SENG475 & ECE596C:

Advanced Programming Techniques for Robust Efficient Computing (With C++)

Michael Adams

Department of Electrical and Computer Engineering
University of Victoria
Victoria, BC, Canada
https://www.ece.uvic.ca/~mdadams

Summer 2025

 youtube.com/iamcanadian1867  github.com/mdadams  @mdadams@mastodon.social

---

Course Outline:
https://www.ece.uvic.ca/~mdadams/courses/cpp/#outline

Agenda

1. preamble
   - video conferencing
2. course overview
   - prerequisite background, course topics, learning outcomes
   - general teaching strategy
   - course website and Brightspace site
   - video lectures
   - lecture sessions, office hours, tutorial sessions
   - required textbook and lecture slides
   - computer and software requirements
   - handouts
   - plagiarism and other forms of academic misconduct
   - advice for succeeding in course
3. questions

Section 1.1

Preamble

Joining Zoom Meetings With Single Sign-On (SSO)

- users are required to join Zoom meetings using Zoom Single Sign-On (SSO) with their UVic Netlink credentials (i.e., Netlink username and password)
- use of SSO allows identity of person to be verified using their UVic Netlink credentials
- allowing person to enter meeting anonymously would pose significant security risk (e.g., Zoom-bombing attacks)
- ***if you are placed in waiting room*** instead of being directly admitted into meeting, you ***did not use SSO*** correctly
- users placed in waiting room ***will not be admitted*** to meeting
- therefore, ***anyone who does not use SSO will be blocked from joining meeting***

Video Conferencing Etiquette

- ***always use real (first and last) name*** for your screen name (or you may be removed from meeting)
- always ***use headset*** in order to minimize feedback when microphone is not muted
- always ***mute microphone*** when not speaking
- in larger meetings, always ***disable video camera*** when it is not strictly needed to avoid network bandwidth problems
- unless instructed otherwise, if you have question for meeting host, ***raise your virtual hand*** (accessible via “Participants” on Zoom), rather than interrupting host

Section 1.2

Course Overview

Course Overview

- interdisciplinary in nature (e.g., engineering and computer science)
- explores variety of advanced programming topics
- considers several application areas, such as:
  - geometry processing and computational geometry
  - numerical analysis
  - signal processing
- uses C++ programming language
- employs Linux-based software development environment with GCC and Clang compiler toolchains

Prerequisite Background

- should possess ***good programming skills***
- ***must know rudimentary C++*** (e.g., classes, templates, and standard library)
- 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
- students should complete Assignment 1 as soon as possible
- any student having significant difficulty with Assignment 1 should ***drop course immediately***

Course Topics

- nominally, major topics to be covered (in approximate order) are:
  1. Algorithms
  2. Data Structures
  3. A Few Remarks About Basic C++, Const, Constexpr
  4. Constexpr, Literal Types, and Compile-Time Computation
  5. Value Categories, Moving and Copying, Temporary Objects, and Copy Elision
  6. Error Handling, Exceptions, and Exception Safety
  7. Computer Arithmetic, Interval Arithmetic, and Exact Arithmetic
  8. Memory Management and Container Classes
  9. Cache-Efficient Algorithms
  10. Concurrency
  11. Smart Pointers
  12. Vectorization

Learning Outcomes

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

General Teaching Strategy

- teaching strategy employed based on student feedback collected in last several offerings of SENG475 (and ECE596C)
- course employs ***flipped classroom*** approach to teaching
- students introduced to course materials through ***prerecorded video lectures*** prepared by instructor (which are required viewing)
- then, students given opportunity to engage with course materials in ***interactive lecture sessions*** held by instructor during lecture time slots

Course Website and Brightspace Site

- course employs both course website and Brightspace site
- course website:
  - https://www.ece.uvic.ca/~mdadams/courses/cpp
- primary information source for course is course website, which has all handouts and links to other important information/resources for course
- students should read all information on course website (as failing to do so may cause important course-related information to be missed)
- some areas of course website are password protected
- Brightspace site:
  - https://bright.uvic.ca/d2l/home/415320
- Brightspace site only intended to be used for:
  - posting username and password required to access password-protected areas of course website
  - providing students with means to review their course grades

Video Lectures

- all core instructional content available as ***prerecorded videos*** via instructor's YouTube channel:
  - `https://www.youtube.com/iamcanadian1867`
- students responsible for all material covered in video lectures
- schedule for viewing video lectures provided [web]
- ***critically important to follow this viewing schedule***
- for more information on video lectures, refer to "Video Lectures" section of course website

2019-05 SENG475 Video Lectures

- recordings of lectures from SENG475 in 2019-05 term will be used for delivery of core course content
- ignore any parts of 2019-05 lectures (mainly in Lectures 1 and 37) that deal with course administrative issues such as: student evaluation/assessment, schedules/deadlines, assignments, projects, and exams
- course video-lecture information package available that includes:
  - 2019-09-01-SENG475 edition of lecture slides, which should match slides used in videos reasonably closely [PDF]
  - fully-cataloged list of slides covered in lectures, where each slide in list has link to corresponding time offset in YouTube video where slide is covered
  - numerous supplemental documents referenced by slide deck
- above video lectures also listed in Section D.2 (titled “2019-05 SENG 475 Video Lectures”) of textbook

Supplemental Video Lectures

- additional supplemental video-lecture content available from:
  - Appendix D (titled “Video Lectures”) of textbook, including:
    - Section D.3 (titled “Rudimentary C++”)
    - Section D.4 (titled “Miscellaneous Video Presentations”)
  - “Video Lectures” section of course website
- some of this content also included in course

Lecture Sessions [1]

- lecture time slots will be used by instructor to hold interactive lecture sessions to assist students in learning course materials more effectively
- sessions held ***face-to-face*** with ***provision for online attendance*** (assuming instructor has computer setup necessary to accommodate online attendance)
- some potential uses of lecture sessions include (but are not limited to):
  - discussing more difficult aspects of course materials and addressing common misunderstandings
  - answering student questions about course materials
  - presenting some extra material that is not officially part of course but may help students in job interviews
- students ***not required to attend*** lecture sessions, ***unless explicitly indicated*** by instructor

Lecture Sessions [2]

- students ***strongly encouraged*** to participate in at least some of lecture sessions, as this will likely lead to improved understanding of course materials
- normally, lecture sessions ***will not be recorded*** by instructor
- some reasons for not recording lecture sessions include:
  - main objective of lecture sessions is to provide opportunity ***interactive engagement***, and recording lecture sessions would run completely contrary to this objective
  - recording any interactions with students raises many ***privacy concerns***, which are best avoided whenever possible
  - some students ***may feel uncomfortable*** to participate if being recorded
  - all core instructional content for course already available in video format
- additional information on lecture sessions available from "Lecture Sessions" section of course website

Office Hours

- office hours will be held by instructor in order to provide extra help with course materials as well as discuss other course-related matters with students
- office-hour sessions will be offered ***online only***
- time slot for office hours will be determined by Brightspace survey and posted on course website
- questions about course materials will be answered in main room so that all students can benefit from questions asked
- private/confidential matters will be discussed one-on-one with student in breakout room
- may attend office-hour session simply to listen to questions from other students or comments from instructor
- office hours cancelled during reading break (and on holidays)
- more information on office hours available from "Office Hours" section of course website

Tutorial Sessions [1]

- tutorial sessions run by instructor
- not all tutorial time slots will be used
- each tutorial will be held in one of three formats:
  - face-to-face (in computer lab used by course) with no provision for online attendance
  - face-to-face with provisions for online attendance
  - online only
- format to be used will depend on nature of tutorial content and limitations imposed by A/V and computer systems
- possible uses of tutorial sessions include (but are not limited to):
  - presentations by instructor to further clarify more difficult aspects of course material
  - software demonstrations by instructor
  - student interviews regarding code submitted for programming assignments (to guard against plagiarism)

Tutorial Sessions [2]

- students ***required to attend*** tutorial sessions, ***unless explicitly indicated*** by instructor
- normally, tutorial sessions will not be recorded (for similar reasons as in case of lecture sessions)
- tutorials start in ***first week*** of classes (i.e., this week)
- first tutorial will introduce software development environment amongst other things

Required Textbook and Lecture Slides

- textbook (i.e., C++ programming exercise book):
  - M. D. Adams, *Exercises for Programming in C++ (Version 2021-04-01)*, Apr. 2021, ISBN 978-0-9879197-5-5 (PDF).
- lecture slides:
  - M. D. Adams, *Lecture Slides for Programming in C++ (Version 2021-04-01)*, Apr. 2021, ISBN 978-0-9879197-4-8 (PDF).
- textbook website:
  - `https://www.ece.uvic.ca/~mdadams/cppbook`
- available under Creative Commons (i.e., open-access) license
- textbook and lecture slides can obtained in PDF format from textbook website
- print copies ***not*** available from UVic Bookstore

Computer and Software Requirements

- each student required to have access to all of following software on their own computer:
  - Zoom:
    - for participating in online meetings in course
  - Secure Shell (SSH) client with support for X11 tunnelling:
    - to allow access to lab machines by remote login
  - X11 server:
    - to facilitate remote execution of programs with GUI
- containerized software development enviroment (SDE) for course provided in form of virtual machine (VM) disk image
- each student required to have access to (64-bit x86) computer with VM hypervisor software that can run containerized environment provided by VM disk image (as having access to VM setup will make working on assignments and project much easier)
- VM disk image about 3 to 4 GiB in size (compressed) and contains SSH client, X11 server, and SDE (as well as various other useful software)

Course Outline and Other Handouts

- some handouts available from course website include:
  - undergraduate (SENG475) course outline [web] [annotated PDF]
  - graduate course (ECE596C) outline, which is essentially same as undergraduate course outline [web]
  - online meetings handout [PDF]
  - video-lecture schedule [web]
  - video-lecture information package [Zip] [HTML] [PDF] [video]
  - assignment-assessment handout [PDF] [annotated PDF]
  - assignment general-information handout [PDF]
  - project handout [PDF] [annotated PDF]
  - course-materials bug-bounty program (CMBBP) handout [PDF]
  - course-materials errata handout [text]
  - GitHub authentication handout [PDF]

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 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 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

Randomized Code Interviews

- for each assignment, some number of students will be randomly selected to be interviewed by TA or course instructor
- student will be asked questions about code in assignment submission in order to make determination of whether submission is student's own work
- mostly likely, tutorial time slot will be partially used for purpose of conducting such interviews
- interviews will be conducted one-on-one with student
- if any assignment submissions flagged as suspicious, student will be added to list of students to be interviewed
- since most interviews result from random selection, being selected for interview does not necessarily indicate any suspicion of guilt

Other Remarks

- likely to obtain ***much faster response*** to questions by using lecture sessions and office hours than using email
- students should enable Brightspace notifications (via email) so that course announcements received in timely fashion
- all assignments and projects submitted via GitHub Classroom (in absence of special arrangements)
- advisable for students to work ahead whenever possible to protect against unexpected
- most handouts versioned (i.e., include date on each page) so that newer versions can be distinguished from older ones
- if you downloaded any handouts (including course outline) before day of first lecture, check to ensure that those handouts have not changed since time downloaded

Advice for Succeeding in Course

- do not fall behind
- work ahead to whatever extent possible (to reduce risk of falling behind due to unexpected circumstances)
- consume video content at rate that either meets or exceeds minimum rate specified on video lecture handout
- start working on assignments as soon as possible after GitHub Classroom assignment invitation URL posted
- submit your project proposal early, have it approved early, and start working on project early

Questions

Section 1.3

Software Development Environment and Assignments

Software Development Environment (SDE)

- course uses custom software development environment (SDE)
- course SDE includes (amongst other things) recent (often 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

Accessing Software Development Environment (SDE)

- two ways to access SDE for course:
  1. use lab machines either by console login or remote login via SSH with X11 tunneling (for graphics)
  2. use VM disk image with SDE [except for assignments package]
- each student must be able to use both methods for accessing SDE
- `assignment_precheck` only available on lab machines (not VM disk image)

Secure Shell (SSH) Client Software

- MacOS, Linux, and most other Unix variants typically include SSH client software
- PuTTY [Windows, Unix]
  - website: https://www.chiark.greenend.org.uk/~sgtatham/putty
  - open source implementation of SSH and telnet
  - for Windows and Unix platforms
- MobaXterm [Windows]
  - see slide on X11 server software
- Secure Shell [ChromeOS]
  - extension for Chrome browser
  - https://chrome.google.com/webstore/detail/secure-shell/iodihamcpbpeioajjeobimgagajmlibd
- see also:
  - https://www.uvic.ca/engineering/ece/faculty-and-staff/home/computing/remote-access/index.php
  - https://servicecatalog.engr.uvic.ca/services/ssh/

X11 Server Software

- most Unix variants including Linux but excluding MacOS typically include X11 server software
- MobaXterm [Windows]
  - website: https://mobaxterm.mobatek.net
  - enhanced terminal with X11 server and SSH client
  - free (Home Edition), proprietary, for Windows platform
- Xming [Windows]
  - website: http://www.straightrunning.com/XmingNotes
  - X11 server for Windows
  - open source
- XQuartz [MacOS]
  - website: https://www.xquartz.org
  - X11 server for MacOS
- for ChromeOS, use Crostini (i.e., containerized Linux)
- see also:
  - https://www.uvic.ca/engineering/ece/faculty-and-staff/home/computing/remote-access/index.php

Hypervisor Software

- Oracle VirtualBox [Windows, MacOS, Linux/Unix]
  - website: https://www.virtualbox.org
  - open source with proprietary extensions
  - supports many operating systems (e.g., Linux, MacOS, Windows, and many UNIX variants)
- VMWare Workstation Player [Windows, MacOS, Linux/Unix]
  - website: https://www.vmware.com/ca/products/workstation-player.html
  - proprietary, free for personal non-commercial use
  - supports many operating systems (e.g., Linux, MacOS, Windows, and many UNIX variants)
- GNOME Boxes [Linux/Unix]
  - website: https://wiki.gnome.org/Apps/Boxes
  - open source
  - supports many UNIX-like operating systems (e.g., Linux)
- running arbitrary VM image on ChromeOS-based system likely not possible unless system supports Crostini with nested VMs

VM Disk Images

- use of VM disk image containing SDE has many advantages, including:
  - reduces dependence on lab machines, which may become overloaded or have downtime
  - reduces need for network connection
  - allows use of GUI-based applications that would require too much network bandwidth to use remote display
  - protects against potential breaking changes made by system administrators on lab machines
- VM disk images contain build of Fedora Linux for 64-bit x86 architecture (apologies to anyone with machines based on 32-bit x86 or ARM)
- has X11 server (which is run when graphical desktop started after login)
- has SSH client (i.e., `ssh` program)
- has same SDE as lab machines, except Aristotle which provides `assignment_precheck`
- aside from running `assignment_precheck`, all work can be done using VM disk image

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 website: `https://github.com`
- GitHub website 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
- information on authentication and credential caching for GitHub can be obtained from “GitHub and GitHub Classroom” section of course website

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 validation stage of assignment precheck as early as possible before submission deadline
3. test code thoroughly at all stages of development
4. enable and take notice of compiler warnings
5. use code sanitizers
6. use code coverage tools
7. always double-check that all requirements for software being developed are met
8. ensure that your Git repository contains correct contents at submission deadline
9. be particularly careful about const correctness of code
10. commit code changes to your Git repository often and with detailed commit log messages

Software Demonstration

- if time permits, give demonstration that covers various software tools, such as:
  1. SDE (i.e., `sde_shell`)
  2. Aristotle (i.e., `assignment_precheck`)
  3. YouCompleteMe (YCM)
  4. Vim Language-Server-Protocol (LSP) Plugin
  5. Address Sanitizer (ASan)
  6. Undefined-Behavior Sanitizer (UBSan)
  7. Git
  8. CMake
  9. Lcov

Section 1.4

Course Project

Warning

The information about the course project presented on these slides is ***not intended as a replacement for the project handout***. These slides are only intended to supplement the information on the project handout. Some very important details about the project are only presented on the project handout (and not on these slides). So, it is extremely important to read the project handout carefully (in addition to looking at these slides).

Project

- develop some significant software project using C++ programming language
- SENG475 students can work either individual or in teams
- size of team considered in grading of project
- ECE596C students must work individually
- stages of project work (in order):
  1. preliminary project discussion
  2. formal written proposal
  3. project software, including brief video presentation
- some video presentations for past projects available via links on course website
- past projects only for illustrative purposes, not for simply copying ideas
- since project software due very late in term, penalty for late project software submissions quite severe (grows exponentially with time)
- some students use project as way to market themselves to prospective employers

Skillsets Developed by Project

- designing (as opposed to implementing) software
- designing software interfaces; for example:
  - command-line interface (CLI)
  - text-based user interface (TUI)
  - graphical user interface (GUI)
  - application programming interface (API)
- writing formal specifications of software interfaces
- software project planning
- effective testing
- written communication skills
- oral presentation skills
- further develop proficiency in C++

Preliminary Project Discussion

- each student required to meet with instructor to introduce idea for project
- instructor will assess whether idea has potential to lead to viable project
- student should only to proceed to prepare written proposal if instructor has indicated that project idea seems viable
- preliminary discussion intended to prevent student from spending time preparing proposal for project that is very unlikely or guaranteed not to be approved
- project idea could be rejected for numerous reasons, such as:
  - another student already doing similar project in current offering of course
  - too many similar projects done in past offerings of course
  - idea would lead to project with too much or too little work
  - project could not be meaningfully evaluated by instructor (e.g., not testable)
  - idea not of sufficient relevance to course
- meetings normally take place during office hours or free time during lab/tutorial time slots

Project Proposal [1]

- student required to prepare formal written proposal describing project software to be developed
- proposal must provide ***clear, concise, and complete*** description of project to be undertaken
- should be as short as possible subject to constraint that ***all required information must be included***
- project handout provides very detailed information about what information must be included in proposal
- student forbidden from starting work on project software until proposal approved by instructor (in order to prevent work being done on project that would not be appropriate for course)
- students strongly encouraged to submit proposal ***as early as possible*** (i.e., very far in advance of deadline)
- proposals received after submission deadline will receive ***grade of zero***

Project Proposal 2

- grade for proposal assigned based on first version submitted, regardless of whether proposal approved
- if proposal not approved, need to resubmit revised version
- if revised version must be submitted, ***must use highlighting or change bars*** (or something similar) to clearly indicate directly on the document itself what text has been changed so instructor only needs to re-read parts that have changed
- instructor will not review revised proposal unless all changes to document are clearly indicated
- if number of revisions of proposal becomes excessive, instructor reserves right to deduct further marks from proposal
- grading approach for proposal intended to encourage good job ***on first version submitted***, eliminating need for revision

Project Software

- project software submitted via GitHub Classroom
- particular Git repository layout must be used
- software must build and run ***on lab machines*** using SDE for course
- ***only libraries installed on system*** (as part of base OS install or SDE) can be used (i.e., students cannot include other libraries as part of their project software)
- project software submission required to include URL for video presentation
- submission deadline for project software during final exam period (with specific date posted on course website)
- if early submission deadline met, entire grade for project software scaled by ***multiplicative*** factor (see course website for specific value)
- must notify instructor by email when project software considered submitted for grading

Video Presentation

- student required to prepare brief video presentation for project
- presentation must not exceed 5 minutes
- generally introduce project using some slides
- give brief demonstration of software developed
- be sure to include student name and project title on title slide
- video must be hosted by YouTube
- use either unlisted or public visibility (do not use private visibility)

Common Problems With Project Proposal I

1. extremely vague project specification
   - how many software components being delivered and of what type (e.g., application program, library, etc.)?
   - what do each of software components do and how are they used?
2. what part of software to be developed by student not specified or ambiguous
   - almost all code relies on libraries to some extent
   - extremely important what part of functionality in project software provided by code written by student versus code in libraries
   - has massive impact on feasibility of project and whether project is even appropriate in first place
3. completely missing or incomplete/ambiguous specification of user interfaces
   - instructor will build, run, and test software
   - instructor cannot run software unless user interface (e.g., CLI, TUI, GUI) completely and unambiguously specified

Common Problems With Project Proposal II

- instructor cannot assess suitability of proposed project unless interfaces clearly understood since interfaces can very significantly impact amount of work involved

4. CLI not fully and clearly specified
   - all command-line arguments and options must be fully documented
   - instructor cannot test software if cannot understand clearly how to run programs that are part of project
5. custom file formats used by software not fully and clearly specified
   - instructor cannot generate datasets for testing software if formats not clearly and unambiguously specified
6. if software has GUI, need some rough sketches to give general idea of how interface will work
7. no evidence that student has ensured that project is feasible; for example, failure to identify
   - what parts of what libraries will be used
   - what methods will be used

Common Problems With Project Proposal III

8. failure to check that library to be employed provides functionality required by project
9. failure to check that library to be employed (***including header files***) installed ***on lab machines*** and version of library is ***sufficiently new*** to provide all required features
10. project is not testable
    - software must be defined in manner that make easy testing possible
    - problematic if software computes solution to problem for which there is no practical way to check if results appear reasonable
    - software that consists only of library is not feasible to test (since instructor would have to learn library and write application programs to test library)
11. poor quality writing
    - writing is unclear/ambiguous, poorly organized, overly verbose, etc.
12. poor choices made in design of user interface
    - using interface that constraints users in arbitrary ways for no justifiable reason

Common Problems With Project Proposal IV

- using many cryptic single-letter CLI options (instead of using long option names) **Boost Program Options**

13. not using standard input, standard output, and standard error appropriately
    - if program has single data stream as input and produces single data stream as output, unreasonable to read input from file and write output to file
    - use standard input and standard output
14. failure to consider instructor time constraints
    - reviewing project proposals takes significant time
    - avoid situations that would lead to many students wanting to meet with instructor in short time window of several days

Good Strategy for Quality Proposal

- swap proposals with another student
- other student must have no knowledge of project
- ask them to read proposal and then explain exactly how they would:
  - generate input datasets
  - specify command line options
  - navigate through GUI if software has GUI
  - examine output for correctness
- if other student cannot do these things, this is indication of deficiencies in proposal

Sorting Software Example

- given collection of numbers, sorts numbers, and outputs sorting result
- appropriate for project?
- feasible for project?

Sorting Software Example: Proposal Considerations

- what exactly is meant by “numbers”?
- what is interface with user? GUI? TUI? CLI (e.g., command-line arguments and their meaning, exit-status conventions, etc.)?
- what is sorting criterion?
- is more than one sorting algorithm supported?
- what sorting algorithm is used?
- where does input originate?
- how exactly is input data formatted?
- where is output sent?
- how exactly is output formatted?
- what aspects of program behavior can be controlled?
- are sorting methods implemented by directly by project software or library?

Sorting Software Example: CLI Specification

- project software consists of single application program called `sort`
- specification for CLI might include information similar to output of “man sort”
- for example, see:

  https://man7.org/linux/man-pages/man1/sort.1.html