Starting up!!!

Julio César

Contents

1 First things first

At this points we expect that you:

- have installed the framework in your personal computer and
- have run ALL the tests and ALL of them passed.

If that is not the case then please go back to the installation instructions which you can find in the main webpage of the project in the `github` repository here, or in the README.md file at the root folder of the project.

1.1 A framework, not a library

Contrary to how a library works, where you are in charge of the main flow of the program and the tools that you include, this project is intented to be a framework, that means, we provide you with the main code structure and you only have to fill in the specific details of for your project.

The code you write becomes part of the framework which provides you with the tools required for your project. You are always free to personalise the framework based on your needs by extending, but not modifying, its current behavior. It means, you may extend, without modifying, the framework with your own implementations to provide it with extra functionalities.

1.2 Folder structure of the framework

This is what each folder is about in the framework:

- `bin`, stores scripts used by the framework at compilation time, there are also scripts that help on the generation of **clean distributions** of the framework. These are compressed files that you may want to generate when copying the framework to a computer without Internet access; a supercomputer node.

- `build`, this folder is automatically generated when compiling the library and all the compilation files are stored there. You do not need to deal with the files within this folder, just leave them alone.

- `configs`, store configuration files, each file corresponds to an specialised configuration of the framework. The default configuraton is stored in the `default` file. If you want to use an specialised configuration of the framework you should store it here and select it at compilation time when prompting for a *configuration file*. If you are curious try any other of the configurations in this folder, some of them make use of external libraries so make sure you have installed them.

- `demos`, stores a set of examples that you may want to use as templates or starting points for your project. These demos provide a good insight on the features available in the framework. This folder also helps for testing the framework and report any found issues after new features are implemented.. All demos must compile, run and pass the tests to perform a *push* operation to the official repository.

- `external_src`, this folder stores any external library packages used within the framework to provide extra features. You should not modify this folder unless you are providing new functionalities that depend on external software packages.

- `private`, stores private files for each user/collaborator of the framework. Each user/collaborator should have its own private folder here, this should be used as the development folder of each user. The users may move their own developments (prior authorisation of the main collaborators) to the demo folder such that the applications become part of the demos of the framework that shown specialised features of the framework.

- `src`, the hearth of the framework, here lies all the source code of the framework. You should create new files in this folder only if you are adding new functionalities to the framework. Otherwise, you should be doing fine with working only in your `private` folder.

- `tools`, a set of tools used for the library, mainly for visualisation issues.

1.3 Running demos

Go to the demo folder you are interested and type `./bin/` followed by the name of the demo. Make sure there is a `./RESLT/` folder in the directory you are running the demo since the results of the computations are stored in there.

Once the demo has started you should see output messages on the terminal with general information about the results of the computations.

1.3.1 Input arguments

Some demos require input arguments to run, you can check what input arguments you need to pass by passing the `--help` or `-h` option to the demo driver.

1.4 Create your `private` folder

In order to modify or create your own code we encourage you to do so in your own private folder, to do so open a terminal and create your folder inside the `private` folder.

In a terminal type

```
cd private
mkdir bob
cd bob
```

Note that we are assuming you are named `bob`, change that with your name. Once you have created your own private folder update the `CMakeLists.txt` file in the private folder by adding your folder name at the end of the file using the following line

```
ADD_SUBDIRECTORY(bob)
```

Substitute `bob` with your folder name.

Compile again the full framework using the `./autogen.sh` comand at the root folder and make sure no problems are found.

2 Creating your own project into the framework

Start by copying a demo driver into your private folder, here we copy the demo driver `demo_nasch_pbcwb_loop_over_bumps.cpp` in the folder `demos/nasch/pbc_with_bumps`. Assuming you are in your private folder type the following in a terminal

```
cp demos/nasch/pbc_with_bumps/demo_nasch_pbcwb_loop_over_bumps.cpp demo_bob.cpp
```

Then copy the `CMakeLists.txt.user_template` file that lives in the `tools` folder into your private directory and change its name to `CMakeLists.txt`, then change its content as follows:

- All the instances of the tag `SRC_demo_bob` for your own tag to identify source code.
- All the instances of `demo_bob.cpp` for the name of your code file.
- All the instances of `demo_bob`, this is the name of your executable and the name you need to type to compile your project.
- All the instances of the tag `LIB_demo_bob` for your own tag to identify libraries required for your code
- Include the libraries you need. In the example only the `general_lib` and the `problem_lib` are included (in section # there is a list for the name of the libraries that you may include into your project).

Compile again the full framework using the `./autogen.sh` comand at the root folder and make sure no compilation errors are found. Once compilation has finished without errors you can compile your code by going to the `build` folder and type

```
make demo_bob
```

The compilation output should be displayed in your screen. Fix any compilation problem you found, then to run your code go back to your `private` folder, make sure you have a `RESLT` folder in your directory and type

```
./bin/demo_bob
```

Any output from your code should be displayed on the terminal.

2.1 Compilation or source code and exection of binaries

As you have noticed, the generation and execution of your project is made in two different folders:

- the `build` folder and
- your `private` folder.

We followed this two-folders strategy to avoid having automatically generated `CMake` files all over the folder structure of the project. By following this strategy we keep a clean structure for the project and group all files generated by `CMake` in the `build` folder. This help us to keep track for changes easily since we can exclude the `build` folder from the `git` repository.

Just keep in mind that whenever you want to compile your source code you need to do so in the `build` directory, just type `make` followed by the name of your project. Then when you want to execute your project go back to your `private` folder and type `./bin/` concatenated with the name of your executable file.

3 Add your project as a demo into the `demos` folder

Once you are fully happy with the results of your project you can include it as part of the framework in the `demos` folder. Here are three things that you need to do before including it in the `demos` folder.

1. You need to think where in the `demos` folder structure to include your demo.
2. You need to create the expected/correct output files that will be used by the testing unit to check for running errors.
3. You will have to check your demo is run and passed properly when running the demos of the project.

After considering these points proceed as follows:

- Create the required folder structure into the `demo` folder.
- Add `ADD_SUBDIRECTORY` lines in the corresponding `CMakeLists.txt` files to include your newly created folder structure.
- Copy the `CMakeLists.txt.demo_template` file into the demo folder and rename it to `CMakeLists.txt`.

- Create a `validate` folder into the newly created folder structure for your demo.
- Store the expected/correct output files generated by your demo in the `validate` folder. The validation files should be named as `validate_double_demo_bob.dat` and `validate_demo_bob.dat` changing `bob` by the name of your project. Please generate double and single precision validation files by running your project with both configurations.
- Rename the same tags and variables as in `* Creating your own project into the framework` section.
- Rename ALL instances of `TEST_demo_bob_run` by the name of your demo. Keep the `TEST` and `_run` prefix and postfix, respectively.
- Rename ALL instances of `demo_bob` with the name of your demo.
- Rename ALL instances of `VALIDATE_FILENAME_demo_bob` with the name of your tag for the validation file.
- Change the name of the validation file `validate_double_demo_bob.dat` by yours, this file should have the expected/correct output of your project using double precision.
- Change the name of the validation file `validate_demo_bob.dat` by yours, this file should have the expected/correct output of your project using single precision.
- Rename ALL instances of `TEST_demo_bob_check_output` by the name of your demo. Keep the `TEST` and `_output` prefix and postfix, respectively.
- Make sure that your demo generates an output file named `output_test.dat` with the information that will be checked by the testing process.

4 [OPTIONAL] Include the `bin` folder of the project to your `PATH` variable

Add the following lines at the end of your `.bashrc` file in your home folder.

```
export PATH="/home/tachidok/local/working/research/cellular_automata/bin/:$PATH"
```

Assumming the working directory of the framework is at

```
/home/tachidok/local/working/research/cellular_automata/
```

change it according to your working directory.