CSCI 161 review/prep for final

- hopefully we took the next step towards being better software developers and understanding the larger field of comp sci
- introduced many processes and tools to support development: version control (git), debuggers (gdb), modularity/abstraction, testing practices
- introduced programming design concepts: sorting/searching techniques, dynamic data structures and abstract data types (lists, stacks, queues, trees), object oriented programming, inheritance, static/dynamic binding, exceptions/handling
- expanded our knowledge of C++: syntax/semantics for classes, objects, inheritance, friend, try, throw, catch, templates, the STL

next steps...

- hopefully we've opened the door to a much wider range of CS topics, both in software development and the field in general
- second year courses expand the basic required skills and knowlege: systems (251), data structures (260), architecture (261), software engineering (265)
  - note you can often take 310, 331, 370, 375 in second year
- third year courses focus on giving a solid foundation in each of the core areas of CS: interaction (310), web (311), theory (320), languages (330), OO (331), digital (355), operating systems (360), databases (370), systems analysis (375)
- fourth year courses dive deeply into specific specialized areas, with a different mix of topics offered each year

where I'll see you next...

courses I teach and hope to see you in!

- 265 Software engineering: getting way deeper into tools and techniques for larger/team projects
- 330 Programming languages: looking at very different styles of language and the design/implementation of languages
- 4xx Topics courses such as the compiler course or metaprogramming course
- 491: a full year course in which a student carries out a much larger independent research project under the supervision of one prof, with a review committee of two other profs

think about coop 307 next year

- coop involves (paid) work experience in the field, usually 3 summers over the course of the degree
- really good experience both as career prep and academically (programming “for real”)
- the coop office helps with the job search and interview process, and reviews your work terms
- the coop prep course (307) can be taken in second year (basically one hour a week, outside regular class hours)
- before your first work term you essentially need CSCI 160, 161, 162, 260, 265, 370 plus Math 121, 123

the final exam

- April 20th, 1-4pm, building 180 room 134
- paper/pencil exam
- closed book, closed notes, no electronics
- you can bring one double-sided 8.5x11in. “cheat sheet” of your own creation (can be word processed, no font limits)
- likely 7-10 equally weighted questions
- questions can cover anything from the course: lectures, labs, quizzes, project
- there are a few specific topics I will *not* include: *git, linux, make, gdb, file i/o, command line arguments, enums, typedefs, auto*

Question styles

- I try to include a wide mix of question types, but common formats include
  - write a program that ...
  - write a function that...
  - given a class definition, implement method(s)...
  - given a set of class methods, write a function that uses them to ...
  - discuss the advantages or disadvantages of ...
  - explain how/why the following works ...
  - explain why the algorithm below is O(N) (logN, N^2, etc)

Grading of questions

- be sure to check the feedback I've given for your quizzes so far, look for the kinds of situations where you've most often lost marks
- for paper/pencil programming questions I'm less concerned about getting the syntax perfect, and not at all concerned about layout or comments
- if you can't remember the syntax for something, or the name of a specific function you want to use then include a note for me
- for discussion/explanation questions, or anyplace I ask you to justify your answer it's crucial your explanation is clear and detailed *(for a question worth 10-15% of your exam mark I'm probably looking for much more than just a 1 sentence answer)*
- ***during the exam, if you're in doubt about what I want, ask me***

Practice questions

- see the bottom of the exams page

  csci.viu.ca/~wesselsd/courses/csci161/exams.html

- the practice final posted there was prepped for this year's course, hopefully I'll have time to prep/post a second one
- two very old final exams (2013/12) are posted, but I've changed the style and content of the course since then, so don't take them as particularly representative

Key topic areas/advice

- going through the topics, identifying areas I'd typically focus on as final exam questions
- searching/sorting
  - binary vs linear searches
    - knowing these algorithms, their efficiency, and when they can be used are each very important
  - we looked at several sorting algorithms
    - not really interested in having you memorize/reproduce them, but do understand how/why they work, and how to reason about their efficiency (e.g. is it order N^2, order NlogN, and why)
    - I'll typically provide an outline or pseudocode for any specific desired algorithm, and may include some algorithms you haven't seen yet

Dynamic data structures

- remember your C++ syntax for working with pointers, structs, new/delete
- be able to write code to work with:
  - linked lists (inserting and removing from front/back/middle, searching the list, printing the list, deallocating, etc)
  - queues (lists where we insert at back, remove from front)
  - stacks (customized operations: pop, push, top, isempty)
  - binary search trees (inserting, searching, printing recursively using inorder traversals, deleting recursively using postorder traversals)

Classes and inheritance

- syntax for class, field, and method definitions and use
  - use of public, protected, private
  - constructors, destructors, copy/move constructors
  - operator overloading, friend functions/classes
- inheritance, deriving one class from another
  - inheriting as public/private/protected
  - accessing inherited fields and methods
- multiple inheritance and the problem of name clashes
- static vs dynamic binding
  - virtual, override, final keywords, pure virtual methods/abstract base class

Templated classes

- code reuse
- templated functions (declaration/use)
- templated classes
  - declaring the class and providing the templated methods
  - creating/using instances of the class
- the standard template library (STL)
  - creating/using instances of common classes
    - list<string> L; stack<int> S; etc

Exceptions/handling

- difference between exceptions and regular error handling
- using exception handling
- try, throw, catch
- throwing and catching different types of exception
- using a heirarchy of exception classes
- using the std::exception classes