EECS 368
Programming Language Paradigms

Dr. Andy Gill

University of Kansas

August 27, 2015

Class Information

How to find me

Office:	2024 Eaton Hall / 252 Nichols Hall
Web:	`http://people.eecs.ku.edu/~andygill`
Phone:	4-8817 / 4-4712
Email:	`andygill@ku.edu`
Office Hours:	2:00-4:00 Wednesdays in Eaton Hall, or by appointment.

About the Class

Time:	4:00-5:15 TR
Class Web:	`https://piazza.com/ku/fall2015/eecs368/home`
Prerequisites:	EECS 268 is a hard prerequisite for this course.

Back in the 50's

Machine Codes

Programing computer by literally giving the codes to perform operations. Examples are

- moving data (0x37)
- adding data (0x17)
- comparing data (0x28)
- storing data to tape (...)

This was interacting with the machine on its terms, 1s and 0s.

Using Software Abstractions

Assembly Language

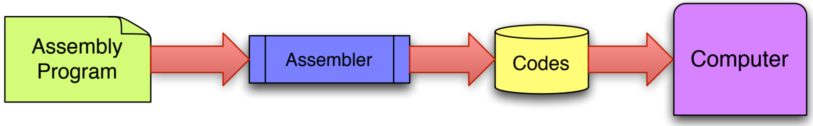
Programing computer by using mnemonics instead of numerical code. Examples are

- moving data (ld r1,r2)
- adding data (add r1,r2,r3)
- storing data to tape (cmp r1,r2)
- goto to another set of instructions (goto label_44)

This is slightly better.

- A transliteration that is easier for humans to understand/remember.
- Still a one-to-one mapping to machine code.

What an Assembler Does



Using Hardware Abstractions

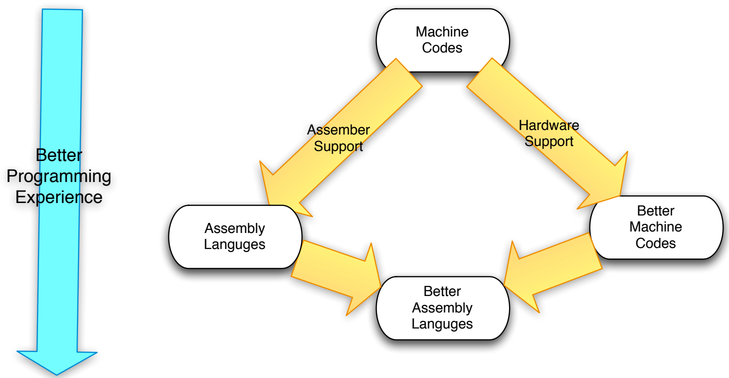
Better Codes

At the same time, the codes provided inside machines became more powerful

- codes for subroutines
- codes for multiplication, division, floating point.
- ...

Can do the same operation faster and better.

Narrative of Early Computer Languages



A Short and Edited History of Computer Languages

((Whiteboard))

Other Languages

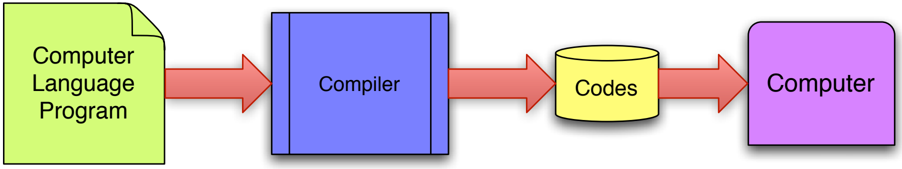
There are many other idioms and ideas in programming languages.

- Different computational models: Logic languages, SQL, XSLT
- Scripting languages: TCL, Perl, Python
- Web services language: PHP, JavaScript

In this class, we are going to focus on the two main computational models, imperative and functional.

Tools for Understanding Languages

- Syntax - precisely describing what a language is
- Semantics - precisely describing what a language does
    - Static Semantics - what happens at compile time
    - Dynamic Semantics - what happens at runtime



Syntax for Languages

C++

```
#include <iostream.h>
main()
{
    cout << "Hello World!";
    return 0;
}
```

Java

```
public class HelloWorld {
    public static void main (String[] args) {
         System.out.println ("Hello World!");
    }
}
```

JavaScript

```
console.log("Hello, World");
```

Scheme

```
(display "Hello World!")
(newline)
```

Haskell

```
main :: IO ()
main = putStrLn "Hello World!"
```

Example Syntax Specification

```
expression ::= ( expression )
            |   expression operation expression
            |   number

operation ::= + | - | * | /
number    ::= digit
            | digit number

digit     ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
```

- `expression`, `operation`, `number` and `digit` are non-terminals.
- +, −, 0, 1 are terminals.

Example of Ambiguity

```
1 + 2 * 3
```

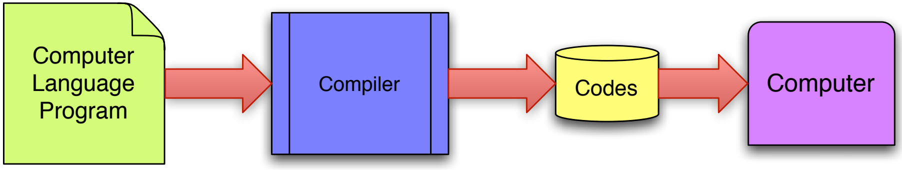
```
(1 + 2) * 3
```

```
1 + (2 * 3)
```

We can pin down exactly what we accept and do not accept for a specific computer language.

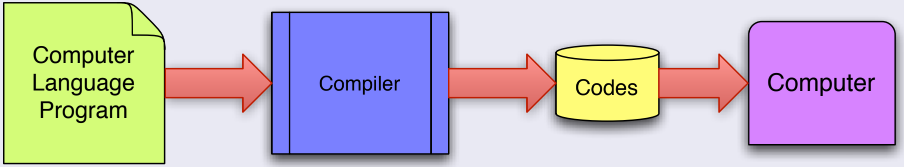
Tools for Understanding Languages (recap)

- Syntax - precisely describing what a language is
- Semantics - precisely describing what a language does
    - Static Semantics - what happens at compile time
    - Dynamic Semantics - what happens at runtime

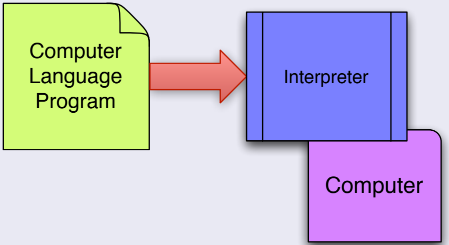


Compiling and Executing Languages

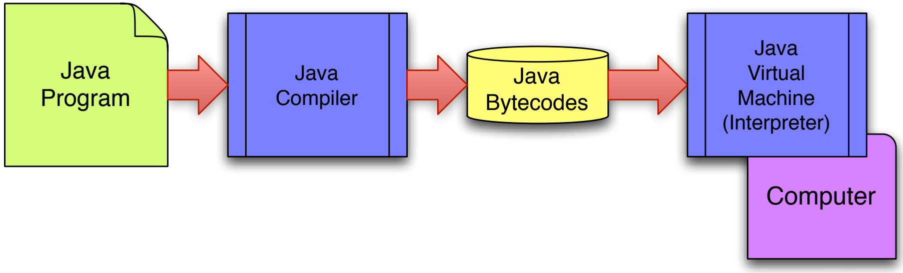
Compiler



Interpreter



Java Compilation and Execution Model



```
$ javac HelloWorld.java
$ ls
-rw-r--r--  1 andy  staff     426 Aug 22 09:29 HelloWorld.class
-rw-r--r--  1 andy  staff     126 Aug 22 09:26 HelloWorld.java
$ java HelloWorld
Hello World!
$
```

What This Class is About

- Having the tools to be precise about a computer language
- Gaining an understanding and historical context for the major components and tradeoffs
- We are going to do this by looking at three languages is some detail
    - JavaScript
    - Scheme
    - Haskell
- We will look at things like
    - Syntactical choices
    - Program idioms (examples, examples, examples)
    - Execution model

Summary of class so far

Machine Codes to Modern Languages

Improving levels of abstraction over time.

- The ability to write using mnemonics, not binary codes
- The ability to write formulas
- The ability to structure code in terms of objects
- The ability to create objects, an have them automatically disposed of after you have finished with them