CSCE155N Introduction to Computer Science I (MATLAB)

Class Syllabus

Fall 2016

Instructor

Leen-Kiat Soh, Professor

Office: Room 122E, Avery Hall  Tel: 402-472-6738  E-mail: lksoh@cse.unl.edu

Skype: profsoh

Homepage: http://cse.unl.edu/~lksoh/Classes/CSCE155N_Fall16/

(class materials are not on Blackboard except for the lab materials)

Office Hours: 12:30-1:30 PM TR and Open Door Policy

Teaching Assistants

Graduate Teaching Assistants: Ankitha Vejandla (E-mail: avejandl@cse.unl.edu) (Lab), Jun Wu (E-mail: junwu@cse.unl.edu) (Lab), Vishnu Sivadasan (E-mail: vsivadas@cse.unl.edu) (Lab)

Office Hours: Ankitha Vejandla: MW 9:30 AM – 10:30 AM
Vishnu Sivadasan: R 1:00 PM – 2:00 PM; F 12:00 noon – 1:00 PM
Jun Wu: T 3:30 – 5:30 PM

Undergraduate Teaching Assistants: Issac Logsdon (ilogsdon@cse.unl.edu), Quan Tran (E-mail: qtran@cse.unl.edu), Hao Yang (E-mail: hyang@cse.unl.edu)

Office Hours: Hao Yang: R 5:00 – 7:00 PM
Quan Tran: W 5:00 – 7:00 PM

Room: Student Resource Center, Avery Hall 12

Lectures & Labs

CSCE 155N-Section 250 COMP SCI I: ENG & SC LEC TR 02:00PM – 03:15PM AVH 106
CSCE-155N-Section 251 COMP SCI I: ENG & SC LAB M 12:30PM – 01:45PM AVH 20
CSCE-155N-Section 252 COMP SCI I: ENG & SC LAB M 02:00PM – 03:15PM AVH 20
CSCE-155N-Section 253 COMP SCI I: ENG & SC LAB M 03:30PM – 04:45PM AVH 20
CSCE-155N-Section 254 COMP SCI I: ENG & SC LAB M 05:00PM – 06:15PM AVH 20
CSCE-155N-Section 255 COMP SCI I: ENG & SC LAB M 06:30PM – 07:45PM AVH 20

Catalog Listing

Recommended for students interested in numerical and graphical applications in engineering and science, such as applied physics, working with time-sequence data, and matrix applications.
Introduction to problem solving with computers. Topics include problem solving methods, software development principles, computer programming, and computing in society.

Course Objectives

1. Mastery of the fundamentals of programming in a high-level language, including data types and rudimentary data structures, control flow, repetition, selection, input/output, and procedures and functions.
2. Familiarity with problem solving methods, including problem analysis, requirements and specifications, design, decomposition and step-wise refinement, and algorithm development (including recursion).

3. Familiarity with software development principles and practices, including data and operation abstraction, encapsulation, modularity, reuse, prototyping, iterative development, exception handling, documentation, coding conventions, and testing.

4. Exposure to computing topics, including algorithms for searching and other problems, graphical user interfaces, event-driven programming, and database access.

5. Exposure to the history of computing.

### Topics Covered

These topics are not purely sequential nor fully separate. Earlier topics continue to be reinforced and amplified in the context of other topics.

1. Problem-solving methods, including problem analysis, requirements and specifications, design (including top-down, bottom-up, object-oriented, and case-based design), decomposition and step-wise refinement, algorithm development.

2. Software development principles and practices, including use of an integrated development environment (IDE), tracing and debugging, data and operation abstraction, encapsulation, modularity, reuse, object-oriented programming, prototyping, iterative development, exception handling, documentation, coding conventions, and testing.

3. Data constructs, including data types, constants, identifiers, variables, assignment, operators, expressions, strings, input/output, arrays, abstract data types, and object properties.

4. Control structures, including selection, repetition, and exception handling.

5. Modular elements, including procedures, functions, variable scope, parameters, parameter passing, and object methods.

6. Algorithms, including searching (sequential and binary), recursion, case studies.

7. Graphical user interface (GUI) programming, GUI APIs, GUI event-driven programming.

8. Programming database access.


### Prerequisites

1. Appropriate score on the CSE Placement Exam.

2. Mastery of basic mathematical problem solving as demonstrated by satisfactory completion of mathematics through college algebra, trigonometry, and pre-calculus. (MATH 103 or equivalent).

3. Familiarity with the use of computers and software applications.

### Text Book


### Programming Projects (aka Homework Assignments) (Group)
There will be about five Programming Projects to help you understand and master the basic concepts and train you on problem solving—to ultimately be proficient in developing software programs. The MATLAB programming language must be used for all the assignments. The assignments are due at class time on the indicated dates. You will be required to hand in a hardcopy of your code at class time on the due date. You are expected to write well-documented, modular code. **Programs which do not compile or which do not run will not receive any credit.** Programs that compile and run will be graded as follows: 45% **Program Correctness**, 15% **Software Design**, 10% **Programming Style**, 15% **Testing**, and 15% **Documentation**.

### Laboratory Assignments (Randomized Pair)

The laboratories are designed to supplement the lectures and provide hands-on experiences on topics that need additional attention. It is a significant part of the course and you are strongly encouraged to fully take advantage of this opportunity. You are required to read the lab handouts and the reading assignments before coming to the lab. There are about 12 laboratories altogether.

### Creative Thinking Exercises (Group)

This course is participating in a project called “Integrating Computational & Creative Thinking” or IC2Think. The intention of this project is to enhance your learning of Computer Science by fostering your ability to think creatively about problems. Specifically, your grade for this component will be based on up to four Computational Creativity Exercises assigned throughout the semester. Each exercise will be assigned over 2-3 weeks. These exercises will involve the use of Blackboard’s Wiki system where you will document your activities and findings and participate in thread-based discussions. Your team will be graded and evaluated based on how well you perform the assignment. No late submissions will be accepted and no late passes may be used for the IC2Think exercises. As part of these exercises, there will also be short surveys and knowledge test administered during the semester.

### Preprocessing Questions (Individual)

Throughout the semester, there will be questions that require an open-ended, online response. These questions are asked to introduce you to a topic that we will be covering in class and to gauge the familiarity that the class has with the topic.

### Clicker Questions (Individual)

While covering each topic, there will be a number of clicker questions in class. These multiple-choice questions are designed to measure your understanding of a topic that is being covered and challenge you to think about how the concept may be used in the future.

### Examinations (Individual)

We will have two mid-term exams and a final exam. The tentative dates for the exams are listed in the schedule. There will be no makeup tests.

### Final Grade

Your final course grade is based on the following: (1) programming projects (38%), (2) creative thinking exercises (5%), (3) laboratory assignments (10%), (4) preprocessing questions (2%), (5) clicker questions (5%), (6) midterms (25%), and (7) final exam (20%). The total % is 105%,
which will be normalized to 100% before assigning the final grades. Final grades in this class will be assigned based on the following scale.

<table>
<thead>
<tr>
<th>Grade</th>
<th>Minimum Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>A+</td>
<td>$\geq 97$</td>
</tr>
<tr>
<td>A</td>
<td>$\geq 93$</td>
</tr>
<tr>
<td>A-</td>
<td>$\geq 90$</td>
</tr>
<tr>
<td>B+</td>
<td>$\geq 87$</td>
</tr>
<tr>
<td>B</td>
<td>$\geq 83$</td>
</tr>
<tr>
<td>B-</td>
<td>$\geq 80$</td>
</tr>
<tr>
<td>C+</td>
<td>$\geq 77$</td>
</tr>
<tr>
<td>C</td>
<td>$\geq 73$</td>
</tr>
<tr>
<td>C-</td>
<td>$\geq 70$</td>
</tr>
<tr>
<td>D+</td>
<td>$\geq 67$</td>
</tr>
<tr>
<td>D</td>
<td>$\geq 63$</td>
</tr>
<tr>
<td>D-</td>
<td>$\geq 60$</td>
</tr>
<tr>
<td>F</td>
<td>$&lt; 60$</td>
</tr>
</tbody>
</table>

**Disabilities**

Students with disabilities are encouraged to contact Christy Horn for a confidential discussion of their individual needs for academic accommodation. It is the policy of the University of Nebraska-Lincoln to provide flexible and individualized accommodation to students with documented disabilities that may affect their ability to fully participate in course activities or to meet course requirements. To receive accommodation services, students must be registered with the Services for Students with Disabilities (SSD) office, 132 Canfield Administration, 472-3787 voice or TTY.

**Academic Misconduct**

Violations of academic integrity will result in automatic failure of the class and referral to the proper university officials. The work a student submits in a class is expected to be the student’s own work and must be work completed for that particular class and assignment. Students wishing to build on an old project or work on a similar topic in two classes must discuss this with both professors. Academic dishonesty includes: handling in another’s work or part of another’s work as your own, turning in one of your old papers for a current class, or turning in the same or similar paper for two different classes. Using notes or other study aids or otherwise obtaining another’s answers for an examination also represents a breach of academic integrity. Those who share their code and those who copy other’s code will be penalized in the same way; both parties will be considered to have plagiarized. Sanctions are applied whether the violation was intentional or not.

Academic dishonesty of any kind will be dealt with in a manner consistent with the CSE Department's Policy on Academic Integrity (http://cse.unl.edu/undergrads/academic_integrity.php). You are expected to know and abide by this policy.

To help avoid these problems, please start assignments early and seek help when you need it.

**PLAGIARISM OF ANY KIND IN THIS COURSE WILL RESULT IN A GRADE OF F.**

**Relationship to ACE Learning Outcome**

This course is approved for ACE Student Learning Outcome 3, according to the following criteria:

*SLO3: Use mathematical, computational, statistical, or formal reasoning (including reasoning based on principles of logic) to solve problems, draw inferences, and determine reasonableness.*

We will provide the following opportunities for learning the above ACE Learning Outcome. This course not only teaches students about how to design an algorithmic solution to **solve a problem**, but also teaches students about how to engineer the design into a working piece of program. Furthermore, the engineering process of implementing a program involves significant debugging, testing, and refining code. These activities teaches and reinforces **inferencing**: a student has to be able to draw inference when diagnosing why a program crashes
or does not compile or generate incorrect output; after making fixes, a student will have to re-evaluate the design to see if the outcome meets his or her expectation, and further **draw inferences** on how to proceed. Finally, an algorithm is fundamentally a logical sequence of steps that, given a set of input, generates definitively a set of output. The correct derivation of the output provides the decidability of the algorithm, which in turn **determines reasonableness**.

**We will assess your achievement of the outcome through three primary tools**, among others: exams, programming homework assignments, and structured laboratory assignments. Note that the artifacts of the programming homework assignments inherently embed the results of **problem solving**, **inferencing**, and **reasonableness reasoning**. This is because in order to produce a working program that compiles, runs, and computes the correct output, a student must devise an algorithmic solution and then implement it. For the structured laboratory assignments, worksheets—where students submit their findings from solving the problems given—are graded, and pre- and post-tests are given to students to test how they have learned.

**Reinforced Skills**

- **Critical Thinking.** Critical thinking is key in the development of algorithms and during the debugging process of implementing a program. The course provides numerous opportunities for critical thinking in lectures, programming assignments, and laboratories. The laboratories and assignments are problem-based and students are tasked to apply critical thinking to solve problems.

- **Problem Solving.** The development of algorithms and the implementation of programs are inherently problem solving. The course provides numerous opportunities for problem solving in programming assignment and laboratories. The laboratories and assignments are problem-based and students are tasked to solve problems.

**Other Matters**

The CSE Department has an anonymous suggestion box ([http://cse.unl.edu/department/suggestion.php](http://cse.unl.edu/department/suggestion.php)) that you may use to voice your concerns about any problems in the course or department if you do not wish to be identified.

It is CSE Department policy that all students in CSE courses are expected to regularly check their email so they do not miss important announcements.

Please make use of the Student Resource Center in Avery 13A: [http://cse.unl.edu/src](http://cse.unl.edu/src)

**Longer Version on Relationship with ACE Learning Outcome**

The course presents many opportunities to learn computational and formal reasoning methodologies and skills to solve problems, draw inferences, and determine reasonableness.

Specifically, the lectures, together with the programming assignments and the weekly laboratory sessions, teach students both algorithms and the implementation of those algorithms to solve problems. That is, the course not only teaches students about how to design algorithmic solutions to solve problems, but how to engineer designs into working programs. This engineering process involves significant debugging, testing, and refining code. These activities teach and reinforce inferencing: a student must draw inferences when diagnosing why a program crashes, does not compile, or generates incorrect output; after making fixes, a student must re-evaluate the design to see if the outcome meets expectations, and further draw inferences on how to proceed.

**Finally, an algorithm is fundamentally a logical sequence of steps that, given a set of input, generates output.** Specifications for the output determine reasonableness. Through algorithmic development, with top-down design, problem analysis and specification, step-wise refinement, and
modularization, the students, when programming, are trained to determine the reasonableness of their solution. For example, students are trained to examine how their algorithms handle exceptions (which could terminate an algorithm prematurely if handled incorrectly), deal with boundary conditions (to prevent their programs from crashing), and prevent infinite loops (which could prevent reaching an outcome).

**The course has approximately 45 hours of lectures each designed to explore concepts and paradigms that are central to the field of computer science.** Students will master control flow, repetition, selection, input and output processes, and procedure and function design and invocation. Students will learn fundamental problem solving paradigms, including abstraction, encapsulation, exception handling, and event-driven programming. Through lectures, laboratory sessions, and programming assignments, students learn about problem analysis and specification, top-down design, algorithm development (including recursion), step-wise refinement, and modularization. Students will also be exposed to various algorithms, such as for searching and sorting.

**The course has approximately 14 hours of laboratory sessions, each designed to train students to apply what they learn in the lectures to actual implementation and analysis of algorithms and software programs.** Laboratory sessions require students to solve problems, to debug or revise programs, to analyze programs. These activities reinforce the students on problem solving, drawing inferences from their design and implementation, and determining reasonableness of a solution.

**The course includes several programming assignments designed to help students learn about designing algorithmic solutions and the practice of implementing solutions as correct software programs,** involving key steps such as program analysis, solution identification and evaluation, solution-to-algorithm mapping, initial feasibility analysis, coding, debugging, testing, and refinement. These steps provide ample opportunities for students to apply their computational and formal reasoning skills to solve problems, draw inferences, and determine reasonableness.