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 Cpt instructions!

Why CMake? https://earthly.dev/blog/cmake-vs-make-diff/

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\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've 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

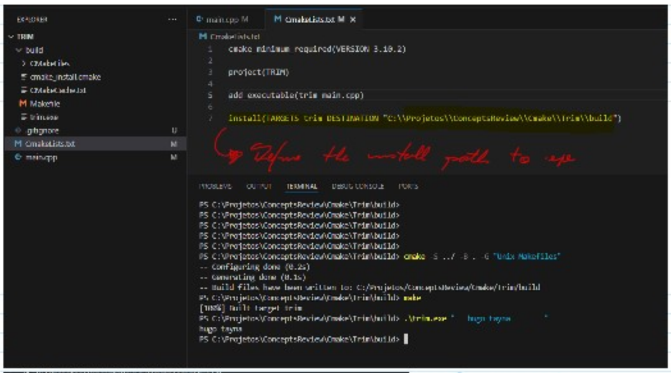
→ Run the executable

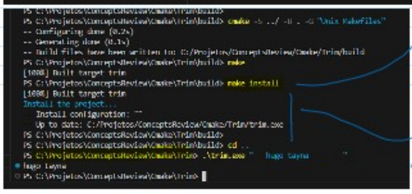
Generate a Build System

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 file that can generate the executable ones when builded
→ focus on a cross platform app





→ This can put the executable some where else in the project, just change the path!

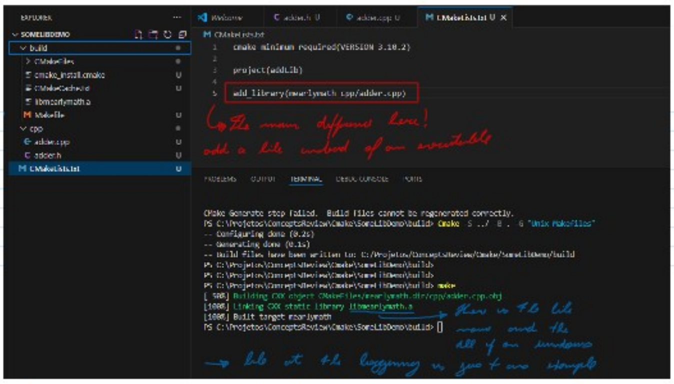
→ Configure the correct path

Libraries

Static Lib

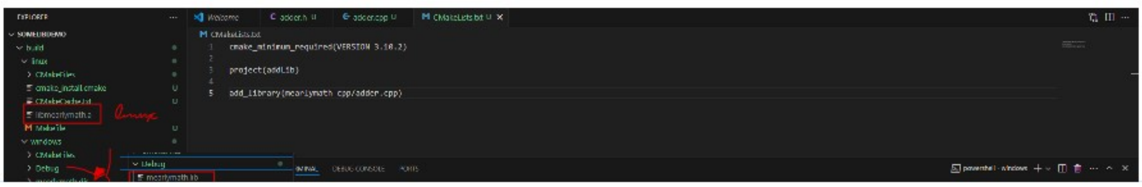
Name Space

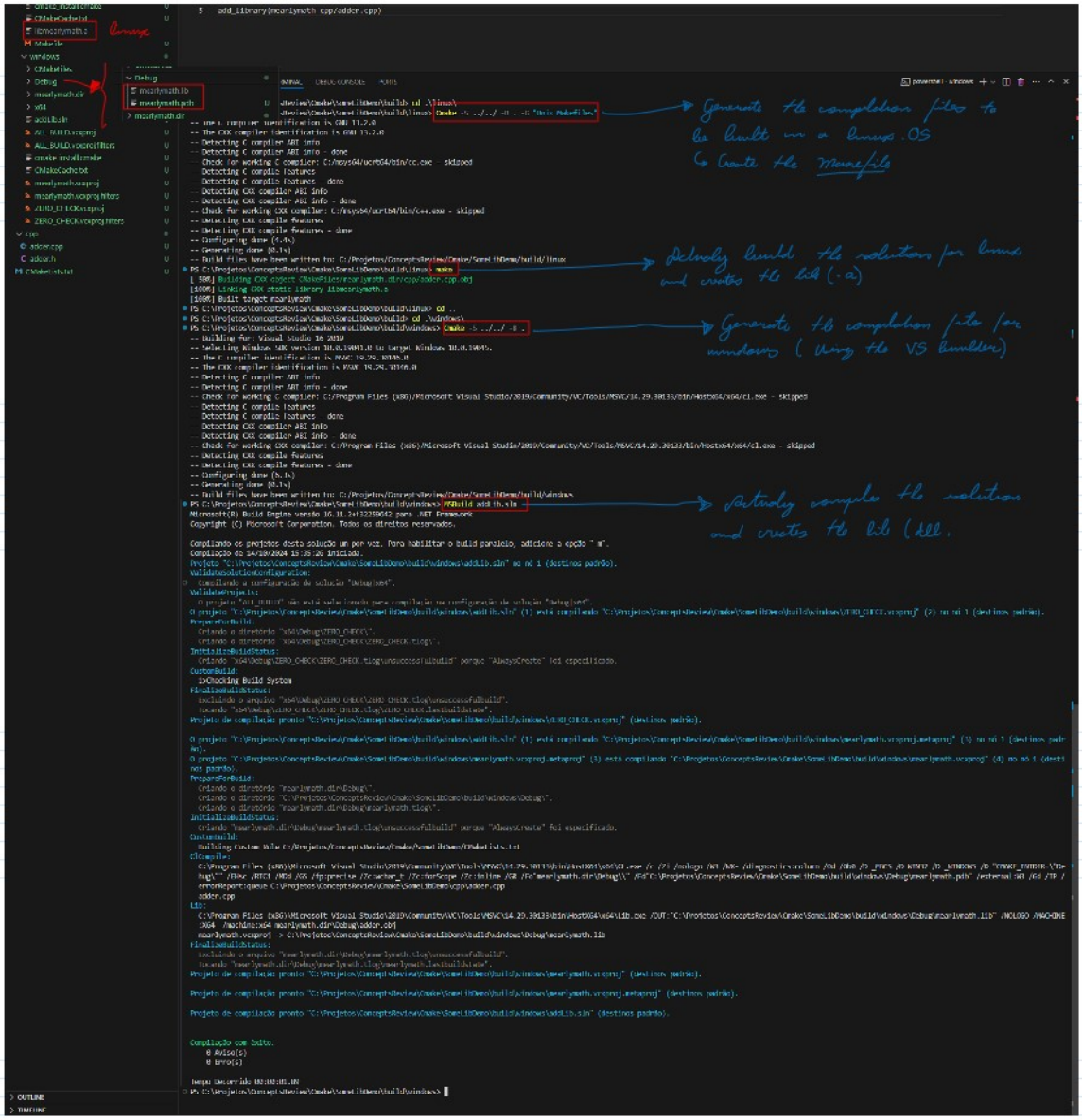
→ Do the C++ code first
→ No need a main because libs don't have an entry, just functions
→ Created a simple example with an overload and 2 static methods

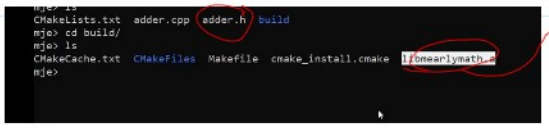


→ And, It's pretty cool → Can be built for linux and windows with similar commands

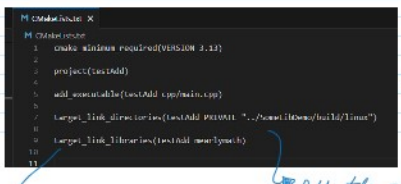
→ Define the CMakeList + the cpp implementation files
→ Make command to build it for linux (like above)
→ MsBuild command for windows → Remember to add the compiler to the paths

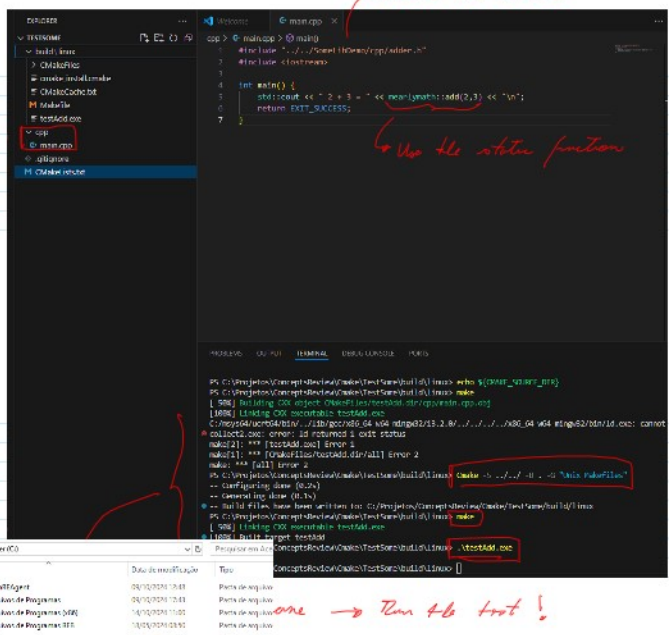


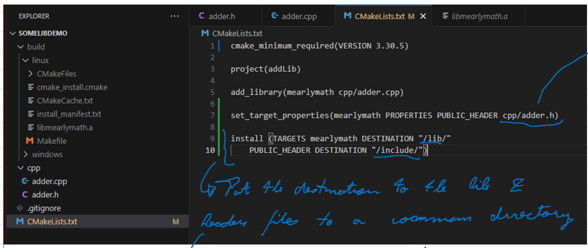




Need to include both!!



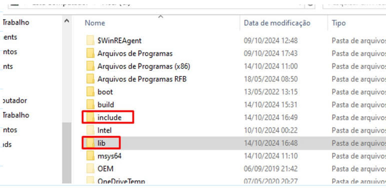


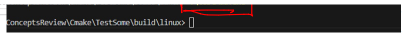


→ Set the target project to find the header with public access

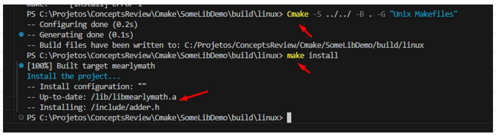
↳ Put the destination to the lib & header files to a common directory

↳ in this case the C drive

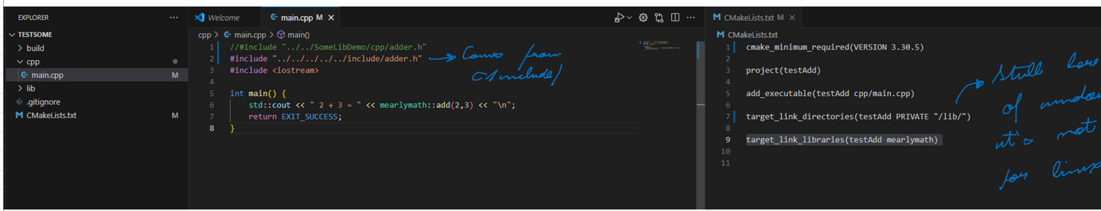




→ Run the test!



→ Need to rebuild every thing



→ Comes from C:/include/

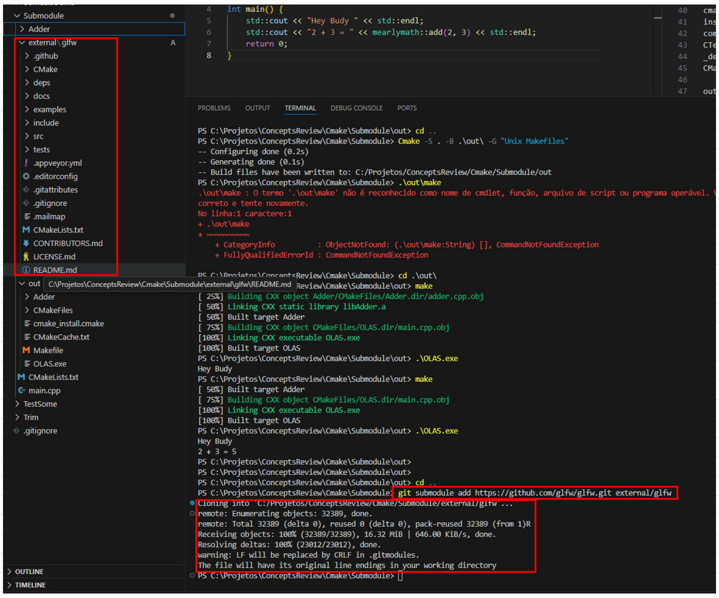
→ Still here because of windows but it's not required for linux

→ Sub modules

↳ Integration with github (libs based on any repos)

command -> git submodule add https://github.com/glfw/glfw.git external/glfw

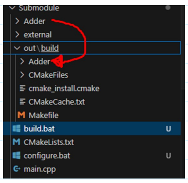
→ This is a git feature to bring libs into the project

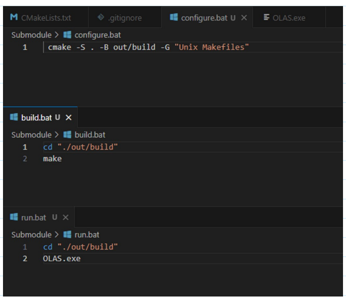


It clones the repo into one of the folders following that command

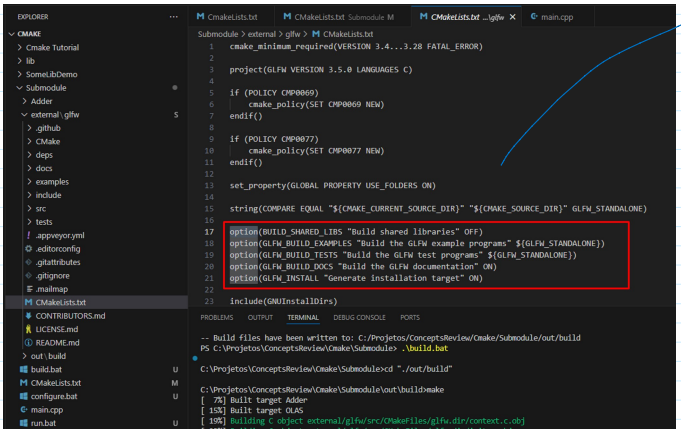
↳ Git submodule add <Repo link> <folder to past>

↳ The up level lib directory is copied to the solution after the build. So, it actually build the lib together with the source code!





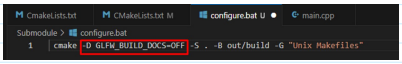
→ Can also create bat files to make the process easier and not type the commands all the time



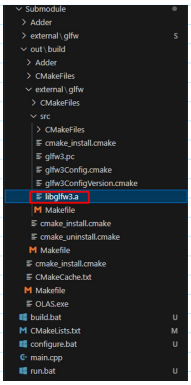
→ OPTIONS

features we can turn on and of on config or run time!

↳ Add 4 to configuration to the Cmake command line bellow



↳ If don't wants everything can turn it off to more lighter



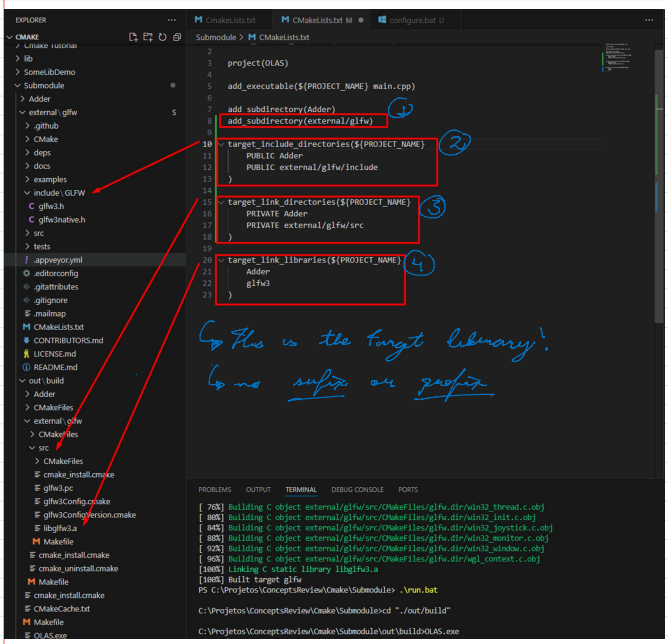
→ The lib is now created in the build folder but we need to include it to use

(1) Include the subdirectory until the submodule into the source folder → Using git command

(2) Directory for the include files!

↳ Still in the solution → Just point to the headers

(3) This one is actually pointing to the output build folder ↳ Where the linker find the lib



↳ This is the target library!

↳ no suffix or prefix

→ Add Versioning

CMake Tutorial EP 4 | Versioning Source Code

Version control in the Cmake files.... seams to be a mass if configuring projects like this... i'll not extend my self here

In fact, it is realy cool... but for a project... not right now

CMake Tutorial EP 5 | Making Libs Optional

CMake Tutorial EP 5 | Making Libs Optional

→ Depending on the project, there is no need for a lib

↳ Create an option for that

https://github.com/HugorTorquato/ConceptsReview/commit/ec6c4f8a155a1d552964a940a4f6162101a9e6c6

↳ Commit with the changes

CMake Tutorial EP 6 | Installing Your Software! (part 1/2 of install)

CMake Tutorial EP 6 | Installing Your Software! (part 1/2 of install)

Way to install the program.... not sure if i need right now but let's keep it

Packaging everything up

CMake Tutorial EP 7 | Installing With CPack! (part 2/2 of install)

find_library(...)

Different ways to import libs

CMake Tutorial EP 8 | find_library(...) (part 1/2 of find lib)

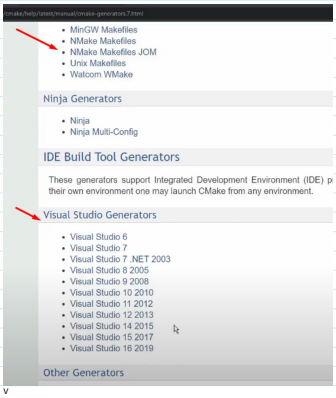
Running CMake on Windows (like linux)

CMake Tutorial EP 10 | Running CMake on Windows (like linux) (Over-explained)

- Add things to PATH variables
  - Cmake
  - git
- Compiler options
  - GCC
    - Can install WSL
    - MinGW
    - VS Generator
    - Controlled by -G

Cmake - VSCode MSBuild example

```
PS C:\Projetos\ConceptsReview\Cmake\test> cmake -S . -B .\out\build -G "Visual Studio 16 2019"
-- Selecting Windows SDK version 10.0.19041.0 to target Windows 10.0.19045.
-- The C compiler identification is MSVC 19.29.30146.0
-- The CXX compiler identification is MSVC 19.29.30146.0
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- Check for working C compiler: C:/Program Files (x86)/Microsoft Visual Studio/2019/Community/VC/Tools/MSVC/14.29.30133/bin/Hostx64/x64/cl.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:/Program Files (x86)/Microsoft Visual Studio/2019/Community/VC/Tools/MSVC/14.29.30133/bin/Hostx64/x64/cl.exe - skipped
-- Detecting CXX compile features
-- Detecting CXX compile features - done
-- Configuring done (6.5s)
-- Generating done (0.1s)
-- Build files have been written to: C:/Projetos/ConceptsReview/Cmake/test/out/build
PS C:\Projetos\ConceptsReview\Cmake\test>
```



Can use any of the generators to create the build but it doesn't build it

Just create the project based on the selected version

```
PS C:\Projetos\ConceptsReview\Cmake\test> cd out
PS C:\Projetos\ConceptsReview\Cmake\test\out> cd build
PS C:\Projetos\ConceptsReview\Cmake\test\out\build> MSBuild.exe .\test.sln
```

```
    Tempo Decorrido 00:00:00.81
    PS C:\Projetos\ConceptsReview\Cmake\test\out\build> cd .\Debug\
    PS C:\Projetos\ConceptsReview\Cmake\test\out\build\Debug> .\test.exe
Hey dude
    PS C:\Projetos\ConceptsReview\Cmake\test\out\build\Debug>
```

V

msbuild.exe ->

- Required to add in the path folder
  - C:\Program Files (x86)\Microsoft Visual Studio\2019\Community\MSBuild\Current\Bin
- nOW CAN RUN THE msbuild.EXE .\TEST.SLN COMMAND IN THE OUTPUT FOLDER TO BUILD IT
- And than execute the command to run the executable file

VSCODE | VCPKG Libraries | Project Setup Basics

Libs
- submodule
- manually and full

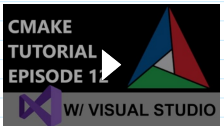
+ VCPKG

CMake Tutorial EP 11 | VSCODE | VCPKG Libraries | Project Setup Basics

Visual Studio (OpenGL & Vulkan examples)

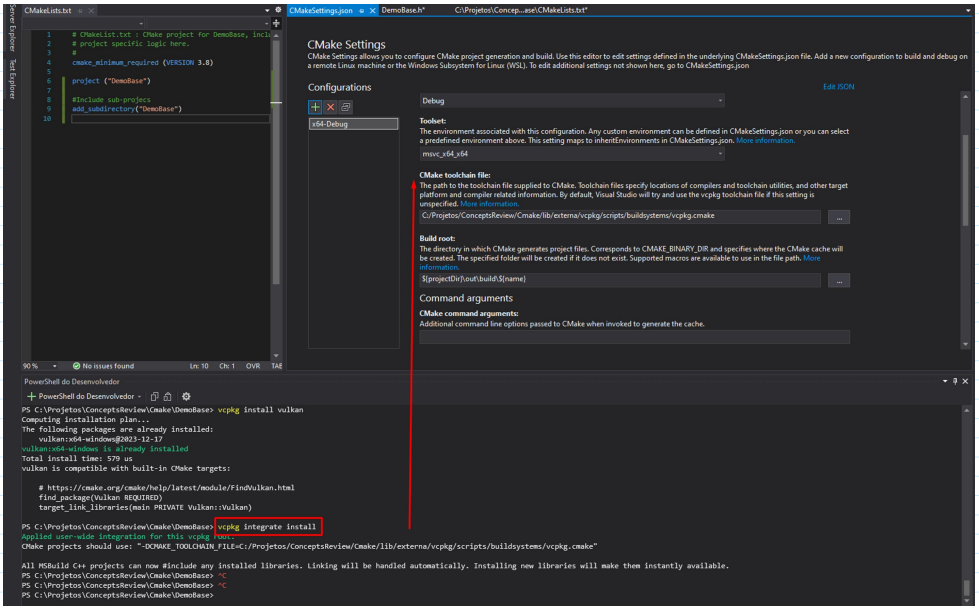
To open libs with VS

CMake Tutorial EP 12 | Visual Studio (OpenGL & Vulkan examples)



- Create new project

- Type "Cmake Project"
  - Install the VCPKG to install libs https://learn.microsoft.com/en-us/vcpkg/get_started/get-started-msbuild?pivots=shell-powershell
  - vcpkg is a free and open-source C/C++ package manager maintained by Microsoft and the C++ community.
- Install a lib with VCPKG
  - VCPKG install vulkan
    - df



CTest

CMake Tutorial EP 13 | CTest