WO1 Clayton E. Williams
C II: potter
2023.06.09

Project Summary

Potter takes a collection of students and time left to finish their pottery project. The grading of n students starts with student[0] and continues to student[n-1]. Each student takes 5 minutes to grade, so the allowable time to finishe is n * 5. The goal is to find the starting index that results in the most students failing with their time compared to time to grade. After this is determined, the program will print who the grading should begin with, and the names of the students who failed based on that iteration.

Challenges

My biggest challenge was trying to think through the logic of the iteration. The first loop was simple, however, incrementing the starting index, and returning to index 0 and continuing to the starting index was difficult to comprehend. I ended up coming up with a solution that didn't seem as elogant as some others I looked at, but in the end, it worked and makes sense to me.

Another challenge was deciphering the errors returned from valgrind. The memory leaks were easy to fix, but some errors such as conditional jumps were harder to track down. Valgrind gives a lot of information, and without continually running it as I built the project (similar issue with make), I was overwhelmed at the end when I finally did run it and had a dozen allocs that weren't freed, and hundreds of errors to be dealt with.

Successes

While my logic for iterating over the list of students multiple times, starting at a different index each time is bulky, I was able to successfully include some boolean logic to "overload" the function to make it do two different things depending on where it was called. A boolean passed to my grade_and_print function allowed me to call the same function in two locations and either identify failures and calculate best starting index, or simply print the failures from that starting index.

I think my design for my structure was a good design plan. After looking at the text, I decided to go with an struct of arrays, rather than an array of structs. With this design, I was able to include the size and capacity of the arrays inside the struct as member variables, which made it very easy to reallocate space as the arrays needed to grow from command line input of names and times. This was easier as well to pass as a pointer as well, which reduced the number of return statements needed and avoided having to copy the entire struct each time it was passed as an argument to a function.

Lessons Learned

Regarding my challenges with valgrind, including debug flags with valgrind was extremely helpful. Prior to including these, I was given lines where the error stemmed from, but those pointed to lines in the actual C library source code, such as printf.c, or strlen.c. The debug flags helped highlight where in my code the errors were occurring for me to fix. For example, when using strncpy on an already parsed string that has no newline character, one is not copied over. Later, when trying to print that string, because printf calls strlen to know how much to print, i was getting errors because strlen was failing due to the lack of a newline character.

Another lesson learned is to build the makefile from the start, and run the program off of that to benefit from the fail fast idea. I waited until I had functional code, then started running the program with make, which resulted in dozens of warnings that I could have caught and corrected incremementally as I built out the program. Some of these warnings required re-writes or refactoring in order to clear them up becasue of the structures or architecture I implemented.