COMP 430: Language Design and Compilers
Spring 2019

Instructor: Kyle Dewey (kyle.dewey@csun.edu)
Course Web Page: https://kyledewey.github.io/comp430-spring19/
Piazza Web Page: **http://piazza.com/csun/spring2019/comp430**
Office: JD 4427, Extension 4316 (not yet connected)

Course Description (From the Catalog)
Examination of the issues involved in the design and subsequent implementation of programming languages. Considerations of implementation difficulties, including various features in a programming language. Tools and techniques to facilitate both the processing of programming languages and the building of programming processors.

Learning Objectives
Successful students will be able to:
- Design a programming language with:
  - Concrete and abstract syntax
  - Statically-checked types
  - Expressions
  - Subroutines
  - Mechanisms for computation abstraction
- Implement a compiler for the designed language, complete with:
  - A parser
  - A typechecker / static semantic analyzer
  - A code generator

Course Motivation
A common question with compilers courses: *When am I ever going to need to implement a compiler?* I'll answer that upfront: probably never. However, consider some related questions:
- *When am I going to need to reuse my own code?*: All the time.
- *When will I need to understand how a language works?*: All the time.
- *When will I need to work in a team?*: All the time.
- *When will I need to understand why a language was designed in a certain way?*: At least as often as you evaluate new programming languages. Languages rise and fall frequently, and it's important to know what's worth your time and what isn't.

In this class, you will incrementally build on your own code, starting likely from scratch. You will live with your coding decisions (good or bad), and will likely have to revisit them. You'll gain a better understanding of how languages work, and further understand what is involved with language design. Perhaps most importantly, you'll learn that **languages aren't magic**: you can implement your own, and they are surprisingly straightforward once you understand the basics.

What this Course Is and Is Not

This is a project-based, implementation-oriented course. Our focus will be on modern compiler development. Grading is based on a series of assignments which will allow you to incrementally implement your compiler. While these assignments will accomplish the same high-level tasks, exactly what these assignments are depends on your particular language (i.e., the assignment implementing typechecking will differ for Teams A and B, since both implement different languages).

Because this is a project-based course, there are no exams, though I am planning to use the final exam slot for team presentations. Your code will serve as a demonstration of understanding class concepts. Additionally, since the focus is on modern compilers, I'm intentionally **not** planning to cover certain topics in-depth, namely:

- Efficient parsing (e.g., LL, LR, flex, bison, etc.). We won't need very efficient parsers for what we plan to do. Moreover, it's not uncommon for modern compilers to forego these things entirely; many of these algorithms and tools were developed at a time when memory was scarce, but this is no longer true. From my own experience, I've been implementing programming languages in various capacities for the past seven years, and I've never needed to reach for any of this.
- Very low-level concerns like register allocation or machine-specific optimizations. It's rare nowadays to compile directly to machine code, given existing backends like LLVM and the JVM. Additionally, these topics are completely irrelevant if everyone selects high-level compilation targets for their projects.

Textbook

No textbooks are required. That said, the following books may be of interest to you:

- Programming Language Pragmatics, by Michael Scott. Discusses a variety of programming language features from a variety of paradigms, and certain key interactions. Good for language design, though light on compilation.
- Compilers: Principles, Techniques, and Tools, by Alfred Aho, Monica Lam, Ravi Sethi, and Jeffery Ullman. AKA the "Dragon Book" (there is a dragon on the cover). Class, commonly-used textbook, which discusses low-level compiler details. Heavy focus on parsing, and it's better-suited as a reference than an introduction.
- Modern Compiler Implementation in Java/ML/C (these are each separate books), by Andrew Appel. Good introduction to compilers, with lots of example code. Not very general.
- Engineering a Compiler, by Keith Cooper and Linda Torczon. Great resource on optimizations and low-level concerns. Not a gentle introduction to compilers, and limited information about things like types.
- Types and Programming Languages, by Benjamin Pierce. Specifically about typechecking and *type theory*, specifically as it relates to a number of common programming language features. Highly specialized and math-heavy; can be intimidating.

Grading

All graded components somehow tie to the project. These are below:

Component	Percentage
Language Design Proposal	5%
Lexer - Initial	1.5%
Parser - Initial	2.5%
Typechecker - Initial	8%
Basic Expression Translation - Initial	8%
Subroutines/Control Structures Translation - Initial	6%
Abstraction of Computations Translation - Initial	9%
Project-Specific Custom Milestone - Initial	5%
Language User Documentation	6%
Bug Hunt	5%
Presentation	4%
Lexer - Final	1.5%
Parser - Final	2.5%
Typechecker - Final	8%
Basic Expression Translation - Final	8%
Subroutines/Control Structures Translation - Final	6%
Abstraction of Computations Translation - Final	9%
Project-Specific Custom Milestone - Final	5%

The components with the initial/final split likely look a little strange; the reasoning for this split follows. Ideally, for each one of these components (the lexer, the parser, etc.), you will turn it in on time, it will be completely correct, and it will never need to be revisited.

However, this are all unrealistic assumptions. As such, each component will be evaluated twice:

1. At the specific component deadline. This is primarily to ensure steady progress is being made on the project.
2. At the end-of-class overall project deadline. This is primarily to ensure that any issues revealed at the first deadline were fixed, and that new bugs haven't crept in.

If a component is submitted late, the late penalty will only be applied to the first deadline, as opposed to the end-of-class overall deadline.

Plus/minus grading is used, according to the scale below:

If your score is >=...	...you will receive...
92.5	A
89.5	A-
86.5	B+
82.5	B
79.5	B-
76.5	C+
72.5	C
69.5	C-
66.5	D+
62.5	D
59.5	D-
0	F

Special Note on Testing and Code Cleanliness

As seen in the previous section, there are a LOT of components for me to grade. This grading difficulty is multiplicative because each team's project is different, so I cannot easily apply a one-size-fits-all strategy. As such, **it is your responsibility to convince me that your code works**. If you have a bunch of good unit tests in your code, this will likely convince me. If I need to read through your code to make the determination that things work, clean code will help me come to this conclusion readily. However, if you lack tests and your code is a mess, I likely won't be convinced.

Special Note About Teams
Considering the amount of work this class demands, it is highly recommended to form teams. To ensure a fair distribution of work among teams, peer evaluations will be used. Additionally, I will count the number of edits made to code per student per graded component (we will use revision control via GitHub, making this easy to do). If your number of edits is significantly lower than everyone else's without explanation, I will give you (and only you, not your team), a 0 on the assignment. If you think a component is completely implemented already, code can **always** be added in the form of tests.

Plagiarism and Academic Honesty
You are permitted to collaborate as much as you'd like; because of the nature of the course, it's not really possible to take someone else's code as your own (i.e., Team A's typechecker won't work for Team B). If you use code from elsewhere, you must **cite** it. Any violations can result in a failing grade for the assignment, or potentially failing the course for egregious cases. A report will also be made to the Dean of Academic Affairs. Students who repeatedly violate this policy across multiple courses may be suspended or even expelled.

Attendance
In the first week of class, I will take attendance. If you miss both sessions in the first week and have not made alternative arrangements with me, you must drop the class, as per University policy (http://catalog.csun.edu/policies/attendance-class-attendance/). After the first week I will not take attendance, though you are strongly encouraged to attend.

Communication
- Piazza is strongly preferred (allows for private messages, anonymous posting, and class-wide public posting)
- Email is a fallback in case Piazza isn't working
- Do **not** use Canvas' messaging (very easy for me to miss messages)

Late Policy
Unless prior arrangements have been made, for each day a a component is late, it will be deducted by 20%. Assignments that are submitted more than 5 days late will not receive any credit.

Class Feedback
I am open to any questions / comments / concerns / complaints you have about the class. If there is something relevant you want covered, I can push to make this happen. I operate off of your feedback, and no feedback tells me "everything is ok". This is the first time I'm teaching this course, and it is the first time the course has had this particular structure, so I'm anticipating that it won't all be smooth sailing.

---Class Schedule is on Next Page---

Class Schedule and Component Due Dates (Subject to Change):
Items in **bold** are deadlines.

Week	Tuesday	Thursday	Friday
1	1/22: Introduction, motivation, project information	1/24: project information, team formation, feature survey	1/25:
2	1/29: feature survey, grammars, lexing	1/31: grammars, lexing	2/1: **Language Design Proposal**
3	2/5: lexing, ASTs	2/7: ASTs, parsing	2/8:
4	2/12: **Lexer**, parsing	2/14: parsing, type theory basics	2/15:
5	2/19: **Parser**, type theory basics, typechecking	2/21: typechecking	2/22:
6	2/26: typechecking	2/28: compilation basics	3/1: **Typechecker**
7	3/5: compilation of expresions	3/7: compilation of expressions	3/8:
8	3/12: **Basic Expression Translation**, compilation of control structures	3/14: compilation of control structures, subroutines	3/15:
9	~~3/19~~: Spring Recess (no class)	~~3/21~~: Spring Recess (no class)	~~3/22~~: Spring Recess (no class)
10	3/26: compilation of subroutines	3/28: compilation of methods, inheritance	3/29: **Subroutines/ Control Structures Translation**
11	4/2: compilation of methods, inheritance	4/4: compilation of higher-order functions	4/5:
12	4/9: additional topics as projects demand	4/11: additional topics as projects demand	4/12: **Abstraction of Computations Translation**

Week	Tuesday	Thursday	Friday
13	4/16: additional topics as projects demand	4/18: additional topics as projects demand	4/19:
14	4/23: additional topics as projects demand	4/25: additional topics as projects demand	4/26: **Project-Specific Custom Milestone**
15	4/30: additional topics as projects demand	5/2: **Language User Documentation**, team consultation	5/3:
16	5/7: team consultation	5/9: team consultation	5/10: **Bug Hunt**