Syllabus
Introduction to Compiler Design

Charles Averill

Fall 2023

E-mail: `charles@utdallas.edu`

Course Website

Class Hours: Monday, Wednesday 11:00AM - 12:15PM
Office Hours: MW 12:15PM - 1:30PM
Class Room: TBA
Course Number: N/A

Course Description

This course will provide a broad introduction to compiler design, focused specifically on code generation. This course will not provide credit for any degree from the University of Texas at Dallas or any other university. Upon completing this course, students will be rewarded with the **University of Texas at Dallas Certification in Compiler Design**.

By the end of this course, students will have built a functioning compiler for either a subset of C should they choose to follow along with the course material, or their own imperative programming language should they choose to do so.

This course will briefly cover popular compiler design technologies, such as `flex` and `bison`, but will **not** use them in the material. Instead, a scanner and parser will be built from scratch. Students are given free reign to implement functionality in their compiler as they see fit, so implementation of these popular technologies is allowed. From-scratch implementation is encouraged, however, to ensure that students have a better working understanding of the internals of a basic compiler.

Soft Prerequisites (not required)

CS2337 or equivalent experience in an imperative programming language (C, C++, Java, Python, etc.). The course will be taught in Python, so familiarity is encouraged. However, confidence in another language will suffice, as no specific Python features will be important. **CS2340** or equivalent knowledge of basic programming in any assembly language. **CS3345** or equivalent knowledge of trees, linked lists, basic hash tables, basic recursive tree traversal. **CS3377** or equivalent knowledge of basic Unix, Bash usage. Provided course materials assume that the compiler will be built and run on a Linux target, so usage of the `cs1.utdallas.edu` server or a physical Linux machine is required, unless students choose to not use the project base. Basic `git` knowledge is assumed.

Should course participants request, the instructor will provide a review video covering needed prior knowledge for this course.

Course Objectives

Successful students will:

1. Design a compiler for either a subset of C or a language of their choice
2. Implement extra features as optional homework
3. Learn how common high-level structures such as loops, conditional statements, functions, and more can be parsed into Abstract Syntax Trees and mapped to generated LLVM-IR pseudo-assembly code.
4. Learn how to apply simple optimizations to parsed high-level code and abstract syntax trees in order to improve generated pseudo-assembly code
5. Compare the (relatively) naive approaches presented in this course to production-level approaches found in compilers like `gcc` and `clang`
6. Complete knowledge-testing quizzes in class

Optional Materials

- "Compilers: Principles, Techniques, and Tools" - Alfred V. Aho, Ravi Sethi, Jeffrey D. Ullman
- SubC
- ACWJ - DoctorWkt
- LLVM's "My First Language Frontend"
- LLVM-IR Language Reference Manual

Course Structure

Class Structure

Twice-weekly classes will outline the updates to our compiler that we need to make in order to generate code for high-level language features, small optimizations we could make (these will not be implemented in class, as `clang` will apply optimizations to our generated LLVM-IR when we compile it), and improvements we can make to our compiler binary to improve user experience and match best software development practices. Some lecture periods will be used to introduce students to more complex topics in compiler design that may not be feasible to implement as part of coursework.

Quizzes

Once-weekly in-class quizzes will be given to ensure that students are keeping up with course content. Quizzes may be of any form as the instructor sees fit, and answers will be graded for correctness with partial credit where applicable.

Class Project

The main goal of this course is for students to have a functioning, hand-built compiler by the end of the course. There is no requirement for students to compile any specific language, or to write the compiler in any specific language. A project base will be provided to offer argument parsing, documentation generation,

and automatic testing, but it is not required for students to use the project base. Completion of any 75% of the selected topics (including required topics listed in the Calendar) is required to receive a certification.

Calendar

The order of later topics is subject to change. After week 4, much of the higher-level constructs can be implemented in any order. **Bold** topics are required to earn the certification. Red topics are tech-talk-style lectures over topics not expected to be implemented for the class project.

Week	Dates	Content	Optional Assignments
1	Aug 28, 30	**Scanning**, **Parsing**, **Precedence**	Bit-shifting operators
2	Sep 4, 6	**LLVM Generation**, General-Purpose Parsing	Your own cool statement
3	Sep 11, 13	**Basic print statements**, Architecture of Large Compilers	Integer types (except `char`)
4	Sep 18, 20	**Global variables**, Compiler Security	N-base integer literals
5	Sep 25, 27	**Comparisons**, **If statements**, **While loops**, For loops	`break` and `continue`
6	Oct 2, 4	`char` type, **Function declarations and calls**, JIT Compiling	While-else and For-else loops
7	Oct 9, 11	**Pointers**, Functional Language Compilation	
8	Oct 16, 18	Catch-up week, just office hours in ECSS 4.619	
9	Oct 23, 25	**Basic Optimizations**, Generalizing Lvalues	
10	Oct 30, Nov 1	Local Variables Part 1, **Function arguments**, LLVM Optimizations	
11	Nov 6, 8	Local Variables Part 2, Loop Optimizations	
12	Nov 13, 15	**Naive arrays**, Higher-level language constructs	
13	Nov 20, 22	**Structs**, Parallelism Optimizations	
14	Nov 27, 29	Certified Compilers, Zero-Knowledge Proofs in Compilers	Triple Test
15	Dec 4, 6	Custom Language Presentations	**Relax**

Course Policies

During Class

I understand that the electronic recording of notes will be important for class and so computers will be allowed in class. Please refrain from using computers for anything but activities related to the class. Eating and drinking are allowed in class but please refrain from it affecting the course. Try not to eat your lunch in class as the classes are typically active.

Attendance Policy

There is lots of material to cover, so please attend all classes in order to stay up. If you miss a class, reviewing the related ACWJ topics is a good way to catch back up.

Recording

The course lectures will be recorded and uploaded to YouTube for student convenience.

Academic Integrity and Honesty

Please don't copy code from ACWJ if you can help it. It's tempting because we're following its evolution very closely, but it's a lot more fun and rewarding to implement everything based on what you learn in class. The ECCO source code also implements everything we talk about in class, please don't copy from there either.

UTD Syllabus Policies and Procedures

The information contained in the following link constitutes the University's policies and procedures segment of the course syllabus. Please go to http://go.utdallas.edu/syllabus-policies for these policies.