CS321 Languages and Compiler Design I
Fall 2010
Lecture 3
How can we judge or compare languages?

Expressiveness

- Technically not interesting; nearly all languages are “Turing-complete.”

Appropriateness to domain

- Scientific (numerical) computing
- Business applications
- Artificial intelligence
- Systems programming
- etc.

High-level goals for code

- Easily readable
- Easily writable
- Maintainable
- Efficient

Goals for languages

- Simplicity
- Uniformity (orthogonality)
- Modularity
- Clean syntax
- Maximizes explicit structure
- Clear execution model
- Efficient implementation model
CHOOSING A PROGRAMMING LANGUAGE

Costs affected by programming language choice

Execution speed (& space)
Development time
• Program writing
• Compilation, testing, debugging
• (Training)
Maintenance time
• Program reading

Factors affecting programming language choice

Costs (as above)
Availability of implementations
Availability of trained programmers (should this matter?)
Politics
Inertia
Domain: Numerical computation (still widely used)

Features:

- Arithmetic expressions (evaluated using stack)
- Statements
- Bounded arrays
- Iterative control structures
- Subroutines (no recursion; call-by-reference; separate compilation (in FORTRAN II))
- Common blocks (and EQUIVALENCE declarations)
- I/O using FORMAT directives

Implementation model:

- Fixed run-time storage requirements
- Optimization of numerical computations
Domain: Numerical computation

Features:

- Carefully defined by “report”; syntax defined with BNF
- Block structure (stack-based implementation)
- Recursive subroutines
- Explicit type declarations
- Scope rules and dynamic lifetimes
- Relational & boolean expressions
- Call-by-value & call-by-name
- Dynamic Array Bounds

Never widely used, but very influential on later languages.

“An improvement on nearly all its successors.” – Hoare
**COBOL 1959-61 DOD-LED COMMITTEE**

Domain: Business data processing

Features:

- Separate data description
- Record data structures
- File description/ manipulation
- English-language-like syntax (“Syntactic sugar”)
- Early standardization

Many, many lines of code are probably still in wide use.
Pascal

Domain: General-purpose programming, education.

- Simplicity of language and implementation
- Rich type definition facility
- Structured programming methodology
- Suitable for proving programs correct

Modula-2

- Modules for abstraction
- Systems programming facilities
- Procedure types

Oberon, Oberon-2, Modula-3

ca. 1990
Domain: Systems Programming; hacking of all kinds.
Implementation language for UNIX kernel and utilities

- Rich set of operators
- Terse syntax
- Easy machine access

Very successful; widely used in engineering and education
Standardized as **ANSI C**
Domain: Everything, but especially embedded systems.

Features:

- Focus on reliability, safety.
- Real-time control and multiprocessing.
- Programming support environments.
- Very large and verbose language.

Was mandated for much DOD work, but no more. Ada95 added object-oriented features.
OBJECT-ORIENTED LANGUAGES

Simula-67 1967 Kristen Nygaard and Ole-Johan Dahl
- Discrete event simulations

Smalltalk 1972- Alan Kay (Xerox PARC)
- Graphical user interfaces
- Everything is an object
- Unusual message-sending syntax

C++ 1980- Bjarne Stroustrup
- Extended version of C.
- Vehicle for main-stream adoption of OOP.
- Direct support for abstract data types
- Large and very complex language
- Used very widely.
Java

- Cut-down, cleaned-up version of C++.
- Initially hyped for network applications.
- Automatic heap storage management (garbage collection).
- Type-safety and runtime memory security.
- Portable runtime environment (Java Virtual Machine).

C#

- Very similar to Java (though supposedly independent).
- Common Language Runtime environment supports multiple source languages.
LISP

1959-60  John McCarthy (MIT)

Domain: Artificial intelligence; symbolic computing

- List processing
- “First-class” functions
- Extremely simple program syntax; programs manipulate programs
- Dynamic typing

Many variants, including **Common Lisp, Scheme**; also related to

**Standard ML, Caml**  

- Static but flexible typing
- Rich, orthogonal type system
- Module support

**Haskell**  
1987-  Academic Committee

- Lazy (demand-driven) evaluation
- No side effects
SCRIPTING LANGUAGES

Domain: Glueing components, system admin, HTML generation, etc.

Perl 1987- Larry Wall
- C-like syntax
- Static typing
- Dynamically-sized and associative arrays

JavaScript 1995- Netscape, Sun
- Browser-side HTML generation, input validation
- Dynamically-typed, limited OOP support

PHP 1994- Rasmus Lersdorf
- Server-side HTML generation, DBMS integration

Python 1990- Guido von Rossum
- Rich built-in support for lists, tuples, dictionaries
- OOP support
- Interpreter can be extended with compiled libraries

Ruby 1993- Yukihiro Matsumoto
- Pure OOP