Course Information

Important: Computer Science courses have been renumbered: undergraduates should register for CS454 and graduate students for CS554!

Course structure for Fall 2001

The course covers introductory topics in compiler construction, including computer organization and architecture, operating system support, code and data layout, memory management, generation of executable code, intermediate representations, simple code optimizations, as well as the traditional topics of syntax analysis.

In addition to lectures and homework, students will work on a semester-long compiler project. The goal of the project is to develop a fully functional translator from a usable programming language to a form close to executable.

The implementation language will be chosen by each participant individually. All project stages will be defined as input/output specifications without prejudice to the implementation language. An important aspect of the project will be to report on the convenience or inconvenience of using a particular implementation language.

This course may be taken concurrently with CS591/491, Memory Management, also given this fall. For this reason there are no lectures on memory management and garbage collection in this syllabus.

This course will be naturally followed by CS555, Advanced Topics in Compiler Consturction, which will be given in the fall of 2002, in which the emphasis will be on code transformation for performance improvement ("optimization").

Previous acquaintance with the procedural and functional programming paradigm is assumed, as described below.

Assignments

Midterm exam, final exam (covering the entire course), 2-3 written homework assignments, one very large programming project.

Prerequisites in detail

There are two essential prerequisites. First, students should be familiar with several high-level programming languages, preferably including representatives of the procedural, object-oriented, and functional programming, so that they can appreciate the purpose and the tasks of a compiler. Second, students should be experienced programmers able to develop very large programming projects implemented in some programming language.

Experience with developing substantial applications in functional and imperative (especially object-oriented) programming languages can be gained by taking UNM CS courses 257 - *Nonimperative Programming*, 451 - *Programming Paradigms* and 351 - *Design of Large Programs* (for Scheme, ML, and C++, respectively).

Lectures

Mondays, Wednesdays, and Fridays, 2:00 - 2:50, in Tapy Hall 219

Instructor

Darko Stefanovic, office FEC 345C, phone 2776561, email darko@cs.unm.edu — office hours Mondays 2:50-3:50 or by appointment

Teaching assistant

None

Textbooks

Required reading

Michael L. Scott: *Programming Language Pragmatics*, Morgan Kaufmann, 2000, ISBN 1-55860-442-1.

Keith D. Cooper and Linda Torczon: *Engineering a Compiler*, Morgan Kaufmann, forthcoming.

Optional reading (newer books on compiler construction)

Reinhard Wilhelm and Dieter Maurer: *Compiler Design*, Addison-Wesley, 1995, ISBN 0-201-42290-5.

Dick Grune, Henri E. Bal, Ceriel J.H. Jacobs and Koen G. Langendoen: *Modern Compiler Design*, John Wiley, 2000, ISBN 0-471-97697-0.

Andrew W. Appel: *Modern compiler implementation in ML* Cambridge University Press, 1998, ISBN 0-521-58274-1.

David A. Watt and Deryck F. Brown: *Programming Language Processors in Java*, Prentice Hall, 2000, ISBN 0-13-025786-9.

Optional reading (old but good books on compiler construction)

Alfred V. Aho, Ravi Sethi, and Jeffrey D. Ullman: *Compilers : principles, techniques, and tools* Addison-Wesley, 1986 (Reprint with corrections March, 1988), ISBN 0-201-10088-6.

Optional reading (advanced topics in compiler construction)

Steven S. Muchnick: *Advanced Compiler Design and Implementation*, Morgan Kaufmann, 1997, ISBN 1-55860-320-4.

Michael Wolfe: *High Performance Compilers for Parallel Computing*, Addison-Wesley, 1996, ISBN 0-8053-2730-4.

Andrew W. Appel: *Compiling with Continuations* Cambridge University Press, 1992, ISBN 0-521-41695-7.

Optional reading (background material)

John L. Hennessy and David A. Patterson: *Computer Architecture: A Quantitative Approach*, second edition, Morgan Kaufmann, 1996, ISBN 1-55860-329-8.

Daniel P. Friedman, Mitchell Wand, and Christopher T. Haynes: *Essentials of Programming Languages*, MIT Press, 1992, ISBN 0-262-06145-7. (cross-listed as McGraw-Hill ISBN 0-07-022443-9)

Richard Jones and Rafael Lins: *Garbage Collection: Algorithms for Automatic Dynamic Memory Management* John Wiley, 1996, ISBN 0-471-94148-4.

Jeffrey D. Ullman: *Elements of ML Programming, ML97 Edition*, Prentice Hall, 2000, ISBN 0-13-790387-1.

Lawrence C. Paulson: *ML for the Working Programmer, 2nd edition*, Cambridge University Press, 1996, ISBN 0-521-56543-X.

Richard Bird: *Introduction to Functional Programming using Haskell*, Prentice Hall, 1998, ISBN 0-13-484436-0.

Paul Hudak: *The Haskell School of Expression*, Cambridge University Press, 2000, ISBN 0-521-64408-9.

David A. Watt: *Programming Language Concepts and Paradigms*, Prentice Hall, 1990, ISBN 0-13-728874-3.

Michael J. C. Gordon: *Programming Language Theory and its Implementation,* Prentice Hall, 1988, ISBN 0-13-7304170-X.

R. Tennent: *Principles of Programming Languages*

C. N. Fischer and R. J. LeBlanc: *Crafting a Compiler*, Benjamin/Cummings, 1988.

To probe further

John C. Reynolds: *Theories of Programming Languages*, Cambridge University Press, 1998, ISBN 0-521-59414-6.

Grading

You are expected to attend class regularly, read the assigned reading before class, and participate in class discussion. The grade will be determined as follows:

Homeworks 10%

Programming project 60% (developed and graded in stages)

Exams 30% (10% midterm exam, 20% final)

Homework and programming assignment hand-in policy

Homework assignments are due on the date assigned, no extensions will be granted, and no credit will be given for late homework. Hard copy solutions must be handed in either in class, or during office hours on the due date. Late programming project submissions will be penalized $3n^3$%, where n is the number of days late.

Programming assignments

The textual layout of the program must be logically sound and aesthetically pleasing. The names used must be descriptive. Code comments and additional documentation must accompany programs, and must provide informal argumentation that the program satisfies the specification.

Cooperation and cheating

Feel free to *discuss* homeworks and the project with classmates, the TA, and the instructor. However, *do not look at or copy another student's solution to a homework or project.* If a problem appears too difficult, or you lack the background to solve it, you are expected to talk to the instructor promptly. Once you have the background necessary to solve a problem, you must provide your own solution. Exchanging homework or project solutions is cheating and will be reported to the University administration; students involved will not be permitted to continue in the class.

Lecture Plan

- Weeks 1 and 2: Course Organization; Translation and Interpretation (Chapter 1)
- Weeks 2 and 3: Syntax Analysis - Scanning and Parsing (Chapter 2)
- Weeks 4 and 5: Names, Scope, and Binding (Chapter 3)
- Week 6: Semantic Analysis - Atttribute Grammars (Chapter 4)
- Week 7: Architecture Review (Chapter 5)
- Weeks 8 and 9: Data Types (Chapter 7)
- Weeks 10 and 11: Control Flow (Chapter 6)
- Weeks 12 and 13: Subroutines (Chapter 7)
- Weeks 14 and 15: Code Generation (Chapter 9)