Keeping a Logbook

Professionals who spend much of their time debugging most often agree on one thing: If you are going to be doing even a moderate amount of debugging in your life (if you are a CS or SE, this will be a way of life for you most likely), you should be keeping a logbook.

So how can this help you...

Now?

Keeping a logbook will help on an immediate basis (more than you might think). When you log a bug that you have found and fixed, it forces you to think about the situation in a different way. You (should) realize you could be reading this very log 3-5 years from now so you better be clear about everything. This forces you to understand the problem MUCH more deeply than just fixing the bug and moving onto the next thing. How many times have you moved code around, "miraculously" fixed a bug and had no idea how? Well... sadly, these fixes don't often fully fix the issue, or they at least cause other problems most of the time. Logging forces you to understand what actually fixed the problem, and most likely how the problem came to be in the first place. This will save you time in the future too: not having to go back to code you "fixed" before only to fix it again.

Later?

For every bug you find and fix, you will most likely see a similar bug again in the future. If you log what you did to find and fix it, you could save the "future-you" a lot of time by logging things in the present.

Format

The most important thing here is that you use what you feel comfortable with (although learning new things is rarely a bad idea). For class, you will be turning in portions of your logbook often. So if you write it in a notebook, you will need a way to turn in a digitally scanned copy.

Remember that this logbook (outside of this class) is mostly ever going to be for you. I (the instructor) realize that out of the entire class, maybe 2 or 3 of you will continue this habit later after college, but it will have been worth it even if one of you does. In the meantime, you will at least have to get in the habit for this class.

For your logbook, there are many options:

1. You can use a physical notebook
    - Pros - Usually easy to reach and access; Can be formatted by hand very easily
    - Cons - Searching capabilities is very limited; Adding things later can be difficult; Very limited sorting capabilities; A finite notebook will eventually fill up and you'll need more; old notebooks may not get used; "oops, I left this in my other office"
2. You can use a digital notebook
    - Pros - Searching through book can be easier depending on type of notebook; Easy to add to later
    - Cons - Access from several places must be coordinated with something like git if the files are local to the machine (unless using Google docs or something)
3. You can use a spreadsheet (Google docs)
    - Pros - Searching AND Sorting (by several fields like date, description, etc.) are easier; easy to add to; can get VERY powerful with formatting; easy access from anywhere
    - Cons - Sometimes formatting is a pain; you don't want the activity of logging to take too much time
4. You can even use a database (like Airtable, or even a DB for bugs like Bugzilla or Jira)
    - Pros - Things like Airtable can be setup exactly how you want it; It either already is or can be setup exactly to track bugs
    - Cons - Can be overkill; Time to set up may or may not be worth it (e.g., not worth it for this course)

What goes in it?

The short answer is any and all information you feel could be useful to you when you are debugging something that is, or appears to be, similar. This is your book, and you know what makes sense to you. **For this course**, there will be a minimal requirement. That is, you should *at least* record:

1. The bug description, which includes the type/classification of the bug, language of the code, full description of the problem, when it happened, etc.
2. **What you did to find the bug (notes as you go along are almost always helpful)**
3. What you did to fix the bug
4. Why the fix worked

For number 2 above, in our first week of the course, this is basically something you write down after you find the bug. However, soon afterwards, we will cover the Scientific Method. When we do, we will discuss the intermediate things you should write **as you are debugging**. There are examples of both below.

An Example

For this example, let's log the first bug we saw in the class from the Sample.c program (the program that takes several numbers from the command line and uses shell sort to sort them).

The log should look something like the following. I chose to use a spreadsheet format:

	A	B	C	D	E	F	G	H
1	**Bugs in C**							
2		**Program Name**	**Date/Time**	**Description**	**How found?**	**How fixed?**	**Why it worked?**	
3		Sample.c	7/25/22 12:15 PM	Out of the command line arguments the program was supposed to sort, some values would go missing and the value 0 would appear.	1) At first I tried X			
4					2) Then I tried Y			
5					...			
6					...			
7					Discovered using printf statements that the value of argc (which is always one more than the number of arguments passed in from the command line) was getting passed in for the number of elements to sort. Therefore, the shell_sort function was always getting a size of one too many.	Passed in argc-1 to shell_sort	Since argc contains the count of arguments plus the name of the program, when using the command line arguments passed in, the number of them will always be argc-1; Passing in argc-1 to shell_sort is needed since that is the actual number of arguments passed in.	
8								
9	**Bugs in Java**							
10		**Program Name**	**Date/Time**	**Description**	**How found?**	**How fixed?**	**Why it worked?**	

Scientific Method

After we discuss the Scientific Method more clearly, your logbook could look more like the following:

Bugs in C						
	Program Name	**Date/Time**	**Description**			
	Sample.c	7/25/22 12:15 PM	Out of the command line arguments the program was supposed to sort, some values would go missing and the value 0 would appear.			
	Hypothesis		**Prediction**	**Experiment**	**Observation**	**Conclusion**
	Infection in shell_sort()		At shell_sort(), expect a[] = {1,4} and size=2	observe a[] and size	a[]={11, 14, 0} and size=3	rejected
	Invocation of shell_sort with size=3 causes failure		Setting size=2 should make sample work	set size = 2 using debugger	as predicted	confirmed
	...		...	...	...	...