Pixida GmbH

 Version 1.0

Understanding Basics of CMAKE

Understanding CMake

*CMake is an open-source, cross-platform family of tools designed to build, test and package software. CMake is used to control the software compilation process using simple platform and compiler independent configuration files, and generate native makefiles and workspaces that can be used in the compiler environment of your choice.*

* Building/running of normal programming (practice) application is totally different than and professional applications in companies or different organizations.

* There are number of build systems available in market for C/C++ projects or other languages.

e.g., Make, Cmake, Autoconf, Premake, Ninja, SCons etc.

* The biggest problem working with C/C++ application development is it's build system. Also, this is one of the bottlenecks, where lot of developers don't stay long with C/C++ application development, and they look for better and easy build system, which they only get in other programming languages like Rust, Java, C# etc not in C/C++.

* Also, other pain in C/C++ application development is they have lot of options for build system. Choosing one build system among many is painful and tough.

* Even after they choose, if project demands to migrate to other build system, it's a lot painful.

* To handle these above problems, we have CMake, build system generator.
* In simple words, CMake is not a build system rather it’s generator of build system.

* You can choose the build system which you want to generate.

* Learning CMake is not as simple as we hear, because we think too much about compiling, linking, and including.

* In CMake, you can concentrate on organizing your folder structure rather than bothering about your build system and how to edit it. that is the beauty of CMake.
* CMake reads the projects to build from CMakeLists.txt files written in a language of its own. From there it generates some Makefiles (or equivalent project files for Xcode or Visual Studio for example) which later can be used to build the project.
* CMake is quite fast (not the fastest though) but it perfectly handles the rules for rebuilding the targets.
* This tool is quite well conceived and very popular in C++ world specifically.
* Knowing CMake and writing CMakeLists.txt is to learn a one more specific language to use it.

**All C/C++ application program has three major things:**

1. How to create executable, shared library and static library

2. How to supply library to consumers

3. How to consume libraries from others

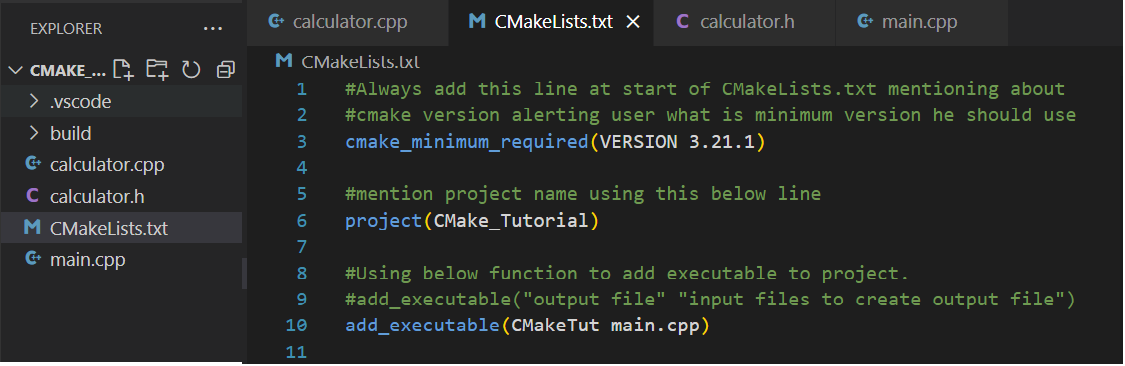
All these can be addressed easily with CMake irrespective of build system.

Let's explore and dig deeper to understand basic concepts of CMake.

1) Create simple C++ project, in this example we have created 'calculator.h' 'calculator.cpp' along with main.cpp. and it's a plain calculator which performs addition of two integer numbers.

2) Create 'CMakeLists.txt' file at same level of your main.cpp.

3) Follow below steps to write 'CMakeLists.txt' and try to perform same on your own with your own project.

Screenshot 1: Simple CMakeLists.txt file

See in comments what each line means in CMakeLists.txt and what’s it’s importance.

So now as we have CMakeLists.txt file ready with us along with our project, how do we execute it to build and run the project?

By executing it we create a build system that will build the project and generate an output for us. Let's see how:

5) Now there are two ways to create a build system.

5.1) By creating build folder manually:

a) Create a folder named 'build' in your project and change your directory to that build folder on terminal.

b) Then type a command *'cmake ..*' to generate the build files for respective build system (on Linux by default it is 'make' which creates respective build files) in your build folder.

c) Once that is successful simply type *'make'* to build the project which on success creates output file in your 'build' folder itself.

d) Execute the output file to get the output.

Please see below screenshot for more details and follow the same to build your project for better understanding.A screenshot of a computer program

Description automatically generated

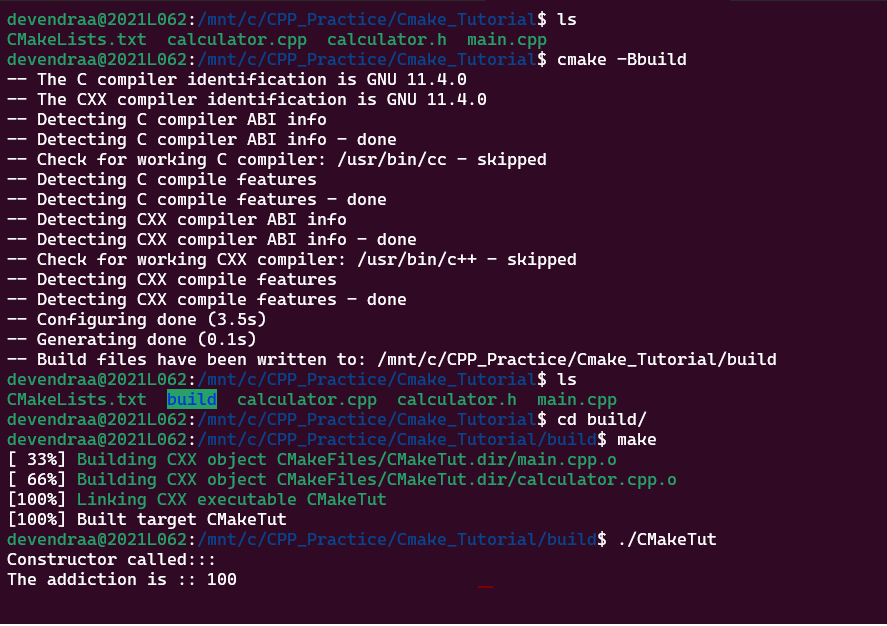
Screenshot 2: Creating build folder manually and run project.

5.2) Creating build folder with command:

a) Another way of creating build folder and it's contents is with

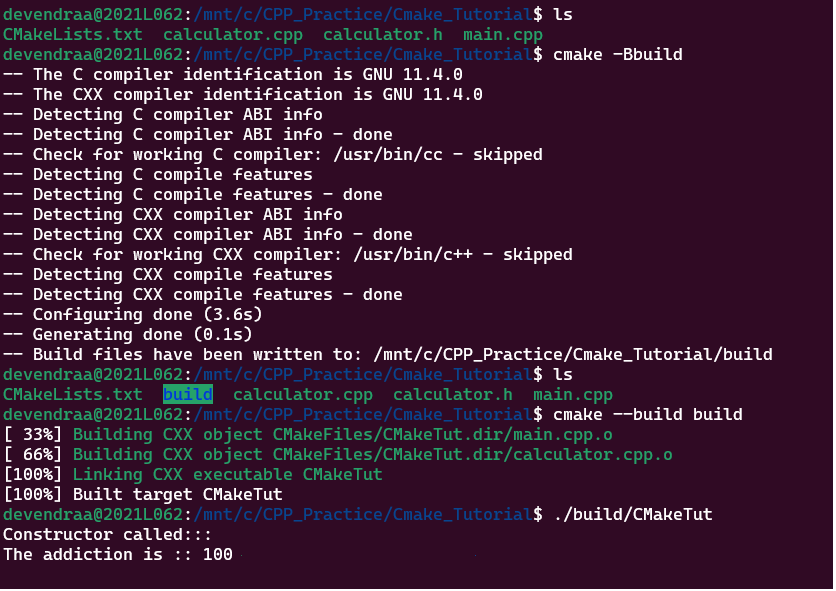
'*cmake -Bbuild*' command on terminal

b) Above command creates build folder automatically and then you can either go to build folder and run the executable as mentioned above. (See In below screenshot)



Screenshot 3.1: Without creating build folder

c) or you can directly execute *'cmake --build build*' command which will generate executable file for you. (Check screenshots below)



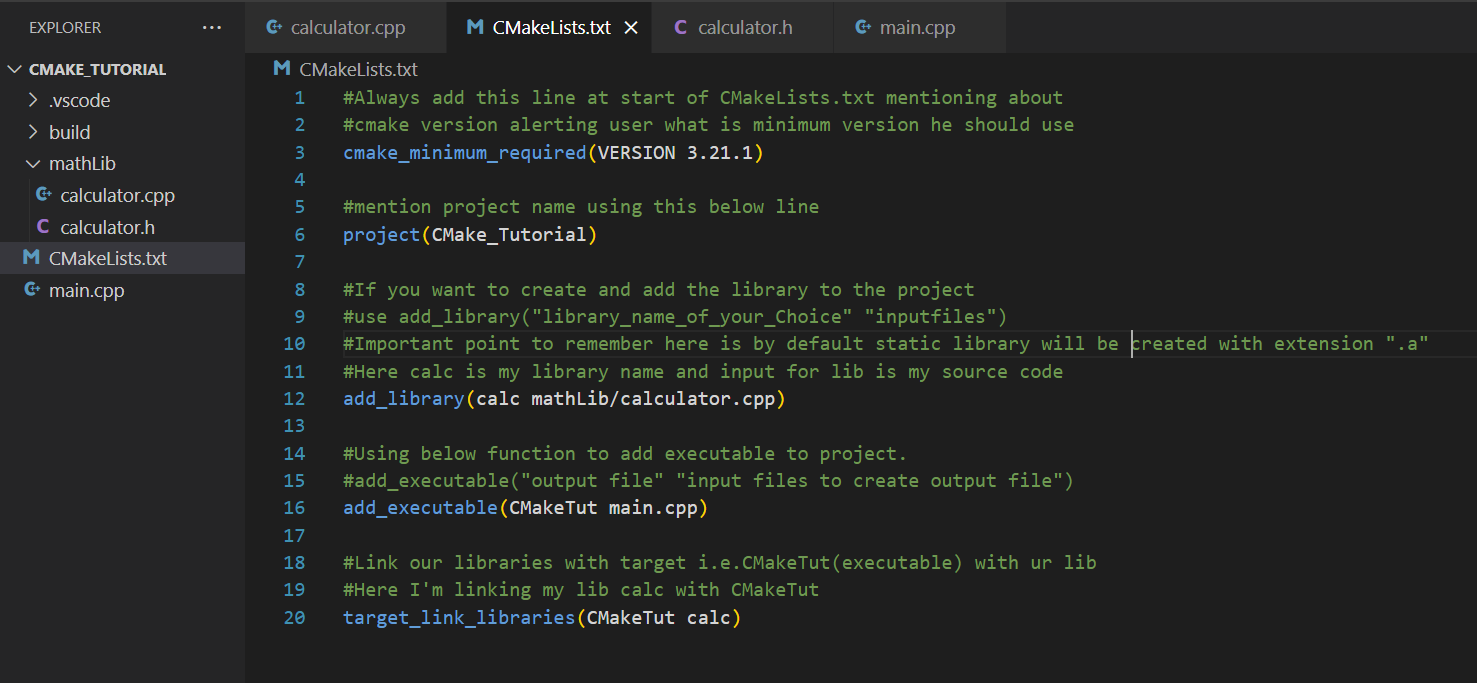
Screenshot 3.2: Without creating and going into build folder.

6) Now as we understand the basic concept of creating build system let's move one step and create our own library instead of using plain source code.

For that, create a folder named 'mathLib' in project and copy your header file and source file into that folder. So now we have our library folder 'mathLib' Is ready with the source code. Let's work on this to create your own library using CMakeLists.txt.

we use add\_library() to add the library Into project and we link that library with our target.

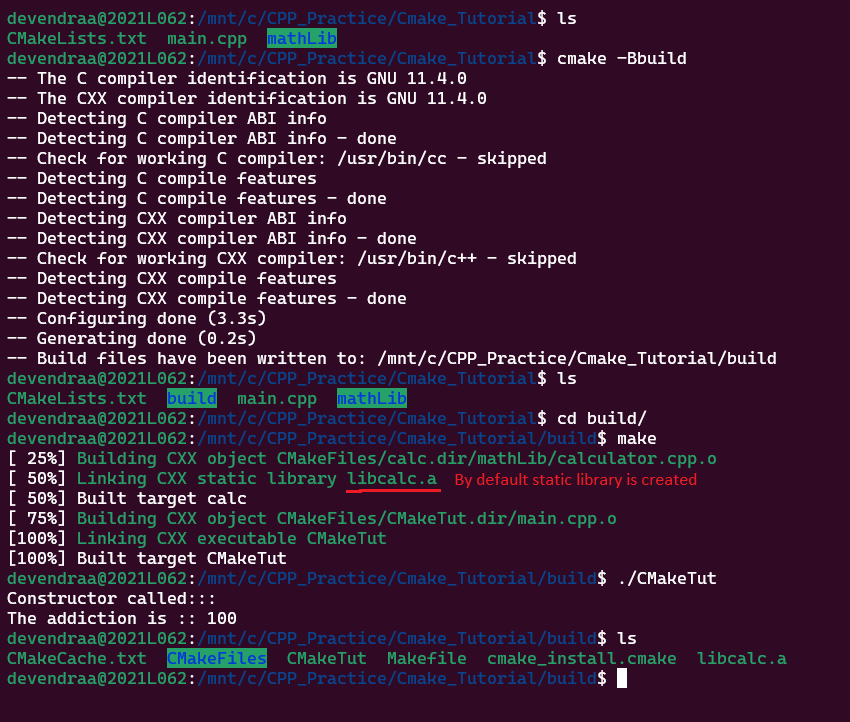
Check below screenshot for more details to understand:





Screenshot 4: Creating my own library using CMakeLists.txt.

Execution or building for above code then would be same as we have seen earlier. Check below screenshot for created library.



Screenshot 5: Created my own library using CMakeLists.txt.

7) Now suppose, user don't want to create the static library due to its disadvantages and want to go ahead with SHARED library for his project. Developer just must add SHARED keyword in CMakeLists.txt to make it a shared library. Check below screenshot for details:

A screenshot of a computer program

Description automatically generated



Screenshot 6: Creation of own SHARED Library

Now try to build and run the application same way we did earlier It should build and run with success.

Also, in build folder you will see the library with '.so' extension which Is shared library as you want.

See below screenshot for the details and follow the same to check:

A computer screen shot of a program

Description automatically generated

Screenshot 7: Own SHARED Library with .so is created.

8) Now, suppose developer wants to organize project more for handling professional way. We can create another folder inside library folder and put our library's header file(s) into it and give a call to include\_ accordingly.

See below screenshot for details:

A screenshot of a computer program

Description automatically generated



Screenshot 8: Separate folder for headers Inside library

Keep in mind that, we need to modify path for header inclusions in cpp files as the path now would be different than earlier path. So once that is done there won't be any error.

Probably, the building and execution process would be same as we have seen earlier. See below screenshot for the folder structure shown on terminal:

A screenshot of a computer program

Description automatically generated

Screenshot 9: Separate folder for headers Inside library on terminal

9) Further, suppose developer wants to create library using CMakeLists file separately and don't want to create it inside our main CMakeLists file. So for that developer can create another CMakelists file inside library folder and use add\_library() there and modify header inclusion path according to folder structure used.

Also In our main CMakeLists file we have to use add\_subdirectory() to tell that we are using this another CMakeLists file for making library.

See below screenshots for more details:

A screenshot of a computer program

Description automatically generated



Screenshot 10: Added new CMakeLists.txt to library and modified original CMakeLists.txt accordingly.

Check below screenshot for new CMakeLists file.

A screenshot of a computer program

Description automatically generated



Screenshot 11: Separated new CMakeLists.txt for Library

This way we can organize and play around with CMakeLists file for project structure.

Please check on your respective system by making a small project like this and follow all details provided in document to make sure you understand this very basic exercise thoroughly for CMake.

It's important utility for generating build systems.