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

Instructor: Beck Hasti
- hasti@cs.wisc.edu
- Office hours (in 5375 CS):
  - Tuesday 2:30 – 3:30 pm
  - Wednesday 1:00 – 2:00 pm
  - Friday 9:30 – 10:30 am
  - and by appointment

TAs
- Aoran Wu
- Nicholas Sorenson
- Evan Wireman

Course websites:
- pages.cs.wisc.edu/~hasti/cs536/
- www.piazza.com/wisc/spring2022/compsci536
- https://canvas.wisc.edu

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 (17%): Wednesday, March 2, 7:30 – 9 pm
- Midterm 2 (17%): Wednesday, March 30, 7:30 – 9 pm
- Final (26%): Sunday, May 8, 12:25 – 2:25 pm

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

Homework Assignments
- ~10 short homeworks (optional, not graded)
What is a compiler?

A compiler is
- a 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
Overview of typical compiler

Source program
  ^ sequence of characters
  |      
  v
Scanner
  ^ sequence of tokens
  |      
  v
Parser
  ^ AST
  |      
  v
Semantic analyzer
  ^ augmented, annotated AST
  |      
  v
Intermediate code generator
  ^ IR
  |      
  v
Optimizer
  ^ optimized IR
  |      
  v
Code generator
  ^ assembly or machine code
  |      
  v
Object program

Symbol table

Source program
  └── front end
      └── back end
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

\[ a = 2 \times b + \text{abs}(-71); \]
**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
- e.g., Java, C, C++

Ideas:
- nested visibility of names
- easy to tell which def of a name applies
- lifetime of date is bound to scope