What is CMake?

🚀 CMake, Make & GCC – Summary for Beginners

📌 Why CMake?

- When building C/C++ projects (like for Raspberry Pi Pico), you'll need to compile multiple files.
- Doing it manually with `gcc` becomes tedious as your project grows.
- **Make** helps automate compilation.
- **CMake** helps automate **creating Makefiles**, supporting **cross-platform builds** (Windows, Linux, macOS, Ninja, Visual Studio, etc.).

✅ 1. Compiling C with `gcc`

🧱 Basic Example:

```
bash
CopyEdit
gcc -o hello hello.c
```

- `o hello`: name of the output binary
- `hello.c`: source file
- Run with `./hello`

⚙️ Multiple Source Files:

```
bash
CopyEdit
gcc -o main main.c random.c -lm
```

- `lm`: links the math library (e.g., `sqrt`, `cos`)

🔧 Separate Compile & Link:

```
bash
CopyEdit
gcc -c main.c    # → main.o
gcc -c random.c  # → random.o
gcc -o main main.o random.o -lm
```

✅ 2. Make – Automating Compilation

🔄 What is `make`?

- A build automation tool using **Makefile** to define how files compile.
- Avoids recompiling everything unnecessarily.

🧾 Example `Makefile`:

```
make
CopyEdit
all: hello

hello: hello.c
    gcc -o hello hello.c

clean:
    rm -f hello
```

🔁 Usage:

```
bash
CopyEdit
make        # Builds hello
make clean   # Cleans up
```

✅ 3. CMake – Automating Makefiles

🧰 Why CMake?

- Helps generate platform-specific build files (e.g., Makefiles, Visual Studio projects).
- Keeps build files **outside source directory** (Out-of-Source Build).
- Great for **cross-platform** development.

🗂️ Typical Project Structure:

```
arduino
CopyEdit
project/
├── CMakeLists.txt
├── main.c
├── random.c
├── random.h
└── build/
```

📄 Minimal `CMakeLists.txt`:

```
cmake
CopyEdit
cmake_minimum_required(VERSION 3.10)
project(main)
```

```
add_executable(main main.c random.c)
target_link_libraries(main m)
```

🛠️ Build & Run with CMake:

```
bash
CopyEdit
mkdir build
cd build
cmake ..
make
./main
```

- `cmake ..` → generates the Makefile using `CMakeLists.txt` in parent dir
- `make` → compiles code into executable
- `./main` → runs the program

🧼 Clean Build:

```
bash
CopyEdit
make clean    # removes built files
```

🔁 Dependency Tracking

- If any source/header file changes, `make` will **only rebuild what's needed**.
- CMake-generated Makefiles track dependencies smartly.

🌍 The Beauty of CMake

- Supports **multiple build systems**:
  - `Make`, `Ninja`, `Visual Studio`, `Xcode`, etc.
- Cross-platform and scalable.
- Widely used in open-source and professional projects.

🎯 TL;DR Workflow

1. Write `CMakeLists.txt`
2. Run `cmake ..` inside `build/`
3. Run `make`
4. Run your program (e.g., `./main`)
5. Modify code → `make` again to rebuild updated parts