Lecture 25

CMSC 350: COMPILER DESIGN
COURSE WRAP-UP

What have we learned?
Where else is it applicable?
What next?
Why CMSC 350?

• You will learn:
  – Practical applications of theory
  – Parsing
  – How high-level languages are implemented in machine language
  – A deeper understanding of code
  – A little about programming language semantics
  – Functional programming in Haskell
  – How to manipulate complex data structures
  – How to be a better programmer
  – How to work with a partner
  – Version control

• Did we meet these goals?
Stuff we didn’t Cover

- We skipped stuff at every level…
- Concrete syntax/parsing:
  - Much more to the theory of parsing…
  - Good syntax is art not science!
- Source language features:
  - Exceptions, recursive data types (easy!), advanced type systems, type inference, concurrency
- Intermediate languages:
  - Intermediate language design, bytecode, bytecode interpreters, just-in-time compilation (JIT)
- Compilation:
  - Continuation-passing transformation, efficient representations, scalability
- Optimization:
  - Scientific computing, cache optimization, instruction selection/optimization
- Runtime support:
  - Memory management, garbage collections
Where to go from here?

• Conferences (proceedings available on the web):
  – Programming Language Design and Implementation (PLDI)
  – Principles of Programming Languages (POPL)
  – Object Oriented Programming Systems, Languages & Applications (OOPSLA)
  – International Conference on Functional Programming (ICFP)
  – European Symposium on Programming (ESOP)
  – ...

• Technologies / Open Source Projects
  – Yacc, lex, bison, flex, …
  – LLVM – low level virtual machine
  – Java virtual machine (JVM), Microsoft’s Common Language Runtime (CLR)
  – Languages: OCaml, F#, Haskell, Scala, Go, Rust, …?
Where else is this stuff applicable?

- General programming
  - In C/C++, better understanding of how the compiler works can help you generate better code.
  - Ability to read assembly output from compiler
  - Experience with functional programming can give you different ways to think about how to solve a problem

- Writing domain specific languages
  - lex/yacc very useful for little utilities
  - understanding abstract syntax and interpretation

- Understanding hardware/software interface
  - Different devices have different instruction sets, programming models
Thanks!