


CS406: Compilers

Spring 2022

Week1: Overview, Structure of a compiler

Why Study Compilers?




Company Description: We at MathWorks believe in the importance of human knowledge and profoundly improve our standard of living. We do their best work. Because of the breadth of work we do, compilers in the middle, and back. If you love geeking out with SIMD intrinsics, parallel analyses, designing elegant intermediate representations, playing with partially evaluating expressions, reasoning about parallel program systems, we've got a job for you.

Company Contact: Dale Martin -martind@mathworks.com

Job Title: Compiler Engineer LLVM

Job Description: Our group is responsible for the core technology



Company Description: At IBM, work is more than a job – it's a calling: To build along with clients and sell. To make markets. To invent. To collaborate. To attempt things you've never thought possible. Are you ready to lead in this the world's most challenging problems? If so, let's talk. IBM compilers technology targeting a variety of hardware and software; including AIX, IBM Linux on IBM Z. We produce compilers for a range of source language Python, Node.js) optimized for IBM Power and IBM Z.

Company Contact: Dickson Chau - dickson.chau@ca.ibm.com

Job Title: Intermediate C/C++ & Fortran Compiler Developer


Job Description: The IBM C/C++ & Fortran Compiler Group is looking for experienced

Company Contact: David Finkelstein - dxf@google.com


Job Title: Chrome OS Toolchain Engineer

Job Description: Our team delivers production quality tool debugging tools) for C, C++, Rust, and Go in Chrome OS. Our to boost the developers' experience and enhance Chrome OS mentoring new team members.

Responsibilities:



Very Very Exciting Jobs!



Company Contact: Ted Kremen

Job Title: Compiler and/or Debugger

Job Description: The LLDB debugger, Clang and the LLDB expression evaluator team, most of your work will be in the LLVM and Swift community. LLDB is a software stack and externally by many other projects.


Job Title: Linker Engineer

Job Description: The dyld team is responsible for Apple platforms (dyld, Id64, cc tools). As a linker, you will link and launch efficiently — an important part of Apple ships. You will also collaborate with the software's CPU and memory efficiency across all Apple platforms.

Job Title: Performance Compiler Engineer

Job Description: The CPU and Accelerator compiler performance and optimization team works on CPUs and Accelerators on all Apple platforms. You will develop industry-impacting technology that enhances performance, battery life, compile-time and the LLVM open source project and get it into the hands of our customers.

Job Title: C++ Compiler Engineer



Company Description: NVIDIA is like no place you've ever worked before. Self-driving cars to blockbuster movies. And a growing list of other exciting projects.

Company Contact: Linda Lim - lilim@nvidia.com

Job Title: Engineering Manager - Deep Learning Compiler


Job Description: In this role, you will be managing a team of engineers who will learn networks and develop compiler optimization algorithms. You will communicate and establish and evolve policies and procedures for deep learning software framework teams and the hardware and software engineering work.

Job Title: Senior Backend Compiler Engineer

Job Description: What you will be doing:

- Guide the design and implementation of a new LLVM backend
- Design and develop new compiler passes and optimization passes
- Work with global compiler, hardware and application teams
- Apply and adapt the latest compiler technologies to production
- Get a chance to influence and work on new GPU architectures

Job Title: Senior Compiler Engineer



Company Description: Qualcomm is the world's leading mobile device manufacturer. We deliver breakthrough technologies that transform the mobile experience.

Company Contact: Taylor Simpson - tsimpson@quicinc.com

Job Title: LLVM Compiler Engineer

Job Description: The Compiler Team at Qualcomm Innovation Center works on LLVM for Qualcomm ARM and other LLVM teams at Qualcomm, as well as the general LLVM community.

Job Title: LLVM Compiler Developer, Senior

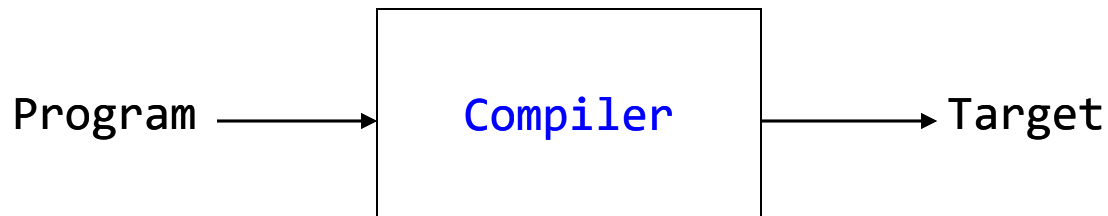
Job Description: Interested in enabling millions of users to have beautiful moments look fantastic? Come join our team!

- Few disciplines with **deep theory + practice**

"..Theory and practice are two sides of the same coin.." - Jeff Ullman, *ACM Turing Award lecture*.

Intro to Compilers

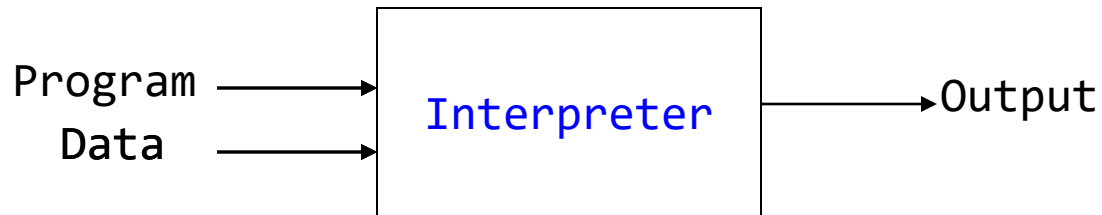
- One way to implement *programming languages*
 - Programming languages are notations for specifying computations to machines

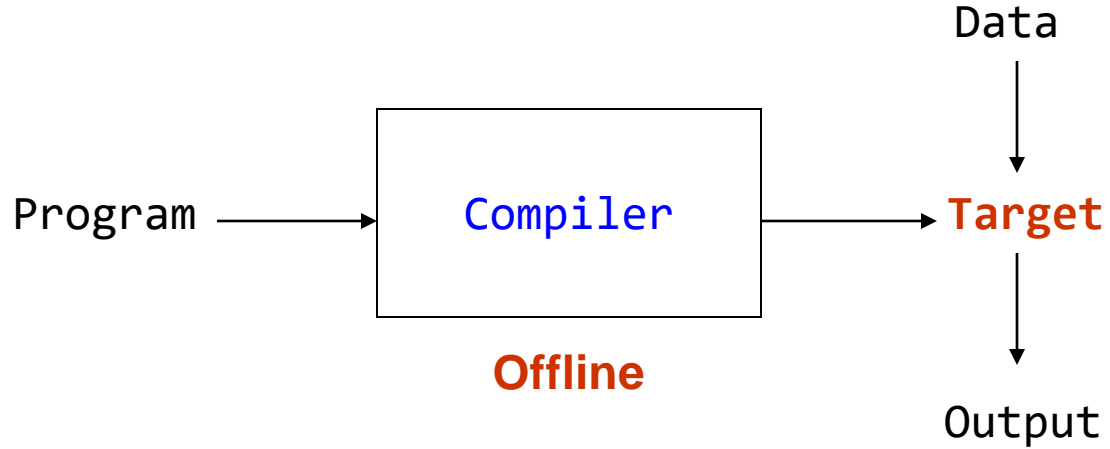


- *Target* can be an assembly code, executable, another source program etc.

Intro to Compilers

- Alternate way to implement programming languages



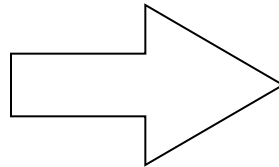


these are the two types of language processing systems

What is a Compiler?

Traditionally: Program that analyzes and **translates** from a high-level language (e.g. C++) to low-level assembly language that can be executed by the hardware

```
int a, b;  
a = 3;  
if (a < 4) {  
    b = 2;  
} else {  
    b = 3;  
}
```



```
var a  
var b  
mov 3 a  
mov 4 r1  
cmpi a r1  
jge l_e  
mov 2 b  
jmp l_d  
l_e: mov 3 b  
l_d: ;done
```


Compilers are *translators*

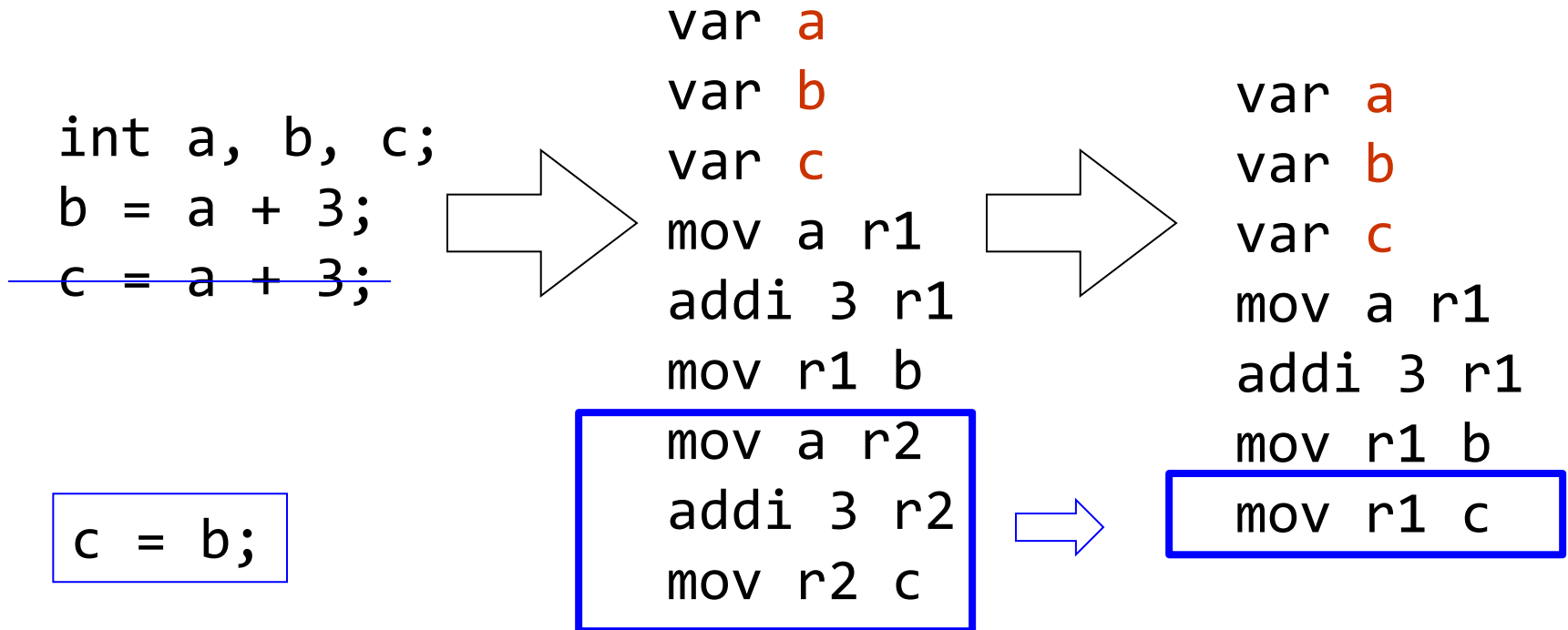
- Fortran
- C
- C++
- Java
- Text processing language
- HTML/XML
- Command & Scripting Languages
- Natural Language
- Domain Specific Language

translate


- Machine code
- Virtual machine code
- Transformed source code
- Augmented source code
- Low-level commands
- Semantic components
- Another language

Compilers are *optimizers*

- Can perform optimizations to make a program more efficient



Compilers as Translators

1. High level language \implies assembly language (e.g. gcc)
2. High level language \implies machine independent bytecode (e.g. javac)
3. Bytecode \implies native machine code (e.g. java's JIT compiler)
4. High level language \implies High level language
(e.g. domain-specific languages, many research languages)

How would you categorize a compiler that handles SQL queries?

HLL to Assembly



- Compiler converts program to assembly
- Assembler is machine-specific translator which converts assembly to machine code

`add $7 $8 $9 ($7 = $8 + $9) => 000000 00111 01000 01001 00000 100000`

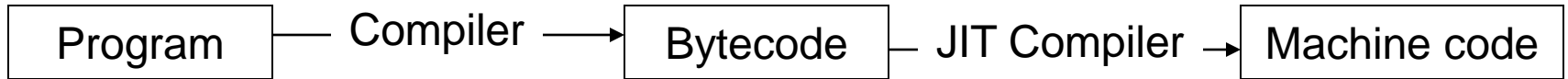
- Conversion is usually one-to-one with some exceptions
 - Program locations
 - Variable names

HLL to Bytecode



- Compiler converts program into machine independent bytecode
 - e.g. javac generates Java bytecode, C# compiler generates CIL
- Interpreter then executes bytecode “on-the-fly”
- Bytecode instructions are “executed” by invoking methods of the interpreter, rather than directly executing on the machine
- Aside: what are the pros and cons of this approach?

HLL to Bytecode to Assembly



- Compiler converts program into machine independent bytecode
 - e.g. javac generates Java bytecode, C# compiler generates CIL
- Just-in-time compiler compiles code *while program executes* to produce machine code
 - Is this better or worse than a compiler which generates machine code directly from the program?

Why do we need compilers?

- Compilers provide *portability*
- Old days: whenever a new machine was built, programs had to be rewritten to support new instruction sets
- IBM System/360 (1964): Common Instruction Set Architecture (ISA) --- programs could be run on any machine which supported ISA
 - Common ISA is a huge deal (note continued existence of x86)
- But still a problem: when new ISA is introduced (EPIC) or new extensions added (x86-64), programs would have to be rewritten
- Compilers bridge this gap: write new compiler for an ISA, and then simply recompile programs!

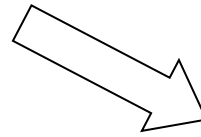
Why do we need compilers?

- Compilers enable **high-performance and productivity**
- Old: programmers wrote in assembly language, architectures were simple (no pipelines, caches, etc.)
 - Close match between programs and machines --- easier to achieve performance
- New: programmers write in high level languages (Ruby, Python), architectures are complex (superscalar, out-of-order execution, multicore)
- Compilers are needed to bridge this ***semantic gap***
 - Compilers let programmers write in high level languages and still get good performance on complex architectures

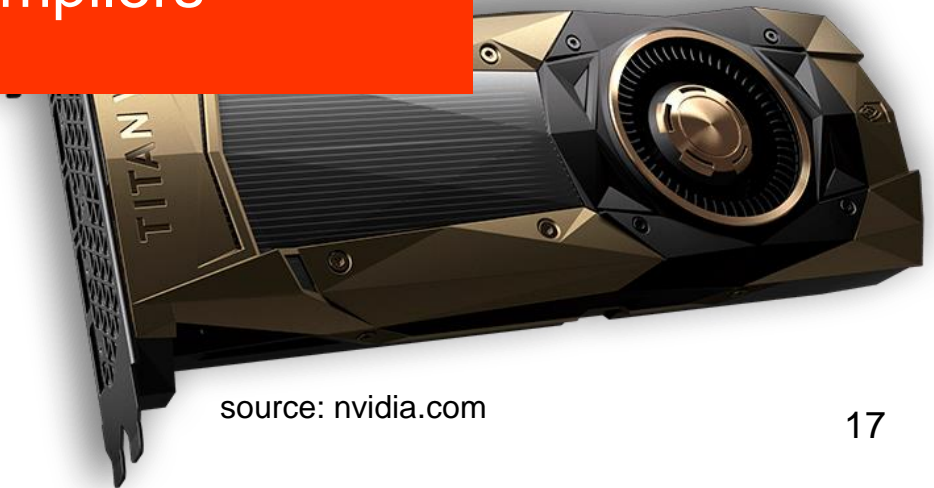
Semantic Gap

- Python code that actually runs on GPU

```
import pycuda
import pycuda.autoinit from pycuda.tools import
make_default_context
c = make_default_context()
d = c.get_device()
```



Impossible without Compilers



source: nvidia.com

History

- 1954: IBM 704
 - Huge success
 - Could do complex math
 - Software cost > Hardware cost

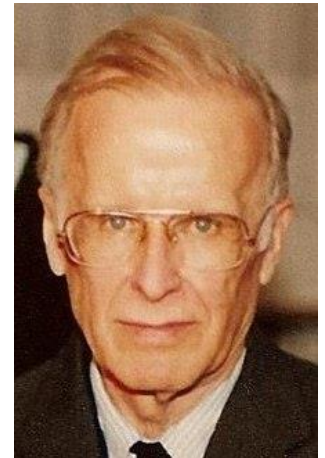


Source: IBM Italy,
<https://commons.wikimedia.org/w/index.php?curid=48929471>

How can we improve the efficiency of creating software?

History

- 1953: Speedcoding
 - *High-level programming language* by John Backus
 - Early form of *interpreters*
 - Greatly reduced programming effort
- About 10x-20x slower
- Consumed lot of memory (~300 bytes = about 30% RAM)

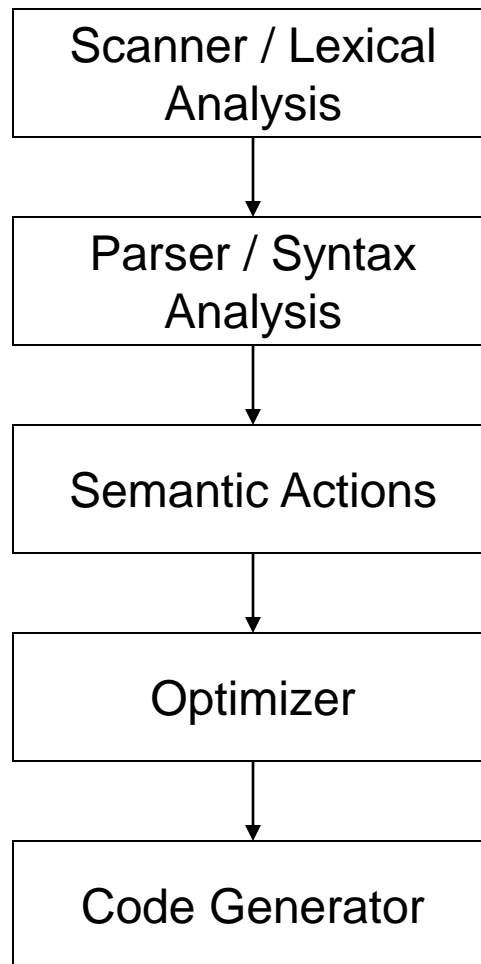


Fortran I

- 1957: Fortran released
 - Building the compiler took 3 years
 - Very successful: by 1958, 50% of all software created was written in Fortran
- Influenced the design of:
 - high-level programming languages e.g. BASIC
 - practical compilers

Today's compilers still preserve the structure of Fortran I

Structure of a Compiler



Scanner

- Analogy: Humans processing English text

Rama is a neighbor.

Ra mais an eigh bor.

You have to do some work to align the spaces and understand the sentence.

Scanner

- Consider the program text

```
if ( a < 4) {  
    b = 5  
}
```

– Has tokens that are:

1. keywords – if
2. Punctuation marks – (,), {, }, blankspaces, tab space (\t), newlines (\n)
3. Identifiers – a, b
4. Constants/Literals – 4, 5
5. Operators - <, =

Scanner

- A compiler starts by seeing only program text

```
if ( a < 4) {  
    b = 5  
}
```

- as a series of letters

```
'i' 'f' ' ' '(', 'a' '<' '4' ')'  
  ' ' '{' '\n' '\t' 'b' '=' '5'  
        '\n' '}'
```


Scanner

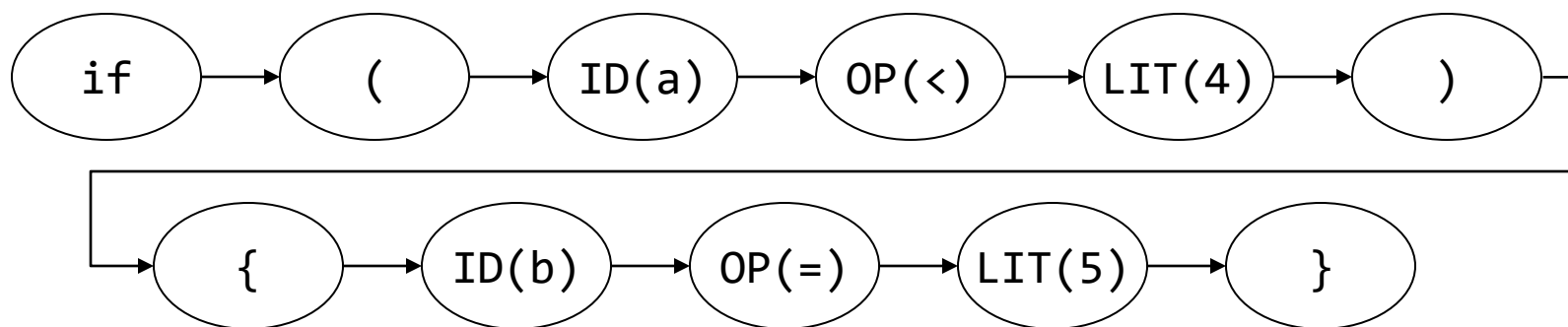
- A compiler starts by seeing only program text
- Scanner converts program text into string of *tokens*

```
'i' 'f' ' ' '(' 'a' '<' '4' ')'
  ' ' '{' '\n' '\t' 'b' '=' '5'
        '\n' '}'
```

- Analogy: Humans processing English text
 - recognize words in Rama is a neighbor.
 - Rama, is, a, neighbor
 - Additional details such as punctuations(.), capitalizations (R), blank spaces.

Scanner - Summary

- A compiler starts by seeing only program text
- Scanner converts program text into string of *tokens*



- But we still don't know what the *syntactic structure* of the program is

Exercise

Convert the following program text into tokens:

`c = a + b * 60`

Suggested Reading

- Alfred V. Aho, Monica S. Lam, Ravi Sethi and Jeffrey D. Ullman: Compilers: Principles, Techniques, and Tools, 2/E, AddisonWesley 2007
 - Chapter 1 (Sections: 1.1 to 1.3, 1.5)
- Fisher and LeBlanc: Crafting a Compiler with C
 - Chapter 1 (Sections 1.1 to 1.3, 1.5)