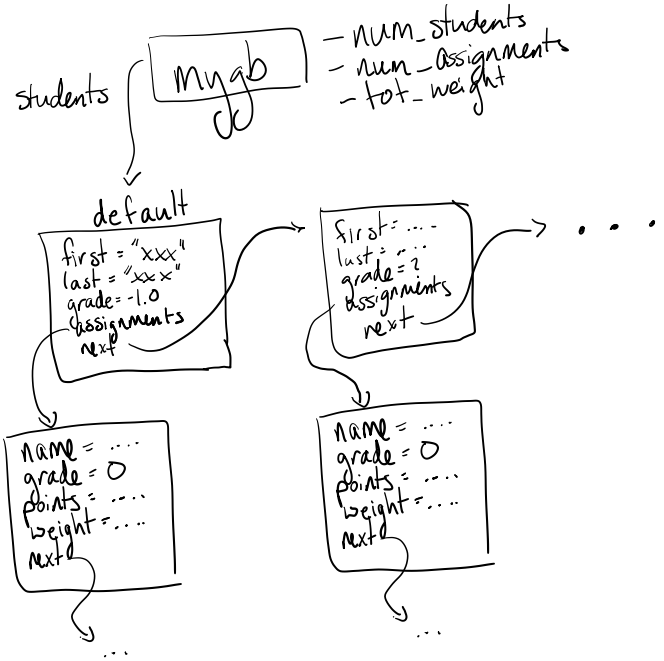
Design Document

I decided to create a structure that would hold all of the necessary information about the gradebook. It was structured so that the gradebook has a linked list of all the students who in turn had a linked list of all the assignments. If a student hadn't completed an assignment, they were given a score of -1 so that it wouldn't be included when calculating their final grade or when printing an assignment. The gradebook also knew how many students there were in the class and how many assignments. This helped when writing to and reading from a file. I essentially made a "recipe" for writing the gradebook to a file and reading it back out. When reading it back out I would rebuild the gradebook structure by reading through the bytes that I had organized in a specific fashion. Since the strings were variable length, I had to include a leading byte before every string so that I would know how many bytes to read in for that string. Every student had a copy of every assignment to make things easy to store and read back out. I calculated their total grades as they received new grades on assignments to make it easy to sort at the end. This was stored in the student grade variable. Each assignment in the student's assignment linked list kept track of its name, points, weight, and what grade that specific student received on the assignment. To make sure the gradebook wouldn't lose track of how many assignments there were, even if all the students in the class were removed, I added a default student with the name "xxx xxx". There are checks to prevent a user from modifying this student by either adding grades or removing them. The job of this student is merely to keep track of all of the assignments. New users are always added to the end of the list of students. During gradebookdisplay the order will be re-arranged as necessary. Below is a visual of the setup of my gradebook. The gradebook is a global variable as well making it easy for each function to access the elements of the structure. The arguments passed in through the command line are also put into a structure so that they don't have to be passed in everywhere but rather accessed by any function.

```
struct student {
	char * first;
	char * last;
	double grade;
	struct assignment * assignments;
	struct student * next;
};

struct assignment {
	char * name;
	int grade;
	int points;
	double weight;
	struct assignment * next;
};
```

```
struct gradebook {
        int num_students;
        int num_assignments;
        double tot_weight;
        struct student * students;
};
```



To make this as easy to debug as possible I made several small functions to do tasks such as validate a filename, students name, or assignment name. I also validate the number inputs to make sure it is digits being input where necessary and not letters or symbols. For each possible action I create a function that will handle that action. The checking for the order and existence of arguments is checked as they are parsed.

Attacks

1. Buffer overflows: I decided on a linked list so that there wouldn't be a concern about a buffer being too small. By dynamically allocating space I am able to prevent buffer overflows.
    a. gradebookadd.c (148, 200, 263, 349, 426), gradebookdisplay(117)
2. Code injection and Overreads: I created functions that check the input to make sure it is safe before allowing the code to continue. This prevents code injection through user input or the ability to include a format string to read data the user should not have access to. I also included the -noexecstack to prevent the injection of code.

    a. setup.c (22), gradebookadd.c (535, 544, 553, 562), gradebookdisplay.c (390, 399, 408), Makefile (2)
3. Malicious files: Before doing any sort of computations on a file I use an authenticated encryption scheme to check that the file hasn't been modified by an adversary.
    a. setup.c (31), gradebookadd.c (35, 90), gradebookdisplay.c (59)
4. Stack smashing: I included the -fstack-protector compilation flag to add stack canaries and prevent stack smashing
    a. Makefile: (2)
5. Memory Leak: At the end of both gradebookadd.c and gradebookdisplay.c I call a destroy function. This goes through the entire gradebook structure and cleans up all of the malloc-ed memory. This is to prevent any sort of memory leak that could allow an adversary to gain information they aren't supposed to have.
    a. gradebookadd.c (571), gradebookdisplay.c (417)
    b. I am aware that I am missing some locations but I tried my best