Programming Project

Terry Marris January 2010

11 Evaluation

Evaluation involves making informed judgements about a product against clear criteria.

The criteria is essentially what was set out in the specification document. The product is the completed program.

11.1 Evaluation

We show the criteria in italics, and the product and judgement in normal case.

An entry has a competitor number, a race category, a name and an address. A category is either half marathon or full marathon or withdrawn.

This is implemented in *entry.h*. The entry structure defines an entry with a number, a name, a category and a status. The *Category* type defined full and half marathon. Withdrawn is defined in the *Status* type. Criteria met.

We define competitor number as an integer: ...-2, -1, 0, 1, 2, 3, ...

The competitor number, also known as the entry number and the runner number, is implemented as a string i.e. an array of char. This was because arithmetic is not performed on the number. However, the program freely converted the string number to an integer, and conversely, when required, as in *entrybase.c*. Criteria met.

There is a limit to the number of entries we can have.

Each entry is given a unique number, in sequence, when added to the file of entries - see *entrybase.c*.   The least number is 1, the largest is 9999. The user is warned when this upper limit is reached, but that does not prevent this number, 9999, being exceeded. However, entries marked as withdrawn are also included in the number of entries. Criteria not met.

We define entry base, the collection of entries.

entrybase.c implements a file of entry objects. Criteria met.

Given a competitor number we can find the corresponding entry. We cannot have duplicate entries. If two different entries have the same name, they must have different addresses. The order in which entries are held is not significant.

Entries that match given competitor numbers, if any, can be retrieved from the entries file in *entrybase.c*. Duplicate entries, as defined above, cannot be stored. However, two entries such as *Jo King 2 Queens Road* and *Jo King 2 Queens Rd* are treated as different entries in *entrybase.c*. Need to distinguish addresses by postcode and house number. Criteria partly met.

The order in which entries are held is not significant.

The entries are held in order of addition to the file in *entrybase.c*. Criteria met.

Initially, to begin with, there are no entries.

A file with no entries is implemented with *newEntryFile()* in *entrybase.c*. Criteria met.

We add a new entry to the entry base. We require that the new competitor number has not been used before and that the new entry is not already in the entry base.

Each new entry is given a previously unused number by *addEntry()* in entrybase.c Criteria met. But preventing duplicate entries is not entirely successful - se the discussion above. Criteria partly met.

... define race time as an integer number of seconds.

Runner is defined in *runner.h*. Here runner's race time is defined as an integer. Criteria met.

We need a function to calculate the race time from race start time and runners finish time.

raceTime(), in *runners.h,* converts a start time and a finish time into an integer number of seconds. Criteria met.

We cannot have a negative race time. A race time of zero means either the competitor did not start, or started but did not finish. We are not concerned with non-starters and non-finishers.

In *raceTime()* in *runner.c*, a race time cannot be less than MINRACETIME if the runner's category is half marathon, and cannot be less than 2 x MINRACETIME if the runner's category is full marathon, where MINRACETIME is defined to be 3000 (seconds) i.e. 50 minutes. If a runner started but did not finish, their race time remains set at zero - see *newRunner()* in *runner.c*. Criteria met.

A runner has a start time and a finish time. Only runners that have started can have a finish time.

Every new runner is given a default start and finish time - see *newRunner()* in *runner.c*. In *setFinishTime()* in *runner.h*, a runner cannot be given an actual finish time if they did not start. Criteria met.

Initially, there are no runners.

newRunnerFile() in *runnerbase.c* creates a new empty runners file. Criteria met.

When a runner crosses the starting mat on the start line their start time is set as the race start time. They are included in the set of those who have started, but only if their competitor number is for a genuine entry.

Just one start time is set for all runners by *startTime()* in *marathonmenu.c*. This start time is used by *finishTime()*, also in *marathonmenu.c*, to set a runners start and finish time. A runner is only given an actual start time if they are given an actual finish time. Criteria partly met. However, *addRunner()* in *runnerbase.c* prevents those with an invalid runner number from being included. A valid runner number is one that is in the entrybase file. However, an entry that has been withdrawn can still be recorded as a runner with start and finish times. Should an entry that has withdrawn be allowed a race time other than zero? Criteria met, but perhaps the criteria need strengthening.

When a runner crosses the timing mat on the finishing line their finish time is recorded. Only those who have started may have a finish time. We require that the finish time be after the start time.

raceTime() in *runner.c* requires that an actual race time be more than MINRACETIME. Every runner that has a stating time set has their status changed from dns (did not start) to dnf (did not finish. Only those runners with status dnf can be assigned an actual finish time - see *setStartTime()* and *setFinishTime()* in *runner.c*. Criteria met.

The current (2010) world record for a half marathon is 58.23 held by Zersenay Tadese. So we require the difference between the start time and the finish time to be greater than, say, 40 minutes, that is 40 x 60 seconds or 2400 seconds. That is for a half marathon.

The current (2010) world record for a full marathon is 2:03:59 held by Haile Gebreselassie. So we require the start and finish times to be greater than, say, 100 minutes, that is 100 x 60 seconds, or 6000 seconds.

runner.h defined MINRACETIME as 3000 seconds, that is 50 minutes. This weakens the specification for half marathon results, see *raceTime()* in *runner.c*. Criteria met.

A result has a competitor number, a category, a name and a time. The number must be for a finisher. The category must be either half marathon or full marathon. The time is the race time and is derived from the runner's start and finish times.

Result with a number, name, race time and category is defined in *result.h*. Since results are derives from runnerbase, and runnerbase contains runners with valid numbers, start and finish times, the criteria is met.

Results are partitioned by category. A result is either for the half marathon or the full marathon. Further, the results are in finishing position order as determined by race time order. We require that there are no duplicates in the results: no runner may appear twice in the results and each runner has just one position.

halfMarathon() and fullMarathon() in marathonmenu.c partitions the results into half marathon and full marathon categories. The results are derived directly from runnerbase, which contains runners in race time order - see *addRunner()* in *runnerbase.c*. Further, since addRunner() also prevents duplicates being stored, there can be no duplicates in the results, and each runner has just one result. Criteria met.

In general, the implementation met the requirements of the specification.

11.2 Improvements and Suggestions for Further Work

1. file io be in one place, perhaps in its own module. Minimises repetition of file io error handling and localises file io errors if they occur

2. collect all the error-reporting functions into one module - currently they are repeated in each module

3. more testing - looking for errors - should be much more exhaustive since the testing is not up to professional standards or distinction student standards

4. split address into street name and postcode. Why? Because then two entries are identical if their names are the same and their house numbers are the same and their postcodes are the same - different variations of the same address e.g. 2 Queens Road and 2 Queens Rd could then be regarded as identical (not different as they are now)

5. organise functions into two groups: interface and application. Makes it easier to implement a Windows interface

6. replace the console line io interface with a Windows interface

7. provide stricter validation for times input e.g. restrict hours input to be within 0..23.

8. when an attempt is made to record results for a runner with a number that is not in the entries file, three error messages are displayed - could reduce this to one error message

9. to maintain a functional style of programming *malloc()* is used extensively to physically allocate memory, bit it is never freed. There could come a time when the program runs out of memory. A solution would be to use the Boehm-Weiser garbage collector to automatically free memory when it is no longer being referred to

10. extend the program to include: management of resources such as marshals, managing donations

11. produce results as a file of text that can be referred to any time after the event has finished. Currently, the results files are in binary format and there is no program to access historic results. Text files are relatively easy to post on the Internet.