CSC 411

Machine Organization

Prof. Noah M. Daniels

noah_daniels@uri.edu
http://www.cs.uri.edu/~ndaniels/
Tyler 250

Changes to course policies or schedule may occur in response to unforeseen circumstances. I will notify the class of any changes immediately.

Course Description: What's this class about and why am I taking it?
CSC 411 presents the hardware foundation on which software is built. *Machine organization* shows how *data* is represented at the machine level, while *assembly language* is how *code* is represented at the machine level.

We require all students to study computation at the machine level because:

- The construction of very high-level abstractions and programming languages on top of very low-level hardware is one of the great intellectual achievements of computer science, and every computer scientist should know about it.
- The data and code model of any particular machine, called its *instruction-set architecture*, represents a *contractual agreement* between hardware designers and systems programmers that allows a collection of programs to run on any hardware which conforms to the instruction-set architecture. The exploitation of instruction-set architecture to enable both hardware and software to be changed independently and reused with each other is one of the great engineering achievements of computer science, and every computer scientist should know about it. (Some of the fruits of this exploitation include operating systems that will run on any x86-compliant hardware; the ability to buy new, more advanced CPUs without having to replace your software; and the ability to create and use "virtualization environments" such as VMware, QEMU, Xen, and so on.)
- A programmer may understand a high-level program's outputs and interactions by understanding only the model of computation presented by the programming language. But to understand how a program *performs*, one needs a working knowledge of computation at the machine level. When something goes wrong in a software system, diagnosing the fault often requires the ability to break the abstraction barriers presented by programming languages and to inspect the state and behavior of the machine at its own level, not the level of programs' source code.

- Much software that is critical to the world's computing infrastructure is written in the languages C and C++, and to understand how this "systems software" works, one must understand machine-level computation.

When you master the material in CSC 411:

- You will be able to understand and improve the real performance of programs written in high-level languages.
- You will be able to diagnose and fix bugs that can't be explained without appealing to the underlying machine model. Many important bugs in C and C++ programs fall into this class.
- You will have started learning how to exploit the underlying hardware in ways that are not possible using high-level languages alone.
- You will be able to write moderate-sized, multi-module programs in the Rust programming language.

Prerequisite(s): C- or better in CSC212.

Texts:

- *Computer Systems: A Programmer's Perspective, 3rd Ed.*, Bryant and O'Hallaron
- *The C Programming Language*, Kernighan and Ritchie
- **https://doc.rust-lang.org/book/** (free)

Should you buy the books?Kernighan and Ritchie (often referred to as K&R) is a classic book that will serve you well, and can be found quite cheaply. Even though this is no longer primarily a C programming course, we will see some C, and you are likely to encounter it in your career. Bryant and O'Halloran is an expensive textbook, but provides excellent coverage of some technical matter, such as floating-point numbers and the memory hierarchy. You will definitely use the Rust book heavily, but it is free.
No exercises or problems will be assigned from any of the books (except possibly the free Rust book). But, having the books may save you time in a very time-intensive course.

Course Goals:

1. Appreciate that there is more to computer science than writing computer programs
2. Decompose problems into more-manageable subproblems
3. Develop abstractions to simplify problem solving.
4. Understand the computational costs of certain operations

Student Learning Outcomes:
Upon successful completion of this course of this course, students will be able to:

1. Solve computational problems using abstraction and modularity
2. Use and implement some elementary data structures
3. Write programs of moderate complexity in Rust
4. Write a simple design document explaining their approach to a programming problem
5. Describe how data are represented in memory in the C/C++/Rust model
6. Reason about the computational costs of certain operations

7. Perform simple operations in a Linux command-line environment

Grade Distribution:

Class Participation	10%
Lab Assignments	20%
Programming Assignments	70%

Letter Grade Distribution:

>= 93.00	A	80.00 - 82.99	B-	67.00 - 69.99	D+
90.00 - 92.99	A-	77.00 - 79.99	C+	63.00 - 66.99	D
87.00 - 89.99	B+	73.00 - 76.99	C	60.00 - 62.99	D-
83.00 - 86.99	B	70.00 - 72.99	C-	<= 59.99	F

Course Policies:

- **Attendance**
  - You are expected to attend class. I do not take attendance *per se*, but in-class exercises will not be announced ahead of time.
  - In-class exercises count towards the class participation component of your grade, so it is essential to attend class.
- **Grades**
  - Grades in the **C** range represent performance that **meets expectations**; Grades in the **B** range represent performance that is **substantially better** than the expectations; Grades in the **A** range represent work that is **excellent**.
  - **Assignments** will be submitted using the Gradescope online system. Grades will be maintained on Gradescope, but may not immediately reflect non-Gradescope sources such as in-class exercises.
  - **Class participation** is very specific: there will be frequent in-class exercises, done in groups of 3-5 students. Participation in these exercises is expected. If you participate in at least **75%** of these exercises, you will receive 100% of the Class Participation grade. Otherwise you will receive **zero** for Class Participation.
- **Cheating**
  - All pair programming assignments must be your own work and that of your partner's.
  - While you may discuss general solutions and algorithms with classmates other than your partner, it is **against the rules to:**
    - share code with other students (besides your partner)
    - look at any other student's code (besides your partner)
    - use code provided to you by anyone else (other than by the instructor)
    - use code that you find on the internet
  - If you use code that has been provided for you by the instructor, or that you have used in a prior assignment, include in your comments where the code came from, and describe how the code works. If you ever have a question about what is acceptable when working on a programming assignment, please contact your instructor.

  - **Any violation of these rules may result in a grade of 0 on the assignment. In addition, you may be reported to the Dean and the Office of Student Life.** See the University Manual for more information about the potential consequences of cheating `https://web.uri.edu/manual/chapter-8/chapter-8-2/`.

- **Assignments**
  - Assignments are pair-programming assignments. With permission of the instructor, you may work alone rather than with a partner. This is not recommended.
  - All assignments will be submitted and graded through Gradescope. More information on Gradescope will be included with Assignment 1.

- **Exams**
  - There will be no exams.

Students with Disabilities

Any student with a documented disability is welcome to contact me as early in the semester as possible so that we may arrange reasonable accommodations. As part of this process, please be in touch with Disability Services for Students Office at 302 Memorial Union, Phone 401-874-2098.

Academic Honesty Policy:

All submitted work must be your own. If you consult other sources (class readings, articles or books from the library, articles available through internet databases, or websites) these MUST be properly documented, or you will be charged with plagiarism and will receive an F for the paper. In some cases, this may result in a failure of the course as well. In addition, the charge of academic dishonesty will go on your record in the Office of Student Life. If you have any doubt about what constitutes plagiarism, visit the following websites: the URI Student Handbook, and Sections 8.27.10 – 8.27.21 of the University Manual (web.uri.edu/manual/).

Programming assignments will be done in pairs. For the purposes of pair programming assignments, "your own" means that the work is the product of you and your partner, together.

Generative AI tools like ChatGPT are very good at writing introductory level computing code. While these tools can be useful to programmers solving large problems, they can be both impediments and enhancements to learning. Therefore, the design and implementation of your programming assignments is expected to be the product of you (and your programming partner). You may use generative AI tools in a tutorial fashion where you ask general questions about coding and problem solving, just as you are encouraged to do a web search for compiler errors. However, you **may not** use generative tools to do your assignments. When doing work for CSC 411, follow these rules if you are using generative tools:

- You may consult generative tools on general questions
- You **may not** copy and paste your assignments or code into the tool
- You **may not** copy and paste any results from the tool into your assignments
- If you use an AI tool while working on an assignment, you **must** provide the prompt that you used and the result of the prompt as documentation in your assignment
- If you use programming constructs that are not taught in class (e.g. advanced lifetimes annotations or complex trait bounds), you **must** cite where you found them and why you have chosen to use them instead of the constructs taught in class.

Any violation of these rules may result in a grade of 0 on the assignment. In addition, you may be reported to the Dean and the Office of Student Life. See the University Manual for more information about the potential consequences of cheating. https://web.uri.edu/manual/chapter-8/chapter-8-2/.

Attendance

Students are expected to attend class and classroom activities. Occasionally, students may miss class activities due to illness, severe weather, or sanctioned University events. If ill, students should not attend class and should seek medical attention especially if they have a communicable disease such as influenza (flu). Students should not attend class when the University announces classes are cancelled due to severe weather. Also, it is the policy of the University of Rhode Island to accord students, on an individual basis, the opportunity to observe their traditional religious holidays. Students desiring to observe a holiday of special importance must inform each instructor and discuss options for missed classes or examinations. See Sections 8.51.11 – 8.51.14 of the University Manual for policy regarding make-up of missed class or examinations.

I do not specifically take attendance. However, 10% of your grade is determined by class participation, which includes participating in group exercises. If you do not participate in at least 75% of group exercises, you will receive a zero for class participation.

Pair Programming:

Professional engineering and computer science are collaborative endeavors. And yet, while you are a student, you earn an individual grade. To balance these competing concerns, we support

- Wide but shallow collaboration when discussing ideas and problems
- Deep but narrow collaboration when creating and debugging computer programs (in labs and on the final project)

Wide but shallow collaboration means that while you are striving to understand a problem and discover possible paths to its solution, you are encouraged to discuss the problem and your ideas with friends and colleagues—you will do much better in the course, and at URI, if you find people with whom you regularly discuss problems. When the time comes to write code, however, group discussions are no longer appropriate.

Deep but narrow collaboration means pair programming. In pair programming, you work with a partner under the following constraints:

- When work is being done on the program, both partners are present at the computer. One partner holds the keyboard; the other watches the screen. Both partners talk, and the keyboard should change hands occasionally.
- You submit a single program (or design) under both your names. That work gets one grade, which you both receive.

Regular homework assignments will be entirely solo; no "deep but narrow" collaboration is allowed. "Wide but shallow" collaboration is still allowed. This will be spelled out clearly in the assignment specification.

The following policies are essential:

- *Every source of assistance must be acknowledged in writing.* This rule applies to discussions with classmates or course staff as well as assistance you might find in the library or on the web. There is never a penalty for seeking help with a problem, but help must be acknowledged.

- If circumstances, such as scheduling difficulties, make it impossible for you to work as part of a pair, you may ask the course staff for permission to divide an assignment into parts and to do some parts as a member of a pair and other parts as an individual. Such parts must appear in different files, and *each file must be clearly identified as the work of an individual or the work of a pair.* Work done jointly by the pair should be submitted by both members of the pair. *Files containing joint work must be identical.* If you as an individual modify a file containing joint work, and you submit the modified file, that act will be considered a violation of academic integrity.

- It is *never acceptable* to divide an assignment into parts and have some parts done by one partner and other parts done by the other. Submitting work done by someone else as your own will be considered an *egregious violation of academic integrity.* Submitting individual work as the product of pair programming is also a violation of academic integrity.

- If your partner disappears in mid-project, the correct procedure is submit the work done in partnership at that point, even if it is incomplete or broken. You may then follow up with an additional submission of whatever you complete on your own.

- Unless you are working with another student as part of a programming pair , it is *not acceptable* to permit that student to see any part of your program, and it is *not acceptable* to permit yourself to see any part of that other student's program. In particular, you may not test or debug another student's code, nor may you have another student test or debug your code. (If you can't get code to work, consult a TA or the instructor.) Using another's code in any form or writing code for use by another will be considered a violation of academic integrity.

- *Never, ever* share any account password (or private key) with another individual (whether student, faculty, or staff). Sharing your password with another student will be considered a violation of academic integrity. If you wish to transfer code between accounts, sending a git bundle via scp or email is one good approach. You are encouraged to submit *general programming questions* to online forums such as Stackoverflow. Questions about particular homework problems must *never* be posted online—send mail to the course staff.

Finally, be aware that *pair programming is a privilege, not a right.* If you foul up and don't fix it, I may revoke your pair-programming privileges. Fouling up consists of any of the following unacceptable behavior:

- Repeatedly failing to keep appointments with your partner
- Lying to your partner about what you have done
- Completing parts of the assignment without your partner present
- Violating academic integrity
- Other similarly egregious offenses

If I find it necessary to revoke your pair-programming privileges and you believe I have done so unfairly, you may appeal to the department chair.

Tentative Course Outline:
Note: This is organized by **module**, not by week. Invariably, the schedule will change.

- **intro**

  Introduction: Software Design with Interfaces in Rust

  Read:

  - Kernighan and Ritchie: pages 114—115 (argc and argv); page 162 (the cat program); pages 164—165 (fgets); pages 153—155 (printf)
  - Williams and Kessler, *All I Really Need to Know About Pair Programming I Learned in Kindergarten*
  - Rust Book, Ch. 1-4

  A1-Intro assignment out 1/23, design due 1/26, implementation due 2/2 Rust Setup lab 1/22 and intro pnmdata lab 1/29

- **iii**

  Interfaces, Implementations, and Images

  Read:

  - Norman Ramsey, *On Design and Design Documents*
  - Rust Book, Ch. 5-7, 13

  A2-III assignment out 2/6, design due 2/9, implementation due 2/16
  Array lab 2/12
  Caching (stride) lab 2/19 (homework server needed)

- **locality**

  Machine storage and cache locality

  Read:

  - Bryant and O'Hallaron: Chapter 1 (skim), section 1.7 (read) and 6.2—6.5
  - Rust Book, Ch. 8-11

  A3-Locality assignment out 2/20, design/predictions due 2/23, implementation due 3/1
  Bash/testing lab 2/26

- **arith**

  Data: Bits, bytes, words, and arithmetic

  Read:

  - Bryant and O'Hallaron, Sections 2.1—2.4
  - Rust book, Ch. 11 (re-read!)

  A4-Arith assignment out 3/5, design/losses due 3/8, implementation due 3/22
  Diff lab 3/4
  Bitpack testing lab 3/18

  From higher-level code to machine instructions

  Read:

- Bryant and O'Hallaron, Sections 3.1—3.7, 3.13 (AMD64), 3.8—3.10
- Rust book, 15-18

A5-Code assignment (binary bomb) out 3/21, due 3/28 (Thursday)
GDB lab 3/25

- **um**

  Machine code: Virtual Machines

  Read:

  - Handout on ICFP 2006 Programming Contest
  - Rust book, 15-18

  A6-UM assignment out 4/2, design due 4/5, implementation due 4/12
  UM-Disassemble lab 4/1 UM-Assemble lab 4/8

- **profile**

  Profiling and performance tuning

  Read:

  - Bryant and O'Hallaron: Chapter 5
  - Handout on performance tuning
  - Rest of the Rust book

  A7-Profile assignment out 4/16, due 4/23 (no design)
  Profiling lab 4/15

- **asm**

  From higher-level code to machine instructions

  Read:

  - Bryant and O'Hallaron, Sections 3.1—3.7, 3.13 (AMD64), 3.8—3.10
  - Bryant and O'Hallaron, Sections 7.1 to 7.9, Section 7.13
  - Handout on the UM Macro Assembler
  - Programming the Universal Machine (Handout)

  A7-asm assignment out 4/23, due 4/30 (no design)