CMSI 284-01, -02

Computer Systems Organization/ Systems Programming

http://myweb.lmu.edu/dondi/spring2016/cmsi284

Spring 2016—Doolan 219 (01, 02)
TR 9:55–11:10am (01), 2:40–3:55pm (02), 3 semester hours
Office Hours: TWR 1:30–2:30pm, TR 4–5:30pm, or by appointment

John David N. Dionisio, Ph.D.
email: *dondi@lmu.edu*
Doolan 106; (310) 338-5782

Objectives and Outcomes

This course explores how computing systems operate at a level that is very close to the actual machines on which they run—i.e., programming with the minimum possible translation between your code and what the computer can access and manipulate directly. Long after the course concludes, my hope is that you will be able to:

1. **Encode, decode, and manipulate bit representations of different types of data.**
2. **Write programs in C, assembly language, or both at the same time.**
3. **Understand and perform computing tasks close to the system level.**

In addition to the course-specific content, you are also expected to:

4. **Follow disciplinary best practices throughout the course.**

Prerequisites/Prior Background

Programming proficiency in at least one high-level language; a prior course in data structures, such as LMU's CMSI 281. A familiarity with current hardware components and specifications is also beneficial but not absolutely necessary.

Materials and Texts

This course does not have a preassigned textbook, with materials consisting solely of assorted handouts, articles, and sample code to be distributed throughout the semester. Key reference works include but are not limited to:

- Randal Bryant and David O'Halloran, *Computer Systems: A Programmer's Perspective*, 2nd Edition, Prentice-Hall, 2011
- Brian Kernighan and Dennis Ritchie, *The C Programming Language*, 2nd Edition, Prentice-Hall, 1988
- Intel Corporation, *The IA-32 and Intel 64 Architectures System Developer's Manual*, 2016
- David Goldberg, "What Every Computer Scientist Should Know About Floating-Point Arithmetic," *ACM Computing Surveys* **23**(1), March 1991

Course Work and Grading

This course uses standards-based grading: your proficiency in each course objective is directly evaluated according to the outcomes shown on page 4 of this syllabus. Proficiency is measured according to the following key:

+	Advanced proficiency
|	Appropriate proficiency
/	Approaching appropriate proficiency
–	Needs practice and support
O	No work submitted

Your submitted work is used to evaluate these outcomes (see below). Letter grades are then assigned as follows:

	+	**|**	**/**	**–**	**O**
A	**many**		*none*	*none*	*none*
B		**many**	**few**	*none*	*none*
C			**some**	**few**	*none*
D				**some**	**few**
F				**many**	**some**

A–, B+, B–, C+, and C– grades are assigned for outlier combinations between the above thresholds. Qualitative considerations (e.g., degree of difficulty, effort, class participation, time constraints, overall attitude) may improve proficiency measures. To resolve close calls, a quantitative calculation with 4, 3, 2, 1, and 0 standing in for **+**, **|**, **/**, **–**, and **O** respectively will be used. You will receive feedback and proficiency updates after every assignment.

Term Portfolio

Your accumulated assignments for the semester comprise the *term portfolio*—the final, definitive artifact that demonstrates the proficiencies you have reached for each course outcome. It is how you show whether you have, indeed, accomplished the objectives of this course.

The final proficiency for any given outcome is approximately the rounded statistical mean over the set of assigned proficiencies, using the numeric mapping given previously and modulated by qualitative considerations such as degree of difficulty, cumulative nature of the material, or demonstrated progress by the student. Incomplete portfolios are evaluated on a case-to-case basis.

Bit-Level Encoding, Decoding, and Manipulation

- Numbers
- Characters
- Machine instructions

Your work here affects your proficiencies for outcomes *1a–1c* and subsets of *4a–4f*.

Assembly Language and C Programming

- Selected programs and algorithms in C
- Selected programs and algorithms in Intel 64 assembly language
- Mixed-language programs using calling conventions to invoke assembly language code from C and vice versa

Your work here affects your proficiencies for outcomes *2a–2d*, *3a–3c*, and *4a–4f*.

Resubmitting Work for Re-evaluation: Once Within Two Weeks of Feedback for the First $n - 2$ Assignments

Standards-based grading focuses ideally on achieving proficiency, not accumulating scores. Thus, all but the last two assignments may be resubmitted for re-evaluation *once within two weeks of receiving feedback* on it. I have no way of automatically knowing when you're finished with something, so *please notify me by email* when a submission is ready.

You must still submit all assignments by their respective deadlines—late work detracts from outcome *4f*. An assignment's number is its due date in *mmdd* format.

Version Control

Version control is an indispensable part of today's computer science landscape in industry, the academe, and the open source community. We use version control heavily in this course: make sure that you get the hang of it.

Workload Expectations

In line with LMU's *Credit Hour Policy*, the workload expectation for this course is that for every one (1) hour of classroom instruction (50 scheduled minutes), you will complete at least two (2) hours of out-of-class work each week. This is a 3-unit course with 3 hours of instruction per week, so you are expected to complete $3 \times 2 = 6$ hours of weekly work outside of class.

Attendance

Attendance at all sessions is expected, but not absolutely required. If you must miss class, it is your responsibility to keep up with the course. The last day to add or drop a class without a grade of W is January 15. The withdrawal or credit/no-credit deadline is March 18.

Academic Honesty

Academic dishonesty will be treated as an extremely serious matter, with serious consequences that can range from receiving no credit to expulsion. It is never permissible to turn in work that has been copied from another student or copied from a source (including the Internet) without properly acknowledging the source. It is your responsibility to make sure that your work meets the standard of academic honesty set forth in the *LMU Honor Code and Process*.

Special Accommodations

Students with special needs who require reasonable modifications or special assistance in this course should promptly direct their request to the Disability Support Services (DSS) Office. Any student who currently has a documented disability (ADHD, autism spectrum, learning, physical, or psychiatric) needing academic accommodations should contact DSS (Daum 224, x84216) as early in the semester as possible. All requests and discussions will remain confidential. Please visit *http://www.lmu.edu/dss* for additional information.

Topics and Important Dates

Correlated outcomes are shown for each topic. Specifics may change as the course progresses. University dates (italicized) are less likely to change.

January	Overview; computer system organization; numeric and character encoding *(1a–1c)*
January 15	*Last day to add or drop a class without a grade of W*
February	A simple computer *(1c, 2c, 2d)*; C programming *(2a, 2b)*
February 29–March 4	*Spring break; no class*
March	Processors; the Intel 64 architecture; assembly language programming *(2c, 2d, 3a, 3b)*
March 18	*Last day to withdraw from classes or apply for Credit/No Credit grading*
March 23–25	*Easter break; no class*
March 31	*Cesar Chavez Day; no class*
April	System calls *(3c)*; executable files *(3a, 3b)*; mixed language programming *(2a–2d, 3a–3c)*

You can view my class calendar and office hour schedule in any iCalendar-savvy client. Its subscription link can be found on the course web site (it's too long to provide in writing).

Tentative Nature of the Syllabus

If necessary, this syllabus and its contents are subject to revision; students are responsible for any changes or modifications distributed in class or posted to the course web site.

Course Outcomes

1 Encode, decode, and manipulate bit representations of different types of data.

1a	*Numbers*	This is the class where you get to "see" the information we use in the same way that current computing machines truly represent them. Encoding, decoding, and manipulation will be performed both manually and in code that you write. You may check your work with tools available online but you may *not* use those tools to do the work for you.
1b	*Characters*	
1c	*Machine instructions*	

2 Write programs in C, assembly language, or both at the same time.

2a	*Operate on strings at the storage level.*	These outcomes pertain to specific features of the languages involved. General correctness, presentation, and other best practices that apply to *any* kind of programming are covered by learning objective 4.
2b	*Use pointers, arrays, and address references.*	
2c	*Perform computations on registers and main memory.*	
2d	*Perform machine-level comparisons and jumps.*	

3 Understand and perform computing tasks close to the system level.

3a	*Compile and link object code.*	These outcomes focus on computing activities that we frequently take for granted—but *someone* somewhere needs to know how to do this. The takeaway here is to demystify these activities and expose what's "under the hood."
3b	*Follow and use calling conventions.*	
3c	*Invoke system calls.*	

4 Follow disciplinary best practices throughout the course.

4a	*Write syntactically correct, functional code.*	Code has to compile. Code has to work. No errors, no bugs. Use unit tests as much as possible.
4b	*Use coding best practices, demonstrating principles such as DRY, proper separation of concerns, correct scoping of variables and functions, etc.*	This is the basis of good software design. It makes software easier to maintain, improve, and extend. Heed feedback well. *What you learn here will apply to future classes.*
4c	*Write code that is easily understood by programmers other than yourself.*	This outcome involves all aspects of code readability and clarity for human beings, including but not limited to spacing & indentation, proper naming, presenting code in a manner that is consistent with its structure, documentation & comments when appropriate, and adherence to conventions or standards.
4d	*Use available resources and documentation to find required information.*	The need to look things up never goes away. Remember also that the course instructor counts as an "available resource," so this outcome includes asking questions and using office hours.
4e	*Use version control effectively.*	In addition to simply using version control correctly, effective use also involves appropriate time management, commit frequency, and descriptive commit messages.
4f	*Meet all designated deadlines.*	

Sample Standards Achievement Report

Based on these proficiencies, the student is a qualitative call between an A– and a B+.

1	**Encode, decode, and manipulate bit representations of different types of data.**	
1a	*Numbers*	+
1b	*Characters*	+
1c	*Machine instructions*	+
2	**Write programs in C, assembly language, or both at the same time.**	
2a	*Operate on strings at the storage level.*	+
2b	*Use pointers, arrays, and address references.*	+
2c	*Perform computations on registers and main memory.*	|
2d	*Perform machine-level comparisons and jumps.*	|
3	**Understand and perform computing tasks close to the system level.**	
3a	*Compile and link object code.*	+
3b	*Follow and use calling conventions.*	+
3c	*Invoke system calls.*	+
4	**Follow disciplinary best practices throughout the course.**	
4a	*Write syntactically correct, functional code.*	+
4b	*Use coding best practices, demonstrating principles such as DRY, proper separation of concerns, correct scoping of variables and functions, etc.*	+
4c	*Write code that is easily understood by programmers other than yourself.*	+
4d	*Use available resources and documentation to find required information.*	+
4e	*Use version control effectively.*	|
4f	*Meet all designated deadlines.*	/

This student reached advanced proficiency in 12 out of the 16 outcomes. Appropriate proficiency was reached in 4 out of 17, but proficiency was not reached in one outcome (*4f*).

The student might have been in the running for an A had work been submitted on time. Instead, the student is a close call between an A– and B+. Qualitative factors such as class participation will determine the final grade.

The student should have avoided the / for a guaranteed A-level grade. More +'s would have ensured the A rather than A–. More |'s rather than +'s would have taken the student to a B-level grade; more /'s would have taken the student to a C-level grade.

Totals	
+	12
|	3
/	1
–	0
O	0