Advanced Makefiles

We will be looking at a sample project I have put together.
You can check out your own copy of the sample project as follows:

```
cvs -d /home/barrett/public/repository co sample_project
```
To build the sample project, edit `Makefile.local` and change the `TOP` variable to point to your local working copy.
Then type `make`.
To use this Makefile system for your own code, you must do the following:

- Mimic the directory structure of the sample project
- Edit `Makefile.local` and change the options as appropriate
- Edit `src/Makefile` and include your files and modules
- Create a `Makefile` for each module similar to the ones in the sample project.

Outline

- Advanced Makefiles
- Using DEBUG macros
- Debugging with gdb

Advanced Makefiles

Why use an advanced Makefile?

- Easy to add new source files and modules
- Support for multiple platforms
- Support for multiple build-types (i.e. optimized vs debug)
- Support for unit testing
- Support for source searching
- More to come
Using DEBUG macros

In the sample project, the header file `src/include/debug.h` contains several useful debugging macros:

- `FatalAssert(cond, msg)`
- `IF_DEBUG(code)`
- `DebugPrint(cond, msg)`
- `DebugAssert(cond, msg)`

Using gdb

We will take a look at an online gdb tutorial at:


I will also demo using gdb in emacs to debug a unit test in the sample project.