COS320: Compiling Techniques

Zak Kincaid

February 5, 2019
Welcome!

- **Instructor**: Zak Kincaid
- **TAs**: Chirag Bharadwaj, Qinshi Wang

- **Website**: [http://www.cs.princeton.edu/courses/archive/spring19/cos320/](http://www.cs.princeton.edu/courses/archive/spring19/cos320/)
- **Office hours**: see website
What is a compiler?

- A compiler is a program that takes a program written in a source language and translates it into a functionally equivalent program in a target language.
  - Source languages: C, Java, OCaml, ...
  - Target languages: x86 Assembly, Java bytecode, C, ...
What is a compiler?

- A compiler is a program that takes a program written in a source language and translates it into a functionally equivalent program in a target language.
  - Source languages: C, Java, OCaml, ...
  - Target languages: x86 Assembly, Java bytecode, C, ...

- A compiler can also
  - Report errors & potential problems
    - Uninitialized variables, type errors, ...
  - Improve ("optimize") the program
Why take COS320?

You will learn:

• How high-level languages are translated to machine language
• How to be a better programmer
  • What can a compiler do?
  • What can a compiler *not* do?
• Lexing & Parsing
• (Some) functional programming in OCaml
• A bit of programming language theory
• A bit of computer architecture
Course resources

- Recommended textbook: Modern compiler implementation in ML (Appel)
- Real World OCaml (Minsky, Madhavapeddy, Hickey)
  realworldocaml.org
Grading

• 60% Homework
  • 6 assignments, evenly weighted
  • HW1: OCaml introduction
  • HW2: Build an x86 simulator
  • HW3-6: Build a compiler

• 20% Midterm
  • March 14, in class

• 20% Final
Homework policies

• Except for HW1, homework can be done individually or in pairs.
• Late assignments will be penalized 1% per hour past the deadline.
• Five late passes, can submit up to 24 hours late without penalty (at most 3/HW).

Feel free to discuss with others at conceptual level.
Submitted work should be your own.
Lecture expectations

- Lecture 1: Intro
- Lecture 2: OCaml (review COS326)
- Lecture 3: x86 (review COS217)
- Lecture 4 + k: not review
Compilers
(Programming) language = syntax + semantics

- **Syntax**: what sequences of characters are valid programs?
  - Typically specified by context-free grammar
    
    \[
    <\text{expr}> ::= <\text{integer}>
    \mid <\text{variable}>
    \mid <\text{expr}> + <\text{expr}>
    \mid <\text{expr}> * <\text{expr}>
    \mid (<\text{expr}>)
    \]

- **Semantics**: what is the behavior of a valid program?
  - *Operational semantics*: how can we execute a program?
    - In essence: an interpreter
  - *Axiomatic semantics*: what can we prove about a program?
  - *Denotational semantics*: what mathematical function does the program compute?
(Programming) language = syntax + semantics

- **Syntax**: what sequences of characters are valid programs?
  - Typically specified by context-free grammar
    
    
    $\texttt{<expr>} ::= \texttt{<integer>}$
    $| \texttt{<variable>}$
    $| \texttt{<expr>} + \texttt{<expr>}$
    $| \texttt{<expr>} * \texttt{<expr>}$
    $| (\texttt{<expr>})$

- **Semantics**: what is the behavior of a valid program?
  - *Operational semantics*: how can we execute a program?
    - In essence: an interpreter
  - *Axiomatic semantics*: what can we prove about a program?
  - *Denotational semantics*: what mathematical function does the program compute?

The job of a compiler is to translate from the syntax of one language to another, but **preserve the semantics**.
```c
#include <stdio.h>

int factorial(int n) {
    int acc = 1;
    while (n > 0) {
        acc = acc * n;
        n = n - 1;
    }
    return acc;
}

int main(int argc, char *argv[]) {
    printf("factorial(6) = %d\n", factorial(6));
}
```
```assembly
__factorial:
## BB#0:
pushl  %ebp
movl  %esp, %ebp
subl  $8, %esp
movl  8(%ebp), %eax
movl  %eax, -4(%ebp)
movl  $1, -8(%ebp)
LBB0_1:
cmpl  $0, -4(%ebp)
jle   LBB0_3
## BB#2:
movl  -8(%ebp), %eax
imull -4(%ebp), %eax
movl  %eax, -8(%ebp)
movl  -4(%ebp), %eax
subl  $1, %eax
movl  %eax, -4(%ebp)
jmp   LBB0_1
LBB0_3:
movl  -8(%ebp), %eax
addl  $8, %esp
popl  %esp
retl  
```
Compiler phases (simplified)

1. **Source text**
2. **Lexing**
3. **Token stream**
4. **Parsing**
5. **Abstract syntax tree**
6. **Translation**
7. **Intermediate representation**
8. **Optimization**
9. **Code generation**
10. **Assembly**
COS320 assignments

By the end of the course, you will build (in OCaml) a complete compiler from a high-level type-safe language (“Oat”) to a subset of x86 assembly.

- HW1: OCaml programming
- HW2: X86lite interpreter
- HW3: LLVMlite compiler
- HW4: Lexing, Parsing, simple compilation
- HW5: Higher-level Features
- HW6: Analysis and Optimizations

We will use the assignments from Penn’s CIS 354, provided by Steve Zdancevic.
OCaml
• Why OCaml?
  • Algebraic data types + pattern matching are very convenient features for writing compilers
• OCaml is a *functional* programming language
  • *Imperative* languages operate by mutating data
  • *Functional* languages operate by producing new data
• OCaml is a *typed* language
  • Contracts on the values produced and consumed by each expression
  • Types are (for the most part) *automatically inferred*.
    • Good style to write types for top-level definitions
Preparation

• Excellent preparation: COS326 (Functional programming)
  • More than you will need for this class.

• Thursday’s lecture + review sessions
  • Poll on Piazza
HW1: Hellocaml

• Available now on the course website
  • Topic: OCaml introduction + interpreter & compiler for a little calculator language
• OCaml dev environment on VirtualBox virtual machine
  • Recommend Emacs + merlin