Class days, times, building, and room number:

For class, tutorial, and laboratory schedules, see the Summer Schedule of Classes.

CIS 22C Data Abstraction and Structures 4 1/2 Unit(s)

(Formerly Computer Information Systems 71C.)

Prerequisite: Computer Information Systems 22B or equivalent.

Advisory: Mathematics 212 or equivalent.

Four hours' lecture, one and one-half hours' laboratory (66 hours total per quarter)

Instructor

Dr. Kamran Eftekhari
mailto:eftekharikamran@fhda.edu
Office Hours: TH 5:00-6:00

Course description

Algorithms and data structures emphasizes the following topics: data structures, abstract data types, recursive algorithms, algorithm analysis, sorting and searching, and problem-solving strategies. Labs alternate weeks.

If you wish, you can read through a four-page course description.

Course objectives

Introduce the student to the concept of data structures through abstract data structures including lists, sorted lists, stacks, queues, deques, sets/maps, directed acyclic graphs, and graphs; and implementations including the use of linked lists, arrays, binary search trees, *M*-way search trees, hash tables, complete trees, and adjacency matrices and lists. Introduce the student to algorithms design including greedy, divide-and-conquer, random and backtracking algorithms and dynamic programming; and specific algorithms including, for example, resizing arrays, balancing search trees, shortest path, and spanning trees.

Outcome-based learning objectives

By the end of this course, the student will:

- Understand numerous examples of relationships between data;
- Understand the purpose and mathematical background of algorithm analysis and be able to apply this to determine the run time and memory usage of algorithms;

- Understand the abstract data types of stacks, queues and deques;
- Understand the variety of ways that linearly and weakly ordered data can be stored, accessed, and manipulated;
- Understand the characteristics and optimal behavior of hash tables for access and retrieval;
- Understand various sorting algorithms and the run-time analysis required to determine their efficiencies;
- Understand various graph algorithms;
- Understand numerous algorithm design techniques including greedy, divide-and-conquer, dynamic programming, randomized algorithms, and backtracking; and
- Understand concepts such as Turing equivalence, decision problems, the question of P = NP, NP completeness and the halting problem.

Suggested texts and readings

Mark Allen Weiss, *Data Structures and Algorithm Analysis in C++*, 4th Ed., Addison Wesley, 2012.

Robert Sedgewick, Bundle of Algorithms in C++, Parts 1-5: Fundamentals, Data Structures, Sorting, Searching, and Graph Algorithms (3rd Edition) 3 Ed. Pearson, 2001.

Cormen, Leiserson, Rivest, and Stein (CLRS), *Introduction to Algorithms*, 3nd Ed., MIT Press, 2009.

For a free online reference to C++, consider Bruce Eckel's Thinking in C++ 2nd Ed., Volumes 1 and 2. You will find a link to a download site. Of course, if you like the book, you always have the option of purchasing it. Other online resources are at http://www.cplusplus.com:

- A Tutorial,
- A Brief Description,
- A Brief History of C++,
- An FAQ, and
- A Collection of Links.

General overview of the topics

- Review of Mathematics and C++
- Asymptotic and Algorithm Analysis
  - Properties of data
  - Asymptotic Analysis
  - Algorithm Analysis
- Abstract Lists and Implementations
  - Linked lists and arrays
  - Stacks

   - Queues
   - Deques
- Abstract Sorted Lists and Implementations
   - General trees, binary (including binary and complete trees), *N*-ary trees, and tree traversals
   - Abstract Sorted Lists
   - Binary search trees
   - Balanced search trees
   - AVL trees
   - B-Trees
- Abstract Priority Queues
   - Heaps
- Abstract Sets/Maps
   - Chained Hash Tables
   - Linear Probing
   - Double Hashing
- Sorting Algorithms
   - Insertion and bubble sort
   - Heap, merge, and quick sort
   - Bucket and radix sort
- Graph and Direct Acyclic Graph Algorithms
   - Topological sort
   - Minimum spanning trees
   - Shortest path
- Algorithm Design
   - Greedy algorithms
   - Divide-and-conquer algorithms
   - Dynamic programming
   - Randomized algorithms
   - Backtracking algorithms
   - NP Completeness, Turing machines, and the halting problem
- Example of an advanced data structure

Evaluation structure

Your mark will be based on five equally-weighted projects; one mid-term examination; and one final examination. Let the variables P, M, and F represent your marks on the projects, mid-term examination and final examination, respectively (all out of 100). Let $E = (M + 2F)/3$ be your *examination* grade; then your grade G is determined by the following equation:

G =	$\min(E, P)$	$E \leq 50$ or $P \leq 50$
	$75 - E/2 - 5*P/4 + E*P/40$	$50 \leq E \leq 60$ and $E \leq P$
	$225 - 15*E/4 - 7*P/2 + 3*E*P/40$	$50 \leq P \leq 60$ and $P \leq E$

3*E/4 + P/4 P ≥ 60 and E ≥ 60

This is a function which is continuous in both variables E and P for values between 0 and 100. The consequences of this formula are as follows:

- You must pass both the projects and the examinations separately in order to pass the course.
- You must achieve a grade greater or equal to 60 in both the projects and the examinations for the normal marking scheme to count (25% projects, 25% mid-term examination, and 50% final examination),
- Otherwise, your mark is linearly interpolated so as to make the above function continuous as specified above.

Your grade based on your marks E and P is shown graphically in Figure 1.

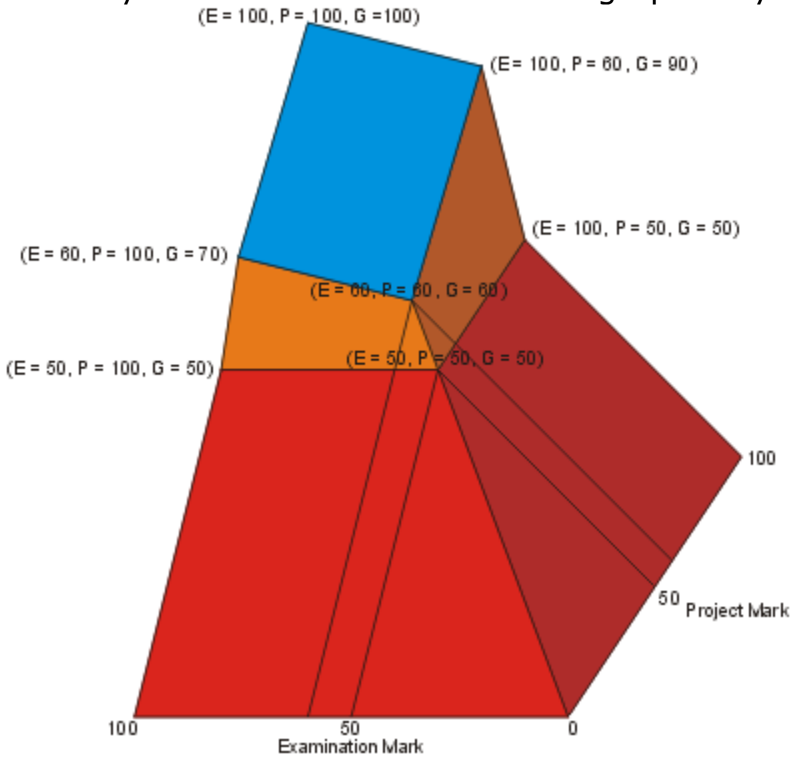

Figure 1. Final grade based on the examination and project grades.

A student who misses the mid-term examination (due to either a severe illness, a death in his or her immediate family, or other extreme conditions) and provides appropriate documentation (*e.g.*, a Verification of Illness Form for an illness, a death certificate, *etc.*) will have the value E = F when calculating the above formula.

The weight of the mid-term examination will not be changed under any other circumstances.

A student who passes the course will receive up to a 3% bonus for properly commenting his or her projects.

The instructor reserves the right convert one or more questions on either the mid-term or final examinations to bonus questions after the examinations are graded.

A student who misses the final examination (due to either a severe illness, a death in his or her immediate family, or other extreme conditions) and provides appropriate documentation (*e.g.*, a Verification of Illness Form for an illness, a death certificate, *etc*.) will write the final examination with the next offering of the course.

Expectations of the course

The student will learn to recognize relationships between objects and understand the means of choosing the appropriate data structures to store the objects so as to minimize the required memory and time required for the desired operations of insertion, access, search, modification, or removal.
The student will understand the basic design of algorithms including greedy, divide-and-conquer, randomized, and backtracking algorithms and dynamic programming and be able to apply these design strategies to solve programming problems.

Acceptable rules for group work

Students may not work together on the projects. Students may not share their source code or fragments thereof with any other student or individual—other than the course instructor, the lab instructor, or the teaching assistants—in any form, including, but not limited to, viewing, electronically sharing, reading, or printing.
Any assistance from an individual other than the course instructor, lab instructor, or teaching assistants must be documented including contact information.
A student who has submitted his or her project through CATALYST may assist other students with debugging; however, this help must be documented and the student providing the assistance may not modify the source code of the individual being help.

Late and missed submissions

All projects are due on dates specified. The drop box will remain open until 06:00 (6:00 A.M.) the following morning, at which point, the project will receive a maximum grade of 80%. Late submissions beyond this point will not be accepted and this will result in a grade of 0. The weight of a project cannot be moved to the final examination for any reason.

Project grading

Project grades will be awarded no later than one week after the due date. P

MOSS (Measure Of Software Similarity):

plagiarism detection

Plagiarism detection software (MOSS) will be used to screen projects in this course. This is being done to verify that use of all original source code is written by the student and not copied from the projects from other students (both from the current and previous terms). Students will be given an option if they do not want to have their projects screened by MOSS. A student may inform the instructor that he or she wishes to opt out of MOSS during the first week of the term. In this case, the instructor and the laboratory instructor will be comparing the code visually to submissions from both previous and current students.
MOSS is based on the paper Winnowing: Local Algorithms for Document Fingerprinting by Saul Schleimer, Daniel Wilkerson, and Alex Aiken.

Academic integrity

In order to maintain a culture of academic integrity, members of the De Anza College community are expected to promote honesty, trust, fairness, respect and responsibility.

Grievance

A student who believes that a decision affecting some aspect of his/her college life has been unfair or unreasonable may have grounds for initiating a grievance.

Discipline

A student is expected to know what constitutes academic integrity to avoid committing academic offenses and to take responsibility for his/her actions. A student who is unsure whether an action constitutes an offense, or who needs help in learning how to avoid offenses (e.g., plagiarism, cheating) or about "rules" for group work/collaboration should seek guidance from the course professor, academic advisor, or the undergraduate associate dean. For information on categories of offenses and types of penalties, students should refer to college policy, Student Discipline. For typical penalties check Guidelines for the Assessment of Penalties.
Plagiarism-detection software will be used on any submitted projects.

Appeals

A decision made or penalty imposed under College Policy , Student Petitions and Grievances (other than a petition) or Policy, Student Discipline may be appealed if there is a ground. A student who believes he/she has a ground for an appeal should refer to related Policy, Student Appeals.

Note for students with disabilities

The Office for Persons with Disabilities (OPD), collaborates with all academic departments to arrange appropriate accommodations for students with disabilities without compromising the academic integrity of the curriculum. If you require academic accommodations to lessen the impact of your disability, please register with the OPD at the beginning of each academic term.

Plagiarism detection software will be used to screen assignments in this course. This is being done to verify that use of all material and sources in assignments is documented. In the first lecture of the term, details will be provided about the arrangements for the use of plagiarism detection software in this course.

Course Projects

There are five projects given out during the Academic Quarter for this course. With the exception of the last project, the projects are due at 22:00 (10:00 P.M.) of the Tuesday immediately following the week with the corresponding laboratory. The last project is due at midnight of the last day of classes. Projects are submitted through CATALYST. For the first four projects, when the drop box closes, a late box will open that will be available for two hours until midnight. Late projects will be penalized by -20 % on the project grade. There are no late submissions for Project 5, as university policy forbids any due dates past the last day of classes.

Project	**Due Date**	**Weighting**	**Comments**
1	An implementation of linked lists in C++. Due at 22:00 on the Monday of Week 2.		Linked Lists
2	Due at 22:00 on the Monday of Week 3.		Stacks, Queues, and Deques
3	Due at 22:00 on the Monday of Week 4.		Trees
4	Due at 22:00 on the Monday of Week 5.		Hash Tables
5	Due at 24:00 on the last day of classes.		Graphs

Each project will be given by a list of requirements and sample test code. The sample test code must be such that passing the sample test code will guarantee a grade of 50 % on the project. That is, it will test that member functions are named correctly and that the basic functionality is present. Students are expected to do their own testing, though testing tools and some minimal tests will be provided.

Regrading

If you receive a grade less than 80 in Projects 1, 2, 3, and 4, you may meet with the instructor to determine what is wrong with your project. Assuming you have made significant progress toward a complete submission, if you can determine yourself the problem and come up with a solution, the maximum grade you can get is 80. If you cannot determine the cause of the errors, you may still meet with the instructor and together you will determine the short-comings of your code. In this case, your maximum grade will vary between 60 and 70, again, depending on your input and your awareness of the issues at hand. If your original submission did not even attempt to implement some of the required functionality, you can still achieve a grade of 50 if you have done a significant amount of work yourself and can explain to the instructor what is happening.

What You May Do

You may:

- Make multiple submissions—we will use the last,
- Make a late submission without making a submission on the original due date,
- Declare and define other classes and other member functions not specified in the project description; however, if these appear in other files, those files must be included with your submission,
- Reuse classes from previous projects; however, you must include those files with your submission,
- If you reuse previous projects, you are, of course, allowed to make corrections to those projects,
- You may even, though you must be very cautious, overload the given member functions or add additional parameters to those member functions so long as those extra parameters have default values and the functions as specified in the project may be called with no issues or ambiguities, and

You may use the following standard library functions (and no others) in Projects 1 through 4:

Library Header File	Functions and Classes
`#include <iostream>`	All functions and classes.
`#include <cassert>`	`assert(` *`predicate`* `);`
`#include <algorithm>`	`std::min( T, T );` `std::max( T, T );` `std::swap( T&, T& );`
`#include <cmath>`	All functions.
`#include <cfloat>`	All constants.
`#include <climits>`	All constants.

See Project 5 for restrictions (if any) for that project alone. If you would like another function added to this list, you must first make a request to the course instructor. The use of any other standard library function or class will result in an automatic 0 on that project.

Purpose

The purpose of the projects is to help you learn the course material and to help you begin to implement your own personal library of tools. Many of the subsequent projects will rely on previous ones (for example, you may be asked to specifically use your linked list classes to implement more complex data structures).

Program Documentation and Style

The following programming style is required for all projects.
Your name and ID must appear at the top of all files which you have created or modified.
Write clear and understandable code. Improve the clarity of your code by using vertical and horizontal spacing, meaningful variable names, proper indentation and comments.
Precede each function with comments indicating:

- What it does
- What each parameter is used for
- Assumptions that it makes
- How it handles errors

Exactly one project will be marked for comments and style. You will not be told which project was marked, nor when the marking has taken place.

Interface

You are allowed to add whatever private or public class functions you feel are necessary.

Standard Libraries

You are not allowed to use any standard libraries other than, where indicated.