Programming Languages

Syllabus

- Prerequisites
  Some familiarity with a contemporary programming language
- Textbook
  Michael L. Scott Programming Language Pragmatics
- Other material
  class notes and handouts
- Web site
- Class times
  MWF: 1:25 - 2:15 pm
- Class location
  101 J. Love Building
- Recitations
  Wed; sec 1 = 2:30 - 3:20 pm, sec 2 = 3:35 - 4:50 pm, sec 3 = 6:45 - 8:00 pm, sec 4 = 8:15 - 9:30 pm

Contact

- Instructor
  Dr. Robert van Engelen
- Email
  engelen@cs.fsu.edu
- Phone
  850 644 9661

Objectives

- Improve the background for choosing appropriate programming languages for certain classes of programming problems
- Be able in principle to program in an imperative (or procedural), an object-oriented, a functional, and a logical programming language
- Understand the significance of an implementation of a programming language in a compiler or interpreter
- Increase the ability to learn new programming languages
- Increase the capacity to express programming concepts and choose among alternative ways to express things
- Simulate useful features in languages that lack them
- Be able in principle to design a new programming language
- Make good use of debuggers and related tools

Note: the on-line syllabus allows you to access more detailed information about the course
**Programming Language History**

- **1940s**: The first electronic computers were monstrous contraptions programmed in machine language
  - Code not reusable or relocatable (displacable in memory)
- **Assembly languages** were invented to allow operations to be expressed with mnemonic abbreviations
  - Enables larger, reusable, and relocatable programs
  - Actual machine code produced by assembler
  - Early assembler: one-to-one correspondence between assembly and machine instructions
  - Later: macro expansion into multiple machine instructions
- **Mid 1950s**: development of Fortran, the first arguably higher-level language
  - Finally, programs could be developed that were machine-independent (is this problem really solved by modern languages? Java is arguably machine independent)
  - Main computing activity was numerical computation
  - Other high-level languages soon followed (Cobol, Algol 58, Lisp)
- **1980s**: Object-oriented programming
  - Important innovation in software development
  - The concept of data type abstraction is inherited from Simula 67, a language for discrete event simulation with classes but no inheritance

**Programming Language Genealogy**

[Diagram showing the evolution of programming languages from 1960 to 2000, with key dates and languages such as Fortran, Cobol, Algol, Pascal, Modula, Smalltalk, C++, Java, and others, illustrating the development of languages like Fortran 1, Fortran IV, Fortran 57, Fortran 90, and Fortran 95.]

*Note: [http://www.cas.american.edu/~jbarlow/40.521/langs.html](http://www.cas.american.edu/~jbarlow/40.521/langs.html) is a link to a resource of programming languages*
Selected Overview of Programming Languages

- **Fortran** (I, II, IV, 77)
  - Dramatic impact on computing in early days
  - Mainly used for numerical computation
  - No recursion
  - Limited data types (no records and no pointers)
  - Limited type checking
  - Very good compilers are available today
- **Fortran** (90, 95, HPF)
  - Major revisions, eg. recursion, pointers, and records added
  - New control constructs (eg. `while` loop)
  - Extensive set of array operations
  - HPF (High-Performance Fortran) includes parallel constructs
- **Lisp**
  - The original functional language developed by McCarthy as a realization of an abstract machine: Church’s lambda calculus
  - Many dialects, including Common Lisp and Scheme
  - Very powerful for symbolic computation using lists (eg. for artificial intelligence)
  - Implicit memory management (allocate/deallocate) by "garbage collection"
  - Influenced functional programming languages (ML, Miranda, Haskell)

Selected Overview of Programming Languages

- **Algol 68**
  - Large and relatively complex
  - Strong influence on Pascal, C, Ada
- **Cobol**
  - Originally developed by Department of Defense
  - Intended for business data processing
  - Extensive numerical formatting features and decimal number storage
  - Introduced the concept of records and nested selection statements
- **Basic**
  - Intended for interactive use (interpreted)
  - Goals: easy to learn and use for non-science students
  - Visual Basic is a popular dialect
- **PL/I**
  - Designed by IBM
  - First exception handling
  - First pointer data type
  - Poorly designed, too large, too complex

- **Algol 60**
  - The original block-structured language (local variable scopes)
  - First use of Backus-Naur Form (BNF) to formally define grammar
  - All subsequent imperative programming languages are based on it
  - No I/O and no character set, not widely used in US
### Selected Overview of Programming Languages

<table>
<thead>
<tr>
<th>Language</th>
<th>Key Features</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Pascal</strong></td>
<td>Designed for teaching &quot;structured programming&quot;</td>
</tr>
<tr>
<td></td>
<td>Small and simple</td>
</tr>
<tr>
<td><strong>Simula 67</strong></td>
<td>Primarily designed for discrete-event simulation</td>
</tr>
<tr>
<td></td>
<td>Based on Algol 60</td>
</tr>
<tr>
<td></td>
<td>Introduced concept of coroutines (co-executing routines, kind of communicating threads)</td>
</tr>
<tr>
<td></td>
<td>Introduced the class concept for data abstraction</td>
</tr>
<tr>
<td><strong>Ada</strong></td>
<td>Originally intended to be the standard language for all software commissioned by the Department of Defense</td>
</tr>
<tr>
<td></td>
<td>Very large</td>
</tr>
<tr>
<td></td>
<td>Elaborate support for packages, exception handling, generic program units, concurrency</td>
</tr>
<tr>
<td><strong>Ada 95</strong></td>
<td>Support for object-oriented programming</td>
</tr>
<tr>
<td></td>
<td>New concurrency features</td>
</tr>
<tr>
<td><strong>Smalltalk-80</strong></td>
<td>Developed by XEROX PARC</td>
</tr>
<tr>
<td></td>
<td>First full implementation of an object-oriented language</td>
</tr>
<tr>
<td></td>
<td>First design and use of window-based graphical user interfaces (GUIs)</td>
</tr>
<tr>
<td><strong>APL</strong></td>
<td>Intended for interactive use (&quot;throw-away&quot; programming)</td>
</tr>
<tr>
<td></td>
<td>Highly expressive functional language makes programs short, but hard to read</td>
</tr>
<tr>
<td></td>
<td>Many array operations</td>
</tr>
<tr>
<td><strong>Prolog</strong></td>
<td>The most widely used logic programming language</td>
</tr>
<tr>
<td></td>
<td>Non-procedural (declarative: states what you want, not how to get it)</td>
</tr>
<tr>
<td></td>
<td>Based on formal logic</td>
</tr>
<tr>
<td><strong>Haskell</strong></td>
<td>The leading purely functional language, based on Miranda</td>
</tr>
</tbody>
</table>
Selected Overview of Programming Languages

- **C**
  - One of the most successful programming languages
  - Designed for systems programming
  - Powerful set of operators, but weak type checking and no dynamic semantic checks
- **C++**
  - The most successful of several object-oriented successors of C
  - Evolved from C and Simula 67
  - Large and complex, because it supports both procedural and object-oriented programming
- **Java**
  - Developed by Sun
  - Based on C++, but significantly simplified
  - Supports only object-oriented programming
  - Safe language (e.g. no pointers but references, strongly typed, and implicit garbage collection)
  - Truly machine-independent (?)

Note: More about Java

So Why is it That There are so Many Programming Languages?

- **Evolution**
  - This course should give you some insight in what constitutes a good or a bad programming construct for language design. Appendix B of the textbook has a long list of historical mistakes
  - Early 70s: "structured programming" in which goto-based control flow was replaced by high-level constructs such as while loops and case statements
  - Late 80s: nested block structure gave way to object-oriented structures
- **Special Purposes**
  - Many languages were designed for a specific problem domain. For example
    - Scientific applications
    - Business applications
    - Artificial intelligence
    - Systems programming
- **Personal Preference**
  - The strength and variety of personal preference makes it unlikely that anyone will ever develop a universally acceptable programming language
What Makes a Programming Language Successful?

- **Expressive Power**
  - All languages are equally powerful in technical sense
  - Language features have a huge impact on the programmer's ability to read, write, maintain, and analyze programs
  - Abstraction facilities enhance expressive power
- **Ease of Use for Novice**
  - Low learning curve and often interpreted, e.g. Basic and Logo
- **Ease of Implementation**
  - Runs on virtually everything, e.g. Basic, Pascal, and Java
  - Freely available
- **Excellent Compilers**
  - Fortran has extremely good compilers (because it lacks recursion and pointers) and is therefore popular for numerical processing
  - Supporting tools to help the programmer manage very large projects, e.g. Visual C++
- **Economics, Patronage, and Inertia**
  - Powerful sponsor: Cobol, PL/I, Ada
  - Some languages remain widely used long after "better" alternatives because of a huge base of installed software and programmer experience

**Classification of Programming Languages**

- **Declarative** ("what the computer is to do")
  - Functional (e.g. Lisp, Scheme, ML, Miranda, Haskell)
  - Dataflow
  - Logic (e.g. Prolog, Excel)
- **Imperative** ("how the computer should do it")
  - Procedural or "von Neumann" (e.g. Fortran, Pascal, Basic)
  - Object-oriented (e.g. Smalltalk-80, C++, Java)