CS344 - Programming Languages

Note Title 1/18/2012

Today

- Syllabus overview
- HW1 - due next Tuesday
- into to the topic
- NO CLASS ON THURSDAY!
  (so plenty of time for your HW)

First Question:

What programming languages have you used?

- Python
- C++
- Java
- Matlab
- Objective C
- C#
- Lua
- C
- Ruby
- Php
- Go

- Lisp
- Javascript

- SQL
- html
- Assembly

Categories

High-level versus low-level:

assembly —<assembler>→ machine code

high-level —<compiler>→ machine or assembly

High-Level Languages

- Began in 1950's with Fortran
- First machine-independant solution
- Slow to become popular, because compiless were not as good as humans

(Not true now - plus, labor costs more than hardware!)

Why so many?

- Evolution: Still very new!
  - Structured programming (using loops instead of go-tos) was only developed in the late 60's.
  - Object orientation was developed in the '80's.
- Personal preference

- <u>Special purposes</u>: Often, the choice depends on what you want to do!
  - C is good for low level systems work
  - Prolog is good for logical relationships among data
  - Awk is good for character & string manipulation
  - Python & perl are good scripting tools

Other issues

- Learning curve
- Ease of use
- Standardization
- Open Source
- Good Compilers
- Economics & patronage
- Inertia

Families of high-level Languages

① Declarative Languages:
- Focus is on what the computer should do
- "higher-level"

② Imperative Languages:
- Focus is on how the computer should do it
- dominant paradigm - often better performance

(Object orientation)

Imperative

Categories:

2

(A) von Neumann: Fortran, C, Ada.
- based on computation with variables

(B) Scripting languages: bash, awk, php, perl, python, Ruby, etc.
- subset of von Neuman, but tailored for ease of expression over speed

(C) Object-oriented: traced from Simula 67.
often related to von Neuman, but object-based

Declarative

Categories & Examples:

(A) Functional languages: Lisp, Scheme, ML, Haskell

- based on recursive definition of functions (inspired by lambda calculus)

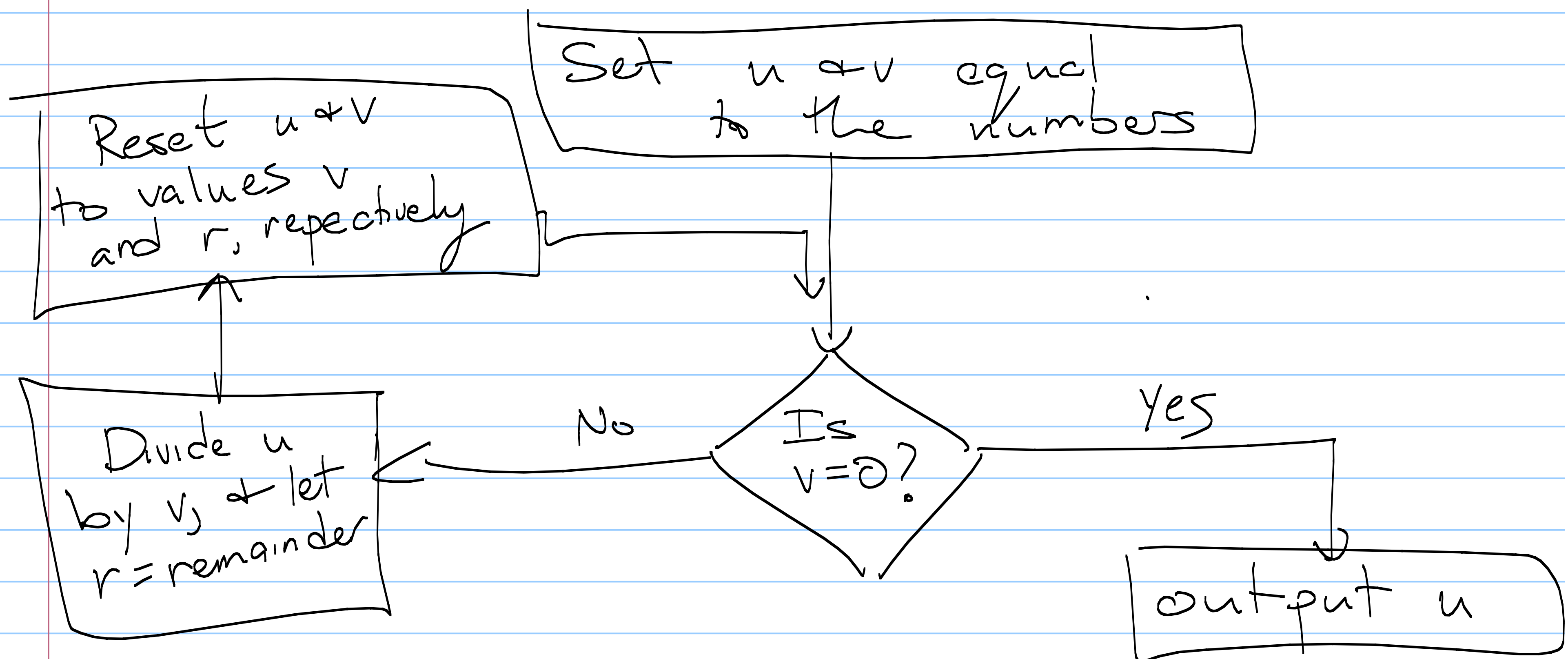
(B) Logic-based: prolog, SQL (?)

- computation is based on attempts to find values that satisfy specified relationships

(C) Data flow: Id, Val

- flow of information (tokens) among nodes

Example: Compute the gcd
(stolen from my 150 lecture)



GCD in a functional language

$$\gcd(a,b) := \begin{cases} a & \text{if } a=b \\ \gcd(b, a-b) & \text{if } a>b \\ \gcd(a, b-a) & \text{if } b>a \end{cases}$$

<u>Claim</u>: This is equivalent to previous algorithm.

GCD in Prolog

gcd(a, b, g) is true if:

- $a = b = g$
- $a > b$ and $\exists c$ such that $c = a - b$ and gcd(c, b, g) is true
- $b > a$ and $\exists$ c s.t. $c = b - a$ and gcd(c, a, g) is true

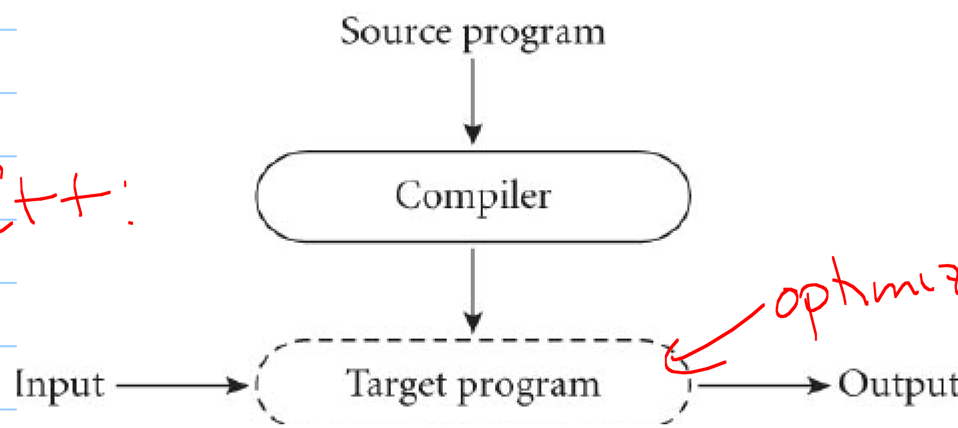
Why study this?!

- Choosing appropriate language is a key step.
- Make learning new languages easier.
- Common terminology for comparison & understanding.
- Understand hidden "features". ex: if (a=b) (in C++)
- Know actual implementation costs.
  if ((①) && (②))

Compliation versus interpretation

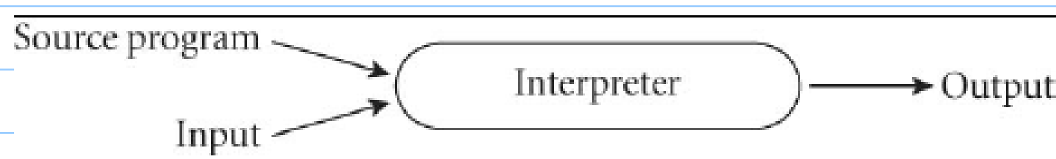
2 models:

→ C/C++:



optimized for my computer

Python:



Pros & Cons

Interpreter:
- greater flexibility
- better debugging
- better with data that is dependant on input

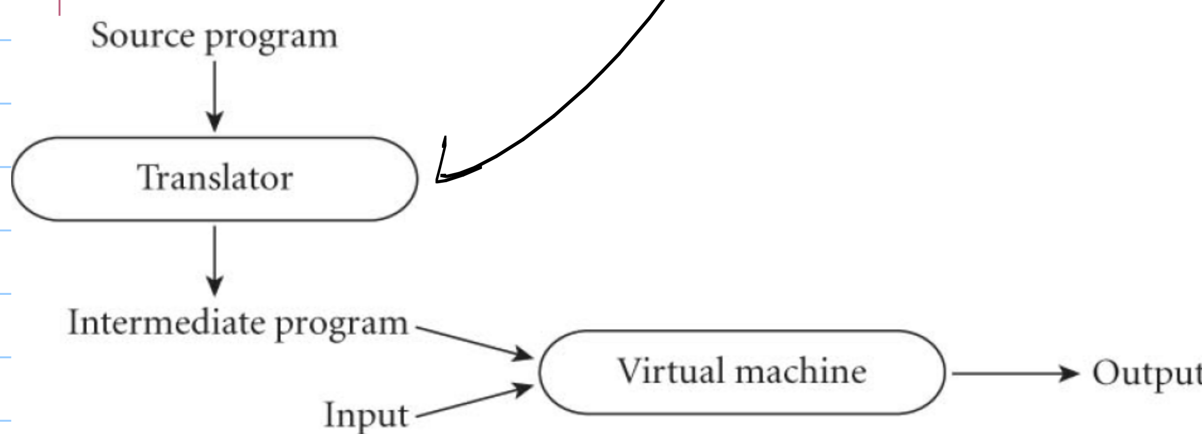
Compilation:
- much faster

Compilation vs. Interpretation

In reality, most languages are **both**.

This is the key.

How much does translator do?



Compilers

The process by which programming languages are turned into assembly or machine code is important in programming languages.

We'll spend some time on these compilers, although it isn't a focus of this class.