V22.0474-001 Software Engineering
Spring 2008

Lecture 13: Sample Project and Makefile

Clark Barrett, New York University

Sample Project

I've put together a sample project to illustrate a suggested physical architecture and Makefile system. I do not expect or suggest that you look at the code, just at the way the files are laid out and at the way the Makefiles are set up.

- Check it out:

  ```
  cvs -d /home/barrett/public/repository co sample_project
  ```

- Browse it online:
  Click on Sample Project from the course web page.

To build the sample project on a CIMS machine, type:

```
./configure; make.
```

Sample Makefile System

To use the Makefile system from the sample project for your own code, you must do the following:

- Mimic the directory structure of the sample project:

  `project`, `project/bin`, `project/src`, `project/src/include`, `project/src/main`, `project/src/`*`module1`*, `project/src/`*`module2`*, ...

- Copy files from the sample project: `configure`, `Makefile` `Makefile.local.in`, `Makefile.std`, `bin/unpack.in`, `src/Makefile`

- Edit `Makefile.local.in` and change the “Project-specific definitions” as appropriate.

- Edit `src/Makefile` and replace the sample header files and module names with your header files and module names.

- Create a `Makefile` in each of your modules similar to the ones in the sample project (note the difference between a *library* module such as `expression` and an *executable* module such as `main`).

Sample Makefile System

Notes

If your project does not use external libraries, then the only thing in your "Project-specific definitions" section of `Makefile.local.in` should be the project name.

The sample project uses an external library called `libbdd` located in `/home/barrett/public/cudd-2.3.1/`. I compiled and installed this external library in this directory. The "Project-specific definitions" in sample project tell the Makefile where to find the headers and libraries of this external package.

Note that you must replace the sample module names with your module names in two places in `src/Makefile`. Look for the comments that say: "List all modules here".

Sample Makefile System

What can you do with this Makefile system?

- *Easy to add new source files*. To add a new cpp file or a private header file, simply add its name to `src/module/Makefile`. Public header files should be place in `src/include` and added to the list in `src/Makefile`. If you create a new module, its name must to the module lists in `src/Makefile`.
- *Support for multiple build-types*. Default build is *optimized*. To compile the *debug* version, do one of the following:
  - Run `./configure --with-build=debug`
  - Edit `Makefile.local` and change `OPTIMIZED=1` to `OPTIMIZED=0`
  - Type `make OPTIMIZED=0`.
- Similarly, the default build is a static build. To build dynamic libraries, do one of the following:
  - Run `./configure --disable-static`
  - Edit `Makefile.local` and change `STATIC=1` to `STATIC=0`
  - Type `make STATIC=0`.

Sample Makefile System

- *Support for multiple platforms*. To compile for a different platform (i.e. object files, libraries, and executables are stored separately), rerun `./configure` and then `make`.

Note that every time you run `./configure`, your `Makefile.local` gets clobbered, so any local configuration options that you changed will get set back to their defaults.

- *Support for unit testing*.
  1. Create test directory `src/module/test`
  2. Write unit test in `src/module/test/test.cpp`
  3. Create local `src/module/test/Makefile` (see example in `sample_project/src/expression/test`)
  4. Type `make` in `src/module/test` to build unit test
  5. Executable for unit test is placed in top level `bin` directory

Sample Makefile System

- *Support for source distributions*.
  1. Make sure you have all your public include files and modules listed in `src/Makefile`
  2. Type `make dist` at the top level.
  3. You will get a file called *`project`*`.tar.gz`.
  4. Type `tar zxvf` *`project`*`.tar.gz` to unpack the distribution.
- *Support for source searching in* `emacs`
  - To find a class declaration:
    1. Move the cursor onto the class name
    2. Type `M-.`
    3. Make sure `emacs` has the right class name and press return
    4. Give path to tags table: *`project/`* and press return
    5. To go back to where you were, use `C-x b`
  - To search through all files in your project
    1. Type `M-x tags-search`
    2. Enter the expression to search for
    3. If you haven't before, give the path to the tags table: *`project/`*
    4. Use `M-,` to "find next"