CSE 110A: Winter 2020

Fundamentals of Compiler Design I

Introduction and Overview

Owen Arden

UC Santa Cruz

Based on course materials developed by Ranjit Jhala

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
```

What is a Compiler?

For example, here are some well-known *compilers*

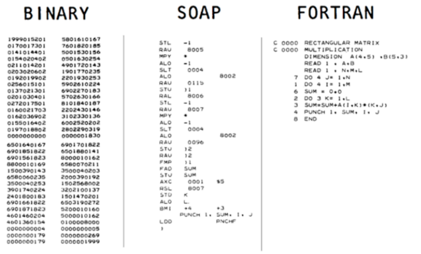
```
gcc, clang :: C          -> Binary         -- a.out, .exe
ghc        :: Haskell    -> Binary
javac      :: Java       -> JvmByteCode    -- .class
scalac     :: Scala      -> JvmByteCode
ocamlc     :: Ocaml      -> OcamlByteCode  -- .cmo
ocamlopt   :: Ocaml      -> Binary
gwt        :: Java       -> JavaScript     -- .js
v8         :: JavaScript -> Binary
nasm       :: X86        -> Binary
pdftex     :: LaTeX      -> PDF
pandoc     :: Markdown   -> PDF | Html | Doc
```

Key Requirements on output program:

- Has the *same meaning* ("semantics") as input,
- Is *executable* in relevant *context* (VM, microprocessor, web browser).

A Bit of History

Compilers were invented to avoid writing machine code by hand



BINARY

```
1999015201   5801610167
0170017301   7601820185
0141014401   5001530156
0154020402   0501630254
0211014201   4901720143
0203020602   1901770235
0192019902   2201930253
0256015101   5902610224
0137021301   6902270183
0201030401   5702630166
0272017501   8101840187
0160021703   2202430146
0162036902   3102330136
0155016402   6002520202
0197018802   2802290319
0000000000   0000001830

6501640167   6901701822
6901851822   6501880141
6901561823   8000010162
8800010169   6580070211
1500390143   3500040203
6580060235   2000390192
3500040253   1502568002
3901740224   3202100137
2401800183   1501470201
6901661822   6503190272
6901871823   5200010160
4601460204   5000010162
4601360154   0100008000
0000000004   0000000005
0000000179   0000000269
0000000179   0000001999
```

SOAP

```
STL   -1
RAU    8005
MPY   *
ALO   -1
SLT    0004
ALO              8002
RAU    0115
STU   )1
RAL    8006
STL   -1
RAU    8007
MPY   *
ALO   -1
SLT    0004
ALO              8002
RAU    0096
STU   )2
RAU   )2
FMP   )1
FAD   SUM
STU   SUM
AXC    0001    $5
RSL    8007
STD   K
ALO   L
BMI   +4       +3
     PUNCH 1, SUM, I, J
LDD            PNCHF
)
```

FORTRAN

```
C 0000 RECTANGULAR MATRIX
C 0000 MULTIPLICATION
       DIMENSION  A(4,5) ,B(5,3)
       READ 1 , A,B
       READ 1 , N,M,L
     7 DO 4 J= 1,N
     1 DO 4 I= 1,M
     6 SUM = 0.0
     2 DO 3 K= 1,L
     3 SUM=SUM+A(I,K)*(K,J)
     4 PUNCH 1, SUM, I, J
     8 END
```

From Binary to FORTRAN

A Bit of History

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.

A Bit of History

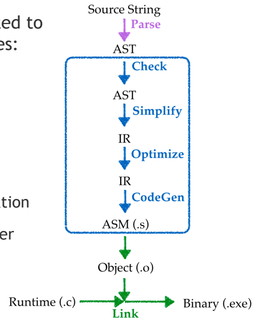
John A.N. Lee, Dept of Computer Science, Virginia Polytechnical Institute

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)



Compiler Pipeline

What is CSE 110A ?

A *bridge* between two worlds

- *High-level:* Haskell (**CSE 116**)
- *Machine Code:* X86/ARM (**CSE 12**)

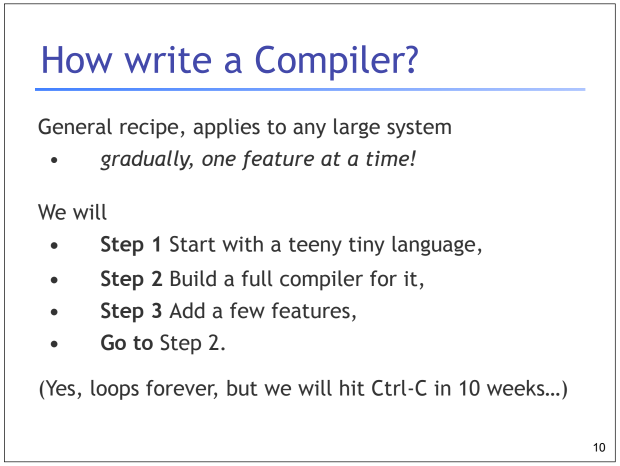
How to write **a compiler** for NanoML -> X86

- 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?

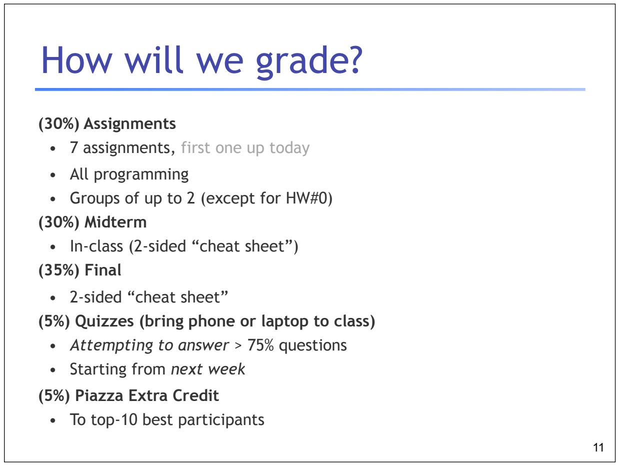
General recipe, applies to any large system

- *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

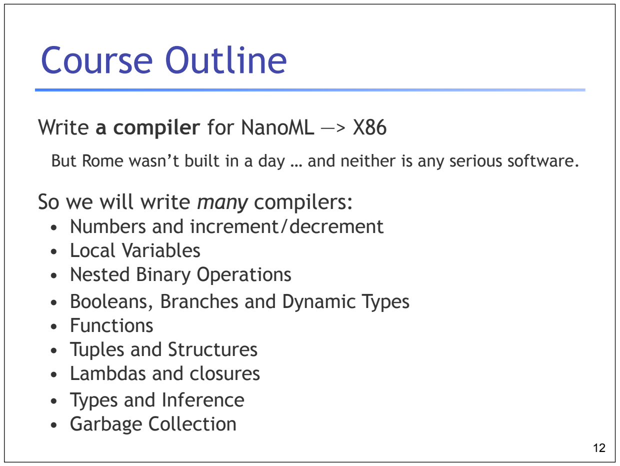
- 2-sided “cheat sheet”

(5%) 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?