What?

- Understand
  - what is a compiler
  - how does it work
  - techniques that can be re-used in other settings

- What will help us
  - Text books
    - Modern compiler design
    - Compilers: principles, techniques and tools
  - 5 homework assignments

- Will also help
  - Taking a deep breath
  - Focusing on material and not on your grade

What is a Compiler?

- “A compiler is a computer program that transforms source code written in a programming language (source language) into another language (target language). The most common reason for wanting to transform source code is to create an executable program.”

-- Wikipedia
What is a Compiler?

source language
C
C++
Pascal
Python
Ruby
Prolog
Lisp
Scheme
ML
OCaml

target language
IA32
IA64
SPARC

Compiler

Executable
code
exe

Source
text
txt

Semantic
Representation

Backend
(synthesis)

int a, b;
a = 2;
b = a*2 + 1;

MOV R1,2
SAL R1
INC R1
MOV R2,R1

Anatomy of a Compiler

Modularity

int a, b;
a = 2;
b = a*2 + 1;

SET   R1,2
STORE #0,R1
SHIFT R1,1
ADD   R1,1
STORE #2,R1
Anatomy of a Compiler

Compiler

Frontend (analysis)  Semantic Representation  Backend (synthesis)  Executable code

Semantic Representation

Frontend (analysis)

Source text

int a, b;
a = 2;
b = a*2 + 1;

MOV R1,2
SAL R1
INC R1
MOV R2,R1

Compiler vs. Interpreter

Compiler

Frontend (analysis)  Semantic Representation  Backend (synthesis)  Executable code

Semantic Representation

Frontend (analysis)

Source text

int a, b;
a = 2;
b = a*2 + 1;

MOV R1,8(ebp)
SAL R1
INC R1
MOV R2,R1

Compiler vs. Interpreter

Interpreter

Frontend (analysis)  Semantic Representation  Execution Engine  Output

Semantic Representation

Frontend (analysis)

Source text

Input

int a, b;
a = 2;
b = a*2 + 1;

MOV R1,3
SAL R1
INC R1
MOV R2,R1
Just-in-time Compiler (Java example)

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

Why should you care?

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

Why should you care?

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

Course Overview

Compiler

Source text

Lexical Analysis

Syntax Analysis

Semantic Analysis

Inter. Rep. (IR)

Code Gen.

Executable code

Runtime environment
Garbage collection
Architecture
Journey inside a compiler

x = b*b - 4*a*c

Token Stream

<ID,"x"> <EQ> <ID,"b"> <MUL T> <ID,"b"> <MINUS> <INT,4> <MUL T> <ID,"a"> <MUL T> <ID,"c">
Error Checking

- In every stage...
- Lexical analysis: illegal tokens
- Syntax analysis: illegal syntax
- Semantic analysis: incompatible types, undefined variables, ...
- Every phase tries to recover and proceed with compilation (why?)
  - Divergence is a challenge

Errors in lexical analysis

- Illegal token
  - $\text{pi} = 3.141562$
  - $\text{pi} = \text{3oranges}$
  - $\text{pi} = \text{oranges3}$
Optimizations

- "Optimal code" is out of reach
  - many problems are undecidable or too expensive (NP-complete)
  - Use approximation and/or heuristics

- Loop optimizations: hoisting, unrolling, ...
- Peephole optimizations
- Constant propagation
  - Leverage compile-time information to save work at runtime (pre-computation)
- Dead code elimination
  - space
- ...

Machine code generation

- Register allocation
  - Optimal register assignment is NP-Complete
  - In practice, known heuristics perform well
- assign variables to memory locations
- Instruction selection
  - Convert IR to actual machine instructions

- Modern architectures
  - Multicores
  - Challenging memory hierarchies
Compiler Construction Toolset

- Lexical analysis generators
  - lex
- Parser generators
  - yacc
- Syntax-directed translators
- Dataflow analysis engines

Summary

- Compiler is a program that translates code from source language to target language
- Compilers play a critical role
  - Bridge from 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

Coming up next

- Lexical analysis