ECE 3574: Applied Software Design

Meeting 08: Building Cross-Platform Software using CMake

The goal of today's meeting it to learn about building larger software projects that have multiple modules of code, unit tests, and main programs.

- Why CMake?
- Running CMake: GUI and command-line
- Writing a basic CMakeLists.txt configuration file
- Exercise

Software Configuration and Build tools

You should be able to build all dependencies and the code itself, in debug and release mode, for all platforms supported **in a single step**.
This can be done by a variety of means, including customs scripts and IDE tooling. We will be using a popular open source tool for this called cmake.

Why CMake? What problem does it solve?

- Once a project gets to a certain size, compilation and linking, setting compiler flags, etc becomes complicated

See example.

- This is especially true for cross-platform projects. It is a pain to maintain build configuration for each platform (VS .sln, XCode .xcodeproject, makefiles, . . .)
- CMake is a build generator, it writes the files needed for the specific IDE or build tool

There are other tools that do this as well, e.g. scons.

Running CMake

- Using the GUI
- Using the command line

See demo.

Basic CMakeLists.txt Syntax

```
cmake_minimum_required(VERSION 3.5)
project(YOURPROJECTNAME CXX)

add_executable(exename1 file1.h file2.cpp ... )

add_executable(exename2 file3.h file4.cpp ... )

enable_testing()
add_test(test_name exename arguments)
```

See demo

More advanced CMake

CMake is a very flexible tool. Some examples

- perform different configurations based on platform
- modify files at configure time (this is used in project 1)
- run external scripts and programs for memory checking, converage analysis, etc

Exercise 08

See the website.

Next Actions

- Add questions to the Piazza post about Project 1.

Next time we will do a Q&A based on these questions.