CS23820 ASSIGNMENT, 2021 – 2022

“Ceredigion Area Drone Racing Competition”

REPORT

Lukasz Kruszynski – lmk6@aber.ac.uk

My C program overview

Beginning with design choices, I have implemented a linked list data structure to store competitors' records in both, Entrant Registration Program (ERP in the rest of the report) and Competition Day Program (CDP in the rest of the report). I have also implemented a linked list data structure to store each track's record and each lap time in CDP. This implementation allowed me to store number of entries limited only by available memory, furthermore I made sure that each list is removed from memory before entering new set of data or exiting the program. Most of string variables stored in each list's element are allocated in memory according to entry's size, in doing so, less memory is used and there is a room for more flexible implementation of input reader to allow entering entries of any size.

I have assumed that main directory, namely "Drone Race Data", will always be in reach of each program and will always have structure similar to the one presented in example files, so I focused on making file selection very smooth.
For opening 'competitors' file, every folder in directory is displayed and when the user chooses one, list of files to choose from is displayed.
For opening 'Track' file(s), I assumed that each file will be named in the same format (e.g. Track1_1) considering that they are all generated by lap timer device. I also assumed that all track files are going to be read, and according to the discussion on discord channel – results should be possible to read in any order in terms of set of data they contain. Each 'Track' file is automatically read from folder containing the 'competitors' file, that was previously read, so user do not have to enter anything, just await the message whether the data was loaded successfully or not.

When it comes to creating a new event in ERP, if main directory was located, program creates a directory with name starting with "Event[index]_" and ending with provided year in which event took place. Program detects events created in the same year and increments [index] accordingly. After that, new file with name formatted correspondingly ('Competitors_ddmmyyyy') is created inside the new directory. By choosing this approach, I am sure that my program provides a sense of order and strict structure, which makes searching for specific file way easier.

There is a possibility that main directory can be not present, and if such situation occurs, user is prompted for full pathname leading to the 'competitors' file. This solution is present in both, ERP and CDP. If 'Track' files are not present within the 'Event' file, an appropriate message will be displayed to the user. It comes to me that this solution is not the best, it is caused by my poor early design choices which I will discus further later on.

I tried to make a maximum use of linked data structures I have implemented and I think that I did it well in CDP. Each track struct and each competitor struct have their own results linked list which provides good code reuse thanks to my design. Each linked data structure is placed in separate file which assures code modularity.

Both of my programs can handle most of unexpected behaviours and, as far as I tested, most of incorrect inputs coming from user. Each creation of struct pointer is followed by if statement checking if object was actually created, if not, it is assumed that there is no memory to allocate, which user is informed about.
Each user's input is checked and an appropriate message is displayed if something is wrong. Unfortunately my program is not very reliable when it comes to reading file's contents; it is assumed that the user is responsible for choosing a correct file and is prompted to confirm his choice.

Focusing on encountered problems and notable issues, my poor early design choices lead to unexpected behaviours connected with buffer which causes some minor bugs even after repeated use of 'fflush(stdout)'. Fortunately, programs are now stabilized, and whenever encountering any issue, simple use of 'Enter' key fixes the problem. Unfortunately, significant changes to code can cause major bugs affecting any input provided by user.

My C++ approach to the program

If I was to write this assignment in C++, I would certainly make use of its object oriented capabilities. Competitor, Result, Track would naturally be separate classes with most of the fields they had in my C implementation as strucks.

First of all, instead of implementing linked data structures, I would simply use already existing vectors for storing tracks, competitors and results but only within each track. Vectors would save a lot of effort in removing a set of data and iterating through one. Instead of using head of list references, I would use a reference to a vector.

Second of all, I would move all functions to all corresponding classes just to make functions' names simpler to read and operate on. It would add a lot to code's readability.

'string' would replace all char pointers and char arrays, allowing more flexibility for input reading and saving all the hassle with memory allocation. It would also reduce a risk of sudden data alteration (which happened to me many times during C approach) completely.

Each class – Competitor, Result and Track – would have destructors making deleting them a lot easier.

Competitor class would have all the same fields except for the head reference to list of tracks, it would be just 'totalResult' result object. It would also have a 'disqualified' bool variable.

After reading in all the tracks' lap results, a function would traverse through each track's result and try to generate a total result for each competitor. It would set 'totalResult' to NULL if the competitor's number was not present in any result or if the competitor was considered disqualified (present more than once), in which case 'disqualified' flag would be set to true. This flag would come in handy when producing a competition results table – it simply would not print any data of competitors with 'disqualified == true'.