CS11 Intro C++ Lab 4: C++ Build and Doc Tools

Now that our unit-converter is in reasonably good shape, we will take a lab to focus on two development tools used widely with C++ projects: `make` and `doxygen`. You won't need to write any C++ code this week, unless you want to fix any bugs or refine any features from previous labs.

The `make` Build-Automation Tool

The `make` tool has been around for a long time; the first version of this tool was actually published in 1976. Certainly it has its strengths and weaknesses, but it is simple to understand and to use. Once any project grows beyond a certain basic size, you will want to automate the build process, and `make` is certainly a reasonable choice.

This week you will write a `Makefile` for your unit-converter project, that satisfies the following requirements:

- Your makefile should build both the `convert` program and the `hw3testunits` program.

- You should have an `all` target that builds both of the above programs.

- You should have a `clean` target that deletes all object files (`*.o`), as well as the `convert` and `hw3testunits` binaries.

- You should also have a `test` target that runs the `hw3testunits` program for you, so for example you can run `make clean test` to rebuild the project from scratch and then run the tests on the unit-conversion code.

- You should make sure to declare build targets as "phony" if they do not specify actual files that are produced by `make`.

Make sure that when you test your `clean` target, you make a backup of your source files first! It is very easy to make a mistake when writing your `clean` target, that accidentally obliterates all of your files. You only have to do this once or twice before you learn your lesson and back up your work!

The `doxygen` Documentation Generator

Automated documentation generation is a very powerful technique for making professional-looking and useful documentation for your projects. Many of these tools make it very easy to explore a programming interface, because the generated documentation will contain links between related topics.

We will use the `doxygen` documentation generator for this class. Doxygen is very similar to `make`, in that it draws all of its configuration out of a config file. You can generate a Doxygen config file by typing:

```
    doxygen -g
```

If you specify no additional commands, this will generate a file called `Doxyfile` in your current directory. The file is *heavily* documented, almost to the point of being overwhelming, but at least it's easy to understand what is going on.

Go ahead and generate a new `Doxyfile` configuration file for your project, and then you can open it up in your editor and customize it for your project. You might want to read through this file once carefully, so you have an idea of what you can do with it.

Here are some specific values to configure:

- `PROJECT_NAME` - set this to a short phrase describing your project

- `OUTPUT_DIRECTORY` - set this to `docs`. All documentation will be generated into this directory.

- `FULL_PATH_NAMES` - normally you will probably want to set this to `NO`. It doesn't really matter for the purposes of CS11 though.

- `JAVADOC_AUTOBRIEF` - set this to `YES` so that Doxygen will use the first sentence of your class/function documentation as a "brief comment." (It is called "Javadoc autobrief" because this is the default behavior of the Javadoc tool.)

- `QT_AUTOBRIEF` - set this to `YES` as well.

- `BUILTIN_STL_SUPPORT` - set this to `YES` since you will be using the C++ Standard Library (which includes the Standard Template Library) a lot.

We want to generate documentation for *everything*, so set these values:

- `EXTRACT_ALL` - set this to `YES`
- `EXTRACT_PRIVATE` - set this to `YES`
- `EXTRACT_PACKAGE` - set this to `YES`
- `EXTRACT_STATIC` - set this to `YES`
- `EXTRACT_LOCAL_CLASSES` - set this to `YES`

Doxygen needs to know what files to generate documentation from:

- `INPUT` - specify the source files you want to generate documentation from. This should include `units.h`, `units.cpp` and `convert.cpp`. (There is no need to include the test code.)

The section labeled `Configuration options related to source browsing` can be fun to tinker with, if you want your documentation to include the actual source code for your project.

You should make sure to only generate HTML output.

- `GENERATE_HTML` - should be set to `YES`
- `GENERATE_LATEX` - should be set to `NO` (it's `YES` by default)
- Other `GENERATE_*` values should also be set to `NO`, and are by default.

Commenting Your Code

Way back in lab 1, we specified these guidelines for documenting your code:

- Write a comment just before every class you declare, explaining the purpose of the class. Your comment doesn't need to be long, but it should completely and concisely describe the purpose of the class.

- Within your class declarations, write a comment just before every data-member in the class, describing the purpose of the data-member. Again, in many situations this will not be a long comment, but it should completely and concisely describe the purpose of the data-member, including any values with special meanings that the member can be set to.

- Write a comment just before every function and member-function you define, explaining the purpose of the function, along with any inputs that are invalid, or outputs that indicate special details. Again, this comment may not be long (particularly for simple accessors and mutators), but it should definitely be present.

- Within any function or member-function, make sure to explain any parts of the code that are particularly tricky or subtle. Don't just repeat the code; explain it.

  If the code is very simple, you don't need to write any comments; you can assume that anyone reading your code will be reasonably proficient with the language.

If you have been following these guidelines faithfully, this part of the assignment will be a piece of cake, because you will simply need to convert all your comments to Doxygen-style comments.

If you have not, this is your chance to fill in the gaps! (Trust me, every project has gaps.)

You might notice that the above guielines will cause you to write your comments next to the code that the comments describe. In a language like C++ where the declaration and definition are often separated, the best rule of thumb is to put your comments with the definitions of things, not the declarations, when they are in two different places.

Testing Doxygen Code-Generation

Once you have gotten your sources converted to Doxygen-style commenting, and you have gotten your `Doxyfile` configured properly, you can give it a try by typing:

```
doxygen
```

This should generate documentation files into the `docs/html` directory that you can open in a web browser. Browse through the generated results, and see if anything needs to be touched up.

Incorporating Doc-Gen into Your Makefile

Once you have gotten automated documentation generation working, update your `Makefile` with a `doc` target that generates your documentation. This is another phony target.

Also, make sure that your `clean` target removes the directory that the documentation is stored into. You may have to use a `rm -rf` style command to delete directories. **As before, when you test this for the first time, make sure you have backed up your work before testing it!**

Submitting Your Work

Once you have completed the above tasks, submit your work through csman. Make sure to submit these files:

- `Makefile`
- `Doxyfile`
- `units.h`
- `units.cpp`

- `convert.cpp`

Assignment Feedback Survey

Please also complete and submit a feedback survey with your submission, telling us about your experience with this assignment. Doing so will help us to improve CS11 Intro C++ in the future.
