Compilers – January 2009

1 Aims, objectives and prerequisites

The aim of this course is to examine some aspects of the design and implementation of high-level languages. The course briefly outlines some of the more important aspects of high-level language design, but the main part of the course is devoted to the study of compiler construction. The course assumes reasonable programming skills in a language such as Java, C or C++.

At the end of the course, students should be able to

- identify the key differences between high-level languages and understand how these differences may influence their implementations
- understand the structure of a typical compiler
- understand the introductory theory and practice behind high-level language design, specification and implementation and understand some of the standard techniques for the implementation of the modules making up a compiler
- implement a complete compiler for a simple source language and a simple target architecture

2 Syllabus

Low-level versus high-level languages. Introduction to language implementation techniques, compilers and interpreters, grammars and parsing.

Lexical analysis – relevance of finite-state automata. Regular grammars. Implementation techniques, automated production of lexical analyzers. Problems for particular languages.

Syntax analysis – overview of grammars and parsing techniques. Top-down and bottom-up parsing. Predictive parsing, shift-reduce parsing, precedence parsing. Implementing hand-coded top-down predictive parsers. Introduction to LR parsing. Automated production of parsers.

Semantic analysis and code generation – from trees and from flat intermediate codes. Symbol tables. Type checking. Handling of specific high-level language constructs. Runtime storage allocation, scoping. Instruction set consequences. Code optimisation, introduction to flow analysis. Implementation techniques. Portability.

3 Teaching and Assessment

This is a one term course, taught in the spring term of the second or third year. There are two lectures per week together with exercise classes and programming work for the practical exercises. I have a weekly office hour (1000–1200 each Tuesday in term, Pevensey 3, 5C9) – feel free to come along to ask questions about this course. Also, feel free to email me (`desw@sussex.ac.uk`) if you want to ask questions or to book a time to see me.

The course is assessed by coursework (25%) and by unseen examination (75%) in June 2009. The coursework consists of three parts (weighted 5%, 5% and 15%, respectively), The first is a programming exercise, the second is a peer assessment of the first programming exercise and the third is another programming exercise (an extension of the first). The first is due on Thursday 12th February 2009 (week 5), the second on Thursday 19th February (week 6) and the third on Thursday 19th March (week 10).

The usual lateness penalties apply. Any coursework submitted up to 24 hours after the deadline will be subject to a penalty of a loss of 10 percentage points; any work submitted later than 24 hours after the deadline will receive no marks. All cases of collusion and plagiarism will be reported to the Misconduct Panel.

4 Lecture and exercise class programme

Lectures are held on Tuesdays at 1200 in 2A1 and Thursdays at 1100 also in 2A1. There is an exercise class at 1400 on Fridays in 2A3.

Here is the proposed timetable for the course – but I can't promise to stick to it precisely!

Week 1	13 Jan	L	Introduction to the course. Motivation for the use of high-level languages.
	15 Jan	L	Comparing high-level languages.
	16 Jan	E	—
Week 2	20 Jan	L	Comparing high-level languages. Approaches to language implementation.
	22 Jan	L	Structure of compilers. Methods of syntax description.
	23 Jan	E	Choosing high-level languages.
Week 3	27 Jan	L	Grammars. Chomsky classification. Parsing, problems of parsing.
	29 Jan	L	Lexical analysis.
	30 Jan	E	Coursework.
Week 4	3 Feb	L	Finite-state automata. DFA's and NFA's. Programming a lexical analyser.
	5 Feb	L	Introduction to syntax analysis.
	6 Feb	E	Coursework – implementation and testing.
Week 5	10 Feb	L	Comparing top-down and bottom-up parsers. Problems of top-down parsing.
	12 Feb	L	Predictive parsing, implementation.
	13 Feb	E	Coursework – how to evaluate.
Week 6	17 Feb	L	Bottom-up parsing. LL(k) vs LR(k). Shift-reduce parsing.
	19 Feb	L	Precedence parsing.
	20 Feb	E	Choosing parsers, analysis of "real" languages.
Week 7	24 Feb	L	Introduction to LR(k) parsing. Introduction to semantic analysis.
	26 Feb	L	Symbol tables. Intermediate representation.
	27 Feb	E	Feedback from the evaluation of the programming exercise.
Week 8	3 Mar	L	Introduction to code generation.
	5 Mar	L	Storage allocation.
	6 Mar	E	Optimisation.
Week 9	10 Mar	L	Code generation for arithmetic expressions.
	12 Mar	L	Register allocation. Flow analysis.
	13 Mar	E	Revision, exam questions.
Week 10	17 Mar	L	Code optimisation.
	19 Mar	L	Implementation techniques. Implementation of Java. Compiler generating tools. Portability.
	20 Mar	E	—

5 Reading list

This is only a small selection of appropriate books for the course. It makes good sense to read widely.

[1] Des Watson. *High-Level Languages and their Compilers*. International Computer Science Series. Addison-Wesley Publishing Company, Wokingham, England, 1989.

[2] David A. Watt and Deryck F. Brown. *Programming Language Processors in Java*. Prentice Hall, 2000.

[3] Alfred V. Aho, Monica S. Lam, Ravi Sethi, and Jeffrey D. Ullman. *Compilers – Principles, Techniques & Tools*. Pearson Education, second edition, 2007.

[4] Charles N. Fischer and Richard J. LeBlanc Jr. *Crafting a Compiler*. The Benjamin/Cummings Publishing Company, 1988.

[5] Dick Grune, Henri E. Bal, Ceriel J. H. Jacobs, and Koen G. Langendoen. *Modern Compiler Design*. Wiley, 2000.

Des Watson
January 2009