CSCI 742 - Compiler Construction

Lecture 1

Course Overview

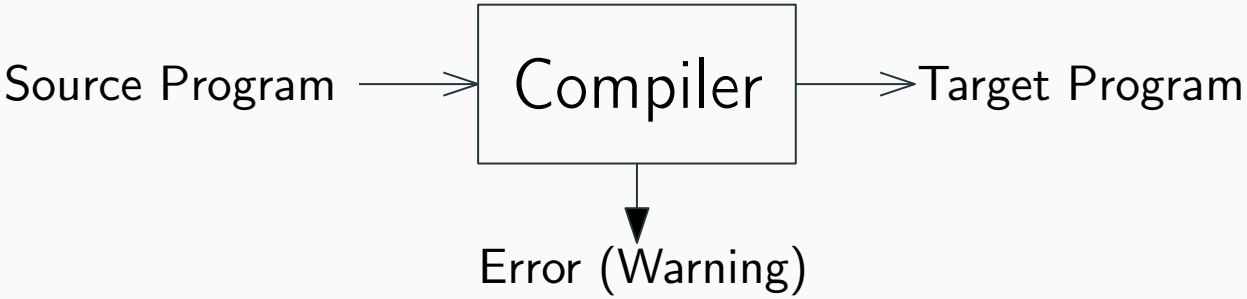
Instructor: Hossein Hojjat

January 17, 2018

What is a Compiler?

Compiler

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



- 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)

Example: Java Compiler

```
⋮
while ( x != y) {
 if (x > y)
   x = x - y;
 else
   y = y - x;
}
System.out.println(x);
⋮
```

javac GCD.java

javap -c GCD

```
7:  iload_1
8:  iload_2
9:  if_icmpeq     31
12: iload_1
13: iload_2
14: if_icmple     24
17: iload_1
18: iload_2
19: isub
20: istore_1
21: goto          7
24: iload_2
25: iload_1
26: isub
27: istore_2
28: goto          7
31: getstatic     #2   // System.out
34: iload_1
35: invokevirtual #3   // println
```

- You will implement a compiler for a small language
  - (syntax similar to Java)

Source Code vs. Machine Code

```
while ( x != y) {
 if (x > y)
   x = x - y;
 else
   y = y - x;
}
```

```
7:  iload_1
8:  iload_2
9:  if_icmpeq  31
12: iload_1
13: iload_2
14: if_icmple  24
17: iload_1
18: iload_2
19: isub
20: istore_1
21: goto       7
24: iload_2
25: iload_1
26: isub
27: istore_2
28: goto       7
```

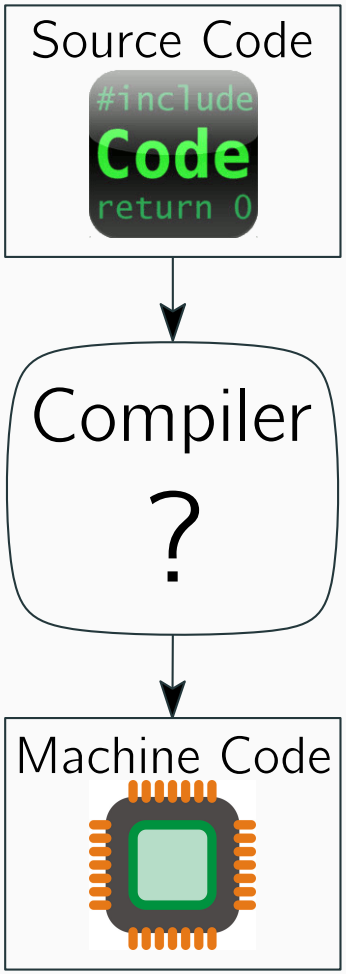
Source Code:

- 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:

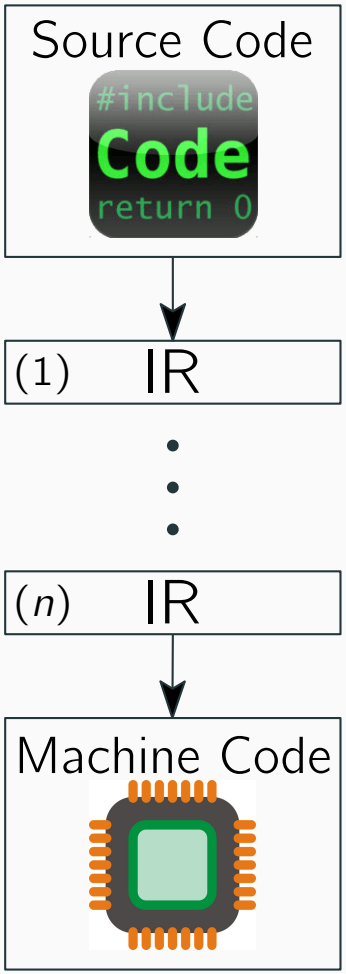
- Optimized for hardware execution
- Basic commands that move bits around in registers and memory
- Redundancy decreased
- Information about source code structure lost

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?

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

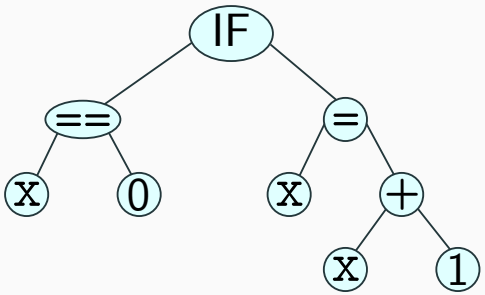
Source Code
(concrete syntax)

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

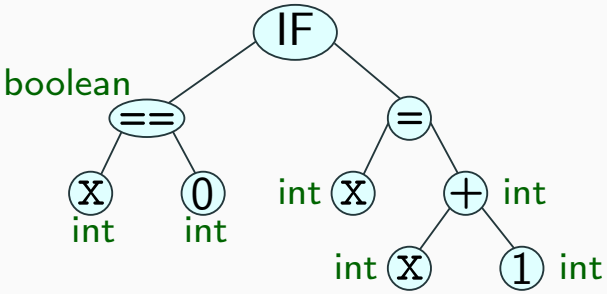
Token Stream

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

Abstract Syntex Tree
(AST)

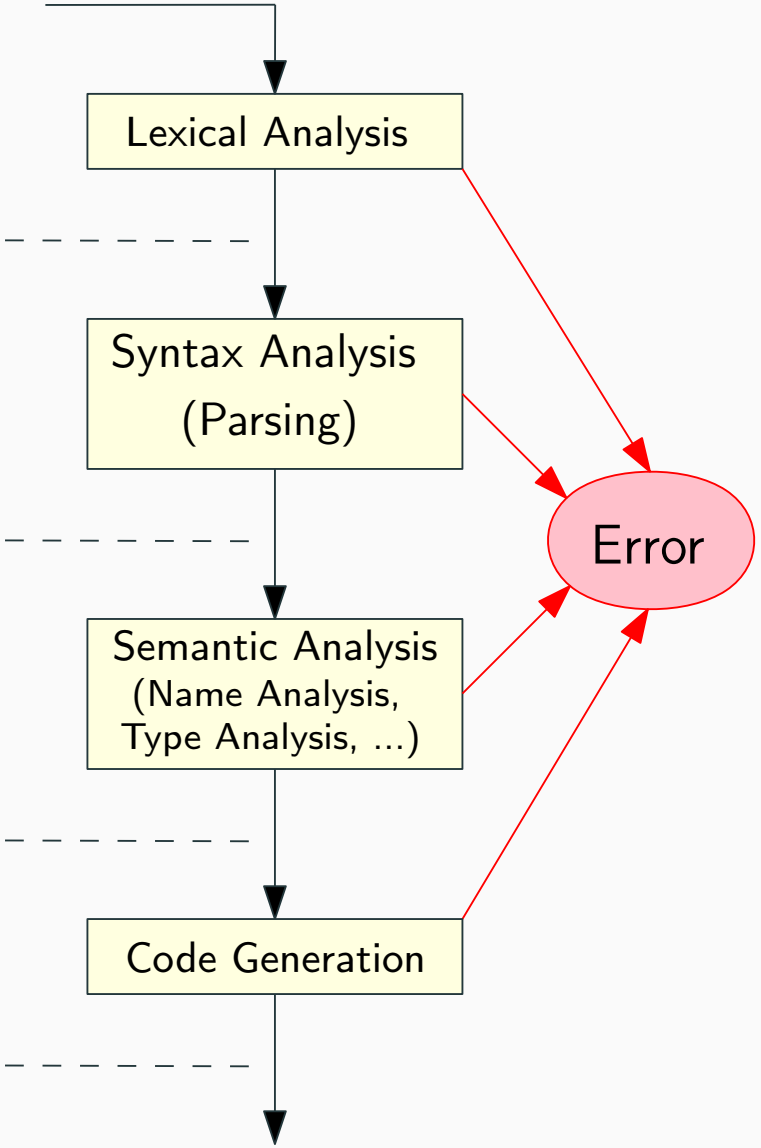


Attributed AST



Machine Code

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



Course Work

Main Project

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

Course Work

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

- **60%** of your final grade is your compiler project

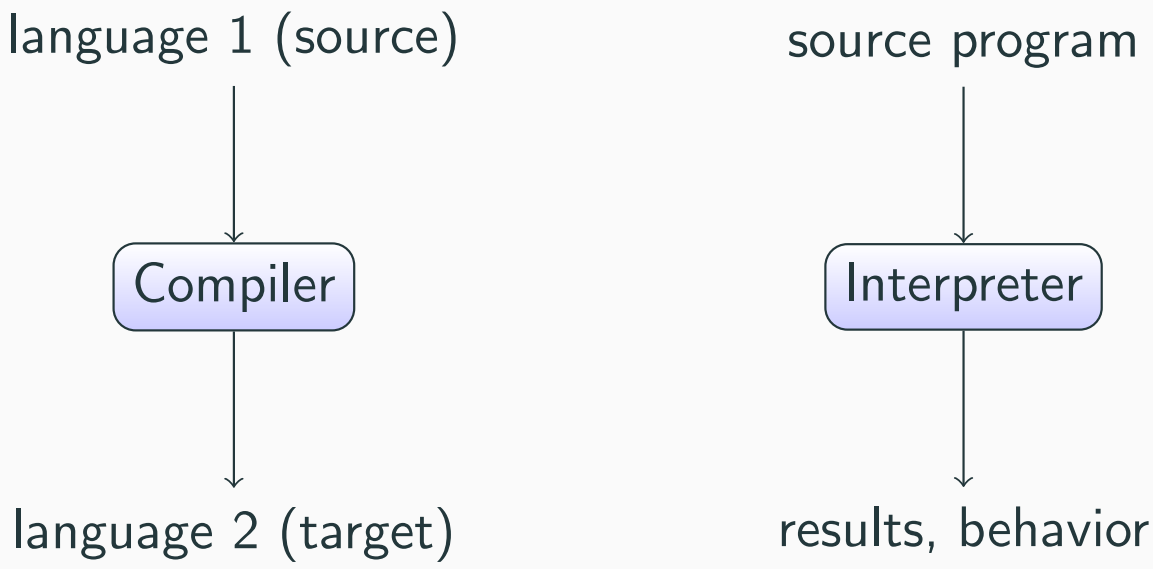
Interpreters vs. Compilers

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



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

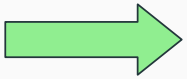
Optimization

- 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

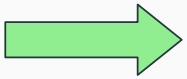
```
a = 7;
b = 2;
...
x = a - b;
while(x < 10){
...
}
```



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

Constant Folding

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



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

Course Work

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

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

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

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

10%: Midterm Exam

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

10%: Midterm Exam

20%: Final Exam

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

Pair Programming

10%: Midterm Exam

20%: Final Exam

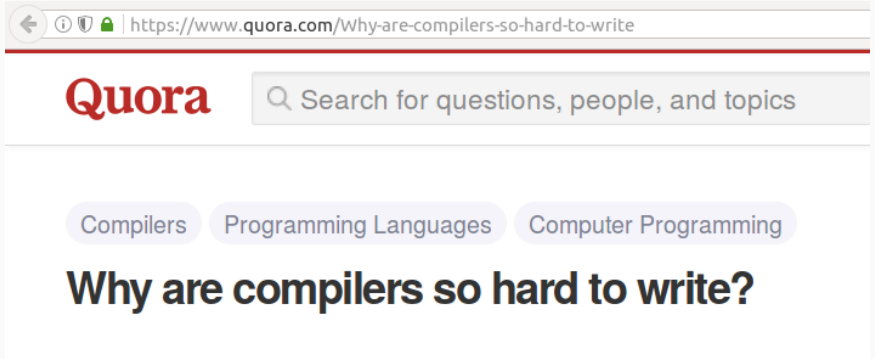
Pair Programming

- Seven programming assignments (1 interpreter, 6 phases of compiler)
- Implementation language: Java
  - Possibility of using another language like C++ if you are more productive with it
- 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
- Workload depends on planning well with your group-mate:

Start early!

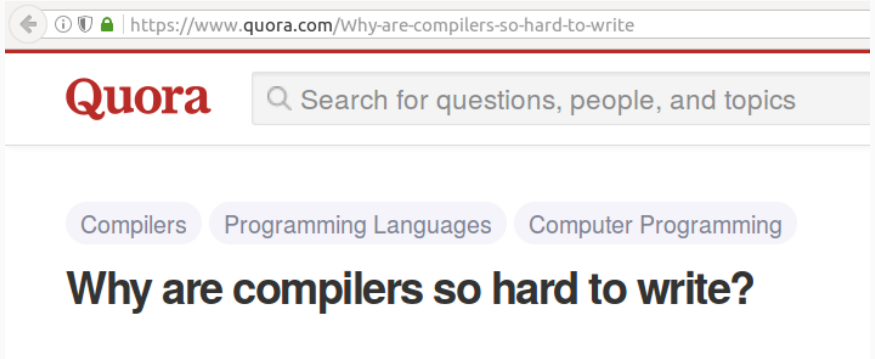
Challenges

- Is it hard to implement a compiler?



Challenges

- Is it hard to implement a compiler?



- No. Implementing a **correct** and **efficient** compiler is tough

Compiler Bugs

Visual Studio

Home Dashboard Directory Help

VISUAL STUDIO AND .NET FRAMEWORK HOME

Feedback Surveys

Visual C++ compiler bug with optimizations enabled; loop condition incorrectly optimized away - by wtbw

Compiler Bugs

Finding and Understanding Bugs in C Compilers

Xuejun Yang Yang Chen Eric Eide John Regehr
University of Utah, School of Computing
{jxyang, chenyang, eeide, regehr}@cs.utah.edu

[PLDI'11]

"Every compiler we tested was found to crash and also to silently generate wrong code when presented with valid input."

Verified Compilers

Automatically Proving the Correctness of Compiler Optimizations

Sorin Lerner Todd Millstein Craig Chambers
Department of Computer Science and Engineering
University of Washington
{lerns,todd,chambers}@cs.washington.edu

[PLDI'03]

Formal Certification of a Compiler Back-end

***or*: Programming a Compiler with a Proof Assistant**

Xavier Leroy
INRIA Rocquencourt
Xavier.Leroy@inria.fr

[POPL'06]

- Several interesting results on correct compilers
  - (see proceedings of PLDI and POPL conferences)

Course Staff

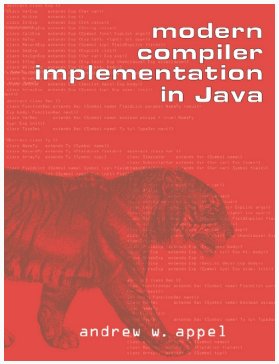
- **Instructor:** Hossein Hojjat (`https://www.cs.rit.edu/~hh/`)
  - University of Tehran
    (Bs. Software Engineering 2001 - 2005)
  - University of Tehran & TU Eindhoven
    (Msc. Software Engineering 2005 - 2007)
  - EPFL Lausanne, Switzerland
    (PhD Computer Science 2008 - 2013)
  - Cornell University
    (Postdoctoral Researcher 2014 - 2016)
- **Email:** hh@cs.rit.edu
- **Office:** GOL(70)-3545
- **Class Hours:** MWF 9:05 AM - 10:00 AM
- **Office Hours:** Tu 11am - 12am, Th 11am - 12am
  - Send email for alternative time
- **Webpage:**

- `https://mycourses.rit.edu/`
- `https://cs.rit.edu/~hh/teaching/cc18/`

Icebreaker

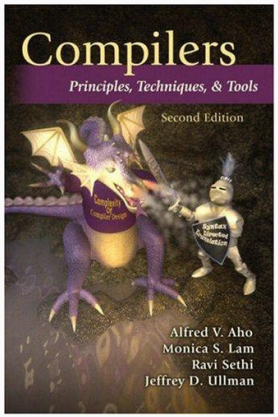
Tell us about your background,
and why do you need to learn about compilers,
and what aspects of a compiler is more interesting to you!

Textbook



- "Modern Compiler Implementation in Java (2nd Edition)" (a.k.a. Tiger Book)
  - Andrew Appel, Jens Palsberg

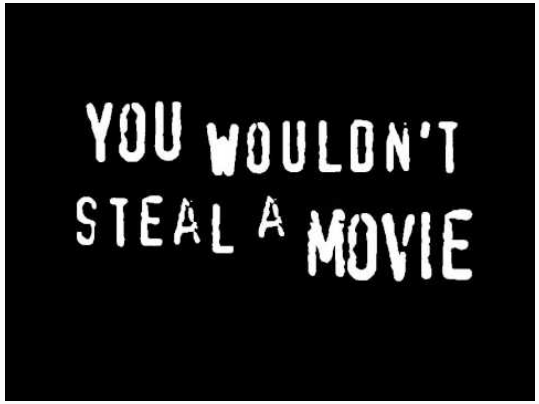
Optional:

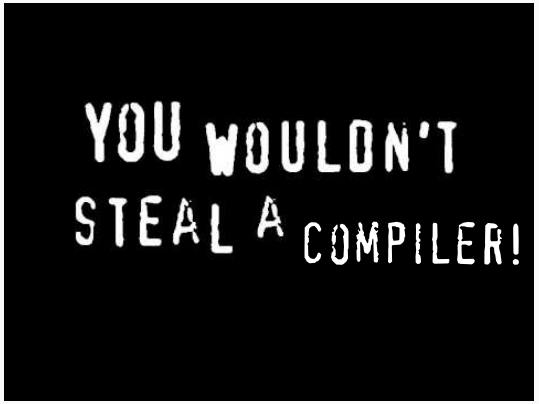


- "Compilers: Principles, Techniques, and Tools (2nd Edition) " (a.k.a. Dragon Book)
  - Alfred Aho, Monica Lam, Ravi Sethi, Jeffrey Ullman

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





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`!

[–] **22SAS** MS CS '18 - Samurai Jack!! 1 point 1 month ago

I am taking compilers under him next semester. I just hope he is not an awful teacher.

permalink **embed** **parent**

[–] **nedolya** CS MS/BS 2018 1 point 1 month ago

I pushed off CC to the next spring after this one in hopes of getting Fluet... I might drop you a line to see how Hojjat was though sometime next semester

permalink **embed** **parent**