Diploma in Computer Science
Syllabus and Booklist 2007-2008

Contents

Introduction

The Computer Laboratory

The Computer Laboratory, formerly known as the Mathematical Laboratory, was founded in 1937 to house a mechanical differential analyser. After the war it became the centre for pioneering work on electronic digital computers, and since 1950 has concentrated entirely on the digital field. Two early computers were designed and built in the Laboratory. The first of these was the EDSAC, which did its first calculations in May 1949 and ran until July 1958. It was succeeded by EDSAC 2, which was finally switched off in November 1965.

While the enthusiasm for building hardware has not diminished, the Laboratory now houses a variety of commercially made computers. There is also an assortment of specialised equipment used mainly in connection with the Department's research work.

The Diploma course

The course was founded in October 1953 as *The Diploma in Numerical Analysis and Automatic Computing*, and was the first taught course in computing to be instituted in Britain. It has continually evolved as the subject has developed and expanded, and now provides students with good opportunities to follow their particular interests, especially in the areas of systems software and computer design. It acts as a conversion course and is intended for graduates of Mathematics, Natural Science or Engineering, who want to become computer professionals. It is not really suitable for those who want a data processing qualification in order to write programs in a business setting: important topics in, for example, systems analysis for such applications are not dealt with. The course is not a preparation for IT managers.

The course takes ten months of full-time study, near the end of which candidates are required to take a written examination and submit a dissertation on a subject of their choosing. The written papers take place in late May/early June and the dissertations must be handed in at the end of July. Oral examinations (based on the dissertation) are sometimes required by the Examiners, and these take place in early August, so students should be prepared either to stay on or to return to Cambridge if they are called for an oral examination. The dissertation project normally comprises practical work on one of the computers in the Laboratory, but can sometimes include the design and construction of new equipment. Candidates who acquit themselves especially well in the examination are awarded a mark of distinction.

The first term of the course is largely devoted to ensuring that all students possess adequate background knowledge of computer structure, computing techniques and programming practice.

Many lecture courses are on subjects regarded as essential core material. Additionally, a selection of more advanced courses are available which can be regarded as options for those with, perhaps, a specialist mathematical or engineering background. Personal supervision is arranged by the student's college and forms a feature of the course. There are also opportunities to become acquainted with some of the research work in progress in the Laboratory.

Requirements for admission

Candidates for admission are normally required to have a good honours degree in a scientific subject; it would be usual for that subject to be Mathematics, Engineering or Natural Sciences although applicants from other disciplines will be considered. Knowledge of mathematics to at least A-level standard is assumed. Candidates for admission will be asked to produce a short statement explaining why they are interested in the course and what they hope to achieve by doing it.

In the last two weeks of July the Computer Laboratory runs a 10-day programming course. Diploma candidates ordinarily attend a repeat of this course in October but those candidates who *are* able to attend the July course are urged to consider doing so.

For those without previous practical experience in programming, attendance at this course is almost essential but even those with considerable experience have found it most helpful to learn a new programming language and become familiar with a new environment without the pressures of attending numerous other courses at the same time. In some cases the July course may be used in a probationary manner to assess the aptitude of an intending candidate, before completing the admission arrangements for the Diploma course itself.

The Engineering and Physical Sciences Research Council, which formerly made a number of studentships available for this course, no longer does so. Cambridge home students currently in their third year of undergraduate studies who wish to continue to be eligible for support from public funds (in the form of student loans and means-tested fee support) and who are interested in the Diploma course may like to consider Computer Science Part II (General) which is essentially the written papers of the Diploma but not the dissertation. This course, like the Diploma, is intended for those who have not taken Part IB of the Computer Science Tripos. Information about the Part II (General) syllabus is given elsewhere.

Applications for admission, together with the short personal statement referred to above, must be received by 31 March. Application forms and copies of the *University of Cambridge Graduate Studies Prospectus* may be obtained from The Secretary of the Board of Graduate Studies, 4 Mill Lane, Cambridge, CB2 1RZ (telephone: +44 1223 332200). The prospectus can be seen at
`http://www.admin.cam.ac.uk/univ/gsprospectus/`

The curriculum

The syllabus information given here is for guidance only and should not be considered definitive. Current timetables can be found at
`http://www.cl.cam.ac.uk/teaching/lectlist/`

For most of the courses listed below, a list of recommended books is given. These are roughly in order of usefulness, and lecturers have indicated by means of an asterisk those books which are most recommended for purchase by College libraries.

The Computer Laboratory Library aims to keep at least one copy of each of the course texts in "The Booklocker" (see `http://www.cl.cam.ac.uk/library/`).

Information on courses is arranged by terms:

Michaelmas (1 October to 19 December)
Lent (5 January to 25 March)
Easter (10 April to 25 June)

For further copies of this booklet and for answers to general enquiries about Computer Science courses, please get in touch with:

Student Administrator
University of Cambridge
Computer Laboratory
William Gates Building
J J Thomson Avenue
Cambridge
CB3 0FD

telephone: +44 1223 334656
fax: +44 1223 334678
e-mail: `postgraduate.admissions@cl.cam.ac.uk`

Michaelmas Term 2007

Algorithms II

Lecturer: Dr F.M. Stajano

No. of lectures: 8

Prerequisite courses: Algorithms I (CST students) or Data Structures and Algorithms (Diploma)

This course is a prerequisite for Computer Graphics and Image Processing, Complexity Theory, Artificial Intelligence I.

Aims

The aim of this course is to give further insights into the design and analysis of non-trivial algorithms through the discussion of several complex algorithms in the fields of graphs and computer graphics, which are increasingly critical for a wide range of applications.

Lectures

- **Advanced data structures.** Fibonacci heaps. Disjoint sets. [Ref: Ch 20, 21] [1–2 lectures]
- **Graph algorithms.** Graph representations. Breadth-first and depth-first search. Topological sort. Minimum spanning tree. Kruskal and Prim algorithms. Shortest paths. Bellman–Ford and Dijkstra algorithms. Maximum flow. Ford–Fulkerson method. Matchings in bipartite graphs. [Ref: Ch 22, 23, 24, 25, 26] [4–5 lectures]
- **Geometric algorithms.** Intersection of segments. Convex hull: Graham's scan, Jarvis's march. [Ref: Ch 33] [1–2 lectures]

Objectives

At the end of the course students should

- have a good understanding of how several elaborate algorithms work
- have a good understanding of how a smart choice of data structures may be used to increase the efficiency of particular algorithms
- be able to analyse the space and time efficiency of complex algorithms
- be able to design new algorithms or modify existing ones for new applications and reason about the efficiency of the result

Recommended reading

* Cormen, T.H., Leiserson, C.D., Rivest, R.L. & Stein, C. (2001). *Introduction to Algorithms*. MIT Press (2nd ed.). ISBN 0-262-53196-8

Sedgewick, R. (2004). *Algorithms in Java, vol 2.* (note that C and C++ editions are also available and are equally good for this course). Addison-Wesley. ISBN 0-201-36121-3.
Kleinberg, J. & Tardos, É. (2006). *Algorithm design.* Addison-Wesley. ISBN 0-321-29535-8.

Students are expected to buy and make extensive use of one of the above references: those not doing so will be severely disadvantaged. The easiest and recommended choice is Cormen *et al.* which covers all the topics in the syllabus: the pointers in the syllabus are to chapters in that book. The other textbooks are all excellent alternatives and are sometimes clearer or more detailed than Cormen, but they are not guaranteed to cover every item in the syllabus. Their relative merits are discussed in the course handout.

Business Studies

Lecturer: Mr J.A. Lang

No. of lectures: 8

Or “How to Start and Run a Computer Company”

This course is a prerequisite for E-Commerce.

Aims

The aims of this course are to introduce students to all the things that go to making a successful project or product other than just the programming. The course will survey some of the issues that students are likely to encounter in the world of commerce and that need to be considered when setting up a new computer company.

Lectures

- **So you’ve got an idea?** Introduction. Why are you doing it and what is it? Types of company. Market analysis. The business plan. Futures: some emerging ideas for new computer businesses.

- **Money and tools for its management.** Introduction to accounting: profit and loss, cash flow, balance sheet, budgets. Sources of finance. Stocks and shares. Options and futures.

- **Setting up: legal aspects.** Company formation. Brief introduction to business law; duties of directors. Shares, stock options, profit share schemes and the like. Intellectual Property Rights, patents, trademarks and copyright. Company culture and management theory.

- **People.** Motivating factors. Groups and teams. Ego. Hiring and firing: employment law. Interviews. Meeting techniques.

- **Project planning and management.** Role of a manager. PERT and GANTT charts, and critical path analysis. Estimation techniques. Monitoring.

- **Quality, maintenance and documentation.** Development cycle. Productization. Plan for quality. Plan for maintenance. Plan for documentation.
- **Marketing and selling.** Sales and marketing are different. Marketing; channels; marketing communications. Stages in selling. Control and commissions.
- **Growth and exit routes.** New markets: horizontal and vertical expansion. Problems of growth; second system effects. Management structures. Communication. Exit routes: acquisition, floatation, MBO or liquidation. Summary. Conclusion: now you do it!

Objectives

At the end of the course students should

- be able to write and analyse a business plan
- know how to construct PERT and GANTT diagrams and perform critical path analysis
- appreciate the differences between profitability and cash flow, and have some notion of budget estimation
- have an outline view of company formation, share structure, capital raising, growth and exit routes
- have been introduced to concepts of team formation and management
- know about quality documentation and productization processes
- understand the rudiments of marketing and the sales process

Recommended reading

Lang, J. (2001). *The high-tech entrepreneur's handbook: how to start and run a high-tech company*. FT.COM/Prentice Hall.

Students will be expected to able to use Microsoft Excel and Microsoft Project.

For additional reading on a lecture-by-lecture basis, please see the course website.

Students are strongly recommended to enter the CU Entrepreneurs Business Ideas Competition `http://www.cue.org.uk/`

Computer Design

Lecturer: Dr S.W. Moore

No. of lectures: 16

Prerequisite courses: none, but Operating Systems, Digital Electronics and ECAD would be helpful

This course is a prerequisite for the Part II courses Comparative Architectures and VLSI Design.

Aims

The aims of this course are to introduce the hardware/software interface models and the hardware structures used in designing computers. The first seven lectures are concerned with the hardware/software interface and cover the programmer's model of the computer. The last nine lectures look at hardware implementation issues at a register transfer level.

Lectures

- Introduction to the course and some background history.
- Historic machines. EDSAC *versus* Manchester Mark I.
- Introduction to RISC processor design and the MIPS instruction set.
- MIPS tools and code examples.
- Operating system support including memory hierarchy and management.
- Intel x86 instruction set.
- Java Virtual Machine.
- Memory hierarchy (caching).
- Executing instructions. An algorithmic viewpoint.
- Basic processor hardware, pipelining and hazards. [2 lectures]
- Verilog implementation of a MIPS processor. [3 lectures]
- Internal and external communication.
- Data-flow and comments on future directions.

Objectives

At the end of the course students should

- be able to read assembler given a guide to the instruction set and be able to write short pieces of assembler if given an instruction set or asked to invent an instruction set

- understand the differences between RISC and CISC assembler
- understand what facilities a processor provides to support operating systems, from memory management to software interrupts
- understand memory hierarchy including different cache structures
- appreciate the use of pipelining in processor design
- understand the communications structures, from buses close to the processor, to peripheral interfaces
- have an appreciation of control structures used in processor design
- have an appreciation of how to implement a processor in Verilog

Recommended reading

* Harris, D. & Harris, S. (2007). *Digital design and computer architecture: from gates to processors.* Morgan Kaufmann.

Recommended further reading:

Hennessy, J. & Patterson, D. (2006). *Computer architecture: a quantitative approach.* Elsevier (4th ed.). ISBN 978-0-12-370490-0. (Older versions of the book are also still generally relevant.)
Patterson, D.A. & Hennessy, J.L. (2004). *Computer organization and design.* Morgan Kaufmann (3rd ed., as an alternative to the above). (2nd ed., 1998, is also good.)
Pointers to sources of more specialist information are included in the lecture notes and on the associated course web page.

Data Structures and Algorithms

Lecturer: Dr F.M. Stajano

No. of lectures: 16

This course is a prerequisite for Algorithms II, Computer Graphics and Image Processing, Complexity Theory, Artificial Intelligence I.

Aims

The aim of this course is to provide an introduction to computer algorithms and data structures, with an emphasis on foundational material. It is suitable for people who did not take discrete mathematics in Part IA.

Lectures

- **Discrete Mathematics.** Common mathematical notation. Proofs. Induction. Sets, tuples, functions. Relations and Graphs. Combinations and permutations. [Ref: Ch 1; App B, C] [3–4 lectures]

- **Algorithm Fundamentals.** Algorithm analysis and design. Models of a computer; costs. Asymptotic notation. Computational complexity. Recurrences. Ideas for algorithm design: divide and conquer, dynamic programming, greedy algorithms and other useful paradigms. [Ref: Ch 1, 2, 3, 4, 15, 16] [2–3 lectures]

- **Sorting.** Insertion sort. Merge sort. Heapsort. Quicksort. Other sorting methods. Finding the minimum and maximum. [Ref: Ch 2, 6, 7, 8, 9] [4–5 lectures]

- **Data structures.** Abstract data types. Pointers, stacks, queues, lists, trees. Hash tables. Binary search trees. Red-black trees. B-trees. Priority queues and heaps. [Ref: Ch 10, 11, 12, 13, 18, 19, 20, 21] [5–7 lectures]

Objectives

At the end of the course students should

- have a good understanding of how several fundamental algorithms work, particularly those concerned with sorting and searching

- have a good understanding of the fundamental data structures used in computer science

- be able to analyse the space and time efficiency of most algorithms

- be able to design new algorithms or modify existing ones for new applications and reason about the efficiency of the result

Recommended reading

* Cormen, T.H., Leiserson, C.D., Rivest, R.L. & Stein, C. (2001). *Introduction to Algorithms*. MIT Press (2nd ed.). ISBN 0-262-53196-8
Sedgewick, R. (2004). *Algorithms in Java, vol 1.* (note that C and C++ editions are also available and are equally good for this course). Addison-Wesley. ISBN 0-201-36120-5.
Kleinberg, J. & Tardos, É. (2006). *Algorithm design.* Addison-Wesley. ISBN 0-321-29535-8.
Knuth, D.E. (1997). *The art of computer programming* (three volumes so far; a boxed set is also available). Addison-Wesley (3rd ed.). ISBN 0-201-89683-4, 0-201-89684-2 and 0-201-89685-0.

Students are expected to buy and make extensive use of one of the above references: those not doing so will be severely disadvantaged. The easiest and recommended choice is Cormen *et al.* which covers all the topics in the syllabus: the pointers in the syllabus are to chapters in that book. The other textbooks are all excellent alternatives and are sometimes clearer or more detailed than Cormen, but they are not guaranteed to cover every item in the syllabus. Their relative merits are discussed in the course handout.

Digital Electronics

This course is taken by Part IA (50% Option), Part II (General) and Diploma students.

Lecturer: Dr I.J. Wassell

No. of lectures and practical classes: 11 + 7

This course is a prerequisite for ECAD (Part IB) and VLSI Design (Part II).

Aims

The aims of this course are to present the principles of combinational and sequential digital logic design and optimisation at a gate level. The use of transistors for interfacing and constructing gates is also introduced.

Lectures

- **Introduction.** The parts of a simple computer. Binary and representation of integers in binary. ASCII codes for characters. Switch logic.
- **Boolean algebra.** Truth tables and Boolean algebra. Idealised logic gates and symbols. DeMorgan's rules. Logic to perform addition with ripple carry.
- **Logic minimisation.** Normal forms. Karnaugh maps for Boolean optimisation.
- **Complexities of logic design.** Multilevel logic. An introduction to timing diagrams. Digital signals in the analog world. Hazards and hazard elimination. Fast carry propagation.
- **Introduction to practical classes.** Use of prototyping box, breadboard and digital logic integrated circuits (ICs or chips).
- **Flip-flops.** Memory elements, state and state diagrams. RS asynchronous flip-flop. Synchronous flip-flops: D, T and JK flip-flops. Setup and hold times.
- **Synchronous state machines.** Moore and Mealy finite state machines. Reset and self starting. State transition diagrams.
- **Further state machines.** State assignment and unique state encoding. One hot encoding.
- **Memories and programmable logic.** SRAM, ROM addressing, busses and control. Tri-state drivers. The structure and use of programmable logic arrays (PLAs). A brief introduction to FPGAs.
- **Discrete components.** Revision of resistance, Ohm's law and capacitance. Characteristics of diodes, NMOS and PMOS field effect transistors. NMOS and CMOS inverters. Rise and fall times. Voltage followers. [2 lectures]

Objectives

At the end of the course students should

- understand the relationships between combination logic and boolean algebra, and between sequential logic and finite state machines
- be able to design and minimise combinational logic
- appreciate tradeoffs in complexity and speed of combinational designs
- understand how state can be stored in a digital logic circuit
- know how to design a simple finite state machine from a specification and be able to implement this in gates and edge triggered flip-flops
- understand how to use MOS transistors

Recommended reading

* Harris, D.M. & Harris, S.L. (2007). Digital design and computer architecture. Morgan Kaufmann.
Katz, R.H. (2004). *Contemporary logic design*. Benjamin/Cummings. The 1994 edition is more than sufficient.
Hayes, J.P. (1993). *Introduction to digital logic design*. Addison-Wesley.

Books for reference:

Horowitz, P. & Hill, W. (1989). *The art of electronics*. Cambridge University Press (2nd ed.) (more analog).
Weste, N.H.E. & Harris, D. (2005). *CMOS VLSI Design – a circuits and systems perspective*. Addison-Wesley (3rd ed.).
Mead, C. & Conway, L. (1980). *Introduction to VLSI systems*. Addison-Wesley (used in Part II VLSI).
Crowe, J. & Hayes-Gill, B. (1998). *Introduction to digital electronics*. Butterworth-Heinemann.
Gibson, J.R. (1992). *Electronic logic circuits*. Butterworth-Heinemann.

Elementary Use of the Unix Teaching Service

Lecturer: Mr R.J. Stibbs

No. of lectures and practicals: 3 + 8

Aims

The aims of the course are to introduce the participants, who are not expected to have any previous knowledge of Unix, to the PWF Linux system and to give them enough basic knowledge to equip them to understand further courses and, if needed, to use the PWF Linux system for exercises and dissertation work.

Lectures

The three 2-hour sessions cover various aspects of Unix and Unix utilities. Considerable use is made of live demonstrations.

Objectives

At the end of the course students should

- be familiar with the use of the system for email communication
- be familiar with the use of Unix utilities for file-handling and archiving
- have the ability to understand and write bash scripts
- have created their own web pages on the system

Floating-Point Computation

Lecturer: Professor A. Mycroft

No. of lectures: 4

This course is useful for the Part II courses Advanced Graphics and Digital Signal Processing.

Aims

This course has two aims: firstly to provide an introduction to (IEEE) floating-point data representation and arithmetic; and secondly to show, mainly by fun examples backed up by simple analysis, how naïve implementations of obvious mathematics can go badly wrong.

Lectures

- **IEEE Floating-point representation and arithmetic (32 and 64 bits).** Overflow, underflow, progressive loss of significance. Rounding modes.
- **How floating-point computations diverge from real-number calculations.** Absolute Error, Relative Error, Machine epsilon. Solving a quadratic.
- **Iteration and when to stop.** Why summing a Taylor series is problematic (loss of all precision, range reduction, non-examinable hint at economisation).
- **Ill-conditioned or chaotic problems. Testing. Packages.** Non-examinable: exact real arithmetic.

Objectives

At the end of the course students should

- be able to convert simple decimal numbers to and from IEEE floating-point format, and to perform simple arithmetic
- be able to identify problems with floating-point implementations of simple mathematical problems

- know when a problem is likely to yield incorrect solutions no matter how it is processed numerically
- know to use a professional package whenever possible

Recommended reading

None.

Foundations of Programming (in Java)

Lecturer: Dr F.H. King

No. of lectures and practicals: 20 + 20

This course is normally held in the last two weeks of July for Part II (General) students; it is mandatory for those students to attend the course in July. Diploma students normally attend the course at the beginning of the Michaelmas Term (starting on the Monday before the start of Full Term), but any prospective Diploma students who can attend the July run of the course are urged to consider doing so.

The implementation of Java that is used on the course runs under a local version of Unix. Features of Unix are introduced as they are required during the course but, in the interests of readability, the information below is principally concerned with Java, and references to Unix are sparse.

Aims

The principal aim of this course is to introduce the rudiments of programming via the Java language. A secondary aim is to provide sufficient instruction in Unix to run both Java applications and Java applets using this operating system.

Lectures

- **Introduction.** A practical introduction to programming in Java using Unix and X-windows. Unix commands. X-windows.
- **Files and directories.** The Unix filing system and associated commands.
- **Elements of programming in Java.** Imperative programming. Compiling and running simple Java programs. The Java Virtual Machine. Java classes, data fields and methods. Method arguments and local variables.
- **Some Java constructs.** Loops, conditions and relational operators. Reserved words. Arrays.
- **Types, classes and objects.** Types and strong typing. Variables, constants, operators, and expressions. Primitive data types `int` and `float`. Reference data types. Introducing objects: instantiation, `this`, constructors. Visibility modifiers, encapsulation. The `toString()` method. The `static` modifier. Instance variables *versus* class variables, instance methods *versus* class methods.

- **Further Java types, more about objects.** Primitive data types `double` and `boolean`. Objects continued: class hierarchy, inheritance, overriding, `super()`, overloading.
- **Yet more Java types, HTML.** Number bases. Primitive data types `long`, `short` and `byte`. Reference data type `String`. Formatted output. Call by value *versus* call by reference. The `switch` statement. Rudiments of HTML.
- **Sorting, abstract classes.** Introduction to sorting. Rounding errors. Generic types, abstract classes. Case study: abstract class `Shape`.
- **Packages, recursion.** Java packages: the `package` statement and the `import` statement. `$CLASSPATH`. Introduction to recursion. Case study: the Tower of Hanoi.
- **Multiple inheritance, interfaces.** Multiple inheritance via interfaces. The `instanceof` operator. Case study: abstract class `Shape` continued. Mid-course exercise: the Power Problem, an $O(\log n)$ algorithm.
- **Applets, bit-level programming.** Introduction to applets: the `appletviewer` command. The Abstract Windowing Toolkit. Wrapper classes. Operations on bits. Case study: the Eight Queens Problem.
- **Exception handling.** Exceptions: declaring exceptions, throwing and catching exceptions, `try–catch–finally`. Lists in Java.
- **Input, enum.** Primitive data type `char`. Unicode. The Java `InputStreamReader`. Use of `enum` classes. Case studies: extracting integers from unformatted data and an `enum` illustration.
- **Stylistic considerations.** Use and misuse of object oriented programming. Helper classes. Case study: abstract class `Shape` improved. Eight-Queens-like problems. Case study: the Croquet Fixtures Problem.
- **Applets, GUIs, threads.** More on applets: adding an `ActionListener`, implementing an `ActionEvent`. Member classes. Case study: a simple Graphical User Interface. Introduction to threaded code. The `finalize()` method.
- **More about threads.** The `synchronized` keyword, monitors. Case study: the Triangular Solitaire Problem.
- **Applets, Thread case studies.** Adding a `MouseListener`, the `MouseAdapter` class. A simple buffer. Blocked on `synchronized` and blocked on `wait()`. Case study: the Dining Philosophers Problem.
- **Class Shape concluded, trees and lattices.** Extending the inheritance hierarchy. Case study: abstract class `Shape` concluded. Tree representation of an arithmetic expression. Tree searches *versus* lattice searches.
- **Classes within classes.** Ordinary top-level classes. Nested top-level classes. Member classes. Local classes. Anonymous inner-classes.

- **An expression parser.** Hashing. The lexical analysis and syntax analysis of an arithmetic expression.

Objectives

At the end of the course students should

- have some familiarity with Unix and the Emacs editor
- be able to write simple Java applications including those which involve exception handling and threaded code
- be able to write simple Java applets

Recommended reading

* Flanagan, F. (1997). *Java in a nutshell.* O'Reilly.
* Eckel, B. (1998). *Thinking in Java.* Prentice Hall.

How to Study Computer Science

This lecture is for all Part IA, Part II (General) and Diploma students.

Lecturer: Dr N.A. Dodgson

No. of lectures: 1

Aims

This single lecture gives an overview of how the course works and a range of advice in how to get the best out of your time at Cambridge.

Lecture

- **Overview.** How Cambridge works. The interaction and split of responsibilities between College and University. The structure of the computer science courses.
- **Support structures.** Directors of studies, tutors, lecturers, supervisors, friends, classmates: your support network and the circumstances in which you should call on each type of person.
- **Being organised.** Why a diary is essential. A typical week's activities analysed and diarised.
- **Lectures.** How to take notes. How to handle the handouts.
- **Supervisions.** What is expected of you. What should you expect.
- **Practical work.** Learning by doing.
- **Health & safety.** How to avoid repetitive strain injury. Support structures for if you get affected.

Objectives

Students should understand the expectations that the Computer Laboratory has of them, should be aware of the resources and support available to them and should understand the importance of good working habits.

Recommended reading

Northedge, A., Thomas, J., Lane, A. & Peasgood, A. (1997). *The sciences good study guide.* Open University.
Fairbanks, A. (1932). *A handwriting manual.* Faber and Faber.
* Pascarelli, E.F. & Quilter, D. (1994). *Repetitive strain injury: a computer user's guide.* New York: Wiley.

Mathematics for Computation Theory

Lecturer: Dr J.K.M. Moody

No. of lectures and practical classes: 12 + 6

This course is a prerequisite for Introduction to Security, Computation Theory, Artificial Intelligence I, Quantum Computing.

Aims

The aims of this course are to introduce the basic notation and constructs of set theory. The course will cover functions and relations, and will treat sufficient of the theory of partial orders to handle well-founded induction. The ideas are illustrated by developing the theory of finite automata and regular languages, including a proof of Kleene's Theorem.

Lectures

Part A. Discrete Mathematics

- **Modelling discrete systems.** Computation as a discrete process. Sets: membership, subsets, union, intersection, complement, difference. Symmetric difference. Boolean algebra. Venn diagrams. Propositions and predicates.

- **Constructions on sets.** Cartesian product. Disjoint union (connection with datatypes). Relations as a subset of a product. Binary relations. Functions and partial functions.

- **Algebra of relations.** Product and inverse relations. Composition of functions. Bijections. Function spaces. Some natural bijections between sets. Power sets. Predicates, characteristic functions. Multi-sets. *n*-ary predicates.

- **Relations on a set.** Reflexive, symmetric and transitive properties of a relation on a set. Equivalence relations. Orders, partial and total. Examples.

- **Well-founded relations and the principle of induction.** Well-founded relations. Their derived partial orderings. Minimal elements. Well-ordering. Examples. Mathematical induction. Well-founded induction via minimal counterexamples.

- **Algebraic languages ("free algebras") and structural induction.** Languages of algebraic expressions. Free algebras. Examples. Parse trees. Well-foundedness in free algebras. Principle of structural induction for free algebras. Languages of regular expressions.

Part B. Finite Automata and Regular Languages

- **Deterministic finite automata (DFA).** Mechanical recognition of strings in a language. Stimulus-response behaviour, event detection. Mealey machines, Moore machines. Examples. Accepting strings via some output of a DFA. Examples.

- **Using automata to recognise languages.** Events/languages as sets of strings. Non-deterministic finite automata (NDFA). Proof that the language classes accepted by DFAs and NDFAs coincide.

- **Algebraic operations on languages.** Finite union and intersection, concatenation, and Kleene closure. Regular expressions and the languages they denote. Examples of regular languages, including recognisers.

- **Regular languages are representable events.** Arden's rule as a recursive definition. Its proof by induction. Extension to matrices of events. Statement of Kleene's Theorem, including an overview of the proof structure. Regular events can be represented: proof by induction on the structure of regular expressions using NDFAs.

- **Representable events have regular expressions.** Representable events are regular: proof by induction on the number of states using event transition matrices and Arden's rule.

- **Properties of regular languages.** Strings accepted by an n-state machine. The Pumping Lemma. Examples of languages that are not regular, including palindromes. Complements and intersections of regular languages are regular. Some decidable problems for regular languages. Extended regular expressions. A non-elementary problem. (Cook's tour)

Objectives

At the end of the course students should

- be familiar with the notation of set theory, and be able to use it to express simple mathematical constructs

- know how to translate between relational algebra and relational calculus

- be able to carry out simple proofs by induction, including structural induction over partially ordered (well-founded) sets
- understand how to derive a regular expression to describe the event accepted by a deterministic finite automaton
- understand how to use the Pumping Lemma to show that a given language cannot be regular

Recommended reading

Part A:

* Devlin, K. (2003). *Sets, functions, and logic: an introduction to abstract mathematics.* Chapman and Hall/CRC Mathematics (3rd ed.). ISBN 1-584-88449-5
Rosen, K.H. (2003). *Discrete mathematics and its applications.* McGraw-Hill (5th ed.). ISBN 0-072-93033-0
Kolman, B., Busby, R.C. & Ross, S.C. (2003). *Discrete mathematical structures for computer science.* Prentice Hall (5th ed.). ISBN 0-130-45797-3
Forster, T.E. (2003). *Logic, induction and sets.* LMS Student Text 56. Cambridge University Press. ISBN 0-521-53361-9

Part B:

Hopcroft, J.E., Motwani, R. & Ullman, J.D. (2001). *Introduction to automata theory, languages and computation.* Addison-Wesley (2nd ed.). ISBN 0-201-44124-1
Sudkamp, T.A. (1998). *Languages and machines: an introduction to the theory of computer science.* Addison-Wesley (2nd ed.). ISBN 0-201-15768-3
Hopcroft, J.E. & Ullman, J.D. (1974). *The design and analysis of computer algorithms.* Addison-Wesley. ISBN 0-201-00029-6

Closely related to parts of the course but out of print:

Conway, J.H. (1971). *Regular algebra and finite machines.* Chapman and Hall.
Stanat, D.F. & McAllister, D.F. (1977). *Discrete mathematics in computer science.* Prentice Hall.
Stone, H.S. (1973). *Discrete mathematical structures and their applications.* SRA.

Operating System Foundations

Lecturer: Professor J.M. Bacon

No. of lectures: 16

This course is a prerequisite for Introduction to Security and Distributed Systems, and is helpful for Computer Design.

Aims

The aims of this course are to introduce the basic principles of computer systems organisation and operation; to show how hardware is controlled by program at the hardware/software interface; to outline the basic OS resource management functions: memory, file, device (I/O) and process management; and to explore the need for, principles of and implementation of concurrency control.

Lectures

Part I. Computer organisation [3–4 lectures]

- **Computer design; the hardware and its interfaces.** Bus/memory/CPU organisation. Representation of numbers and characters; instruction sets and addressing. Operation of a simple computer; the fetch–execute cycle.
- **Device management.** I/O devices and interfaces; polling and interrupts. The general exception mechanism for interrupts, errors, system calls, memory management. Character and DMA interfaces.

Part II. Operating system structure and functions [6 lectures]

- **Operating system evolution and structure.**
- **Process management.** Creating virtual processor abstractions. Support for processes (or threads) in operating systems. CPU scheduling.
- **Memory management.** Processes in memory. Logical addresses (virtual address space). Segmented memory. Memory fragmentation. Paged memory: concepts, in-process fragmentation, page tables, hardware support. Paged segments.
- **File management.** File concept. Directory and storage services. File names and meta-data. Directory name-space. File and directory operations. Access control. Existence and concurrency control.

Part III. Concurrency control [3–4 lectures]

- **I/O management- aspects.** Process-device synchronisation. Process - process mutual exclusion and synchronisation via I/O buffers.
- **Multi-threaded processes.** User threads and kernel threads.
- **System structure and interprocess communication (IPC).** Requirements for concurrency control; System structure; Shared memory mechanisms; Semaphores and their implementation; examples; No-shared-memory mechanisms.
- **Composite concurrent operations.** Problem of deadlock. Introduction to transaction concept.

Part IV. Case studies [2–3 lectures]

- **Case studies.** Unix and Windows NT/2000.

Objectives

At the end of the course students should

- be prepared for hardware and architecture courses
- understand the interaction of software and hardware
- understand the concepts of process, thread and address space
- understand how memory, devices (I/O), files and processes are managed
- understand the need for and implementation of concurrency control
- understand the design of widely used operating systems at a high level of abstraction

Recommended reading

For hardware/architecture please browse the books recommended for Part IA hardware courses (see `http://www.cl.cam.ac.uk/teaching/current/part1a50.html`) and for Computer Design.

* Bacon, J. & Harris, T. (2003). *Operating systems: distributed and concurrent software design*. Addison-Wesley.
Tanenbaum, A.S. & Woodhull, A. S. (2000). *Operating systems design and implementation*. Addison-Wesley (2nd ed.).
Silberschatz, A., Galvin, P.B. & Gagne, G. (2005, 2001). *Operating system concepts*. Addison-Wesley (7th, 6th eds.).

For further detail on the case studies (not required reading):

Bach, M.J. (1986). *Design of the Unix operating system*. Prentice Hall.
Leffler, S.J. *et al.* (1989). *The design of the 4.3BSD Unix operating system*. Addison-Wesley.
McKusick M.K. *et al.* (1996). *The design and implementation of the 4.4BSD Unix operating system*. Addison-Wesley.
Soloman D.A. & Russinovich, M.E. (2000). *Inside Microsoft Windows 2000*. Microsoft Press. (3rd ed.)

Software Engineering and Design

Lecturer: Dr A.F. Blackwell

No. of lectures: 12

This course is a prerequisite for the Group Project (Part II (General)).

Aims

This course aims to provide students with a practical set of skills for the development of usable, reliable and maintainable software systems. It assumes basic programming skills in Java, describing how these can be deployed in the construction of larger programs. Systematic design processes are described, suitable for use where a development team must create a product that will meet customers' needs. These include issues related to usability and relevance to the context of use. Some specialised problems related to safety-critical systems and real-time systems are discussed, with the help of case histories of software failure to illustrate what can go wrong. Finally a range of alternative planning and management methods for software projects are contrasted.

Lectures

- **Introduction.** The software crisis; why projects fail (London Ambulance case study); design and engineering perspectives.
- **Software construction.** Decomposition and modularity; coding style; naming; configuration; testing; efficiency. [2 lectures]
- **Object-oriented design.** Design as modelling; The Unified Modelling Language; use case analysis; class modelling; object interaction; state and activity descriptions. [2 lectures]
- **Interaction design.** Interaction styles; user models; requirements gathering; prototyping; evaluation. [3 lectures]
- **Design challenges.** Hazards; risk; reliability; management failure (CAPSA case study). [2 lectures]
- **Project management.** Models and metaphors of development; waterfall model, including technical review structure; evolutionary and incremental models (Spiral, RAD, RUP); novel structures (Chief programmer, Pair programming, XP). [2 lectures]

Objectives

At the end of the course, students should understand the ways in which writing programs that are usable, reliable and maintainable differs from the programming exercises they have engaged in so far. They should be able to undertake design of a moderately complex application in a systematic way: researching user requirements, decomposing and analysing system architecture, implementing that design so that it will be reliable and maintainable, and evaluating alternatives at various project phases from prototyping to commissioning. They should be able to select and manage appropriate organisational structures when asked to do so. They should be prepared for the organisational aspects of their Group Project (in the case of Part II (General) students) or the design elements of their project and dissertation (in the case of Diploma students).

Recommended reading

* Pressman, R.S. (2001). *Software engineering* (European ed.). McGraw-Hill.
* McConnell, S. (1993). *Code complete: a practical handbook of software construction.* Microsoft Press.
Fowler, M. (2000). *UML distilled.* Addison-Wesley (2nd ed.).
Preece, J., Sharp, H. & Rogers, Y. (2002). *Interaction design: beyond human-computer interaction.* Wiley.
Further reading:

Borenstein, N.S. (1991). *Programming as if people mattered.* Princeton.
Brooks, F.P. (1975). *The mythical man month.* Addison-Wesley.
Neumann, P. (1994). *Computer-related risks.* ACM Press.
Simon, H.A. (1996). *The sciences of the artificial.* MIT Press.
Schon, D.A. (1990). *Educating the reflective practitioner.* Jossey-Bass.
Finkelstein, A. (1993). *Report of the inquiry into the London Ambulance Service.* `http://www.cs.ucl.ac.uk/staff/A.Finkelstein/las.html`
Finkelstein, A. & Shattock, M. (2001). CAPSA and its implementation. *Cambridge University Reporter.*
`http://www.admin.cam.ac.uk/reporter/2001-02/weekly/5861/`

Lent Term 2008

Bioinformatics

Lecturer: Dr P. Liò

No. of lectures and examples classes: 12 + 2

Aims

The immense growth of biological information stored in databases has led to a critical need for people who can understand the languages and techniques of Computer Science and Biology. The first two lectures will focus on some aspects of Molecular Biology that are relevant to the course; other biological insights will be given as examples and applications in other lectures. This course will discuss algorithms for some important computational problems in Molecular Biology that are pertinent to Biotechnology and to the so-called "Post-genome era". We shall focus on Hidden Markov models and Phylogenetic inference algorithms for Comparative Genomics. Students will learn how to describe biological processes using Pi calculus and will be encouraged to think in two ways: How can Computer Science be useful to Biology? How can Biology be useful to Computer Science? In order to seed and to stimulate this approach, part of the course will be based on biological networks that offer many opportunities for comparing biological and electronics systems.

Lectures

- **DNA and protein sequence alignment.** Biology and information content of DNA and protein sequences. Why alignment is important. Dynamic programming. Global *versus* local. Scoring matrices. 4 Russians approach. [2 lectures]

- **Biological data mining.** Biological context. Algorithms related to Fasta, BLAST and its family, PatternHunter. Significance of alignments. [1 lecture]

- **Hidden Markov Models in Bioinformatics.** Biological context. Examples of the Viterbi, the Forward and the Backward algorithms. Parameter estimation for HMMs. [3 lectures]

- **Sequence comparison and evolution.** Biological context. Algorithms related to Distance, Parsimony and Likelihood methods; Bootstrap. Models of DNA and protein evolution. [2 lectures]

- **Introduction to microarray data analysis.** Biological context. Steady state and time series microarray data. Clustering methods. From microarray to networks. [2 lectures]

- **Introduction to biological networks.** Biological context. Basic networks topologies. Finding regulatory elements in aligned and unaligned sequences. Gillespie algorithm. Concept of System Biology. [2 lectures]

Objectives

At the end of this course students should

- understand the terminology of Bioinformatics, be aware of most common types of biomolecular data and their format. Be able to interface with biologists and physicians.
- understand important algorithms and data analysis techniques and be able to code them.

Recommended reading

* Durbin, R., Eddy, S., Krough, A. & Mitchison, G. (1998). *Biological sequence analysis: probabilistic models of proteins and nucleic acids*. Cambridge University Press.
Felsenstein, J. (2003). *Inferring phylogenies*. Sinauer Associates.
Fall, C., Marland, E., Wagner, J. & Tyson, J. (2002). *Computational cell biology*. Springer-Verlag Telos (1st ed.).
Bower, J.M. & Bolouri, H. (2001). *Computational modeling of genetic and biochemical networks*. MIT Press.

Compiler Construction

Lecturer: Professor A. Mycroft

No. of lectures: 18

This course is a prerequisite for Optimising Compilers (Part II).

Aims

This course aims to cover the main technologies associated with compiling programming languages, viz. lexical analysis, syntax analysis, type checking, run-time data organisation and code-generation.

Lectures

- **Survey of execution mechanisms.** The spectrum of interpreters and compilers; compile-time and run-time. Structure of a simple compiler. Java virtual machine (JVM), JIT. Simple run-time structures (stacks). Structure of interpreters for result of each stage of compilation (tokens, tree, bytecode). [3 lectures]
- **Lexical analysis and syntax analysis.** Regular expressions and finite state machine implementations. Phrase Structured Grammars. Chomsky classification. Parsing algorithms: recursive descent and SLR(k)/LALR(k). Syntax error recovery. Abstract syntax tree; expressions, declarations and commands. [3 lectures]
- **Simple type-checking.** Type of an expression determined by type of subexpressions; inserting coercions. [1 lecture]

- **Translation phase.** Intermediate code design. Translation of expressions, commands and declarations. [2 lectures]
- **Code generation.** Typical machine codes. Code generation from intermediate code. Simple peephole optimisation. [1 lecture]
- **Compiler compilers.** Compiler compilers. Summary of Lex and Yacc. [1 lecture]
- **Object Modules and Linkers.** Resolving external references. Static and dynamic linking. [1 lecture]
- **Non-local variable references** Lambda-calculus as prototype. Problems with `rec` and class variables. Environments, function values are closures. Static and Dynamic Binding (Scoping). Landin's principle of correspondence. [2 lectures]
- **Machine implementation of a selection of interesting things.** Free variable treatment, static and dynamic chains, ML free variables, closure conversion. Argument passing mechanisms. Objects and inheritance; implementation of methods. Labels, `goto` and exceptions. Dynamic and static typing, polymorphism. Storage allocation, garbage collection. [4 lectures]

Objectives

At the end of the course students should understand the overall structure of a compiler, and will know significant details of a number of important techniques commonly used. They will be aware of the way in which language features raise challenges for compiler builders.

Recommended reading

* Appel, A. (1997). *Modern compiler implementation in Java/C/ML* (3 editions). Cambridge University Press.
Aho, A.V., Sethi, R. & Ullman, J.D. (1986). *Compilers: principles, techniques and tools*. Addison-Wesley.
Bennett, J.P. (1990). *Introduction to compiling techniques: a first course using ANSI C, LEX and YACC*. McGraw-Hill.
Bornat, R. (1979). *Understanding and writing compilers*. Macmillan.
Fischer, C.N. & LeBlanc, J. Jr (1988). *Crafting a compiler*. Benjamin/Cummings.
Watson, D. (1989). *High-level languages and their compilers*. Addison-Wesley.

Computation Theory

Lecturer: Dr J.K.M. Moody

No. of lectures: 12

Prerequisite course: Discrete Mathematics or Mathematics for Computation Theory

This course is a prerequisite for Complexity Theory (Part IB and Diploma), Quantum Computing (Part II and Diploma).

Aims

The aim of this course is to introduce several apparently different formalisations of the informal notion of algorithm; to show that they are equivalent; and to use them to demonstrate that there are uncomputable functions and algorithmically undecidable problems.

Lectures

- **Introduction: algorithmically undecidable problems.** Decision problems. The informal notion of algorithm, or effective procedure. Examples of algorithmically undecidable problems. [1 lecture]

- **Register machines.** Definition and examples; graphical notation. Register machine computable functions. Doing arithmetic with register machines. [1 lecture]

- **Universal register machine.** Natural number encoding of pairs and lists. Coding register machine programs as numbers. Specification and implementation of a universal register machine. [2 lectures]

- **Undecidability of the halting problem.** Statement and proof. Example of an uncomputable partial function. Decidable sets of numbers; examples of undecidable sets of numbers. [1 lecture]

- **Turing machines.** Informal description. Definition and examples. Turing computable functions. Equivalence of register machine computability and Turing computability. The Church–Turing Thesis. [2 lectures]

- **Primitive recursive functions.** Definition and examples. Primitive recursive partial function are computable and total. [1 lecture]

- **Partial recursive functions.** Definition. Existence of a recursive, but not primitive recursive function. Ackermann's function. A partial function is partial recursive if and only if it is computable. [2 lectures]

- **Recursive and recursively enumerable sets.** Decidability and recursive sets. Generability and recursive enumeration. Example of a set that is not recursively enumerable. Example of a recursively enumerable set that is not recursive. Alternative characterisations of recursively enumerable sets as the images and the domains of definition of partial recursive functions. [2 lectures]

Objectives

At the end of the course students should

- be familiar with the register machine and Turing machine models of computability
- understand the notion of coding programs as data, and of a universal machine
- be able to use diagonalisation to prove the undecidability of the Halting Problem
- understand the mathematical notion of partial recursive function and its relationship to computability
- be able to develop simple mathematical arguments to show that particular sets are not recursively enumerable

Recommended reading

* Hopcroft, J.E., Motwani, R. & Ullman, J.D. (2001). *Introduction to automata theory, languages, and computation*. Addison-Wesley (2nd ed.).
Cutland, N.J. (1980). *Computability. An introduction to recursive function theory*. Cambridge University Press.
Davis, M.D., Sigal, R. & Weyuker E.J. (1994). *Computability, complexity and languages*. Academic Press (2nd ed.).
Sudkamp, T.A. (1995). *Languages and machines*. Addison-Wesley (2nd ed.).

Computer Graphics and Image Processing

Lecturer: Dr N.A. Dodgson

No. of lectures: 16

Prerequisite courses: Algorithms, Mathematical Methods for Computer Science (IB only, for one lecture of Image Processing part of the course)

This course is a prerequisite for Advanced Graphics (Part II).

Aims

To introduce the necessary background, the basic algorithms, and the applications of computer graphics and image processing. A large proportion of the course considers the design and optimisation of algorithms, so can be considered a practical application of the lessons learnt in the *Algorithms* course.

Lectures

- **Background.** What is an image? What are computer graphics, image processing, and computer vision? How do they relate to one another? Image capture. Image display. Human vision. Resolution and quantisation. Colour and colour spaces. Storage of images in memory, and double buffering. Display devices: the inner workings of CRTs, LCDs, and printers. [3 lectures]

- **2D Computer graphics.** Drawing a straight line. Drawing circles and ellipses. Cubic curves: specification and drawing. Clipping lines. Filling polygons. Clipping polygons. 2D transformations, vectors and matrices, homogeneous co-ordinates. Uses of 2D graphics: HCI, typesetting, graphic design. [5 lectures]
- **3D Computer graphics.** Projection: orthographic and perspective. 3D transforms and matrices. 3D clipping. 3D curves. 3D scan conversion. z-buffer. A-buffer. Ray tracing. Lighting: theory, flat shading, Gouraud, Phong. Texture mapping. [5 lectures]
- **Image processing.** Operations on images: filtering, point processing, compositing. Halftoning and dithering, error diffusion. Encoding and compression: difference encoding, predictive, run length, transform encoding (including JPEG). [3 lectures]

Objectives

At the end of the course students should be able to

- explain the basic function of the human eye and how this impinges on resolution, quantisation, and colour representation for digital images; describe a number of colour spaces and their relative merits; explain the workings of cathode ray tubes, liquid crystal displays, and laser printers
- describe and explain the following algorithms: Bresenham's line drawing, mid-point line drawing, mid-point circle drawing, Bezier cubic drawing, Douglas and Pucker's line chain simplification, Cohen–Sutherland line clipping, scanline polygon fill, Sutherland–Hodgman polygon clipping, depth sort, binary space partition tree, z-buffer, A-buffer, ray tracing, error diffusion
- use matrices and homogeneous coordinates to represent and perform 2D and 3D transformations; understand and use 3D to 2D projection, the viewing volume, and 3D clipping
- understand Bezier curves and patches; understand sampling and super-sampling issues; understand lighting techniques and how they are applied to both polygon scan conversion and ray tracing; understand texture mapping
- explain how to use filters, point processing, and arithmetic operations in image processing and describe a number of examples of the use of each; explain how halftoning, ordered dither, and error diffusion work; understand and be able to explain image compression and the workings of a number of compression techniques

Recommended reading

* Foley, J.D., van Dam, A., Feiner, S.K. & Hughes, J.F. (1990). *Computer graphics: principles and practice*. Addison-Wesley (2nd ed.).
Gonzalez, R.C. & Woods, R.E. (1992). *Digital image processing*. Addison-Wesley. [Gonzalez, R.C. & Wintz, P. (1977). *Digital image processing* is the earlier edition and is almost as useful.]

* Slater, M., Steed, A. & Chrysanthou, Y. (2002). *Computer graphics and virtual environments: from realism to real-time*. Addison-Wesley.

Concepts in Programming Languages

Lecturer: Dr M.P. Fiore

No. of lectures: 8

Aims

The general aim of this course is to provide an overview of the basic concepts that appear in modern programming languages, the principles that underlie the design of programming languages, and their interaction.

Lectures

- **Introduction, motivation, and overview.** What is a programming language? Application domains in language design. Program execution models. Theoretical foundations. Language standardization. History.
- **The first procedural language: FORTRAN (1954–58).** Execution model. Data types. Control structures. Storage. Subroutines and functions. Parameter passing.
- **The first declarative language: LISP (1958–62).** Expressions, statements, and declarations. S-expressions and lists. Recursion. Static and dynamic scope. Abstract machine. Garbage collection. Programs as data. Parameter passing. Strict and lazy evaluation.
- **Block-structured procedural languages: Algol (1958–68), BCPL (1967), Pascal (1970), C (1971–78).** Block structure. Parameters and parameter passing. Stack and heap storage. Data types. Arrays and pointers.
- **Object-oriented languages — Concepts and origins: Simula (1964–67), Smalltalk (1971–80).** Dynamic lookup. Abstraction. Subtyping. Inheritance. Object models.
- **Types, data abstraction, and modularity: C++ (1983–98), SML (1984–97).** Types in programming. Type systems. Type checking and type inference. Polymorphism. Overloading. Type equivalence. Information hiding. Modularity. SML module system: signatures, structures, and functors. [2 lectures]
- **State of the art.**

Objectives

At the end of the course students should

- be familiar with several language paradigms and how they relate to different application domains

- understand the design space of programming languages, including concepts and constructs from past languages as well as those that may be used in the future

- develop a critical understanding of the programming languages that we use by being able to identify and compare the same concept as it appears in different languages

Recommended reading

Books:

* Mitchell, J.C. (2003). *Concepts in programming languages.* Cambridge University Press.
* Pratt, T.W. & Zelkowitz, M.V. (2001). *Programming languages: design and implementation.* Prentice Hall.

Papers:

Kay, A.C. (1993). The early history of Smalltalk. *ACM SIGPLAN Notices*, Vol. 28, No. 3.
Kerninghan, B. (1981). Why Pascal is not my favorite programming language. AT&T Bell Laboratories. *Computing Science Technical Report* No. 100.
Koenig, A. (1994). An anecdote about ML type inference. *USENIX Symposium on Very High Level Languages.*
McCarthy, J. (1960). Recursive functions of symbolic expressions and their computation by machine. *Communications of the ACM*, 3(4):184–195.
Stroustrup, B. (1991). What is Object-Oriented Programming? (1991 revised version). *Proceedings 1st European Software Festival.*

Digital Communication I

Lecturer: Dr A.W. Moore

No. of lectures: 12

This course is a prerequisite for Digital Communication II (Part II), Distributed Systems (Part II and Diploma) and Security (Part II).

Aims

The aims of this course are to develop an understanding of communications networks from a wide perspective, within a framework of principles rather than technologies or architectures. Technologies and architectures will, however, be used as examples and motivation.

Lectures

- **Scope.** Two example systems: Ethernet and the telephone system: basic operation; common issues; differing constraints; differing approaches.

- **Partitioning the problem.** Abstraction, service *versus* implementation; layering as a restricted form of abstraction; motivation for layering; the channel as an abstraction; layered channels.

- **Fundamental transmission.** Emphasis on the service provided by physical channel; limitations: noise, attenuation. Channel capacity (bandwidth). Modulation techniques for digital systems.

- **Coding.** Coding as a general concept: modulation as a form of coding, A/D,D/A, error correcting and detecting codes, other forms of coding, relation to layering.

- **Multiplexing.** Basic definitions, FDM, synchronous and asynchronous TDM. Circuit switching, packet switching, ATM. Shared media networks with particular emphasis on media access control. Packet scheduling. Non orthogonal multiplexing. Multiplexing and channel characteristics.

- **Switching and routing.** Introduction from LAN perspective (repeaters, bridges, routers). Fundamental view of switching extended to telephone network, connectionless *versus* connection oriented.

- **Protocols and state.** Imperfect view of state at far end of channel. ARQ as an example of an error control protocol; sliding window ARQ as an example of a flow control protocol; flow control in general: X.25 as an example.

- **Naming, addressing and routing.** Service access points, binding. Hierarchical *versus* flat address spaces. Routing classifications and algorithms.

- **The Internet.** Internet architecture, context of development, addressing and routing, transmission control protocol (TCP), higher layer protocols, evolution.

- **Standards.** Role of standards, dynamics of standards process, standards bodies.

Objectives

At the end of the course students should

- be able to analyse a communication system by separating out the different functions provided by the network

- understand that there are fundamental limits to physical transmission systems

- understand the general principles behind multiplexing, addressing, routing and stateful protocols as well as specific examples of each

- understand what FEC is and how CRCs work

- be able to compare communications systems in how they solve similar problems

- have an informed view of the internal workings of the Internet

Recommended reading

* Peterson, L.L. & Davie, B.S. (2007). *Computer networks: a systems approach*. Morgan Kaufmann (4th ed.).
Comer, D. & Stevens, D. (1995). *Internetworking with TCP-IP, vol.* 1 *and* 2. Prentice Hall (3rd ed.).
Schwartz, M. (1987). *Telecommunication networks: protocols, modeling and analysis*. Addison-Wesley.

Introduction to Functional Programming

Lecturer: Dr M.P. Fiore

No. of lectures: 12

Aims

The general aim of the course is to introduce the functional style of programming using the programming language Standard ML (SML). Specifically, the course will introduce, illustrate, and examine the principles of functional programming using key features of SML: structured datatypes, higher-order functions, and type inference. Throughout the course applications will be demonstrated via case studies.

Lectures

- **Overview and motivation.** Functional programming. Expressions and values. Functions. Recursion. Types.
- **Introduction to SML.** SML/NJ and Moscow ML. Value declarations. Static binding. Basic types: integers, reals, truth values, characters, strings. Function declarations. Overloading. Types. Recursion. Expression evaluation. Call-by-value.
- **Functions and lists.** Types. Polymorphism. Curried functions. Nameless functions. Lists. Pattern matching. Case expressions. List manipulation. Tail recursion. Accumulators. Local bindings.
- **Higher-order functions.** Higher-order functions. List functionals.
- **Sorting.** Insertion sort, quick sort, merge sort. Parametric sorting. Queues. Signatures. Structures. Functors. Generic sorting.
- **Datatypes.** Records. Enumerated types. Polymorphic datatypes. Option type. Disjoint-union type. Abstract types. Error handling. Exceptions.
- **Recursive datatypes.** Lists, trees, lambda calculus. Tree manipulation. Tree listings: preorder, inorder, postorder. Tree exploration: breadth-first and depth-first search. Polymorphic exceptions. Isomorphisms.
- **Data structures.** Tree-based data structures. Binary search trees. Red/black trees. Flexible functional arrays. Heaps. Priority queues.

- **Lazy datatypes.** Call-by-value, call-by-name, and call-by-need evaluation. Sequences, streams, trees. Lazy evaluation. Sieve of Eratosthenes. Breadth-first and depth-first traversals.
- **Program specification and verification.** Testing and verification. Rigorous and formal proofs. Structural induction on lists. Law of extensionality. Multisets. Structural induction on trees.

Objectives

At the end of the course students should

- be familiar with key concepts of programming in a functional style
- be able to develop software in SML in a competent manner
- understand how to use the typing discipline for clearer and verifiable programs

Recommended reading

Books:

* Paulson, L.C. (1996). *ML for the working programmer.* Cambridge University Press.
* Okasaki, C. (1998). *Purely functional data structures.* Cambridge University Press.

Papers:

Backus, J. (1978). Can programming be liberated from the von Neumann style? A functional style and its algebra of programs. *Communications of the ACM*, vol. 21, pp. 613–641.
Landin, P.J. (1966). The next 700 programming languages. *Communications of the ACM*, vol. 9, pp. 157–166.

Programming in C and C++

Lecturer: Dr A.R. Beresford

No. of lectures: 8

Prerequisite courses: None, though Operating Systems would be helpful.

Aims

The aims of this course are to provide a solid introduction to programming in C and C++ and to provide an overview of the principles and constraints that affect the way in which the C and C++ programming languages have been designed and are used.

Lectures

- **Introduction to the C language.** Background and goals of C. Types and variables. Expressions and statements. Functions. Multiple compilation units. [1 lecture]

- **Further C concepts.** Preprocessor. Pointers and pointer arithmetic. Data structures. Dynamic memory management. Examples. [2 lectures]
- **Introduction to C++.** Goals of C++. Differences between C and C++. References *versus* pointers. Overloading functions. [1 lecture]
- **Objects in C++.** Classes and structs. Operator overloading. Virtual functions. Multiple inheritance. Virtual base classes. Examples. [2 lectures]
- **Further C++ concepts.** Exceptions. Templates and meta-programming. STL generic programming. Examples. [2 lectures]

Objectives

At the end of the course students should

- be able to read and write C and C++ programs
- understand the interaction between C and C++ programs and the host operating system
- be familiar with the structure of C and C++ program execution in machine memory
- understand the object-oriented paradigm presented by C++
- be able to make effective use of templates and meta-programming techniques as used in the STL
- understand the potential dangers of writing programs in C and C++

Recommended reading

* Eckel, B. (2000). *Thinking in C++, Vol. 1: Introduction to Standard C++.* Prentice Hall (2nd ed.). Also available at `http://www.mindview.net/Books/TICPP/ThinkingInCPP2e.html`
Kernighan, B.W. & Ritchie, D.M. (1988). *The C programming language.* Prentice Hall (2nd ed.).
Stroustrup, B. (1994). *The design and evolution of C++.* Addison-Wesley.
Lippman, S.B. (1996). *Inside the C++ object model.* Addison-Wesley.

Quantum Computing

Lecturer: Dr A. Dawar

No. of lectures: 8

Prerequisite courses: Mathematical Methods for Computer Science or Mathematics for Computation Theory, Computation Theory

Aims

The aims of the course are to introduce students to the basics of the quantum model of computation. The model will be used to study algorithms for searching and factorisation. Issues in the complexity of computation will also be explored.

Lectures

- **Bits and qubits.** Introduction to quantum states with motivating examples. Comparison with classical discrete state systems.
- **Linear algebra.** Review of linear algebra. Vector spaces, linear operators, Dirac notation.
- **Quantum mechanics.** Postulates of quantum mechanics. Evolution and measurement. Entanglement.
- **Quantum Computation.** Models of quantum computation. Quantum circuits, finite state systems, machines and algorithms.
- **Some Applications.** Applications of quantum infomation. Bell States, quantum key exchange, quantum teleportation.
- **Quantum search.** Grover's search algorithm. Analysis and lower bounds.
- **Factorisation.** Shor's algorithm for factorising numbers and analysis. Quantum Fourier transform.
- **Quantum complexity.** Quantum complexity classes and their relationship to classical complexity. Comparison with probabilistic computation.

Objectives

At the end of the course students should

- understand the quantum model of computation and how it relates to quantum mechanics
- be familiar with some basic quantum algorithms and their analysis
- see how the quantum model relates to classical models of computation

Recommended reading

* Nielsen, M.A. & Chuang, I.L. (2000). *Quantum computation and quantum information*. Cambridge University Press.
Mermin, N.D. (2007). *Quantum computer science*. Cambridge University Press.
Gruska, J. (1999). *Quantum computing*. McGraw-Hill (now out of print, but try a library).
Kitaev, A.Y., Shen, A.H. & Vyalyi, M.N. (2002). *Classical and quantum computation*. AMS.

Easter Term 2008

Artificial Intelligence I

Lecturer: Dr S.B. Holden

No. of lectures: 12

Prerequisite courses: Algorithms I + II (CST students) or Data Structures and Algorithms (Part II (General)/Diploma students). In addition the course requires some mathematics, in particular some use of vectors and some calculus. For Part II (General) and Diploma students therefore Mathematics for Computation Theory is desirable. For CST students Part IA Natural Sciences Mathematics or equivalent, and Discrete Mathematics I + II, are likely to be helpful although not essential.

This course is a prerequisite for Artificial Intelligence II (Part II).

Aims

The aim of this course is to provide an introduction to some basic issues and algorithms in artificial intelligence (AI). The course approaches AI from an algorithmic, computer science-centric perspective; relatively little reference is made to the complementary perspectives developed within psychology, neuroscience or elsewhere. The course aims to provide some basic tools and algorithms required to produce AI systems able to exhibit limited human-like abilities, particularly in the form of problem solving by search, representing and reasoning with knowledge, planning, and learning.

Lectures

- **Introduction.** What is it that we're studying? Why is something that looks so easy to do actually so difficult to compute? Theories and methods: what approaches have been tried? What does this course cover, and what is left out?

- **Agents.** A unifying view of AI systems. How could we approach the construction of such a system? How would we judge an AI system? What should such a system do and how does it interact with its environment?

- **Search I.** How can search serve as a fundamental paradigm for intelligent problem-solving? Simple, *uninformed search* algorithms.

- **Search II.** More sophisticated *heuristic search* algorithms.

- **Search III.** Search in an adversarial environment. Computer game playing.

- **Constraint satisfaction problems.**

- **Knowledge representation and reasoning.** How can we represent and deal with commonsense knowledge and other forms of knowledge? Semantic networks, frames and rules. How can we use inference in conjunction with a knowledge representation scheme to perform reasoning about the world and thereby to solve problems? Inheritance, forward and backward chaining.

- **Planning.** Methods for planning in advance how to solve a problem. The partial-order planning algorithm.
- **Learning.** A brief introduction to supervised learning from examples, focusing on neural networks. [4 lectures]

Objectives

At the end of the course students should

- Appreciate the distinction between the popular view of the field and the actual research results.
- Appreciate different perspectives on what the problems of artificial intelligence are and how different approaches are justified.
- Be able to design basic problem solving methods based on AI-based search, reasoning, planning, and learning algorithms.

Recommended reading

* Russell, S. & Norvig, P. (2003). *Artificial intelligence: a modern approach.* Prentice Hall (2nd ed.).
Cawsey, A. (1998). *The essence of artificial intelligence.* Prentice Hall.
Luger, G.F. & Stubblefield, W.A. (1998). *Artificial intelligence: structures and strategies for complex problem solving.* Addison-Wesley.
Dean, T., Allen, J. & Aloimonos, Y. (1995). *Artificial intelligence: theory and practice.* Benjamin/Cummings.

Business Studies Seminars

Lecturer: Mr J.A. Lang and others

No. of lectures: 6

Aims

This course is a series of seminars by former members and friends of the Laboratory about their real world experiences of starting and running high technology companies. It is a follow on to the Business Studies course in the Michaelmas Term. It provides practical examples and case studies, and the opportunity to network with and learn from actual entrepreneurs.

Lectures

Six lectures by six different entrepreneurs.

Objectives

At the end of the course students should have a better knowledge of the pleasures and pitfalls of starting a high tech company.

Recommended reading

Lang, J. (2001). *The high-tech entrepreneur's handbook: how to start and run a high-tech company*. FT.COM/Prentice Hall.

See also the additional reading list on the Part II Business Studies web page.

Complexity Theory

Lecturer: Dr A. Dawar

No. of lectures: 12

Prerequisite courses: Algorithms, Computation Theory

Aims

The aim of the course is to introduce the theory of computational complexity. The course will explain measures of the complexity of problems and of algorithms, based on time and space used on abstract models. Important complexity classes will be defined, and the notion of completeness established through a thorough study of NP-completeness. Applications to cryptography will be considered.

Lectures

- **Algorithms and problems.** Complexity of algorithms and of problems. Lower and upper bounds. Examples: sorting and travelling salesman.
- **Time and space.** Models of computation and measures of complexity. Time and space complexity on a Turing machine. Decidability and complexity.
- **Time complexity.** Time complexity classes. Polynomial time problems and algorithms. P and NP.
- **Non-determinism.** Non-deterministic machines. The class NP redefined. Non-deterministic algorithms for reachability and satisfiability.
- **NP-completeness.** Reductions and completeness. NP-completeness of satisfiability.
- **More NP-complete problems.** Graph-theoretic problems. Hamiltonian cycle and clique.
- **More NP-complete problems.** Sets, numbers and scheduling. Matching, set covering and bin packing.
- **coNP.** Validity of boolean formulas and its completeness. NP$\cap$coNP. Primality and factorisation.
- **Cryptographic complexity.** One-way functions. The class UP.
- **Space complexity.** Deterministic and non-deterministic space complexity classes. The reachability method. Savitch's theorem.

- **Hierarchy.** The time and space hierarchy theorems and complete problems.
- **Descriptive Complexity.** Logics capturing complexity classes. Fagin's theorem.

Objectives

At the end of the course students should

- be able to analyse practical problems and classify them according to their complexity
- be familiar with the phenomenon of NP-completeness, and be able to identify problems that are NP-complete
- be aware of a variety of complexity classes and their interrelationships
- understand the role of complexity analysis in cryptography

Recommended reading

* Papadimitriou, Ch.H. (1994). *Computational complexity.* Addison-Wesley.
Sipser, M. (1997). *Introduction to the theory of computation.* PWS.

Databases

Lecturer: Dr T.G. Griffin

No. of lectures: 12

Aims

The overall aim of the course is to cover the fundamentals of database management systems (DBMSs), paying particular attention to relational database systems. The course covers modelling techniques, transferring designs to actual database implementations, SQL, models of query languages, transactions as well as more recent developments, including data warehouses and On-line Analytical Processing (OLAP), and use of XML as a data exchange language. The lectures will make use of the open source DBMS, MySQL.

Lectures

- **Introduction.** What is a database system? Database systems are more than just a collection of data. Three level architecture. OnLine Transaction Processing (OLTP) *vs.* OnLine Analytic Processing (OLAP).
- **Entity-Relationship (E/R) modelling.** A bit of set theory. Entities have attributes. Relations have *arity*. Database design and data modelling.
- **The relational data model.** Relations are sets of records. Representing entities and relationships as relations. Queries as derived relations. Relations are the basis of SQL.

- **Relational algebra.** Relational algebra as an abstract query language. Core operations – selection, projection, product, renaming, and joins.
- **Relational calculus.** Relational calculus as an abstract query language that uses notation from set theory. Equivalence with relational algebra.
- **SQL and integrity constraints.** An overview of the core of SQL. SQL has constructs taken from both the relational algebra and the relational calculus. Integrity constraints as special queries.
- **Schema refinement I.** The evils of redundancy. The benefits of redundancy. Functional dependencies as a formal means of investigating redundancy. Relational decomposition. Armstrong's axioms.
- **Schema refinement II.** Schema normalisation. Lossless-join decomposition. Dependency preservation. Boyce–Codd normal form. Third normal form.
- **Further relational algebra, SQL.** SQL is really based on multi-sets (bags). Extending the relational algebra to bags. NULL values as an SQL design error?
- **Transaction management overview.** ACID properties – Atomicity, Consistency, Isolation, and Durability. Serialisability in the database context.
- **On-line Analytical Processing (OLAP).** When to forget about data normalisation. Beware of buzz-words and the Data Warehouse Death March. More on OLTP *vs.* OLAP. What is a *data cube*? Data modelling for data warehouses — *star schema*.
- **XML as a data exchange format.** What is XML? XML can be used to share data between proprietary relational databases. XML-based databases?

Objectives

At the end of the course students should

- be able to design entity-relationship diagrams to represent simple database application scenarios
- know how to convert entity-relationship diagrams to relational database schemas in the standard Normal Forms
- be able to program simple database applications in SQL
- understand the basic theory of the relational model and both its strengths and weaknesses
- be familiar with various recent trends in the database area

Recommended reading

* Date, C.J. (2000). *An introduction to database systems*. Addison-Wesley (7th ed.).

Elmasri, R. & Navathe, S.B. (2000). *Fundamentals of database systems.* Addison-Wesley (3rd ed.).
Silberschatz, A., Korth, H.F. & Sudarshan, S. (2002). *Database system concepts.* McGraw-Hill (4th ed.).
Ullman, J. & Widom, J. (1997). *A first course in database systems.* Prentice Hall.
Miszczyk, J. and others (1998). *Mastering Data Warehousing Functions.* (IBM Redbook DB2/400) Chapters 1 & 2 only.
`http://www.redbooks.ibm.com/abstracts/sg245184.html`
Garcia-Molina, H. *Data Warehousing and OLAP.* Stanford University.
`http://www.cs.uh.edu/ ceick/6340/dw-olap.ppt`
London Metropolitan University, Department of Computing. *Data Warehousing and OLAP Technology for Data Mining.*
`http://learning.unl.ac.uk/csp002n/CSP002N_wk2.ppt`

Distributed Systems

Lecturer: Professor J.M. Bacon

No. of lectures: 8

Prerequisite courses: Concurrent Systems and Applications (if available), Operating Systems, Digital Communication I, Introduction to Security

Aims

The aims of this course are to study the fundamental characteristics of distributed systems and their implications on software design; their models and architectures; the design of distributed algorithms and applications.

Lectures

- **Introduction.** Characteristics specific to distributed systems. Software structure. Modelling, architecting and engineering distributed software. Issues of scale and trust.

- **Time.** Physical and logical time. Event ordering. Clock synchronisation. Message delivery ordering.

- **Algorithms and application protocols.** Replication management. Strong and weak consistency. Asynchronous and synchronous algorithms. Atomic commitment. Process groups. Election. Mutual exclusion.

- **Communication.** Overview of synchronous, asynchronous and event-based middleware. Web services.

- **Naming.** Design of names, pure or hierarchical. Interpretation of names in context. Binding. Long-term consistency.

- **Access control.** ACLs and capabilities in distributed systems. Role-based access control. Policy expression and enforcement.

- **Storage.** Design issues for network-based storage services.
- **Applications.** Pervasive computing environments: active office, home and city. Events, composite events, mobility and location-tracking. Electronic health, police and transport services.

Objectives

At the end of the course students should

- understand the fundamental properties of distributed systems and how to design software to accommodate them
- understand some basic distributed algorithms and the assumptions on which they are based
- for widely distributed and/or large scale systems, understand how naming and access control can be designed
- understand the tradeoffs involved in selecting various styles of middleware
- be aware of some emerging application areas, such as pervasive and mobile computing.

Recommended reading

* Bacon, J. & Harris, T. (2003). *Operating systems: distributed and concurrent software design*. Addison-Wesley.
Tanenbaum, A.S. & van Steen, M. (2002). *Distributed systems*. Prentice Hall.
Coulouris, G.F., Dollimore, J.B. & Kindberg, T. (2005, 2001). *Distributed systems, concepts and design*. Addison-Wesley (4th, 3rd eds.).
Mullender, S. (ed.) (1993). *Distributed systems*. Addison-Wesley (2nd ed.).

Introduction to Security

Lecturer: Dr M.G. Kuhn

No. of lectures: 8

Prerequisite courses: Discrete Mathematics or Mathematics for Computation Theory, Operating Systems

This course is a prerequisite for Distributed Systems (Part II and Diploma) and Security (Part II).

Aims

This course is a broad introduction to both computer security and cryptography. It covers important basic concepts and techniques.

Lectures

- **Cryptography.** Introduction, terminology, classic ciphers, perfect secrecy, Vernam cipher, pseudo-random functions and permutations, computational security, random bit generation, secure hash functions, birthday problem.
- **Symmetric cryptography.** Block ciphers, modes of operation, message authentication codes, applications of secure hash functions.
- **Asymmetric cryptography.** Key-management problem, signatures and certificates, number theory revisited, discrete logarithm problem, Diffie–Hellman key exchange, ElGamal encryption and signature, hybrid cryptography.
- **Authentication techniques.** Passwords, one-way and challenge–response protocols, Needham–Schroeder, protocol failure examples, hardware tokens.
- **Access control.** Discretionary access control in POSIX and Windows, elevated rights and setuid bits, capabilities, mandatory access control, covert channels, Clark–Wilson integrity.
- **Operating system security.** OS security functions, trusted computing base, security evaluation methodology and standards.
- **Software security.** Malicious software, viruses, common implementation vulnerabilities, buffer overflows, meta characters, integer overflows, race conditions, side channels.
- **Network security.** TCP/IP vulnerabilities, firewalls. [0.5 lecture]
- **Security policies and management.** Application-specific security requirements, targets and policies, security management. [0.5 lecture]

Objectives

By the end of the course students should

- be familiar with the most common security terms and concepts
- have a basic understanding of the most commonly used attack techniques and protection mechanisms
- have gained basic insight into aspects of modern cryptography and its applications
- appreciate the range of meanings that “security” has across different applications

Recommended reading

* Gollmann, D. (2006). *Computer Security*. Wiley.
Stinson, D. (2005). *Cryptography: theory and practice*. Chapman & Hall/CRC (3rd ed.).

Further reading:

Anderson, R. (2001). *Security engineering: a guide to building dependable distributed systems*. Wiley.
Schneier, B. (1995). *Applied cryptography: protocols, algorithms, and source code in C*. Wiley (2nd ed.).
Cheswick, W.R., Bellovin, S.M. & Rubin, A.D. (2003). *Firewalls and Internet security: repelling the wily hacker*. Addison-Wesley (2nd ed.).
Garfinkel, S., Spafford, G. & Schwartz, A. (2003). *Practical Unix and Internet security*. O’Reilly (3nd ed.).