SENG 475 & ECE 569A: Advanced Programming Techniques for Robust Efficient Computing (With C++)

Michael Adams

Department of Electrical and Computer Engineering
University of Victoria
Victoria, BC, Canada
E-mail: mdadams@ece.uvic.ca

* *While waiting for the lecture to begin, please complete the initial course questionnaire.* *

Section 1.1

Course Overview

Course Overview

- interdisciplinary in nature (e.g., engineering and computer science)
- explores variety of programming topics, which may include:
  - data structures and algorithms
  - computer arithmetic
  - compile-time versus run-time computation
  - generic programming techniques
  - error handling, exceptions, and exception-safe coding
  - resource management, memory management, and smart pointers
  - cache-efficient coding
  - concurrency, parallelism, and vectorization
- considers several application areas, which may include:
  - geometry processing and computational geometry
  - numerical analysis
  - signal processing
  - computer graphics
- uses C++ programming language (C++17)
- employs Linux-based software development environment with GCC and Clang compiler toolchains

Prerequisites and Requirements

- should possess ***reasonably good programming skills***
- must be willing to ***attend lectures regularly***
- should have ***basic familiarity with C++*** (e.g., classes, templates, and standard library)
- ideally, this knowledge of C++ would be acquired prior to start of term
  - through other courses taken (e.g., CSC 116 or ELEC/ECE 486/586); or
  - by watching instructor's video lectures
- will not rely on knowledge of C++ until second week of classes
- if no prior knowledge of C++ and have strong programming skills, may attempt crash course on C++ by watching instructor's video lectures (about 9 hours in duration) before second week of classes (but will require considerable amount of time and effort)
- first programming assignment (excluding software tools exercise) (i.e., Assignment 1) ***intended solely as review*** of basic C++
- students can use Assignment 1 to help judge if they possess sufficient knowledge of C++ for course

Course Topics

- nominally, major topics to be covered (in order) are:
  1. Algorithms (2 lectures)
  2. Data Structures (2.5 lectures)
  3. A Few Remarks About Basic C++, Const, Constexpr, and Literal Types (2.5 lectures)
  4. Value Categories, Moving and Copying, Temporary Objects, and Copy Elision (3.5 lectures)
  5. Error Handling, Exceptions, and Exception Safety (3.5 lectures)
  6. Smart Pointers (1.5 lectures)
  7. Computer Arithmetic, Interval Arithmetic, and Exact Arithmetic (3 lectures)
  8. Memory Management and Container Classes (5 lectures)
  9. Cache-Efficient Algorithms (5 lectures)
  10. Concurrency (5 lectures)
- if time permits, additional topics may be covered, such as:
  - Vectorization
  - Lvalue and Rvalue References, Move Semantics, and Perfect Forwarding
  - Memory Allocators
  - Atomics and Memory Models

Additional Comments on Course Content

- languages, libraries, and tools considered:
  - C++ programming language in some depth
  - C++ standard library as well as some other industry-standard libraries (e.g., Boost)
  - C++ compiler (i.e., GCC and Clang)
  - build tools (i.e., CMake)
  - debugging and testing tools (e.g., ASan, UBSan, and Catch2)
  - version control systems (i.e., Git)
- emphasis on libraries and tools commonly used in industry
- rationale for using C++:
  - general purpose and efficient
  - international standard, vendor neutral, supported on many platforms
  - many jobs require knowledge of C++
  - likely to continue to be dominant language into future (built on top of C which is still going strong after 40 years)
  - superset of C (so can learn two languages for price of one)
  - easier to migrate from C++ to C, Java, and many other languages than other way around

Learning Outcomes

- Upon completion of the course, students should be able to:
  1. identify many of the factors that can impact the performance and robustness of code
  2. ***select data structures and algorithms*** that are appropriate for solving a given problem and justify the choices made
  3. develop software to ***meet a detailed set of specifications***
  4. recognize the importance of ***thoroughly testing*** code
  5. demonstrate an intermediate-level competency in the ***C++ programming language***
  6. demonstrate a basic competency with the ***C++ standard library*** as well as several other libraries (e.g., Boost and CGAL)
  7. make effective use of the tools available in a typical C++ software development environment

Course Outline and Various Other Handouts

- Discuss the following documents, all of which are available from the course web site:
  1. course outline
  2. open-access course-materials support handout
  3. course-materials bug-bounty program (CMBBP) handout
  4. video-lectures handout
  5. assignments handout (which is split into several separate PDF documents)
  6. non-programming exercises handout
  7. project handout (for graduate-level version of course)
  8. lecture-slide supplements

Course Web Site

- course web site:
  - `http://www.ece.uvic.ca/~mdadams/courses/cpp`
- vast wealth of course-related information available from web site, including (but not limited to):
  1. course outline
  2. course-overview slides
  3. course-materials bug-bounty program (CMBBP) handout
  4. assignments handout (which is split into several PDF documents)
  5. non-programming exercises handout
  6. project handout (for graduate-level version of course)
  7. lecture-slide supplements
  8. video-lectures handout
- students should read all information on course web site
- some information is password protected (usually marked by padlock)

Video Lectures

- many programming-related video lectures available via instructor's YouTube channel:

  `https://www.youtube.com/user/iamcanadian1867`

- summary of available video lectures (including URLs) can be found on video-lectures handout
- some course content delivered by video lectures:
  - some topics covered only in video lectures
  - other topics covered mainly in video lectures with regular (i.e., in-class) lectures focusing only on more difficult aspects of material
- vast majority of course content covered only in regular (i.e., in-class) lectures (***not in video lectures***)
- video lectures expected to be extremely helpful to those who may have less background in C++

Computer-Based Tutorial

- tutorial is not tutorial in usual sense employed by most courses
- run by ***instructor***, not teaching assistant
- scheduled in computer lab for access to C++ software development environment
- particular uses of tutorial to be determined by needs of course as course proceeds and may include (amongst other things):
  1. students have opportunity to ask for help (i.e., in-lab office hours)
  2. instructor may give presentations on various topics to fill (unanticipated) gaps in student knowledge or clarify more difficult topics
  3. instructor may give software demonstrations
  4. instructor or markers may conduct interviews with students regarding code submitted for programming assignments (in order to guard against plagiarism)
  5. students can work on programming assignments or exercises
- tutorials start in ***first week*** of classes
- tutorials do not necessarily run for full duration of class schedule
- tutorial attendance is ***mandatory***

Plagiarism and Other Forms of Academic Misconduct

- plagiarism taken ***very seriously*** by instructor
- some examples of plagiarism include:
  - using code from another source without clearly acknowledging source
  - helping another student to commit plagiarism (e.g., by providing code)
  - posting assignment solutions to any public forum (e.g., public Git repository) during or ***after*** having taken course
- all plagiarism cases ***will be reported to the Department Chair***
- plagiarism offense will result in ***automatic zero grade*** for assignment or project in question
- instructor and teaching assistants ***may, at any time, question student*** regarding any aspect of their submitted work in order to ensure that this work is student's own
- instructor and teaching assistants ***may employ plagiarism-detection tools*** in the review and grading of student work
- help classmates by pointing them in direction of solution but never give them (all or part of) your code

How to Succeed in Course

1. Attend all lectures and tutorials.
2. Do not fall behind in the course.
3. Work ahead whenever possible. Start working on all assignments (and project) as soon as possible.
4. When encountering difficulties, seek help in a timely manner.
5. Read the specifications for each assignment very carefully and check that all requirements are met ***exactly***.
6. Test the code for an assignment ***thoroughly*** at all stages of development.
7. Ensure that each assignment submission passes the precheck.
8. Provide detailed commit log messages that can be understood clearly by others (especially markers).
9. Provide code comments that can be clearly understood by others (especially markers).

Disclaimer

- course is entirely new (i.e., has never been offered before)
- although instructor has made tremendous effort in preparing this course, many aspects of course are expected to be suboptimal, due to factors such as:
  - uncertainty about demographics of students that will take course (e.g., how many students from which majors/programs)
  - uncertainty about programming abilities of students that will take course
  - uncertainty as to whether students are actually taught all of what is supposed to have been taught in prerequisite courses
- furthermore, some problems are inevitable, due to factors such as:
  - unanticipated gaps in student knowledge
  - material taking more or less time to cover than expected
  - material being more or less difficult than expected for typical student
  - unanticipated problems with software development environment
  - unanticipated problems with tools used for assignment grading
- students are strongly encouraged to provide feedback to instructor so that course can be improved (either in current or future offerings)

Questions?

Dying to ask a question?
Here's your chance.

Section 1.2

Software Development Environment and Assignments

Software Development Environment (SDE)

- course uses custom software development environment (SDE)
- course SDE includes (amongst other things) most recent versions of GCC and Clang
- critically important to use course SDE for all assignments
- assignments are graded using course SDE
- to access course SDE, use `sde_shell` or `sde_make_setup` command
- `sde_shell`: starts new subshell configured to use course SDE
- `sde_make_setup`: prints shell commands needed to configure shell to use course SDE so that user may invoke them
- use of `sde_shell` is recommended over `sde_make_setup`, since easier to use
- more information about course SDE can be found at:
  - `https://www.ece.uvic.ca/~mdadams/courses/cpp/#sde`
  - `https://github.com/mdadams/sde`

Assignments

- two types of assignment problems: programming and non-programming
- programming problems require development of code to meet prescribed specifications
- non-programming problems typically require written (i.e., English) answers which may include short code fragments
- programming problems in assignments specified in great detail and typically include requirements related to:
  - organization of code in files and directories (e.g., file and directory names, directory structure, file contents)
  - application programming interfaces (APIs)
  - user-interface (UI) behavior, such as command-line interface (CLI)
  - data formats for program input and output
  - program exit-status conventions
- critically important that all specifications for programming problem met exactly
- if requirements not met exactly, code may fail to build successfully with instructor's test code

GitHub and GitHub Classroom

- GitHub is web-based hosting service for Git repositories (i.e., hosts Git repositories for commercial, open-source, and other software projects)
- GitHub web site: `https://github.com`
- GitHub web site provides mechanism for creating and managing Git repositories for programming assignments called GitHub Classroom
- course uses GitHub Classroom for assignment submission
- each student needs GitHub account
- to create GitHub account, visit: `https://github.com/join`
- student sent email invitation to undertake assignment
- accepting invitation will cause private Git repository to be created for storing assignment submission

Assignment Submission

- assignment submission performed using Git repository in conjunction with GitHub Classroom
- files in Git repository must be organized in very specific manner
- submissions are self-identifying via `IDENTIFICATION.txt` file
- required to provide detailed history of code development (i.e., detailed commit log messages)
- code must be well commented
- assignment submissions must pass validation phase of precheck otherwise automatic grade of zero
- to perform assignment precheck, use command `assignment_precheck`
- late assignment submissions not accepted
- incomplete assignment submissions will be accepted, provided that they pass validation phase of precheck

Assignment Evaluation

- ***code correctness is very important***, as marking scheme for programming problems weights code correctness quite heavily
- code will be built and run through many test cases
- testing uses instructor test code (not student test code)
- critical that code builds (i.e., compiles and links) successfully; otherwise no testing can be performed
- any test that cannot be performed is ***assumed to fail***
- code visually inspected (code itself and comments)
- commit history log messages examined
- code comments and commit log messages must be clearly understandable to others
- as part of evaluation process, each student ***may be questioned*** about their submitted code by instructor or teaching assistant (in order to ensure code is student's own work)

Assignment Solutions

- solutions for non-programming problems usually posted
- solutions for programming problems not posted
- solutions to programming problems not posted for two main reasons:
  1. to avoid bias implicit in advocating one particular correct solution over all others
  2. to eliminate possibility of students in future offerings of course plagiarizing from instructor's solutions
- students welcome to meet with instructor in order to view his solutions to programming problems
- students will not be permitted to make copies of these solutions, however

Advice for Success on Assignments

1. Start working on each assignment as soon as possible.
2. Ensure that each assignment submission passes the validation stage of the assignment precheck as early as possible before the submission deadline.
3. Test code thoroughly at all stages of development.
4. Enable and take notice of compiler warnings.
5. Use code sanitizers.
6. Always double-check that all requirements for the software being developed are met.
7. Ensure that your Git repository contains the correct contents at the submission deadline.
8. Be particularly careful about the const correctness of code.
9. Commit code changes to your Git repository often and with detailed commit log messages.

Software Demonstration

GIVE A DEMONSTRATION OF THE SDE AND VARIOUS SOFTWARE TOOLS.