Using GoogleTest as shared library

1. Set up

Compiling the library

1. Get the sources

```
git clone git@github.com:google/googletest.git <path_to_folder_git>
```

2. Create folder where you want the output of the compilation

```
mkdir <path_to_folder_build>
cd <path_to_folder_build>
```

3. Prepare compilation. This is the right moment to use the flags mentioned at https://github.com/google/googletest/blob/main/googletest/README.md . However the flag for compiling as shared library `-DGTEST_CREATE_SHARED_LIBRARY=1` mentioned there does not seem to work. The following flag does (found on old stackoverflow discussion https://stackoverflow.com/questions/13513905/how-to-set-up-googletest-as-a-shared-library-on-linux)

```
cmake -DBUILD_SHARED_LIBS=ON <path_to_folder_git>
```

Move relevant files to permanent folder

Ideally we would place the files in the `/usr/local/` directory but since we do not have sudo rights everywhere, the following strategy is chosen.

```
cp -r <path_to_folder_build>/lib path_to_permanent_google_test_folder
cp -r <path_to_folder_git>/googletest/include path_to_permanent_google_test_folder
cp -r <path_to_folder_git>/googlemock/include path_to_permanent_google_test_folder
```

Update system links to library

💡 Note that the sudo method does not require this step

If this step is not completed, launching the executable will fail with the error message:

> error while loading shared libraries: libgtest.so.1.11.0: cannot open shared object file: No such file or directory

This is due to the fact that the system does not know where to find the .so file. The following lines create a temporary reference. Make the reference permanent by adding it to the config file of your shell, `~/.zshrc` for example

```
LD_LIBRARY_PATH=$LD_LIBRARY_PATH:<path_to_permanent_google_test_folder>/lib
LIBRARY_PATH=$LIBRARY_PATH:<path_to_permanent_google_test_folder>/lib
CPLUS_INCLUDE_PATH=$CPLUS_INCLUDE_PATH:<path_to_permanent_google_test_folder>/include
export LD_LIBRARY_PATH
export LIBRARY_PATH
export CPLUS_INCLUDE_PATH
echo export LD_LIBRARY_PATH=$LD_LIBRARY_PATH >> ~/.zshrc
echo export LIBRARY_PATH=$LIBRARY_PATH >> ~/.zshrc
echo export CPLUS_INCLUDE_PATH=$CPLUS_INCLUDE_PATH >> ~/.zshrc
```

LD_LIBRARY_PATH is needed to find the .so during execution

LIBRARY_PATH is used by gcc to find the .so. Alternatively you can point to the location with -L

CPLUS_INCLUDE_PATH is used by gcc to find the .h files. Alternatively you can point to the location with -I

Compiling and Linking

💡 Do forget to use the c++ postfix naming convention for the test_files (*.cpp instead of *.c) - otherwise the compilation fails on missing references

```
gcc <test_and_production_use_files>.cpp -pthread -lgtest -lstdc++
```

This needs to contain the test main.cpp

An example can be found in `<path_to_folder_git>/googletest/src/gtest_main.cc`

2. Testing C code

Prequisites

Weirdly enough, integrating C code into C++ is not trivial. Googletest is written in C++ and testing plain C code requires some modifications. In The following we examine a simple example using three files:

1. main.cpp: Contains boilerplate and includes the test files (here, we are only using one file: test.h)
2. test.h: Contains the unit tests
3. plain_c_function_to_be_tested.c: Is called during the unit testing

```
#include <cstdio>
#include "gtest/gtest.h"
#include "test.h" // if you have more test suites, include them like this

GTEST_API_ int main(int argc, char **argv) {
  printf("Running main() from %s\n", __FILE__);
  testing::InitGoogleTest(&argc, argv);
  return RUN_ALL_TESTS();
}
```

```
#include "gtest/gtest.h"

extern "C" {  // this is important - else linker error
int foo();
}

TEST(simpleTest, testing_equality_one) {
  EXPECT_EQ(foo(), 1);
}

TEST(simpleTest, testing_equality_two) {
  EXPECT_EQ(foo(), 2);
}
```

```
int foo()
{
  return (1);
}
```

Normally a missing or wrong #include leads to a compilation error. When including a plain C file into C++ without explicit stating to do so, this will lead to a linker error:

> /usr/bin/ld: main.o: in function FactorialTest_Positive_Test::TestBody()': main.cpp:(.text+0x1d): undefined reference to pr()'
> collect2: error: ld returned 1 exit status

To avoid this, wrap all plain C function declarations with `extern "C" { int foo(); ... }`. You can simply drop #include statements there as well.

Compiling and Linking

You need to compile your C files separately from your C++ files.

```
gcc plain_c_function_to_be_tested.c -c
g++ main.cpp -c
g++ -o executable plain_c_function_to_be_tested.o main.o -pthread -lgtest
```

The linking has to be done with g++