Who?

Shachar Itzhaky
shachar@cs.technion.ac.il
Office hours should be scheduled in advance

TAs:
• Omer Katz
• Alex Sivak
• Yaakov Sokolik

What?

• Understand:
  ▶ What a compiler is
  ▶ How it works
  ▶ Proven techniques
    (most can be re-used in other settings)

How?

• What will help us:
  ▶ Textbooks
    • Modern Compiler Design
    • Compilers: Principles, Techniques & Tools
    • Modern Compiler Implementation in C
  ▶ Homework assignments
    ▶ "Dry": deepen understanding of theory
    ▶ "Wet": build a compiler yourself
  ▶ Ask questions, answer questions!

How Not?!

• “Your slides don’t have everything you say written on them” (common complaint)
  ▶ Yes, I know, this is by design
  ▶ Presentations are a teaching aid, not a substitute for coming to lectures
  ▶ If you don’t attend lectures or attend and don’t listen, you will inevitably miss some things
  ▶ If you want slides that have all the material written on them nicely, that format is indeed available and commonly known as a textbook
  ▶ See how horrible this slide is? This is why you won’t see many slides with as much text as this one for the rest of the course
Exam

- 75% of the final grade
- Look at Eran’s old exams from previous years
- Don’t worry too much...
  - If you attend lectures and finish the assignments, you should do well in the exam; if you don’t attend try to keep up with the material
  - Historical evidence — attending leads to higher pass rate in the final exam

ENIAC

“The Education of a Computer”
(Grace Hopper)

“Calculus I”

John Backus and team at IBM
The first complete compiler
What is a Compiler?

“A compiler is computer software that transforms computer code written in one programming language (the source language) into another language (the target language), ...primarily to a lower level language (e.g. assembly language, object code, or machine code) to create an executable program.”

-- Wikipedia
Compiler vs. Interpreter

Executable code

Source text

Semantic Representation

Backend (synthesis)

Frontend (analysis)

Output

Compiler vs. Interpreter

Input

\( b = a^2 + 1 \)

\( b = a^2 + 1 \)

\( b = a^2 + 1 \)

Just-in-time Compiler

(Javascript example)

The compiled code is optimized dynamically at runtime, based on runtime behavior

Just-in-time (JIT) Compiler

(Java example)

Just-in-time (JIT) compilation: bytecode interpreter (in the JVM) compiles program fragments during interpretation to avoid expensive re-interpretation.

Anatomy of a Compiler: Why?
### Why should you care?

- Every person in this class will build a parser some day
  - Or wish they knew how to build one...
- Better understanding of programming languages
- Understand internals of compilers
- Understand (some) details of target architectures
- Useful techniques and algorithms
  - Lexical analysis / parsing
  - Semantic representation
  - Abstraction layers
  - Modularity

### Why should you care?

- Useful formalisms
  - Regular expressions
  - Context-free grammars
- Data structures
- Algorithms

### Course Overview
$x = b^2 - 4ac$
Error Checking

In every stage:

- **Lexical analysis**: illegal tokens
- **Syntax analysis**: illegal syntax
- **Semantic analysis**: incompatible types, undefined variables, ...
- **Even runtime**: division by zero, array bounds,...
- Every phase tries to recover and proceed with compilation

Lexical Errors

- `pi = 3.141.562`
  - Illegal token
- `pi = oranges`
  - Illegal token
- `pi = oranges3`
  - `(ID("pi"), .EQ., ID("oranges3"))`

Syntax Errors

- `x = / oranges`
  - Wrong number of arguments to operator `/`
- `x = func(int a)`
  - A declaration is not expected here
Semantic Errors: Type Checking

Runtime Errors

The Real Anatomy of a Compiler

Optimizations

Loop Hoisting

```java
for (int i = 0; i < 100; ++i) {
    array[i] = x + y;
}
```

```java
int t = x + y;
for (int i = 0; i < 100; ++i) {
    array[i] = t;
}
```
Loop Unrolling

```cpp
for (int i = 0; i < 100; ++i) {
    delete array[i];
}
```

```cpp
for (int i = 0; i < 100; i += 5) {
    delete array[i];
    delete array[i+1];
    delete array[i+2];
    delete array[i+3];
    delete array[i+4];
}
```

Machine code generation

- Register allocation
  - Optimal register allocation is NP-Complete
  - In practice, known heuristics perform well
- Assigning variables to memory locations
- Instruction selection
  - Convert IR to actual machine instructions
- Modern architecture challenges
  - Multicores
  - Memory hierarchies
  - SIMD instructions

Compiler Construction Toolset

- Lexical scanner generators
  - Flex
- Parser generators
  - Bison

Summary

- Compiler = a program that translates code from source language (high level) to target language (low level)
- Compilers play a critical role
  - Bridge programming languages to the machine
  - Many useful techniques and algorithms
  - Many useful tools (e.g., lexer/parser generators)
- Compiler constructed from modular phases
  - Reusable
  - Different front/back ends

Next