Lecture 1. Introduction

Wei Le

2015.8
Instructors Team

- Prof Wei Le: Atanasoff 210 (weile@iastate.edu)
- TA: Jucheol Moon (moon@iastate.edu)
- Office hours: Th 2-3pm, (TA) W 9-10am
Textbooks and Resources

- Compilers: Principles, Techniques, and Tools
  - Alfred V. Aho, Ravi Sethi, Jeffrey D. Ullman

  - Alfred V. Aho, Monica S. Lam, Ravi Sethi, Jeffrey D. Ullman

- Engineering a Compiler, Second Edition
  - Keith D. Cooper & Linda Torczon

- Modern Compiler Implementation in C
  - Andrew W. Appel
Evaluation

- Class participation (5%): Please attend the class on time and participate in the class discussions.
- Exams (35%): Mid-term (15%), Final (20%)
- Written assignments (15%): Answer questions to help understand the theoretical aspects of compilers.
- Programming Assignments (45%): You will work in a team (up to 3 students) to write a compiler for the *Cool* language.
How I Run the Course

- Lecture-based
- Course web for slides and assignments
- Blackboard for submitting homework
- Emails for questions
Why Study Compilers? – informal version

- Course credit :(

- Interesting :)

- Computer scientist: foundational knowledge of CS

- System building

- Working on compilers to earn big money

- Compiler concepts in daily software development career (parsing, optimization ...)

Why Study Compilers? – informal version

- Course credit :(  
- Interesting :)

Computer scientist: foundational knowledge of CS  
System building  
Working on compilers to earn big money  
Compiler concepts in daily software development career (parsing, optimization ... )
Why Study Compilers? – informal version

- Course credit :(  
- Interesting :)  
- Computer scientist: foundational knowledge of CS
Why Study Compilers? – informal version

- Course credit :(  
- Interesting :) 
- Computer scientist: foundational knowledge of CS 
- System building
Why Study Compilers? – informal version

- Course credit :(  
- Interesting :)  
- Computer scientist: foundational knowledge of CS  
- System building  
- Working on compilers to earn big money
Why Study Compilers? – informal version

- Course credit :
- Interesting :
- Computer scientist: foundational knowledge of CS
- System building
- Working on compilers to earn big money
- Compiler concepts in daily software development career (parsing, optimization ... )
Compiler

- a translator from one language to another
Compiler

- a translator from one language to another
- translating programming languages to executable

gcc 5.2.0 (total lines 9 million, lines of code 6 million, files 96 k, majority of the code in C)

Python 2.7.0 (total lines 548 k, lines of code 391 k, files 811, majority of the code is in C)
Compiler

- a translator from one language to another
- translating programming languages to executable
- implementation of programming languages
Compiler

- a translator from one language to another
- translating programming languages to executable
- implementation of programming languages
- complex software
  - users: developers
  - specification: syntax and semantics of a language
  - can be millions lines of code:
    - gcc 5.2.0 (total lines 9 million, lines of code 6 million, files 96 k, majority of the code in C)
    - Python 2.7.0 (total lines 548 k, lines of code 391 k, files 811, majority of the code is in C)
Compiler

- recognize legal and illegal programs
- generate correct code
- manage storage of all variables and code
- agree with OS and linker in terms of the format of the code
Compiler and Interpreter

Compilers:
- translate the entire project to executable
- run executable with data

Interpreter
- run one instruction with data (like a shell command)
Why Study Compilers: Intellectually Interesting

The science of building a compiler:

- **NP-Complete problems:**
  - Register allocation
- **Algorithms**
  - Fixed-point algorithm
  - Greedy algorithm
- **Mathematical models:** finite state machine, regular expression, context-free grammars
- **Data structures used in implementation:** trees
- **Code optimization:** theory support on why the optimization is correct, why it can improve performance, reasonable time for compilation,

Important contributions to software field:

- Backus-Naur form (BNF)
- Abstract interpretation
Why Study Compilers: Practicality

- Parsing routines: strings (web URL), files (jpeg)
- Compiler teams: Microsoft, Wall Street Companies, Huawei, Firefox...
The Motivation

- 1954 IBM develops the 704, Successor to the 701
- Problem: Software costs exceeded hardware costs!
- All programming done in assembly
1953, "Speed coding" – An interpreter for mathematics on IBM 701 and IBM 650 written by John Backus
Ran 10-20 times slower than hand-written assembly
The first compiler – goal: reduce the cost of software
Translate high-level code to assembly
Many thought this impossible
Had already failed in other projects
1954-7: FORTRAN project

1958, 50% of the programs written in FORTRAN, development time halved

John Backus: the 1977 ACM Turing Award for profound, influential, and lasting contributions to the design of practical high-level programming systems, notably through his work on FORTRAN, and for publication of formal procedures for the specification of programming languages.
FORTRAN I

- Huge impact on computer science
- Led to an enormous body of theoretical work
- Modern compilers preserve the outlines of FORTRAN
A Language Processing System

source program

Preprocessor

modified source program

Compiler

target assembly program

Assembler

relocatable machine code

Linker/Loader

library files
relocatable object files

target machine code
The Structure of A Compiler: Translating an Assign Statement

```
position = initial + rate * 60
```

- **Lexical Analyzer**
- **Syntax Analyzer**
- **Semantic Analyzer**
- **Intermediate Code Generator**
  - \( t1 = \text{inttofloat}(60) \)
  - \( t2 = \text{id3} \times t1 \)
  - \( t3 = \text{id2} + t2 \)
  - \( \text{id1} = t3 \)
- **Code Optimizer**
  - \( t1 = \text{id3} \times 60.0 \)
  - \( \text{id1} = \text{id2} + t1 \)
- **Code Generator**
  - \( \text{LDF R2, id3} \)
  - \( \text{MULF R2, R2, #60.0} \)
  - \( \text{LDF R1, id2} \)
Target Code - Various forms

- Assembly Code - symbolic instruction and addresses (modern compiler does not use it, too slow)
- Relocatable format: Binary form except external references, instruction addresses and data addresses not bound to address; need linker and loader (Both assembly & relocatable allow program modules to be separately compiled)
- Another language
Front, (Middle), Back End

- Front end: is this a valid program? machine independent, language dependent
  - input: source code
  - output: Intermediate Representation (IR)
- Middle end: compiler optimization
  - input: IR
  - output: IR
- Back end: code generation, machine dependent, language independent
  - input: IR
  - output: target machine code
  - Instruction selection, scheduling, register allocation
Phases and Passes

- Phases - conceptual and sometimes physical stages, Symbol table coordinating information between phases
- Passes - (implementation): a program representation go through number of times – front end groups into a pass, the middle end is a pass, the back end is a pass (Further Reading Multi-pass compiler)