CS342: Organization of Prog. Languages

Topic 1: Some History and Viewpoints

Topics:

• Why study programming languages?
• Languages all around us
• Hundreds of programming languages
• Yahoo!'s programming languages page
• A historical chart of modern programming language families
• Dissecting a language
• Computer language paradigms
• Key emphasis in different languages and families
• Different ways of looking at programming languages

Why Study Programming Languages?

- Be aware of the choices different languages offer.
- Understand concepts which show up in several languages.
- Learn new languages more quickly.
- Learn to simulate useful features in languages which lack them.
- Understand costs of features you use.
- Understand implementations to better use support tools (debuggers, etc).
- To learn what features will be added to your favourite languages in the future.
- So *you* will design reasonable languages in your future careers.
Languages All Around Us

- Natural languages: English, French, Russian, Mandarin, Mohawk, ...
- Specialized jargons
  - Air traffic control English,
  - Legal language,
  - Computer hardware installation language,
  - Cajun cooking recipe language,
  - ...
- Basic formal languages
  - Setting a VCR. Time/Date.
    Recording a program every weekday.
  - Click language for a mouse.
  - Decision tree at a call-center.
  - ...
Hundreds of Programming Languages

- Programming-language design began in the mid 1950-s.
- By mid 1990-s estimate 1000 programming languages had been defined.
- Perhaps 200-300 in active use today.
Some Historical Highlights

- Fortran

1960
- Cobol
- Algol 60
- APL
- Snobol
- BASIC
1965
- ANSI Fortran
- BCPL
- ISWIM*
- Cobol 68
- Algol 68
- Simula
- Maclisp
- Interlisp
- Forth
1970
- Pascal
- Concurrent
- C
- Prolog
- ML
- Scheme
1975
- Cobol 74
- CLU
- Mesa
- Fortran 77
- Icon
- CSP*
1980
- Modula 2
- Smalltalk
- Ada
- C++
- Common Lisp
1985
- Ada
- Modula 3
1990
- Fortran 90
- Haskell
- 00-Cobol
1995
- Fortran 95
- Java
- Ada 95
- Std C++
- Aldor
- XSLT
Disecting a Language

- Language vs library vs convention
- Syntax vs semantics

```java
import java.io.*;
import org.w3c.Entity;

class Example {
    public static void main(String[] args) {
        System.out.println("Hello.");
        System.out.println("Good-bye.");
    }
    public static void showVersion(Entity ent) {
        System.out.print(ent.getVersion());
    }
}
```
Dissecting a Language II

- **Languages** define what are the legal ways to combine symbols to make meaningful programs.
  E.g. above: use of braces (syntax), meaning of "static" (semantics).

- **Libraries** populate the environment with functions, objects, data.
  E.g. org.w3c.*
  borderline – standard libraries

- **Conventions** make programs more readable, and might be checked by auxiliary software tools.
  E.g. "args" as parameter of "main"
Two Ideas

- Orthogonality => independence.
  Example with basis vectors.

- Program vs data.

  printf("%x, %d, %*s\n", 8, 9, 10, "goodbye")
  (cons 2 3) VS (quote (cons 2 3))
**Computer Language Paradigms**

- **Functional languages** (Lisp, ML, Haskell)
  Functions take values, produce results, no side-effects, functions can create new functions from others

- **Dataflow languages** (Id, Val, OpenInventor)
  Data flows through a program, and is transformed by nodes

- **Constraint-based languages** (Prolog, Excel, Yacc)
  Logical or mathematical constraints among parts of the input determine which computations are performed

- **Imperative languages** [von Neumann languages] (C, Pascal)
  Assignments modify memory according to a set of prescribed steps.

- **Object oriented languages** (Smalltalk, C++, Java)
  Data objects provide methods to allow themselves to be used and modified.
• Hardware description languages (VHDL)
  Describe parallel hardware state transitions in time

• Data description languages (VRML, RPG, XML)
  Describe the layout of data

• Pattern-matching languages (Snobol, Icon, XSLT)
  Patterns or templates determine which rules are applied.
Key emphasis in different languages and families

<table>
<thead>
<tr>
<th>Language</th>
<th>Emphasis</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fortran</td>
<td>algebraic formulas, arrays</td>
</tr>
<tr>
<td>Lisp</td>
<td>linked list, reflection</td>
</tr>
<tr>
<td>Cobol</td>
<td>records, files</td>
</tr>
<tr>
<td>APL</td>
<td>interaction, high-level fns</td>
</tr>
<tr>
<td>Algol</td>
<td>lexical block structure</td>
</tr>
<tr>
<td>Snobol</td>
<td>pattern matching</td>
</tr>
<tr>
<td>Algol 68</td>
<td>orthogonality</td>
</tr>
<tr>
<td>Simula</td>
<td>classes</td>
</tr>
<tr>
<td>ISWIM, Scheme, ML</td>
<td>functional programming</td>
</tr>
<tr>
<td>Concurrent Pascal</td>
<td></td>
</tr>
<tr>
<td>CSP, SR</td>
<td>parallel programming</td>
</tr>
<tr>
<td>CLU</td>
<td>data and control abstraction</td>
</tr>
<tr>
<td>Smalltalk</td>
<td>object oriented programming</td>
</tr>
<tr>
<td>Ada</td>
<td>generics, exception handling</td>
</tr>
<tr>
<td>Aldor</td>
<td>higher-order programming</td>
</tr>
</tbody>
</table>
Different ways of Looking at Programming Languages

• Side-effecting vs pure
• Strict vs lazy evaluation
• Explicit sequencing vs constraint or pattern engine
• Data abstraction level
• Model of parallelism
• Early binding (static) or late binding (dynamic)
• Closed vs extensible (open-ended)
• Language vs library