CSCI 742 - Compiler Construction

Lecture 1
Course Overview
Instructor: Hossein Hojjat

January 23, 2017
What is a Compiler?

THE #1 PROGRAMMER EXCUSE FOR LEGITIMATELY SLACKING OFF:
“MY CODE’S COMPILING.”

HEY! GET BACK TO WORK!
COMPILING!
OH, CARRY ON.
Compiler

- Compiler is a program that translates high-level programs into equivalent low-level programs.

```
Compiler

Source Program ➔ Compiler ➔ Target Program

Error (Warning)
```

- What is this course about?
- This course is about “compiler construction”:
  1- you will learn how to construct compilers (theory)
  2- you will construct your own compiler (practice)
You will implement a compiler for a small language
• (syntax similar to Java)
Source Code vs. Machine Code

**Source Code:**
- Written in high-level programming language (e.g. Java)
- Human-readable notation
- Expressive: variety of constructs to represent computations
- Redundant: helps programmers avoid errors

**Assembly (Machine) Code:**
- Optimized for hardware execution
- Basic commands that move bits around in registers and memory
- Redundancy decreased
- Information about source code structure lost
• Compiler translates a high-level programming language to a low-level programming language
• How does a compiler work?
From High-level to Low-level Code

- Compiler translates a high-level programming language to a low-level programming language
- How does a compiler work?
- Compiler uses a series of different program Intermediate Representations (IRs)
- Different IRs are suitable for different program manipulations (analysis, optimization, code generation)
Compiler Major Phases

Source Code (concrete syntax)

(if \((x == 0)\) \(x = x + 1;\))

Token Stream

(if (x == 0) x = x + 1;)

Lexical Analysis

Syntax Analysis (Parsing)

Semantic Analysis (Name Analysis, Type Analysis, ...)

Error

Abstract Syntax Tree (AST)

(\(=\) \(x\) \(0\) \(=\) \(x\) \(+\) \(x\) \(+\) \(1\))

Attributed AST

Code Generation

Machine Code

16: iload_2
17: ifne 24
20: iload_2
21: iconst_1
22: iadd
23: istore_2
24: ...
Main Project

- Implement a complete compiler for a small object-oriented language
Main Project

- Implement a complete compiler for a small object-oriented language

10%: Lexical Analysis (Scanner)
10%: Syntax Analysis (Parser)
10%: Semantic Analysis (Name Analyzer)
10%: Semantic Analysis (Type Analyzer)
10%: Code Generation
10%: Optimization (Bonus: +5%)

- 60% (65% with bonus) of your final grade is your compiler project
Interpreters vs. Compilers

**Interpreter**
Reads a source program and produces the results of executing that program

**Compiler**
Translates a program from high-level source program to low-level target program

- **language 1 (source)**
  - Compiler
  - language 2 (target)
  - results, behavior

Interpreter appears to execute a source program as if it were machine language
Interpreters vs. Compilers

**Difficulty**
- Usually it is easier to build an interpreter than a compiler

**Errors**
- Interpreter executes source program from first line, stops execution only when it encounters an error
- Compiler does not translate source program with error

**Optimization**
- Compiler preprocesses and analyzes source program
- Optimizing compiler can generate code that is far faster than interpretation
- Until 2013 Facebook was translating PHP (interpreted language) to C++
### Optimization Example

#### Constant Propagation

```plaintext
a = 7;
b = 2;
...
x = a - b;
**while**(x < 10){
    ...
}
```

```plaintext
a = 7;
b = 2;
...
x = 7 - 2;
**while**(x < 10){
    ...
}
```

#### Constant Folding

```plaintext
a = 7;
b = 2;
...
x = 7 - 2;
**while**(x < 10){
    ...
}
```

```plaintext
a = 7;
b = 2;
...
x = 5;
**while**(x < 10){
    ...
}
```
Course Work

Compiler Phases:

<table>
<thead>
<tr>
<th>Percentage</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>10%</td>
<td>Lexical Analysis (Scanner)</td>
</tr>
<tr>
<td>10%</td>
<td>Syntax Analysis (Parser)</td>
</tr>
<tr>
<td>10%</td>
<td>Semantic Analysis (Name Analyzer)</td>
</tr>
<tr>
<td>10%</td>
<td>Semantic Analysis (Type Analyzer)</td>
</tr>
<tr>
<td>10%</td>
<td>Code Generation</td>
</tr>
<tr>
<td>10%</td>
<td>Optimization (Bonus: +5%)</td>
</tr>
</tbody>
</table>

• Compiler Optimization Race:
  - We measure the size of the generated code of your compilers
  - Winner gets an extra 5% bonus!
Course Work

5%: Interpreter for a small language (while language)

Compiler Phases:

10%: Lexical Analysis (Scanner)
10%: Syntax Analysis (Parser)
10%: Semantic Analysis (Name Analyzer)
10%: Semantic Analysis (Type Analyzer)
10%: Code Generation
10%: Optimization (Bonus: +5%)

• Compiler Optimization Race:
  - We measure the size of the generated code of your compilers
  - Winner gets an extra 5% bonus!
Course Work

5%: Attendance & Participation
5%: Interpreter for a small language (while language)

Compiler Phases:

<table>
<thead>
<tr>
<th>10%:</th>
<th>Lexical Analysis (Scanner)</th>
</tr>
</thead>
<tbody>
<tr>
<td>10%:</td>
<td>Syntax Analysis (Parser)</td>
</tr>
<tr>
<td>10%:</td>
<td>Semantic Analysis (Name Analyzer)</td>
</tr>
<tr>
<td>10%:</td>
<td>Semantic Analysis (Type Analyzer)</td>
</tr>
<tr>
<td>10%:</td>
<td>Code Generation</td>
</tr>
<tr>
<td>10%:</td>
<td>Optimization (Bonus: +5%)</td>
</tr>
</tbody>
</table>

- Compiler Optimization Race:
  - We measure the size of the generated code of your compilers
  - Winner gets an extra 5% bonus!
Course Work

5%: Attendance & Participation
5%: Interpreter for a small language (while language)

Compiler Phases:

<table>
<thead>
<tr>
<th>10%:</th>
<th>Lexical Analysis (Scanner)</th>
</tr>
</thead>
<tbody>
<tr>
<td>10%:</td>
<td>Syntax Analysis (Parser)</td>
</tr>
<tr>
<td>10%:</td>
<td>Semantic Analysis (Name Analyzer)</td>
</tr>
<tr>
<td>10%:</td>
<td>Semantic Analysis (Type Analyzer)</td>
</tr>
<tr>
<td>10%:</td>
<td>Code Generation</td>
</tr>
<tr>
<td>10%:</td>
<td>Optimization (Bonus: +5%)</td>
</tr>
</tbody>
</table>

15%: (Individual) Homeworks (∼5)

- Compiler Optimization Race:
  - We measure the size of the generated code of your compilers
  - Winner gets an extra 5% bonus!
Course Work

5%: Attendance & Participation
5%: Interpreter for a small language (while language)

Compiler Phases:

| 10%: | Lexical Analysis (Scanner)  
| 10%: | Syntax Analysis (Parser)  
| 10%: | Semantic Analysis (Name Analyzer)  
| 10%: | Semantic Analysis (Type Analyzer)  
| 10%: | Code Generation  
| 10%: | Optimization (Bonus: +5%)  

15%: (Individual) Homeworks (∼ 5)
15%: Final Exam

- Compiler Optimization Race:
  - We measure the size of the generated code of your compilers
  - Winner gets an extra 5% bonus!
Course Work

5%: Attendance & Participation
5%: Interpreter for a small language (while language)

Compiler Phases:

10%: Lexical Analysis (Scanner)
10%: Syntax Analysis (Parser)
10%: Semantic Analysis (Name Analyzer)
10%: Semantic Analysis (Type Analyzer)
10%: Code Generation
10%: Optimization (Bonus: +5%)

15%: (Individual) Homeworks (∼ 5)
15%: Final Exam

- Compiler Optimization Race:
  - We measure the size of the generated code of your compilers
  - Winner gets an extra 5% bonus!
Pair Programming

- Seven programming assignments (1 interpreter, 6 phases of compiler)
- Implementation language: Java
- Groups of 2 students
  - Same group for entire class
  - Same grade for members of group (typically)
- Form groups by the end of this week, email me your group members
- Contact me if you are having trouble finding a group
- ~2 weeks for each project
- Workload depends on planning well with your group-mate: Start early!
Challenges

- Is it hard to implement a compiler?

[Quora search: Why are compilers so hard to write?]
Challenges

- Is it hard to implement a compiler?
- No. Implementing a **correct** and **efficient** compiler is tough.
Visual C++ compiler bug with optimizations enabled; loop condition incorrectly optimized away - by wtbw
“Every compiler we tested was found to crash and also to silently generate wrong code when presented with valid input.”
Several interesting results on correct compilers

- (see proceedings of PLDI and POPL conferences)
Course Rhythm

- Textbook reading
- Lectures: 2 – 4 class periods
- Homework
- Recitation (review homework): 1 class period
- Programming assignment

- Final Exam
Course Staff

- **Instructor:** Hossein Hojjat (https://www.cs.rit.edu/~hh/)
  - University of Tehran
    - (Bs. Software Engineering 2001 - 2005)
  - University of Tehran & TU Eindhoven
    - (Msc. Software Engineering 2005 - 2007)
  - EPFL Lausanne, Switzerland
    - (PhD Computer Science 2008 - 2013)
  - Cornell University
    - (Postdoctoral Researcher 2014 - 2016)

- **Email:** hh@cs.rit.edu
- **Office:** GOL(70)-3545
- **Class Hours:** MWF 9:00AM - 9:50AM
- **Office Hours:** Tu 11am - 12am, Th 11am - 12am
  - Send email for alternative time

- **Webpage:**
  - https://mycourses.rit.edu/
  - https://cs.rit.edu/~hh/teaching/doku.php?id=cc17:top
Tell us about your background, and why do you need to learn about compilers, and what aspects of a compiler is more interesting to you!
• “Modern Compiler Implementation in Java 2nd Edition” (a.k.a. Tiger book)
  • Andrew W. Appel, Jens Palsberg
Academic Integrity

- Read the academic integrity policy of RIT and the department https://www.cs.rit.edu/SemesterConversion/common.html
- You are allowed to discuss with other groups, however code sharing is strictly forbidden
- If you aren’t sure what is allowed and what isn’t, please ask

PIRACY. IT'S A CRIME.
YOU WOULDN'T STEAL A MOVIE
YOU WOULDN'T STEAL A COMPILER!
Feedback

- Do not hesitate to give constructive feedback at anytime
- Whatever you feel to make this course better
- Come to office hours, drop me an email if you miss office hour
- Speak up openly, just like when you comment in reddit!