Syllabus
Practical Compiler Design

Charles Averill

Spring 2023

E-mail: `charles@utdallas.edu`
Web: **https://charles.systems/PCD**
Class Hours: Monday, Wednesday 1:00PM - 2:15PM
Office Hours: MW 2:15PM - 3:30PM
Class Room: TI Auditorium (1/23 - 2/1), ECSS 3.910 (2/6 - end)
Course Number: N/A

Course Description

This course will provide a near-comprehensive introduction to compiler design, and will focus 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 Practical 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`

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.

Assignments

Homework assignments will be optional extensions we can add to our language. Completing assignments is strongly encouraged, as it will provide students an opportunity to branch out and customize their language and compiler. Homework assignments can be submitted to be reviewed by the instructor once weekly or after class.

Class Project

The main goal of this course is for students to have a functioning, hand-built compiler by the end of the course. Because grading is optional, 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 70% 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. **Bolded** topics are required to earn the certification.

Week	Dates	Content	Optional Assignments
1	Jan 23, 25	**Scanning**, **Parsing**, **Precedence**	Bit-shifting operators
2	Jan 30, Feb 1	**Basic code generation**, **Basic print statements**, **Global variables**, **Comparisons**, **If statements**, **While loops**	While-else loops
3	Feb 6, 8	For loops, Integer types, **Functions**	Boolean value printing
4	Feb 13, 15	**Pointers**, Revisiting global variables, Type checking, Pointer offset scaling	Character literals
5	Feb 20, 22	Revisiting Lvalues, **Naive arrays**, **String literals**, Prefix and postfix operators	Binary logical operators
6	Feb 27, Mar 1	Local variables, **Function parameters and arguments**, Function prototypes	N-base integer literals
7	Mar 6, 8	Restructuring and refactoring, Introduction to structs and unions	
8	Mar 13, 15	**Structs**, unions, typedefs	Enums
9	Mar 20, 22	**Break**, **continue**, switch	External Preprocessor
10	Mar 27, 29	Refactoring compound statements, refactoring variable assignments, type casting	
11	Apr 3, 5	Revisitng operators, **Basic optimizations**, Revisiting global declarations, Void function parameters	
12	April 10, 12	Static, Ternary operators, Cleanup	Sizeof
13	April 17, 19	Revisiting arrays and pointers, More cleanup, Lazy logical operator evaluation	Consecutive switch cases
14	April 24, 26	Revisiting local arrays, General cleanup	Triple Test
15	May 1, 3	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.