U08028: Software Development in C/C++

Deadline: **9:00am, Wednesday, Week 10 for the report.**
With demonstration during the practical session in week 9.

Dr. A. Jabir and Dr. M. Rolf
Department of Computing and Communication Technologies
Oxford Brookes University

What you have to do
You are required to *design, implement*, *test* and *document* a software system to satisfy the (fictitious) requirement as given in the attached requirements "World Distances." This is an individual coursework which we hope that you will enjoy.

Module assessment weighting
50%

Submission
- Students should submit their report to Turnitin on U08028 Moodle and hand in a hard-copy to the module leader. The report should include all the components listed below and you should include all source code and configuration setup (e.g. makefiles).

Elements of your coursework documentation
Your report should include outcomes of *all* the following where the relative assessment weight of the outcome is indicated.

1) **Design**: Study the requirements and consider how you will implement them. Your report will need to include a description of the design of your system, and how it satisfies the provided requirements. You must not use excerpts or "narrations" of source code to describe your design. You should resolve any ambiguity in the specification based on your own expert judgement, and include a description of the decisions you took and the rationale behind these in your report.

[20%]

2) **Planning your testing**: Devise a test plan, stating what cases you will test, and with test data showing precisely what values you will use for each of your test cases. Show the expected results from applying your test data to the final system.

[10%]

3) **Implementation**: Implement your solution in C++. Include all source texts in your document. Note that your solution **must** make use of Object Oriented programming; even if your program works fine, you will lose marks for not using Object Oriented techniques.

[40%]

4) **System test**: Test your system using the test plan and test data that you devised earlier **and show the actual results of the testing**. If you discover any bugs and manage to fix them, describe how you fixed them, and show the actual

testing results before and after the fix. You do not lose any marks for finding bugs that you later fix. If you "update" your test results to make it appear that the bug never existed then you will be penalised.

[20%]

5) **Summary**: Write a few paragraphs describing your experience in undertaking this project and commenting honestly on the degree of success you achieved. If you were unable to fulfil all the requirements of the desired system, try at least to achieve a system with some worthwhile subset of the required features. Note: you do not lose marks if, during your implementation, you find that you need to change your design plans (this happens often in real software projects). There is no need to alter your design section to "hide" the fact that a change was made; describe the change and the reason for it in the summary.

[10%]

Requirements Specification: World Distances

A program is required to store the names of world places (cities) and their co-ordinates (latitude and longitude), and to calculate the distance between any two places. It should be possible to:

- Perform the usual housekeeping operations on the database such as, *add, delete* and *modify* a world place and its co-ordinates;
- *Find* the distance between any two places, identified by name; if necessary, replace or update a city and its co-ordinates.
- *Display* the names of all places and their co-ordinates.
- Any other feature, which may make the tool more practical or interesting, e.g. features for sorting the cities, dealing with cities having the same name, but located in different countries (e.g. Oxford in England, and Oxford in the USA etc.). During the demonstration, we will note the additional features incorporated in your program.

The program should be able to store at least 100 cities (you may set this to a lower value, e.g. 10, for initial testing). The program should store the list of names and locations permanently on disk. They should be automatically loaded when the program begins, and automatically saved when the user quits.

As a minimum the program should provide a text-based user interface. However, focus should be given on the demonstration of the fully working version of the program to meet the functional requirements. Therefore the quality of the user interface should not be prioritised.

Alpha Sting (attempt only when you have completed all other parts of the exercise to a high standard):
Modify the program to store a number of places limited only by available memory. The program should use only the amount of memory required to store the number of places actually entered. In this regard the efficiency of the considered method should be taken into account. For these features, you **must** construct the data-structures on your own, based around the object oriented aspects of C++, as opposed to relying on pre existing library routines. [Hint: Think about dynamic memory allocation based on linked lists, trees, hash tables, etc.] . (There are many text books on data-structures which you may find useful for this, e.g. Data Structures and Program Design in C++, by Robert L Kruse and Alexander J. Ryba, etc.)

Technical notes

From Astronomical Algorithms (Second edition) Jean Meeus, Willman-Bell Inc. ISBN 0-943396-61-1, p84:

"If the geographic co-ordinates of two points on the surface of the Earth are known, the shortest distance *s* between these points, measured along the Earth's surface, can be calculated. Let the first point have longitude and latitude L_1 and $\varphi 1$, respectively. Let L_2 and $\varphi 2$ be the co-ordinates of the second point. We will suppose that these points are at sea level. If no great accuracy is needed, we may consider the Earth as being spherical with a mean radius of 6371 kilometres.

Find the angular distance *d* between the two points by means of the formula

$$\cos d = \sin \varphi 1 \sin \varphi 2 + \cos \varphi 1 \cos \varphi 2 \cos (L_1 - L_2)$$

Then the required linear distance s is

$$\frac{6371 \pi d}{180}$$ kilometres, where *d* is expressed in degrees."

You will find *sin*, *cos*, *acos* (radians) in *cmath* library. Use: `#include<cmath>`
To assign a string to a variable you must use: `strcpy (target, source);`
You can look up the co-ordinates of places to enter in your program using Google Maps. Alternatively, you can also find the co-ordinates by using: http://www.astro.com/cgi-bin/atlw3/aq.cgi?lang=e .