Sept. 5, 2024

* The PL Landscape
* Types of PLs
* Language Implementation
* Compilation Process

Cacophony of Programming Languages

Major Classes/Types of PLs

1. Imperative: Fortran, COBOL, ALGOL, C, Pascal

   do this
   do that
   do this next
   etc.

2. Functional: LISP, ML, Miranda, Haskell, Eiffel

   a program is a pure function: $f(x) = x^2$

e.g. LISP ①

```
(defun (x)
    (* x x))
```

use: (x 5) → 25

②

```
(defun gcd (a, b)
  (cond ((= a b) a)
        ((> a b) (gcd (- a b) b))
        (t (gcd a (- b a)))))
```

use: (gcd 9 15) → 3

3. <u>OOP - object-oriented Programming</u> : Simula 67, SmallTalk

- package data + operations into a class/object
- inheritance

Also, C evolved into C++, Objective C, C#
and Java

4. <u>Declarative (logic, relational)</u> : Prolog

A program specifies WHAT to compute
not HOW to compute

eg. gcd(X,Y,G) read as gcd of X and Y is G.

```
gcd (X,Y,G) :- X=Y, G=X.
      (if)       (and)
gcd (X,Y,G) :- X > Y, Y1 is Y-X,
               gcd (X,Y1,G).

gcd (X,Y,G) :- Y > X, gcd (Y,X,G).
```

<u>use</u> ?- gcd(9,15,G).

G=3

<u>Other Dichotomies</u>

- Low-level PL vs high-level PL
  (provides access to hardware e.g. addresses)   (more abstract)

- Unstructured vs Structured

e.g. FORTRAN IV (my first PL!)

```
  FUNCTION GCD (INTEGER A, INTEGER B)

5  IF (A - B) 10, 30, 20
10    A = A - B
      GOTO 5
20    B = B - A
      GOTO 5
30    RETURN A
      END
```

$\boxed{\text{spaghetti code}}$

Most modern PLs are structured

e.g. Python

```
def gcd(a, b):
    while a != b:
        if a > b:
            a = a - b
        else:
            b = b - a
    return a
```

- Hacker-Friendly language: C
- Scientific Computing: FORTRAN
- Business Computing: COBOL (very verbose)
- Languages for teaching: PASCAL, BASIC, LOGO

Language Implementation - Overview

- Compilers
- Interpreters
- Intermediate Code

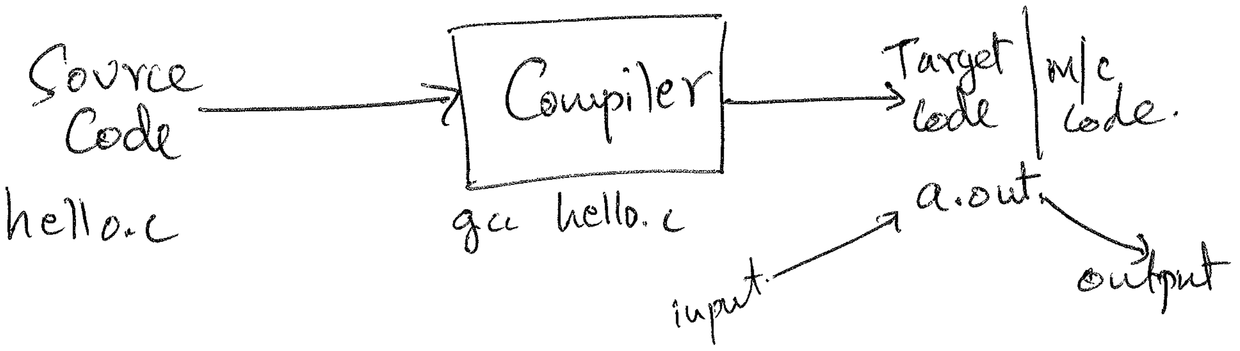
- Compilation process

Compilers

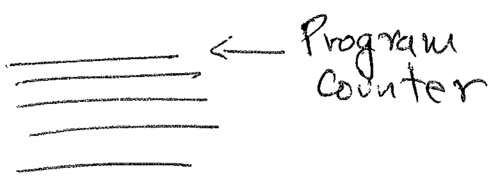
hello.c

```
#include <stdio.h>
int main(){
    println("Hello, world!\n");
    return 0;
}
```

Source code



Target code is executable machine code



$\boxed{\text{CPU}}$

```
do forever
    get next instruction from memory
    decode it
    execute it
```

Interpreters (e.g. Python)

$$\begin{tikzcd}
\text{Source Code} \arrow[r] & \boxed{\text{Interpreter}} \arrow[r] & \text{output} \\
\text{Input} \arrow[ur] & &
\end{tikzcd}$$

Uses a Read-Eval-Print-Loop (REPL)

```
do forever at prompt
    read next statement
    evaluate the statement
    print result of evaluation
```

Compilers	vs	Interpreter
Faster		slower
Little runtime support		more expressive
		more runtime support

Q: What is runtime support?

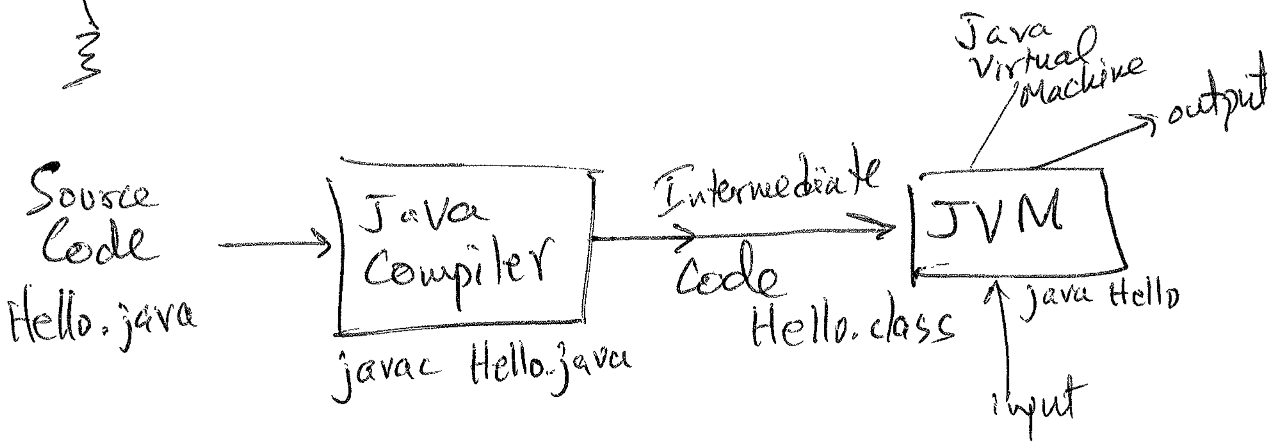
- type checking
- bounds checking
- garbage collection
- error checking + correction
- etc.

Intermediate Code

Java

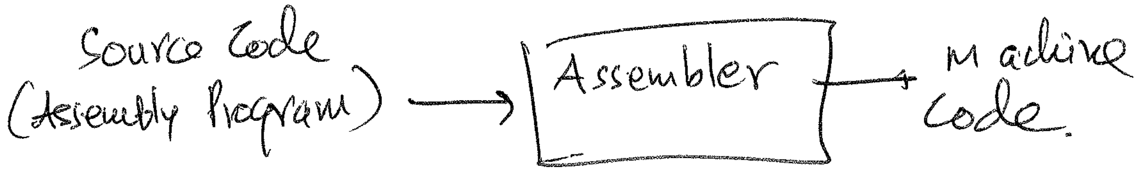
Hello.java

```
public class Hello {
    public static void main(String[] args){
        ~~pub~~ System.out.println("Hello, world!");
    }
}
```

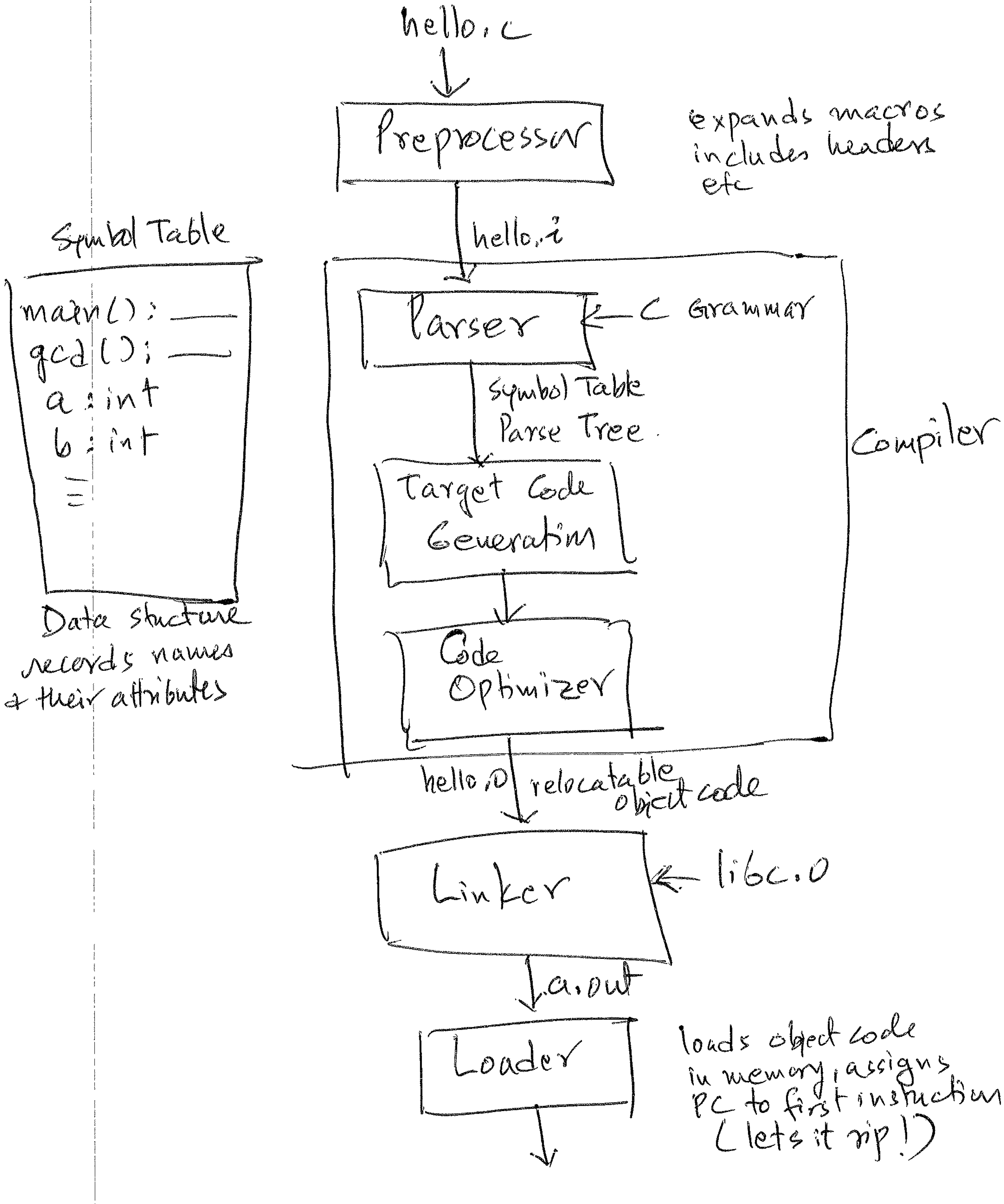


JVM ≡ Just In Time Compiler (JIT)

Assembly



The Compilation Process


hello.c
Preprocessor
expands macros
includes headers
etc
hello.i
Symbol Table
main() : ——
gcd() : ——
a : int
b : int
⋮
Data structure
records names
& their attributes
Parser
C Grammar
Symbol Table
Parse Tree
Compiler
Target Code
Generation
Code
Optimizer
hello.o
relocatable
object code
Linker
libc.o
a.out
Loader
loads object code
in memory, assigns
PC to first instruction
(lets it rip!)