CSCI 598B: Robotic Mapping & Localization

Kaveh Fathian
Assistant Professor
Computer Science Department
Colorado School of Mines

Lecture 2: Linux & C++

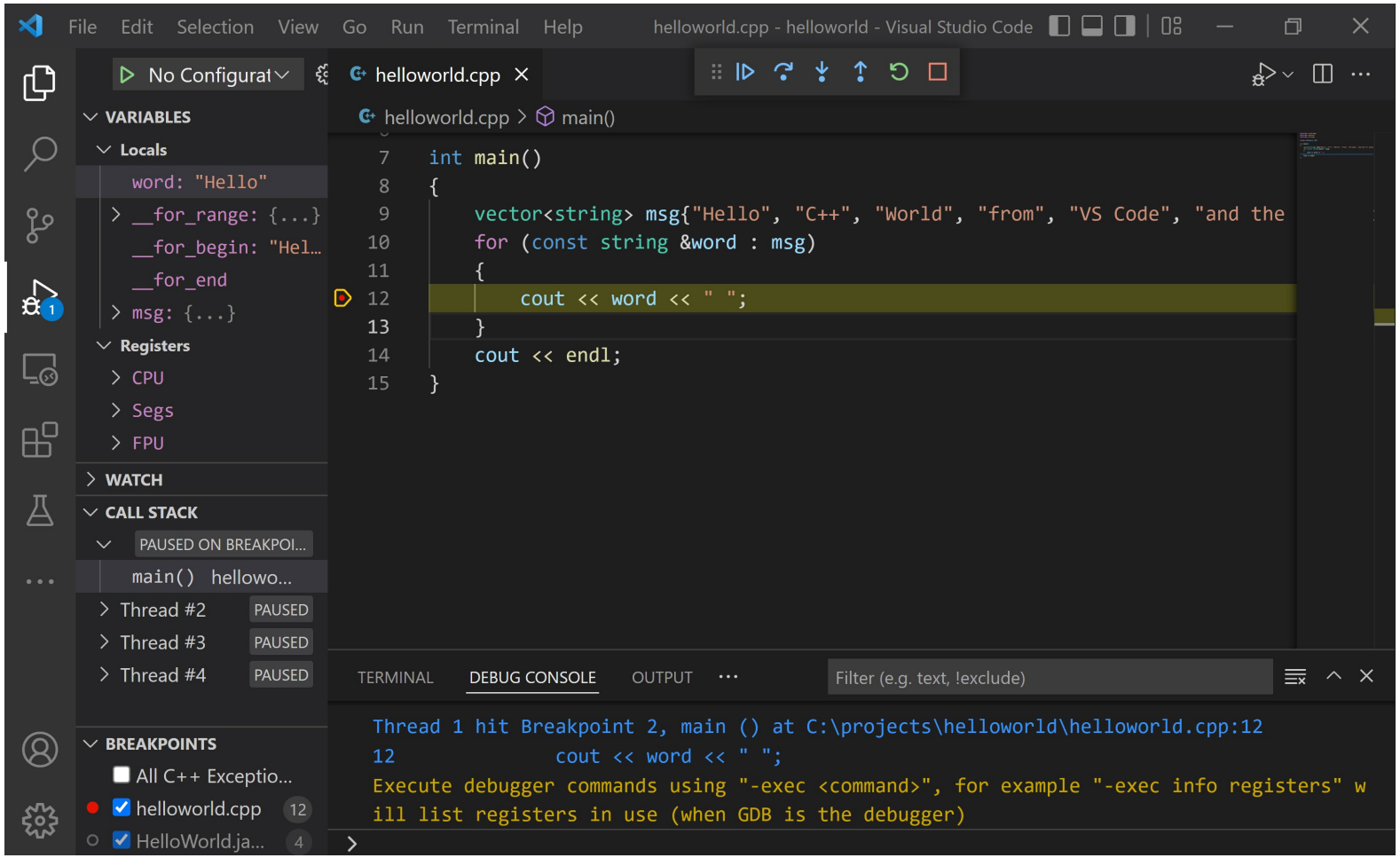
*Courtesy of Luca Carlone, Ignacio Vizzo, Igor Bogoslavskyi, Cyrill Stachniss

Outline

Lecture:

- Ubuntu setup
- Linux terminal
- C++

References



Why C++? Why Linux?

- 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

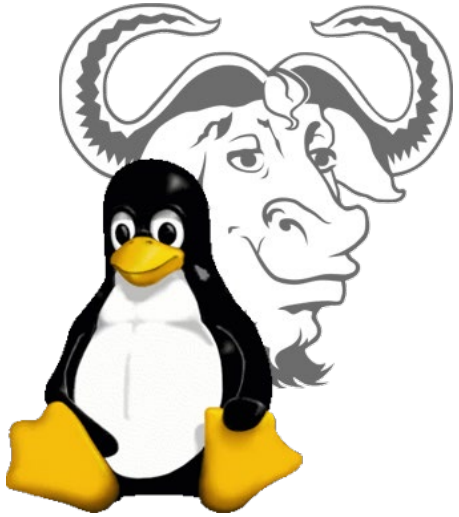
Stack Overflow survey: https://insights.stackoverflow.com/survey/2018/

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

Ubuntu

Install Ubuntu 22.04 (or 24.04):

1. Download the ISO image from https://releases.ubuntu.com/jammy/
2. Create a bootable USB stick
    1. How to create a bootable USB stick on **Windows**
    2. How to create a bootable USB stick on **Mac OS**
    3. How to create a bootable USB stick on **Ubuntu**
3. Boot from USB stick and install
    1. Install Ubuntu desktop (full erase)
    2. Install Ubuntu alongside Windows (dual boot)



Warning:

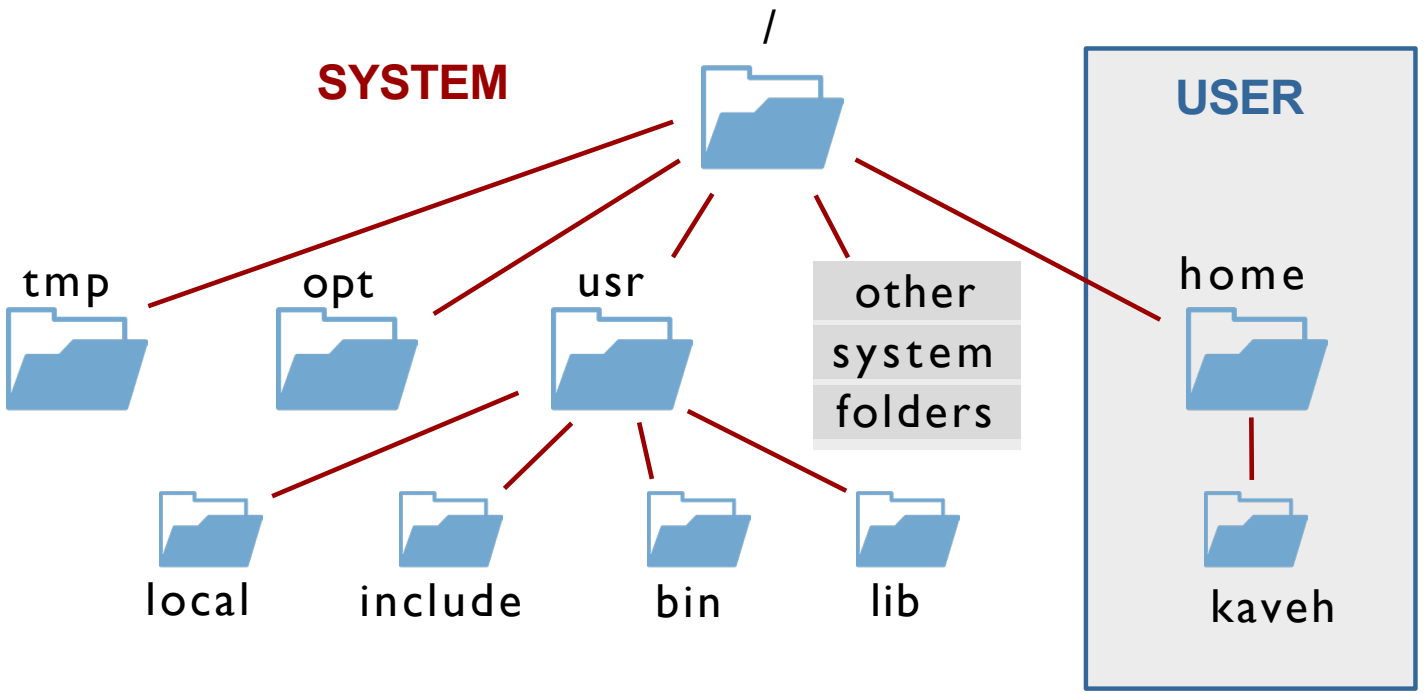
Partitioning can be tricky if you are installing Linux for the first time. There are plenty of guides for “dual-boot Ubuntu installation” alongside both Windows and OS X. In most cases, you would first need to shrink one of your partitions (e.g., in Windows) and create an “unallocated space” which will be used during the Ubuntu installation process.

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
- Our course 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

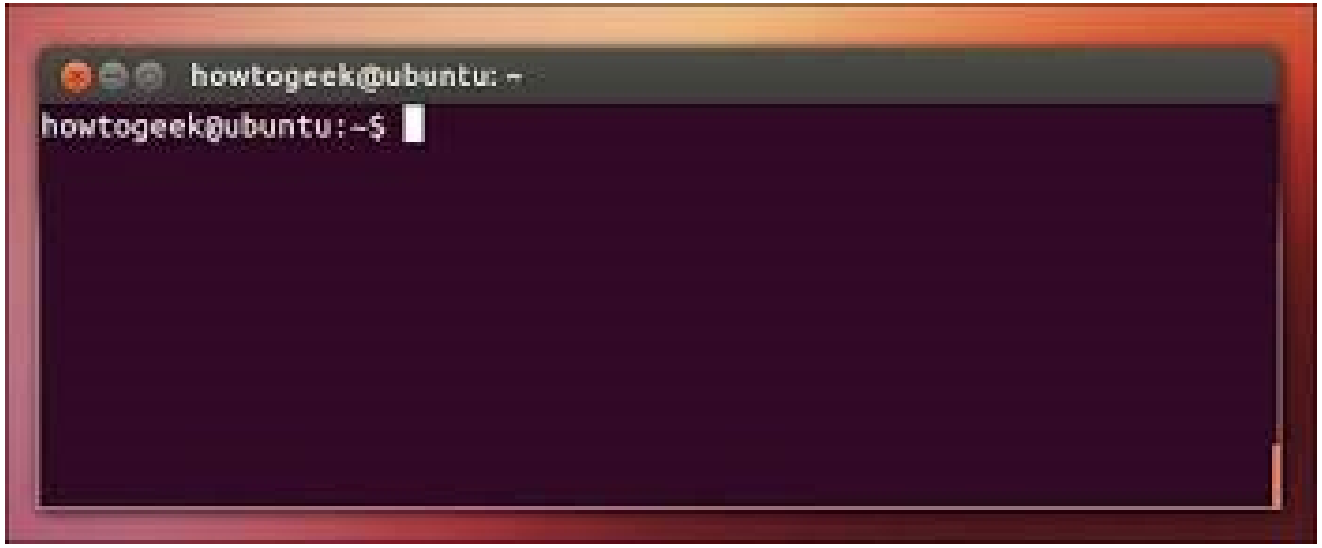
Understanding files and folders

- 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/kaveh/folder/ — **absolute** path to a folder
  - /home/kaveh/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>**: **ls**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**: 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.

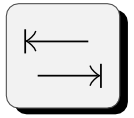
Use help with Linux programs

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

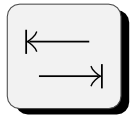
```
[/home/student]$ cat --help
Usage: cat [OPTION]... [FILE]...
Concatenate FILE(s) to standard output.
  -A, --show-all           equivalent to -vET
  -b, --number-nonblank    number nonempty output lines

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/ kaveh]$ cd D [TAB] [TAB]
Desktop/        Documents/      Downloads/
```

Standard input/output channels

- Single input channel: **stdin**
- Two output channels:
  - **stdout**: Standard output: channel 1
  - **stderr**: Standard **err**or output: channel 2
- Redirecting **stdout**
  - command 1> out.txt
  - command >> out.txt
- Redirecting **stderr**
  - command 2> out.txt
- Redirect **stdout** and **stderr** into a file
  - progamm > out.txt 2>&1
- Write **stdout** and **stderr** into different files
  - progamm 1>stdout.txt 2>stderr.txt

Files and folders

- **mkdir** [-p] <foldername> — **m**ake **dir**ectory
  Create a folder <foldername>
  (with all parent folders [-p])
- **touch** <filename>
  Create an empty file < filename >
  (was created to modify file timestamps, but can also be used to quickly create an empty file)
- **rm** [-r] <name> — **r**em**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/]$ ls
u01 .tex        v01 .pdf        v01    .tex
 u02.tex        v02.pdf         v02    .tex
u03 .tex        v03 .pdf        v03 .tex

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

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

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

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

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

Working with files

- **more/less/cat** <filename>
  Print the contents of the file
  Most of the time using **cat** if enough
- **nano** or **vim** <filename>
  nano is a minimalistic command-line text editor (great for beginners)
  More demanding users prefer **vim**
- **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>

Download, uncompress, execute

- **wget** <address>
  example: wget http://www.mit.edu/vnav.tar.gz

- **tar** <flags> <filename>
  example: tar -xvf vnav.tar.gz
  The flags “xvf” are respectively extract, verbose, file

Run:

```
$ ls -l hello.sh
  -rw-r--r--  1 username  staff  40 Aug 23 18:36 hello.sh

$ chmod +x hello.sh

$ ls -l hello.sh
  -rwxr-xr-x  1 username  staff  40 Aug 23 18:36 hello.sh

$ ./hello.sh
  Hello world!
```

← Verify permissions

← Change permissions

← Can now execute

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**

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

Ubuntu Setup

Once Linux is installed we need to update all the packages, to do so open a terminal (CTRL+Alt+T) and type

```
sudo apt update
sudo apt upgrade
sudo apt install build-essential cmake
```

Icon taken from Wikipedia

Why C++?

Slides adapted from Avery Wang

Companies that use C++

Google Microsoft

intel IBM amazon.com

Adobe facebook

Software written in C++

Java Ps

Windows Microsoft Office

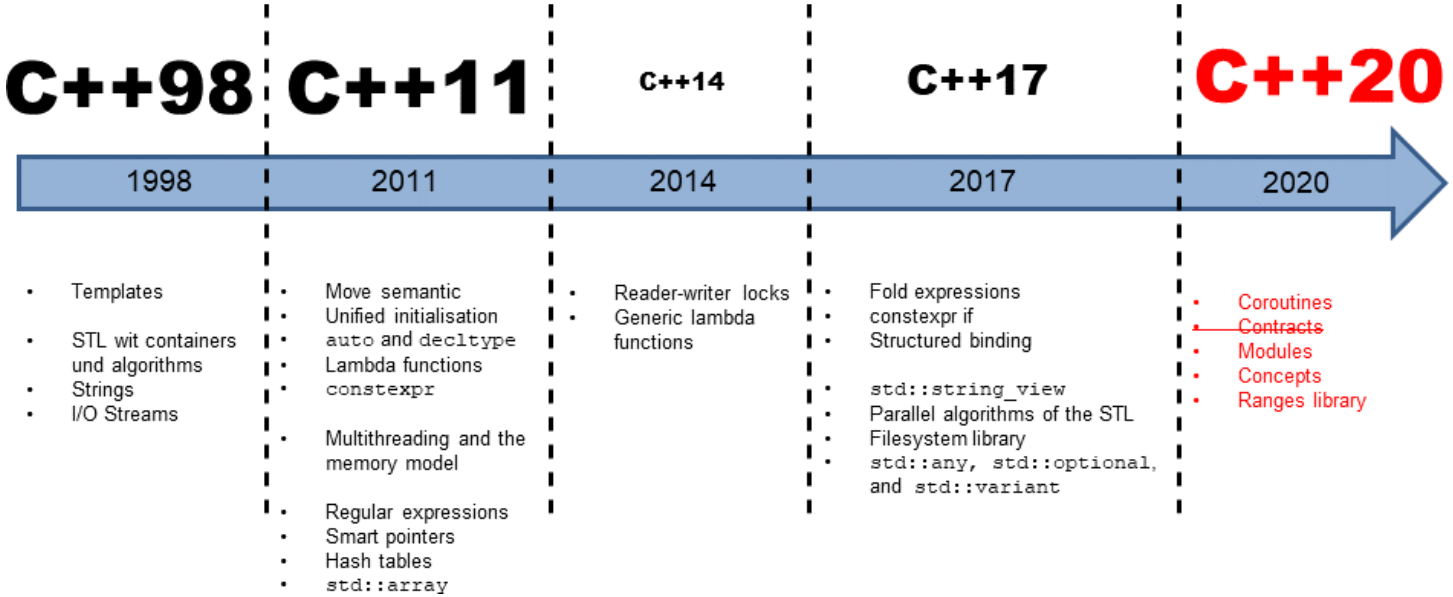
Browsers written in C++

Games written in C++

Assassin's Creed Call of Duty Mass Effect

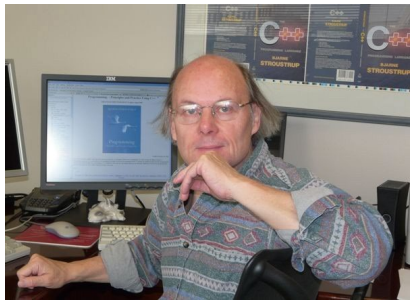
World of Warcraft StarCraft Halo

Evolution of C++



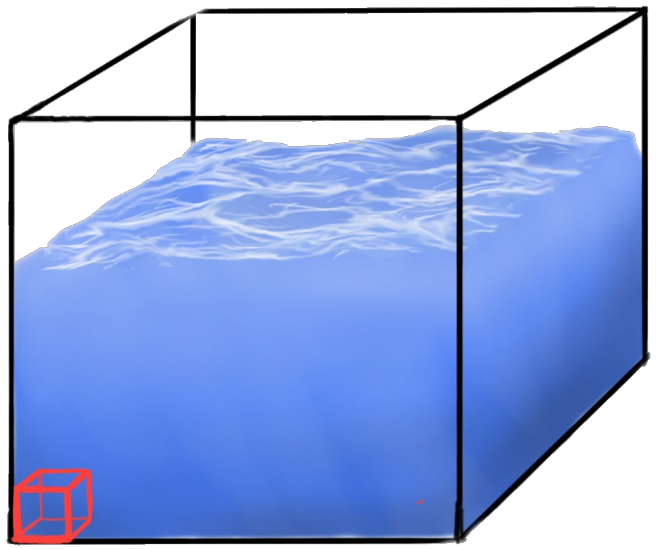
- Thompson & Ritchie created **C** in 1972
  - No objects/classes.
  - Difficult to write code that worked generically
  - Tedious when writing large programs
- The first vestiges of **C++** were created by Stroustrup in 1983
  - Fast
  - Simple to Use
  - Cross-platform
  - High-level features

Ken Thompson & Dennis Ritchie



Bjarne Stroustrup

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!**

```
# 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 very 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** https://google.github.io/styleguide/cppguide.html
- 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]**

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

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

#include directive

- Two variants:
  - `#include <file>` — system include files
  - `#include "file"` — local include files
- Copies the content of `file` into the current file

```
# include " some_file.h"
// We can use contents of file " some_file.h"
now. 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
  - GCC

Compile and **run** Hello World example:

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

References

- MIT Visual Navigation course: https://vnav.mit.edu/
- Modern C++ for Computer Vision:
  - https://www.ipb.uni-bonn.de/teaching/cpp-2020/index.html
  - https://www.ipb.uni-bonn.de/teaching/modern-cpp/index.html
- C++ Reference: https://en.cppreference.com/w/cpp
- C++ 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/