This document includes the guidelines for developing the programming exercises for the Operating System course (CIIC 4050 / ICOM 5007).

1. Initial System Requirements

Before setting up your project structure, you must install the core development tools. These include the compiler (GCC/Clang), the build system generator (CMake), and the standard library headers.

A. Ubuntu Linux

On Ubuntu, the easiest way to get everything you need is to install the build-essential package, which includes the GCC compiler and the make utility.

1. **Update your package list:** `sudo apt update`
2. **Install the core tools:** `sudo apt install build-essential cmake clang-format`
    - `build-essential`: Provides gcc, g++, and make.
    - `cmake`: Required to process the CMakeLists.txt files.
    - `clang-format`: Required to automate the Google coding style formatting.

B. macOS

macOS users should use the Clang compiler (provided by Xcode) and can use a package manager like **Homebrew** to install CMake and Clang-Format.

1. **Install Command Line Tools:** Open a terminal and run: xcode-select --install
2. **Install Homebrew (if not already installed):** Visit brew.sh for the installation command.
3. **Install CMake and Clang-Format:** `brew install cmake clang-format`

2. File structure

The project needs to have the following file structure:

```
project/
├── CMakeLists.txt
├── include/
│   └── functions.h
├── src/
│   ├── functions.c
│   └── main.c
├── build
│
└── test
```

Where:

- The include folder contains all the header files with the prototype of the functions and macro definitions.
- The src folder contains all the source code files including the main file and the necessary files for the function definitions
- The build folder will contain all the files resulting from the compilation process, including object files and the executable file for the program.
- The test folder includes some scripts that will be used to evaluate the style and the performance of the developed programs.
- The CMakeLists.txt is a **mandatory file** that contains the necessary steps to build the project using cmake. It will be used for standardization purposes.

3. Coding Style

All developed code must comply with the Google style guidelines, as outlined in the style document. The mentioned document includes guidelines regarding header *files, naming,* and *formatting*.

Of all these aspects, the formatting can be automated using software tools. A .clang-format file is a configuration file used by the Clang-Format tool to format code according to specified style rules automatically. This file allows developers to define consistent code formatting conventions, such as indentation, spacing, and brace placement across a project or team. By placing a .clang-format file in a project's directory, Clang-Format can apply the defined style to all code files within that directory, ensuring uniformity and improving code readability. The .clang-format file is highly customizable, allowing developers to set formatting rules that match their organization's preferred coding standards or guidelines.

In this case, the tool *clang-format* can be used in VSCode to automatically apply the Google coding style. To achieve this, you must:

1. Copy the `.clang-format` file to the folder **containing** your working directory. **It's not the project folder, but one level above.**
2. Go to `settings->formatting` and enable `"format on save"`
3. On `"format on save mode,"` choose `"file"`.

4. **Set the tab key to use "two spaces."**
   Click the bottom-right bar labeled "spaces," then set its value to 2.

Testing the code style

Inside the test folder, you will find the `test_style.sh` script. If you execute this script, it will highlight every aspect of the code that does not comply with the required style. Please modify every aspect of the code highlighted by this tool, since it will be used to grade your submissions.

To execute the script you will need to:

- Move to the test folder: `cd test`
- Modify the execution permissions of the files in **the test folder**: `chmod +x *`
- Execute the script: `./test_style`

4. Compile the project

The course projects will be compiled using a `CMakeLists.txt` file. This file is a script used by CMake, a cross-platform build system generator, to configure the build process for a software project. It defines the project's structure, specifies source files, sets compiler options, handles dependencies, and manages platform-specific configurations.

The `CMakeLists.txt` file is essential because it enables developers to generate build files (like Makefiles on Linux or Visual Studio projects on Windows) in a way that is consistent across different environments. This makes it easier to compile and maintain the project across platforms, ensuring a flexible, repeatable build process. Additionally, CMake simplifies the management of complex projects by automating the detection of libraries, setting include paths, and handling compiler and linker flags, thereby reducing manual configuration effort.

a. Example of project compilation using CMakeLists.txt

- Create a project using the file structure of section 1. In the `include` folder, create the file `hello.h`, and in the `src`
-
- folder, create the files `main.c` and `hello.h`.
- In the `hello.h` file put the following code:

```
#ifndef HELLO_H
#define HELLO_H

void PrintHelloWorld();

#endif  // HELLO_H
```

- In the `hello.c` file put the following code:

```
#include "../include/hello.h"
#include <stdio.h>

void PrintHelloWorld() {
  int IndexForPrint;
  for (IndexForPrint = 0; IndexForPrint < 10; IndexForPrint++) {
    printf("Hello, World!...%d\n", IndexForPrint);
  }
  return;
}
```

- Finally, in the `main.c` file put the following code:

```
#include "../include/hello.h"
```

```
int main() {
  PrintHelloWorld();
  return 0;
}
```

- Now, create a `CMakeLists.txt` file in the root folder of the project with the following code:

```
cmake_minimum_required(VERSION 3.10)

# Set the compiling mode flags
set(CMAKE_CXX_FLAGS_DEBUG "-g -O0 -DDEBUG")
set(CMAKE_CXX_FLAGS_RELEASE "-O3 -DNDEBUG -s")

# Set the project name
project(HelloWorld)

# Include the current directory for header files
include_directories(include)

# Add the executable
add_executable(hello_world
    src/main.c
    src/hello.c)
```

- Generate the build files from a terminal by getting into the `build` and using cmake:

```
cd build
cmake ..
```

- Now compile the project:

```
make
```

- Finally, run the program:

```
./hello_world
```

5. Compile and debug using CMake in VSCode

- Verify the C/C++ Extension Pack in VSCode is installed.
- Open the folder from the exercise in VS code.
- Make sure the root folder contains the CMakeLists.txt file.
- Go to the cmake extension mode on the left bar (it looks like a triangle and a wrench):

- Select project folder
- In configure, select GCC compiler and then click on the 'sheet icon' to configure
- Select debug or release mode
- In Build, click the build button

- Now you can run the program, **or you can debug the program** (if compiled in debug mode). To debug, set a breakpoint first, then click Debug. To run, you only need to click on the Launch button.