**GCC C-MAKE FRAMEWORK**

Author : Long Dao | <https://louisvn.com>

1. **Introduce**

* GCC C-Make Framework is designed to manage C/C++ software projects.
* Allows users to create, delete, compile, run, and export test reports for any project.
* To manage a project, I need you to install some of the following software to use the framework.
  + [**Toolchain GCC**](https://winlibs.com/). This is a compiler that takes care of the process of translating from C code to machine code so that the program you write runs on your PC. Download **GCC version 13.2.0 (with POSIX threads - UCRT) 32bit** for environmental consistency.
  + Makefile. This is a replacement for typing complex commands into simple commands on your PC's Terminal control window.
  + [**Cygwin**](https://www.cygwin.com/install.html). If you are a Windows user, you need to install this tool to emulate a Linux environment to make Makefile and other tools work stably.
  + [**VSCode**](https://code.visualstudio.com/download). This is extremely popular word processing software when you write code. It helps you display colors, command prompt when writing, supports debugging configuration, ...
  + [**Python**](https://www.python.org/downloads/). In Python, [**Gcovr**](https://gcovr.com/en/stable/) tool allows you to generate code coverage reports.

1. **Install GCC**

* To install GCC, go to the following URL (Github): [**Click here**](https://github.com/brechtsanders/winlibs_mingw/releases/download/13.2.0-16.0.6-11.0.0-ucrt-r1/winlibs-i686-posix-dwarf-gcc-13.2.0-llvm-16.0.6-mingw-w64ucrt-11.0.0-r1.zip)
* After the download is complete, press **Window + E**. At the File Manager window, go to drive **C**, create a folder called "**Toolchain**”.
* Then extract the downloaded file to the newly created folder. After extracting, you will have a path:“**C:\Toolchain\mingw32**”.
* Click **on Window** and search for "**environment**". You'll see the following app:



* Tap select the app. Select the **Advanced** tab > **Environment variables…**
* A new window appears. At the **System variables** table> **Path** > **Edit…**
* A new window appears. Click **New** to add the following paths:
  + **C:\Toolchain\mingw32\bin**
  + **C:\Toolchain\mingw32\i686-w64-mingw32\bin**
* Once the addition is complete, press **OK** one by one to exit.
* Finished! To check if you have installed successfully, press **Window** and search for "**cmd**”



* Open the application, type **"gcc -v**" in the window. If you see a message like this, done.



1. **Install VSCode**

* To install VSCode, go to the following URL: [**Click here**](https://code.visualstudio.com/download)
* After downloading, click open setup file. Choose “**I accept the agreement**” > “**Create a desktop icon**” > **Next** > **Install**
* Wait for the installation to finish and it is successful!
* Next, you need to open the newly installed VSCode application to install the necessary tool packages.
* Press Ctrl  **+ Shift + X**. An "Extensions" tab pops up.
* Type in the search bar and download the following packages in turn:
  + **C/C++** ; **C/C++ Themes** ; **C/C++ Extension Pack**
  + **Code Runner**
  + **Doxygen Documentation Generator**

A screenshot of a computer program

Description automatically generated with medium confidence

A screenshot of a computer

Description automatically generated with medium confidence

A screenshot of a computer

Description automatically generated with medium confidence

* After the installation is complete, press the key combination [ **Ctrl + ,** ] to open the "Settings” tab.
* Type "**Code-runner: Run In Terminal**" into the search bar. Then click on the square according to the picture below:



1. **Install Cygwin**

* To install Cygwin, go to the following URL: [**Click her**](https://www.cygwin.com/setup-x86_64.exe)**e**
* Same step 2 of GCC installation, Go to drive **C**, create a folder that is “**Cygwin**”.
* Next, copy the downloaded setup file into the folder you just created: **C:\Cygwin**
* Click open the copied app. Choose **Next** > **Install from Internet**
* In the new window, enter **"C:\Cygwin**" in "**Root Directory**". At the bottom select **All Users** > **Next**
* In the new window, enter “**C:\Cygwin**” to “**Local Package Directory**” > **Next**
* Select Use System Proxy **Settings** > **Next**
* Click on any URL to download the data. (Should choose the 4th URL) > **Next**
* Wait a while until a "**Select Packages**" window appears. At **View** select **Full**
* Search for the following packages and select the latest versions : **make**, **zip**, **unzip**, **sed**, **cygrunsrv, bc, tree**

Note, click on the square with the down arrow to select the version. For example, the following figure is shown:

A screenshot of a computer

Description automatically generated with medium confidence

* After selecting the installation version for all the packages listed above, click  **Next**  > **Next** to proceed with the installation.
* Wait until the installation is complete, click **Finish**.
* Then, you open "**Environments**" similar to step 4+ of the GCC installation and add the following paths:
  + **C:\Cygwin\bin**
  + **C:\Cygwin\sbin**
  + **C:\Cygwin\usr\sbin**
* After adding Environments, **right-click** on the newly installed application icon. Select **"Run as administrator**" and wait for the application to show up.
* Type in "**cygserver-config**" > **Enter** >... Type "**yes**". Wait for the run to finish and close the window.
* **Note** : If you have Git installed on your PC. Please move Cygwin's paths above Git's paths in Enviroments.
* Restart your PC - required.

1. **Install Python**

* To install Python, go to the following URL: [**Click her**](https://www.python.org/downloads/)**e**
* Click the button "**Download Python ...**" to download the app.
* Press **Window + E**. In the File Manager window, go to drive **C**, create nested folders as follows: "**Toolchain > Python > Python311**". (Replace "311" as your version of Python).
* After downloading, Click open the application. At the settings window, select "**Customize installation**". Next, click **all** the options in "**Optional Features**", especially "**Pip**". Then click "**Next**".
* In the "**Advanced Options**" window, also click all options. Then in the "**Customize install location**" section, enter the path to the previously created folder **"C:\Toolchain\Python\Python311**" and select "**Install**".
* Wait for the installation to complete and click “**Close**”.
* After the installation is complete, open **"Path**" under "**Eviroments**". If you don't see the same 2 paths placed at the top (as shown below), copy the exact path on your PC and add it (move up to the top).

A screenshot of a computer program

Description automatically generated

* Then, open "**cmd**" and type **"python --version**" to check the version of Python. If you have version information printed, you have successfully installed it.

A black text on a white background

Description automatically generated

* Next, at the "**cmd**" window, enter **"pip install gcovr" to install** the Gcovr **tool**. Wait until the tool is successfully installed, you will see the text "Successfully installed gcovr-...". Type **"gcovr --version**" to check its version.



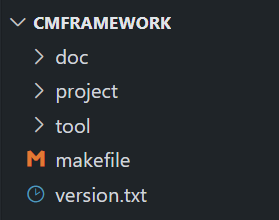
* Python version **3.11.4** and **Gcovr version 6.0** are running stably on this framework.

1. **Using the Framework**

* You need to download and unzip the Framework before working with it.
* To use the Framework, first open the **VSCode** editor.
* Để thêm Framework vào trình soạn thảo, nhấn liên tục 2 tổ hợp **Ctrl + K** và **Ctrl + O**.
* An "**Open Folder" window** will appear, select the path to the previously **extracted "CMFramework**" folder. Then select "**Select Folder**”.
* If a window similar to the one below appears,  **click** on the box and click "**Yes**”.



* You will see in the "**Explorer**" tab showing all the newly added frameworks.
* **The structure of the framework**



* + **doc** : Store documents for common use for every project. This "**Readme.en.docx**" file is also located in this folder.
  + **project** : Manage all your projects. Note, I have provided a sample project "**~temp**" as a foundation for creating other projects. You force not to delete it at all costs.
  + **tool** : Includes common libraries and test report generation tools, …
  + **makefile** : this is a file that manages all the backbone features to ensure the Framework works. You also **cannot** customize this file.
  + **version.txt** : this file records the history of the current and version versions of the Framework.
* **The structure of the project in the framework**



Take the example on the template project "**~temp**" provided. Note that you **may not** edit or delete this project. You need to create a new project and execute on it.

* + **doc** : store of documents of this project.
  + **inc** : contains project header files (**.h**)
  + **dev** : contains project source files developed to test code coverage (**.c .cpp**)
  + **src** : contains the source files of the project (**.c .cpp .o**)
  + **user\_cfg.mk** : A makefile allows users to configurations for each specific project.
* **Use make commands**

First, open a Terminal window on VSCode. Go to **Terminal** > **New Terminal**

A screenshot of a phone

Description automatically generated with medium confidence

At the **TERMINAL tab**, enter your requirements. Note, the root path on the terminal should be the path to "**CMFramework**". If this is not the case, use the command **"cd**" to move to.

* + **make setup** : initialize the tools, this command should run 1 time when you first use this framework. It will create executables inside the "**tool/bin"** directory.
  + **make info** : used to print to the screen some information about the project.
  + **make** | **make all** : These 2 commands allow running {**make build run**}, see below.
  + **make clean** : Used to delete the output folder, where object files and executable files are stored. It also deletes test reports (if any).
  + **make build** : used to compile files (.**c .cpp .cc**) to object files (**.o**), then link it to an executable file (**.exe**).
  + **make run** : used to run executable files (**.exe**) on Terminal.
  + **make report** : used to generate test reports if you use the "**utest.h**" library (available in the **tool**) in the C program to write test cases.
  + **make pack** : used to package the project, allowing you can share it with anyone.
  + **make vsinit** : used to create configuration files in VSCode, helping the software correctly link file paths on the display interface. When you move to another project, this command is **automatically** run. You just need to run this command again when you **update** the path in makefile (**user\_cfg.mk**).
  + **make move** : used to move to any project. Automatically add new ones if the project doesn't exist.
    - For example, **make move.proj1** : will mean moving to the project "proj1”.
    - Or: **make move.Group1: proj1**: move to project "Group1/proj1". You can nest in multiple groups, using the "**:**" to separate.
  + **make remove** : used to delete any project. If you delete the current project, it will automatically switch back to the "**~temp"** template project. The usage is the same as the **"move**" command.
  + **make import** : used to add a new project from a zip file shared fromsomeone else's "pack" command. The usage is the same as the **"move**" command. However, the variable "ZIP" will indicate the path to the zip file.
    - Example : **make** **import.proj1 ZIP=path/to/file.zip**
    - Note: the variable "ZIP" must be written adjacent to the "=" and path. Don't use white space in the middle.
  + **make list** : used to list all the files and folders that are in the project.
  + **make print** : used to print out the values of variables used inside makefile.
    - Example : **make print.VAR1.VAR2** : Print out the value of 2 variables : **VAR1** and **VAR2**.
    - Similarly, if printed multiple variables will be separated by dots.
    - This command is only for use by Admins or Framework developers.
* **Notes**
  + All paths and file names must be written continuously. That is, there is no white space in the middle. You should be mindful when naming any one file or folder.
  + It is not allowed to name source files or header files that coincide with the name of the project. This may cause conflicts when creating output files.
  + In a project, each filename is unique across all directories. You are not allowed to compile 2 identical filenames as it may also have a conflict at the output.

1. **Instructions for using the tools**

* **Use a code testing tool (utest)**
  + To use this tool, you first need to add the path of this library to the makefile.
  + Open the file "**user\_cfg.mk**". In the variable "**SRC\_DIRS**" used to scan and retrieve source code files (.c) and the variable "**INC\_DIRS**" used to scan and retrieve header files (.h), you need to add the following path to both variables: **"$(TOOL\_DIRS)/lib**".

A screenshot of a computer program

Description automatically generated

Specifically, **$(TOOL\_DIRS)** is the path to the "**tools**" directory of the Framework. Then **"/lib**" is for accessing the "**lib**" directory inside. The **"\**" sign to join the line below. You can write values that are the same line and separated by "**spaces**”.

* + After adding the path, you run the "**make vsinit**" command to update the path into VSCode, thereby being able to prompt you better.
  + Go back to your source file, just include "**utest.h**" and experience.

A screen shot of a computer

Description automatically generated

* + To run tests, you need to create test functions in the given format.

Syntax: **FuncTest( name\_test ) { /\* body \*/ }**

In particular, **"name\_test**" is the name of the function specified by yourself. The upper jaw is also equivalent to :  **void name\_test ( void ) { /\* body \*/ }** . However, writing the correct syntax will make it easier for readers to understand.

A screenshot of a computer

Description automatically generated

In the example above, we have a test function called "**Test\_001**”.

* + The purpose of testing is created to check the correctness of the source code. So, to know if your testing is right or wrong, you need to have the test conditions.
  + For example, you want to prove that "1+1=2" and "2+2=4" ... Those are the test conditions. Then, "**UT\_Assert**" is the command that helps you check the conditions.

A screenshot of a computer code

Description automatically generated

In the example above, I have 3 conditions to test. As long as one of the conditions is wrong, the whole test function will result in failure. So we can say that "**Test\_001**" failed because the 3rd test condition was wrong.

* + After creating the test functions, you need to add these functions to a container to execute it.

A black background with white lines

Description automatically generated

The above 2 macros will create a container, you just need to add the test functions in the syntax: **UT\_AddTest( name\_test, "Brief for name\_test" )**

Where "**name\_test**" is the name of the test function that was created earlier."**Brief for name\_test**" is a title you set yourself.

A computer screen shot of a message

Description automatically generated

If you have multiple test functions, you can simply add more **"UT\_AddTest**" commands. Note that there is no "**;**" after these commands. At the same time, this container is not written in any one function. Tests that don't currently want to execute, just discard or note the inside of the container.

* + After adding to the container, you need to make sure that the test functions are declared before using it.

A black background with white lines

Description automatically generated

* + To execute the tests inside the container one by one, you need to use the following command: **UT\_RunTests()**

A screenshot of a computer

Description automatically generated

* + Finally, you run the "**make**" compiler command and run the executable. Then, run **"make report**" to export the result to an html file and view it in a web browser. This html file is saved in the project**'s "**doc" folder.

A screenshot of a computer

Description automatically generated

A screenshot of a test

Description automatically generated

A screenshot of a computer code

Description automatically generated

As you can see, the 3rd Test Condition was wrong at line 55 in the source code file.

* + Good luck!
* **Use the code coverage measurement tool (CCOV)**
  + Code coverage is an indispensable tool for source code testing, it shows how much coverage of the testing process on the source code is.
  + To run the coverage code, go to "**user\_cfg.mk**", add the variable **"$(DEV\_DIR)**" to the paths of "**SRC\_DIRS**". That allows makefile to scan source files in the "**dev**" folder.

A screenshot of a computer screen

Description automatically generated

The "dev**" directory** is the directory used to measure CCOV. All functions inside all files in this directory are measured.

* + Next, fix the value of the variable **"RUN\_CCOV**" to "**on**" to enable the use of CCOV.

A close up of a report

Description automatically generated

* + After that, you need to create a source code file inside **"dev**" to write the functions to measure CCOV.

A screenshot of a computer program

Description automatically generated

The above example created a "**checkpos**" function to check for positive integers (n > 0) in the "**dev.c**" file located inside the "**dev**" folder.

A computer code with numbers and symbols

Description automatically generated

Then, I created 2 conditions so that it scans both the "**if**" and "**else**" branches inside the "**checkpos**" function. From there, my CCOV index will be completely covered.

* + Finally, you run "**make**" to compile and execute the source code. Then run **"make** **reports**" to generate reports including test results and CCOV. They are all stored in the project's "**doc**" folder.

A screen shot of a computer

Description automatically generated

A screenshot of a test

Description automatically generated

A table with numbers and text

Description automatically generated A close-up of a computer code

Description automatically generated

* + Good luck!

===================== **END** =====================

Author : Long Dao | <https://louisvn.com>