What is a Compiler?

A function that maps an input string to an output string.

compiler :: String -> String

Typically, the input and output strings are “programs”

compiler :: SourceProgram -> TargetProgram

Key Requirements on output program:

- Has the same meaning (“semantics”) as input.
- Is executable in relevant context (VM, microprocessor, web browser).
Compilers were invented to **avoid writing machine code by hand**

**From Binary to FORTRAN**

Richard Hamming – The Art of Doing Science and Engineering, p25:

*In the beginning we programmed in absolute binary... Finally, a Symbolic Assembly Program was devised - after more years than you are apt to believe during which most programmers continued their heroic absolute binary programming. At the time [the assembler] first appeared I would guess about 1% of the older programmers were interested in it - using [assembly] was “sissy stuff”**, and a real programmer would not stoop to wasting machine capacity to do the assembly.*

John A.N. Lee, Dept of Computer Science, Virginia Polytechnical Institute

*One of von Neumann’s students at Princeton recalled that graduate students were being used to hand assemble programs into binary for their early machine. This student took time out to build an assembler, but when von Neumann found out about it he was very angry, saying that it was a waste of a valuable scientific computing instrument to use it to do clerical work.*
What does a Compiler look like?

An input source program is converted to an executable binary in many stages:
- Parsed into a data structure called an Abstract Syntax Tree
- Checked to make sure code is well-formed (and well-typed)
- Simplified into a convenient Intermediate Representation
- Optimized into (equivalent but) faster program
- Generated into assembly x86
- Linked against a run-time (usually written in C)

What is CSE 110A?

A bridge between two worlds
- High-level: Haskell (CSE 116)
- Machine Code: X86/ARM (CSE 12)

How to write a compiler for NanoML -> X86
- Parsing
- Checking & Validation
- Simplification & Normalizing
- Optimization
- Code Generation

What is CSE 110A?

But also, how to write complex programs
- Design
- Implement
- Test
- Iterate
How write a Compiler?

General recipe, applies to any large system
• gradually, one feature at a time!

We will
• Step 1 Start with a teeny tiny language,
• Step 2 Build a full compiler for it,
• Step 3 Add a few features,
• Go to Step 2.

(Yes, loops forever, but we will hit Ctrl-C in 10 weeks...)

How will we grade?

(30%) Assignments
• 7 assignments, first one up today
• All programming
• Groups of up to 2 (except for HW#0)

(30%) Midterm
• In-class (2-sided "cheat sheet")

(35%) Final
• 2-sided “cheat sheet”
• Quizzes (bring phone or laptop to class)
• Attempting to answer > 75% questions
• Starting from next week

(5%) Piazza Extra Credit
• To top-10 best participants

Course Outline

Write a compiler for NanoML → X86

But Rome wasn’t built in a day … and neither is any serious software.

So we will write many compilers:
• Numbers and increment/decrement
• Local Variables
• Nested Binary Operations
• Booleans, Branches and Dynamic Types
• Functions
• Tuples and Structures
• Lambdas and closures
• Types and inference
• Garbage Collection
What will you learn?

Core principles of compiler construction
• Managing Stacks & Heap
• Type Checking
• Intermediate forms
• Optimization

Several new languages
• Haskell to write the compiler
• C to write the “run-time”
• X86 compilation target

More importantly how to write a large program
• How to use types for design
• How to add new features / refactor
• How to test & validate

What do you need to know?

This 110A uses many concepts from CSE 116
• Familiarity with Functional Programming
• Datatypes (e.g. Lists, Trees, ADTs)
• Polymorphism
• Recursion
• HOFs (e.g. `map`, `filter`, `fold`)

However: there will be lots of support for those picking up Haskell “as they go”

Also depends on CSE 12
• Experience with some C programming
• Experience with some assembly (x86)

Questions?