ECE 3574: Applied Software Design

Changwoo Min

Welcome to ECE 3574

- **Applied Software Design**
  - CRN 12965
  - Web site: https://computing.ece.vt.edu/~changwoo/ECE3575-2018S/
  - Instructor: Changwoo Min, changwoo@vt.edu

Today's schedule

- About your instructor
- Description of the course
- Administrative details
- Expectations
- Course Tools Setup

About me

- Changwoo Min
- Assistant Professor at ECE @ VT
- Web site: https://sites.google.com/site/multics69/
- Email: changwoo@vt.edu
- Office: Durham 455

My research interests

- Many-core performance scalability of operating system
  - What happen if we run Linux on 448-core machine?
  - Will your application run 448x faster?
- Storage performance on blazing fast storage device
  - What happen if storage performance is becoming closer to DRAM performance?
  - Will your application achieve DRAM-like IO performance?
- System security
  - Are you sure if your hardware/software is not hacked?

Some of my open source projects

- Mosaic: Graph processing engine
  - Written in C++
- FxMark: File system benchmark
  - Written in C and Python
- Juxta: Automatic bug finding tool in linux file systems
  - Written in C++, C, and Python
- DANBI: Dataflow Parallel Runtime for Manycore Systems
  - Written in C++
- …

Communication

- Course website
  - https://computing.ece.vt.edu/~changwoo/ECE3575-2018S/
  - syllabus, schedule, notes, etc.
  - primary way materials are *distributed*.
- GitHub
  - https://github.com
  - distribute starter code, demonstrate progress
  - share code with instructors/TAs
  - how you submit your assignments

Communication

- Canvas
  - https://canvas.vt.edu/courses/66470
  - grades posted
- Piazza
  - https://piazza.com/class/jcf48hx8yk53op
  - use it to ask (and answer) questions
  - forum/wiki like software for QA, polls, announcements
  - replaces email listserv, but has a configurable email digest

Course objectives

- Use software design patterns and application programming interface (API) specifications to implement efficient and portable software.
- Design and implement multi-threaded and multi-process applications that rely on standardized inter-process communication and synchronization mechanisms.
- Design and implement complex software applications based on portable software frameworks and event-driven programming.
- Design, implement, and perform testing strategies including unit and integration testing.

Course objectives

- In short, practice to design and implement large-scale software
- **Q: Lines of code of your largest software written in C/C++?**
    1. < 500 lines of code
    2. < 1,000 lines of code
    3. < 5,000 lines of code
    4. > 10,000 lines of code

Course topics

- Generics and containers
- Inheritance and polymorphism
- Unit and integration testing
- Design patterns
- Using class-based software libraries
- Event-driven programming
- Concurrency: processes and threads
- Communication using shared memory and messages

Prerequisites

- ECE 2574 Data Structures and Algorithms. You are expected to be competent in the basics of programming with C++ and the use of data structures and algorithms to solve problems.
- It is helpful to be familiar with Unix systems (e.g. taken 2524) but it is not considered a prerequisite.

Text and resources

- Readings will be assigned from the following books and various online sources.
  - Clean Code: A Handbook of Agile Software Craftsmanship
  - The Pragmatic Programmer: From Journeyman to Master
  - Effective C++: 55 Specific Ways to Improve Your Programs and Designs

Additional resources

- Pro Git book: https://git-scm.com/book/en/v2
- CMake Tutorial: http://www.cmake.org/cmake-tutorial/
- C++ Reference: http://en.cppreference.com/w/
- QT Documentation: http://doc.qt.io

Software

- C++ compiler with sufficient C++11 support
  - Only specified compiler extensions and libraries may be used.
  - GCC >= 4.8
  - Clang >= 3.5
  - VC++ >= 19 (e.g. Visual Studio 2015 or 2017)
- CMake for managing the build process (see www.cmake.org)
- `git` for source code management tool (see git-scm.com)

Development environments

- For development you can use your favorite editor and a command console to invoke the compiler toolchain, or you can use any integrated development environment (IDE) supported by CMake, including Visual Studio on Windows and XCode on the Mac
- There are several other options including QT Creator and CLion.
- Use whatever works for you but note each project will define a reference environment using a **Linux virtual machine** that, **for grading purposes**, is the final arbiter of working code.
- Highly recommend to use Linux (e.g., Ubuntu, Fedora)

Grading and honor code

- Coursework consists of in-class exercises, project milestones, and a final exam. The grades will be computed as follows:
  - Exercises: 10%
  - Project: 75%
  - Final Exam: 15%
- All graded work, other than the in-class exercises, is expected to be the original work of the individual student.
  - **Do not copy other's code.**
  - **I do use a code comparison system across sections**.

Exercises

The exercises are worked through in-class after a short lecture on the material for that day, and are *due by midnight the day of class*.

Project

- The project is divided into milestones with requirements that correspond to the material covered to that point in the course.
- These milestones have explicit due dates.
- The total project grade is distributed across these milestones – approximately 20% each.
- **No project work will be accepted past the due date.**

Prior expectations

You are expected to understand basic computer organization and C++ syntax from ECE 1574

- Types, including references and pointers
- How to write and call a function, passing arguments and returning values
- How to write and use a basic class
- How to read and write files, including parsing techniques
- How to do proper memory allocation and handling

Prior expectations

You are expected to understand selection and use of common data structures and algorithms from ECE 2574

- Array-based and Link-based lists, stacks, queues, deques, and priority queues
- Tree and Hash Table based dictionaries
- Algorithms and used for sorting and searching

Prior expectations

You are expected to be able to write, compile, and debug C++ code using good engineering practices.

- Perform the mechanics of compiling a program
- Understand how to read and correct compile-time errors
- Understand runtime-errors and how to identify why they are occurring
- Use an incremental development technique
- Have good debugging skills (hypothesis testing)
- Be able to identify and use reference material to solve problems

Prior expectations

- A Readiness Exercise and Milestone 0 will be used to ascertain your competence in these areas.
- If you do poorly, you will be asked to meet with me to discuss your preparation

Tips for success

- Here are the tips I have for doing well in the course.
  - **Attend class** and do the exercises
  - **Start projects early** and ask questions as soon as you have them
  - **Test your code early** because bug fixing is more difficult
  - **Attend office hours** and recitations. If you can't make these then contact me to setup one-off meetings.
  - **Find peers** to work with (just take care with the honor code)
- Ultimately however if you cannot consistently devote **12-15 hours per week** on the course it is unlikely you will do well.

Semester project: MIPS CPU simulator

- Project page
- Milestone 0
  - The goal of this first milestone is to get used to the course workflow and be sure you understand how to submit code for grading.
  - You will write a lexer for our simulator
  - **Due 2/5 by 11:59 pm**

Questions?

Exercise 01: setup

See Website

Next actions

- Take the Readiness Exercise
- Make sure you complete today's Exercise 01.
- Read Chapter 1 and 2 of the Pro Git Book
    - You can skip sections 1.2 and 1.5
- Start on Milestone 0 (at least read the description and related material)

Takeaway

- Here are the tips I have for doing well in the course.
  - **Attend class** and do the exercises
  - **Start projects early** and ask questions as soon as you have them
  - **Test your code early** because bug fixing is more difficult
  - **Attend office hours** and recitations. If you can't make these then contact me to setup one-off meetings.
  - **Find peers** to work with (just take care with the honor code)
- Ultimately however if you cannot consistently devote **12-15 hours per week** on the course it is unlikely you will do well.

Takeaway

Here are the tips I have for doing well in the course.

Attend class and do the exercises

Start projects early and ask questions as soon as you have them

Test your code early because bug fixing is more difficult

Attend office hours and recitations. If you can't make these then contact me to setup one-off meetings.

Find peers to work with (just take care with the honor code)

Ultimately however if you cannot consistently devote 12-15 hours per week on the course it is unlikely you will do well.