The compiler is the programmer’s primary tool. Understanding the compiler is therefore critical for programmers, even if they never build one. Furthermore, many design techniques that emerged in the context of compilers are useful for a range of other application areas. This course introduces students to the essential elements of building a compiler: parsing, context-sensitive property checking, code linearization, register allocation, etc. To take this course, students are expected to already understand how programming languages behave, to a fairly detailed degree. The material in the course builds on that knowledge via a series of semantics preserving transformations that start with a fairly high-level programming language and culminate in machine code.

Course Code: COMP_SCI 322

When: Winter

Instructors:
[Simone Campanoni](mailto:simonec@eecs.northwestern.edu)

Peer mentors: Souradip Ghosh and Mekki Rachedine

Course Objectives:
Students that complete this course should:

1. understand how to efficiently implement a programming language,
2. have an accurate performance model for the primitives in a programming language,
3. have a good sense for the kind of assembly code that a compiler produces
4. be able to maintain and extend a compiler for a programming language

Assignments:
In this class, you will learn how to design and efficiently implement a compiler able to generate Intel x86_64 machine code from a high level programming language (a modern C-based language). The work is divided in eight assignments, one per week. Subsequent assignments build on top of all the previous ones.

Each compiler implemented by each team will be checked in at least two ways.

1. All the tests included in the framework distributed via Canvas will be checked automatically.
2. Simone will inspect the source code.
3. If you will present your code in a panel (see below), then the manager(s) will double check its correctness.

Completing all assignments will obtain an overall compiler capable to translate a program written in a C-like language to a semantically-equivalent Intel x86_64 executable binary. Such binaries will run correctly on real Intel-based platforms running a Linux OS. Furthermore, the generated binaries will be able to compete with production-quality compilers like gcc and clang.

**Competition:**
Compilers built by successful teams will compete at the end of the class. The team that has designed and implemented the compiler that generates the most performant binary will win. The students that compose the team that wins the competition will get an A independently on their points they have accumulated. Also, the winners will have the option to include their names and pictures in the Hall of Fame of this class (see the winners of prior years [here]).

**Panels:**
Every assignment is going to be evaluated in class with a code walk just after its deadline. To do so, a code walk is organized as a panel where a team describes their compiler, a manager asks questions to guarantee the correctness of the design and implementation of such compiler, manager helpers help the manager to do his/her job, and a secretary writes down all important discussions. Only teams that have submitted their correct solution on time can be selected to participate to the panels.

We will have four panels for every assignment. The first three panels will be composed only by teams. Simone will describe his code as the last panel and the selected teams will cover the other roles of the last panel.

**Materials:**
Course slides
Recommended book: Modern compiler implementation in C (or Java).

**Grading Policy:**
Your grade depends on points you will earn on assigned homework and on the code walks. You can earn up to eight points for each homework assigned (one per assignment). Also, you can earn up to three points for the panelist experiences. Finally, you earn an extra point if you submit the final compiler on time for the final competition and such compiler passes all tests.

The map between points and grades is the following:

<table>
<thead>
<tr>
<th>Grade</th>
<th>Points</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>12</td>
</tr>
<tr>
<td>A-</td>
<td>10 - 11</td>
</tr>
<tr>
<td>B+</td>
<td>8 - 9</td>
</tr>
<tr>
<td>B</td>
<td>7</td>
</tr>
<tr>
<td>C</td>
<td>6</td>
</tr>
<tr>
<td>D</td>
<td>5</td>
</tr>
<tr>
<td>F</td>
<td>0 - 4</td>
</tr>
</tbody>
</table>

**Notes:**
This course satisfies the Systems breadth and the project requirement.
This course is C++ programming project heavy.
Pre-requisites:

- CS 213: Introduction to Computer Systems (or equivalent)
- CS 214: Data Structures and Data Management

While we assume you know and remember the concepts thought in the classes listed above, peer mentors will run tutorials during their first half of their office hours to refresh your memory. While the participation to these tutorials is not mandatory, it is strongly suggested.

Recommended classes:

- CS 321: Programming Languages

Other compiler-heavy classes:

- CS 323: Code Analysis and Transformation
- CS 397/497: Advanced Topics in Compilers

Northwestern University is committed to providing the most accessible learning environment as possible for students with disabilities. Should you anticipate or experience disability-related barriers in the academic setting, please contact AccessibleNU to move forward with the university’s established accommodation process (e: accessiblenu@northwestern.edu; p: 847-467-5530). If you already have established accommodations with AccessibleNU, please let me know as soon as possible, preferably within the first two weeks of the term, so we can work together to implement your disability accommodations. Disability information, including academic accommodations, is confidential under the Family Educational Rights and Privacy Act.