Course Syllabus for CS3358 (Data Structures and Algorithms) by Lee S. Koh

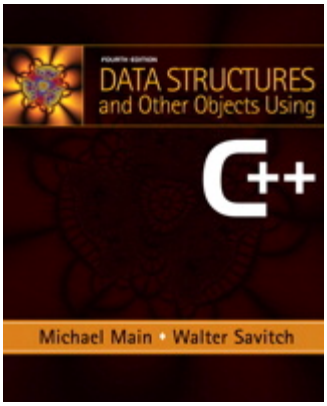
■ **Goal and Learning Objectives:**

- Most computer-based solutions to nontrivial problems involve the *manipulation of collections of data*. To effectively and efficiently perform such manipulations, appropriate *storage and access techniques* are essential. In other words, the data collections must be suitably represented and organized (stored in structures of some kind), and suitable algorithms must be devised to operate on the stored data. Some such *data structures* and *algorithms* have proven, over the years, to be so natural and powerful that they have come to be regarded as classic mechanisms. An appreciation of the variety and uses of these mechanisms is now considered to be fundamental to the understanding and training of computer-based problem solving and software development. This course will introduce you to these mechanisms, in the context of firming up your grasp on the prevailing problem solving approaches. The goal is to enhance your computer-based problem solving and software development skills beyond what you have acquired in CS 1428 (a prerequisite to CS 2308) and CS 2308 (a prerequisite to this course).
- Learning objectives:
  - Understanding Abstract Data Types: motivations and basic concepts.
  - Understanding of the behavior of basic data structures (lists, stacks, queues, trees (binary trees and tree traversals, height-balanced trees), graphs, hash tables).
  - Ability to analyze a problem and determine the appropriate data structure for the problem.
  - Understand the importance of data modeling and data structures in advanced programming.
  - Understand and analyze elementary algorithms: sorting, searching and hashing.
  - Ability to analyze the impact of data structures technique on the performance of algorithms (time and space complexity)/programs.
  - Deep understanding of recursion and its applications.
  - Data structure implementation issues. Understanding of dynamic versus array implementations of data structures, factors involved in deciding on an implementation technique.
  - Practice in writing modular programs using the data structures that have been studied.
  - Understanding the mechanics of code design, organization, and the development environment.
  - Understanding data structure implementation in C++ using header files and implementation files.

■ **Prerequisites:**

- C or better in CS 2308 (Foundations of Computer Science II) or equivalent.
- C or better in MATH 2358 (Discrete Mathematics I).

■ **Textbook:**



TITLE:
Data Structures and Other Objects Using C++
AUTHORS:
Michael Main and **Walter Savitch**
EDITION / PUBLISHER / YEAR / ISBN:
4th / Addison Wesley / 2011 / 0-13-212948-5

- Be sure to check out the **Textbook Style to Avoid** (http://cs.txstate.edu/~lk04/3358/TextbookStyleToAvoid.html).
  The previous edition (**3rd / Addison Wesley / 2005 / 0-321-19716-X**) will serve just as well.

■ **Platform and Programming Environment:**

GNU C++ in *Linux command-line environment* (as supported by the computer science department).

■ **Course Plan:**

Lectures/Discussions (main bullets) *and* *Descriptions (sub-bullets)* *(contents guide meant to be adaptively applied, not rigidly followed)*	*Readings/References* (C,S -> Chap.,Sec. of text) (LN -> Lecture Note) *(besides any in-class handouts)*
• Introduction.	
▸ Class website. ▸ Highlights -> administrivia, success tips, *etc*.	
• Big picture: CS - problem solving - data structures and algorithms - data types - abstraction.	S1.1, S1.3 LN305 thru LN307 LN308s01, LN308s02
• Object-based data type development using C++ -> review/augment by example.	C2, C4 LN301 thru LN304 LN304s00 thru LN304s02
▸ Data type notions/concepts, approaches and techniques/methodologies: - data type defined -> representable set of *values* and applicable set of *operations*. - data type *how-to* aspects -> consumers *use* and suppliers *build*, respectively: - <u>what</u> operations supported -> *interface*. - <u>how</u> values represented and <u>how</u> operations implemented -> *implementation*. - one trendsetting observation on *interface* and *implementation*: - *interface* -> less prone to change than *implementation*. - good to separate *interface* from *implementation* -> *information hiding*. - reduces disturbance on client programs when implementation changes. - another trendsetting observation on *interface* and *implementation*: - relationship between *interface* and *implementation* -> one-to-many. - good to abstract away *implementation* -> *abstract data type* (ADT). - simplifies description/classification/evaluation/... of data type (structure). - *contract*-oriented design technique -> mimicking everyday client-supplier contract: - *precondition*s } what to *expect* (as *benefit*) or what to *guarantee* (as *obligation*). - *postcondition*s } (which is which -> depends on standpoint, client or supplier) - *invariant* -> what to *maintain* (to uphold *rules* that *reveal/constrain/...* the *properties/relationships/...* of an object of the data type). ▸ C++ **`class`** what and how: - simpler **`class`** features and *indigenous* data types. - more involved **`class`** features and *exogenous* data types.	
• Set, bag (multiset) and list (sequence) ADTs.	C3 LN308
▸ Interface specifications: the *what*. ▸ Implementations using fixed-sized and dynamic arrays (non-template): the *how*. ▸ Variant: *sorted list* (*sorted sequence*) ADT.	

- Generic programming. — S6.1 thru S6.3, LN311

  - Toward developing more generally-applicable software components.
  - C++ function and class templates, and S(tandard) T(emplate) L(ibrary).
  - Implementation of bag and list ADTs using fixed-sized and dynamic arrays (templated).

- Basic algorithm analysis. — S1.2, Appendix B, LN309, LN310

  - Space-time trade-offs and other practical considerations.
  - *Best*, *average* and *worst* case scenarios.
  - *Big-O* characterization.

- Linked lists. — C5, LN312, LN313, LN322

  - Motivation:
    - basic searching and sorting:
      - practical significance of searching and sorting.
      - *linear/sequential/serial search* and *binary search*.
      - *selection*, *bubble* and *insertion* sorting algorithms.
    - insert/delete and grow/shrink anomalies associated with (sorted) arrays.
  - What's gained and lost -> array vs linked list.
  - Pointer-based implementation of linked lists.
  - Implementation of ADTs using linked lists.
  - Variants -> doubly-linked list, circular linked list, *etc*.

- Stack and queue ADTs. — C7, C8, LN314 thru LN316

  - Restricted forms of list ADT with widely useful operational characteristics:
    - *last-in-first-out* (LIFO) -> order reversal.
    - *first-in-first-out* (FIFO) -> order echoing.
  - Select applications:
    - managing invocation flow of control -> *system*/*call*/*runtime stack*.
    - transforming/evaluating expressions.
    - breadth-first (level) traversal of non-linear structures.
  - Implementation using array and linked list.
  - Variant: *priority queue* ADT.

- Recursion and recursive thinking -> divide-and-conquer + "results of division" are identical and smaller versions of "what gets divided". — S9.1, LN317

  - Power of recursion.
  - Criteria for application.
  - Advantages and disadvantages.
  - Success tips for designing/implementing recursive algorithms.
  - Tail recursion and indirect recursion.

- Trees. — C5, S11.1; LN318 thru LN321

  - Introduction:
    - common *nonlinear* data structure.
      - motivation for binary search tree.
    - recursive definition and other properties.
      - tree vis-a-vis graph and linked list.
    - general tree -> binary tree -> binary search tree and heap:
      - from general to more and more specialized.
    - some tree terminologies.
  - Representation:
    - array implementation (complete binary tree).
    - pointer-based implementation.
  - Binary tree traversals:
    - breadth-first (level).
    - depth-first.
  - Binary search tree:
    - storage rule.
    - insert, search, remove.
    - balancedness.
    - underlying data structure for ADTs -> bag as example.
  - Heap:
    - storage and shape rules.
    - insert and reheap up, remove and reheap down.
    - underlying data structure for *priority queue* ADT.

- More advanced sorting algorithms. — S13.2, S13.3; LN323

  - Heap sort.
  - Merge sort.
  - Quick sort.

- Hashing. — S12.2 thru S12.4; LN324, LN324a

  - Introduction:
    - another storage/access technique -> for *key*ed (IDed) data only.
      - comparison with "sequential/serial" and "ordered/binary".
    - perfect hashing.
      - 100% temporal consideration (constant-time search), 0% spatial consideration.
    - hashing in general.
      - compromise between temporal and spatial efficiencies.
    - some terminology -> hash table, hash function, hash value, collision, *etc*.
  - Some intuitive perspectives:
    - essence of approach:
      - *table*-lookup -> much like *dictionary* lookup.
      - key *map*ped to index (into hash table).
      - table/dictionary/map ADT.
    - hash function -> fast, deterministic, "randomized".
    - placement and retrieval -> must follow same mapping and collision resolution.

- Design factors/issues -> collision-reduction challenges:
  - table size -> load factor and rehash.
  - bucket size.
  - hash function.
  - collision resolution.
    - closed hashing (open addressing).
      - linear probing and primary clustering.
      - quadratic probing and secondary clustering.
      - double hashing.
  - open hashing -> chained hashing (separate chaining).
- Search, insert and delete algorithms.

- Other advanced topics (time permitting) -> height-balanced trees, graph, *etc*. S11.3, C15
  (Some are best covered at other appropriate junctures, not necessarily toward the end.) LN320s01a-c, LN325, LN326-8

■ **Attendance and Quizzes:**

- Attendance is ***compulsory***. Short quizzes may be given at the ***beginning*** of classes, and there will be no prior warning - you are expected to be ready to take a quiz on any day the class meets. You will not be given additional time if you are late for class so please be on-time for classes. Attendance will be taken unless a quiz is given, in which case the quiz papers will be used to record attendance.
  - I typically come to class slightly ahead of time and take the attendance of those who are already present. I will make a final pass at taking the attendance just before the class officially begins. If you joined the class late (after I have made the final pass), you will be considered absent (since your presence is not recorded); you *may* be able to change that ***only if you come forward and tell me about it immediately after class*** - my decision on whether to grant you the change from absence to presence is final. (Note that it would not be fair to grant you the change if, for no excusable reasons, you attended only a relatively small ending portion of a class meeting).
- ***A less than 70% (but not less than 50%) overall attendance will result in a "loss of one letter grade".***
  - For example, if your overall attendance is in the range [50%, 70%), you will get a C even though your combined score is in the B range.
- ***A less than 50% overall attendance will result in a grade no better than D (no matter how well you fared in the other components).***
  - In other words, you will get a D if you have a passing combined score, otherwise you will get a grade worse off than D.
- ***Missing the first class meeting is especially bad*** since pivotal "must-knows" (class website, class calendar, how class is administered, grading criteria, success tips, . . .) are made known and "*missing class(es) is the bane of successful course completion*" is underscored.

■ **Assignments:**

- 8 required assignments (typically):
- ***A less than 50% average score for required assignments will result in a penalty of "loss of one letter grade" (e.g., if your overall score is in the B range, you will get a C instead).***
- An assignment is due ***at the start of class*** on the due date.
  - Assignment papers turned in after the *final pass at taking the attendance* has been made will be considered late.
  - I reserve the right to grant due-date/due-time extensions or relaxations where appropriate.
- In general, you will earn no credit if you turn in your work late.
  - I reserve the right to grant partial credits (such as a 20% penalty per day of lateness) where appropriate.

■ **Exams:**

- 3 required exams:
  - Exam 1 (1hr 20min)

- Exam 2 (1hr 20min)
- Final (2hr 30min) - comprehensive (cumulative), with emphasis on concepts that have not been tested.

- All exams are of the *in-class* and *written* type, and *closed book* and *closed note*.

■ **Grading Criteria:**

- Exams: 70% (20% Exam 1, 20% Exam 2, 30% Final).
- Assignments: 25%.
- Class attendance and participation (including any quizzes): 5%.

■ **Withdrawal Policy:**

- Registrar's Office guidance: http://www.registrar.txstate.edu/registration/dropping-or-withdrawing.html.
- After the automatic W period, you are expected to check with me prior to dropping the class to see if you will receive a grade of W or F.

■ **Make-Up Exams, Tests or Quizzes:**

- Make-up tests and/or exams will only be given under ***unexpected and truly severe*** situations, which ***must be supported by some official document***. There will be no make-up quizzes.

■ **Academic Honesty:**

- Unless indicated otherwise, all work submitted is to be your individual work. Violations will be dealt with according to university policies - see Student Handbook pages 46-47 or look under http://www.dos.txstate.edu/handbook/rules.html. Programs turned in may be subject to review through Moss (http://theory.stanford.edu/~aiken/moss/).

■ **Course Related Material, Announcements, *etc.*:**

- These will be posted on the class homepage at http://www.cs.txstate.edu/~lk04. ***Students are responsible for visiting the homepage often to keep abreast of the latest postings.***

■ **Guidelines, Success Tips, and Other Pertinent Issues:**

- These will be mentioned in class and/or posted on the class homepage. ***Students are responsible for noting these*** while attending class (one reason why class attendance is compulsory) and by visiting the class homepage often. I typically spend some time on the ***first day of class*** highlighting these – if you should miss them, be sure to get the information from a classmate that takes good note.
- Except where specifically granted, ***all mobile computers and phones must be turned off*** when attending classes.

■ **Collective (Mass), Outside-of-Classroom, Instructor-to-Students Communication:**

- Electronically via students' *Texas State email accounts* (using a mailing list created from students' *Texas State NetIDs*).

■ **Special Needs:**

- Students with special needs (as documented by the Office of Disability Services) should have the instructor informed at the beginning of the semester/session.
