Bradfield
SCHOOL of COMPUTER SCIENCE

March 2022

Preparation Phase

Congratulations on your acceptance to the Bradfield Computer Science Intensive! We look forward to joining you on what we hope will be a deeply rewarding journey.

Prior to the start of regular classes, we would like everybody to independently work towards a baseline level of familiarity with some foundational topics.

If you're relatively new to a topic, we've provided suggested resources and study techniques. Once you're ready, please attempt the "completion exercises" to consolidate your understanding and check your progress.

If on the other hand you're already familiar with a topic, that's great! You're welcome to skip the suggested resources and study techniques, but we still suggest attempting the "completion exercise" to confirm your familiarity.

You're welcome to complete these in any order. However, we suggest going in the order given, primarily to maintain some synchronization among the group and prioritize the most relevant topics. Our weekly checkins during the preparation phase will be open-ended, but we generally plan to discuss topics in the order given.

If you have questions or comments, please post them to the #csi-4 Slack channel as doing so will likely benefit your classmates. We're happy to provide feedback on the completion exercises in whatever manner you'd like, but we suggest posting your solutions to Slack so that we can discuss them as a group.

1 The C Programming Languge

C is one of the most widely used programming languages, and possibly the "canonical" systems language. It's a powerful language that gives more control and visibility over the underlying hardware, but of course "with great power comes great responsibility". Throughout CSI we'll use C for some lower-level exercises.

We suspect that you'll quickly pick up C syntax, and that control flow will feel familiar. However, working with structs, pointers, and arrays may present more of a challenge, so you may want to focus your attention there. Furthermore, it may take you some time to get used to manual memory management (e.g. using `malloc` and `free`).

Our suggested approach for learning C is to work through the classic "K&R" book (*The C Programming Language*). If you've had some prior exposure to C, you may be able to focus on chapters 5 and 6. As you work through the book, we suggest picking out interesting exercises to test your understanding, and completing as many as you need to feel comfortable with the material (perhaps every other or every third exercise).

As an alternative to "K&R", for instance if you're more "challenge" oriented, you may wish to work through some of the C track on Exercism. As usual when diving in to a new language, you may find it helpful to have Hyperpolyglot and/or Learn X in Y minutes open while you work.

Completion exercise

Please implement a minimal clone of the `ls` program. We have chosen this exercise as it will require you to use structs, pointers and arrays, as well as some C standard library functions with interesting interfaces. It will also likely to be substantial enough to merit some degree of code organization.

Minimally, it should list the contents of a directory including some information about each file, such as file size. As a stretch goal, use `man ls` to identify any interesting flags you may wish to support, and implement them.

2 The Go Programming Language

We'll be using Go as the primary language for CSI. "Go is an open source programming language that makes it easy to build **simple**, **reliable**, and **efficient** software." Sometimes described as "C for the 21st century", Go is especially well suited for building modern server-side infrastructure.

Getting started with Go will primarily involve getting used to some minor syntax differences (e.g. `x int` instead of `int x`), but mastering Go will involve internalizing many interesting principles (for example, see Effective Go), as well as familiarizing yourself with the module system, command line tools, and standard library.

Our suggested approach for learning Go is to work through *The Go Programming Language* (note that the coauthor Brian Kernighan is the same "K" from the "K&R" book).

You may want to pay closer attention to chapters 4 (Composite Types), 7 (Interfaces), and 8 (Goroutines and Channels). On the other hand, feel free to skip chapters 9 and 12-13. Again, we suggest picking out interesting exercises to test your understanding.

If you'd prefer a more "challenge" oriented approach, you may wish to work through some of the Go track on Exercism while keeping Hyperpolyglot and/or Learn X in Y minutes open.

Completion exercise

Please complete **one** of the following two exercises from *The Go Programming Language*:

- **Exercise 4.12**: The popular web comic *xkcd* has a JSON interface. For example, a request to https://xkcd.com/571/info.0.json produces a detailed description of comic 571, one of many favorites. Download each URL (once!) and build an offline index. Write a tool `xkcd` that, using this index, prints the URL and transcript of each comic that matches a search term provided on the command line.
- **Exercise 8.7**: Write a concurrent program that creates a local mirror of a web site, fetching each reachable page and writing it to a directory on the local disk. Only pages within the original domain (for instance, `golang.org`) should be fetched. URLs within the mirrored pages should be altered as needed so that they refer to the mirrored page, not the original.

In addition, please add some tests and benchmarks to familiarize yourself with the capabilities of the `testing` package.

3 The Unix Shell

While most of our programming will be in Go and C, we will sometimes use the shell and many common Unix utilities as "glue code". We also strongly believe that familiarity with the Unix command line will significantly increase your productivity as an engineer. We suggest Bash as a shell program of choice due to its ubiquity.

To increase your familiarity with Bash, we suggest working through the tutorials (and particularly, the exercises) for any unfamiliar concepts on learnshell.org. We also recommend reading through the Bash man page and writing short scripts to explore any unfamiliar concepts you encounter there. Finally shellcheck is one of the best linters we've seen, so we encourage you to install it, integrate it into your prefered text editor, and heed its warnings.

We unfortunately don't have a recommended general resource for learning about Unix commands, but we suggest familiarizing yourself with the following:

Text processing	grep, sort, uniq, cut, wc, sed, strings, head, tail, awk
Process management	ps, fg, bg, jobs, kill
Filesystem	cd, cat, cp, rm, ls, mv, ln, file, chmod, chown, du, mkdir, mkfifo
Shell use	man, help, echo, apropos, tee, test, xargs

We appreciate that it may be hard to learn these without motivating problems, but they are so diverse that we haven't been able to find a completion exercise that covers a significant portion of them. Aside from the completion exercise below, we'd also encourage you to identify any entirely unfamiliar commands and consider ways that they may be helpful in your own work.

Finally, you may appreciate More shell less egg, a fun case study of the power of the shell.

Completion exercise

Write a Bash program that shows summary information from Wikipedia. `wiki walrus` should show the first sentence of the "Walrus" article, along with a list of section headings. `wiki walrus anatomy` should show the first sentence of the anatomy section, along with a list of subsection headings.

- You are welcome to use `curl` to retrieve the page itself, as well as any commands that you'd expect to find on a typical Unix family OS distribution.
- As a stretch goal, add support for tab completion in Bash, ie `wiki walr[tab]` should expand to walrus, and `wiki walrus anat[tab]` should expand to anatomy.
- As another stretch goal, add a flag to support querying multiple pages in parallel.

4 Algorithms and Data Structures

"Algorithms + Data Structures = Programs" –Niklaus Wirth

We agree with decades of conventional wisdom that familiarity with common algorithms and data structures is one of the most empowering aspects of a computer science education. This is also a great place to train your general problem solving abilities, which pays off in every other area of technical study.

Our most important recommendation for learning algorithms and data structures is to **solve a lot of problems**. Even when working through a textbook, we recommend attempting each example on your own before reading the solution in the text. That being said, please don't "grind" problems! Instead, treat each problem as an opportunity to discover interesting ideas.

- Focus your attention on problems where you're genuinely interested in a solution
- Being "stuck" is to be expected and welcomed; when this happens, we recommend applying a methodical problem solving process
- Be patient, take your time, and embrace Hammock Driven Development

Working through *Practical Algorithms and Data Structures* will introduce you to many fundamental algorithms and data structures in a practical context. You might want to skip the following sections, especially on a first reading: `Analysis 4; Stacks 5; Recursion 5-6; Trees 6-7; Graphs 4, 7-9`.

Optionally, if you'd like to augment your reading with video lectures, you might enjoy Steven Skiena's algorithms course. To avoid being a passive consumer, whenever Skiena presents a problem you find interesting, we suggest pausing the video and trying the problem yourself.

Completion exercise

Please solve **three** of the following problems, prioritizing problems you haven't seen and/or problems you find particularly interesting:

Course Schedule	Number of Islands
Open the Lock	Spiral Matrix II
Word Ladders	Minimum Depth of a Binary Tree
Merge k Sorted Lists	Search a 2D Matrix
Valid Parentheses	Word Search