**GCC C-MAKE FRAMEWORK**

Tác giả : Long Dao | <https://louisvn.com>

1. **Giới thiệu**

* GCC C-Make Framework được thiết kế để quản lý các dự án phần mềm C/C++.
* Cho phép người dùng tạo, xóa, nhập, xuất, biên dịch, chạy và xuất báo cáo thử nghiệm cho một dự án bất kỳ.
* Để quản lý được một dự án, tôi cần bạn cài đặt một số phần mềm sau để sử dụng framework.
  + [**Toolchain GCC**](https://winlibs.com/). Đây là một trình biên dịch đảm nhiệm quá trình dịch từ mã C sang mã máy để chương trình bạn viết ra chạy được trên PC của bạn. Hãy tải xuống phiên bản **GCC 13.2.0 (with POSIX threads - UCRT) 32bit** để thống nhất về mặt môi trường (không bắt buộc).
  + **Makefile**. Đây là công cụ thay thế cho việc nhập liệu các câu lệnh phức tạp thành các câu lệnh đơn giản trên cửa sổ điều khiển Terminal của PC.
  + [**Cygwin**](https://www.cygwin.com/install.html). Nếu bạn là người dùng hệ điều hành Window, bạn cần cài đặt công cụ này để giả lập một môi trường Linux giúp cho Makefile và các công cụ khác hoạt động ổn định.
  + [**VSCode**](https://code.visualstudio.com/download). Đây là phần mềm soạn thảo văn bản cực kỳ phổ biến khi bạn viết code. Nó giúp bạn hiển thị màu sắc, nhắc lệnh khi viết, hỗ trợ cấu hình gỡ lỗi, …
  + [**Python**](https://www.python.org/downloads/). Trong Python, một công cụ [**Gcovr**](https://gcovr.com/en/stable/) cho phép bạn tạo bản báo cáo về độ bao phủ của mã nguồn (code coverage).

1. **Cài đặt GCC**

* Để cài đặt GCC, hãy truy cập đường dẫn sau (Github): [**Nhấn vào đây**](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)
* Sau khi tải xong, nhấn **Window + E**. Tại cửa sổ Quản lý tệp, vào ổ **C**, tạo một folder là “**Toolchain**”.
* Sau đó giải nén file vừa tải xuống vào folder vừa tạo. Sau khi giải nén, bạn sẽ có một đường dẫn : **C:\Toolchain\mingw32**
* Nhấn **Window + S** và tìm kiếm “**environment**”. Bạn sẽ thấy ứng dụng sau:



* Nhấn chọn ứng dụng. Chọn Tab **Advanced** > **Environment variables…**
* Một cửa sổ mới hiện lên. Tại bảng **System variables** > **Path** > **Edit…**
* Một cửa sổ mới hiện lên. Nhấn **New** và thêm vào các đường dẫn sau:
  + **C:\Toolchain\mingw32\bin**
  + **C:\Toolchain\mingw32\i686-w64-mingw32\bin**
* Sau khi thêm xong, hãy nhấn **OK** lần lượt để thoát.
* Xong! Để kiểm tra xem bạn đã cài đặt thành công hay chưa, hãy nhấn **Window + S** và tìm kiếm “**cmd**”



* Mở ứng dụng, nhập “**gcc -v**” vào cửa sổ. Nếu bạn thấy thông báo như sau là đã thành công.



1. **Cài đặt VSCode**

* Để cài đặt VSCode, hãy truy cập đường dẫn sau: [**Nhấn vào đây**](https://code.visualstudio.com/download)
* Sau khi tải xuống, nhấn mở file setup. Chọn “**I accept the agreement**” > Chọn “**Create a desktop icon**” > **Next** > **Install**
* Đợi cài đặt xong là đã thành công!
* Tiếp theo, bạn cần mở ứng dụng VSCode vừa cài lên để cài đặt các gói công cụ cần thiết.
* Nhấn tổ hợp phím **Ctrl + Shift + X**. Một tab “**Extensions**” hiện lên.
* Gõ vào thanh tìm kiếm và tải xuống lần lượt các gói sau:
  + **C/C++** ; **C/C++ Themes** ; **C/C++ Extension Pack**
  + **Code Runner**
  + **Doxygen Documentation Generator**
  + **Gdb Syntax**

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

A screenshot of a computer

Description automatically generated

* Sau khi cài đặt xong, nhấn tổ hợp phím [ **Ctrl + ,** ] để mở tab “**Settings**”.
* Nhập “**Code-runner: Run In Terminal**” vào thanh tìm kiếm. Sau đó nhấn chọn vào ô vuông theo hình bên dưới:



1. **Cài đặt Cygwin**

* Để cài đặt Cygwin, hãy truy cập đường dẫn sau: [**Nhấn vào đây**](https://www.cygwin.com/setup-x86_64.exe)
* Giống như bước **2** của cài đặt GCC, Bạn vào ổ **C**, tạo một folder là “**Cygwin**”.
* Tiếp theo, copy file setup vừa tải xuống vào folder vừa tạo: **C:\Cygwin**
* Nhấp mở ứng dụng vừa được copy. Chọn **Next** > **Install from Internet**
* Tại cửa sổ mới, nhập “**C:\Cygwin**” vào “**Root Directory**”. Phía dưới chọn **All Users** > **Next**
* Tại cửa sổ mới, nhập “**C:\Cygwin**” vào “**Local Package Directory**” > **Next**
* Chọn **Use System Proxy Settings** > **Next**
* Chọn vào 1 đường link bất kỳ để tải xuống dữ liệu. (Tôi thường chọn link thứ 4) > **Next**
* Đợi 1 lúc đến khi một cửa sổ “**Select Packages**” hiện ra. Tại **View** chọn **Full**
* Search các gói sau và chọn các phiên bản mới nhất : **make**, **zip**, **unzip**, **sed**, **cygrunsrv, bc, tree**

Lưu ý, nhấn vào ô vuông có mũi tên trỏ xuống để lựa chọn phiên bản. Ví dụ như hình sau:

A screenshot of a computer

Description automatically generated with medium confidence

* Sau khi chọn xong phiên bản cài đặt cho tất cả các gói được liệt kê phía trên, nhấn **Next** > **Next** để tiến hành cài đặt.
* Chờ đến khi cài đặt xong, nhấn **Finish**.
* Sau đó, bạn mở “**Environments**” tương tự như bước **4+** của cài đặt GCC và thêm vào các đường dẫn sau:
  + **C:\Cygwin\bin**
  + **C:\Cygwin\sbin**
  + **C:\Cygwin\usr\sbin**
* Sau khi thêm Environments, bạn **kích chuột phải** vào biểu tượng ứng dụng vừa cài đặt. Chọn “**Run as administrator**” và chờ cho ứng dụng dụng hiện lên.
* Nhập vào “**cygserver-config**” > **Enter** >… nhập “**yes**”. Chờ chạy xong và đóng cửa sổ.
* **Lưu ý** : Nếu bạn có cài đặt Git trên PC. Vui lòng di chuyển các đường dẫn của Cygwin lên phía trên các đường dẫn của Git trong Enviroments.
* Khởi động lại PC của bạn - bắt buộc.

1. **Cài đặt Python**

* Để cài đặt Python, hãy truy cập đường dẫn sau: [**Nhấn vào đây**](https://www.python.org/downloads/)
* Nhấn vào nút “**Download Python …**” để tải xuống ứng dụng.
* Nhấn **Window + E**. Tại cửa sổ Quản lý tệp, vào ổ **C**, tạo các folder lồng nhau như sau: “**Toolchain > Python > Python311**”. (Thay “**311**” thành phiên bản của Python của bạn).
* Sau khi tải xong, Click mở ứng dụng. Tại cửa sổ cài đặt, chọn “**Customize installation**”. Tiếp theo nhấn chọn **tất cả** các lựa chọn trong “**Optional Features**”, đặc biệt là “**Pip**”. Sau đó nhấn “**Next**”.
* Tại cửa sổ “**Advanced Options**” cũng nhấn chọn tất cả các lựa chọn. Sau đó, tại mục “**Customize install location**”, nhập vào đường dẫn đến folder đã tạo trước đó “**C:\Toolchain\ Python\ Python311**” rồi chọn “**Install**”.
* Chờ việc cài đặt hoàn tất và nhấn “**Close**”.
* Sau khi cài đặt xong, mở “**Path**” trong “**Eviroments**”. Nếu bạn không thấy 2 đường dẫn tương tự được đặt ở trên cùng (như hình bên dưới), hãy copy đường dẫn chính xác trên PC của bạn và thêm nó vào (di chuyển lên trên cùng).

A screenshot of a computer program

Description automatically generated

* Sau đó, mở “**cmd**” và nhập “**python --version**” để kiểm tra phiên bản của Python. Nếu có thông tin phiên bản được in ra tức là bạn đã cài đặt thành công.

A black text on a white background

Description automatically generated

* Tiếp theo, tại cửa sổ “**cmd**”, nhập “**pip install gcovr**” để cài đặt công cụ **Gcovr**. Chờ đến khi công cụ được cài đặt thành công, bạn sẽ thấy dòng chữ “Successfully installed gcovr-…”. Nhập “**gcovr --version**” để kiểm tra phiên bản của nó.



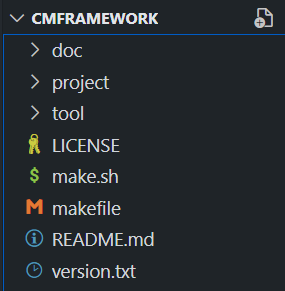
* Lưu ý. Phiên bản **Python 3.11.4** và phiên bản **Gcovr 6.0** đang chạy ổn định trên Framework này.

1. **Sử dụng Framework**

* Bạn cần tải và giải nén Framework trước khi làm việc với nó.
* Để sử dụng Framework, trước tiên hãy mở trình soạn thảo **VSCode** lên.
* Để thêm Framework vào trình soạn thảo, nhấn liên tục 2 tổ hợp **Ctrl + K** và **Ctrl + O**.
* Một cửa sổ “**Open Folder**” hiện lên, chọn đường dẫn đến folder “**CMFramework**” đã được giải nén trước đó. Sau đó chọn **Select Folder**
* [**Nếu**] một cửa sổ tương tự hình bên dưới hiện ra, hãy **nhấn chọn** vào ô vuông và nhấn “**Yes**”.



* Bạn sẽ nhìn thấy ở tab “**Explorer**” hiển thị toàn bộ các mục của Framework vừa được thêm vào.
* **Cấu trúc của Framework**



* + **doc** : lưu trữ tài liệu sử dụng chung cho mọi dự án. File “**Readme.vi.docx**” này cũng nằm trong thư mục này.
  + **project** : quản lý tất cả các dự án của bạn. Lưu ý, tôi đã cung cấp sẵn một dự án mẫu “**~temp**” làm nền tảng để tạo ra các dự án khác. Bạn buộc không được xóa nó. Nếu xóa, bạn sẽ không thể sử dụng bất kỳ một tính năng nào của Framework.
  + **tool** : bao gồm các thư viện chung, và các công cụ tạo báo cáo thử nghiệm, …
  + **makefile** : đây là tệp quản lý toàn bộ các tính năng xương sống nhằm đảm bảo Framework hoạt động. Bạn cũng **không** được tùy chỉnh file này.
  + **make.sh** : tệp này cho phép bạn chạy các lệnh **make** cho nhiều dự án cùng một lúc.
  + **version.txt** : file này ghi lại lịch sử các phiên bản và phiên bản hiện tại của Framework.
* **Cấu trúc của Project trong Framework**



Lấy ví dụ trên dự án mẫu “**~temp**” được cung cấp sẵn. Lưu ý bạn **không** được chỉnh sửa hay xóa bỏ dự án này. Bạn cần tạo dự án mới và thực thi trên nó.

* + **doc** : lưu trữ các tài liệu của dự án này.
  + **inc** : chứa các file header của dự án (**.h .hh .hpp**)
  + **dev** : chứa các file source của dự án được phát triển để thử nghiệm về độ bao phủ của mã (**.c .cc .cpp**)
  + **src** : chứa các file source của dự án (**.c .cc .cpp .o**)
  + **user\_cfg.mk** : một makefile cho phép người dùng cấu hình đường dẫn và thiết lập một số cài đặt cho từng dự án cụ thể.
* **Sử dụng các lệnh Make**

Trước tiên, mở một cửa sổ Terminal trên VSCode. Vào **Terminal** > **New Terminal**

A screenshot of a phone

Description automatically generated with medium confidence

Tại tab **TERMINAL**, nhập các yêu cầu của bạn. Lưu ý, đường dẫn gốc trên terminal phải là đường dẫn đến “**CMFramework**”. Nếu không đúng, sử dụng lệnh “**cd**” để di chuyển đến.

* + **make setup** : khởi tạo (hoặc khởi tạo lại) Framework, dọn dẹp tất cả các tệp phụ thuộc của tất cả các dự án (nếu có).
  + **make info** : dùng để in ra màn hình một số thông tin về dự án.
  + **make** | **make quick** : 2 câu lệnh này cho phép chạy {**make build run**}. Nó kiểm tra sự phụ thuộc trước đó và chỉ build lại các tệp đã bị thay đổi.
  + **make force** : câu lệnh này cho phép chạy {**make clean build run**}. Nó buộc build lại tất cả các tệp mã nguồn mà không cần kiểm tra sự phụ thuộc.
  + **make clean** : dùng để xóa thư mục đầu ra, nơi lưu trữ các file object và file thực thi. Nó cũng xóa các bản báo cáo thử nghiệm (nếu có).
  + **make build** : dùng để biên dịch các file (**.c .cc .cpp**) sang các file object (**.o**), sau đó liên kết nó thành file thực thi (**.exe**).
  + **make run** : dùng để chạy file thực thi (**.exe**) trên Terminal.
  + **make debug** : dùng để gỡ lỗi chương trình trên Terminal sử dụng **GDB**.
  + **make report** : dùng để tạo bản báo cáo thử nghiệm nếu bạn sử dụng thư viện “**utest**” (sẵn trong “**tool > lib**”) trong chương trình C/C++ để viết **test case**. Hoặc (Và) tạo bản báo cáo về độ bao phủ mã (nếu có).
  + **make vsinit** : dùng để tạo ra các file cấu hình trong VSCode, giúp phần mềm này liên kết đúng các đường dẫn file trên **giao diện** hiển thị. Khi bạn move sang một project khác, lệnh này sẽ **tự động** được chạy. Bạn chỉ cần chạy lại lệnh này khi bạn **update** các đường dẫn hoặc các cài đặt trong makefile (**user\_cfg.mk**).
  + **make move** : dùng để di chuyển đến một dự án bất kỳ. Tự động thêm mới nếu dự án không tồn tại.
    - Ví dụ 1 : make move.proj1 : sẽ hiểu là di chuyển tới dự án “**proj1**”.
    - Ví dụ 2 :  make move.group1/proj1 : di chuyển đến dự án “**proj1**” nằm trong nhóm dự án “**group1**”. Bạn có thể lồng trong nhiều group, sử dụng dấu “**/**” để phân cách.
  + **make remove** : dùng để xóa bỏ một dự án hoặc một nhóm bất kỳ. Nếu xóa dự án hiện tại, nó sẽ tự chuyển về dự án mẫu “**~temp**”. Cách dùng giống như lệnh “**move**”.
  + **make import zip** : dùng để thêm một dự án hay một nhóm mới từ một file zip được chia sẻ từ lệnh “**export**” của bất kỳ ai. Cách dùng giống như lệnh “**move**”. Tuy nhiên, biến “**zip**” sẽ cho biết đường dẫn đến file zip.
    - Ví dụ : make import.proj2 zip=”path/to/file.zip” .
    - Lưu ý : biến “zip” phải viết liền với dấu “=” và đường dẫn. Không được sử dụng khoảng trắng ở giữa.
  + **make export** : dùng để đóng gói một dự án hoặc một nhóm bất kỳ. Kết quả là bạn sẽ được một tệp zip được lưu trong thư mục “share”. Bạn có thể lưu trữ hay chia sẻ cho bất kỳ ai. Cách dùng giống như lệnh “**move**”.
  + **make list** : dùng để liệt kê tất cả các file và folder đang có trong dự án.
  + **make print** : dùng để in ra giá trị của các biến được sử dụng bên trong makefile.
    - Ví dụ : make print.VAR1.VAR2 : in ra giá trị của 2 biến là “**VAR1**“ và “**VAR2**”.
    - Tương tự nếu in nhiều biến sẽ phân cách nhau bởi dấu chấm “**.**”.
    - Câu lệnh này chỉ sử dụng cho Admin hoặc nếu bạn là nhà phát triển Framework.
  + **bash make.sh <options> ::: <projects>** : Lệnh này cho phép người dùng thực thi các lệnh “**make**” trên nhiều dự án cùng một lúc, giúp đẩy nhanh quá trình lấy báo cáo khi chúng đã được thực thi ổn định.
    - Ví dụ :  bash make.sh force report ::: test1 group1/proj1 group1/proj2 : trong ví dụ trên, ta sẽ thực hiện “**make clean build run report**” trên lần lượt các dự án “**test1**”, “**group1/proj1**”, và “**group1/proj2**”.
    - Ví dụ khác :  bash make.sh force report .

hoặc :  bash make.sh ::: test1 group1/proj1 .

* **Lưu ý**
  + Tất cả các đường dẫn, tên file đều phải được viết liền. Tức là không có khoẳng trắng ở giữa. Bạn nên lưu ý khi đặt tên cho bất kỳ một tệp hoặc thư mục nào.
  + Không được phép đặt tên các source file hay header file trùng với tên của dự án. Điều này có thể xảy ra xung đột khi tạo các file đầu ra.
  + Trong một dự án, mỗi một tên file là duy nhất ở toàn bộ các thư mục. Bạn không được phép biên dịch 2 tên file giống nhau vì nó cũng có thể xảy ra xung đột tại đầu ra.
* **Đầu vào đặc biệt**
  + Bạn có thể truyền dữ liệu vào chương trình của bạn thông qua các đầu vào đặc biệt như: các định nghĩa (**USER\_DEFS**), các đối số (**VAR\_ARGS**), hay qua các biến môi trường (**USER\_ENVS**).
    - **USER\_DEFS** : Một định nghĩa phải có cờ “**-D**” phía trước định nghĩa đó.

Ví dụ : -DMAC\_A -DMAC\_A=1 -D”MAC\_A=Hello world” -D”MAC\_A=\”Hello world\””

* + - **VAR\_ARGS** : Các đối số phân cách nhau bởi các khoảng trắng. Trong trường hợp bên trong một đối số chứa khoảng trắng, nó cần đặt trong **dấu ngặc kép**.

Ví dụ : Hello\_world “Hello world” Hello” “world

* + - **USER\_ENVS** : Các biến môi trường cần được định nghĩa bên trong **makefile**. Sau đó **export** các biến đó. Cuối cùng, thêm tên biến vào **USER\_ENVS**.

Ví dụ : VAR\_ENV := Hello world > export VAR\_ENV > USER\_ENVS += VAR\_ENV

* + Ngoài ra, các đầu vào đặc biệt có thể chứa các ký tự ASCII đặc biệt như sau:

|  |  |  |  |
| --- | --- | --- | --- |
| **Input** | **Output** | | |
| **USER\_DEFS** | **VAR\_ARGS** | **USER\_ENVS** |
| \\ | \ | \ | \\ |
| \” | “ | “ | \” |
| \\” | \” | {ERROR} | \\” |
| \n | [ENTER] | \n | \n |
| \t | [TAB] | \t | \t |
| “a b” | a b | a b | “a b” |

1. **Gỡ lỗi**

Có 2 giải pháp để gỡ lỗi một dự án sau khi đã hoàn thành biên dịch (đã có file **.exe**).

* **Sử dụng dòng lệnh (CLI)**
  + Chạy lệnh “**make debug**” và thực hiện gỡ lỗi bằng GDB.
* **Sử dụng giao diện VSCode (GUI)**
  + Chạy lệnh “**make vsinit**” để đảm bảo bạn đã cập nhật tất cả các biến môi trường, các định nghĩa, các đường dẫn cần thiết lên giao diện VSCode.
  + Trên giao diện **VSCode**, chọn tab “**Run and Debug** (**Ctrl + Shift + D**)”. Sau đó chọn tác vụ “**CMFramework Debug**”.

A screen shot of a computer

Description automatically generated

* + Cuối cùng, nhấn vào nút “**Play** (**F5**)” trong hình trên để thực hiện gỡ lỗi.
* **Lưu ý**
  + Bạn cần nắm được các câu lệnh sử dụng cho việc gỡ lỗi của GDB, hoặc/và cách gỡ lỗi trên VSCode trước khi sử dụng tính năng này.

1. **Sử dụng các công cụ khác**

* **Sử dụng công cụ kiểm thử code (utest)**
  + Để sử dụng công cụ này, trước tiên bạn cần thêm đường dẫn của thư viện này vào makefile.
  + Mở file “**user\_cfg.mk**”. Tại biến “**SRC\_DIRS**” dùng để quét và lấy các tệp mã nguồn (.c) và biến “**INC\_DIRS**” dùng để quét và lấy các tệp header (.h), bạn cần thêm đường dẫn sau vào cả hai biến trên : “**$(TOOL\_DIRS)/lib**”.

A screenshot of a computer program

Description automatically generated

Cụ thể thì **$(TOOL\_DIRS)** là đường dẫn đến thư mục “**tool**” của Framework. Sau đó “**/lib**” là để truy cập vào thư mục “**lib**” bên trong. Dấu “**\**” để nối với dòng phía dưới. Bạn có thể viết các giá trị cùng một dòng và phân cách nhau bởi “**dấu cách**”.

* + Sau khi thêm đường dẫn, bạn chạy lệnh “**make vsinit**” để cập nhật các đường dẫn vào VSCode, từ đó có thể nhắc lệnh cho bạn tốt hơn.
  + Quay trở lại file mã nguồn của bạn, chỉ cần include “**utest.h**” và trải nghiệm.

A screen shot of a computer

Description automatically generated

* + Để chạy các kiểm thử, bạn cần tạo ra các hàm kiểm thử theo format cho sẵn.

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

Trong đó, “**name\_test**” là tên hàm do bạn tự quy định. Hàm trên cũng tương đương với : **void name\_test( void ) { /\* body \*/ }** . Tuy nhiên, viết đúng cú pháp sẽ giúp người đọc dễ hiểu hơn.

A screenshot of a computer

Description automatically generated

Trong ví dụ trên, ta có một hàm kiểm thử là “**Test\_001**”.

* + Mục đích kiểm thử được tạo ra nhằm kiểm tra tính đúng đắn của mã nguồn. Vì vậy, để biết kiểm thử của bạn đúng hay sai, bạn cần có các điều kiện kiểm thử.
  + Ví dụ, bạn muốn chứng minh rằng “1+1=2” và “2+2=4” … Đó chính là các điều kiện kiểm thử. Khi đó, “**UT\_Assert**” là lệnh giúp bạn kiểm tra các điều kiện.

A screenshot of a computer code

Description automatically generated

Trong ví dụ trên, tôi có 3 điều kiện cần kiểm thử. Chỉ cần một trong số các điều kiện sai, thì cả hàm kiểm thử sẽ mang kết quả là thất bại. Vì vậy, ta có thể nói “**Test\_001**” đã thất bại do điều kiện kiểm thử thứ 3 đã sai.

* + Sau khi tạo ra các hàm kiểm thử, bạn cần thêm các hàm này vào một thùng chứa để thực thi nó.

A black background with white lines

Description automatically generated

2 Macros trên sẽ tạo ra một thùng chứa, bạn chỉ cần thêm các hàm kiểm thử vào theo cú pháp : **UT\_AddTest( name\_test, "Brief for name\_test" )**

Trong đó, “**name\_test**” là tên hàm kiểm thử đã được tạo trước đó. "**Brief for name\_test**" là tiêu đề do bạn tự đặt.

A computer screen shot of a message

Description automatically generated

Nếu có nhiều hàm kiểm thử, bạn chỉ cần thêm nhiều lệnh “**UT\_AddTest**”. Lưu ý rằng không có dấu “**;**” sau các lệnh này. Đồng thời, thùng chứa này không được viết trong bất kỳ một hàm nào. Các kiểm thử nào hiện không muốn thực thi, chỉ cần loại bỏ hoặc note lại bên trong thùng chứa.

* + Sau khi thêm vào thùng chứa, bạn cần chắc chắn rằng các hàm kiểm thử phải được khai báo trước khi sử dụng nó.

A black background with white lines

Description automatically generated

* + Để thực thi lần lượt các kiểm thử bên trong thùng chứa, bạn cần sử dụng câu lệnh sau : **UT\_RunTests()**

A screenshot of a computer

Description automatically generated

* + Cuối cùng, bạn chạy lệnh “**make**” biên dịch và chạy tệp thực thi. Sau đó, chạy “**make report**” để xuất kết quả ra tệp html và xem nó trên trình duyệt web. Tệp html này được lưu tại thư mục “**doc**” của dự án.

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

Như bạn đã thấy, Điều kiện kiểm thử thứ 3 đã sai tại dòng 55 trong tệp mã nguồn.

* + Chúc bạn thành công!
* **Sử dụng công cụ đo đạc độ bao phủ mã (CCOV)**
  + Code coverage là công cụ không thể thiếu cho việc kiểm thử mã nguồn, nó cho biết độ bao phủ của quá trình kiểm thử lên mã nguồn là bao nhiêu.
  + Để chạy code coverage, bạn truy cập “**user\_cfg.mk**”, thêm biến “**$(DEV\_DIR)**” vào các đường dẫn của “**SRC\_DIRS**”. Điều đó cho phép makefile có thể quét các tệp mã nguồn trong thư mục “**dev**”.

A screenshot of a computer screen

Description automatically generated

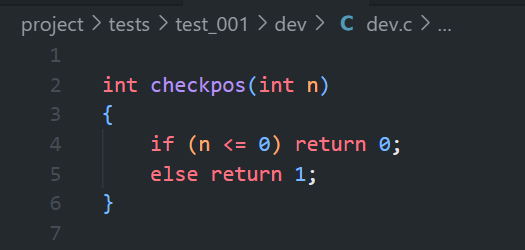
Thư mục “**dev**” là thư mục được sử dụng để đo đạc CCOV. Tất cả các hàm bên trong tất cả các tệp trong thư mục này đều được đem ra đo đạc.

* + Tiếp theo, sửa giá trị của biến “**RUN\_CCOV**” là “**on**” để cho phép sử dụng CCOV.

A close up of a report

Description automatically generated

* + Sau đó, bạn cần tạo file mã nguồn bên trong “**dev**” để viết các hàm cần đo CCOV.



Ví dụ trên đã tạo ra một hàm “**checkpos**” để kiểm tra số nguyên dương (n > 0) trong file “**dev.c**” nằm bên trong thư mục “**dev**”.

A computer code with numbers and symbols

Description automatically generated

Sau đó, tôi đã tạo ra 2 điều kiện để nó quét được cả nhánh “**if**” và nhánh “**else**” bên trong hàm “**checkpos**”. Từ đó, chỉ số CCOV của tôi sẽ được bao phủ hoàn toàn.

* + Cuối cùng, bạn chạy “**make**” để biên dịch và thực thi mã nguồn. Sau đó chạy “**make report**” để tạo các bản báo cáo bao gồm cả kết quả kiểm thử và CCOV. Chúng đều được lưu trữ trong thư mục “**doc**” của dự án.

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

* + Chúc bạn thành công!

============================================

HẾT

Tác giả : Long Dao | <https://louisvn.com>