AERE 361 Lab 8

Jordan Reese

March, 8 2018

1 Libraries, libraries everywhere and not a book to read

The first part of this lab was pretty straight forward. We had to create a library that held the integration functions that we wrote last week. This makes sense because what if someone were to want to use our code. Then having a library would make their lives much easier. We had to do a couple things first before everything fell together. We had to add a struct to our integration.c so passing in values would be easy. We also had to make a make file to compile everything easily. I did this after I figured out how to write the whole gcc line on the command line. This gave me a better idea on how to structure my make file. I really didn't struggle too much with this section after I figured out how a makefile was structured. I just read the manual to figure that out. The first part of the first section was quite like the first set of exercise. All we had to do was make a makefile to place some executables in the bin directory.

2 Full implementation

I call this section full implementation for good reason. This assignment pushed us to our limits. We had to know most of what we have learned this semester. We first had to start by reading all the documentation given to us or we didn't stand a chance to get this lab done. Once you read the pdf for parse csv you were even more confused. Well after I read the pdf I went to the tool directory and started reading those codes. There I found how to actually use some of the functions. I didn't know all the notation so I decided to reference my father. He has been a programmer all his life, so he is a wealth of knowledge. I also talked to Brian and he explained that we really didn't need to type much of it. We were meant to steal most of the code from our csv library and our past codes. I then started by pasting in the entirety of csvinfo.c. I used this to allocate the memory needed to store my matrix. I then had to steal more from the csvinfo to parse through the data. There was a caveat to that part though. We needed to write new callback function. That is where the meat of the program happens. I want to talk a little about my thinking. Starting from the top. I created my own struct called matrix write. This stores all the values I will be using in the call back functions to to the storing. I added a float option too that can be called if the -f flag is triggered. The next part is my two call back functions. I really had to tweak the first one, I also changed the names because they weren't very useful. I have a if statement checking to see if the -f flag was detected on the command line. If it was then I start using float data types, if not then continue on with integers. I also found out from my dad, the the -¿ deferences so it tell the program where to find the variables in the memory. I then will count my columns up by one. The countrow function is the call back 2 this just resets the column count and counts up my rows. Since call back 2 is only called at the end of each line it makes it easy to write. At the end of the the csvinfo you can see my memory allocation. Now I want to say, there is a bug in my code. It assumes that all the rows have the same number of columns. This is because it would be difficult to tell how many columns were in each row. Info doesn't set things up that way. So yes, a bug. I then just used the malloc code from lab 5. The part after that is just initializing my structure so it doesn't have garbage in it. Then it is all code that you have seen before. At the end I free up the memory and close the file.

3 Code Details

Loop invariants were fairly infrequent at least from my end. I didn't have many loops that I wrote. Most of it was taken from csvinfo.c. The complexity varies throughout the code. In the parsing it is completely linear. Only once we get to the print matrix do we have a larger complexity. It is N squared.