- Deleting existing Windows partitions

If you have Windows version 1903 or higher and your machine supports virtualization, you can install Linux under WSL in Windows:

Run this in *cmd* or *powershell*:
- *wsl –-install -d Ubuntu-22.04*

or install Ubuntu 22.04 from the Windows store:
- https://apps.microsoft.com/store/detail/ubuntu-22041-lts/9PN20MSR04DW?hl=cs-cz&gl=

or install manually from an AppxBundle:
- https://aka.ms/wslubuntu2204

```
PS C:\Windows\system32> wsl --list --verbose
  NAME        STATE           VERSION
* Ubuntu      Running         2
PS C:\Windows\system32> wsl --list --online
The following is a list of valid distributions that can be installed.
Install using 'wsl --install -d <Distro>'.

NAME                FRIENDLY NAME
Ubuntu              Ubuntu
Debian              Debian GNU/Linux
kali-linux          Kali Linux Rolling
openSUSE-42         openSUSE Leap 42
SLES-12             SUSE Linux Enterprise Server v12
Ubuntu-16.04        Ubuntu 16.04 LTS
Ubuntu-18.04        Ubuntu 18.04 LTS
Ubuntu-20.04        Ubuntu 20.04 LTS
```
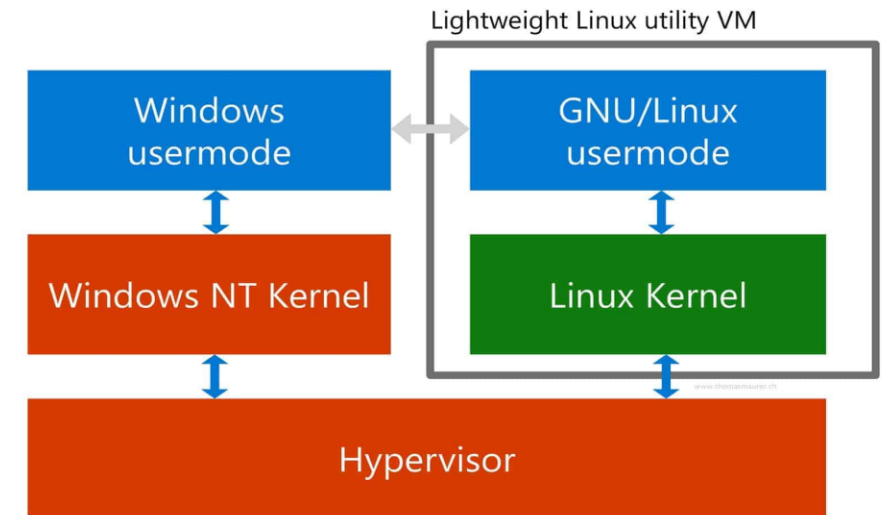
With WSL 2 you are as close to bare hardware as possible.

More detailed steps:

- https://learn.microsoft.com/en-us/windows/wsl/install

- https://learn.microsoft.com/en-us/windows/wsl/install-manual

WSL 2 architecture overview
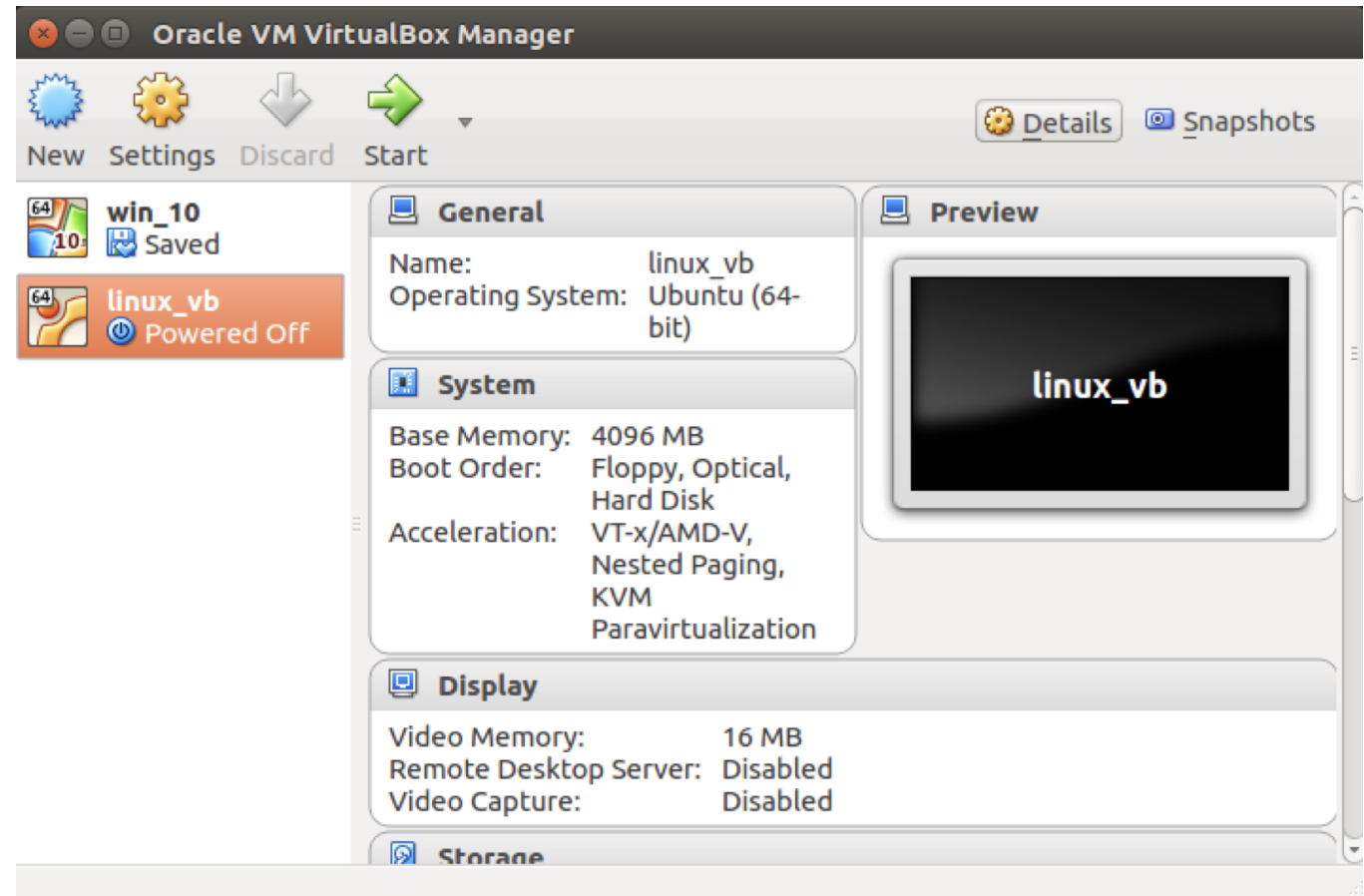
```
user@DESKTOP-112MLLI: /mnt/c/Windows/system32

user@DESKTOP-112MLLI:/mnt/c/Windows/system32$ uname -a
Linux DESKTOP-112MLLI 5.10.102.1-microsoft-standard-WSL2 #1 SMP Wed Mar 2 00:30:59 UTC 2022 x86_64 x86_64 x86_64 GNU/Linux
user@DESKTOP-112MLLI:/mnt/c/Windows/system32$ lsb_release -a
No LSB modules are available.
Distributor ID: Ubuntu
Description:    Ubuntu 20.04.5 LTS
Release:       20.04
Codename:      focal
user@DESKTOP-112MLLI:/mnt/c/Windows/system32$
```

In case you cannot create dual boot on your computer, or you have Mac.

Remember to add more than one CPU core, add more GPU memory, enable virtualization acceleration.

C++ and CMake

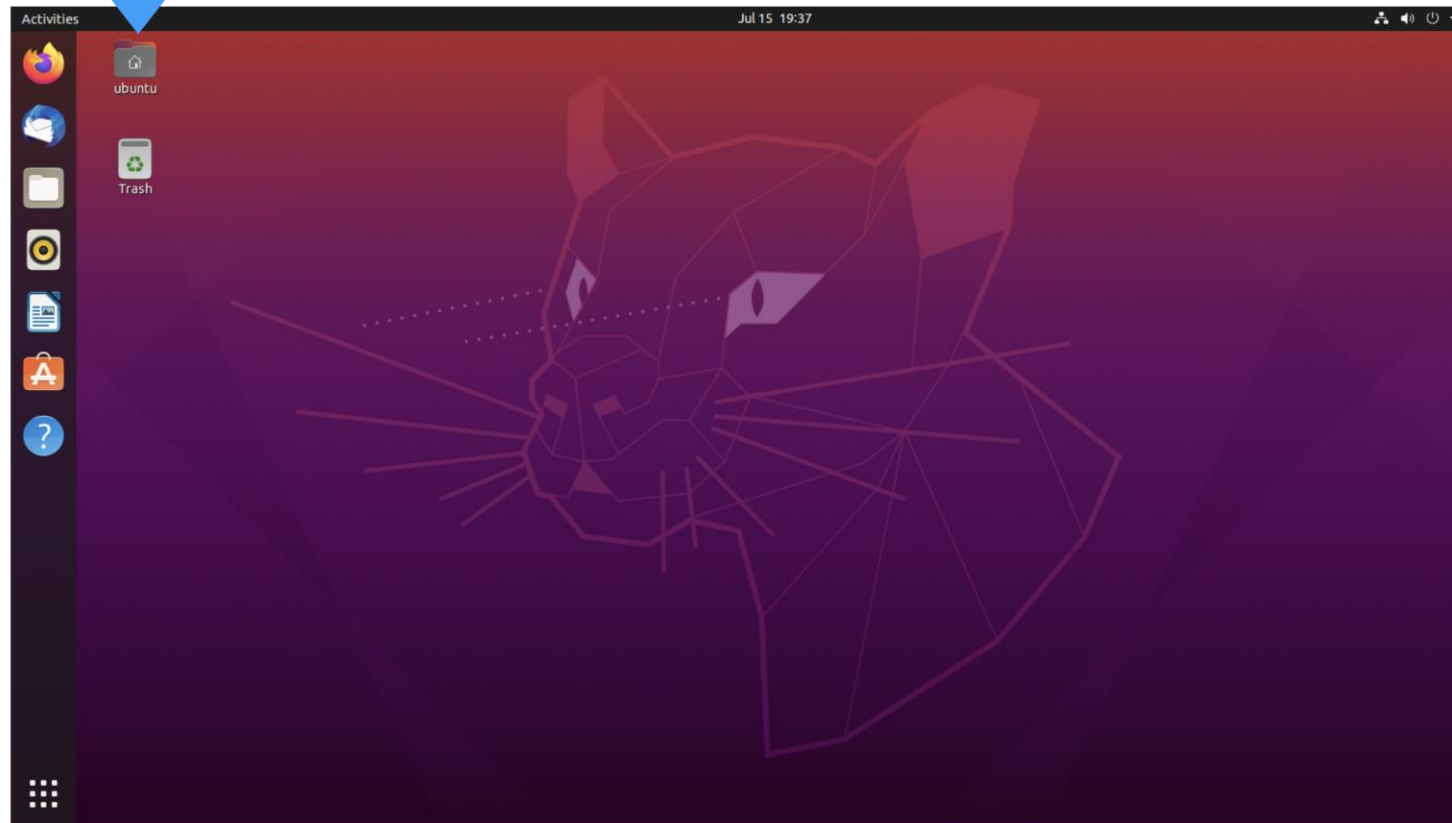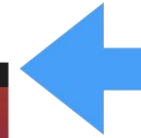# Compilation Process Overview

Preprocessor   Compiler   Linker

.h → .cpp → .o → Binary

.cpp → .cpp → .o

Source Code   Objects
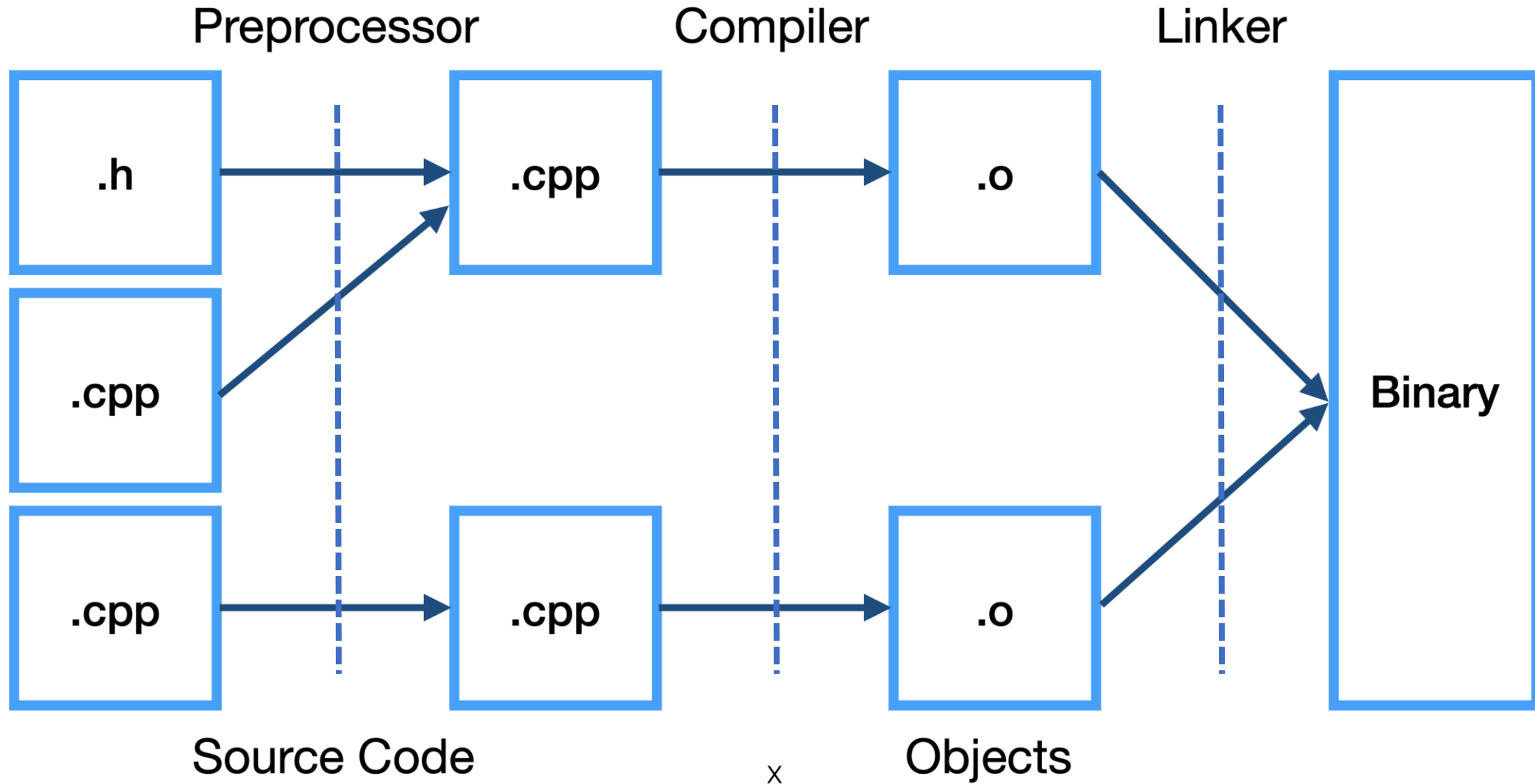
Let's write a simple program:

To compile, type this in the same directory:
- `g++ main.cpp -o my_bin`

But things can get worse:
- `g++ src_1.cpp src_2.cpp ... src_n.cpp -I include/ -Wall -Werror -Wpedantic -Wextra -pthread -ldl -llib1 ... -llibn -o mybin`

```cpp
// main.cpp
#include <iostream>

int main(int argc, char **argv) {
    std::cout << "Hello" << std::endl;
    return 0;
}
```

Now, let's make a compilation recipe -> the makefile.

- `nano makefile`

```
all: my_bin
my_bin: main.o
        g++ -o main main.o
main.o: main.cpp
        g++ -c main.cpp
```

To compile according to the makefile definitions, simply type:

- `make`

Look into the file system, you'll see the *.o (object) files.

CMake vs Makefile

```cmake
cmake_minimum_required(VERSION 3.13)
project(bpc_prp_opencv_lib)

set(CMAKE_CXX_STANDARD 17)
find_package(OpenCV REQUIRED)

set(library_name bpc_prp_opencv_lib)

include_directories(include)
add_library(${library_name} SHARED src/ImageProcessor.cpp)
target_link_libraries(${library_name} ${OpenCV_LIBS})
```
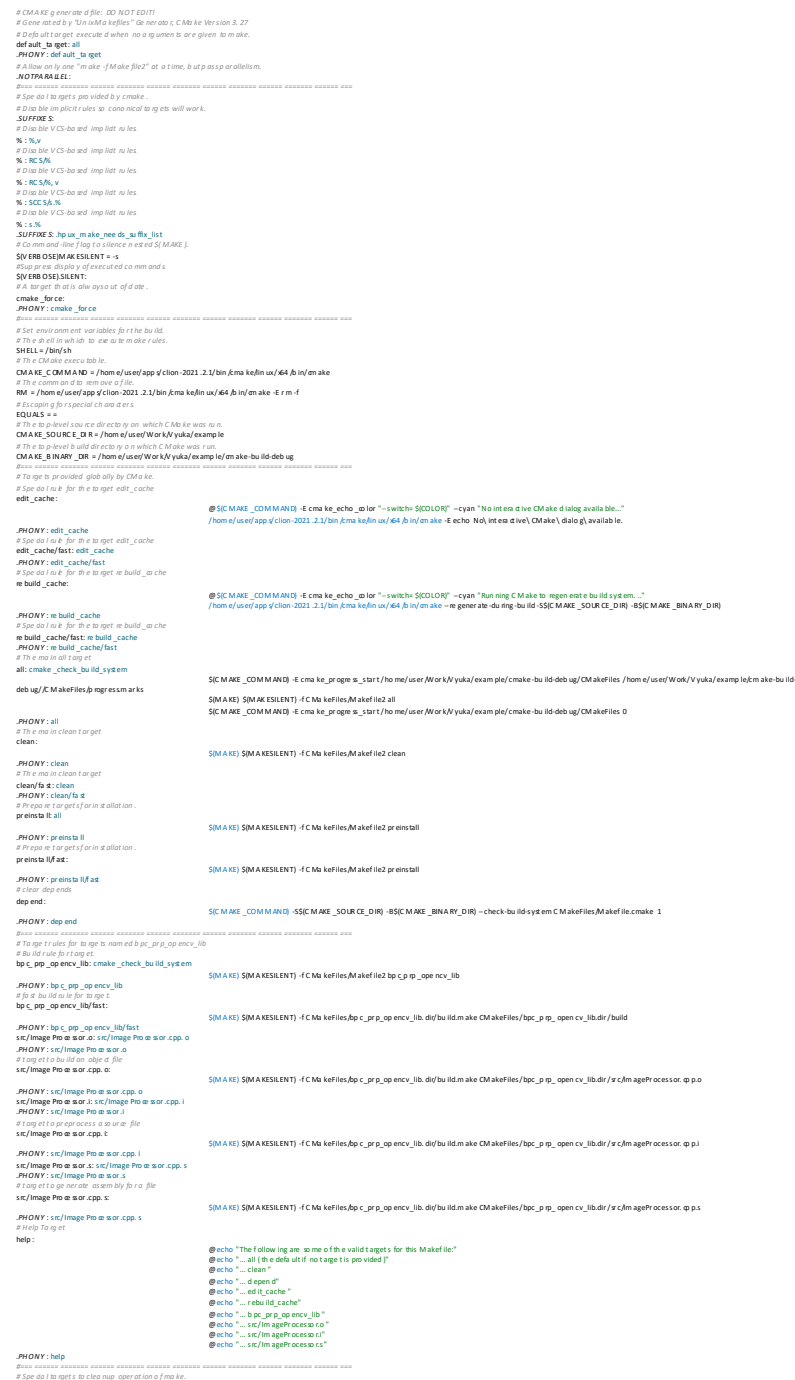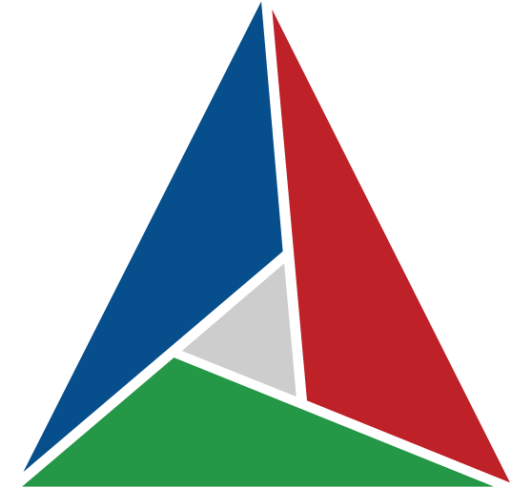
CMake is an open-source, cross-platform family of tools designed to build, test, and package software.

CMake is used to control the software compilation process using simple platform and compiler independent configuration files.

Generates native makefiles and workspaces that can be used in the compiler environment of your choice.

- Scripting language – has variables, conditions, loops, macros and functions

- Create and configure targets – *add_executable*, *add_library*, *add_custom_target*

- Manage dependencies just with *find_package, find_library*

```
cmake_minimum_required(VERSION 3.10)
project(my_cool_project)

set(CMAKE_C_COMPILER "gcc")

set(SOURCES src/main.c src/library.c)

add_executable(program ${SOURCES})

target_include_directories(program PUBLIC include)
```

Your typical project structure often looks like this:

```
/my_cool_project
 |-- CMakeLists.txt
 |-- build/
 |-- include/
 | \-- library.h
 \-- src/
    |-- library.c
    \-- main.c
```

Integrated Development Environment (IDE)

CLion is an "lightweight"
(~1GB on-disk-size) integrated
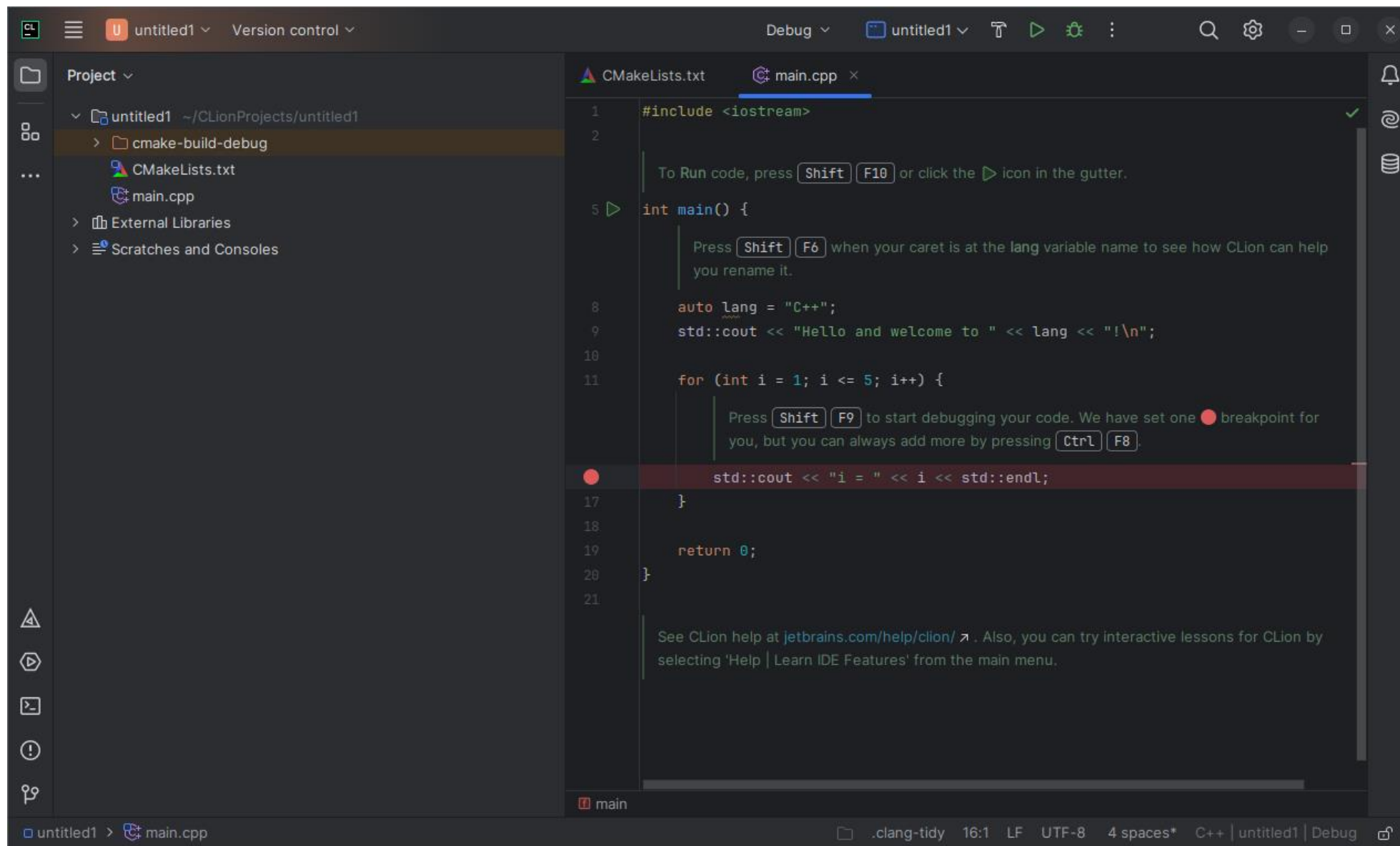development environment for C/C++
development.

Integrates all modern functions, like
code generation, on-the-fly static code
analysis, integrated debugger, remote
development,
3rd party plugins, etc …

Available free for academic use
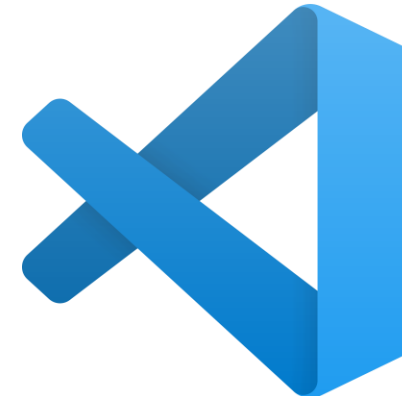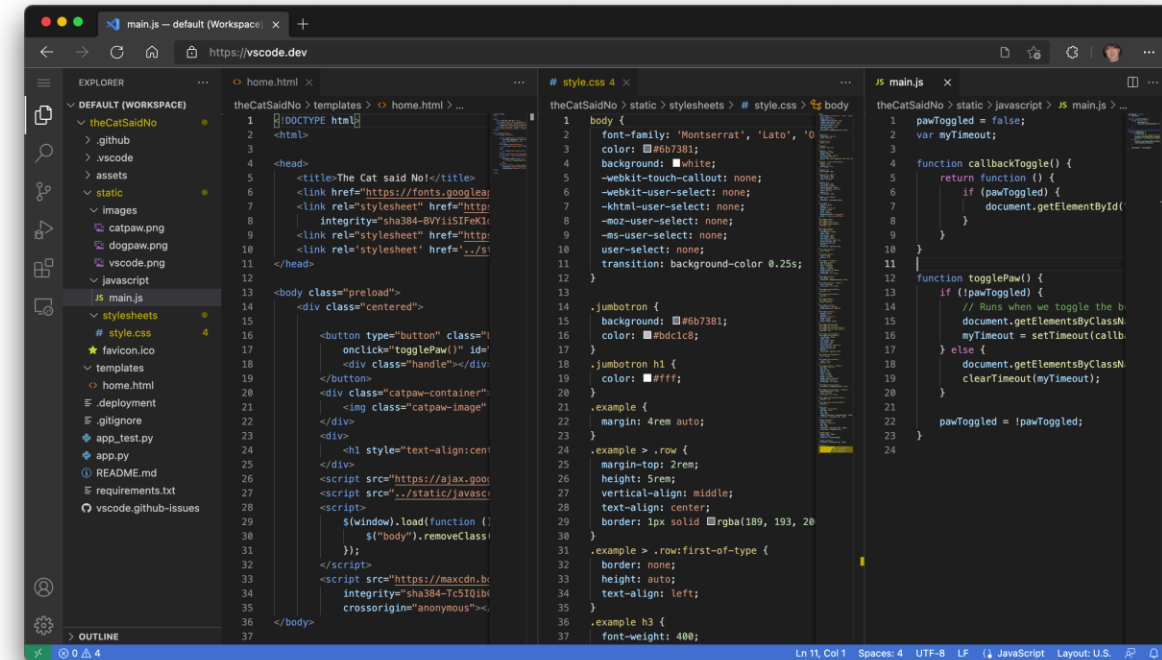
CLion

Visual Studio Code

VSC is a lightweight, modular and open-source text editor made by Microsoft

Allows to install plugins and customize functionality for any language/technology

According to the Stack Overflow Survey the VSC is the most used IDE worldwide

```
sudo snap install --classic code
```
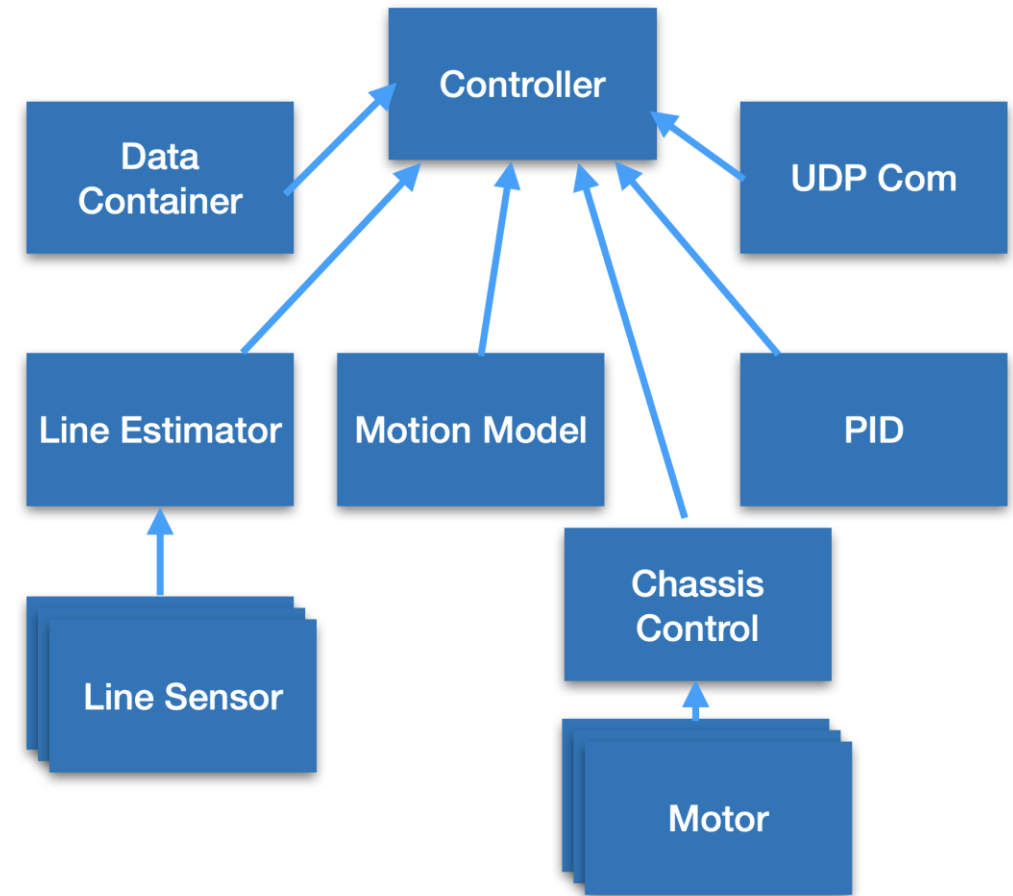
https://github.com/microsoft/vscode

C/C++ Coding Tips

Use OOP paradigm

- Design classes as black-boxes, where each box handles just one problem.

- Be able to describe your class and methods with one sentence.

- The connection between black boxes is called an API.

- Separate **Data** from **Algorithms**.

Use C++17

- Minimize using new and delete
- Use references - &
- Don't be afraid to use auto and templates, but don't abuse it...
- Use constexpr instead of macros
- Use scoped enums – enum class
- Avoid using raw C pointers use smart pointers instead:
    std::unique_ptr<> - single ownership
    std::shared_ptr<> - multiple ownership
    std::weak_ptr<> - temporary ownership

- Use STL functions and containers like – std::vector<>, std::array<>, std::string, std::find, ...
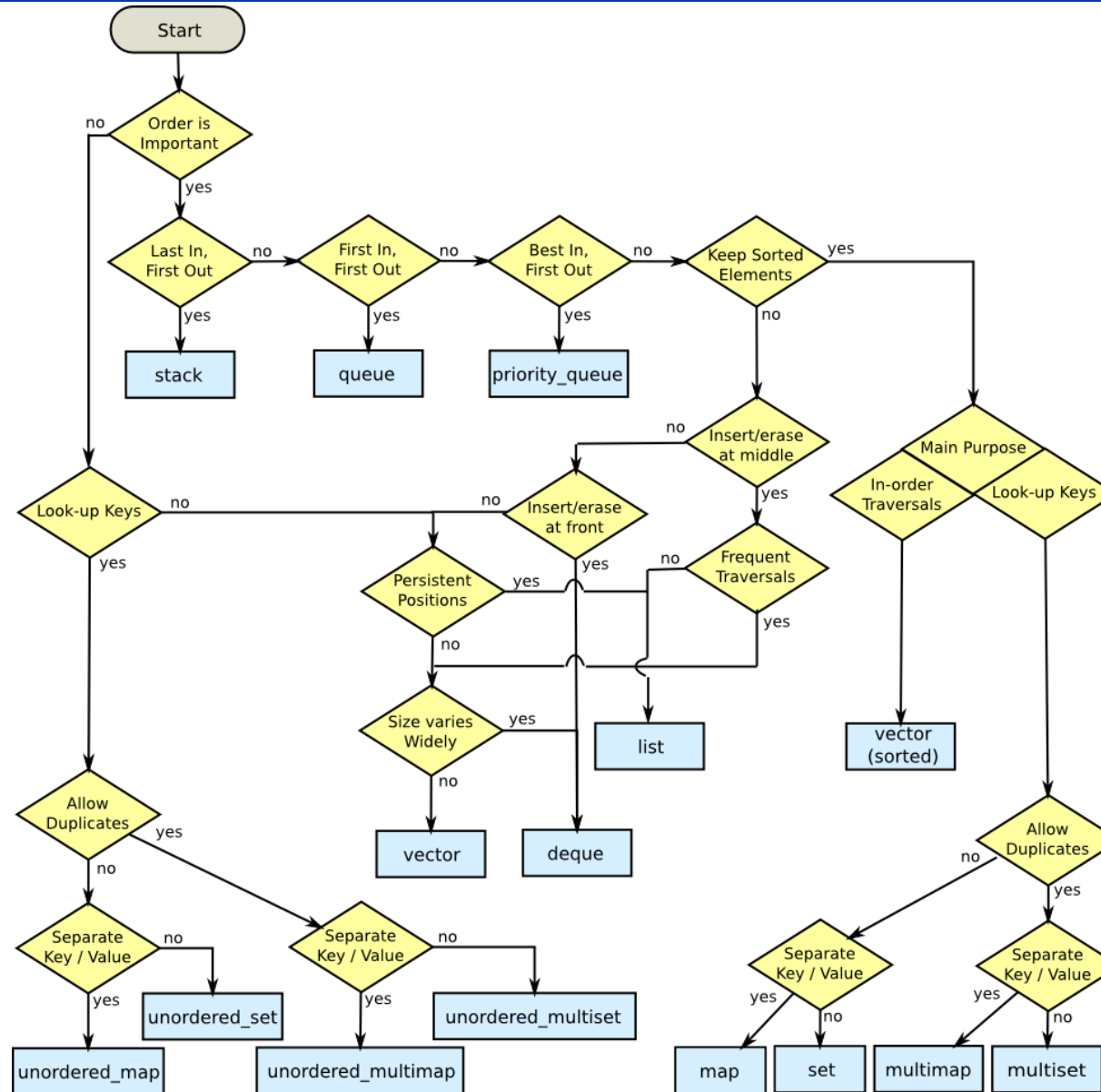
When designing classes and methods think about their purpose, and how can they be reused.

Mark data that are not going to be modified as const.

Mark member methods that do not modify member data as const.

Return a const & if the data should not be modified and is "non-trivially copyable".

const helps to keep a clear program design and better optimisations in compile time.

It also prevents you from making unwanted changes and hard to find bugs.

Testing – Unit Testing

- Catch bugs early – you can test new code right away

- Save time in long run – easier to catch or find bugs

- Easy to verify changes – immediately verify changes to old code

- Improve quality – think about design before implementing

- Easier collaboration – no more commits which breaks everything

Think about how your class (and methods) should and shouldn't be used.

Write tests for each scenario.

This will save you with debugging in the future.

If you solve a bug → Write a test for it!

```cpp
#include <gtest/gtest.h>

#include <my_project/MyLibrary.h>

TEST(MyLibraryTests, someCoolTest) {
    bool stuffWorks = true;
    ASSERT_TRUE(stuffWorks);
}

int main(int argc, char **argv) {
    testing::InitGoogleTest(&argc, argv);
    return RUN_ALL_TESTS();
}
```

Non-fatal failure - EXPECT_*
Fatal failure - ASSERT_*

EXPECT_FALSE(*condition*)
EXPECT_TRUE(*condition*)
ASSERT_EQ(*val1, val2*)
EXPECT_NE(*val1, val2*)
ASSERT_LT(*val1, val2*)
EXPECT_THROW(*statement, exception_type*)
ASSERT_ANY_THROW(*statement*)
ASSERT_STREQ(*str1, str2*)
EXPECT_DOUBLE_EQ(*val1, val2*)
ASSERT_DEATH(*statement, matcher*)

Jakub Minařík

203294@vut.cz

Brno University of Technology
Faculty of Electrical Engineering and Communication
Department of Control and Instrumentation

Robotics and AI
Research Group