Course Wrap-up (!!)

CSC148, Introduction to Computer Science

Winter 2022

Diane Horton, Jonathan Calver, and Sadia Sharmin

Administrative things

Marking stuff

We will announce when A2 grades are available.

Remark requests for the midterm are done.

Other requests to the course account

- We are quite caught up on these

Course grades

Will likely not be available until early May

- We are super keen to get these done asap, so will be going as fast as we can

Office hours

The regular schedule of instructor and TA office hours ends today.

We will be holding a series of exam prep sessions instead.

- These will be online
- See Quercus for the schedule

Exam Prep sessions

Attendees will lead these.

We encourage you to make requests in advance for

- Old test questions you'd like solved. Tell us which test (what term, midterm or final) and which question.
- Topics you'd like to have reviewed.
- Any other questions you have about the course.

Have the old tests handy to follow along.

You are welcome to attend and just listen.

Preparing for the final

- Re-solve parts of the assignments that your partner did.
- Write code we gave you.
- Re-do / complete lab exercises.
- Run examples from class. Modify them and explore the impact.
- Test out your “what if” questions / theories.
- Make up your own problems.
- Focus on topics you aren’t fully confident in.

Preparing for the final

- Solve the bonus exercises we'll be posting on the Lectures page (including a mock test).
- Solve old tests and finals on the Tests page.
  - NB: Some use content or corners of Python we didn't touch

Practicing on old tests, on paper, is crucial for you this year!

The test itself

- Comprehensive (covers the whole term), including:
  - Class design
  - Inheritance
  - ADTs in general; stack and queue in particular
  - Trees, BSTs, linked lists
  - Recursion
  - Recursive sorting
  - Time-efficiency and big-O analysis
  - Memory models
  - Use of preconditions, assertions, representation invariants

The final

- As before, we'll provide an aid sheet.
  (See the Tests page.)
- There is very little to memorize.
  - It's about understanding, not regurgitation.
- Comments are not required, but may help us give you part marks.

Course Evaluations due Sunday

- Please take time to fill yours out.
- We especially appreciate written comments.

UNIVERSITY OF TORONTO

What have we learned?

Abstraction

A watch

Interface

Implementation

Advantages of separating these:

- Wearer: don't need to understand the mechanism in order to use the watch.
- Maker: can change the mechanism and everyone still knows how to use the watch.

A function

Interface: defined by the function header and docstring

Implementation: the function body

Advantages of separating these:

- Client: don't need to understand the body in order to use the function.
- Implementer: can change the implementation and all client code still works.

A class

Interface: defined by the public attributes and methods

Implementation: the private attributes, private methods, and bodies of all methods.

Advantages of separating these: as before.

A class hierarchy

Interface: the shared public interface defined by the parent class

Advantages of separating these: as before, plus:

- Client: don't even need to know what kind you have!
- Implementer: can even define new kinds and all client code still works!
- This is monumentally powerful.

Abstraction

- If a function provides a consistent interface
  - Client code can ignore implementation and think abstractly.
  - Implementation details change with no impact on client code.
- The same holds for classes and methods.
- Inheritance can capture what classes share.
  - Specifies what any future descendant class must implement.
  - Specifies what every descendant class has.
  - Allows client code to call methods on an object without knowing which kind it is …
  - … even for new classes written in the future!

Abstraction

- ADTs take abstraction up a notch
  - They are above the level of any implementation or language.
  - We saw List, Stack, and Queue.
  - A tree can be though of as an ADT for representing hierarchical information.
  - Or it can be thought of as an implementation detail. E.g., a BST can merely be a good implementation of the Bag ADT. (A bag is like a set but allows duplicates.)

Abstraction

Barbara Liskov won the Turing Award in part for her work on data abstraction.

BARBARA LISKOV

United States – 2008

CITATION

For contributions to practical and theoretical foundations of programming language and system design, especially related to data abstraction, fault tolerance, and distributed computing.

Good design

- Allows plug-out plug-in compatibility.
- Allows polymorphism.
- Think hard about the interfaces to your classes.
  - If you have to change the interface, all client code must change.
- Much more on this in **csc207**, Software Design.

Common structures

- Some new data structures are in your repertoire:
  - Linked list, tree, binary search tree, expression tree
- There are many more:
  - AVL tree, red-black tree, trie, heap, hash table, graph…
- More on this in **csc263,** Data Structures and Analysis.
- There are also algorithm structures:
  - Eg, greedy algorithms (a whole *category* of algorithms)
- More on this in **csc373**, Algorithm Design, Analysis & Complexity

Design by Contract

- We saw a new way of thinking about code.
  - A function or method assumes certain pre-conditions
  - And guarantees certain post-conditions
  - A representation invariant is *always* true
- This is called “design by contract.”
- Design by contract is about pieces of code hooking together properly.

Use of assertions *within* code

- Some algorithms require significant reasoning to invent / understand /assess for correctness.
- Assertions within a piece of code / algorithm can help us express that reasoning.
  - Example from in-place partition:
    Everything in lst[1:small_i] is <= pivot.
    Everything in lst[big_i:] is > pivot.
- More on reasoning about correctness in **csc236**, Introduction to the Theory of Computation.

Run-time analysis

- Some algorithms are much faster than others.
- Some algorithms are so slow they are infeasible.
- Big-oh helps us express such things.
- More on this in **csc236**, Introduction to the Theory of Computation.
- Sometimes the problem itself is infeasible.
- More on that in **csc373,** Algorithm Design, Analysis & Complexity and **csc463,** Computational Complexity and Computability.

Complexity theory

Steven Cook (UofT Professor Emeritus) won the Turing Award for foundational work.

PHOTOGRAPHS

BIRTH:

1939, Buffalo NY

EDUCATION:

STEPHEN ARTHUR COOK

Canada – 1982

CITATION

For his advancement of our understanding of the complexity of computation in a significant and profound way. His seminal paper, "The Complexity of Theorem Proving Procedures," presented at the 1971 ACM SIGACT Symposium on the Theory of Computing, laid the foundations for the theory of NP-Completeness. The ensuing exploration of the boundaries and nature of NP-complete class of problems has been one of the most active and important research activities in computer science for the last decade.

Trade-offs

- Time vs time
  - E.g., sorted lists require slower insert and delete than unsorted lists, but have faster search.
  - E.g., balanced search trees require slower insert and delete than regular search trees, but have faster search.
- Time vs space
  - E.g., we saw several augmented data structures in lectures and labs. We added attributes to save time.
- Speed vs accuracy
  - E.g., sometimes we opt fo a “good enough” vs optimal solution.

Gaining confidence in our code

- Doctest examples are very important.
- But we need to design thorough test cases.
- For complex code, this is not easy.
- Tools like pytest help us implement testing.
- Property-based testing lets us easily check more cases, but only check for properties.
- Assertions and proofs are valuable for very tricky algorithms.

Your software tools

- You've started to gain proficiency with
  - A professional IDE
  - A unit testing framework
  - A property-based testing framework
  - The debugger
- This is part of what makes you a professional.
- More on this in **csc207**, Software Design.

Your thinking tools

- You've also added new tools for thinking about:
  - Data
  - Algorithms
  - Code design
  - Analysis of correctness
  - Analysis of algorithms
- Knowing when and how to use them is part of what makes you a computer scientist.
- The journey is just beginning!

Code aesthetics is communication

"Programs must be written for people to read, and only incidentally, for machines to execute."

Harold Abelson

Documentation is communication

"The most important single aspect of software development is to be clear about what you are trying to build."

Bjorne Stroustrup

Software design is communication

"There are two ways of constructing a software design: One way is to make it so simple that there are obviously no deficiencies, and the other way is to make it so complicated that there are no obvious deficiencies. The first method is far more difficult."

Sir Charles Antony Richard Hoare

Foundations outside CS

- Calculus is important, for example, for
  - Machine learning, computer graphics, computational finance, …
- Linear algebra is important, for example, for
  - Machine learning, computer vision, medical imaging, …
- Statistics is important, for example, for
  - Machine learning, data mining, computational finance, user-experience (UX) studies, CS education research, …

Foundations outside CS

- Embrace opportunities to strengthen skills in
  - communication
  - teamwork
- Employers value these greatly
- Embrace opportunities to explore other interests
- No matter how “far” from CS, there are ways to connect.
- And regardless, your interests matter!

Payoff in 3rd and 4th year

- csc320, 420 / 418: image processing / graphics
- csc321, 411: machine learning
- csc369, 469: operating systems
- csc343, 443: databases
- csc488: compilers
- csc436, 446, 456, 466: numerical computing
- csc404: computer games
- csc490+454@DCSIL: entrepreneurship
- Etc . . . .

UofT has another Turing Award

Geoff Hinton won the Turing Award for breakthroughs in machine learning.

GEOFFREY E HINTON

Canada – 2018

CITATION

For conceptual and engineering breakthroughs that have made deep neural networks a critical component of computing.

Each winner has a Turing Award lecture you can watch. See https://amturing.acm.org.

Maximizing your experience

Research opportunities

In the Department of Computer Science (DCS) https://web.cs.toronto.edu/undergraduate/mentorship-research-work - research

- For course credit: csc299Y, csc399&, csc490H, csc494/5H
- For pay: Undergraduate Summer Research Program
- PRISM program: **P**reparation for **R**esearch through **I**mmersion, **S**kills, and **M**entorship

Research seminars

- DCS Distinguished Lecture Series
  https://web.cs.toronto.edu/dls
- DCS events more broady
  https://web.cs.toronto.edu/events
- Schwartz Reisman Institute for Technology and Society
  https://srinstitute.utoronto.ca/events

Chad Jenkins

Semantic Robot Programming... and Maybe Making the World a Better Place

Thursday, January 13, 2022 | 11AM – 12PM ET

Abstract:

The visions of interconnected heterogeneous autonomous robots in widespread use are a coming reality that will reshape our world. Similar to "app stores" for modern computing, people at varying levels of technical background will contribute to "robot app stores" as designers and

Regina Barzilay

Modeling Chemistry for Drug Discovery: Current State and Unsolved Challenges

Thursday, January 28, 2021

Abstract:

Until today, all the available therapeutics are designed by human experts, with no help from AI tools. This reliance on human knowledge and dependence on large-scale experimentations result in prohibitive development cost and high failure rate. Recent developments in machine learning

SRI Kitchen Table: What is freedom of expression in the digital age?

MAR 10, 2022

Other opportunities

- DCS Alumni-Student Mentorship program
  https://web.cs.toronto.edu/mentorship
- DCS Student Ambassadors
  https://web.cs.toronto.edu/undergraduate/ambassadors-program
- CSSU
  https://www.cssu.ca/
- Arts and Science Internship Program (ASIP)
  https://www.artsci.utoronto.ca/current/academics/asip
- Read the CS Undergrad Update
- Enrol in the Quercus shell for CS undergrads
  https://q.utoronto.ca/enroll/EMFCPB

My #1 advice: Get to know people!