Term Project

doc rev. 1.0 (7 March '18)

1 General Instructions

This handout is the specification for your individual term project assignment. The project builds on learning from the labs and give the student the opportunity to demonstrate their mastery of the modules learning outcomes as well as to stretch their skills and knowledge beyond the core algorithms and data structures introduced in the module. The project will consist of a report and working problem solution implemented in Python.

1.1 Developing your project

This project is designed to practice, develop and assess your programming skills and object oriented programming competence. Reflect upon the learning outcomes and that upon completion of the module the student should be able to:
1. Understand how to determine the amount of resources (such as time and storage) necessary to execute a particular algorithm (algorithm analysis). 2. Understand the structure, nature and use of fundamental data structures including, Arrays, Linked Lists, Stacks, Queues, Trees and Dictionaries. 3. Implement the data structures in Python. 4. Understand the object-oriented programming constructs needed to encode a data structure and its access algorithms. 5. Design programs using these constructs to solve large problems. 6. Successfully write, compile, debug and run programs using these constructs.
In developing your project, you are encouraged to interact with your classmates during and outside of the formal laboratory sessions. Through discussion, you can develop your understanding and awareness of techniques and methods for designing, developing and implementing your project. You are encouraged to carry out self-guided research and reading in addition to the suggested texts and web learning tools to augment your programming skills and knowledge surrounding the project problem. This will differentiate your solution and enhance your learning experience.
Collaboration is talking or discussing strategies. It is not sharing, looking at or using another person's code. You should not use code you find online (snippets of 4–5 lines explaining a general problem from stackexchange is ok – downloading a github project isn't!). Using sources to research better ways of solving problems is encouraged. Look at books, websites or research papers to help design your algorithm. If you do remember to cite any source you use in your report.
Use the discussion forum on moodle for questions, answers or discussions. Interaction and discussion between students is encouraged. I will regularly monitor it and try to help with queries. Feel free to help classmates out if you have already encountered an issue and worked out the answer to problems they might be querying.

1.2 Plagiarism

School and University plagiarism policies should be read and understood by all students. There are serious consequences for confirmed instances of plagiarism. School Policy: `www.cs.ucd.ie/sites/default/files/cs-plagiarism-policy_august2017.pdf`.
The school policies and procedures should be read in conjunction with the UCD policy on Plagiarism and Academic Integrity: `www.ucd.ie/registry/academicsecretariat/docs/plagiarism_po.pdf`
as well as the relevant sections (especially 6.2 and 6.3) of the UCD Student Code: `www.ucd.ie/registry/academicsecretariat/docs/student_code.pdf`
Your project should be all your own work. As a quick reminder and non-exhaustive list from a programming perspective plagiarism includes:

Sharing code or using someone else's code

Reworking other students or work found on the web and changing variable names

Copying another students design, report, tests

Getting someone else to write code for you

If there is any suspicion of code being shared or copied, students submission may be directly referred to the school plagiarism committee or required to attend an oral defence of their assignment to discuss their submissions. Students should be able to explain their coding decisions and validate their understanding of their project submission.

2 Project Submission

- Submit via `csmoodle.ucd.ie`.
- Multiple submissions possible. Only last one is graded.

- Link WILL stay open after deadline.
- Late submissions be penalised by an absolute mark of 10% per day late (so after 10 days link will close).
- Marking Scheme below – remember to address each item to maximise marks
- Zip up your project directory and submit.

2.1 Deliverables

Project document, python code for program and tests, supporting files.

2.2 Marking Scheme

2.2.1 40% Project Implementation

This covers items such as:

1. Project implements the specification and program has the ability to correctly use the data provided to perform the specified functionality
2. Code quality and style: good programming practices displayed (e.g. using suitable packages, naming conventions, resource management, appropriate use of comments)
3. Error and Exception Handling: Program is robust to unexpected data inputs and handles errors in an appropriate way; program makes appropriate use of Python exception handling in error management
4. Tests: Appropriate tests to exercise the code at a system and unit level
5. Jazz: Project goes beyond the minimum specifications and demonstrates students ability to use a variety of programming skills beyond the basic minimum defined scope of the project.

2.2.2 40% Project Design

This will be assessed through the code and documentation. It covers items such as:

1. Identifying and applying appropriate algorithms and data structures
2. Justification of trade-offs between correctness, speed, and efficiency
3. Consideration of other factors e.g. security, robustness, clarity, maintainability
4. Applying OOP concepts to solution (e.g. selection of suitable classes, modules, interfaces between classes, potential for code reuse)
5. Class Design (e.g. appropriate methods, attributes, attribute visibility)

2.2.3 20% Documentation Quality

This is the opportunity to demonstrate your understanding of how algorithms and data structures influence the entire software development life cycle from design through implementation and testing. The document does not need to be long or verbose but should contain information about the program assumptions, inputs and outputs. The document will outline the design of the solution should address the key algorithms and data structures with details of the modules, classes and/or function used and data files used or created. It should address the performance of the key algorithm and their complexity. The tests strategy should also be covered.
High marks in documentation will be achieved through:

1. accuracy (matching the implementation)
2. clarity (well structured, written and without typographical, spelling, grammar mistakes)
3. succinctness (concise but understandable)
4. completeness (covering the required sections)

3 Guidelines

3.1 Specification

Imagine you work for a small software company that has won the tender to deliver a fuel management software solution to a large Airline company. The airline is entering the European air freight cargo business and wants a software tool to manage their fuel purchasing strategy.
Each week, an aircraft will fly up to six trips. The airplane will start and end in the same home airport each week. The company has a number of different aircrafts that have different fuel capacities and the cost of fuel varies from airport to airport. Given a list of 5 airports (including to the home airport) that a given plane needs to visit in a week, the most economic route must be found.

- The distance between airports is calculated as the great circle distance between them

- The cost of fuel is assumed to be 1 euro per 1 litre at Airports where the currency is Euros
- The cost in of fuel in airports where the local currency is not euros is assumed to be the exchange rate from the local currency to euro. e.g. in you travel from London to Dublin and the exchange rate is GBP£1 = €1.4029 and you purchase 1000 litres of fuel it will cost €1402.
- The fuel capacity of all aircraft are given in litres with an assumed fuel capacity of 1 litre per km flown

3.2 Product Features

The program with take file inputs and give file outputs using CSV format. It will provide a command line interface to allow input of input/output filenames.
Break the problem down and prioritise getting getting small contained features completed. Don't try and code them all at once. For example, below is a suggested order of completion. By breaking the project up into these steps like this you would always have a minimum viable product set of features that you can test, document and submit even if you don't complete all of the features.
Basic Features:

1. Data structures for holding the information provided, i.e. implement Classes to load into memory Airport, Currency Rate and Country Currency informations
2. Code to calculate the distance between airports (You have this from the earlier lab
3. Code to price the cost of a single itinerary leg, e.g. DUB → LHR
4. A data structure to store an itinerary (a multi-leg trip) that can be either a set for inputs or a sequence for the best route
5. Calculate the distance of each leg of a given route (undirected graph with weighted edges)
6. Calculate the cost of each leg of a given route (directed graph with weighted edges)
7. Code to permute all of the possible itineraries
8. Code to price an given itinerary
9. Calculate and display the cost of each leg of a given route
10. Calculate the best round trip route (cheapest cost) for an itinerary returning to the home airport using each airport once (5 airport solution)
11. Read in a `testroutes.csv` with airports and equipment as input and return a `bestroutes.csv` with the optimal route and cost of the route.
12. Basic unit tests

Advanced Features:

1. Add support to calculate the best route from Airport 1 to Airport 5 visiting airports 2,3, and 4. (Dijkstra)
2. Add support to ensure that the aircraft has fuel capacity to fly a given route, i.e. if there is any single leg that cannot be flown mark the route as impossible to complete with the given equipment. You'll need an Aircraft class that has aircraft ranges data.
3. Add visualisation of routes using graphs or maps
4. Add caching optimisations
5. Performance tests

3.3 Code style and efficiency

The program code should be created in a logical and efficient manner. This includes: efficient use of data structures (think about whether they are being modified or are static – do they need to be adaptable or fast access), algorithms (are you implementing them yourself or using libraries– would your own implementation be less complex/more efficient/optimised to the task), code re-use in functions and classes (breaking code up into logical chunks that keep code-blocks and files reasonable length i.e. no 1000 line files!), efficient storage and searching of data (think dictionaries, caching).

3.4 Class Model Structure

Create classes that model the problem and abstract the main program from the underlying data structure. For example, the mainline code shouldn't need to understand anything about parsing a CSV file or calculating great circle distance or currency rates calculation. This means it should be apparent that you have you created classes that model the problem and assigned useful attributes and methods to the classes. The solution should define classes for holding data that will be used in the problem, e.g. there will need

to be a class for Currency and Airport? (HINT: Yes, there will!)

3.5 Errors and Exception Handling

How robust is your program to errors? Have you implemented error checking and exception handling, especially to handle unexpected inputs or data?
e.g. do you have error handling to identify and warn if the wrong number of airports are contained in the input (e.g. will it crash if zero or one airport is entered?), or one or more of the airport code is not a valid airport in your list?

3.6 Comments

Is your code well commented? Are the variable names and methods/functions meaningful?

3.7 Testing

Test code to run on and demo the program and unit tests to demonstrate the subcomponents of your solution.

3.8 Jazz

Anything I didn't ask for but is a useful feature for the product, e.g. a GUI interface

3.9 Documentation

1. No cover page – include your name and student number should be in the header of each page.
2. Main body of document should be 5 pages.
3. You can include an appendix with any figures and tables (e.g. any class diagrams, example graphs, unit test results etc.)

1. Function Specification: Describe what does your program does. (max. 1 page)
2. Design: How your solution has been designed to address the requirements described in the previous functional specification. This can be a series of short sections that describe the different challenges you faced and your solutions. For instance, take a component (e.g. Airport Atlas or Routing Algorithm), describe the key algorithm(s) and data structure(s) and the difficulty you faced and your solution. These sections can be short – the objective here is to show how you crafted the solutions with the tools you have learnt so far and any assumptions you had to make.
   - Class diagram
   - Block diagram/Flow chart/pseudo code of key algorithms
   - Text description to explain algorithms: what did you implement yourself, what optimisations did you include, estimates of running time/complexity.
3. Testing: How have you tested your program. Have you got a suite of tests that exercise the key components? (max. 0.5 page)

Document what you have done well and the features will be marked based on what you describe as completed. If you want to put features in the design document that are not implemented (i.e. explaining how you would do it), that's find, but be sure to make it clear which parts of the design are not in the implementation.

3.10 Supporting materials

CSV files for airports, countries, currencies and aircraft
Example Airport Data:

```
596,Cork,Cork,Ireland,ORK,EICK,51.841269,-8.491111,502,0,E,Europe/Dublin
597,Galway,Galway,Ireland,GWY,EICM,53.300175,-8.941592,81,0,E,Europe/Dublin
599,Dublin,Dublin,Ireland,DUB,EIDW,53.421333,-6.270075,242,0,E,Europe/Dublin
```

Example Currency Data:

```
name is column 0
currency code 14
Iraq,Iraq,IQ,IRQ,368,IRQ,iq,IQ,IRQ,964,IRQ,IZ,118,IRQ,IQD,IRAQ,3,Iraqi Dinar,368,Yes
Ireland,Irlande,IE,IRL,372,IRL,ie,IE,IRL,353,IRL,EI,119,IRL,EUR,IRELAND,2,Euro,978,Yes
```

Example Currency Rates:

```
currencyName, currencyCode, toEuroRate, fromEuroRate
Australian Dollar,AUD,0.7253,1.3791
Euro,EUR,1,1
US Dollar,USD,0.9488,1.0541
```

Example Aircraft Data:

```
code,type,manufacturer,range
A319,jet,Airbus,3750
A320,jet,Airbus,12000
747,jet,Boeing,9800
MD11,jet,McDonall Douglas,12670
```

Example Test Route Data (with optional aircraft type):

```
DUB,LHR,SYD,JFK,AAL,777
SNN,ORK,MAN,CDG,SIN,A330
BOS,DFW,ORD,SFO,ATL,737
```

Example Best Routes:

Origin	Dest 1	Dest 2	Dest 3	Dest 4	Home	Cost
DUB	LHR	AAL	SYD	JFK	DUB	20356
SNN	ORK	CDG	SIN	MAN	SNN	19782
BOS	ORD	SFO	DFW	ATL	BOS	8924