Introduction

COS 301

School of Computing and Information Science
University of Maine

Fall 2018

Outline

Preliminaries

Me

- Roy M. Turner
- PhD: Georgia Tech
- Research: AI (intelligent agents, robot control, software agents, multiagent systems, computational ecology, computer science education)

The course

- Programming languages
- Design issues/trade-offs
- Types of languages
- Comparison of languages
- Language implementation

Course objectives

- Good understanding of what a programming language is
- Understanding of major language paradigms
- Grasp of issues having to do with syntax and semantics of programs and programming languages
- Knowledge of how control and data types are handled in a variety of languages
- Knowledge of the commonalities and differences between programming languages
- A basis for understanding how to select a programming language for a problem
- Deeper insight into programming languages you already know
- Better professional written communication skills

Syllabus

- Office hours: MW 2-3 (or by appointment), 240 Boardman Hall
- Contacting me: rturner@maine.edu
- TA – Lwam Ghebreggergish
- Online: Course website + Blackboard (grades)
- Homework/project/class participation
- Academic honesty

Homework

Introduction
COS 301
Preliminaries
Why study programming languages?
Programming language paradigms
Programming domains

1. Make sure that you can access the COS 301 website and Blackboard area
2. Project part 1:
   - Programming language selection for the project
   - Due 9/14

Why study programming languages?

Expressing solutions to problems

- Can view PL as language for expressing solutions to problems
- Languages constrain what can be expressed $\Rightarrow$ what can be solved
- Studying PL $\Rightarrow$ learn/create new ways to express/solve problems

Choosing right PL for problems

- All PLs are theoretically equivalent in power ("Turing equivalent")
- PLs are tools: some better for some jobs

Choosing right PL for problems

- All PLs are theoretically equivalent in power ("Turing equivalent")
- PLs are tools: some better for some jobs
- Some example problem areas: computational biology, simulation, business data processing, GUIs, AI, data mining, statistical processing, CAD/CAM,. . .

Choosing right PL for problems

- All PLs are theoretically equivalent in power ("Turing equivalent")
- PLs are tools: some better for some jobs
- Some example problem areas: computational biology, simulation, business data processing, GUIs, AI, data mining, statistical processing, CAD/CAM,. . .
- Limited if only know a couple of languages – even if you are proficient

Choosing right PL for problems

- All PLs are theoretically equivalent in power ("Turing equivalent")
- PLs are tools: some better for some jobs
- Some example problem areas: computational biology, simulation, business data processing, GUIs, AI, data mining, statistical processing, CAD/CAM,. . .
- Limited if only know a couple of languages – even if you are proficient
- More languages you know ⇒ more ways to express solutions
  - Can choose language with feature you need
  - If you know about a feature that language doesn't have ⇒ implement it in the language

Learning new PL

- Learning abstract concepts underlying languages helps learn new languages – vocabulary for talking/thinking about them
- Increases ability to read and understand unfamiliar languages
- Popularity of PLs change over time (e.g., Tiobe index). . .
- . . . but the theoretical underpinnings don't

Implementing PLs

- One view: PL defines a virtual machine for solving problems
- But VM has to run on real one
- Understanding PL concepts $\Rightarrow$ essential if implementing compiler/interpreter
- Understanding PL implementation can also help:
  - predict performance
  - write more efficient programs
  - avoid subtle bugs caused by the implementation
- Exploit any helpful features of the implementation

Improving your use of PLs

- Programmer may not know or use all ways of using the language
  - PLs usually very large $\Rightarrow$ seldom use entire language
  - May have different ways of programming (functional, imperative, OO)
- Studying PL $\Rightarrow$
  - understand the language features better
  - understand what features could be present
  - $\therefore$ better use of languages you already know

Understanding computer science

- Knowing history of PLs $\Rightarrow$ know what computer scientists have tried, used, discarded, etc. – and why
- Thus helps avoid making past mistakes, reinventing the wheel
- Helps understand current SOA:
  - Trends in PL design and use
  - Why some languages are more popular than others

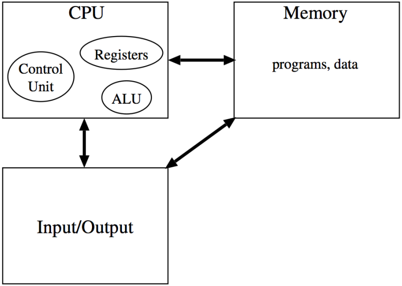
Programming language paradigms

What is a paradigm?

- Dictionary (Kuhnian definition): paradigm is "a worldview underlying the theories and methodology of a particular scientific subject" [New Oxford American Dictionary]
- Looser usage in computer science: an archetype, category
- PL paradigm: way of thinking, pattern of characteristics that underlie a set of languages
- Main PL paradigms:
  - Imperative (or procedural)
  - Object-oriented
  - Functional
  - Logical
  - Declarative

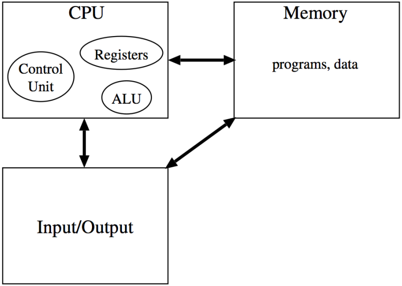
Imperative/procedural paradigm

- Oldest
- Based on the von Neumann computer architecture



Imperative/procedural paradigm

- Oldest
- Based on the von Neumann computer architecture



- This is the paradigm's ontological commitment

Imperative/procedural paradigm

- Program = series of instructions

Imperative/procedural paradigm

- Program = series of instructions
- State = contents of memory location

Imperative/procedural paradigm

- Program = series of instructions
- State = contents of memory location
- Program and data both in memory, indistinguishable

Imperative/procedural paradigm

- Program = series of instructions
- State = contents of memory location
- Program and data both in memory, indistinguishable
- Language features:
  - Variables (state), assignment
  - Conditional execution
  - Loops
  - Procedure calls

Imperative/procedural paradigm

- Program = series of instructions
- State = contents of memory location
- Program and data both in memory, indistinguishable
- Language features:
  - Variables (state), assignment
  - Conditional execution
  - Loops
  - Procedure calls
- Examples: Fortran, Python, Perl, parts of Java, C,. . .

Object-oriented paradigm

- Ontological commitment:
  - World consists of objects
  - Objects have internal state
  - Objects have encapsulated behavior
- Language features:
  - Classes, instances, inheritance
  - State: instance variables
  - Behavior: methods or messages
- Polymorphism
- Examples: Smalltalk, Java, C++, C#, Python, Lisp,. . .

Functional paradigm

- Ontological commitment: world consists of things (values) and functions on those things
- Language features:
  - Functional composition
  - Recursion
  - No state (in “pure” FL)
- Some have aspects of other paradigms
- Some languages from other paradigms have functional aspects
- Examples: ML, Scheme, Haskell, Lisp,. . .

Logic paradigm

- Ontological commitment: world consists of things and statements about things that are true/false
- Language features:
  - Declarative style: make statement about what should be true
  - Facts are stated in logic (usually Horn clauses)
  - Usually contain theorem prover (e.g., resolution TP)
- Examples: Prolog (mainly), ToonTalk, OWL (sort of, with TP support)

Declarative paradigm

- Ontological commitment: there are statements that can be made about the things in the world
- Logic languages $\subset$ declarative languages
- Examples: database languages (SQL, e.g.), XPath (for XML)

Programming domains

Programming domains

- Problems to be solved fall into domains – e.g.,
  - Scientific applications
  - Business applications
  - Databases, “big data”
  - Healthcare applications
  - Media applications & games
  - Artificial intelligence & data mining
  - Systems programming
  - Internet/Web programming
  - Embedded systems: IoT, industrial control, robotics
  - Consumer apps
  - Military applications
- Different domains $\Rightarrow$ different requirements for the language(s)

Programming domains

- Some languages: domain-specific (or created for a domain)
- Some languages: general-purpose

Scientific applications

- First domain for computers (with military): 1940s
- Require floating-point operations
- Few sophisticated data structures or control structures needed
- Historically imperative (e.g., Fortran, C, Python), now OOP too (e.g., Java, C++, Python)
- Critical feature: efficiency
- Wide range in use: Fortran, C, Python (e.g., w/ NumPy), R, Java, C++,. . .

Business applications

- Business applications gained importance in 1950s ($\Rightarrow$ special-purpose computers)
- Floating point not very important
- I/O capacity and sophistication very important
- First business language: COBOL (COmmon Business Oriented Language) – Adm. Grace Hopper
  - Very verbose
  - Supposedly easy for business people to learn
  - Still in use – some estimates: possibly most common language in world (in lines of code)
  - Contemporary COBOL has OOP, other modern features
- Other languages used, too: RPG, general-purpose languages

Databases & big data

- Database management systems (DBMS) require:
  - Fault tolerance for data
  - Efficient storage and retrieval mechanisms
- Relational databases, OO databases
- Languages for DBMS:
  - Efficiency
  - Able to refer to persistent data
  - Ability to express sophisticated queries to the database
- SQL (Structured Query Language): declarative, succinct, powerful access to relational algebra
- "Big data": ↑↑ need for data storage, efficient retrieval
  - Data framework (e.g., Apache Hadoop) rather than DBMS
  - General- and special-purpose languages (e.g., Pig framework w/ Pig Latin statements)

Healthcare applications

- Requirements from DBMS, business, science domains
- Additional requirements on language/frameworks:
  - privacy protection
  - security
  - assurance of correctness
- Languages:
  - Many general-purpose languages
  - In 2013, CIO.com lists these among "hottest healthcare" programming skills for healthcare computing professionals:

SQL	Java	JavaScript
C	C++	C#
PHP	XML/HTML	ASP.net

(HTML, XML, and ASP.net aren't programming languages)

Media & games

- Media: movies (CGI), music, VR, . . .
  - Efficiency
  - Ability to manipulate binary data
  - Access large amount of data
  - Access hardware
  - General-purpose languages, e.g., Python + libraries
- Games:
  - Access hardware
  - Languages: C++, e.g. (Unity, Unreal Engine)
  - Extensions for games: e.g., C#, JavaScript, Boo (Unity), C++ (Unreal)

Artificial intelligence & data mining

- Symbolic AI:
  - Mostly symbolic, not numeric, processing
  - Data structures: trees, lists
- Languages:
  - Need easy support for symbols
  - Linked list data structures useful
- First AI language: Lisp (LISt Processing language) – 1958
  - Symbols, linked lists - built-in data types
  - Programs & data: both lists
  - Easy introspection, program creation by programs
- Other languages: Scheme, Prolog, Haskell, general-purpose languages

Artificial intelligence & data mining

- Deep learning (neural nets), other machine learning
  - $\Rightarrow$ numeric processing, speed
  - C, C++, Python (NumPy, Theano, Tensorflow, Keras, . . .)
- Data mining: shares requirements with AI and DB/big data

Systems programming

- Systems programming – operating systems, drivers, networking, compilers, . . .
- Language requirements:
  - Need access to raw machine
  - Need extreme efficiency
  - Helpful if assembly can be mixed w/ HLL
- Languages:
  - Reason C was created
    - Trades safety for speed
    - Low-level HLL
  - PL/S: version of PL/I, IBM's systems language

Internet/Web programming

- Markup languages (XML, HTML):
  - Not programming languages
  - For data, display description
- Need dynamic content
  - Server side:
    - DB access, access to other programs, ability to create HTML
    - PHP, Python, Perl, Ruby, Java, .NET
  - Client side:
    - Need access to DOM (document object model), control of canvas
    - JavaScript, Flash, Java applets
  - AJAX (Asynchronous JavaScript and XML)
    - Group of technologies for client-server communication
    - Display: HTML/XHTML + CSS
    - Communication: XML, JSON, XMLHttpRequest
    - JavaScript on client, PHP (etc.) on server

Embedded systems, etc.

- Software as integral part of hardware system
- E.g., robotics, “Internet of Things” (IoT), industrial control
- Shares many similarities with systems programming
- Real-time requirement
- Languages: general-purpose – C, etc., even (especially?) Java

Consumer apps

- Wide range of applications: desktop, laptop, mobile
- Wide range of requirements: many of preceding
- Languages: Many: C, C++, C#, Swift, Objective C,. . .

Military applications

- Wide range of applications
- Virtually all of the preceding
- Languages: C, Ada, C++,...