Welcome to CS 536: 
Introduction to Programming Languages and Compilers!

Instructor: Beck Hasti
- hasti@cs.wisc.edu
- Office hours to be determined

TAs
- Andrey Yao
- Robert Nagel
- Sadman Sakib
- Saikumar Yadugiri
- Ting Cai

Course websites:
- canvas.wisc.edu
- www.piazza.com/wisc/spring2024/compsci536
- pages.cs.wisc.edu/~hasti/cs536

About the course
We will study compilers
We will understand how they work
We will build a full compiler

Course mechanics
Exams (60%)
- Midterm 1 (18%): Thursday, February 29, 7:30 – 9 pm
- Midterm 2 (16%): Thursday, March 21, 7:30 – 9 pm
- Final (26%): Sunday, May 5, 2:45 – 4:45 pm

Programming Assignments (40%)
- 6 programs: 5% + 7% + 7% + 7% + 7% + 7%

Homework Assignments
- 8 short homeworks (optional, not graded)
What is a compiler?

A compiler is
- recognizer of language S
- a translator from S to T
- a program in language H

Front end vs back end

front end = understand source code S; map S to IR
IR = intermediate representation
back end = map IR to T
Scanner

**Input:** characters from source program

**Output:** sequence of tokens

**Actions:**
- group characters into lexemes (tokens)
- identify and ignore whitespace, comments, etc.

**What errors can it catch?**
- bad characters
- unterminated strings
- integer literals that are too large

Parser

**Input:** sequence of tokens from the scanner

**Output:** AST (abstract syntax tree)

**Actions:**
- group tokens into sentences

**What errors can it catch?**
- syntax errors
- (possibly) *static semantic* errors

Semantic analyzer

**Input:** AST

**Output:** annotated AST

**Actions:** does more static semantic checks
- Name analysis
- Type checking

Intermediate code generator

**Input:** annotated AST

**Output:** intermediate representation (IR)
Example

\[ a = 2 \times b + \text{abs}(-71); \]

Scanner produces tokens:

AST (from parser)

Symbol table

3-address code
Optimizer

**Input:** IR

**Output:** optimized IR

**Actions:** improve code
- make it run faster, make it smaller
- several passes: local and global optimization
- more time spent in compilation; less time in execution

Code generator

**Input:** IR from optimizer

**Output:** target code

Symbol Table

**Compiler keeps track of names in**
- semantic analyzer
- code generation
- optimizer

**P1:** implement symbol table

Block-structured language
- nested visibility of names
- easy to tell which def of a name applies
- lifetime of data is bound to scope

**Example:** (from C)

```c
int x, y;

void A() {
    double x, z;
    C(x, y, z);
}

void B(){
    C(x, y, z);
}
```