CSE 250
Lecture 0

Course Overview, Logistics, Intro

Course Overview

Who we are

- Oliver Kennedy [okennedy@buffalo.edu]
  - Tentative Office Hours (Database, Data, or HEMA Questions also welcome)
    - Capen 212: Weds 11:00-1:00

- Eric Mikida [epmikida@buffalo.edu]
  - Tentative Office Hours
    - Capen 212: M/T 1:00 - 3:00, W 3:00-5:00

Please keep discussions on Piazza (private posts exist)

Always include [CSE-250] in the subject line when emailing

212 Capen: Take these elevators, then turn right.

Who are the TAs

Undergraduate TAs

Senior TAs

- Hope Kara
- Vrund Patel
- Sean Grzenda
- Andrew Schick
- Heba Mahran
- Nawar Khouri
- Jacky Lin
- Amelia Graca
- Anton Kalinin
- Joey Poblete
- Thinh Ho
- Riad Mukhtarov
- Kyle Geffner
- David Lam
- Kartike Chaurasia
- Tirth Shah
- Dikshit Khandelwal

Graders

- Sphoorthi Keshannagari
- Vindhya Nuthalpati
- Rohit Joseph

Logistics

- **Course Forums + Live Q&A**: Sign up for Piazza
  - https://piazza.com/buffalo/fall2022/cse250
- **Course Website / Syllabus:**
  - https://odin.cse.buffalo.edu/teaching/cse-250/2022fa/
- **Assignment Submission**: Autolab
  - https://autograder.cse.buffalo.edu
- **Assignment Distribution**: Github Classroom

Development Environment

- Supported Operating Systems
  - MacOS
  - Ubuntu Linux
  - Windows + WSL/Ubuntu
- Supported Dev Environments
  - Emacs + Scala-SBT
  - IntelliJ (Community Edition is Free) + Scala Plugin
    - https://www.jetbrains.com/community/education/

Other setups are ok, but the more your setup differs, the lower the chance we'll be able to help you

What is Data Structures?

What is a Data Structure?

What is a Data Structure?

SameData

Different Container

more defensible

What is a Data Structure?

SameData

more efficient access to skritches()

Different Container

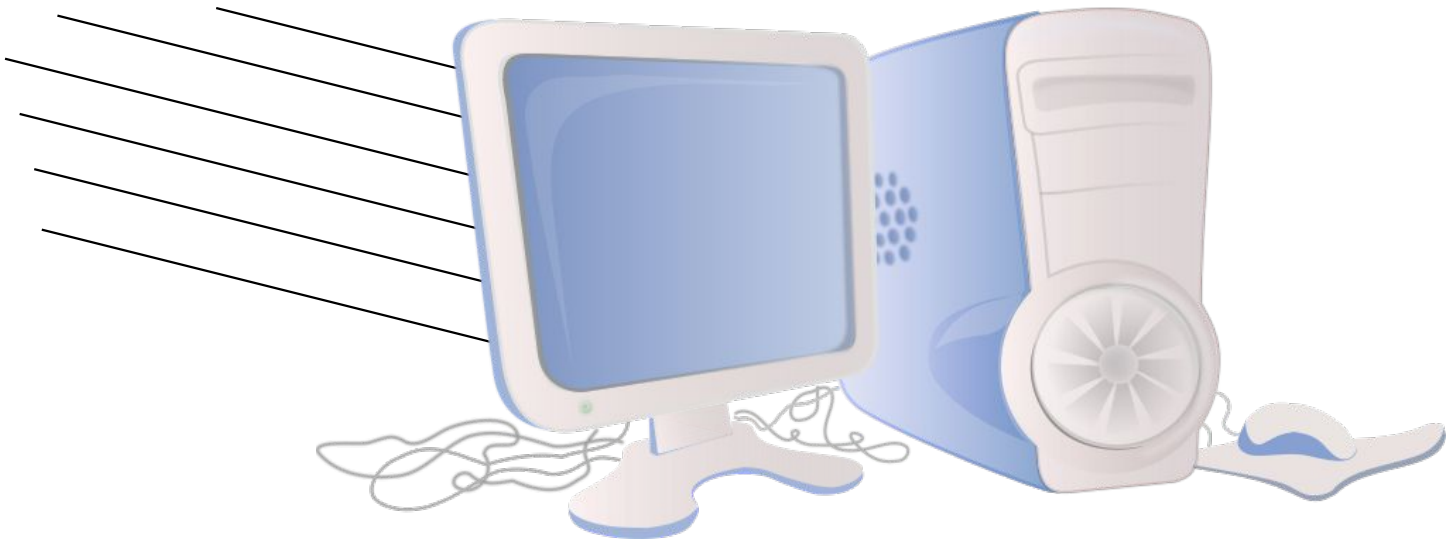
What is a Data Structure?

- Store a list of things in some order (“List”)
  - Array
  - LinkedList
  - ArrayBuffer

- Store things organized by an attribute (“Map”, “Dictionary”)
  - Hash Table
  - Binary Search Tree
  - Red-Black Tree

Why should I care?

How do I make my code efficient?



(image: openclipart.org)

How do I make my code efficient?

- **Tactical**: Optimize your Code (“reducing the constants”)
  - Understand the memory hierarchy
  - Understand the CPU / OS

- **Strategic**: Optimize your Design (“reducing the complexity”)
  - Understand how your algorithm scales
  - Understand repetition in your code

CSE 250

Example

- You have:
  - A list of UBIT / Grade pairs
  - A list of UBIT / Person # pairs

- You want:
  - A list of Person # / Grade pairs

Example

- Option 1
  - For every UBIT / Person# pair
    - Find the UBIT in the UBIT / Grade dataset
    - Output the name/email

- Option 2
  - Store the UBIT / Grade pairs in a dictionary (UBIT as key)
  - For every UBIT / Person# pair
    - Look up the UBIT
    - Output the name/email

Extra work upfront

Pays off inside the loop

Which is better?

Example

(Option 2 is called a “hash join”)

(**~8 of the top 10** Fortune 500 software companies have a database product)

Demo

(thanks to Prakshal Jain; 2021 TA for the suggestion/prototype)

The Arcane Lore of Data Structures

`std::piecewise_construct, std::forward_as_tuple(std::move(key)), std::tuple<>()`.
When the default allocator is used, this means that key_type must be *MoveConstructible* and mapped_type must be *DefaultConstructible*.

No iterators or references are invalidated.

Parameters

key - the key of the element to find

Return value

Reference to the mapped value of the new element if no element with key key existed. Otherwise a reference to the mapped value of the existing element whose key is equivalent to key.

Exceptions

If an exception is thrown by any operation, the insertion has no effect

Complexity

Logarithmic in the size of the container.

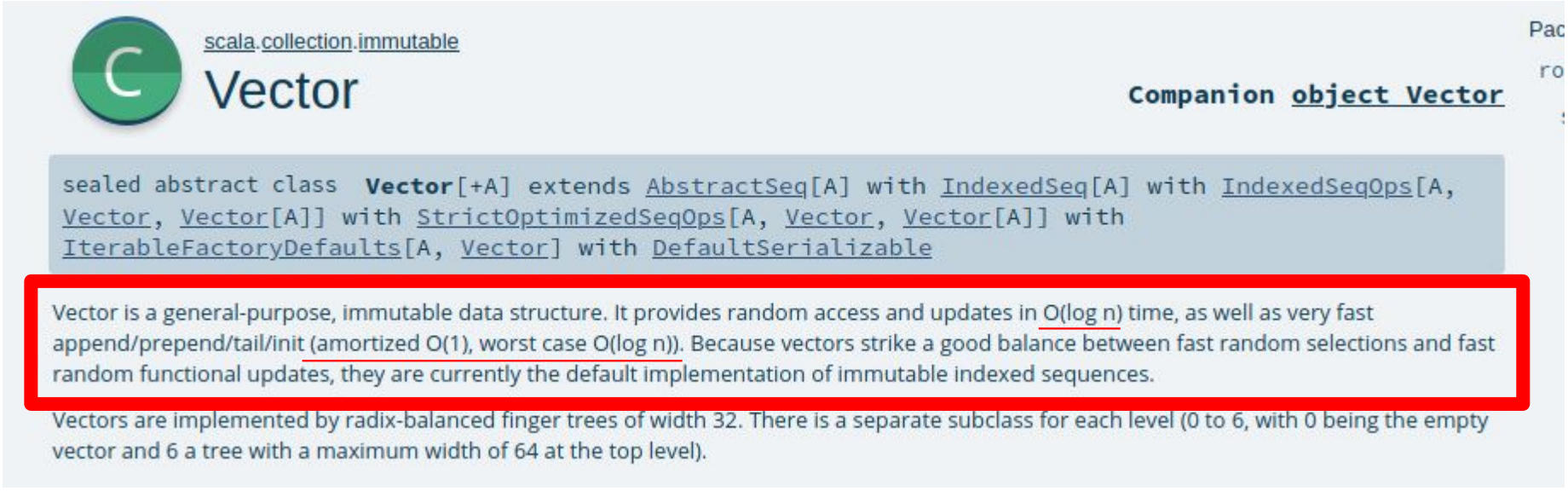
Notes

In the published C++11 and C++14 standards, this function was specified to require mapped_type to be *DefaultInsertable* and key_type to be *CopyInsertable* or *MoveInsertable* into `*this`. This specification was defective and was fixed by LWG issue 2469, and the description above incorporates the resolution of that issue.

However, one implementation (libc++) is known to construct the key_type and mapped_type objects via two separate allocator `construct()` calls, as arguably required by the standards as published, rather than emplacing a value_type

(screenshot: cppreference.com)

The Arcane Lore of Data Structures



scala.collection.immutable

Vector

Companion object Vector

```
sealed abstract class Vector[+A] extends AbstractSeq[A] with IndexedSeq[A] with IndexedSeqOps[A,
Vector, Vector[A]] with StrictOptimizedSeqOps[A, Vector, Vector[A]] with
IterableFactoryDefaults[A, Vector] with DefaultSerializable
```

Vector is a general-purpose, immutable data structure. It provides random access and updates in O(log n) time, as well as very fast append/prepend/tail/init (amortized O(1), worst case O(log n)). Because vectors strike a good balance between fast random selections and fast random functional updates, they are currently the default implementation of immutable indexed sequences.

Vectors are implemented by radix-balanced finger trees of width 32. There is a separate subclass for each level (0 to 6, with 0 being the empty vector and 6 a tree with a maximum width of 64 at the top level).

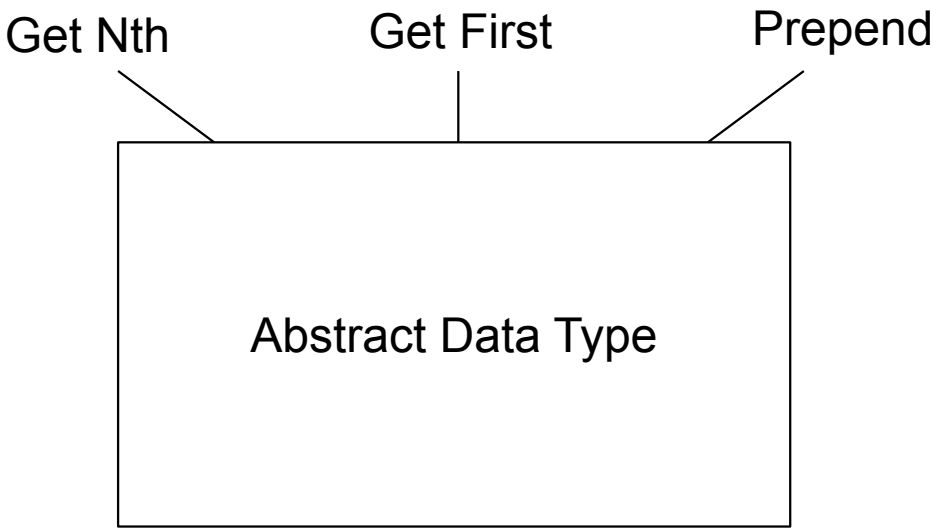
(screenshot: https://www.scala-lang.org/api)

The Arcane Lore of Data Structures

- Every (good) standard library's provides guarantees on the complexity of its data structures' operations

- Understanding complexity bounds can be the difference between code that runs in 6 hours vs code that runs in 8 seconds.

Containers



Containers

- Option 1 (Linked List)
  - Very fast Prepend, Get First
  - Very slow Get Nth
- Option 2 (Array)
  - Very fast Get Nth, Get First
  - Very slow Prepend
- Option 3 (ArrayBuffer... in reverse order)
  - Very fast Get Nth, Get First
  - Occasionally slow Prepend

Which is better?

Data Structures in a Nutshell

More work now

vs

More work later

Topics Covered - Tools

- **Specific Data Structures/ADTs** (organizational strategies)
  - Collection Types (Lists, Arrays, Vectors, Sets, Heaps)
  - Maps (Hash tables, Search trees)
  - Graphs

- **Algorithms** (recipes for standard tasks)
  - Collection queries / updates
  - Sort
  - Graph/Tree Traversal

Topics Covered - Techniques

- Pseudocode
  - Top-down algorithm design
- Algorithm Analysis / Asymptotic Notation
  - Understand algorithm scaling
- Recursion
  - Expressing tasks in terms of themselves

Topic Order

- Scala
- Asymptotic Notation
- Sequence Collections
- Recursion
- Graphs
- Specialized ADTs
- Hash-based Data Structures
- Advanced Topics

Course Syllabus

Grading

- Grade Breakdown
  - Assignments: 30%
  - Participation: 10%
  - Midterm: 20%
  - Final Exam: 40%

Score (x)	Letter Grade	Quality Points
90% ≤ x ≤ 100%	A	4
85% ≤ x < 90%	A-	3.67
80% ≤ x < 85%	B+	3.33
75% ≤ x < 80%	B	3
70% ≤ x < 75%	B-	2.67
65% ≤ x < 70%	C+	2.33
60% ≤ x < 65%	C	2
55% ≤ x < 60%	C-	1.67
50% ≤ x < 55%	D	1
0% ≤ x < 50%	F	0

Written Assignments

- ~ Bi-Weekly Written Assignments.
  - Expect to spend about a week working on it
  - Submit up to 24 hours after deadline with a 50% penalty

- You are responsible for submission format
  - Submit only PDFs
  - Submissions that do not load will receive a 0

- We recommend writing solutions by hand
  - Better retention of what you have written
  - Easier to write math

Programming Assignments

- Grading for most projects
  - Write Test Cases (~15/100 points)
    - Submit as many times as you like
  - Test Submission (0/100 points; predicts ~50/85 points on final tests)
    - Submit as many times as you like
  - Final Tests (~85/100 points)
    - Tests run once after the deadline passes
- Your grade is based on the **LAST** submission you make

Programming Assignments

- You'll have 2-3 weeks per submission
  - Plan to start early and work throughout
  - A 25% penalty per day late, up to 48 hours
  - Bonus for early submissions (up to 5/100 points)

- 3 "grace days" for the semester
  - Applied automatically, even if your score does not increase

Programming Assignments

- Implementation in **Scala 2.13.8**

- Submissions MUST WORK on the grader.
  - If your code does not compile, the submission gets a 0
  - If your code relies on an external library, it gets a 0
  - If your code targets Scala 2.12 or earlier, it might get a 0

Exams

- One In-Class Midterm (Wednesday, **October 19**)
  - Content Coverage is roughly Weeks 1-7 in the syllabus
  - More details as the exam approaches

- One Final Exam (Monday, **December 12; 7:15-10:15;** 😕)
  - Comprehensive (all topics are fair game)
  - Determine if you have a conflict ASAP
  - If HUB changes, trust the date in HUB

- If you need accommodations, contact Accessibility Resources ASAP.

Attendance / Participation

- Lecture
  - No recorded attendance (unless you make me)
  - You are paying $$ to be able to ask questions live (don’t waste it)

- Recitation
  - Recitations start Tue, Sept 6 (Next week)
  - Attendance is mandatory

Collaboration, AI, Resources

Collaboration

- Do…
  - Work together to brainstorm ideas
  - Explain concepts to each other
  - Include a list of your collaborators on all submitted work
- Do NOT…
  - Write solutions when working together
  - Describe the details of solutions to problems or code
  - Leave your code in a place where it is accessible to another student

- If in doubt, ask a member of course staff

Resource Policy

- Do…
  - Use materials provided by course staff (Piazza, Class, OH)
  - Use materials from the course textbook or readings
  - Cite all materials you reference for written work
  - **Cite sources** for all code you reference / copy

Resource Policy

- Do NOT…
  - Reference random videos on YouTube that “helped you solve the problem”
  - Hire “private tutors”
    - Save the money from Chegg
    - If you’re not doing the work yourself, you’re not learning
    - If you have an actual tutor, contact course staff
  - Reference exact solutions found online

If we catch you using unauthorized resources, you get an F

Other Ways to Get an F

- Work in a group by assigning each person to a problem
- Copying your friend's homework because you forgot
  - Each homework is not worth a lot on its own
- Sharing your homework with your friend
  - I have no way to know who did the work and who shared
- Submitting work without citations
  - Citing outside work will help you avoid AI repercussions
  - (we grade you on the work you did, but you won't get an AI violation)

Other Ways to Get an F

You are liable/punishable if someone else submits your work as their own.

Ways to Avoid an F

I will grant amnesty for any AI violation IF you tell us about it before we discover it

Life Lesson

If the only skill you gain from this class is searching for answers on Google…

… you will not be employable for very long.

Why does AI Matter?

- Solutions to problems from this class do exist.
  - Learning requires simplified problems
  - The goal is to get **you** to think through the solution.
- Avoid “cargo-culting”
  - You can’t understand why the solution/design is the way it is from just the solution
- Solutions to problems from the real world usually do not exist
  - Stack Overflow cannot do your job
- Using someone else’s code comes with licensing issues
  - Possibly applies to GitHub Copilot as well

How to form a question

When to Ask Questions

- In-Class
  - Just raise your hand or use Piazza Live Q&A
- Piazza
  - Ask anytime, get responses from course staff, classmates
- Office Hours
  - All of the TAs have been where you've been!
- Recitations
  - Small group sessions

How to ask questions

- Instead of searching Google, try to form a question on Piazza
  - Simply trying to write out the question can help you understand the problem better.
  - You will get targeted help, in the context of **this** course.

- Come to Office Hours prepared with some question
  - Sometimes just thinking about what to ask can help you solve your problem on your own.
  - Course staff are not there to review your work to figure out what your question is.

Check if an answer already exists

- Most logistics questions are answered by the syllabus!
- Someone may have already asked your question on Piazza.
- Better to get multiple upvotes on one Live Q&A question.

Form your question carefully

- Explain the context of your question.
  - What are you trying to accomplish?
  - For code: what do you expect to happen?
- Explain the background you're coming in with.
  - What have you tried already?
  - For code: what is actually happening when you try it?
- Use complete sentences and avoid abbreviations.
- Read your post before you hit submit.
  - You may find that simply writing the question answers it.

Assignments

AI Quiz

- Will be posted before Aug 31 @ 1:00 AM
  - Posted on Autolab
  - Notifications via Piazza

- Should take < 10 minutes

- Due Wednesday, **September 7 @ 5 PM**

Scala Project

- Will be posted before Wednesday, Aug 31 @ 11 PM
  - Posted on Course Website
  - Notifications via Piazza

- Submit your GitHub username due by **Tuesday, September 6**

- Scala Hello World due Wednesday, **September 14**
  - Load an Open Dataset
  - Transform & Analyze the Data

Questions?