ECE 468: Intro to Compilers and Translation Systems Engineering

Spring 2010

Lectures: Mondays, Wednesdays and Fridays, 12:30–1:20, ME 118
Course web page: https://engineering.purdue.edu/~milind/ece468/2010fall/

Instructor: Milind Kulkarni (milind@purdue.edu)
Office: EE 324A
Office Hours: Mondays and Wednesdays, 1:30–2:30 and by appointment.
TA: Chenguang Sun (ee468@ecn.purdue.edu)
TA Office Hours: TBD (will be in EE 306)

Course Description: This course covers the tools and techniques required to build a compiler for computer programs. The basics of compiler front ends (which translate a source program into an internal representation) will be covered in the first third of the course. The remainder of the course will discuss "back-end" compiler techniques, such as register allocation, instruction scheduling and program analysis. Students will implement a full, front-to-back compiler for a simple programming language.

Prerequisites: A solid grounding in C++, Java or some other high level programming language, as well as of data structures. Familiarity with assembly language and computer architectures is recommended, but not required. Note that *the course project in this class is a large programming project—comfort with programming is a must!*

Textbook: Fisher and LeBlanc, *Crafting a Compiler in C*. Lectures and class notes will supplement the textbook.

Course Outcomes: At the end of the course, a student who has successfully met the course objectives will be able to:

1. Describe and explain the terminology, representation and use of formal languages and grammars;
2. Describe and explain the terminology and techniques of lexical analysis, parsing, semantic actions and code generation;
3. Design and implement a compiler for a small language based on their knowledge of the previous two points.

More specifically, at the end of the course, you will be able to:

- Explain the various passes of a compiler (scanners, parsers, semantic actions and code generation, register allocation and basic optimizations) and how they relate to the overall compilation process.

- Explain and implement the algorithms for each of these processes.
- Be able to implement each of these passes and integrate them into a full compiler.
- Explain program analysis techniques that are used for code optimization, such as dataflow analysis, def-use analysis, liveness analysis, etc.
- Describe basic code transformations and their application to program optimization.

Course assessment: The achievement of course objectives will be assessed through a combination of problem sets (~1 a week), tests (2 midterms and a final) and a substantial course project. The problem sets and tests will assess students' achievement of the first two outcomes, while the project will assess students' achievement of the third outcome.

Course grading: Grades will be assigned as follows:

35% — Tests (2 midterms @10%, 1 comprehensive final @15%)
50% — Project
10% — Problem sets (10 total)
5% — Class participation

There may be a constant curve (*i.e.*, all grades will be increased by a fixed amount) for individual exams at the instructor's discretion. Your course grade will be determined using an absolute scale: 97–100: A+; 91–97: A; 88–91: A-; and continuing down.

Problem sets: There will be 10 problem sets, approximately one per week (excluding the week of Thanksgiving, the weeks that exams are given, and the last week of class. Each problem set will cover material covered in that week (and potentially earlier weeks). Each set will be posted online on Monday and will be due in class the next Monday (you may submit earlier via email). The sets will be graded on a ternary scale: 1 for everything (close to) right, 0.5 for attempting the problems, 0 for not turning it in. Late submissions will be given a 0. While the problem sets factor in to your grade, the primary benefit is for your own study; midterm and exam questions will often be in the same format as the questions on the problem sets.

Exams: Exams will be held in the evening, on the dates given below, unless otherwise announced. Exams are open-book and open-notes. Note that while each exam mostly covers certain topics, there may be information from earlier in the course on each; especially, the final exam will be comprehensive. If you cannot make it to an exam for some reason, let me know as soon as possible (and no later than 2 weeks before the exam) so we can schedule a make up for you. Exams missed without prior notice will result in a grade of 0, absent a compelling excuse.

Exam topics and dates:

- Midterm 1 — Scanning, parsing, semantic routines (September 30th, 6:30–7:30, EE 117)
- Midterm 2 — Semantic routines, code generation, instruction scheduling (November 4th, 6:30–7:30, EE 117)

- Final — Cumulative, with emphasis on register allocation, program optimizations and program analysis (TBA)

Note that the midterms are in the evening! The class period before each midterm will be devoted to a review session.

Course Topics: Below is the list of topics that will be covered in this course, and a rough estimate of how long we will spend on each. There is 1 week of slack in the schedule, and 1 week allocated for exams.

Topic	# of weeks	Reading
Structure of a compiler (introduction and overview)	1	Chapters 1 & 2
Scanning	1	Chapter 3
Parsing (recursive descent, overview of shift-reduce)	2	Chapters 4–6
Semantic routines (building a symbol table and AST)	2	Chapters 7–12
Semantic routines (for functions)	1	Chapter 13
Code generation (generating three-address code from AST, peephole optimizations, etc.)	1	Chapter 15
Instruction scheduling	1	Handouts, notes
Register allocation	1	Handouts, notes
Program optimizations (code motion, strength reduction, etc.)	1	Handouts, notes
Program analysis	2	Handouts, notes

Project: The bulk of your grade will be determined by a course project. This project involves implementing a full-fledged optimizing compiler for a simple language. You may implement your project in any language, although using a high-level language such as C++ or Java will probably make your life easier.

The project consists of multiple steps, each of which will be graded separately. However, each step builds on the results of previous steps, so it behooves you to ensure that each step works properly. The bulk of your project grade (60%) is based on the performance of your final compiler on several predetermined test programs; the intermediate steps will, together, constitute 30% of your project grade; the final 10% will be based on your compiler's performance on several undisclosed test programs.

The project steps (and due dates) are as follows:

Step	Description	Due date
0	Verify that you can properly turn in project steps	Friday, Sept. 3rd
1	Use Lex, other tools, or a hand written lexer to scan a program written in the language given on the course web page. The output of this step will be one line for each token encountered, which contains the token and its type (an integer value that you decide on).	Friday, Sept. 10th
2	Using YACC, or another parser generator, write a parser for the grammar for the project language. Lexical analysis will be done by project step one. The output of this step is a parse tree (one symbol per line, indented by the depth of the tree). Parser error recovery does not need to be implemented. However the parser must stop correctly upon a detected syntax error.	Friday, Sept. 24th
3	Implement the semantic actions associated with variable declaration. The symbol table entry object has an identifier name field and a type field. In particular, when an integer or float variable declaration is encountered, create an entry whose type field is integer (or "float") and its return type to N/A. Functions declarations do not need to be handled at this time. The string corresponding to the identifier name can either be part of the identifier entry in the symbol table, or can be part of an external string table that is pointed to by the symbol table entry. When a new scope is encountered, a new symbol table should be created. Thus, when entering a function, or the body of an IF, ELSE, WHILE or FOR loop a new symbol table needs to be created, and the symbols declared in that scope added to the symbol table. The output of your compiler should be a listing of the type of scope the symbol table is for (an IF, ELSE, WHILE, FOR, FUNCTION or PROGRAM), the name, if any of the scope, and the symbol table entries, with each line containing the variable name and its type.	Monday, October 4th (later to compensate for exam the previous week)

Step	Description	Due date
4	Process assignment statement and expressions. For this step, expressions will only appear in assignment statements. In this step an internal representation (IR) of the program will be formed. This IR consists of a list of nodes, one node per IR statement. The nodes will appear in the list in the order they are generated by the semantic routines. The node will have the following fields for assignment statements: successor edges; node type; left-hand side variable (a symbol index); operation type; first and second operands (symbol indices.) Eventually these nodes will be part of a flow graph – make allowances for lists pointers to successor and predecessor nodes. The semantic actions for each sub-expression will produce code of the form <lhs>=<1st operand> <operation> <2nd operand>. The node will contain this information and pointers to the operation that is immediately reachable after this node. Implement the **read** and **write** statements.	Friday, Oct. 15th
5	Implement semantic actions for **if**, **while** and **for** statements. This includes creating IR nodes for the statements and writing a pass that traverses the IR and generates code executable on the Tiny simulator. The output for this step will be the output from running your program on the Tiny simulator. I would recommend that you be able to handle if statements by the 18th, with only do while statements remaining for the following week.	Friday, Oct. 29th
6	Implement semantic actions for subroutine definitions and subroutine invocation. I would suggest creating a separate IR and symbol table for each subroutine. As with the previous step, the output from this step will the be Tiny simulator output for the program	Friday, Nov. 12th
7	Perform an intraprocedural liveness analysis. Perform register allocation. You may use the allocator in the book, or one of the other ones described in the handouts. The output will be the Tiny simulator output.	Friday, Dec. 3rd
8	Turn in the final version of your project. This gives you a chance to correct any remaining bugs and test your compiler	Saturday, Dec. 11th

Group work policy: You can (optionally) work on the project in teams of 2. If you do so, you must send me an email with the subject heading “ECE 468 Project Team” listing your two team members before the first project step is due.

Late submission policy: Except for medical and family emergencies (accompanied by verification), there will be *no extensions* granted for project submissions. Late submissions will be penalized 10% per day late, up to a maximum of 50%. Submissions more than 5 days late will be assigned a score of 0.

Campus Interruptions: In the event of a major campus emergency, course requirements, deadlines and grading percentages are subject to changes that may be necessitated by a revised semester calendar or other circumstances. In such an event, information will be provided through the course website and email.

Academic Honesty: Unless expressly allowed, you are expected to complete all assignments by yourself. However, you are allowed to discuss general issues with other students (programming techniques, clearing up confusion about requirements, etc.). You may discuss particular algorithmic issues on the newsgroup (but do not copy code!). *We will be using software designed to catch plagiarism in programming assignments, and all students found sharing solutions will be reported to the Dean of students.*

Punishments for academic dishonesty are severe, including receiving an F in the course or being expelled from the University. By departmental rules, all instances of cheating will be reported to the Dean. On the first instance of cheating, students will receive a 0 on the assignment; the second instance of cheating will result in a failure of the course.