Preliminary version of 25 July 2019

Course Information

Lectures

Tuesdays and Thursdays, 2:00-3:15, in SMLC352.

Instructor

Darko Stefanovic, office hours Mondays 11:00–12:00, Tuesdays 3:15–4:00, Thursdays 3:15–4:00, in FEC2020.

Teaching assistant

None.

Course topics

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. Students will implement several components of a compiler, ranging from parsing to native code generation, with an emphasis on tools for automated translation. The course will focus on the implementation of imperative languages; techniques for functional and logic languages will be left out, as indeed the special techniques for object-oriented languages. The course will treat simple program analysis and code optimization techniques.

Prerequisites

Students should be familiar with computer organization and with several high-level programming languages, so that they can appreciate the purpose and the tasks of a compiler. Students should be experienced programmers able to develop fairly large programming projects quickly, and comfortable working on a team.

Course format

The course will consist of lectures, written homework assignments, and projects. Students enrolled in CS554 (i.e., graduate students) will be given additional work as part of their homework assignments and projects.

Assignments

Midterm exam, 1–3 short written homework assignments to consolidate lecture material, 3–4 programming projects.

Written homework assignments

Homework assignments will be carried out individually. Detailed submission instructions will be given with each assignment.

Projects

Each project will be an implementation of an algorithm or phase in a compiler, or an algorithm or tool used to automatically generate a phase in a compiler. Detailed input/output specifications will be provided. Students will be free to choose any implementation language(s), subject to the constraint that a standalone executable file (runnable on CS machines) must be generated and submitted as part of the solution. (Similarly, the instructor will be free to discuss implementation strategies using any implementation language(s) by way of example.)

Projects will be carried out in teams of two. Detailed submission instructions will be given with each project.

Textbooks

Required reading

Torben Æ. Mogensen, *Introduction to Compiler Design*, 2nd Edition, Springer, `https://doi.org/10.1007/978-3-319-66966-3`.

General reading on compilers

Keith D. Cooper and Linda Torczon, *Engineering a Compiler*, Morgan Kaufmann, 2nd Edition, 2011, ISBN-10: 9780120884780.

Alfred V. Aho, Monica S. Lam, Ravi Sethi, and Jeffrey D. Ullman, *Compilers: Principles, Techniques, and Tools*, Addison Wesley, 2nd Edition, 2006.

Dick Grune, Henri E. Bal, Ceriel J.H. Jacobs and Koen G. Langendoen: *Modern Compiler Design*, Springer, 2nd Edition, 2012, ISBN-13: 978-1461446989.

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 20%

Programming projects 60%

Midterm exam 20%

Homework and programming assignment hand-in policy

Written homework assignments are due on the date assigned, no extensions will be granted, and no credit will be given for late homework. Late programming project submissions will be penalized $3n^3\%$, where n is the number of days late.

Lecture Plan

- Week 1: Course organization. Translation and interpretation. Code generation. Register allocation.
- Week 2: Lexical analysis. Regular expressions. Finite automata. Theory, pragmatic issues, tools.
- Week 3-4: Syntax analysis. Parsing algorithms, top-down, bottom-up, pragmatics, tools.
- Week 5: Names, scope, and binding. Types. Semantic elaboration. Attribute grammars.
- Week 6: Intermediate representations.
- Week 7: Representing data types. Representing control flow.
- Week 8: Representing the procedure abstraction.
- Week 9: Code generation. Instruction scheduling. Register allocation.
- Week 10-14: Code optimization. Program analysis.

Mailing list

A mailing list will be used for class discussion. It may also be used for administrative announcements.

Course materials

UNM Learn or Slack or Piazza - to be decided.

UNM statement of compliance with ADA

Qualified students with disabilities needing appropriate academic adjustments should contact the instructor as soon as possible to ensure their needs are met in a timely manner. Handouts are available in alternative accessible formats upon request.