Announcements

- Syllabus: posted on website
- Academic Integrity
BUILD A COMPILER!
Why?

- Deeper understanding of languages
- Become a better programmer
- Learn how to build tools
- Build special-purpose languages (DSLs)
- Theory meets practice
- High-level meets low-level
Assessment plan

- **Homework (20%)**
  - about five
  - mostly final exam prep

- **Project (60%)**
  - multiple phases
  - team-based

- **Final Exam (20%)**
  - most likely a take-home final
  - I will poll class to find out time-zone and other constraints students may have
<table>
<thead>
<tr>
<th>Learning outcome</th>
<th>Instructional methods</th>
<th>Assessment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Identify and describe the function of the major phases of a compiler.</td>
<td>Lecture-based instruction</td>
<td>HW, EX, PRES</td>
</tr>
<tr>
<td>Define formally the grammars used in the front end of a compiler, their application in the front end, and techniques for parsing such grammars.</td>
<td>Hands-on lecture activities</td>
<td></td>
</tr>
<tr>
<td>Evaluate (compare and contrast) different intermediate representations.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Explain the compiler’s role in creating and managing run-time environments.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Explain and evaluate (compare and contrast) different approaches to code generation.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Identify and explain the applicability and operation of code optimizations.</td>
<td>Lecture-based instruction</td>
<td></td>
</tr>
<tr>
<td>Build both the front and back ends of a compiler.</td>
<td>Team project w/team-faculty meetings</td>
<td>PROJ, PRES</td>
</tr>
<tr>
<td>Collaborate effectively as a member of a software development team.</td>
<td></td>
<td>PROJ, PRES</td>
</tr>
</tbody>
</table>
Teams & Meetings

- Form teams (no later than Tuesday next week).
- Teams must be of size 3 or 4.
- Teams must set up mandatory weekly meetings with me (~50 minutes).
- One member of each team must make a private post in Piazza with the UBIT of each person on their team.
- All code must be maintained in private git repo hosted on GitHub - I will help teams set this up at first meeting via GitHub Classroom - don't set this up on your own before then.
Goal: build a compiler

code

source program

executable
Phases of a compiler

source program

executable

Figure 1.6, page 5 of text
Deep understanding - ex 1

name

vs

identifier

vs

variable
name

y.x

identifier

x

refers to

variable
location
in
memory
void foo() {
    int x = 0;
    printf(x);
}

void bar() {
    double x = 3.8;
    printf(x);
}
int func(int x) {
    if (x == 0) { return 1; }
    else { return x * func(x-1); }
}
struct Pair {
    int x;
    int y;
};

void bar() {
    Pair r, s;
}
variables in distinct scopes, variables in distinct records/objects, or variables in distinct function invocations
Deep understanding – ex 2

order of evaluation

Does source code completely determine order of evaluation/execution at machine language level?
Deep understanding - ex 2

What is the order of evaluation?

\[ a + b \times c; \]
Deep understanding – ex 2

\[ f() + g() \ast h(); \]

What is the order of evaluation?
Deep understanding - ex 2

$f(0) + f(0) \times f(0)$;

What is the order of evaluation?
Deep understanding - ex 2

In most languages the result will be consistent with the evaluation of
\( a + ( b \times c ) \)
Deep understanding - ex 2

\[ a + b \times c; \]

Order of operations is important here, but order of evaluation of the variables a, b, and c is not (as long as they are evaluated before they are needed).
Deep understanding - ex 2

```
  a++ + a++ * a++;
```

Order of operations is important here, but order of evaluation of the variables is not . . . except if there are side effects!