CARDIFF UNIVERSITY

EXAMINATION PAPER

Academic Year:	**2015-2016**
Examination Period:	**Spring**
Examination Paper Number:	**CMT304**
Examination Paper Title:	**Programming Paradigms**
Duration:	**2 hours**

Do not turn this page over until instructed to do so by the Senior Invigilator.

Structure of Examination Paper:

There are **three** pages.
There are **four** questions in total.
There are no appendices.
The maximum mark for the examination paper is 60 and the mark obtainable for a question or part of a question is shown in brackets alongside the question.

Students to be provided with:

The following items of stationery are to be provided:
One answer book.

Instructions to Students:

Answer **three** questions.

Important note: if you answer more than the number of questions instructed, then answers will be marked in the order they appear only until the above instruction is met. Extra answers will be ignored. Clearly cancel any answers not intended for marking. Write clearly on the front of the answer book the numbers of the answers to be marked.

Students are permitted to introduce to this examination any textbook, any printed or handwritten notes, and other similar materials. These may be annotated, highlighted and bookmarked as desired.

The use of a translation dictionary between English or Welsh and another language is permitted in this examination.

The use of electronic devices is not permitted.

Q1. (i) A programming language should ideally be based on concepts which are *general, orthogonal, regular,* and *uniform.* Discuss briefly, using *your own* concrete example in each case to back up your arguments, whether the language Pascal possesses these desirable attributes. (Do *not* use examples given in the notes). [8]

(ii) *Java* quickly replaced *Pascal* as a teaching language of choice when it was introduced. What are the two main reasons other than any given above? [2]

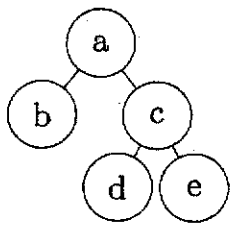
(iii) What are the key *observations* that lead to the development of *formal methods*? [3]

In what main way do formal methods *differ* from other programming paradigms, and what *advantages* does this lead to in terms of the code? [3]

A computer-on-a-chip is to be developed and programmed to perform the communications between a cell phone and a phone company's base stations. Discuss why this programming task may be particularly suited to use of *formal methods.* [4]

Total Marks [20]

Q2. Consider the *binary tree* below where the letters are values stored in nodes:



(i) Show how to write a *Prolog functor* `node` to represent a general node in such a tree. How could you use this functor to *represent* the tree shown above? Define a functor `leaf` which tests if a node is a leaf node. [5]

(ii) Write simple code for a *Prolog functor* `member` which will check if a given value is stored somewhere in the tree. What changes are needed to this simple code to make it more *efficient,* to ensure that Prolog does not continue searching the rest of the tree after it finds the desired value? How do these changes work? [7]

(iii) Write a *Prolog functor* `count` which *counts* the number of nodes in a tree. What happens if we call `count` with an *uninstantiated* value for both the tree and the count? [8]

Total Marks [20]

Q3. (i) How do (a) *functional* languages, (b) *object oriented* languages, view the concept of *state*? How does *Scala*, as a *multi-paradigm* language, reconcile these viewoints? [5]

(ii) Write a Scala *function* `count(a, p)` which takes an array `a` whose elements are of some type `T`, and a predicate `p` which operates on an item of type `T`, and *counts* the number of times `p` is satisfied by the elements of array `a`. [10]

(iii) Show how to *define* an *array* consisting of pairs of integers like (1,2), (5,25), etc. Use a *lambda function* and the `count` function you have written to count how many of these number pairs are of the form (x, x^2). [5]

Total Marks [20]

Q4. Universities have to solve complex *timetabling* problems. These involve allocating the lecturer and students on a given module to one of several classrooms, for the modules in a given semester, and providing timetables and other useful information to interested parties. Not all students take the same combinations of modules, and the classrooms are not all the same. However, for simplicity, you may *assume* that

- each module has just one lecturer,
- each module has just one lecture per week,
- teaching takes place in fixed 1 hour slots at specific times from Monday to Friday each week,
- all classrooms provide the same kind of equipment.

(i) Why is this a suitable problem area for *declarative programming*? [5]

(ii) Outline the *basic* kinds of *declarative statements* needed for such problems, justifying why they are needed, and showing how they might be written. [10]

(iii) Why are *imperative statements* needed, too? What *imperative statements* might be provided? [5]

Total Marks [20]