CSCI 400 Course Overview

David Grisham

22 August 2017

Introduction

Who am I?

- David Grisham
  - dgrisham@mines.edu
- Master's student under DJ Yang
  - Distributed systems, game theory
- Member of Protocol Labs
  - IPFS, Filecoin

Why am I teaching PL?

- Took PL in 2014
- Previous prof. (Cyndi Rader) retired
- Experience/enjoyment

(also, `alias PL='Programming Languages'`)

Why are you taking PL?

Is it worth an entire semester?

Motivation

- Toolbox
    - Partly choose language based on problem
- Crossover knowledge
    - Haskell → C++
- Research
    - Type systems, memory management, ...

Motivation

When the only tool you have is a hammer, everything looks like a nail." - Abraham Maslow

What are we going to do?

Roughly...

1. Discuss programming language concepts
2. Explore specific languages/paradigms
3. Implement (simple) languages

PL Concepts

- Syntax & features decisions
- Code reuse, polymorphism
- Error handling
- ***Type system***
- Meta-programming

Exploring Languages (Example Criteria)

	Ruby	**Haskell**
Paradigm	Multi, Object-oriented	Functional
Typing	Dynamic	Static
Meta-programming?	Yes	With an extension

Exploring Languages

Ruby

- Learn Ruby
  - While keeping in mind higher-level PL concepts
- Discussion on design/etc.
- Exam

Exploring Languages

Haskell

- Similar to Ruby
  - But no exam
- Implement simple (subsets of) programming languages
  - Better understanding of PL implementation
  - Assignments provided by Mattox Beckman @ UIUC

Criteria for Evalutating PLs

Language Evaluation Criteria

What kind of criteria do you use to evaluate/choose a language?

Categories of programming languages (link)

Example Criteria

1. **Writeability**: How easy is it to write a program?
2. **Readability**: How easy is it to read a program?
3. **Reliability**: Does it include features that help produce more reliable software?
4. **Cost**: What costs are involved?

Categories proposed by Sebesta

(1,2) Write/Read-ability

- Support for abstraction
- Control statements
- Data types
- Syntax
- **Orthogonality**
- **Expressivity**

Orthogonality

- Small set of primitive constructs + ways of combining them
- Every syntactically correct combination is legal
  - C: Arrays, `void` (both have ‘illegal’ cases)
- No side-effects
  - Changing one thing has no effect on another

Orthogonality

> *Orthogonality means that features can be used in any combination, the combinations all make sense, and the meaning of a given feature is* ***consistent*** *regardless of other features with which it is combined.*
> \- ***Michael Scott***

Orthogonality

*Orthogonality means that features can be used in any combination, the combinations all make sense, and the meaning of a given feature is **consistent** regardless of other features with which it is combined.*
- ***Michael Scott***

Sometimes I'll start a sentence and I don't even know where it's going. I just hope I find it along the way.
- ***also Michael Scott***

(3) Reliability

- Type checking
  - Compiler- or run- time
- Exception handling
- Aliasing
  - Multiple references to the same memory
- Read/write-ability
  - Unnatural algorithm implementations are less reliable

(4) Cost

- Learning curve
- Fitting language to problem
- Compiling, executing
- Implementation support (e.g. free compilers)
- Maintaining programs
  - *How does this relate to PL?*

Other Criteria

- Portability
  - Moving between implementations
- Generality
  - Range of applications (tradeoff)
- Well-definedness
  - Completeness/precision of language definition

Language Design

Design Tradeoffs

- **Reliability** vs. **cost of execution**
  - E.g. memory/type safety
- **Readability** vs. **writeability**
  - E.g. APL (link)
- **Flexibility** vs. **reliability**
  - Pointers, types (`NULL`)

Expressivity

Find something to share with the class – turn in for attendance points

- http://babel.ls.fi.upm.es/~jjmoreno/expre.html
- http://redmonk.com/dberkholz/2013/03/25/programming-languages-ranked-by-expressiveness/
- http://stackoverflow.com/questions/638881/what-does-expressive-mean-when-referring-to-programming-language
- http://en.wikipedia.org/wiki/Expressive_power
- http://mt4.radified.com/2009/08/expressive-power-computer-programming-language-literature.html
- http://gafter.blogspot.com/2007/03/on-expressive-power-of-programming.html

Criteria: The Players

- Implementors
  - Difficulty of implementating constructs/features
- Users
  - Care about writeability, eventually readability
- Designers
  - Elegance, accessibility
  - http://www.paulgraham.com/popular.html

Influences on Language Design

- Computer Architecture
- Programming Metodologies

Influences: Computer Architecture

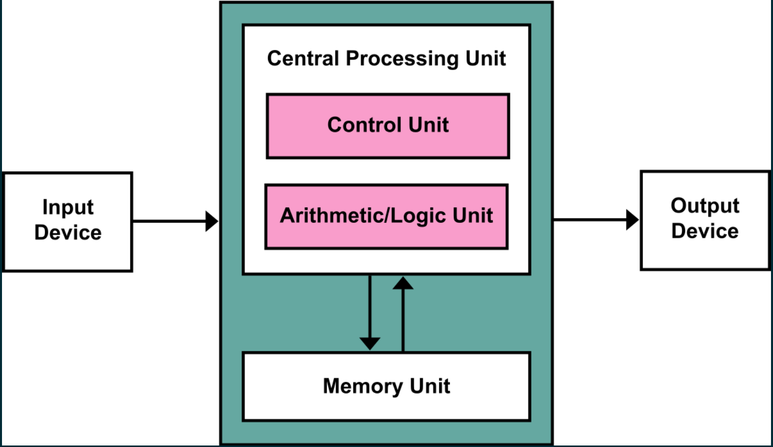

Figure 1: von Neumann Architecture

Influences: Computer Architecture

- Von Neumann architecture
  - Data/programs stored in memory
  - Memory separate from CPU
  - Instructions/data piped from memory to CPU
- Basis for imperative languages
  - **Memory cells → variables**
  - **Piping → assignment**
  - Binary → assembly → C

Influences: Programming Methodologies

Time	**Focus**
50s/60s	Simple applications, computational effiency
60s	People-efficiency: Readability, control structures
70s	Process- to data- oriented
80s	Object oriented
90s-Now	Functional

Language Categories

Category	**Characteristics**	**Examples**
Procedural	Variables, iteration	C, Pascal, Perl
Functional	Functions, composition	Scheme, Haskell
Logic	Rule-based, unordered	Prolog, SQL, K
Object-oriented	Inheritance, late-binding	Java, C++, C#
Markup	Text + formatting/etc.	HTML, XML, Markdown

Conclusion

Links

- Juan Benet: *IPFS: The Distributed Web*
  - https://www.youtube.com/watch?v=HUVmypx9HGI
- Paul Graham: *Being Popular*
  - http://www.paulgraham.com/popular.html
- Brett Victor: *The Future of Programming*
  - https://www.youtube.com/watch?v=8pTEmbeENF4
- John Backus: *Can Programming be Liberated from the von Neumann style?*
  - https://www.cs.ucf.edu/~dcm/Teaching/COT4810-Fall/%202012/Literature/Backus.pdf

Credit

Significant credit to Cyndi Rader for slide content