Assignment 2

Due Date

The assignment is due at 3PM Wednesday May 4th 2022. It is worth 20% of your mark and you should complete it as an individual.

Background

The AFL runs Australian Rules football. And the best team in the League is the Carlton football team.

A game in Australian rules comprises four quarters and in each quarter there can be a numerous kicks ("shots") for goal. Goals are worth six points, near misses ("behinds") are worth one, and shots in which the ball doesn't cross the goal line are worth 0. A score of 3 goals and 2 behinds is written as follows: 3 2 20.

You've been asked to track the score throughout a game, present the score at the end of a nominated quarter, draw a kind of histogram illustrating who has the lead and by how much, and finally, report the winning margin. See the section entitled **Program Specification** for details.

The data are organised by quarter (and are presented in the order the shot on goal is kicked). Each shot is identified by its team (a 0 or a 1) and the outcome: goal (6), behind (1), nothing (0). You don't know how many shots on goal will occur during any given game. You should not score shots on goal which don't register a goal or a behind, and all scoring shots should be stored in the order in which they are presented within their quarter.

A text file exists with the required data in it. I will give you code to read this into your program — and it may be different data to the examples shown here.

Task

Based only on the information above (and not what is in the text file or after this point in this document):

a In two–three sentences, explain the functionality required for your model of a **collection of shots on goal** for each of the quarters. Which kind of abstract data type (*binary tree, general tree, array, stack, priority queue, double-ended queue, set, ordered/unordered list,* etc.) is most appropriate for that functionality?

b Which underlying data structure (*array* or *linked-list*) will you use as a basis to model the **collection of shots on goal** for each of the quarters? In two–three sentences, justify your answer.

c In two–three sentences, explain the functionality required for your model of the game (i.e. a **collection of quarters**). Which kind of abstract data type (*binary tree, general tree, array, stack, priority queue, double-ended queue, set, ordered/unordered list,* etc.) is most appropriate for that functionality?

d Which underlying data structure (*array* or *linked-list*) will you use as a basis to model the game (i.e. a **collection of quarters**)? In two–three sentences, justify your answer.

To implement this you would need to define a type called `quarter_format` to implement your answer to (b) above to represent the collection of scores for a quarter, and also another type (called `game_format`) to implement your answer to (d) above to create a collection of `quarter_format`s.

From this, the whole game could be defined as a global variable as follows:

```
game the_game;
```

Each of these two types (`quarter_format` and `game_format`) can be defined as either an array or as a linked-list. Given the linked-list `node` from lectures:

```
struct node_int;
typedef struct node_int *node;

struct node_int
{
     void *data;
     node next;
};
```

you would either define `quarter_format` as a linked-list, i.e.

```
typedef node quarter_format;
```

or you would define it as a dynamic array[1], i.e.

```
typedef score *quarter_format;
```

Similarly, you would either define `game_format` as a linked-list, i.e.

```
typedef node game_format;
```

or as a dynamic array, i.e.

```
typedef quarters *game_format;
```

Within the given program files, as with `node` above, each type (`score`, `quarters`, and `game`) is defined as we've done in lectures. For example, for a `score`, the *header* file, `score.h`, has:

```
struct score_int;
typedef struct score_int *score;
```

together with function declarations for the available functions. The *source* file, `score.c`, has:

```
struct score_int {
     int quarter;
     int team;
     int worth;
};
```

together with function definitions for the available functions on a score.

A Visual Studio project is available on MyLO for you to download and use as a starting point. (If you are not using Visual Studio, then just grab the source files and data file from within the folder.) This starting point comprises the following files:

- `node.h` and `node.c` — the *Node* ADT from lectures as the building blocks for linked lists (should you need them). *These files are complete*;
- `score.h` and `score.c` — the *Score* ADT as specified above. *These files are complete*;
- `quarters.h` and `quarters.c` — the *Quarters* ADT (a collection of scores) as described above. *These files are* incomplete;
- `game.h` and `game.c` — the *Game* ADT (a collection of quarters) as described above. *These files are* incomplete;
- `assig_two122.c` — the file which contains the `main()` function, a function to read the scores in from the text file, some constants, and the `the_game` global variable. *This file is complete.*

[1] It would probably be more intuitive to define it as a static array, e.g.

```
typedef score quarter_format[200];
```

but unfortunately this presents problems when trying to return a `quarter_format` value from a function — C won't allow an array to be returned — and so we'll keep it 'simple' and instead define `quarter_format` as a pointer, and use `malloc()` to create the array and its elements.

e You must complete `quarters.h`, `game.h`, `quarters.c`, and `game.c`.

Start by adding the `quarter_format` and `game_format` type definitions to `quarters.h` and `game.h` (respectively) as indicated above according to your choices in (b) and (d).

Then, complete the missing functionality from `quarters.c`, and `game.c`.

The project also contains the data file. This is just a text file which can be opened and read with most applications. It contains details of scores.

Program specification

First you must obtain the scores from a text file. The data must be stored in appropriate collections. At the top level there will be a collection of quarters (which comprise the game) and for each quarter there should be a collection of scores (stored in the order they are provided, i.e. first-in-first-out).

As a score is read from the file, the details should be checked. If it is not a goal or a behind, then it should not be included in the collection. Otherwise it should be appended to the data for the quarter in which it occurs.

As the data are read in and stored in the collections, the following should occur:

i. The nature of the score (none, behind, or goal), the team with the shot on goal, and the quarter in which it occurs should be displayed.

Each quarter should have a subtitle when it commences.

The game should have a title indicating the names of the two teams.

Once the data have been read in and stored in the collections, the following should occur:

ii. The user should enter a quarter number (1–4). Values below 1 should default to 1 and values above 4 should default to 4.

The scores for each team should then be shown, one line for each, for every quarter from 1 up to the quarter number entered (inclusive). The quarter should be shown as a sub-title, and the activity should have its own title.

iii. A kind of horizontal histogram should be shown which demonstrates the magnitude of the lead for the leading team following every score. The histogram should grow from the centre of the screen, to the left when the first team is leading and to the right when the second team is leading.

Each team's name should be displayed above the graph and the activity should have its own title.

You may assume that the screen has a width of 80 characters and that the lead will never exceed 40 points.

iv. The final (winning) margin should be displayed.

For all of these, the formatting of your output should conform to the example shown below (with 4 entered for the score breakdown for ii).

```
Carlton v Losers
================

Starting Quarter 1

Score received: Quarter 1, a GOAL for Carlton
Score received: Quarter 1, a GOAL for Losers
Score received: Quarter 1, a Behind for Carlton
Score received: Quarter 1, no score for Carlton
Score received: Quarter 1, a GOAL for Carlton
Score received: Quarter 1, no score for Losers
Score received: Quarter 1, no score for Carlton
Score received: Quarter 1, a GOAL for Losers
Score received: Quarter 1, no score for Carlton
Score received: Quarter 1, a Behind for Losers
Score received: Quarter 1, no score for Losers
Score received: Quarter 1, a GOAL for Losers
Score received: Quarter 1, no score for Losers
Score received: Quarter 1, no score for Carlton
Score received: Quarter 1, no score for Carlton
Score received: Quarter 1, no score for Losers
Score received: Quarter 1, a Behind for Losers

Starting Quarter 2

Score received: Quarter 2, no score for Carlton
Score received: Quarter 2, a GOAL for Carlton
Score received: Quarter 2, a Behind for Carlton
Score received: Quarter 2, a GOAL for Carlton
Score received: Quarter 2, no score for Losers
Score received: Quarter 2, no score for Carlton
Score received: Quarter 2, no score for Carlton
Score received: Quarter 2, no score for Losers
Score received: Quarter 2, a Behind for Carlton
Score received: Quarter 2, no score for Losers
Score received: Quarter 2, no score for Losers

Starting Quarter 3

Score received: Quarter 3, no score for Carlton
Score received: Quarter 3, no score for Losers
Score received: Quarter 3, a Behind for Carlton
Score received: Quarter 3, a Behind for Losers
Score received: Quarter 3, a Behind for Losers
Score received: Quarter 3, no score for Carlton
Score received: Quarter 3, a GOAL for Carlton
Score received: Quarter 3, a Behind for Losers
Score received: Quarter 3, a GOAL for Carlton
Score received: Quarter 3, a Behind for Losers
Score received: Quarter 3, a GOAL for Carlton
Score received: Quarter 3, a Behind for Losers
Score received: Quarter 3, no score for Losers
Score received: Quarter 3, no score for Carlton
Score received: Quarter 3, a Behind for Carlton
Score received: Quarter 3, a GOAL for Losers
Score received: Quarter 3, a Behind for Carlton
Score received: Quarter 3, a GOAL for Losers
```

```
Starting Quarter 4

Score received: Quarter 4, no score for Losers
Score received: Quarter 4, a Behind for Losers
Score received: Quarter 4, no score for Carlton
Score received: Quarter 4, no score for Losers
Score received: Quarter 4, a GOAL for Carlton
Score received: Quarter 4, no score for Losers
Score received: Quarter 4, a Behind for Carlton
Score received: Quarter 4, a GOAL for Carlton
Score received: Quarter 4, a Behind for Losers
Score received: Quarter 4, no score for Losers
Score received: Quarter 4, no score for Losers
Score received: Quarter 4, a Behind for Carlton
Score received: Quarter 4, no score for Carlton
Score received: Quarter 4, a Behind for Carlton


Scores up to Selected Quarter
-----------------------------

Enter quarter number #4

Quarter 1
Carlton   2  1 13
Losers    3  2 20

Quarter 2
Carlton   4  3 27
Losers    3  2 20

Quarter 3
Carlton   7  6 48
Losers    5  7 37

Quarter 4
Carlton   9  9 63
Losers    5  9 39


Vertical Worm
-------------

                       Carlton                    Losers
                                  ******

                                       *
                                 *******
                                       *

                                        ******
                                        *******
                                        *

                                   ******
                                  *******
                                 ********
                                  *******
                                   ******
                              ************
                               ***********
                          ****************
                           ***************
                     ********************
                      *******************
                     ********************
                           ***************
```

```
Final margin
------------
The final margin is 24 points.
```

Program Style

The program you write for this assignment must be contained in nine files as follows:

- `node.h` and `node.c` — for nodes of linked lists (if relevant);
- `score.h` and `score.c` — for managing a score;
- `quarters.h` and `quarters.c` — for managing quarters;
- `game.h` and `game.c` — for managing the game;
- `assig_two122.c` — the main algorithm which reads the data, and controls the management of the collections, the calculation of the results, and the display of the information.

No function that you write should be longer than about half a screen-full of code.

Your program should follow the following coding conventions:

- `const` variable identifiers should be used as much as possible, should be written all in upper case and should be declared before all other variables;
- `#defined` symbols should only be used for constant values if `const` is inappropriate, for example the size of an array.
- global variables should be used sparingly with parameter lists used to pass non-global information in and out of functions.
- local variables should only be defined at the beginning of functions and their identifiers should start with a lower-case letter;
- every `if` and `if-else` statement should have a block of code (i.e. collections of lines surrounded by `{` and `}`) for both the `if` part and the `else` part (if used)
- every `do`, `for`, and `while` loop should have a block of code (i.e. `{}`s)
- the keyword `continue` should not be used;
- the keyword `break` should only be used as part of a `switch` statement;
- opening and closing braces of a block should be aligned;
- all code within a block should be aligned and indented 1 tab stop (or 4 spaces) from the braces marking this block;
- commenting:
    - There should be a block of header comment which includes at least
        - file name
        - student name
        - student identity number
        - a statement of the purpose of the program
        - date

- Each variable declaration should be commented
- There should be a comment identifying groups of statements that do various parts of the task
- Comments should describe the strategy of the code and should not simply translate the C into English

What and how to submit

You should submit `quarters.h`, `game.h`, `quarters.c`, and `game.c`. You need not submit any other files. If you wish, you may submit the entire project (as a ZIP file). Do not submit a RAR archive — it *will not* be marked.

You should submit the files to the "Assignment 2" assignment drop-box on *KIT107*'s MyLO site.

Marking scheme

Task/Topic	Maximum mark
Design	
ADTs chosen wisely	4
Data structures correct and justified	4
Program operates as specified	
Quarters initialised ('constructed')	1
Game initialised ('constructed')	2
Quarters added in order	2
Score added —	
• As the first for a quarter	2
• Appropriately displayed	2
• In first-in-first-out order	3
• Discarding 'no scores' (Zeroes)	1
Vertical Worm (Histogram) displayed —	
• With title	1
• With team names	1
• With asterisks (when appropriate)	2
• Growing left and right	2
• Centred on the screen	1
Scores listed for selected quarters —	
• With default values for input	2
• Showing title and subtitles	2
• Showing scores for both teams including names, break-down, and subtotals	2
Final margin shown —	
• Calculated, formatted correctly, and shown	2
Program Style	
Does not unnecessarily repeat tests or have other redundant/confusing code	4
Uses correctly the C naming conventions	4
Alignment of code and use of white space makes code readable	4
Always uses blocks in branch and loop constructs	4
Meaningful identifiers	4
Variables defined at the start of functions only	4
Header comments for the program (author, date etc)	4
Each variable declaration is commented	4
Comments within the code indicate the purpose of sections of code (but DO NOT just duplicate what the code says)	4

The program will be marked out of 72.

Plagiarism and Cheating:

Practical assignments are used in the ICT discipline for students to both reinforce and demonstrate their understanding of material which has been presented in class. They have a role both for assessment and for learning. It is a requirement that work you hand in for assessment is substantially your own.

Working with others

One effective way to grasp principles and concepts is to discuss the issues with your peers and/or friends. You are encouraged to do this. We also encourage you to discuss aspects of practical assignments with others. However, once you have clarified the principles, you must express them in writing or electronically entirely by yourself. In other words you must develop the algorithm to solve the problem and write the program which implements this algorithm alone and with the help of no one else (other than staff). *You can discuss the problem with other students, but not how to solve it.*

Cheating

- Cheating occurs if you claim work as your own when it is substantially the work of someone else.
- Cheating is an offence under the Ordinance of Student Academic Integrity within the University. Furthermore, the ICT profession has ethical standards in which cheating has no place.
- Cheating involves two or more parties.
    - If you allow written work, computer listings, or electronic version of your code to be borrowed or copied by another student you are an equal partner in the act of cheating.
    - You should be careful to ensure that your work is not left in a situation where it may be stolen by others.
- Where there is a reasonable cause to believe that a case of cheating has occurred, this will be brought to the attention of the unit lecturer. If the lecturer considers that there is evidence of cheating, then no marks will be given to any of the students involved. The case will be referred to the Head of the School of ICT for consideration of further action.

Julian Dermoudy, April 5th 2022.