CSCE 240: Advanced Programming Techniques
Lecture 29: Wrap-up and Conclude

PROF. BIPLAV SRIVASTAVA, AI INSTITUTE

25th APRIL 2024

Carolinian Creed: "I will practice personal and academic integrity."

Credits: Some material reused with permission of Dr. Jeremy Lewis. Others used as cited with thanks.

Organization of Lectures 29

- Introductory section
- Main section
  - Common project presentation
  - Course Recap
    - Course goals
    - Highlights
      - Lectures
      - Homework assignments and peer-evaluation
      - Programming assignment
  - Future: Programming, Research, AI
- Concluding section
  - Ask me anything

Introduction Section

Last Class

- PA #5 and Quiz-2 marks were earlier posted
- Grades to be posted by Tuesday, April 30
    - PA #6 report marks remaining

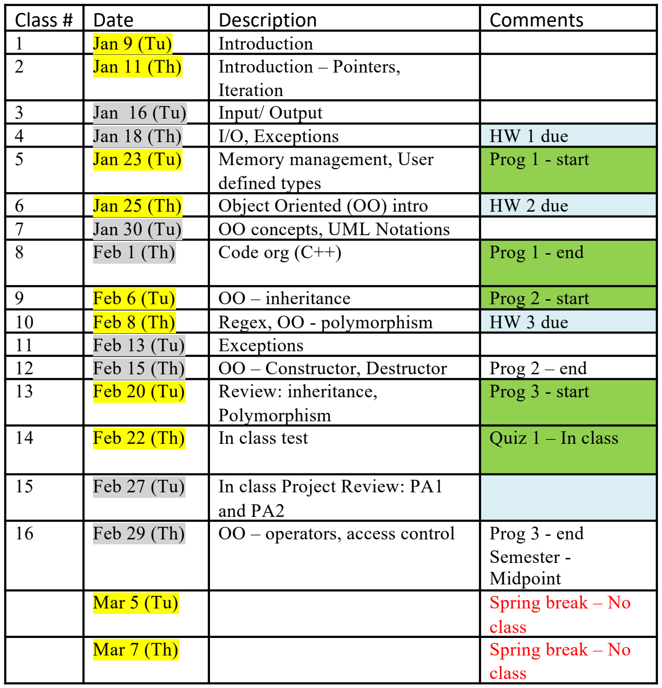
Main Section

Course Wrap-Up

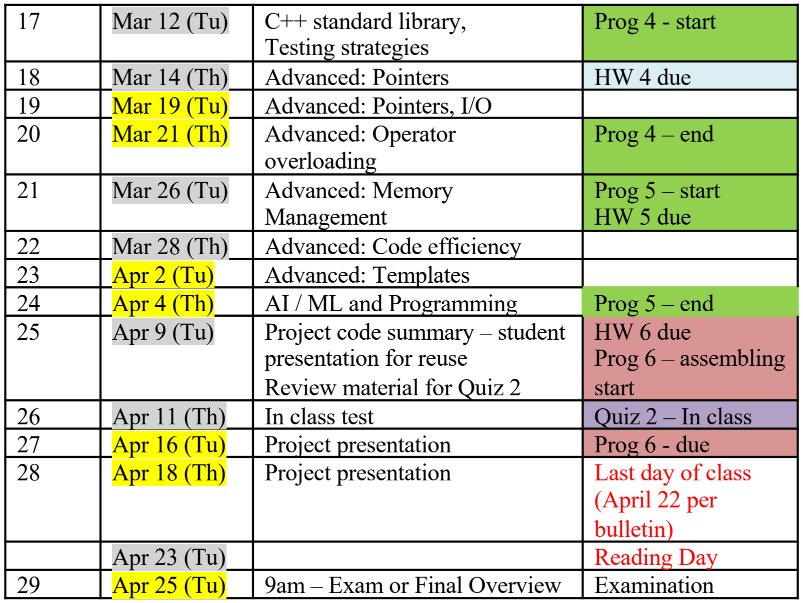
Learning Objectives

- Develop language-independent understanding of programming concepts by being exposed to multiple languages (C++, Java, Python)
- Independently design and implement programs in multiple language of choices (C++, Java or Python based on choice) in a Unix environment
- Demonstrate mastery of pointers, iterators, memory management including object creation and destruction, and parameter passing in C++
- Demonstrate mastery of object-oriented programming concepts including: inheritance, polymorphism, operator overloading, template functions and classes, and the use of STL containers.
- Develop object-oriented models using UML
- Able to work in programming teams with code review and walk throughs
- Solve practical problems that matter

Lectures: Topics Covered and In-Scope



Class #	Date	Description	Comments
1	Jan 9 (Tu)	Introduction	
2	Jan 11 (Th)	Introduction – Pointers, Iteration	
3	Jan 16 (Tu)	Input/ Output	
4	Jan 18 (Th)	I/O, Exceptions	HW 1 due
5	Jan 23 (Tu)	Memory management, User defined types	Prog 1 - start
6	Jan 25 (Th)	Object Oriented (OO) intro	HW 2 due
7	Jan 30 (Tu)	OO concepts, UML Notations	
8	Feb 1 (Th)	Code org (C++)	Prog 1 - end
9	Feb 6 (Tu)	OO – inheritance	Prog 2 - start
10	Feb 8 (Th)	Regex, OO - polymorphism	HW 3 due
11	Feb 13 (Tu)	Exceptions	
12	Feb 15 (Th)	OO – Constructor, Destructor	Prog 2 – end
13	Feb 20 (Tu)	Review: inheritance, Polymorphism	Prog 3 - start
14	Feb 22 (Th)	In class test	Quiz 1 – In class
15	Feb 27 (Tu)	In class Project Review: PA1 and PA2	
16	Feb 29 (Th)	OO – operators, access control	Prog 3 - end Semester - Midpoint
	Mar 5 (Tu)		Spring break – No class
	Mar 7 (Th)		Spring break – No class



17	Mar 12 (Tu)	C++ standard library, Testing strategies	Prog 4 - start
18	Mar 14 (Th)	Advanced: Pointers	HW 4 due
19	Mar 19 (Tu)	Advanced: Pointers, I/O	
20	Mar 21 (Th)	Advanced: Operator overloading	Prog 4 – end
21	Mar 26 (Tu)	Advanced: Memory Management	Prog 5 – start HW 5 due
22	Mar 28 (Th)	Advanced: Code efficiency	
23	Apr 2 (Tu)	Advanced: Templates	
24	Apr 4 (Th)	AI / ML and Programming	Prog 5 – end
25	Apr 9 (Tu)	Project code summary – student presentation for reuse Review material for Quiz 2	HW 6 due Prog 6 – assembling start
26	Apr 11 (Th)	In class test	Quiz 2 – In class
27	Apr 16 (Tu)	Project presentation	Prog 6 - due
28	Apr 18 (Th)	Project presentation	Last day of class (April 22 per bulletin)
	Apr 23 (Tu)		Reading Day
29	Apr 25 (Tu)	9am – Exam or Final Overview	Examination

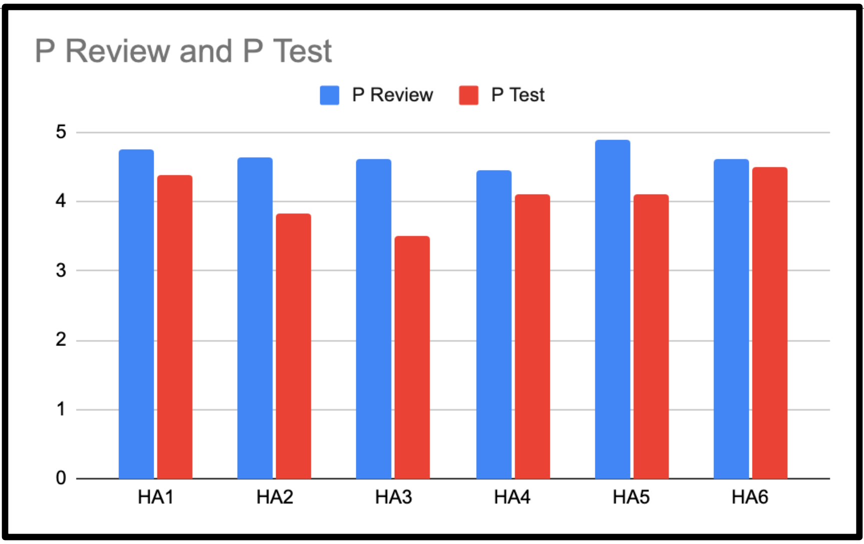
Lecture Logistics

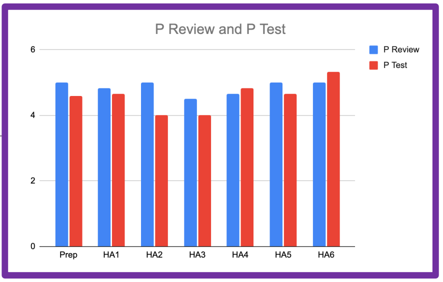
- Material on github
- Code in C++
  - Java (and Python too whenever feasible)
- Homeworks (6) in C++; peer evaluated
- Prog. Assignments (6) in C++/Java /Python
- Quizzes (2) – in-class and pseudo-code
- Other practices encouraged
  - Hackathons
  - AI

Homework Assignments and Peer-Evaluation

- HA1 to HW6, all in C++
- Maximum improvement achieved
  - Peer review (4.75 -> 4.6; ↓)
    Peer test (4.375 -> 4.5; ↑)
  - More than what is reflected in numbers, expectations from code increased
- Why
  - Peer review scores came down slightly (to 4.4; 7%) before going up again
  - Peer test scores came down by 1 point (23%; drastic) before going up again
- Caveat
  - Each HW was different
  - Small change in number of students taking each HW

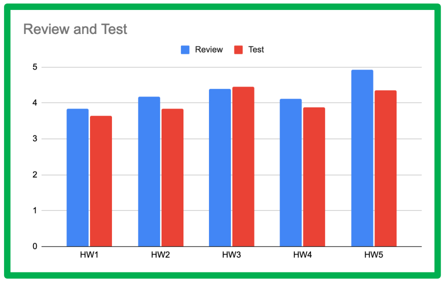
Still, the results are encouraging!





2023 Spring

2022 Spring



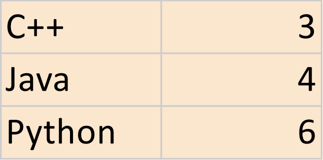
At least 1 point improvement
with min participation: 32
(max class strength: 50+)

- Peer review (3.83 -> 4.92; 28.3% ↑)
- Peer test (3.64 -> 4.35; 19.4% ↑)

Many Interesting Insights

- Individual initiative to learn – from project (2024)
  - Similarity concepts: Simpson coefficient (Szymkiewicz–Simpson coefficient), related to Jaccard coefficient
  - Spelling correction library
  - Optimization of regex, Q-A pairs
  - Dynamic fetching / parsing of content, HTML
  - Swing-based UI

- Choice of languages
  - First time Python was most preferred



C++	3
Java	4
Python	6

Insights Trends

- Initiative to learn – from project (2023)
  - Spelling correction library
  - Optimization of Q-A pairs
  - Dynamic fetching of content, HTML
  - Swing-based UI
- Choice of languages

C++	1
Java	5

- Initiative to learn – from project (2022)
  - Synonyms of terms, to detect intents better
  - Comparison at level of letters, to handle noisy text
  - Handling additional languages – Spanish
  - New UML diagramming tool – Mermaid - https://mermaid-js.github.io/mermaid/#/
  - Grouping concepts to answer higher concepts (knowledge graph)
  - Trying multiple programming languages for different project assignments
- Choice of languages

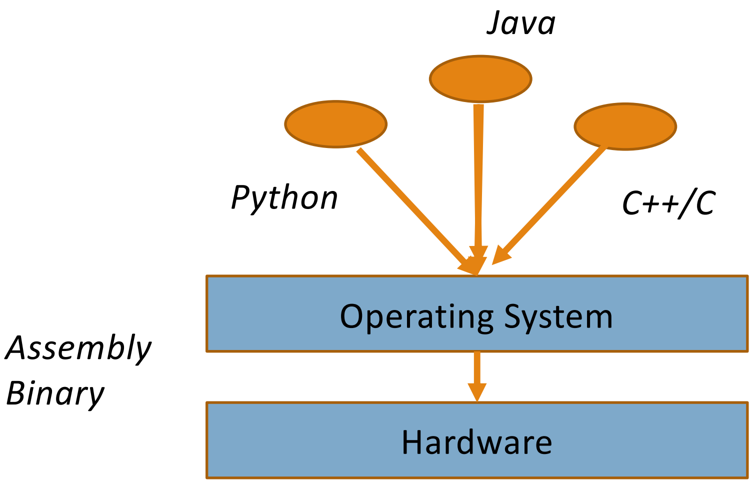
C++	12
Java	26
Python	6

Future: Programming, Research, AI

Programming Techniques

- Languages inevitably change over time
- Code practices remain
  - Adopt a language as mother tongue
    - Understand concepts in-depth
    - Experiment and settle on a coding style
  - Programming: variable initialization, understanding types, … usage of libraries
  - Memory: using just the right amount
  - Algorithms: focusing on efficiency
  - Documentation
  - Debugging methods
  - Testing, …

Programming languages are really for ***communicating among developers*** for building systems on OS/Hardware collaboratively



Research: Complexity of Code

Source: From code complexity metrics to program comprehension, Dror Feitelson, CACM May 2023

- "***Developers spend 58-70% time understanding code, 5% editing it***"
- Hundreds of metrics to measure complexity – independent of language
  - Lines of code
  - Branching factor
  - …
- Shift towards understandability of code
  - Background and assumptions (about computers, problems, language constructs, performance, …) impact understanding of code

“Technical Debt” in Software Engg

- Technical debt, a metaphor introduced by Ward Cunningham in 1992 to help reason about the long term costs incurred by moving quickly in software engineering.
  - As with fiscal debt, there are often sound strategic reasons to take on technical debt. Not all debt is bad, but all debt needs to be serviced.
- Technical debt may be paid down by **refactoring code, improving unit tests, deleting dead code, reducing dependencies, tightening APIs, and improving documentation**.
  - The goal is not to add new functionality, but to enable future improvements, reduce errors, and improve maintainability.
  - Deferring such payments results in compounding costs.
  - Hidden debt is dangerous because it compounds silently.

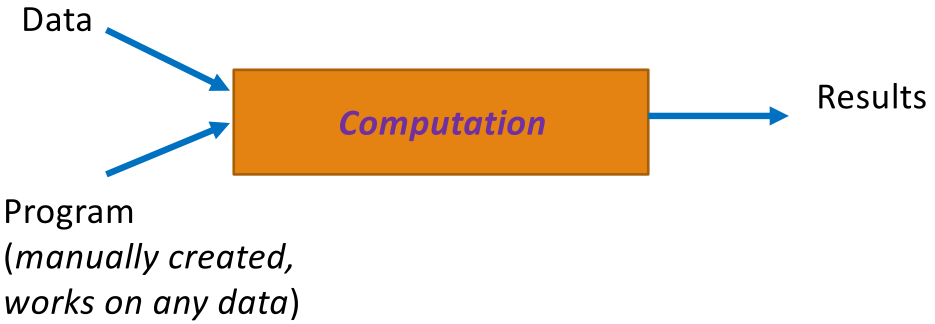
Reference

- D. Sculley, Gary Holt, Daniel Golovin, Eugene Davydov, Todd Phillips, Dietmar Ebner, Vinay Chaudhary, Michael Young, Jean-Francois Crespo, and Dan Dennison. 2015. Hidden technical debt in Machine learning systems. In Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 2 (NIPS'15). MIT Press, Cambridge, MA, USA, 2503–2511.
- M. Fowler. Refactoring: improving the design of existing code. Pearson Education India, 1999.

Research: Where is AI in IT?

- People have traditionally changed themselves to use IT
  - Examples: Typing, fixed menus
  - Focus on repeatability, user control
  - Disadvantage: usage barrier, entry barrier
- With AI: IT changing to enable people to use them naturally
  - Example: Natural language based interaction … chatbot
  - Focus on dynamicity, data-driven behavior
  - Disadvantage: hard to debug, audit and establish accountability

Traditional Programming v/s Machine Learning

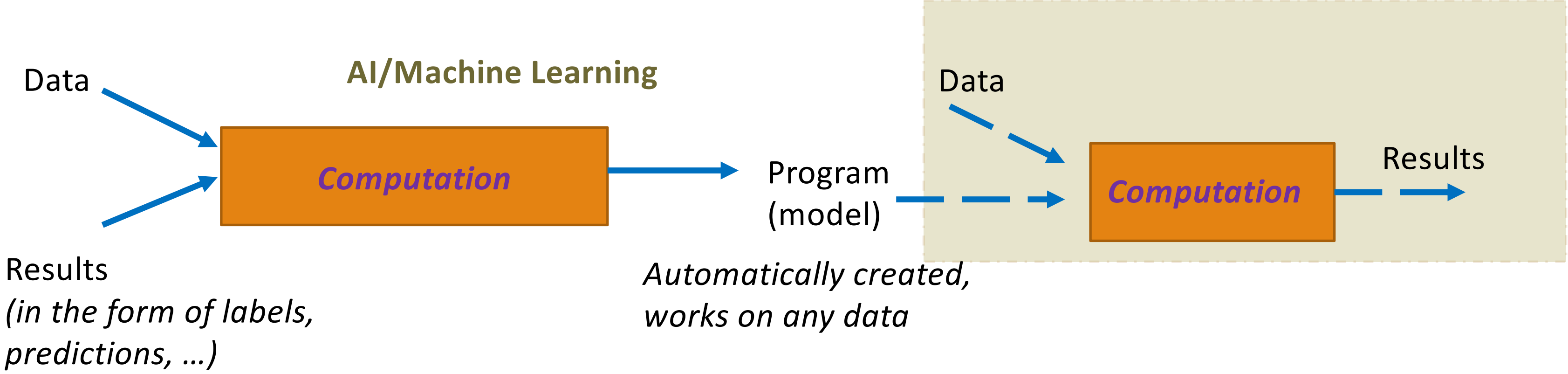


Data

Computation

Results

Program
(*manually created,
works on any data*)



AI/Machine Learning

Data

Computation

Results
*(in the form of labels,
predictions, …)*

Program
(model)

*Automatically created,
works on any data*

Data

Computation

Results

Programming Trends

- Expect more languages to improve developer productivity
  - But good developers understand the underlying operating environment and have sound programming technique
- Expect more automatic code-generation
  - Example: OpenAI's Co-pilot: https://copilot.github.com/
  - Example: ChatGPT
- Automatic software generation is a long-established area
- AI in programming, and programming for AI will grow

Trends

- Stack Overflow: bans answers by ChatGPT –
  - Details: https://meta.stackoverflow.com/questions/421831/temporary-policy-chatgpt-is-banned
  - Overflow of wrong answers
- ChatGPT helps programmer productivity
  - Especially beginner programmers
  - Details: Measuring GitHub Copilot's Impact on Productivity, CACM 2024 https://cacm.acm.org/research/measuring-github-copilots-impact-on-productivity/

Ask Me Anything