Getting Ready for CS 367

A guide for students coming from CS 262

What CS 367 is really about

CS 367 is where you connect C code to the machine underneath it:

- **Representation:** bits, bytes, two's complement, floating point, and why hex shows up everywhere.
- **Execution:** assembly, stack frames, calling conventions, and what a program *actually* does when it runs.
- **Systems:** linking, processes, signals, Unix I/O, virtual memory, caches, and the heap.
- **Projects:** you will extend provided skeleton code and implement key pieces correctly and safely.

0. The single most important skill: reading code

In CS 367, projects commonly provide a working framework and ask you to implement specific functions. This mirrors real software work where you rarely start from a blank file.

A code-reading routine that pays off (both in class and in future internships/jobs)

When you get a project handout:

1. **Build and run first.** Make sure the starter code compiles and you understand how to execute it.
2. **Map the architecture.** Identify the main loop / control flow: who calls what, and when?
3. **Read headers before implementations.** Learn the structs and function behaviors.
4. **Trace one feature end-to-end.** Follow a single action through the system (even if it hits stubs that you will later be implementing).
5. **Write down or add comments for invariants (the rules that the code is relying on).** Examples: "Ready queue is sorted by PID", "flags store state in bits", etc. Adding these comments as you read will help you quickly remember how everything fits together

1. What to be proficient in before day 1

1.1 Hex and base conversion (non-negotiable)

Memory addresses, masks, bitfields, cache tables, and debugging output are commonly in hex.

Hex to binary map cheat sheet

0	0000	1	0001	2	0010	3	0011
4	0100	5	0101	6	0110	7	0111
8	1000	9	1001	A	1010	B	1011
C	1100	D	1101	E	1110	F	1111

Can you do these comfortably?

- Convert between binary, decimal, and hex (both directions), including negative numbers.
- Interpret a hex address as bytes.
- Extract bit ranges using shifts and masks (lecture 17).

1.2 Pointers and dynamic memory (you will use them constantly)

You should understand and be able to reason about:

- stack vs heap, lifetime (like what happens to a function's local stack data after it returns?), and what `free()` must pair with.
- pointer aliasing (two pointers to the same memory) and how this works.
- how arrays and pointers relate (and where they do not).

Can you do these without guessing?

- Write a function that allocates a struct, initializes it, and returns it (and document who frees it).
- Explain a segfault by drawing a quick memory diagram.

1.3 Linked lists and linked structures

Linked structures show up directly in CS 367 projects (queues, job lists, etc.).

Common linked list patterns to practice and revisit

You should be able to implement each pattern cleanly and safely:

- insert at head / tail
- insert into sorted order (by key like PID)
- delete by key (including deleting the head). *Note:* just like in our 262 section, avoid using dummy nodes.
- move a node from one list to another

2. Topics you can expect in CS 367

- **Bitwise ops in C:** masks, shifts, extracting/setting bits.

- **Signed vs unsigned:** two's complement, casts, overflow behavior.
- **Floating point:** IEEE-754 (floating point) ideas, rounding, special values (NaN, $\pm\infty$), and custom encodings. (Just Google IEEE 754 converter if you want to see the awesomeness of how floating points work the same way we did in class)
- **assembly:** registers, addressing modes, `mov`, `lea`, arithmetic, control flow, calls/returns, stack frames. (We only looked at assembly code and talked about it at a high-level. In 367, you will even learn how to write it)
- **Linking:** symbols, object files, relocation, strong vs weak symbols, and what the linker resolves.
- **Processes and signals:** `fork/exec/waitpid`, signal handlers, job control, process groups.
- **Unix I/O:** file descriptors, `open/close/read/write`, redirection via `dup2`.
- **Memory systems:** caches and locality, virtual memory and address translation, heap allocation concepts.

3. What the projects feel like (so you can prepare mentally)

Project style

A common pattern is: the framework is given, and your job is to implement specific functions correctly. Adopting the philosophy of "finish the system, don't rewrite it" is helpful for this course (and even more so for 471).

Two habits that save hours

- **Treat every function like an API:** assume other code will call your function. Validate inputs, follow the written requirements, and return the exact values it expects.
- **Keep the program's promises true (formally, this is called preserving invariants):** if the code assumes "this queue is sorted" or "this count equals the number of nodes," make sure your changes don't break those assumptions.

4. Tools you should be ready to use

Tool readiness checklist

- Compile from the terminal with flags (warnings matter).
- Use `gdb` (or `lldb`) to step, set breakpoints, inspect memory/registers.
- Use `valgrind` to find leaks and invalid memory access.
- Use `man` pages for system calls (`man 2 open`, `man 2 fork`, `man 2 waitpid`, etc.).
- Comfortably navigate a Linux filesystem and use `make`.

5. Final advice

How to stay (relatively sane) in CS 367

- **Be systematic.** Small changes, frequent tests, and careful reading will always beat 16 hours of frantic programming the day something is due.
- **Draw pictures.** Memory diagrams make invisible bugs visible.
- **Respect invariants.** Most "mystery bugs" are invariant violations.
- **Ask good questions when debugging.** "Here's what I tried, here's what I expected, here's what happened."

Lastly, if there is anything you encounter that you feel you should have been more prepared for after leaving 262, please send me an email so I can make it right. Part of my teaching philosophy is that my commitment to you doesn't end just because you've left my class.