Cmake

sexta-feira, 11 de outubro de 2024 20:16

CMake Tutorial EP 1 | Understanding The Basics

→ The idea is to follow this tutorial and the Cpp instructions!

Why CMake?

Make files
- file that runs commands
- Something that runs in bash
- What cmakes turns into
- Cmake creates the make files
- Runs the comands we tell it too

Basically we compile it and it runs our program... hummm based on the .cpp file we send as source

Entrypoint documentation https://cmake.org/cmake/help/latest/

How to install Installing CMake in 2 minutes on Windows

Install make gcc and g++ on windows How to Install Make/GCC/G++ on Windows 10/11 - Scoop Package Manager

Install make
- https://gnuwin32.sourceforge.net/packages/make.htm
- Download the installer
- Add to environmental variables

install gcc
-
install g++

https://code.visualstudio.com/docs/cpp/config-mingw (microsoft) - Ok this is the best link

Pacman is a package manager that can be used to install, update, and remove packages, including C/C++ packages:

But still need to install the MAKE one from the previous video

Once the project is building.... let's move on
- makefiles doesn't accepts spaces
- make files relies on a top level file called CmakeLists.txt
- With the cmakefile, we can run cmake
  - Need to tell where the source is and build folder is
  - don't use the same folder :D
    - Create a build directory
    - or out/build
  - But there is no standard
  - cmake [options] -S <path-to-source> -B <path-to-build>

```
PS C:\Projetos\ConceptsReview\Cmake\Cmake Tutorial> cd out/build/
PS C:\Projetos\ConceptsReview\Cmake\Cmake Tutorial\out\build> cmake -S ../../ -B .
```

→ Build with Cmake

→ Relies on the top level file CMakeLists.txt

```
PS C:\Projetos\ConceptsReview\Cmake\Cmake Tutorial> cd out/build/
PS C:\Projetos\ConceptsReview\Cmake\Cmake Tutorial\out\build> cmake -S ../../ -B .
```

→ To run Cmake needs to specify the source and output

→ It'll generate a bunch of files don't touch it!

→ Edit only the Cmakelist

This first test was with na empty CmakeList.txt file. It doesn't build anything but generates a build sorce file with some configurations. That is nice but let's start to write some commands in the CmakeList.txt

```
-- Build files have been written to: C:/Projetos/ConceptsReview/Cmake/Cmake Tutorial
PS C:\Projetos\ConceptsReview\Cmake\Cmake Tutorial> cmake -S . -B .\out\build\ -G "Unix Makefiles"
-- The C compiler identification is GNU 13.2.0
-- The CXX compiler identification is GNU 13.2.0
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- Check for working C compiler: C:/msys64/ucrt64/bin/cc.exe - skipped
-- Detecting C compile features
-- Detecting C compile features - done
-- Detecting CXX compiler ABI info
-- Detecting CXX compiler ABI info - done
-- Check for working CXX compiler: C:/msys64/ucrt64/bin/c++.exe - skipped
-- Detecting CXX compile features
-- Detecting CXX compile features - done
-- Configuring done (4.2s)
-- Generating done (0.1s)
-- Build files have been written to: C:/Projetos/ConceptsReview/Cmake/Cmake Tutorial/out/build
PS C:\Projetos\ConceptsReview\Cmake\Cmake Tutorial> _
```

→ Force to generate the mingw files instead of the . sln!

cmake -S . -B .\out\build\ -G "Unix Makefiles"

→ First you prepare the mingw files! Here we build, yet but prepare the makefiles with all instructions to build anywhere (windows / mac / linux)

```
PS C:\Projetos\ConceptsReview\Cmake\Cmake Tutorial> cmake -S . -B .\out\build\ -G "Unix Makefiles"
-- Configuring done (0.2s)
-- Generating done (0.1s)
-- Build files have been written to: C:/Projetos/ConceptsReview/Cmake/Cmake Tutorial/out/build
PS C:\Projetos\ConceptsReview\Cmake\Cmake Tutorial> cd out
PS C:\Projetos\ConceptsReview\Cmake\Cmake Tutorial\out> cd build
PS C:\Projetos\ConceptsReview\Cmake\Cmake Tutorial\out\build> make
[ 50%] Building CXX object CMakeFiles/OLAS.dir/main.cpp.obj
[100%] Linking CXX executable OLAS.exe
[100%] Built target OLAS
PS C:\Projetos\ConceptsReview\Cmake\Cmake Tutorial\out\build> .\OLAS.exe
Hey Indy
PS C:\Projetos\ConceptsReview\Cmake\Cmake Tutorial\out\build>
```

→ Here we when the build process worns! Based on the OS it in this case generates an executable

→ Run the executable

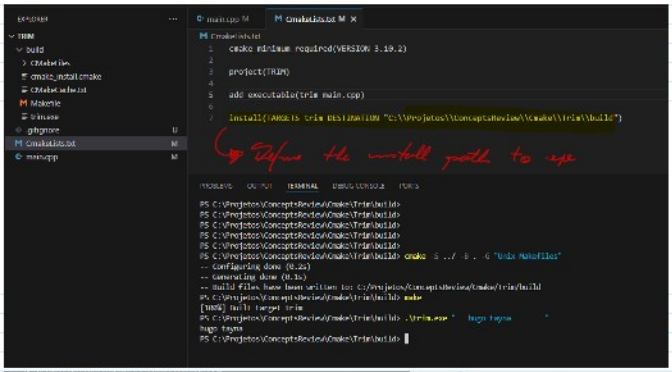
Generate a Build System

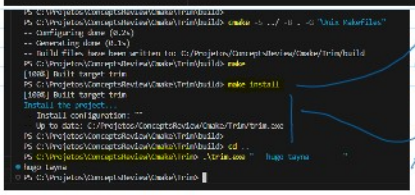
One of the main differences between CMake and Make is that CMake creates output that can be used by build systems like Make. This means that CMake acts as a generator for other build systems and it's not responsible for the actual compilation. In contrast, the output of Make is a compiled binary that can be executed on the target computer.

Make → Responsable for compiling the solution to be executed in the operational system

Cmake → Generates the config files that can generate the executable once when invoked

↳ Forms on a cross platform app





Libraries

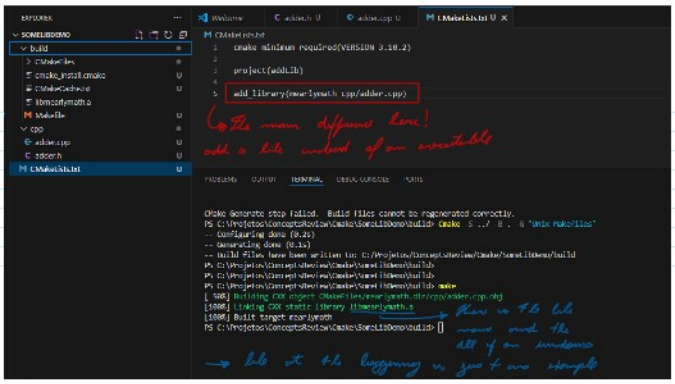
Static Lib

Name Spaces

↳ Do the c++ code first

↳ We need a main because libs don't have an entry point function

↳ Created a simple example with an overload and 2 static methods

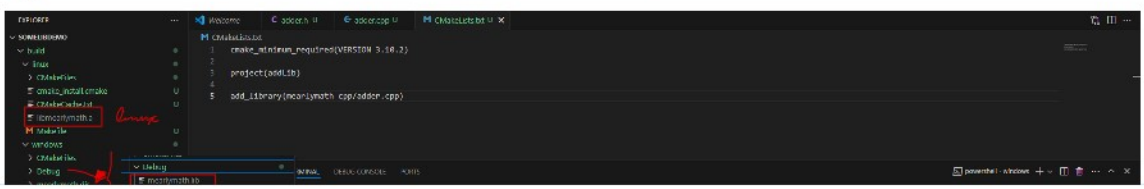


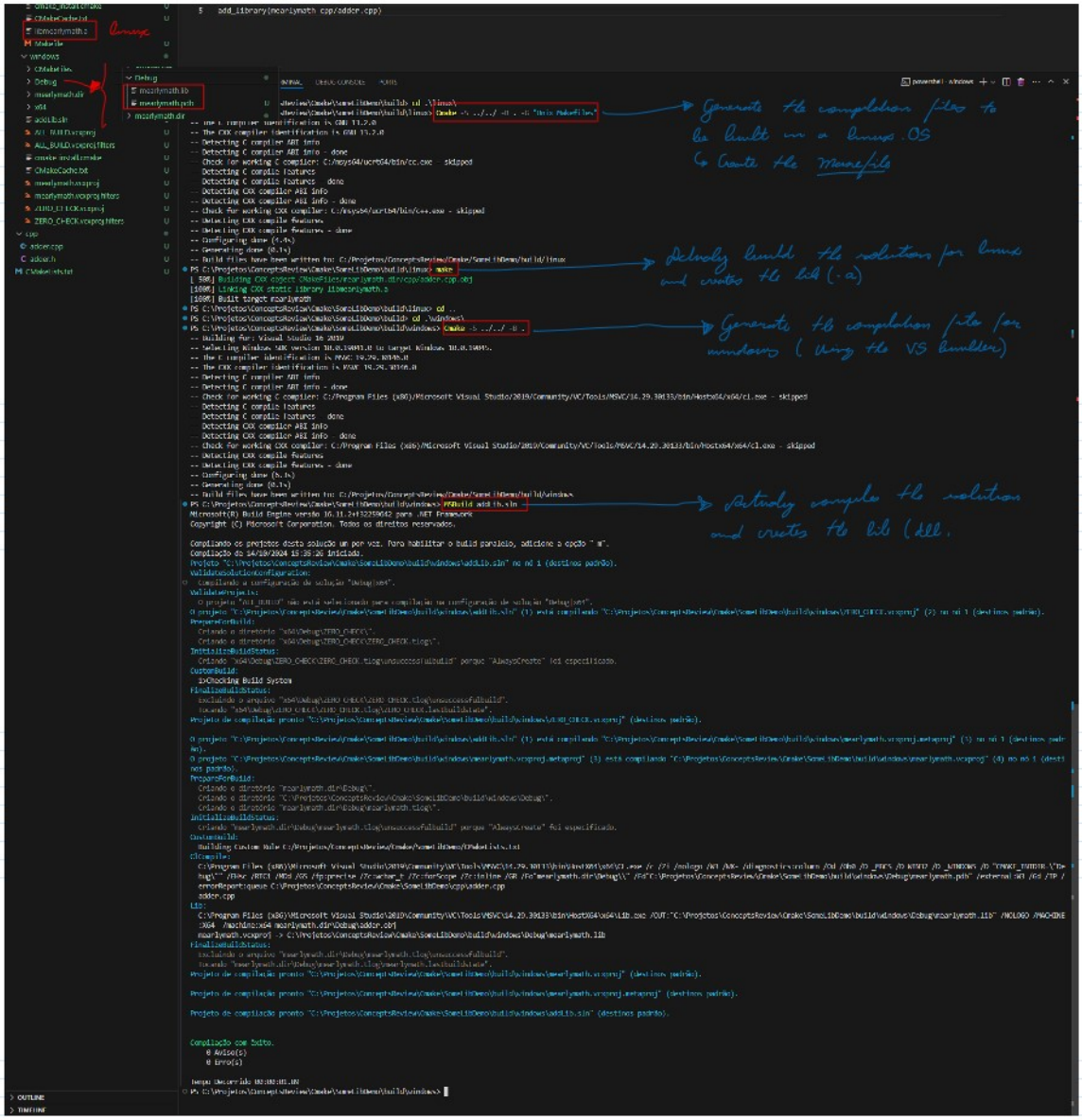
↳ Ok, it's pretty cool → Can be built for linux on windows with similar commands

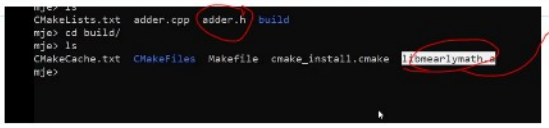
↳ Define the CMakeList + the cpp implementation files

↳ Same command to build it for linux (lines above)

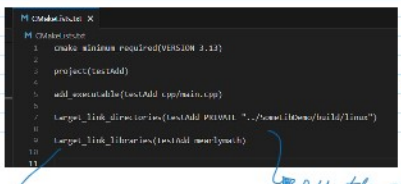
↳ MsBuild command for windows → Remember to add the compiles to the paths

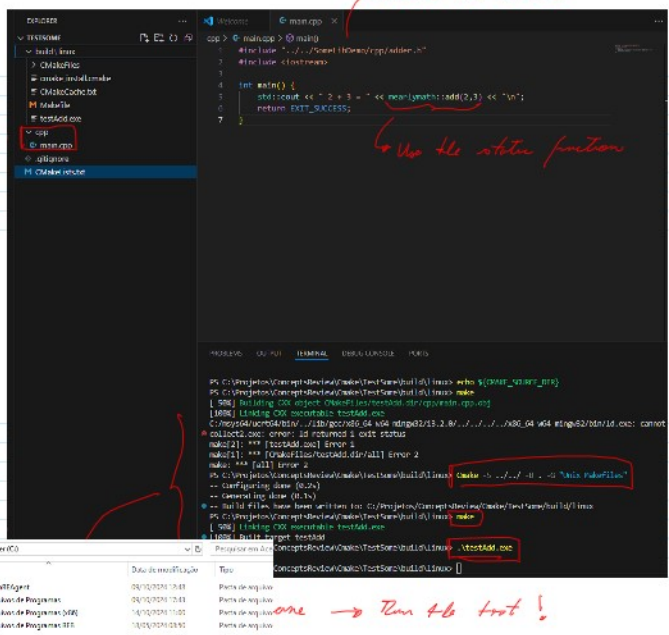


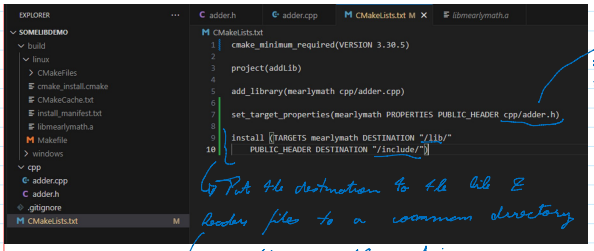


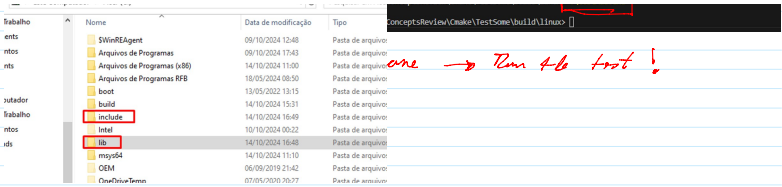


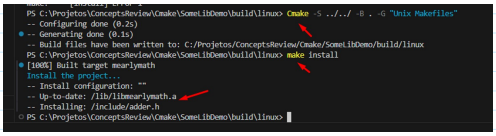
Need to include both!!











→ Need to rebuild every thing