CSE 3302
Programming Languages
Lecture 1: Course Overview, Introduction, and History

(based on slides by Chengkai Li)

Leonidas Fegaras
University of Texas at Arlington

Information

- Class: TuTh 2:00-3:20pm
- Instructor: Leonidas Fegaras
- Office: ERB 653 (Engineering Research Bldg)
- Phone: (817) 272-3629
- Email: fegaras@cse.uta.edu
- Office hours: TuTh 3:30-5:30pm (after class)
- Web site: http://lambda.uta.edu/cse3302/

Visit the class web page often. It will contain reading assignments, homework and programming assignments, class notes, grades, etc.

Description

Catalogue Description:

- Introduction, analysis, and evaluation of the important concepts found in a variety of programming languages. Formalisms useful in specifying language syntax and semantics; programming language paradigms such as algorithmic, functional, logic, and object-oriented.

Objectives:

- To analyze and evaluate important features found in a variety of programming languages, to study formalisms for specifying language syntax and semantics, and to gain understanding of the important programming language paradigms.

Why do you need this Course?

- To gain exposure to various kinds of programming languages (PLs) and paradigms
- To understand and evaluate the various features that make these languages different
- To understand the principles behind these features
  - will help you learn new PLs faster
  - will help you take advantage of the full power of a PL
  - will improve your programming skills
  - will be able to apply ideas and styles across paradigms
- To understand programming better
  - **Syntax**: the form of a program
  - **Semantics**: the meaning of a program
  - **Pragmatics**: the implementation of a PL

What this Course is **not** about

- To become an expert in a particular PL
  - We will learn the principles behind the design of many modern PLs, but we will not fully cover any PL
- To learn how to build compilers for PLs
  - This is the topic of CSE 4305 (Compilers for Algorithmic Languages)
- To fully understand programming semantics

Prerequisites

Prerequisites:

- CSE 2320 (Algorithms & Data Structures) or equivalent
- or consent of instructor

Students must:

- have experience with at least one major programming language
  - eg, C, Java, C++, ...
- be familiar with data structure concepts and algorithms
  - such as lists, trees, sorting, hashing, etc

Students without adequate preparation are at substantial risk of failing this course.

Reading Material

Required Textbook and Notes:

- Kenneth C. Louden: *Programming Languages - Principles and Practices*, 2nd Edition, Thomson Course Technology.
  - Look at the textbook homepage (http://www.cs.sjsu.edu/~louden/pltext/) for errata, selected solutions, and reference materials
- Lecture slides (to be available soon)

Other optional textbooks (useful for additional background and explanation):

- Bruce J. MacLennan: *Principles of Programming Languages: Design, Evaluation, and Implementation*, second edition
- Robert W. Sebesta: *Concepts of Programming Languages*, seventh edition

Grading

- The final score will be based on
  - 20% homework assignments
  - 20% programming assignments
  - 20% midterm exam
  - 30% final exam (comprehensive)
  - 10% essay

- Final grades will be assigned according to the following scale:
  - A: score >= 90
  - B: 80 <= score < 90
  - C: 70 <= score < 80
  - D: 60 <= score < 70
  - F: score < 60
- Sometimes, I use lower cutoff points, depending on the overall performance of the class

Exams

- Both exams are open textbook and open notes
  - only the class textbook
  - all notes must be securely bound in one notebook
- The final exam will cover the material from the first lecture up to and including the last lecture
- Once the exam grades are posted, you will have 10 business days to dispute your grade and get your exam re-evaluated
  - Before you request for re-evaluation, make sure to compare your answer with the solution
  - No re-evaluation will be entertained after the 10 day period
- No makeup exams will be given unless there is a justifiable reason (such as illness, sickness or death in the family)
  - If you miss an exam and you can prove that your reason is justifiable, you should arrange with the instructor to take the makeup exam within a week from the regular exam time
  - For any other case, you will get a zero grade for the missed exam

Homework and Programming Assignments

- Homework and programming assignments must be done individually
  - No copying is permitted
- There will be 5 homework and 5 programming assignments
  - Each assignment will be worth the same number of points
    - 4% of the final score
  - Each assignment is due by 11:55pm on its due date
  - All submissions must be done electronically using the class web site
    - we don't accept email submission or hard-copy
  - Late assignments will be marked 20 points off per day (out of 100 max). So, there is no point submitting a homework more than 4 days late!
    - This penalty cannot be waived, unless there was a case of illness or other substantial impediment beyond your control, with proof in documents from the school

Essay

- The essay assignment will count for the ABET requirements for written communication
- You must get a passing score (37.5 out of 100) for the essay in order to satisfy ABET requirement
- If a student does not demonstrate timely achievement of the assessment and would otherwise pass the course, then the student will receive a grade of incomplete (I) for the course until such time when the assessment instrument is satisfactorily complete
- The student will be allowed to re-attempt demonstration of satisfactory completion during the next offering of the course
  - In such case, certain penalty will be given when calculating the final course grade

Cheating

- Homework and programming assignments, as well as the essay must be done individually. **No copying is permitted**
- Cheating involves giving assistance to or receiving assistance from other students or from other individuals, copying material from the web, etc
- I strictly adhere to the University of Texas at Arlington rules and guidelines for handling violations of academic dishonesty
- Please refer to the pamphlet "CHEATING: Definitions and Consequences" for additional information
- If any one is caught for cheating, or indulge in plagiarism or collusion on a programming assignment or on a exam, the grade for the entire course will be an automatic Fail grade (F)

How to do Well in this Course

- Students who get the most out of this course will be the ones who put in the most effort
- If you want to do well
  - attend all the lectures
  - read the assigned sections of the book
  - start early on your assignments
- Working out the questions from book chapters will immensely help in doing well on assignments and exams
- If you are having difficulty, the instructor and the GTA will be more than happy to help you
  - In addition to regular office hours, the best way of communication with the instructor or the GTA is through email
  - If you can't make it to the scheduled office hours but really need help, contact one of us for an appointment

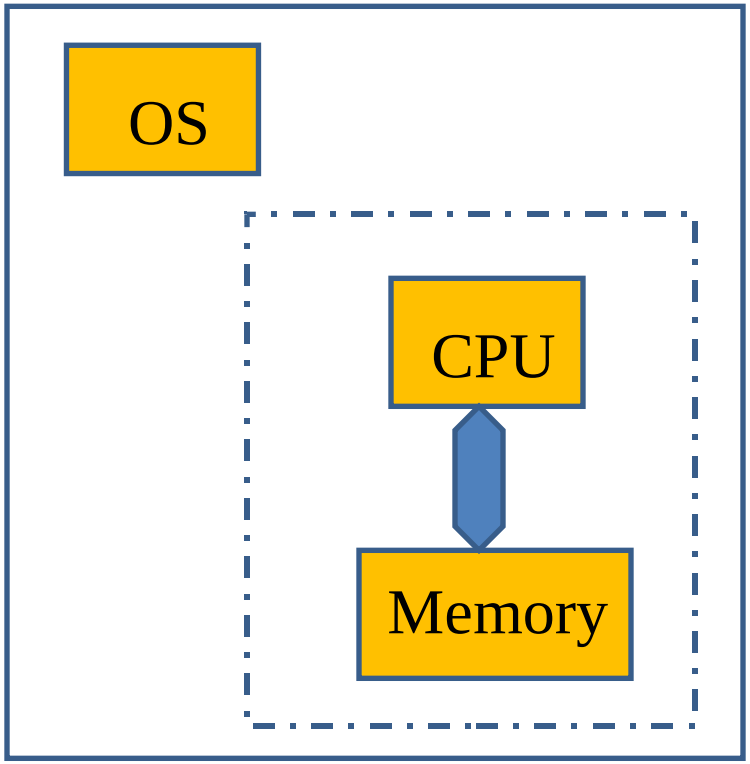
Tentative Schedule

- Introduction (1 lecture)
- History (1 lecture)
- Syntax (4 lectures)
- Semantics (2 lectures)
- Data Types (4 lectures)
- Control (3 lectures)
- Abstract Data Types (2 lectures)
- OO (Java, Smalltalk) (3 lectures)
- Functional (Lisp, Scheme, ML, Haskell) (4 lectures)
- Logic (Prolog) (3 lectures)

What is a Programming Language?

- Programmers use PLs to instruct computers to perform various tasks

```
printf("hello, world\n");
```



What kind of Programming Language?

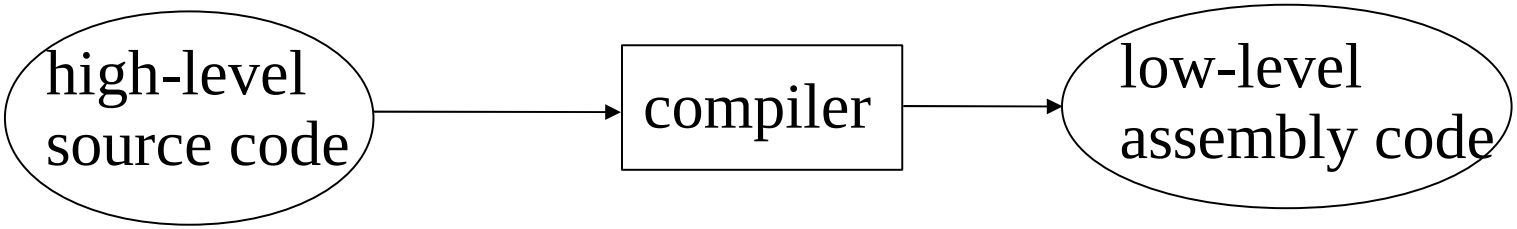
We are talking about general-purpose, high-level programming languages

We are *not* interested in:

- Assembly or machine language
- Domain-specific languages (to be used by domain experts)
  - SQL for relational databases
  - XPath for XML
  - Spice for hardware
  - Mathematica for engineers
  - Postscript for document publishing
  - Lex/YACC to describe compilers
- Visual languages
  - GUI

The Machinery

- Program compilation:



eg, Java program

easy to understand
user-friendly syntax
many high-level programming constructs
machine-independent
variables, procedures, classes, ...

eg, Intel Pentium assembly

hard to understand
specific to hardware
registers & unnamed locations

This is the topic of the Compiler's course (CSE 4305)

Various Paradigms

Imperative/Procedural: (ALGOL, FORTRAN , Pascal, C, Ada)

- Use control statements to manipulate the data and program states

Object-Oriented: (C++, Java, Smalltalk)

- Based on objects

Functional: (Lisp, Scheme, ML, Haskell)

- Use function evaluations only

Logic: (Prolog)

- Use axioms (statements about truth) and inference

Scripting: (Python, Perl, Javascript, PHP)

- Mixed paradigms; allow fast prototyping

What Makes a PL Good?

- Human readability
  - Abstractions for describing actions
    - **Data abstraction**: the subject of computation
    - **Control abstraction**: the transfer of control in computation
  - Human-to-human communications
- Easy to learn and use (writability)
- Ease of development and maintenance of programs
  - Facilitate software engineering processes
  - Reusability
  - Use of well-designed standard libraries
- Has unambiguous syntax and semantics
  - General and orthogonal features
- Easy to implement and optimize
- Efficiency of execution
- Efficiency of translation

A Day in the Life of a Web Programmer

- To develop a Web site
  - Need to separate presentation, style, and functionality
  - Need to do both client and server programming
- Client-side programming
  - Javascript (functionality) embedded in HTML (presentation), which uses CSS stylesheets (style)
- Server-side programming
  - Heavy-duty programs that generate HTML code to interact with the client browser through HTML forms or Ajax asynchronous requests
    - CGI scripts
    - Scripting: PHP, Perl
    - Java servlets, C# ...
  - Need to access relational databases to access application/customer data
    - SQL
  - Need to manipulate XML for web services
    - XPath, XSLT, XQuery

My Own History

- Started with FORTRAN in early '80
  - I learned Pascal, Algol, Lisp, and a little bit Cobol (!!!) for my BS degree
  - I did my senior thesis in PL-I
- I used C for my MS (late '80)
  - I worked heavily in the development of Common Lisp
  - I did most of my MS/PhD work on a Lisp machine!
- I did my PhD degree mostly in C++ (early '90)
  - I also used SML and Prolog for some prototypes
- When I was a post-doc, I mostly used C++ and Haskell
- After I joined UTA ('96), I used C++ and then Java
- Now, I use Haskell and Java for most of my projects and Java for teaching
  - I also use SQL, XQuery, PHP, Javascript

Why do we have so many PLs?

- Why don't we just pick the best PL and forget the others?
  - Aren't they all supposed to be general purpose, Turing-complete anyway?
- Many 'best' PLs:
    - Sun says it's Java
    - Microsoft says it's C#
    - I say it's Haskell
- Reasons:
  - Some PLs are more effective in some domains than others
    - Prolog for AI
    - OO for GUI
  - Tradition
    - Fortran for scientists
    - C for system programmers
  - Compatibility with legacy code and with existing libraries
  - Cost of retraining programmers
  - CS curriculum
  - Politics

Some Comparison

(Just browse these sites briefly)

- See how the Fibonacci Number Program is expressed in some of the most popular PLs
  - http://www.scriptol.com/programming/fibonacci.php
- The computer language benchmarks game
  - http://shootout.alioth.debian.org/u32q/

History

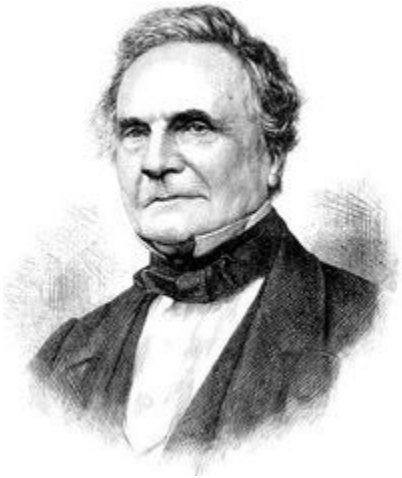
- Early History : The First Programmer
- The 1940s: The First Computers
- The 1950s: The First Programming Languages
- The 1960s: An Explosion in Programming Languages
- The 1970s: Simplicity, Abstraction, Study
- The 1980s: New Directions and OO
- The 1990s: Consolidation, Internet, Libraries, and Scripting
- The Future

The First Programmer

- Before the birth of computers
- Jacquard Loom (early 1800s)
  - translated card patterns into cloth designs

The First Programmer

- Charles Babbage's analytical engine (1830s and 1840s)
  - Devoted entirely to computation
  - Programs: cards with data and operations
  - Difference Engine: which inspired Analytical Engine (the design was realized in 1991)
- Ada Lovelace – first programmer (daughter of Byron)



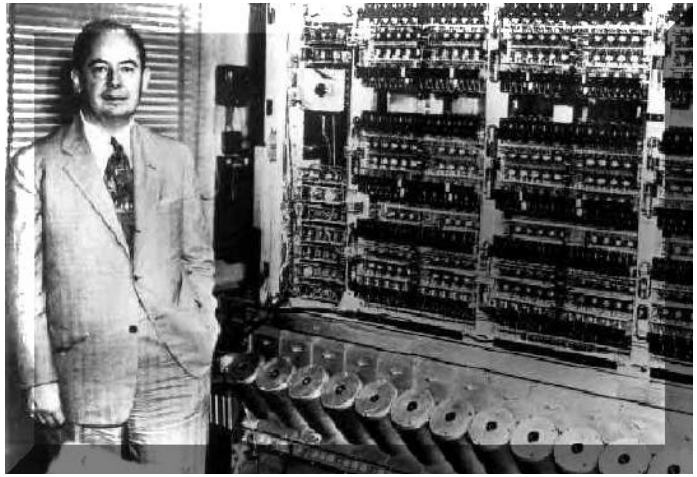
The First Computers

- **ENIAC (1943)**
  - First electronic computer
  - U. Penn

http://en.wikipedia.org/wiki/ENIAC

- **EDVAC (1945)**
  - John von Neumann
  - von Neumann architecture

    "Stored program": data and programs in the same space



www.agers.cfwb.be/apsdt/figinfo26.htm

The First Computers

- **Z3 (1941)**
  - Konrad Zuse
  - First digital computer
  - Electromechanical, rather than electronic

http://user.cs.tu-berlin.de/~zuse/Konrad_Zuse/index.html

- **Plankalkul (Plan Calculus)**
  - 1945
  - Eventually published in 1972
  - First compiler implemented in 2000

http://en.wikipedia.org/wiki/Konrad_Zuse

Machine Code and Assembly Language

- **Machine code**: bit sequences

```
000000 00001 00010 00110 00000 100000
100011 00011 01000 00000 00001 000100
000010 00000 00000 00000 10000 000001
```

- **Assembly program**: symbolic representation of machine codes

```
LDA   SUB
CMA
INC
ADD  MIN
STA   DIF
```

The '50s: The First Programming Languages

- FORTRAN: the first higher-level programming language

Languages following FORTRAN

- COBOL
- Algol60
- LISP
- APL

FORTRAN

- The first language

  1954-1957

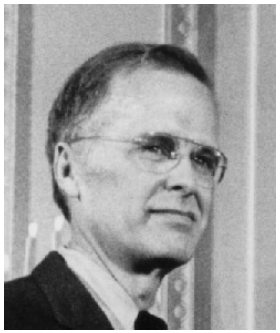
  John Backus, et. al. (IBM)

- Scientific and engineering applications (FORmula TRANslation).
- Goal: generate fast machine code. Its compiler is still among the most efficient.
- Contributions: array, loops by indexed variables, if-statement
- Still widely used today (Fortran, II, III, IV, 66, 77, 90, 95, 2003, 2008).

FORTRAN

- John Backus: IBM group

1977 ACM Turing Award: “for profound, influential, and lasting contributions to the design of practical high-level programming systems, notably through his work on FORTRAN, and for seminal publication of formal procedures for the specification of programming languages.”



www.columbia.edu/acis/history/backus.html

Major languages following FORTRAN

- COBOL
- Algol60
- LISP
- APL

COBOL

- COmmon Business-Oriented Language
  1959-1960
  Grace Hopper, et. al. (US Department of Defense)

- Business applications: banks and corporations
- Still widely used

http://en.wikipedia.org/wiki/Grace_Murray_Hopper

COBOL

- Goal: allow nonprogrammers to read/understand programs
- Consequences:
  - Very wordy, like English
    - *c++* vs. *ADD 1 TO C GIVING C*
    - Can be difficult to write complex algorithms
  - Human readability improved (or only complicated?)
- Contributions:
  - Record structure
  - Separate data structures from execution
  - Output formatting by examples
- COBOL 2002: OO programming

Algol60

- ALGOrithmic Language, 1958-1960

  A committee of European and American computer scientists (John Backus and John McCarthy involved)
- Contributions:
  - free-format
  - Backus-Naur forms (BNF) for defining syntax
  - type declarations for variables
  - block-structure, begin-end
  - recursion
  - pass-by-value parameters
- Impacts:
  - one of the most influential programming languages
  - most imperative languages are derivatives of Algol: Pascal, C/C++, Ada, Java
  - standard way of describing algorithms in research papers for 30 years

LISP

- LISt Processor
  Developed in late 1950s by John McCarthy (MIT, at Stanford now)
  1971 Turing Award for contributions in AI
- AI applications, still dominating
- Contributions:
  - first one to depart from imperative/procedural paradigm: functional programming language
  - Garbage collection
  - Recursion, s-expression
- Limitations;
  - Could not run efficiently on von Neumann architecture
  - LISP-specific machines
- Variants: Common LISP, Scheme
- Following LISP: ML, Haskell

APL

- A Programming Language

  Developed in late 1950s to early 60s by Kenneth E. Iverson (Harvard and IBM)

  1979 Turing Award for contributions to mathematical notation and PL theory

- A language for programming mathematical computations
  - arrays and matrices
- Functional style, influenced FP and modern function languages
- Drawbacks:
  - No structuring
  - Greek symbols, requires special terminal keyboard
  - Extremely difficult to read

Summary of 1950s

- Huge success and big impact:
  - Pioneered imperative and functional programming
  - Many of these Pls are still used much today
  - Many derivatives

- The 1960s is not equally fruitful

The '60s: An Explosion in PLs

- Hundreds of programming languages
- PL/I
- Algol 68
- SNOBOL
- Simula67
- BASIC

PL/I

- 1963-1964, by IBM
- Goal:
  - Universal language, “language to end all languages.”
    - combine features of FORTRAN, COBOL and Algol60
    - concurrency
    - exception handling
  - for IBM 360
- Can be considered to be a failure:
  - translators were difficult to write, slow, huge and unreliable
  - difficult to learn and use
  - forward-looking, but simply ahead of its time

Simula67

- 1965-1967

  By Kristen Nygaard and Ole-Johan Dahl (Norwegian Computing Center)

  2001 Turing Award for OO and Simula

- Based on Simula I and Algol60
- Designed for simulations
- First OO language
  - object, class, subclass (inheritance), virtual method, coroutine
- Ahead of its time
- Inefficient

Basic

- Beginner's All-purpose Symbolic Instruction Code
- Developed in 1964 by John Kemeny and Thomas Kurtz (Dartmouth)

- Goal:
  - Simple language for non-experts to use

- Popular for schools and homes
  - Altair BASIC for personal computers, by Bill Gates, Paul Allen, and Monte Davidoff (1975)
- Dialect: Visual Basic

The '70s: Simplicity, Abstraction, Study

- Tremendous success
  - few new concepts
  - simplicity and consistency
- Algol-W by Niklaus Wirth (ETH Zurich, 1984 Turing Award) and C.A.R. Hoare (1980 Turing Award)
  - response to the direction in 1960s
- Pascal 1971 by Niklaus Wirth
  - popular for teaching PL
- C 1972 by Dennis Ritchie (Bell Labs, 1983 Turing Award)
  - Successful partially due to the popularity of UNIX
- CLU, Euclid, Mesa: Abstract Data Type (ADT)

The '80s: New Directions and OO

- Following the experiments of ADT in 70s
  - Ada, Modula-2
- Object-Oriented Programming
  - Smalltalk, C++, Eiffel, Object C, Object Pascal, Oberon
- Functional Programming
  - Scheme, ML, Haskell, Miranda, FP
- Logic Programming
  - Prolog

Ada

- Named after Ada Lovelace

  Developed in 1980 by the Department of Defense
  - Hundreds of languages were used by DoD
  - Required in DoD projects, 1987-1997
- Contributions:
  - ADT (package)
  - concurrency (task)
  - exception handling
- Universal language, the PL/I of 80s, but didn't fail
  - carefully designed
  - required to use

Smalltalk

- Developed in 1980 by Alan Kay (2003 Turing Award), Dan Ingalls, et. al. (Xerox PARC)

- Inspired by Simula67
- Contributions
  - purest OO language
  - graphical user interface, mouse (limited its use, as such hardware was not generally available)
  - Push C++ and OO into spotlight
- Still used much today

C++

- Developed in 1980 by Bjarne Stroustrup (Bell Labs, now at TAMU)
- Extensions from Simula67 and C, “C with Classes”

Other Paradigms

- Functional Programming:
  - Common Lisp, Scheme, ML, Haskell (pure functional programming language)
- Logic Programming:
  - Prolog

The '90s: Internet, Scripting

- OO widely adopted (C++ was going to dominate)
- ... then Java came
- ... and C# followed
- Scripting languages became general-purpose languages: Perl, Tcl, Python, Javascript, PHP, …

Java

- Developed in 1995 by James Gosling et. al. (Sun)
- Was for embedded consumer-electronic applications (set-top box), then for Internet/Web and network applications
- Based on C++
- Differences
  - Pros: richer libraries (API), portability (compile-once, run-anywhere)
  - Cons: slower than C++, no ISO/ANSI standard (controlled by Sun).
  - references vs. pointers
  - garbage collection
  - security

What's next?

- C/C++/C#?
- Java?
- Haskell?
- A new language?