# 1 System design

I originally was going to use getopt, but due to the non-POSIX nature of the command structure, I decided to manually interpret the user input. Obviously as there were user requirements for this assignment, the system design was outlined around that. The solution was generated by creating functions for each command line option, sketching out the likely algorithm to use, and divide functionality into reusable subfunctions as appropriate.

One of the critical design decisions was to ensure that instances of `struct ar_hdr` remain null terminated. Thus the static buffer-based `fix_*` functions for time, permissions, and generic strings were implemented. Headers are grabbed using an iterator (`get_next_header`), although I originally decided to construct a list of all headers this seemed to be more efficient overall.

The major subfunction (not related to archive integrity or conversion) that was able to be used multiple times was `delete_file`. If I were to rewrite this code, I would probably attempt to break down into functions further, however there were no major possible refactoring based on the current implementation.

# 2 Work log

# 3 Challenges

The biggest challenge was simply building an intuitive interface with rigorous error checking enabled. All system calls require error checking to be ran properly, so implementing this in a helpful, informative, and visually pleasing method was... difficult. I also had difficulty reporting errors in an informative fashion, instead of just puking and exiting. Before my next assignment begins, I intend to spend a lot of time brushing up on C error handling (especially in regards to a CLI program).

# 4 Questions

## 4.1 Main point of the assignment

The main point of the assignment was most likely to get everyone comfortable in C programming by providing a thorough refresher. Due to the complexity of this program, a lot of research and man page viewing was required to get up to speed, as well as a plethora of library functions and system calls.

## 4.2 Solution testing

Since we were essentially black boxing `ar`, my testing largely consisted of (1) does it meet assignment requirements for expected functionality and (2) do so in a way that is highly compatible with the existing `ar` utility. This meant ensuring that all archive were valid to `ar` after all operations performed by my implementation of it. Fortunately the archive is very human readable, so it was easy to tell when things weren't working correctly (and then fairly easy to fix using `gdb`).

## 4.3 I learned...

I have a reasonable amount of C and GNU/Linux experience, so what I mostly learned was how to access Linux system calls properly. I am familiar with the utilities that you interface with (such as `rm`) however I had not previously used their system call correspondents (e.g. `unlink`).