EE 538: Computing Principles for Electrical Engineers
Units: 2
Term—Day—Time:
Lecture: 2 hours/week
Discussion: 1.5 hours/week

Location: **TBD**

Instructor: **Arash Saifhashemi**
Office: EEB 504B
Office Hours: *TBD*
Contact Info: saifhash@usc.edu

Teaching Assistant:
Office: TBD
Office Hours: TBD
Contact Info:

For Summer 2022:

If you are taking an internship or are not in Los Angeles and want to take the course online, sign up in the DEN Section.

All students (even non-DEN on-campus students) should be able to sign up in the DEN Section for the Summer. Use https://courses.uscden.net/d2l/login or contact DEN at dentsc@usc.edu or 213-740-9356 for help.

Course Description

This course provides a survey of topics required for advanced computer programming. It is targeted to graduate electrical engineering students with some prepatory (but perhaps informal) background programming experience. It teaches knowledge and develops practices required to understand and implement software problem solving. The course consists of two main parts: (1) foundations of software engineering, and (2) algorithms and data structures. The course includes the necessary theoretical basis to analyze algorithms, data structures, and mathematical methods. The course presents abstractions of methods and algorithms applicable in most robust computer languages. It emphasizes learning through implementing and all assignments and projects include major programming components that develop correct execution over suggestive pseudo-code. Assignments will extend your understanding by requiring concrete implementations with the C++ programming language. These components place emphasis on modern software engineering methodologies such as source control and testing using the C++ programming language.

Learning Objectives

A student that successfully completes this course will:

- Understand code to execution including tokenization, compiling, and linking.
- Apply step-wise debugging to identify causes of software defects.
- Utilize modern programming practices such as STL objects and object oriented (OO) programming including inheritance, overloading, templates, and polymorphism.
- Possess knowledge to identify and explain standard algorithms (sort, search, recursion) and data structures (tree, hash, lists).
- Develop skills to analyze novel code for its performance in both time and space complexity.
- Be comfortable deciding between multiple solutions given optimization or cost criteria.
- Apply test-driven software design and understand its role in minimizing errors and regression.
- Understand the connection between running software and underlying hardware including basics of threading, interrupts, memory management, caching, and device access.

Prerequisite(s): None

Co-Requisite(s): None

Concurrent Enrollment: None

Recommended Preparation: Standard undergraduate ECE education, including: (1) software proficiency in an object-orient language at the junior level or above (e.g., EE455x or EE541), and (2), mathematical proficiency and familiarity with proof structures – EE 141L, EE 364, MATH 225, or equivalent.

Technological Proficiency and Hardware/Software Required

You need access to a full stack for C++ development. You may consider installing a linux virtual machine to ensure maximum interoperability and access to any tools. The instructor and teaching assistants will give guidance during the first weeks of class.

Course Notes

Discussion section is ***not*** optional. The homework assignments are discussed primarily during the discussion section. Teaching assistants will cover and demonstrate tools during the discussion. Discussion sessions couple with the lecture and may also cover important additional material.

Required Readings and Supplementary Materials

1. Introductions to Algorithms, 3rd edition (required)
   Thomas Cormen, Charles, et. al. (available at the campus store)
2. The C++ Programming Language, 4th Edition (recommended)
   Bjarne Stroustrup (available at the campus store)
3. Code Complete: A Practical Handbook of Software Construction, 2nd Edition (recommended)
   Steve McConnell (available at the campus store)

Note: The texts are secondary to in-class lecture material and homework sets.

Description of Exams

All exams are cumulative and test both theoretical and applied aspects of the course. Exams include multiple-choice and/or short answer questions to demonstrate progress toward the learning objectives. They may also include free-response or open-ended questions to demonstrate comprehensive mastery. Students are expected to write correct code (abstract pseudo-code and C++). You may also be asked to read algorithms and determine expected behavior of novel computer code. Exam grading primarily follows correct reasoning but may include deductions for major syntax errors, algorithmic inefficiency, or poor implementation.

You may use a single 8.5"x11" reference sheet (front and back OK). You may not use any additional resources. You are expected to bring a non-graphing scientific calculator. You must show how you arrived at your answers to receive full credit.

Description and Assessment of Assignments

All projects and assignments must be submitted electronically either through source code management (e.g. Github) or for auto-grading. There is no "paper-copy"submission required or allowed. Review requirements for each assignment before submission.

No submission should be "headless" and should include a README file to describe any methods, testing, and validation.

Grading Breakdown

Assessment Tool (assignments)	% of Grade
Homework	40%
Project	20%
Midterm Exam	20%
Final Exam	20%

Assignment Submission Policy

Grading will be handled by both automatic and manual means. Code submissions may be reviewed and applied a series of test cases to determine function.

Course Schedule: A Weekly Breakdown

	Topics/Daily Activities	Readings/Preparation	Deliverables
Week 1	Intro: motivation and goals Methods of proof (Induction, proof by contradiction) Introduction to unit tests and version control	Lecture slides	Software and environment setup SCM and tooling
Week 2	Runtime complexity and Big-O notation C++ introduction: basics,	Lecture slides [1]: Ch. 2, 3 [2]: Pt. 1, Sec. 6, 9, 12, 13	HW1 assigned
Week 3	C++ functions, pointers, and references C++ STL (string, vector)	Lecture slides [1]: Ch. 10 [2]: Ch. 14, 18-21, 27, Pt. 4	
Week 4	Data structures: stack, linked list, queue, tree C++ classes, member variables and methods Other STL darta structures, set, map, algorithm	Lecture slides [1]: Ch. 10 [2]: Ch. 14, 18-21, 27, Pt. 4	HW2 assigned
Week 5	Data structures II: heap, tree Binary search and binary search trees Recusion: divide and conquer	[1]: Ch. 11 [1]: Ch. 12	HW3 assigned Project Discussion
Week 6	Dynamic Programming		HW4 assigned Midterm
Week 7	Introduction to graphs Graph representations Graph search and traversal (DFS, BFS, topological sort)		Project Discussion

Week 8	Graph search and traversal (DFS, BFS, topological sort) Shortest distance algorithms	Lecture slides [1]: Ch. 22	Project Phase 1
Week 9	Backtracking	Lecture Slides	Project Phase 2
Week 10	Sorting algorithms	[1]: Ch. 6 [1]: Ch. 7, 8	Project Phase 3
Week 11	Review and reflection	Lecture slides	Final Project Presentation
Week 12	Final exam		

Projects

Teams of two students (teams of one or three with instructor approval) must design and develop a software solution to a self-identified "mock" industry or research problem. The instructor will guide teams with difficulty identifying a suitable topic. Groups may implement solutions to problems with prior solutions provided their efforts demonstrate mastery of the course material. Teams are also encouraged to devise solutions to novel problems of particular interest to their backgrounds, interest, or research. Groups may substantially abstract problems from original context to fit within the project timeline and simplify both constraints and scale. All projects must obtain the instructor's written approval. Teams will prepare and present their *approved* project and show how it applies course material, concepts, and best-practices. Attendance *and participation* during the project presentation session(s) are mandatory.

Requirements

Project topics must include sufficient mathematical and algorithmic complexity and either include or extend substantive material from the course. Teams should treat the project as a platform to demonstrate mastery of design specification, algorithmic analysis, testing, debugging, and result validation. All projects must use the C++ language as the primary computer language unless approved explicitly in writing by the instructor. But teams may use or integrate additional languages for tooling and support.

Example projects

1. ***Document database engine***: Develop an interface for writing and retrieving documents from a managed storage engine. The system should not wrap another database engine but should expose a protocol for authentication, communication, and error handling. It may also explore disaster or corruption recovery.

2. ***Semantic translator***: Build a system convert an input string or stream from one defined semantic/language to another. The mapping should not be a trivial one-to-one and should require the retention of state or combination of non-contiguous information.

3. ***Digitial signal processor***: Design a complete software package to load and manipulate digital signal data (e.g. audio, image, video). It should be a complete user experience and provide facility beyond a single-purpose tool. It should include state information to provide feedback based on a sequence of user input such as Undo, Redo, and real-time updating.

Grading and Milestones

Topic proposal	week 10	10%
Phase 1 – Design, components, classes, and tests	week 13	15%
Phase 2 – Integration and deployment	week 15	20%
Demo and presentation	finals	20%
Project report and video		35%

Deliverables and demo

1. Written project report: the project report should summarize the topic, provide relevant background (theoretical or applied), timeline and contributions, and document challenges and extensions. It should provide discussion sufficient that an uninformed expert could understand the logic, algorithmic decisions, and implementations. Teams should provide quantifiable metrics to justify engineering tradeoffs.
2. Presentation: Approximately 10 minute (depends on class size) presentation to describe to the class their topic problem and their solution. It should provide only what is necessary to understand the "what" and "why" and include minimal theoretical background.
3. Video: 3-4 minute video that describes the problem, your design, and implementation. You may choose to upload this to a video sharing site such as YouTube but that is not required. All team members must participate equally.
4. Source code: submitted to instructor by providing link to pull from github.

Statement on Academic Conduct and Support Systems

Academic Conduct:

Plagiarism – presenting someone else's ideas as your own, either verbatim or recast in your own words – is a serious academic offense with serious consequences. Please familiarize yourself with the discussion of plagiarism in SCampus in Part B, Section 11, "Behavior Violating University Standards" policy.usc.edu/scampus-part-b. Other forms of academic dishonesty are equally unacceptable. See additional information in SCampus and university policies on scientific misconduct, policy.usc.edu/scientific-misconduct.

Support Systems:

Counseling and Mental Health - (213) 740-9355 – 24/7 on call
studenthealth.usc.edu/counseling
Free and confidential mental health treatment for students, including short-term psychotherapy, group counseling, stress fitness workshops, and crisis intervention.

National Suicide Prevention Lifeline - 1 (800) 273-8255 – 24/7 on call
suicidepreventionlifeline.org
Free and confidential emotional support to people in suicidal crisis or emotional distress 24 hours a day, 7 days a week.

Relationship and Sexual Violence Prevention Services (RSVP) - (213) 740-9355(WELL), press "0" after hours – 24/7 on call
studenthealth.usc.edu/sexual-assault
Free and confidential therapy services, workshops, and training for situations related to gender-based harm.

Office of Equity and Diversity (OED) - (213) 740-5086 | Title IX – (213) 821-8298
equity.usc.edu, titleix.usc.edu
Information about how to get help or help someone affected by harassment or discrimination, rights of protected classes, reporting options, and additional resources for students, faculty, staff, visitors, and applicants.

Reporting Incidents of Bias or Harassment - (213) 740-5086 or (213) 821-8298
usc-advocate.symplicity.com/care_report
Avenue to report incidents of bias, hate crimes, and microaggressions to the Office of Equity and Diversity |Title IX for appropriate investigation, supportive measures, and response.

The Office of Disability Services and Programs - (213) 740-0776
dsp.usc.edu
Support and accommodations for students with disabilities. Services include assistance in providing readers/notetakers/interpreters, special accommodations for test taking needs, assistance with architectural barriers, assistive technology, and support for individual needs.

USC Campus Support and Intervention - (213) 821-4710
campussupport.usc.edu
Assists students and families in resolving complex personal, financial, and academic issues adversely affecting their success as a student.

Diversity at USC - (213) 740-2101
diversity.usc.edu

Information on events, programs and training, the Provost's Diversity and Inclusion Council, Diversity Liaisons for each academic school, chronology, participation, and various resources for students.

USC Emergency - UPC: (213) 740-4321, HSC: (323) 442-1000 – 24/7 on call
dps.usc.edu, emergency.usc.edu
Emergency assistance and avenue to report a crime. Latest updates regarding safety, including ways in which instruction will be continued if an officially declared emergency makes travel to campus infeasible.

USC Department of Public Safety - UPC: (213) 740-6000, HSC: (323) 442-120 – 24/7 on call
dps.usc.edu
Non-emergency assistance or information.