Jay Staddon

CS23820
Ceredigion Area Drone Racing Competition Assignment

Email: jas130@aber.ac.uk

Date: 22/12/2021

My C program overview

Data storage

When it came to tackling this task, I first looked at the different ways I could store data in the program. As the assignment requires there to not be a finite number of competitors the two ways I came up with were a linked list and a dynamic array. I settled on the dynamic array for storing competitors, as this way I would be able to treat all the competitors as just simple array values. My choice for declaring the dynamic array was calloc because it can hold a number of elements with a certain size of bytes. I used a linked list to store each competitor's track results, meaning I would be able to easily add data without having to reallocate memory space for each track added.

To increase the size of the array I used realloc however, I had some trouble implementing it in a separate function. I attempted lots of different designs however I kept finding that once the function that called realloc had finished the array would go back to its old size. The way I got around this problem was by implementing the realloc function code inside the main function, meaning I would have to quit the menu and then reopen it when increasing the array size.

Structures

To make the competitors and tracks I opted for using structures as I would be able to hold multiple pieces of data in one variable. I gave the competitor in the CDP a pointer to a track, allowing the competitor to find the top of the linked list.

Menu

I decided to create a main menu for both programs to make it easier for the user to control the different functions I have created. It has an exit program option as well, meaning the user can also end the program whenever they like. I used a switch statement to traverse the menu meaning it will loop until the user finally presses the exit button.

Loading file

I considered the different ways of loading in data from a file and decided that using fscanf was going to be best. I would load in the data from the file and then save each value to a new competitor or track. However, I realised names can have spaces meaning fscanf wouldn't pick up on that. I then changed any data that was allowed a space to be found using fgets, which will read a line instead of just the word.

Saving file

Saving to a file turned out to be rather simple as I was only required to empty the file and then run through all the competitor data and use fputs on each value. I had to make sure to remember to put a new line between each data as otherwise I found it saved all data on one line, not fitting the required file layout.

New file

For the ERP I had to make a way for the user to create a new event file. I realised by making a file pointer with the type set to 'w' it will either empty the existing file or create a new one using the given name.

Printing

To print out the data in both the dynamic array and linked list I had to first figure out a way of sorting the data. I remembered that we had used qsort in one of the practicals and thought that would be the smartest approach. I created three different functions, one for sorting competitors by names, another for sorting competitors by score and the third for sorting the track names to show up in alphabetical order. I ran into the problem that the assignment wanted competitors with a score of zero to appear at the bottom but they kept appearing at the top. I looked at how the larger values got set to be at the bottom of the list and realised I had to do the opposite of what the function was meant to do if the value was zero. I did this by implementing a small if statement that would just check if it was zero or not.

After making these I wanted to make each competitor's id line up so I made an equation that would out the size of the largest word and set the distance the id was from the surname accordingly.

Then all I needed to do was loop through the list of competitors and print out the necessary data.

Calculation total

To calculate the total score of each competitor I had to make sure that if they had any missing data their score would be set equal to zero. I checked this by simply having a count for each data item the competitor had and if they had a zero it would remove one from the count. Once the loop reached the end, if the competitor didn't have a full count of tracks then their total score was set to zero.

Robustness

I found that making my program not crash when incorrect data got inputted was a fairly simple task to accomplish, but a rather slow one. This is due to me trying to implement this towards the end of the program, going back and adding it. This meant I wasn't sure if I had found all the places which needed to be updated. If I were to make this program again I would do this whilst writing the code instead of leaving it.

Extended Features

ERP

Creating a function that would only output one competitor at a time and would have to use a different function to cycle competitors didn't sound too hard to begin with, however, after attempting to code this I ran into a problem. I was unsure on how to set the titles of the two different printing methods as they were displayed differently. After some thinking, I discovered that I could put them inside the switch statement meaning they would be printed before the print function even ran. I was also unsure if I would be able to make the table appear the same as the max value was a local variable from my original print function. But I soon realised that I could make this a global variable meaning I wouldn't have to pass it to the function.

CDP

The CDP extended features asked for a way to recover lost data and make it possible to load folders of data.

My thoughts for recovering lost data was that I could have a backup folder that held all the track data and save it when it was edited. I would then give the user an option to load this file the next time the program was run. Unfortunately, I was unable to get this working and didn't implement it into my final design. The main hurdle I faced when trying to do this was I couldn't find a way to save, then load the data in an efficient manner and it would most likely slow the program down too much.

When attempting to implement the loading folders I was unable to figure out what I needed to do to achieve this. I tried to use direct.h to load the folder however I just couldn't get it to function correctly.

Conclusion

In conclusion, I believe my program was a success as I managed to complete all the main features and half the extended features. I also believe my code has been made to be robust enough to handle lots of different data inputs and return the necessary error message if entered.

My C++ approach to the program

When I started planning out my approach to developing these programs I found myself having to create ideas which I wouldn't have necessarily thought of as I am used to using an object oriented programming language. With C++ being an object oriented programming language it would allow me to go about developing my programs very differently.

To begin with I would have a Competitor class which would hold the functions which are involved in setting competitors details and making them. From there I would have a header file with the competitor variables defined and setters and getters.

The main body of the code would be quite similar as I would still make an array of competitors and manage them that way but would also have a different class for printing out the data and sorting it as that would help manage the code. I would also have a different class for the menu as it would allow me to focus my code and not have functions that aren't necessary for the program.

When it comes to the tracks I would most likely continue to use a linked list to store them as it will allow me to add tracks easily to competitors.

If I did recreate this code in C++ I would also make sure to improve on the mistakes I have made in my previous implementation of this program in C, e.g. Make catches for input errors as I go.