UNIVERSITY OF CALIFORNIA, SANTA CRUZ

BOARD OF STUDIES IN COMPUTER ENGINEERING

CMPE-13/L: COMPUTER SYSTEMS AND C PROGRAMMING

TIME: T-Th 11:00am - 12:40pm

CLASS: Nat. Sci. Annex 103

LAB: Soc. Sci. 1 135

Instructor:

MAXWELL LICHTENSTEIN

E-mail: mnlichte@ucsc.edu

Office: Engineering 2 (E2), 316

Hours: TBD

Phone: (303) 524-4634 (Office)

TA: Pavlo Vlastos

Introduction

Computer Systems and C Programming is a class intended to build the fundamentals of the C language and its applications. Originally written in 1978, C remains one of the most widely used programming languages, and in many ways it forms the foundation of modern computing. Most operating systems are written in C or C variants. Java and Python run on a virtual machines, which are usually written in C. Nearly all embedded devices are programmed with code written in C. Even if you never write another line of C code after this class, a firm understanding of its mechanisms will greatly improve your ability to understand the world of computers.

This course's formal prerequisites are CMPE 12 and 12L. In addition, you should be at least somewhat familiar with programming and computers. C is often a challenging language for beginners. Also, this is a programming class and you will be writing lots of code. Many students spend 15-20 hours a week outside of class playing with the code to get things to work.

In this class, we are going to approach C from two directions. About two-thirds of the coursework will focus on embedded C, using a 32-bit embedded microcontroller, the Microchip PIC32. About a third of the coursework will focus on C in a Unix environment, using the UCSC Unix timeshare.

While this class is nominally about learning the C language, this class is also where many students begin learning more general principles of effective coding: how to write modular,

testable code, how to write readable code, how to think about scope, how to manage a workflow, how to ask for help in technical classes, and how to read error messages.

Textbooks and other readings:

There is quite a bit of material to cover in this class, and you are expected to have read the assigned reading before coming to class. You will get out of this class what you put into it. Simply put, if you do not do the reading, you will not effectively learn the material, so take the time to read them.

The main textbook, *The C Programming Language*, (K&R) is one of the best books you will ever read on a technical subject. Like the language C itself, it is precise and to the point, and assumes you know what you are doing. There is quite a bit of subtle hints on how to code well in the book, and it should be read carefully. The Notes to Accompany K&R point out some of these subtleties and expand on some of the examples. Make sure to read both K&R and the accompanying Notes as assigned (you will most likely want to read them again after lecture).

Main textbook:

[K&R]: "The C Programming Language, 2nd Edition" by Kernighan and Ritchie, Prentice-Hall, 1988, ISBN-10: 0131103628. Available in the bookstore.

[Notes]: "Notes to accompany K&R," by Steve Summit available on the class website and at: http://www.eskimo.com/~scs/cclass/krnotes/top.html

Optional Textbooks (good references):

[SummitIntro]: "C Programming Notes: Introductory C Programming Class Notes," by Steve Summit available on the class website and on the web for free at: http://www.eskimo.com/~scs/cclass/notes/top.html

[SummitIntermediate]: "C Programming Notes: Intermediate C Programming Class Notes," by Steve Summit available on the web for free at: http://www.eskimo.com/~scs/cclass/int/top.html

[Wiki]: "C Programming: A comprehensive look at the C programming language and its features," Wikibook, available on the class website and also available online at: http://en.wikibooks.org/wiki/C_Programming

[C-Book]: "The C Book" by Mike Banahan, Declan Brady and Mark Doran, originally published by Addison Wesley in 1991. Available for free online as HTML and PDF versions at: http://publications.gbdirect.co.uk/c_book/

[Hard C]: "Learn C The Hard Way," online set of modules and practice programs that will take you through a fairly complete C knowledge. Available

free online at: http://c.learncodethehardway.org/book/ (videos and PDF available but not free).

[Zyante]: "Programming in C," interactive C book online by Frank Vahid and Smita Bakshi (not free, but can be gotten at a discount if you are interested) at: https://zybooks.zyante.com/#/zybook/7VU9pYQ5ue/tableofcontents

"The Cartoon Guide to Computer Science" by Larry Gonick, Barnes and Noble Books, 1983. Out of print, but it is available on the class website in three parts at: http://www.soe.ucsc.edu/classes/cmpe013/Spring11/Gonick/

Reading Schedule:

Each week, expect a quiz that covers that week's readings. Handouts can be found on CANVAS.

Week	Readings for the Week
1 Intro, History, Tools, C Runtime, Heap/Stack, Hello World, Toolchain, C basics, Comments, Variables, Primitives, Header files	[Handout] CMPE13_Syllabus, CMPE13_IntroToGit [Gonick] Parts 1-2 [K&R]: Chapter 1.1-1.2, 2.1, 4.5, 4.11.1
2 Expressions, If-statements, while loops, Functions, printf(), Scanf(), Arrays, Strings, Operators	[K&R]: Chapter 2.11, 3.1, 3.2, 3.5, 1.7, 4.1, 7.2, 7.4, 7.1, 1.6, 1.9, 5.7, 2.5-2.6
3 Operators (cont'd), loops (cont'd), Unit testing, Pass by value, Pass by reference, Scope, Data lifetimes	[K&R]: 2.8-2.12, 1.3, 3.5, 3.6, 1.8, 4.2, 4.4, 4.8, 4.9, 4.6, 4.7
4 Literals/Constants, Structs, typedef, Text input, String processing, Pointers	[K&R]: Chapter 1.4, 2.3, 6.1-6.4, 6.7, 7.7, 7.8.1, 5.1
5 Pointers (cont'd), Dynamic memory, Dynamic memory (cont'd), Pointers (cont'd), Enums, Interrupts,	[K&R]: Chapter 5.2-5.5, 5.6-5.9, 7.8.5, 2.3
6 Dynamic memory example, Implementing malloc() example, Hardware peripherals, Event-driven programming, Bit masking/flags	[Handout]: CKO Chapter 5.1-5.7 [K&R]: Chapter 4.6, 2.9, 6.9, 8.7
7 Bitfields, Preprocessor, Switch statements, Switch statements (cont'd), State machines	[Handout]: CKO Chapter 5.8-5.10, Bit Manipulation handout [K&R]: Chapter 3.4, 4.11
8 State machines (cont'd), Recursion, Binary Trees	[K&R]: Chapter 4.10, 7.8.5
9 Recursion (cont'd), Software engineering, Random number generation, Encryption, Checksumming, Communications protocols	[Handouts]: Iterative software design [K&R]: Chapter 7.5, 7.8.7
10 void pointers, Function pointers, Unions, variable argument lists, x86 programming, File I/O, File formats	[K&R]: Chapter 5.11-5.12, 6.8, 7.3, 7.5

Course Software and Hardware:

PIC Development Environment

We are going to be using a Microchip 32-Bit Processor, the PIC32, for most of the programming assignments in this class. Most of our development should be done within the MPLAB-X integrated development environment (IDE), which can be downloaded from the class website (see the Installing Software handout). We are using the Microchip XC-32 compiler for PIC32. We have put together a small kit that includes a PIC32, and IO_Shield with screen, LEDs, and buttons, and a PicKIT 3 programmer so that you will be able to develop on real hardware. We will also be using the built-in simulator within the MPLAB-X IDE.

You will check out the Uno32 kit from BELS (in the basement of JBE), which will charge you a lab fee. When you return your lab kit at the end of the quarter, if it's in working order, you will be refunded most of this fee. You cannot keep your lab kit and must return it at the end of the quarter (you can buy and assemble your own if you want one).

All of the programming tools are available for free, and can be downloaded from the links on the class website. You are strongly encouraged to install these on your own computers so that you can work outside of the lab sections. Note that while these tools are all supposedly cross-platform, we will only be supporting Windows installations.

The development software is installed on all Windows computers campus-wide, including the 2 24-hour labs on campus. Your own computer is not required for any of this coursework, and any toolchain you set up on your own labtop is your own responsibility (in other words, if your computer crashes, that is not grounds for a due date extension).

Other course software:

Some of the work in this course will be done using GNU-gcc on the UCSC Unix timeshare. The timeshare can be accessed from any campus computer, or any computer supporting an SSH client. You may use whatever workflow suits you for these labs, so feel free to choose your favorite editor or IDE.

This course uses Git version control software. For the duration of the quarter, you will have access to a course Git repo on the UCSC Gitlabs server. We expect you to do your coursework inside this repo. We also expect that you will commit frequently, and develop professional Git practices. All coding assignments will be submitted via this Git repo.

Please note that your Git activity serves as a record of your progress in this class. Committing frequently protects you against data loss, and also provides the best defense if you need to prove that your work is your own.

Grading

This course is based on a combination of the lab and class. The distinction is only on paper: You will receive the same grade in both courses. This is programming class, and you will be

graded largely on the programs you write. Note that we have a strict late policy on the labs, make sure you turn in what you have on time.

COURSE/LAB:	70%	Programming assignments
	30%	In-class quizzes (every week on reading)

A small curve may be added at the end of the course, in the form of a slight offset to bring the top student or two up to 100%.

Passing threshold:

In order to pass the class, you will need to demonstrate basic mastery of the subject matter as demonstrated by your labs and quizzes; no one may pass the class with less than 50% average on their lab scores. Note that it is possible to fail the course even with a greater than 50% average on lab scores.

Labs:

There will be a programming assignment every week, (start it as soon as you get it!). The lab grades will be assessed on the following criteria:

- Completeness – Did you turn in everything you were supposed to?
- Functionality – Does your code compile/run/produce correct results?
- Documentation – For each lab, you must write a README that serves as a sort of lab report.
- Readability – Is your code well-structured, organized, well-named, and styled according to course guidelines?

Labs are not all weighted identically, and typically labs later in the quarter are weighted more highly.

One lab assignment, Lab 9, is a paired assignment. In this assignment, both partners will submit the same work, and will receive the same grade.

Quizzes:

There will be a quiz in class covering the required reading material *at least once a week*. This quiz serves to assess both attendance and your progress in the reading material.

Each week, the quiz will be at the beginning of class on a randomly-chosen day (Tuesday or Thursday). No make-up quizzes will be given, nor will there be any opportunities to take the quiz later in that class. If you must miss a quiz, contact the instructor or TAs *before* the class to arrange taking the quiz at a different time. The only accepted excuse *after* a quiz is a visit to the emergency room.

Lab Sections (CE13/L)

Lab sections are held on Monday and Wednesday, 1-2pm (excepting June 25th and July 4th), in Social Sciences 1-135. Though we do not take attendance, we do expect you to attend lab section, and work on your lab there. We also expect that you will have **already** thoroughly read the lab, **and started work** on it, prior to section.

During the lab sections, the TA and tutor will answer questions about the lab, discuss course concepts related to the lab, and help you test and debug your assignments. Typically, a "help queue" is maintained on the whiteboard, and staff will help students in order.

Assignments are designed to take more time than your lab section to complete; the lab section is to give you guidance and help, it is expected that you will go off and program on your own both **before and after** sections. The total time given to complete the labs is sufficient if you start early; *starting during lab section is a recipe for failure*.

Open Lab Hours:

Over the summer, campus computer lab access is limited. Only Social Sciences 1-135 is open over the summer, and its hours are limited to:

M-Th, 9am-8pm

F, 9am-5pm

In addition, several other classes will be using this lab over the summer. See https://its.ucsc.edu/computer-labs/class-schedules/summer/socsci1-pc.html for a full schedule. If the instructors of other sections allow you to work in the lab during their times, please be respectful of their class environment. Also, July 4th is a holiday, and the lab will not be open on that day.

Submitting Assignments

Labs will be released each Monday, and are due on the following Monday at midnight (excepting Lab 10, which will be due on August 31st, the last day of the quarter).

Extensions will be granted only in the case that external circumstances prevent the whole class from accessing a crucial resource (a campus-wide power outage, for example). With appropriate planning, reading, and use of course resources, every lab can be done comfortably in a week, so labs will not be extended to accommodate poor planning.

Submitting a Lab:

Labs are submitted via your GitLab repository. First, commit and push your Git repository containing your work. Then, use the GitLab web interface to add a tag to the commit you wish to submit (Tags are git objects that simply point to a particular commit). To designate a commit as the commit you wish to have graded, append it with a tag of the form "LabX_submission_0".

This process is covered extensively in accompanying documentation, but here is a quick summary of the steps needed to do a submission:

- Commit your work (Git add, git commit)
- Push your work (Git push)
- Log in to the GitLab server, select “Tags”, click “New Tag”, and add a tag with the appropriate name.

Inside of the repo, there are 10 folders, one for each lab. Each lab contains a list of graded files. Exactly one copy of each graded file should be in the appropriate lab folder of the commit you tag for submission. That file can be in the lab folder, or within a .X project folder within that lab folder (so “myrepo/Lab3/README.txt” and “myrepo/Lab3/MatrixMath.X/README.txt” are both valid locations for the file README.txt).

Much of the grading in this class is automated, so you must be very careful with file names (and tag names, and function names). Capitalization matters. If you submit a file with the wrong name, your work is graded as though you had not submitted that file. If you tag a commit incorrectly, your work is graded as though you had not tagged that commit at all.

Resubmitting:

After creating a git tag to designate a commit as your submission, you may decide that you wish to modify your submission. In this case, simply add a new tag of the form “LabX_submission_Y”, where X is the number of the lab, and Y is any positive integer. The officially submitted lab is the lab with highest Y. You may add a new tag at any time, up to 72 hours after the lab (the latest possible time any student could submit that lab).

Your late hours will be deducted according to the *time of commit* of the tag with the highest Y. We do *not* consider the time that the tag was created when deducting late hours.

As an example, consider the following timeline:

- On Monday (the due date for lab 7), at 11:50pm, you make and push a commit on with the ID 733ff4. You fall asleep moments later. At this point, if you take no other action, you will receive a 0 on the assignment (since you have not tagged any commit for lab 7).
- Tuesday morning, you wake up, and realize you have not tagged a submission for lab 7. You give commit 733ff4 the tag “Lab7_submission_0”. At this point, if you take no other action, your assignment will be graded normally with commit ID 733ff4, and no late hours will be deducted (late hours are calculated using the *commit time*, not the tag creation time).
- Tuesday afternoon, you realize that there is a significant bug in your code, so you modify your code, and make and push a new commit with ID 3d4ca4, at 4:30 pm. You give this commit the tag “Lab7_submission_1”. At this point, if you take no other action, your assignment will be graded with commit ID 3d4ca4, and 18 late hours will be deducted (4:30pm is 16.5 hours after Wednesday at midnight, and 16.5 rounded up to the nearest 6 is 18).
- On Wednesday, you decide that the bug really wasn’t that serious, and you’d rather have your earlier commit graded. You add *another* tag to commit 733ff4 with the name

"Lab7_submission_2". At this point, if you take no other action, we will grade commit 733ff4, and no late hours will be deducted.

- On Friday, you change your mind again and decide that bug was serious, and you want to use commit 3d4ca4. At this point, it is more than 72 hours after the due date. You can try adding a new tag to commit 3d4ca4 with the name "Lab7_submission_3". If you are lucky, your grader has not pulled your repo yet, and your change will be accepted. Otherwise, you may use your git do-over.

Grace Period and Git Do-over

We understand that sometimes things in life occur that are beyond your control, and you will occasionally need extra time. Each student is granted 72 hours of "grace period" to be used throughout the quarter on your lab and homework assignments. It is consumed in 6 hour increments, and you will be able to check the "late hours remaining" assignment on CANVAS to see how many hours you have left.

In the case of paired assignments, the larger of the two grace periods is used.

If there is some extenuating circumstance that is not covered by the grace period, we need to be informed of this ASAP (not weeks after the fact).

Git is challenging to use, and it is common for students to accidentally make errors during the commit process. For this reason, we grant a one-time Git do-over. This is meant to be used for the following situations:

- You accidentally tagged the wrong commit.
- You forgot to tag a commit, or used the wrong name for your tag.
- You added a tag in your local repo, but did not push that tag.

The do-over is NOT meant for the following situations:

- You failed to include the appropriate files in your repo.
- You forgot to commit.
- You want to free late hours from a previous lab

Late hours will be deducted automatically based on your Google form. If you wish to use your Git do-over, please make a private post on Canvas with a title like "Lab 5 commit do-over". In this post, include your ucsc.edu email, the lab you wish to modify, and the commit ID that you would prefer we grade.

Late hours and the Git do-over are tracked via two assignments on canvas. These are not part of your grade, and exist only to help you and us track these quantites.

Once you have used up your late hours, all late assignments are graded as 0s.

Regrading:

Regrading labs is done only in the case of a clear error on the part of the grader. To request a regrade, make a private post on Piazza specifying your ucsc.edu email, the lab number, the

commit ID, and a detailed description of the grading error. Beware: Regrades are handled by me (Max L), and I am a significantly harsher grader than anyone on the grading staff.

Regrade requests *must* be made within 2 weeks of the grade's release (the first time at which you can view your grade on Canvas).

Communication and Getting Help

This class uses a Piazza forum for nearly all communication. Find it at piazza.com/ucsc/summer2018/cmpe13lsummer18/home. We expect you to check Piazza regularly, and to participate in discussions there. The forum is closely monitored by the TA and the instructor.

Please be sure to follow these guidelines for posting on Piazza:

- Make sure the title of your post is descriptive. "Strange error" is not a good post title. Why am I getting "Error: main.c: cannot find BOARD.h" is a much better post title. The better you make the title, the easier it will be for others to navigate to your post.
- If someone asks a question and you know the answer, you should answer! Teaching is a very effective method for improving your own mastery and retention.
- Before asking, check to see if a question has already been answered. We recommend reading every post on Piazza, but if you do not, then at least search for some key terms in your own question. If you see a duplicate question that someone else posted, please note that in a response.
- Don't publicly post code onto Piazza; if you need one of the teaching staff to look at your code, post it in a private message to the teaching staff.
- Communicate professionally. Your posts are anonymous to each other, but not to the teaching staff. You are training to be a professional, so treat each your peers (and the staff!) with dignity and respect.

Piazza is the preferred mode of communication in this course. If you have a need to contact the teaching staff, do so with a private post on Piazza. Do not email the TA.

You should email me (Max) directly only in personal, sensitive cases. If you do need to email me, use the mnlichte@ucsc.edu address, and be sure to use the string "CMPE13" in the subject line. Emails will be checked at least once per business day, so you can expect an answer by the end of the next business day.

Disability Resources

UC Santa Cruz is committed to creating an academic environment that supports its diverse student body. If you are a student with a disability who requires accommodations to achieve equal access in this course, please submit your Accommodation Authorization Letter from the Disability Resource Center (DRC) to me privately during my office hours or by appointment, preferably within the first two weeks of the quarter. At this time, I would also like us to discuss

ways we can ensure your full participation in the course. I encourage all students who may benefit from learning more about DRC services to contact DRC by phone at 831-459-2089, or by email at drc@ucsc.edu.

Title IX:

The university cherishes the free and open exchange of ideas and enlargement of knowledge. To maintain this freedom and openness requires objectivity, mutual trust, and confidence; it requires the absence of coercion, intimidation, or exploitation. The principal responsibility for maintaining these conditions must rest upon those members of the university community who exercise most authority and leadership: faculty, managers, and supervisors.

The university has therefore instituted a number of measures designed to protect its community from sex discrimination, sexual harassment, sexual violence, and other related prohibited conduct. Information about the Title IX Office, the online reporting link, applicable campus resources, reporting responsibilities, the UC Policy on Sexual Violence and Sexual Harassment and the UC Santa Cruz Procedures for Reporting and Responding to Reports of Sexual Violence and Sexual Harassment can be found at titleix.ucsc.edu.

The Title IX/Sexual Harassment Office is located at 105 Kerr Hall. In addition to the online reporting option, you can contact the Title IX Office by calling 831-459-2462.

University Deadlines:

The 10-week summer session has a drop deadline of Monday, July 9 and a withdraw deadline of Friday, July 27. Neither Summer Session nor instructors drop students for non-attendance or non-payment. Students must drop themselves. Dropping results in full tuition reversal/refund. Withdraw posts a W for the grade and full tuition is charged (no refund).

For all dates and deadlines, including 'change of grade option' (P/NP) and grades due, here is the summer academic calendar : https://summer.ucsc.edu/fundamentals/academic-calendar.html

Lecture Videos and Slides

The class lecture will be recorded and the slide decks recorded using the screen capture capability of the Surface computer. These will prove to be useful when reviewing the material for midterm and final, or merely to go back and go over material to refresh it in your mind. However, in order to keep attendance in the lectures, the videos and class slides will be uploaded with a one week delay (they will be on the class website, not on CANVAS).

Academic Honesty

Academic honesty is a requirement for the course. All assignments must be your own independent work; this includes both quizzes and programming assignments.

What is cheating? It is presenting work that is not yours as your own. You can—and are encouraged to—discuss and strategize with your colleagues on the material and labs, but **your work should be your own**. Copying is **NEVER** acceptable.

On the labs, **cheating is sharing code** unless explicitly told that it is permitted. If a student is caught cheating in either the class or the lab this will result in an immediate failure in the class and the lab. It will be reported to your college and your department. **DO NOT CHEAT**; it is not worth it.

Collaboration vs Plagiarism

Collaboration is a key aspect of education, and of professional engineering. As such, we want to encourage you to help each other understand the labs and the material. However, this is NOT license to copy others' work. Credit for collaboration should be explicitly noted in each assignment's README. Failure to give credit on collaboration is considered a form of cheating and will be dealt with accordingly.

Unfortunately, in most quarters, we find evidence of plagiarism in a significant fraction of the CMPE13 student body. In many instances, students are (or claim to be) unaware that what they are doing *is* plagiarism. To clarify, here are examples of acceptable collaboration:

- Discussing the labs, or lab requirements with peers.
- Planning, diagramming, or writing pseudocode for a lab with peers.
- Helping a peer understand an error or assist in debugging.

In the above cases, this collaboration must be described in the README that you submit.

Here are examples of unacceptable collaboration (ie, cheating):

- Reading someone else's code for inspiration before you start your own code.
- Reading someone else's code as you write your own.
- Checking with a friend every few minutes to be sure you've written similar things.
- Using more than a line or two from Stack Overflow or a similar online resource.
- Using code from a friend or online resource, but altering it slightly.
- Reading someone else's code to find a solution to a flaw in your own code.
- Typing out code that friend dictates to you as you type.

A good practice for collaborating while avoiding cheating is to use a whiteboard or a piece of paper to communicate ideas, and look at each other's monitors as little as possible.

Acknowledgements

I would like to acknowledge Gabriel Elkaim and Max Dunne, who have spent countless hours developing nearly every aspect of this course. Further thanks to Steve Summit, who taught an Introductory and Intermediate Programming class at the Experimental College at the University of Washington in Seattle, WA, and who has generously allowed us to use his notes on K&R and other supplementary materials. Microchip Corp. has generously provided slides and software

through their academic partner program, and some of the slides on file systems come from Henry Cheng at UC Davis. Finally, thanks to UCSC's IT department and Baskin Engineering Lab Support, without whom this course would not run.