# Syllabus

Written by Julie Zelenski

Below is our plan for the quarter. Topics may occasionally be shuffled around and we will keep this syllabus updated to reflect the current schedule. Exam dates are set at quarter start and will not change.

In the readings, B&O is Computer Systems (Bryant and O'Hallaron), K&R is The C Programming Language (Kernighan and Ritchie), and EssentialC is a PDF available at http://cslibrary.stanford.edu/101 (http://cslibrary.stanford.edu/101). A digital copy of K&R is available to Stanford students via Safari Books Online (http://proquest.safaribooksonline.com.ezproxy.stanford.edu/book/programming/c/9780133086249). Read before attending lecture and lab for best effect. Code from lecture will be posted to the class directory in `/afs/ir/class/cs107/samples/lectN` where N is the lecture number.

| Topics | Reading/Handouts |
|---|---|
| *Week 1* | |
| **Lecture 1 (Mon): Welcome**<br>Join us for an overview of what to expect in CS107. We'll go through course logistics, learning goals, and a few demonstrations of the surprising realities of computer systems. (my slides (lect1.pdf)) | Handout 1: Course overview (handout1.pdf)<br>B&O Ch 1 (skim chapter for quick overview of what is meant by systems and preview of topics to come)<br>Systems programmers and zombies (https://www.usenix.org/system/files/1311_05-08_mickens.pdf) (a fun must-read on the adventure ahead!) |
| *No lab during first week (optional unix help sessions listed on calendar (officehours.html))* | |
| **Lecture 2 (Fri): Transitioning to C**<br>A whirlwind tour of C for C++ programmers using the unix environment. (my slides (lect2.pdf)) | Brush up on C syntax, data types, operators, control structure, function calls.<br>K&R Ch 1, 2, 3 (skip sections on arrays until next lecture) or EssentialC sections 1, 2, 4 |
| *Week 2* | |
| **Lecture 3 (Mon): C pointers & arrays**<br>Now it's time for the tricky parts of C, including use of * and &, arrays and pointers, pointer arithmetic, and the dynamic allocation functions malloc/realloc/free. | K&R Ch 1.6, 5.1-5.5 or Essential C section 3 on mechanics of pointers and arrays. Pay special attention to the relationship between arrays and pointers and how pointers/arrays are passed as parameters. Oddly enough, K & R doesn't have much to say about using malloc/free (although |

| | |
|---|---|
| | section 8.7 talks about how to implement malloc!); for this, I recommend a careful read of section 6 of Essential C. |
| **Lab 1: C programming under Unix**<br>Hands-on practice with C-strings and unix development tools | K&R (1.9, 5.5, Appendix B3) or Essential C section 3 for C-strings and string.h library functions. C-strings are primitive compared to Java/C++ strings, take note of the manual efforts required for correct use and pitfalls to avoid. Peruse our Mercurial (guide_mercurial.html) and gdb (guide_gdb.html) tool guides. |
| **Lecture 4 (Fri): More on pointers**<br>Follow-up on C-strings after this week's lab, trying to tease out the minute differences between arrays/pointers (while reinforcing ways they are the same), and compare/contrast stack and heap allocation. One tricky issue to work through is passing pointers themselves by reference. | K&R 5.6-5.9 (skim parts on multi-d arrays) |
| *Week 3* | |
| **Lecture 5 (Mon): void * , function pointers**<br>Pointer typecasts, untyped `void*` pointers, and how used for generic behavior. Function pointers and use as client callbacks. Writing generic functions, being a client of generic functions. | K&R 5.11 (function pointers). There isn't much new reading for `void*` , be prepared for further application of mechanics of pointers, arrays, and pointer arithmetic to manipulating pointers of indeterminate type. |
| **Lab 2: C memory**<br>Examining pointers/arrays, using/writing C generics, using tools to understand and debug memory | Our Valgrind (guide_valgrind.html) and gdb (guide_gdb.html) tool guides |
| **Lecture 6 (Fri): Pointer wrap, bits, bytes, and bitwise operators**<br>Resolve any open issues on `void*` and review rules for wise use of memory. Bring questions about anything still unclear in the realm of pointers. Then we drop down to examine bits and bytes and introduce the C bitwise operators. | K&R 2.9 and B&O 2.1 on bit ops and data representation (skip 2.1.7 on boolean rings). |
| *Week 4* | |
| **Lecture 7 (Mon): Data representation, integer types**<br>Integer-family types (chars/short/long/pointers). ASCII | For integers, B&O Ch 2.2-2.3 (skim the formal proofs, but important to take away solid working |

| | |
|---|---|
| character set. Two's complement representation and properties of integer arithmetic. Unsigned vs signed types. Operations on mixed types. Truncation and sign extension. | knowledge of two's complement and behaviors of integer operations). This is important reading to have done before lecture/lab! |
| **Lab 3: Bits and bytes**<br>Hands-on practice with bits and integers | |
| **Lecture 8 (Fri): Floating point**<br>IEEE floating point representation. Understanding the features/limitations of the floating point number line. Rounding, exceptions. | B&O 2.4 on floats. (skim formal proofs, strive for reasonable understanding of floating point representation and its limitations). You'll get much more from lecture/lab if you have done this reading beforehand! |
| *Week 5* | |
| **Lecture 9 (Mon): Intro to IA32, data movement**<br>Representation of pointers, structs, instructions. Introduce assembly/machine language and find out what's happening underneath the hood of the C compiler. The IA32 instruction set architecture and its almighty `mov` instruction. Addressing modes, data layout, and access to variables of various types. | B&O 3.1-3.3 for background info on IA32 and machine code. Very carefully read B&O 3.4 on addressing modes and data transfer. The multitude of addressing modes is one of the things that puts the C in CISC for IA32. |
| **Lab 4: Floats**<br>Hands-on float dissection | |
| **Lecture 10 (Fri): Data layout and access, ALU ops, control structures**<br>Relate remaining addressing modes to their use in accessing C variables/pointers/arrays/structs. Then it's on to the arithmetic and logical ops and a start on how control structures are implemented (if-else, loops, switch). We'll be tracing through much assembly code in these lectures. By having done the reading before lecture, you'll be in a better position to follow along without getting lost :-) | B&O Ch 3.4 on data layout and access, 3.5 on ALU ops, 3.6 on control structures. There is a lot of detail in these sections, especially when absorbing the IA32 assembly, but resist the temptation to skim. It is key to get a solid foundation on these IA32 basics which requires following the assembly closely and working the self-test exercises to be sure you have it down |
| *Week 6* | |
| **Lecture 11 (Mon): Function calls, runtime stack**<br>Implementing loops (for/while/do-while) and two implementations for switch statements. Implementing function calls: IA32 instructions push/pop/call/return, how `ebp` and `esp` are managed, and linux C calling conventions | B&O Ch 3.6 on switch. B&O Ch. 3.7 on function calls/stack. |

| | |
|---|---|
| and parameter passing rules. | |
| **Lab 5: IA32**<br>IA32 in all its glory | Our IA32 (guide_ia32.html) guide |
| **MIDTERM EXAM**<br>Friday Oct 30th 1:30-2:50pm<br>No alternate/makeup exams | |
| *Week 7* | |
| **Lecture 12 (Mon): Runtime stack, IA32 wrap**<br>Caller/callee protocol for function calls, parameter passing, call/return, and management of stack housekeeping. Use this understanding to explore stack features: why recursion works, why locals are cheap, how variable arguments work, consequences of various stack misuses (uninitialized locals, returning pointer to local, overrun/underrun on stack variables, and so on). Resolve dangling IA32 issues: caller/callee-saved registers, how stack fits into address space, and binary encoding of machine instructions. | B&O 3.7 |
| **Lab 6: Dissecting the runtime stack**<br>One of my favorites of all the labs-- lots of fun explorations with the stack! | |
| **Lecture 13 (Fri): Build process**<br>The step-by-step process of how C code becomes an executable. Preprocessor, compiler, assembler, linker, loader, what each tool does and how they fit together. Symbol resolution, relocation. Basics about libraries: static, dynamic, shared. | B&O Ch 7 (skip 7.12 on PIC) |
| *Week 8* | |
| **Lecture 14 (Mon): Managing the heap**<br>How the heap fits into the address space. Design decisions for implementing malloc/realloc/free. Performance tradeoffs (throughput, utilization). Compare/contrast stack versus heap allocation. | B&O Ch. 9.9-9.11 covers heap allocation implementation, garbage collectors, and memory misuses. There is a lot of very detailed code in 9.9.12 It's ok to skim this code for now (if you are sure to understand the underlying principles) but you will eventually be assigned the job of writing a heap allocator and will want to be intimately familiar |

| | |
|---|---|
| | with this code at that point, so making the investment now will pay off later. :-) |
| **Lab 7: Build process**<br>Compilation steps, tools for examining executables | man pages for `nm` and `readelf` |
| **Lecture 15 (Fri): Code optimization**<br>What optimizing compilers do and don't do. Explore transformations such as constant folding, common subexpression elimination, strength reduction, code motion, loop hoisting. Quick overview of instruction-level parallelism, superscalar execution, and pipelining. Timing tools and profilers to measure code performance. | B&O 5.1-5.6 and 5.13-5.15, skim 5.7-5.11. (read 5.12 for next lecture) |
| *Week 9* | |
| **Lecture 16 (Mon): Memory hierarchy, locality, caching**<br>Understanding storage hierarchy (registers -> caches -> memory -> files) and how properties change as you move downward (fast/small/expensive => large/slow/cheap). Temporal and spatial locality of reference and its effect on performance. Caches and writing cache-friendly code. Measuring memory performance using callgrind with cache simulation. | B&O 5.12, 6.1-6.3 and 6.5-6.7, skim 6.4. The reading gets fairly dense, most important to get big picture. |
| **Lab 8: Code and memory optimization**<br>Experiments in optimization, profiling using callgrind | Our callgrind (guide_callgrind.html) guide |
| **Lecture 17 (Fri): STOKE**<br>Prof. Alex Aiken, chair of CS department, joins us to share lessons learned and surprising results from a research project to investigate whether it is possible to automate (or at least improve) low-level, performance-oriented programming through randomized search techniques and massive amounts of computation. Prof. Aiken is a great speaker, the topic is very accessible to a 107 audience, and you'll gain some insight into the fascinating research happening in computer systems.**Don't miss this neat opportunity!** | |
| *Week 10* | |
| Thanksgiving recess | |

*Week 11*

| | |
|---|---|
| **Lecture 18 (Mon): Python**<br>And now for something completely different... an introduction to Python. Still have same metal underneath but it's intriguing to see the different abstraction presented by a dynamic, interpreted language compared to the static/compile-time orientation of C. | Python notes (http://cs.stanford.edu/people/nick/python-in-one-easy-lesson/) (stolen from Nick Parlante) |
| **Lab 9: Python**<br>Fun with python | |
| **Lecture 19 (Fri): Wrap**<br>Compare/contrast python versus C after experiencing python first-hand in lab. Wrap up course themes, preview courses/opportunities post-107, and try to reach closure on any loose ends. Bring questions if you have them! | |

*Week 12*

**FINAL EXAM**
Monday Dec 7th 12:15-3:15pm
No alternate/makeup exams