Michaelmas Term 2016: Part IA lectures

Paper 1: Foundations of Computer Science

Lecturer: Professor L.C. Paulson

No. of lectures and practicals: 12 + 5 (NST and PBST students will take 4 practicals)

Suggested hours of supervisions: 3 to 4

This course is a prerequisite for Programming in Java and Prolog (Part IB).

Aims

The main aim of this course is to present the basic principles of programming. As the introductory course of the Computer Science Tripos, it caters for students from all backgrounds. To those who have had no programming experience, it will be comprehensible; to those experienced in languages such as C, it will attempt to correct any bad habits that they have learnt.

A further aim is to introduce the principles of data structures and algorithms. The course will emphasise the algorithmic side of programming, focusing on problem-solving rather than on hardware-level bits and bytes. Accordingly it will present basic algorithms for sorting, searching, etc., and discuss their efficiency using O-notation. Worked examples (such as polynomial arithmetic) will demonstrate how algorithmic ideas can be used to build efficient applications.

The course will use a functional language (ML). ML is particularly appropriate for inexperienced programmers, since a faulty program cannot crash. The course will present the elements of functional programming, such as curried and higher-order functions. But it will also introduce traditional (procedural) programming, such as assignments, arrays and references.

Lectures

- **Introduction to Programming.** The role of abstraction and representation. Introduction to integer and floating-point arithmetic. Declaring functions. Decisions and booleans. Example: integer exponentiation.
- **Recursion and Efficiency.** Examples: Exponentiation and summing integers. Overloading. Iteration *versus* recursion. Examples of growth rates. Dominance and O-Notation. The costs of some representative functions. Cost estimation.
- **Lists.** Basic list operations. Append. Naïve *versus* efficient functions for length and reverse. Strings.
- **More on lists.** The utilities `take` and `drop`. Pattern-matching: zip, unzip. A word on polymorphism. The “making change” example.
- **Sorting.** A random number generator. Insertion sort, mergesort, quicksort. Their efficiency.

- **Datatypes and trees.** Pattern-matching and case expressions. Exceptions. Binary tree traversal (conversion to lists): preorder, inorder, postorder.
- **Dictionaries and functional arrays.** Functional arrays. Dictionaries: association lists (slow) *versus* binary search trees. Problems with unbalanced trees.
- **Functions as values.** Nameless functions. Currying. The "apply to all" functional, `map`. *Examples*: matrix transpose and product. The predicate functionals `filter` and `exists`.
- **Sequences, or lazy lists.** Non-strict functions such as *IF*. Call-by-need *versus* call-by-name. Lazy lists. Their implementation in ML. Applications, for example Newton-Raphson square roots.
- **Queues and search strategies.** Depth-first search and its limitations. Breadth-first search (BFS). Implementing BFS using lists. An efficient representation of queues. Importance of efficient data representation.
- **Polynomial arithmetic.** Addition, multiplication of polynomials using ideas from sorting, etc.
- **Elements of procedural programming.** Address *versus* contents. Assignment *versus* binding. Own variables. Arrays, mutable or not. Introduction to linked lists.

Objectives

At the end of the course, students should

- be able to write simple ML programs;
- understand the fundamentals of using a data structure to represent some mathematical abstraction;
- be able to estimate the efficiency of simple algorithms, using the notions of average-case, worse-case and amortised costs;
- know the comparative advantages of insertion sort, quick sort and merge sort;
- understand binary search and binary search trees;
- know how to use currying and higher-order functions;
- understand how ML combines imperative and functional programming in a single language.

Recommended reading

* Paulson, L.C. (1996). *ML for the working programmer.* Cambridge University Press (2nd ed.).
Okasaki, C. (1998). *Purely functional data structures.* Cambridge University Press.

For reference only:
Gansner, E.R. & Reppy, J.H. (2004). *The Standard ML Basis Library.* Cambridge University Press. ISBN: 0521794781

Paper 1: Object-Oriented Programming

Lecturer: Dr R.K. Harle, Dr A.C. Rice and Dr S. Cummins

No. of lectures and practicals: 12 + 5

Suggested hours of supervisions: 3 to 4

Aims

The goal of this course is to provide students with the ability to write programs in Java and make use of the concepts of Object-Oriented Programming. Examples and discussions will use Java primarily, but other languages may be used to illustrate specific points where appropriate. The course is designed to accommodate students with diverse programming backgrounds; it is taught with a mixture of lectures and practical sessions where students can work at their own pace from a course handbook. Each practical class will culminate in an assessed exercise.

Lecture syllabus

- **Types, Objects and Classes** Moving from functional to imperative. Distinguishing state and behaviour. Primitive types. Function prototypes. Objects and classes as custom types. Introduction to parameterised types (templates/Generics).
- **Pointers, References and Memory** Pointers and references. The call stack and heap. Iteration and recursion. Pass-by-value and pass-by-reference. Objects as reference types in Java.
- **Creating Classes** Modularity. Encapsulation. Information hiding. Access modifiers. Advantages of immutability. Creating Generic types in Java. Static data.
- **Inheritance** Inheritance. Casting. Shadowing. Overloading. Overriding. Abstract Methods and Classes.
- **Polymorphism and Multiple Inheritance** Polymorphism in ML and Java. Multiple inheritance. Interfaces in Java.