ECE 46800 Introduction to Compilers and Translation Engineering

Instructor:
Xiaokang Qiu
Office: EE 334C
Phone: 765-494-9987
Email: xkqiu@purdue.edu
Office Hours: Tu 1:30-3
F 9:30-11
https://engineering.purdue.edu/~xqiu/

Teaching Assistants:
Chris Wright
Office: EE 207
Email: wrigh338@purdue.edu
Office Hours: MWF 8:30-9:30
Kangjing Huang
Office: EE 207
Email: huang989@purdue.edu
Office Hours: F 2-3:30

Course Information

Fall 2018
WMF 7:30-8:20
ME 1061
Credit Hours: 4

Course Description

The design and construction of compilers and other translators. Topics include compilation goals, organization of a translator, grammars and languages, symbol tables, lexical analysis, syntax analysis (parsing), error handling, intermediate and final code generation, assemblers, interpreters, and an introduction to optimization. Emphasis is on engineering a compiler or interpreter for a small programming language–typically a C or Pascal subset. Projects involve the stepwise implementation (and documentation) of such a system.

Prerequisites

ECE 36200, ECE 36800.

Learning 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 [1];
2. Describe and explain the terminology and techniques of lexical analysis, parsing, semantic actions and code generation [1];
3. Design and implement a compiler for a small language based on their knowledge of the previous two points [1, 2, 5, 6].

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, reaching definitions analysis, liveness analysis, etc.
- Describe basic code transformations and their application to program optimization.

Required Texts

Fisher and LeBlanc: *Crafting a Compiler in C.* Lectures and class notes will supplement the textbook. The textbook will primarily be useful for the project.

Course Requirements

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

Grading: Grades will be assigned as follows:

45% — Tests (2 midterms @10%, 1 comprehensive final @25%)
40% — Project
10% — Problem sets (5–6 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 5-6 problem sets, approximately one every two weeks. Each set will be posted online on Wednesday and will be due in class the next Wednesday (you may submit earlier via email). The problem sets will be graded on a 0-1 system: 1 point for turning in a serious attempt, 0 points for not turning in a set or not making an honest attempt at the problems. 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.

Problem sets normally be due by 8:30pm on Wednesday, right before the class begins. Project steps normally due by 11:59pm of the due date. Due dates are subject to change at the instructor's discretion. **Late problem sets will not be accepted.**

Project: 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 majority 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.

You can (optionally) work on the project in teams of 2. You must decide if you want to work on a group prior to turning in Step 1. If you choose to work in a group, you must indicate it in your submission. Once you choose a partner, you must continue to work with this partner for the remainder of the project.

We reserve the right to make appropriate adjustments to ensure that grades are appropriate in rare and unusual circumstances, e.g. performance that gets dramatically better or worse over the course of the term, extreme mismatches between exam and homework averages, sickness affecting an exam, etc. In cases of doubt, we will try to be generous, within the limits of common sense.

Class Schedule

Exams and midterms: We will have evening midterms. The midterms and final are open book and open notes. The tentative times and topics for the exams are below:

Exam	Time	Location	Topics
Midterm 1	Tuesday, September 25th, 8-9pm	EE 170 ME 1061	Introduction, scanning and parsing
Midterm 2	Tuesday, November 1st, 8-9:15pm	FRNY G140	Semantic routines, code generation, register allocation, instruction scheduling, peephole optimizations
Final	Tuesday, December 11st, 7-9pm	FRNY G140	Cumulative, with emphasis on loop optimizations, dataflow analysis, pointer analysis and dependence analysis

Due to the extra hours for midterms, the meetings on Friday, November 16th, Monday, November 19th and Monday, December 3rd are to be omitted.

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)	0.5	Chapters 1 & 2
Scanning	0.5	Chapter 3
Parsing (recursive descendent, overview of shift-reduce)	1	Chapters 4-6
Semantic routines (building a symbolic table and AST)	1.5	Chapters 7-12
Semantic routines (for functions)	1	Chapter 13
Code generation (generating three-address code from AST, peephole optimization, etc.)	1	Chapter 15
Instruction scheduling	0.5	Lecture notes
Register allocation	0.5	Lecture notes
Program optimization (code motion, strength reduction, etc.)	1	Lecture notes
Control flow analysis (building a CFG)	0.5	Lecture notes

Data flow analysis (lattice theory, specific DFAs)	2.5	Lecture notes
Pointer/alias analysis	0.5	Lecture notes
Parallelism (dependence analysis, optimistic parallelization)	2	Lecture notes

Project: This project involves implementing a full-fledged optimizing compiler for a simple language. The project steps (and due dates) are as follows:

Step	Description	Due date
0	Verify that you can properly turn in project steps, choose project partner	Mon, August 27th
1	Use ANTLR, 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).	Wed, September 5th
2	Using ANTLR, 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.	Fri, September 14th
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, or WHILE 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, 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.	Mon, October 1st
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. Build the IR by first constructing an Abstract Syntax Tree	Fri, October 12th

	(AST), then producing an IR from that tree. 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 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.	
5	Implement semantic actions for if 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 23rd, with only for statements remaining for the following week.	Fri, October 26th
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 be the Tiny simulator output for the program.	Fri, November 16th
7	Perform an intraprocedural liveness analysis. You will have to construct a control flow graph (CFG) according to the procedure discussed in class, then write a dataflow analysis to compute liveness. Perform register allocation using the results of this liveness analysis, using either bottom-up register allocation or graph coloring. The output will be the Tiny simulator output. You will use a new version of the Tiny simulator that provides a limited number of registers.	Mon, December 3rd
8	Turn in the final version of your project. This gives you a chance to correct any remaining bugs and test your compiler.	Fri, December 7th

How to Succeed in This Course

1) If you think a homework problem is buggy or unclear, post a query to Piazza.
2) For the project, it is highly recommended that you work in pairs. Please pair up in the two weeks of classes.
3) Although you can use any language to implement the project, it is recommended that you use Java with ANTLR (http://www.antlr.org) due to the TA's personal expertise. You can also use C/C++ with Flex/Bison or any other language but the TA may not be able to help you as much as with Java.
4) You are highly encouraged to start working on the steps as soon as they are posted and not wait until the last day.

5) Although the preliminary steps (0 through 7) will only be tested on the published test cases, you are encouraged to write your own test cases to make sure that your compiler fully supports the provided grammar. Doing this throughout the semester will help you ensure that your final compiler (step8) will pass the hidden test cases.
6) In any case, if you feel lost, please come and speak with the instructor/TA during their office hours, and/or/xor send them email.

Policies

Academic Dishonesty

"As a boilermaker pursuing academic excellence, I pledge to be honest and true in all that I do. Accountable together - we are Purdue." – Purdur Honors Pledge

We hope (and assume) that you are all honest and have no intentions of cheating. Cheating ruins the experience for everyone and we will pursue appropriate penalties if we catch someone cheating. Please don't -- it's not worth it, both for you and for us. You can find Purdue's student guide for academic integrity at https://www.purdue.edu/odos/academic-integrity/ .

If we do catch someone cheating for the first time, the following penalties will be applied:

- On a midterm exam: zero for the exam.
- On the final exam: failure in the course.
- On a homework: zero on the assignment. Moreover, we will subtract 10% from your final average in the course, or reduce your letter grade by one category (e.g., B to B-), whichever results in the lower grade.

For second cheating offense of any kind, you will fail this course.

Attendance

You are expected to be present for every meeting of the classes in which they are enrolled. When conflicts or absences can be anticipated, such as for many University sponsored activities and religious observations, you should inform the instructor of the situation as far in advance as possible. For unanticipated or emergency absences when advance notification to an instructor is not possible, you should contact the instructor as soon as possible by email. If you are unable to do so, or in cases of bereavement, contact the Office of the Dean of Students (ODOS).

If you notice major problems with our schedule (e.g. our midterm conflicts with an exam in another course that many of you are taking), please tell us promptly so that we have the best chance to fix it. If you have a disability or other special circumstance which may require special accommodations, please speak to us.

Emergencies

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 beyond the instructor's control. Relevant changes to this course will be posted onto the course website

or can be obtained by contacting us via email or phone. You are expected to read your @purdue.edu email on a frequent basis.

CAPS Information

Purdue University is committed to advancing the mental health and well-being of its students. If you or someone you know is feeling overwhelmed, depressed, and/or in need of support, services are available. For help, such individuals should contact Counseling and Psychological Services (CAPS) at (765)494-6995 and http://www.purdue.edu/caps/ during and after hours, on weekends and holidays, or through its counselors physically located in the Purdue University Student Health Center (PUSH) during business hours.

Students with Disabilities

Purdue University strives to make learning experiences as accessible as possible. If you anticipate or experience physical or academic barriers based on disability, you are welcome to let me know so that we can discuss options. You are also encouraged to contact the Disability Resource Center at: drc@purdue.edu or by phone: 765-494-1247.

Disclaimer

This syllabus is subject to change.