1	**MODULE CODE:**	**CFM 2175**
2	**MODULE TITLE:**	**Computing Science and Mathematics**
3	**SCHOOLS(S) INVOLVED IN DELIVERY:**	Computing and Engineering
4	**NAME OF COURSE(S):**	BSc(Hons) Software Development BSc(Hons) Computing Science BSc(Hons) Computer Games Programming BSc(Hons) Secure and Forensic BA(Hons) Business Computing with Software Development MEng Software Engineering MEng European Software Engineering
5	**MODULE LEADER:**	John Turner
6	**LOCATION:**	Queensgate Campus
7	**MODULE TYPE:**	Core
8	**CREDIT RATING:**	20 credits
9	**LEVEL:**	F
10	**LEARNING METHODS:**	**Lectures:** 24 hrs **Tutorials/Practicals:** 24 hrs **Directed Unsupervised Learning:** 152 hrs
11	**PRE0REQUISITE(S):**	None
12	**RECOMMENDED PRIOR STUDY:**	None
13	**CO-REQUISITE(S):**	None
14	**PROFESSIONAL BODY REQUIREMENTS:**	N/A
15	**GRADED OR NON-GRADED:**	Graded
16	**BARRED COMBINATIONS:**	None

17 SYNOPSIS:

This module is both knowledge-based and practice-based, and is designed to provide students with an introductory understanding and perspective of basic computing science and mathematical concepts as they relate to the activity of software development. Practical use of the theoretical aspects of computation will be studied with the aid of the currently taught programming languages (e.g. Java) and a package with an interface allowing for the direct implementation of finite-state machines (e.g. Kara).

18 OUTLINE SYLLABUS

Basic set theory: notation, union, intersection, difference, cardinality. Venn diagrams. Sequences: notation and operations.

Graphs: paths, cycles, connectivity. Trees: notation and representation of algebraic expressions.

Relations: representation by sets of ordered pairs, Boolean matrices and digraphs. Functions.

Propositional logic: connectives, requirements capture, truth tables, equivalence, semantic consequence, inference rules.

Finite-state transducers: state-transition diagrams and tables. Finite-state recognisers and their relationship to formal languages.

Introduction to push down automata and Turing machines.

Grammars: production rules, BNF and EBNF notation, regular expressions. Classification of grammars.

The basic components of algorithms described in a suitable language-independent notation:

sequence, selection and iteration;

basic data structures – arrays, lists, trees;

concepts for the modularisation of algorithms;

spanning tree and shortest distance algorithms for graphs;

basic searching and sorting algorithms.

19 LEARNING OUTCOMES:

19.1 Knowledge and understanding outcomes

On completion of this module, the student will be able to:

19.1.1 Appreciate the role of grammars and state machine descriptions as models for understanding languages and the process of computation.
19.1.2 Demonstrate a basic understanding of fundamental algorithms and data structures.
19.1.3 Demonstrate a basic understanding of the notions of proposition, set, sequence, relation and function.
19.1.4 Discuss the relationship between various models of computation.
19.1.5 Understand the role of mathematics in data and process abstraction.

19.2 Ability outcomes

On completion of this module, the student will be able to:

19.2.1 Analyse graphs and apply these to a problem context.
19.2.2 Apply sets, sequences and relations in an appropriate way.
19.2.3 Manipulate simple propositions.
19.2.4 Conduct proofs framed within the propositional calculus.
19.2.5 Identify the language generated by a grammar, and build a finite-state automaton to perform a specific task.
19.2.6 Use simple grammars as a basis for the description of formal languages.
19.2.7 Apply fundamental algorithms in an appropriate way.
19.2.8 Use simple computational model concepts in the analysis of software.
19.2.9 Develop basic demonstration programs using the taught algorithms and data structures.

20 Assessment strategy:

20.1 Formative assessment

Students will be required to undertake a computing science self-test in the second term. Those students who fail the test will be encouraged to attend additional workshops.

20.2 Summative assessment

An assignment including both mathematics and computing science questions, and incorporating a programming component, will contribute 60% to the overall coursework mark. This will assess learning outcomes 19.1.1, 19.1.3, 19.1.5, 19.2.1-19.2.9. For a pass, a majority of the mathematical and computing science concepts and calculations used must be accurate and appropriate. In addition, a significant amount of the functionality required for the programming component must be correctly implemented, and must be based upon appropriate data structures. For a grade B (or A), most (or all) of the relevant concepts and mathematical /computing science structures must be applied correctly, and the structures must be manipulated accurately.

In addition, most (or all) of the functionality required for the programming component must be achieved, and the data structures must be entirely appropriate.

An in-class test will focus upon the computing science aspect of the module and contribute 40% to the overall coursework mark. This will assess the learning outcomes 19.1.1, 19.1.2, 19.2.5-19.2.7. For a pass, a majority of the concepts and techniques required for the given algorithms must be applied correctly. For a grade B (or A), most (or all) of the concepts and techniques used must be appropriate and accurate.

The overall coursework mark will contribute 50% to the final module mark.
A 2 hour end-of-year examination will also contribute 50% to the final module mark. This will assess the learning outcomes 19.1.1-19.1.5 and 19.2.1-19.2.7.

21 LEARNING STRATEGY:

Both mathematical and computing science theoretical concepts will be introduced in a lecture series. This will be supported by a programme of practicals in which students will air issues in a broader context and study exercises to develop skills in developing and understanding algorithms, both expressed in abstract notations and in the main programming language taught (Java). In tutorials students will apply mathematical theory to problems drawn principally from computing. It is intended that unsupervised time will be spent in the preparation of solutions to larger scale problems.

22 INDICATIVE REFERENCES

Brookshear, J. Glenn	Computer Science an overview. Addison Wesley, 2001
Parkes, A	Computable Languages and Abstract Machines. International Thompson, 1996
Goldschlager & Lister	Computer Science – A Modern Introduction. Prentice Hall, 1989
Wood, D	Theory of Computation. Harper & Row, 1987
Wiitala, S.A	Discrete Mathematics. McGraw Hill, 1987.
Molluzzo, J.C	A First Course in Discrete Mathematics.
Munro, J.E	Discrete Mathematics for Computing. Chapman & Hall, 1992.

CFM 2175 Computing Science and Mathematics

Assessment

There will be a change to the assessment this year, which is not yet shown in the module specification. There will no longer be an in-class test in term 2.
Assessment will be by a two-part assignment , contributing **50%** overall, and a two-hour written examination at the end of the module, contributing **50%**.

The following table shows how each outcome is assessed:-

Outcome	Assignment	End - of Module Examination
19.1.1	●	●
19.1.2	●	●
19.1.3	●	●
19.1.4		●
19.1.5	●	●
19.2.1	●	●
19.2.2	●	●
19.2.3		●
19.2.4		●
19.2.5	●	●
19.2.6	●	●
19.2.7	●	●
19.2.8	●	
19.2.9	●	

CFM 2175 Computing Science and Mathematics

Assessment

The assignment will consist of two equally weighted parts. The first part of the assignment will be handed out in the week commencing Monday 09/11/09 (week 7, first term), to be handed in by Friday 15/01/10 (week1, second term). The second part of the assignment will be handed out in the week commencing 08/02/10 (week 17 second term) and the latest hand-in date will be Friday 12/02/10 (week 21, second term).

You will be notified of the exact date of the exam, but it will be during the University exam period in May 2010.