CPP-02

* Compilation flags

-std=c++11 → {this tells we want to use c++11}

-o → {for name of executable}

-Wall → {Show all the Variables}

* Optimization options:

-O0 ⇒ no optimization

-O3 or -Ofast ⇒ full optimizations.

* Keep debugging Symbols : -g

⇒ Recommended Debugger ⇒ gdb → gdbgui

* Functions

ReturnType FunctName(ParamType1 in_1, ParamType2 in_2) {

return return_Value;

}

→ Code can be organized into functions.
→ Functions create a scope.
→ Single return value from a function.
→ Any number of input variables of any types.
→ Should do only one thing and do it right.
→ Name must show what function does.
→ GOOGLE-STYLE
  ↳ Name function in CamelCase
  ↳ Write small functions.

★ Declaration and definition

⇒ Function declaration can be separated from the implementation details.
  ↳ (definition)
  ↳ (Semicolon and in the end)

★ Passing big object

⇒ By default in C++, objects are copied when passed into functions.
⇒ If objects are big it might be slow.
  ↳ Pass by reference to avoid copy.
→ Use snake_case for all function arguments.

→ GOOGLE-STYLE
↳ Avoid using non-const refs.

* Function Overloading

⇒ Compiler infers a function from argument not return type.

→ GOOGLE-STYLE
↳ Avoids non-obvious overloads

* Default argument

Google Style
↳ (Only use them when readability get much better)

* Dont reinvent the wheel

⇒ use #include <algorithm.h>

* Header / Source Separation

⇒ Move all declarations to header files (*.h)

⇒ Implementation goes to *.cpp or *.cc

* How to build this?

Compiler Error

linker Error

#Progma once
↳ {Ensure files are included only once}

* Use modules and libraries

① Compile models

```
C++ -std=C++11 -c tools.cpp -o tools.o
```

⊛ Organize modules into libraries

ar rcs libtools.a tools.o <other_modules>

⊛ Link libraries when building code.

c++ -std=c++11 main.cpp -L . -ltools -o main

{Where to search for libraries}

* Libraries

↳ Multiple object files that are logically connected.

Static

⇒ faster
⇒ Takes lot of space
⇒ Become part of the end binary.
⇒ named lib*.a

Dynamic

⇒ Slower
⇒ Can be copied
⇒ referenced by a Program
⇒ named lib*.so

ar rcs libname.a module.o module.o

⇒ To use a library we need a header and the compiled library object.

★ CMake → {Scripting language}

⇒ One of the most popular build tools.

⇒ Cross platform

⇒ The library creation and linking can be rewritten as follows:

```
add_library(tools tools.cpp)
add_executable(main main.cpp)
target_link_libraries(main tools)
```

★ Typical project Structure

```
→ Project_name/
    → CMakeLists.txt
    → build/ # All generated build files
    → bin/
        → tools_demo
    → lib/
        → libtools.a
    → src/
        → CMakeLists.txt
        → tools.h
        → tools.h
        → tools.cpp
        → tools_demo.cpp
    → tests/ # Tests for your code
        → test_tools.cpp
        → CMakeList.txt
    → readme.md # How to use your code
```

* Build Process

⇒ CMake reads CMakeList.txt sequentially.

1. cd <project_folder>
2. mkdir build
3. cd build
4. cmake
5. make -j2 # pass your number of cores here.

* First working CMakeList.txt

```
Project (first_Project)
cmake_minimum_required (VERSION 3.1)
Set (CMake_CXX_STANDARD 11)
Set (EXECUTABLE_OUTPUT_PATH ${PROJECT_SOURCE_DIR}/bin)
    # Location to store all the binaries
Set (LIBRARY_OUTPUT_PATH ${PROJECT_SOURCE_DIR}/lib)
    # Location to store all the Libraries
include_directories (include)
add_library (tools src/tools.cpp)
add_executable (main src/tools_main.cpp)
target_link_libraries (main tools)
```

* Mandatory
* Mandatory
# Use C++11

* Useful commands in CMake

→ Has features of a scripting language (i.e. functions, control structures, variable etc.)
→ All variables are string.
→ Set value with set(VAR VALUE)
→ Get value of a variable with ${VAR}

⇒ Show a message message (STATUS "message")

{WARNING, FATAL_ERROR}

* Set compilation option in CMake

```
if (NOT CMAKE_BUILD_TYPE)
   set(CMAKE_BUILD_TYPE Release)
endif()
```

Release → {binary is small and runs fast}

Debug → {no optimization}

⇒ Set additional flags

```
Set (CMAKE_CXX_FLAGS "-Wall -Wextra")
set (CMAKE_CXX_FLAGS_DEBUG "-g -O0")
Set (CMAKE_CXX_FLAGS_RELEASE "-O3")
```

* Performing a clean build

① cd project/build

② make clean #remove generated binaries

③ rm* -r *

* Use pre-compiled libraries

```
add_library (tools SHARED IMPORTED)
set_property (TARGET tools
              PROPERTY IMPORTED_LOCATION
              "${LIBRARY_OUTPUT_PATH}/libtools.so")
```