New arrangements for COVID-19

As you are aware, there will be no in-house written exam for this module in Summer 2020. This is due to the national restrictions arising from the COVID-19 pandemic. We are mindful that you may have children at home, have sick or quarantining relatives, or may be in that situation yourself. We know that some have difficulties accessing the internet.

None the less, we want to provide you the opportunity to complete this module in the normal time frame. We have put in place an alternative assessment, involving a re-scoping of the project you worked on during the semester. The re-scoped project mark will now be used for the entire module.

By re-scoping, we mean that there are now four additional requirements for the project. Note that the work you have previously completed is still valid and will be assessed. The new requirements are as follows.

1. You must immediately make your GitHub repository private and add `ian.mcloughlin@gmit.ie` as a collaborator.

2. You must include testing to ensure that the algorithms in your code are correct. As described in previous lecture videos, you might use the `assert` keyword in Python for this, or you might elect to use a package such as `unittest` in the Python Standard Library.

3. Your program must accept command line arguments. At the very least it should accept the `--help` command line argument, which should print to the screen help as to how a user can run your program. I recommend you add at least one or two other command line arguments, along the lines of the typical ones used in standard command line programs. You might use the `argparse` package for this purpose.

4. You must add an `overview.md` file to your repository. This Markdown document should contain an explanation of your project work, pitched at students in the year below you. I recommend you consult GitHub's documentation on Markdown as to how to incorporate images, lists, and links to your document. I recommend your document, when printed at normal size is at least six pages long in total. Please include at least the following items in it.

   Introduction An introduction to your repository and the code. Describe what is contained in the repository and what the code does.

   Run You should explain how to download and run your code, including instructions of how to install Python. Remember, this is to be pitched at students in the year below you.

Test Explain how to run the tests incorporated in your code.

Algorithm Give an overview and explanation of the main algorithm(s) in your code. A well-thought out diagram is a great aide here. You may use a referenced diagram from online or, better yet, create your own. Make sure any image displays correctly in GitHub.

References Provide a list of references used in your project and any further materials readers might find interesting. The references should not just be a list of links. There should be a short explanation of why each reference is relevant to your document.

I appreciate it may take time to adjust to the ideas in this alternative assessment, and to this end the deadline has been pushed out to a last commit on or before May 18th, 2020. The original 50% will be marked as before, although you may use this opportunity to enhance that work by the new deadline. The following marking scheme will apply for the extra 50%.

15%	**Arguments**	Useful command line arguments informed by common command line tools in a typical Linux environment and as depicted by `--help`.
15%	**Testing**	Concise, well-documented, and effective tests of the algorithms.
20%	**Report**	Clear report explaining the technical aspects of the project. Language pitched at students in the year below. Typo-free and appropriately referenced.

Finally, we will work with you wherever possible to provide flexibility in achieving a fair and valid mark. It is always possible to defer assessment until the next sitting if there is documentary evidence that this is necessary. Unfortunately, it is not clear that the situation will have changed when the next sitting comes around in August. If possible, I recommend you try your best to complete this assessment now.

The rest of this document remains unchanged.

Project 2020

Graph Theory

Due: last commit on or before May 18th, 2020

This document contains the instructions for Project 2020 for Graph Theory. It involves writing a program in Python [2] to execute regular expressions on strings using an algorithm known as Thompson's construction, named after the well-known computer scientist Ken Thompson. You will be required to demonstrate your program to the lecturer towards the end of the semester.

Please note that all students are bound by the Quality Assurance Framework [4] at GMIT which includes the Code of Student Conduct and the Policy on Plagiarism. The onus is on the student to ensure they do not, even inadvertently, break the rules. A clean and comprehensive git [1] history (see below) is the best way to demonstrate that your submission is your own work. It is, however, expected that you draw on works that are not your own and you should systematically reference those works to enhance your submission.

Context

The following project concerns a real-world problem that you could be asked to solve in industry. You are not expected to know how to do it off the top of your head. Rather, it is expected that you will research and investigate possible ways to tackle the problem, and then come up with your own solution based on those. A quick search for solutions online will convince you that many people have written solutions to the problem already, in many different programming languages, and many of those are not experienced software developers. Note that a series of videos will be provided to students on the course page to help them with the project.

Students will also be required to demonstrate their submission to the lecturer towards the end of the semester during their normal timetabled lecture and lab times. Failure to do this will result in no mark being recorded for this project. In exceptional circumstances and by agreement with the lecturer an individual student may be allowed to present at another time,

such as during exam week. To request this facility, please email the lecturer (`ian.mcloughlin@gmit.ie`) as soon as you are aware of the exceptional circumstance you find yourself in and include documentation to support your case. Agreement is at the discretion of the lecturer and in accordance with GMIT's policies.

Problem statement

You must write a program in the Python programming language [2] that can build a non-deterministic finite automaton (NFA) from a regular expression, and can use the NFA to check if the regular expression matches any given string of text. You must write the program from scratch and cannot use the `re` package from the Python standard library nor any other external library.

A regular expression is a string containing a series of characters, some of which may have a special meaning. For example, the three characters `.`, `|`, and $*$ have the special meanings *concatenate*, *or*, and *Kleene star* respectively. For example, the regular expression `0.1` means a 0 followed by a 1, `0|1` means a 0 or a 1, and `1*` means any number of 1's. These special characters must be used in your submission.

Other special characters you might consider allowing as input are brackets `()` which can be used for grouping, `+` which means *at least one of*, and `?` which means *zero or one of*. You might also decide to remove the concatenation character, so that 1.0 becomes 10, with the concatenation implicit. You may initially restrict the non-special characters your program works with to 0 and 1. However, you should at least attempt to expand these to all the digits, and the characters a to z, and A to Z.

Minimum Viable Project

The minimum standard for this project is a git [1] repository (see below for details) containing a single Python script that can execute regular expressions against strings over the alphabet $0, 1$ and with the special characters `.`, `|`, and $*$. The README should clearly document how to run and test your program. It should also explain how your program works, and how you wrote it.

You are expected to be able to break this project into several smaller tasks that are easier to solve, and to plug these together after they have been completed. You might do that for this project as follows:

1. Parse the regular expression from infix to postfix notation.
2. Build a series of small NFA's for parts of the regular expression.

3. Use the smaller NFA's to create the overall NFA.

4. Implement the matching algorithm using the NFA.

A better project will be well organised and contain detailed explanations. The architecture of the system will be well conceived, and examples of running the program will be provided.

Submissions

The git software package [1] must be used to manage the development of your project and your git repository must be synced with an online provider such as GitHub [3]. Your repository will form the main submission of the project and will be submitted by providing the URL for your repository via the Moodle page. You can submit the URL at any time, the earlier the better, as the last commit before the deadline will be used as your final submission for the project.

Any submission that does not have a full and incremental git history with informative commit messages over the course of the project timeline 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, any unreferenced documentation and software that is contained in your submission must have been written by you. You can show this by having a long incremental commit history and by being able to explain your code.

Marking scheme

This project will be worth 50% of your mark for this module. The following marking scheme will be used to mark the project out of 100%. Students should note, however, that in certain circumstances the examiner's overall impression of the project may influence marks in each individual component.

25%	**Research**	Investigation of problem and possible solutions.
25%	**Development**	Clear architecture and well-written code.
25%	**Consistency**	Good planning and pragmatic attitude to work.
25%	**Documentation**	Detailed descriptions and explanations.

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 information, code and data from outside sources is sometimes acceptable – so long as it 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 have trouble explaining it a couple of weeks later. Write a short explanation of it in your README, so that you can jog your memory later.

- Everyone is susceptible 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 several 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 it. 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 preventable issues.

- 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] Software Freedom Conservancy. Git.
`https://git-scm.com/`.

[2] Python Software Foundation. The python programming language.
`https://python.org/`.

[3] Inc. GitHub. Github.
`https://github.com/`.

[4] GMIT. Quality assurance framework.
`https://www.gmit.ie/general/quality-assurance-framework`.