Project 2017

Theory of Algorithms

Due: 23/04/2017

The following document contains the instructions for Project 2017 for Theory of Algorithms. The project will be worth 30% of your mark for this module. You are required to develop a program that solves the Countdown [1] Numbers game, written in the Racket [2] functional programming language. You must use GitHub to manage the development of your project.

Rules of the game

In the Countdown Numbers game contestants are given six random numbers and a target number. The target number is a randomly generated three digit integer between 101 and 999 inclusive. The six random numbers are selected from the following list of numbers:

$$[1, 1, 2, 2, 3, 3, 4, 4, 5, 5, 6, 6, 7, 7, 8, 8, 9, 9, 10, 10, 25, 50, 75, 100]$$

Contestants must use the four basic operations – add, subtract, multiply and divide – with the six random numbers to calculate the target number (if possible). They don't have to use all six of the numbers, however each of the six random numbers can only be used once. If the same number appears twice in the list of six then it may be used twice. At each intermediate step of the calculation, negative numbers and fractions are not allowed. For instance, you can't subtract a 7 from a 2, as that would give -5, and likewise you can't divide 2 by 7 as that gives a fraction.

As an example, suppose the target number is 424 and the six random numbers are $[100, 25, 10, 2, 2, 1]$. Then the following calculation works: $(100 \times (2+2))+25-1$. The steps in this calculation are broken down into individual operations as follows. First add 2 and 2, to give 4. Then multiply this by 100, to give 400. Then add 25 to give 425. Finally subtract 1 to give 424.

Minimum Viable Project

The minimum standard for this project is a GitHub repository containing a single Racket script, a gitginore file, a README, and a LICENSE. The Racket script must be well-commented and contain a function that accepts two arguments: the target number and the list of six random numbers. The function must return a solution, if one exists, within the rules of the game. The following is an example of the expected behaviour of the function, where the function is called `solvecount`.

```
> (solvecount 424 (list 100 25 10 2 2 1))
'(- (+ (* 100 (+ 2 2)) 25) 1)
```

A better project will be organised, with extensive comments and explanations. The Racket script (or scripts) will be easy to read and understand. The function will output all possible solutions, or a message indicating that no solution is possible, and it will not accept incorrect arguments. The README will explain how to run the script and detail how the algorithm works.

Submissions

GitHub must be used to manage the development of the software. Your GitHub repository will form the main submission of the project. You must submit the URL of your GitHub repository for this project using the link on the course Moodle page before the deadline. You can do this at any time, as the last commit before the deadline will be used as your submission for this project.

Any submission that does not have a full and incremental git history with informative commit messages over the course of the project work will be accorded a proportionate mark. It is expected that your repository will have at least tens of commits, with each commit relating to a reasonably small unit of work. In the last week of term, or at any other time, you may be asked by the lecturer to explain the contents of your git repository. While it is encouraged that students will engage in peer learning, all software that is contained in your submission must have been written by you yourself. You can show this by having a long incremental commit history and by being able to explain your code.

Marking scheme

The following marking scheme will be used to mark the project. Students should note, however, that in certain circumstances the examiner's overall

impression of the project may influence marks in each individual component.

40%	**Programming**	Well-written code; logical file structure.
40%	**Explanation**	Extensive comments; excellent README.
20%	**Management**	Incremental work; informative commit messages.

Advice for students

- Your git log history should be extensive. A reasonable unit of work for a single commit is a small function, or a handful of comments, or a small change that fixes a bug. If you are well organised you will find it easier to determine the size of a reasonable commit, and it will show in your git history.

- Using code from outside sources is sometimes acceptable – so long as the code is licensed to permit this, you clearly reference the source, and the overall project is substantially your own work. Using a source that does not meet these three conditions could jeopardise your mark.

- You must be able to explain your project during it, and after it. Bear this in mind when you are writing your README. If you had trouble understanding something in the first place, you will likely hve trouble explaining it a couple of weeks later. Write a short explanation of it in your README, so that you can jog your memory at a later date.

- Everyone is suseptable to procrastination and disorganisation. You are expected to be aware of this and take reasonable measures to avoid them. The best way to do this is to draw up an initial straight-forward project plan and keep it updated. You can show the examiner that you have done this in a number of ways. The easiest is to summarise the project plan in your README. Another way is to use a to-do list like GitHub Issues.

- Students have problems with projects from time to time. Some of these are unavoidable, such as external factors relating to family issues or illness. In such cases allowances can sometimes be made. Other problems are preventable, such as missing the submission deadline because you are having internet connectivity issues five minutes before. Students should be able to show that up until an issue arose they had completed a reasonable and proportionate amount of work, and took reasonable steps to avoid the issue.

- Go easy on yourself – this is one project in one module. It will not define you or your life. A higher overall course mark should not be determined by a single project, but rather your performance in all your work in all your modules. Here, you are just trying to demonstrate to yourself, to the examiners, and to prospective future employers, that you can take a reasonably straight-forward problem and solve it within a few weeks.

References

[1] Channel Four Television Corporation. Countdown. `http://www.channel4.com/programmes/countdown`.

[2] PLT Inc. Racket – solve problems, make languages. https://racket-lang.org/.