Programming Project

Terry Marris January 2010

3 Design

In the specification we state *what* is to be done. In the design we describe *how* it is to be done.

We base the design on the specification. If the specification is wrong we cannot hope to get the design right.

3.1 Preamble

The language to be used is ANSI C, and the compiler MinGW GCC. The programming environment to be used is the Microsoft Windows Command prompt window, with Notepad as the editor.. The user interface will be primitive, with the expectation that a windows-based interface will be added in another project. To this end the functions will be organised into three main groups.

3.2 Model-View-Controller

Model

The functions appearing in the model section will be entirely concerned with implementing the registering of entries, recording the start and finish times, and producing the results ready for printing. This set of functions will have nothing whatsoever to do with the user interface, but will have everything to do with data structures such as C *structs* and files, and their supporting algorithms.

View

The functions listed in the view section will be responsible for presenting a user's view of the model. For example, it will include functions to display the results.

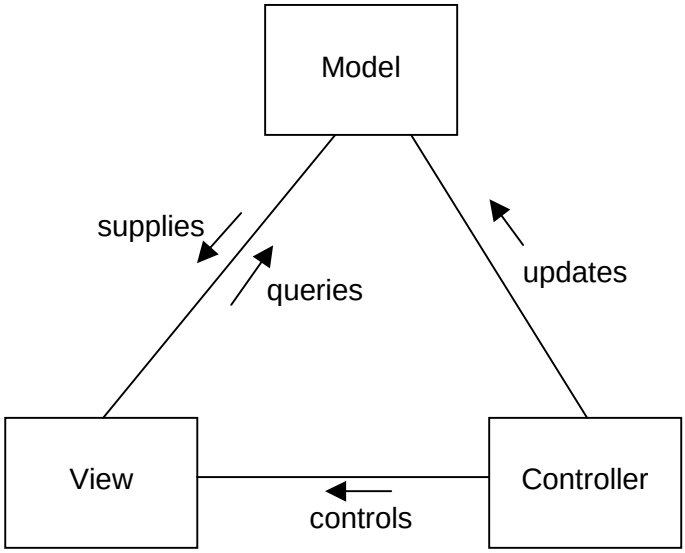
Controller

The functions listed in the controller section will be concerned with getting and processing user input from the keyboard. This section will include prompts and menus.

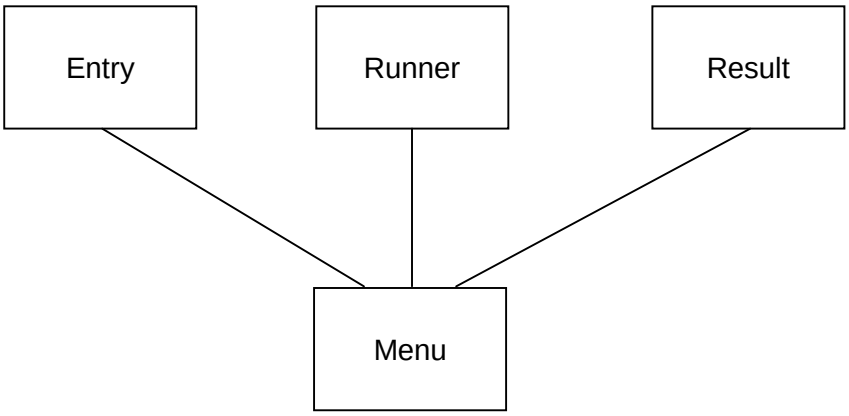
3.3 Modular Structure

A module comprises a set of functions with a coherent purpose. We have two possible models for modular structure.

A



B



We will choose option B because each module is relatively self-contained, you can add a new module, or change an existing one, without affecting other modules, and you can embed model-view-controller architecture in each module.

Entry is concerned with maintaining the entry file, the collection of entries. You can add a new entry, or mark an entry as withdrawn.

Runner is concerned with validating, processing and recording the runners' times. Runner will need to access the entry file to validate the runner number. The race times can be recorded in the results file.

Result is concerned with formatting the contents of the results file ready for printing.

3.4 Bottom Up or Top Down

Some programmers prefer to develop their programs from the bottom up. For example, they will write and test utility functions to be used by other functions and modules before writing anything else. Then each function subsequently written will be based on previous functions. The menu will be the last thing they write and test. A bottom up approach can lead to untidy program structure and unused functions. Further, in the light of experience and familiarity as the program develops, the programmer might have to re-visit previously written modules to provide extra functionality.

Some programmers prefer to develop their programs from the top down. For example, the menu will be the first function they write. They will have the entire structure of the program written, with stubs for each function, then begin to fill in the details. A stub has a *printf()* statement that announces the name of the function, and perhaps a *getchar()* statement that waits for the programmer's acknowledgement when the program is run. Testing of completed functions is delayed with the top down approach.

Bottom up or top down? It does not really matter. I shall primarily use bottom up because that suits the way I think and like to work. Definition before use is my guiding principle.

3.5 Cohesion and Coupling

Cohesion refers to the number of tasks a function performs. A function with the highest level of cohesion performs the smallest task.

Coupling refers to the amount of data passed between functions. Functions communicate with each other via arguments and parameters. A pair of functions that share the smallest amount of data has the least coupling.

We seek to maximise cohesion and to minimise coupling. Why? If an error is found in a program its cause can usually be traced to just one small function, which can then be corrected without affecting other functions.

3.6 Size and Complexity

One of the difficulties encountered in a programming project is the sheer volume of coding. We manage size and complexity be creating small modules, then combining them at link time. It is much easier to plan, write and get right a small unit than it is a large program.

3.7 Error Handling

We have several options when it comes to handling errors. We can ignore them and let the program crash with incomprehensible messages from the operating system. We can print a descriptive error message and halt program execution. We can acknowledge the error by informing the user and carrying on as best we can. In general, we shall adopt both the second and third options.

3.8 File Content and Structure

EntryFile: binary file. Easier than a text file for programmer to amend the contents. But text file contents would be easier for a user to update (and corrupt) with a text editor.

- EntryRecord
  - number: 1..9999
  - name: 25 chars max e.g. Robert Llewelyn-Smith
  - address: 60 chars max e.g. 365 Ocean Boulevard SU99 1AX
  - category: full, half
  - status: current, withdrawn

TimesFile: binary file. Necessary to record starters.

- TimesRecord: records held in race time order
  - number: must be in entry file
  - startTime: e.g. 09:15:00
  - finishTime: e.g. 12:30:45 must have a start time
  - raceTime: from start and finish times e.g. 3:15:45.

ResultsFile: binary file. Automatically maintains a position i.e. record number.

- ResultRecord: in race time order
  - number: must be in times file
  - name: must be in entry file matched with the number
  - raceTime: from times file

NOTE: present the project module by module