Principles of programming languages (2007)
Lecture 1


Natalia Silvis-Cividjian
e-mail: nsilvis@few.vu.nl

What makes programming languages such an interesting subject?

• The amazing variety
• The odd controversies
• The intriguing evolution

Lecture 1. Topics
n Student survey
n Introduction
n History of major programming languages
n About this course

• Which programming languages do you know?
  Very well:
  A little:
  Just heard about:
• Which language is your favourite?
• What languages do you find interesting or would you like to learn?
• What is the first high-level programming language and when did it appear?
• When did Java appear and for what purposes?
• Can you give a definition of the following concepts? If yes, give an example.
  • Anonymous function
  • Coercion
  • Polymorphism
  • Automatic garbage collection
  • Strong/weak type checking
• What is:
  • Python
  • Ruby
  • Eiffel
  • Self
• Explain to a layman what is OOP and why is it so important?
• What do you expect from this course?
The amazing variety

There are very many, very different languages (ca 2500)

Often grouped into four families:
- Imperative
- Functional
- Logic
- Object-oriented

Imperative Languages

Example: a factorial function in C

```
int fact(int n) {
    intsofar = 1;
    while (n>0)sofar *= n--;
    returnsofar;
}
```

Hallmarks:
- Assignment
- Iteration
- Order of execution is critical

Visual languages

Subcategory of imperative languages
Visual Basic
Once called fourth-generation languages

Object-Oriented Languages

Example: an Java object

```
public class MyInt {
    private int value;
    public MyInt(int value) {
        this.value = value;
    }
    public int getValue() {
        return value;
    }
    public MyInt getFact() {
        return new MyInt(fact(value));
    }
    private int fact(int n) {
        intsofar = 1;
        while (n > 1)sofar *= n--;
        returnsofar;
    }
}
```

Hallmarks:
- Usually imperative, plus...
  - Constructs to help programmers use “objects”—little bundles of data that know how to do things to themselves

Object-Oriented Languages
### Functional Languages
- Example: a factorial function in ML

```plaintext
fun fact x = if x <= 0 then 1 else x * fact(x-1);
```
- Hallmarks:
  - No assignment, no side effects
  - Heavy use of recursion: no iterations

### Another Functional Language
- Example: a factorial function in Lisp

```plaintext
(defun fact (x) (if (<= x 0) 1 (* x (fact (- x 1))))
```
- Looks very different from ML
- But ML and Lisp are closely related

### Logic Languages
- Example: a factorial function in Prolog

```plaintext
fact(X,1) :- X =:= 1.
fact(X,Fact) :- X > 1, NewX is X - 1, fact(NewX,NF), Fact is X * NF.
```
- Hallmark:
  - Program expressed as rules in formal logic

### The Odd Controversies
- Programming languages are the subject of many heated debates:
  - Partisan arguments
  - Language standards
  - Fundamental definitions

### The best programming language
- Java
- Fortran
- Cobol
- C
- C++
- PHP
- JavaScript
- C#
- ML
- Prolog
No clear winner
Obviously, there is no best language for all situations. The best language might depend on many things:
- Type of program
- Reason the program is built
- Size of program
- Programmer familiarity
- Time available
- Cost
- Legacy

Language Partisans
- There is a lot of argument about the relative merits of different languages
- Every language has partisans, who praise it in extreme terms and defend it against all detractors
- To experience some of this, explore newsgroups: comp.lang.*

Evaluation Criteria
- Readability
- Writability
- Reliability

Evaluation criteria

<table>
<thead>
<tr>
<th>Characteristic</th>
<th>Readability</th>
<th>Writability</th>
<th>Reliability</th>
</tr>
</thead>
<tbody>
<tr>
<td>Simplicity/orthogonality</td>
<td>•</td>
<td>•</td>
<td>•</td>
</tr>
<tr>
<td>Control structures</td>
<td>•</td>
<td>•</td>
<td>•</td>
</tr>
<tr>
<td>Data types and structures</td>
<td>•</td>
<td>•</td>
<td>•</td>
</tr>
<tr>
<td>Syntax design</td>
<td>•</td>
<td>•</td>
<td>•</td>
</tr>
<tr>
<td>Support for abstraction</td>
<td>•</td>
<td>•</td>
<td>•</td>
</tr>
<tr>
<td>Expressivity</td>
<td>•</td>
<td>•</td>
<td>•</td>
</tr>
<tr>
<td>Type checking</td>
<td>•</td>
<td>•</td>
<td>•</td>
</tr>
<tr>
<td>Exception handling</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Restricted aliasing</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Programming domains
- Scientific applications
- Business applications
- Artificial intelligence
- Scripting languages
- Systems programming
- Internet and Web

Language Standards
- The documents that define language standards are often drafted by international committees
- Can be a slow and complicated process
- Fortran 8X, 88, 90 standard released in 1991
Basic Definitions

- Some terms refer to fuzzy concepts: all those language family names, for example
- No problem; just remember they are fuzzy
  - Bad: Is X really an object-oriented language?
  - Good: What aspects of X support an object-oriented style of programming?

The amazing variety
- The odd controversies
- The intriguing evolution

History of major programming languages

An updated genealogical diagram can be found at http://www.levenez.com

Read Ch.2 Sebesta, ch. 24 Weber

Plankalkül (1945)

Plankalkül (Konrad Zuse) = calculus for a computing plan
a high level language to express computations on computer Z4
Has:
- Machine independent operations, assignment statements,
- floating point data types, records, expressions with parenthesis,
- conditional statements but no else, repetition of statements, subroutines
- algorithms from array sorting to playing chess on 60 pages
Plankalkül

| A + 1 => A |
| V | 4 5 |
| S | 1.n 1.n |

But...

- Difficult notation
- Z4 had a memory of 64 words of 32 bits each
- Never implemented
- Published vary late in 1972

Intermediate steps

- Machine code
  - Poor readability
  - Difficult to modify
  - No hardware with floating point arithmetic, no indexing
- Pseudocode

Fortran (1954)

- FORTRAN = mathematical FORMula TRANslating System – first compiled high level language
- IBM 704 system has floating point instructions in hardware
- Promised the efficiency of machine code and the ease of programming of pseudocodes. Almost succeeded. Code was very fast.
- Most of the calculations were numeric.
- Computers were more expensive than programmers, so no dynamic storage

First step to sophistication:

ALGOL 58 and ALGOL 60

**Situation:** Languages were developed around single architecture IBM or UNIVAC, communication was difficult.

- No universal language
- No portable language

In 1958 ACM (USA) + GAMM (EUR) came together to discuss the design of one international language – compromises about spheres of influence.

**Goals:**

- Close to mathematical notations
- Good for describing algorithms
- Must be translatable to machine code
ALGOL 58 features
- Concept of type
- Names have any length
- Compound statements
- Semicolon as separator
- Assignment operator as :=
- Else-if clause
- But: abandoned by IBM.

ALGOL 60
- **New features:** block structure, pass by value and pass by name, subprogram recursion
- **Success:** standard way to publish algorithms for 20 years
  - All imperative languages are based on it.
  - First machine independent language
  - First language whose syntax was formally defined by BNF grammar
- **Failure:** Never widely used in USA, lack of support from IBM (Fortran compilers were faster), no I/O, formal syntax description

**Figure 2.3**
Genealogy of ALGOL 60

COBOL (1959)
- First language required by DoD
- Must look like simple English (managers can read code).
- Still the most widely used business applications language.

**Figure 2.4**
Genealogy of COBOL

**Figure 2.5**
Genealogy of programming languages
**Time sharing: BASIC (1964)**

A BASIC program [from R. Clark, Comparative Programming Languages]

---

**BASIC**

- Easy to learn and use by non-science (liberal arts) students
- Supposed to be a liberal arts programming language
- First widely used with time sharing (terminals instead of punch cards)

---

**PL/I**

- Developed by IBM
- Built as a language for both kinds of applications: scientific computing and business
- First named Fortran VI, then NPL, PL/I
- Has: pointers, concurrency, recursivity, error handling
- Nowadays nearly dead language: too complicated

---

**Everything for everybody: PL/I**

"Using PL/I must be like flying a plane with 7,000 buttons, switches, and handles to manipulate in the cockpit". (Edsger Dijkstra)
Data abstraction: SIMULA 67

**Figure 2.7**
Genealogy of SIMULA 67

- ALGOL 60 (1960)
- SIMULA I (1964)
- SIMULA 67 (1967)

SIMULA 67

- Based on ALGOL 60 for system simulations (Norway)
- Contributions:
  - coroutines
  - A structure called class
  - Classes are base for data abstraction
  - Classes include data and functionality
  - Objects and inheritance

Orthogonal design: ALGOL68

**Figure 2.8**
Genealogy of ALGOL 68

- ALGOL 60 (1960)
- ALGOL 68 (1968)

Descendants of ALGOL: Pascal

**Figure 2.9**
Genealogy of Pascal

- ALGOL W (1964)
- ALGOL 68 (1968)
- Pascal (1971)

Descendants of ALGOL:C

- Pascal (1971)
- Ada 83 (1983)
- Ada 95 (1995)

Largest design effort: Ada 1983
A D A

- Huge effort, much money, hundreds of people, DoD support (Honeywell/Bull)
- Contributions:
  - Packages for data abstraction
  - Exceptions handling
  - Generic programming units
  - Concurrency
- But: too large and complex, Compilers very difficult to build, the role of C++

Object-oriented: Smalltalk 1980

- [Fig. 2.12] Genealogy of Smalltalk
  - ALGOL 58 (1958)
  - ALGOL 60 (1960)
  - SIMULA 1 (1964)
  - SIMULA 67 (1967)
  - Smalltalk-80 (1980)

Combining imperative with OO: C++

- [Fig. 2.13] The ancestry of C++
  - Simula 67 (1967)
  - C (1971)
  - C++ (1985)

Impure based OO language: Java

- [Fig. 2.14] The ancestry of Java
  - C++ (1985)
  - Java (1994)

What makes a language successful?

- Expressive power
- Ease of use for a novice
- Ease of implementation
- Open source
- Excellent compilers
- Economics, patronage and inertia
Aims

- To understand the general concepts underlying programming languages
- To experience the diversity of programming languages. Not all languages are like Java!
- To be aware of the design and implementation trade-offs
- To learn to make a fast evaluation of a language and easily learn a new language.
- To be able to argue your own language choice and understand the choice made by others.

Questions?