CS321 F’04 Lecture Notes
Lecture 3
Evaluating Programming Languages

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
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
FORTRAN 1954-58 John Backus (IBM)

Domain: Numerical computation

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

Still used very widely: FORTRAN IV, FORTRAN 77, FORTRAN 90, HPF
ALGOL 60

1957-60 Committee (incl. Backus, McCarthy, Naur)

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 in US; somewhat used in Europe.

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

Still used very widely.

Ada 1977-83  DOD-sponsored committee

(Ichbiah)

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.
Pascal Family 1971- Niklaus Wirth

Pascal 1971

Domain: General-purpose programming, education.

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

Modula-2 1979-81

- Modules for abstraction
- Systems programming facilities
- Procedure types

Oberon, Oberon-2 1988-90, 1992

- Further simplification!
- Addition of object-oriented features.

Modula-3 1988-89

- Separate development by DEC SRC.
C  
1972-74  Dennis Ritchie (Bell Labs)

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

C++  
1980-  Bjarne Stroustrup

Domain: As C.

- Extended version of C.
- Direct support for abstract data types
- Object-oriented programming
- Large and very complex language

Used very widely.
Java 1995- Arnold & Gosling (Sun)

Domain: Internet applet programming; as C++.

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

Wildly hyped for network applications; may or may not take over C++ territory.

C# 2001- Microsoft

- Very similar to Java (though supposedly independent).
- Common Language Runtime environment intended to support multiple source languages.
LISP and Functional Languages

LISP 1959-60  John McCarthy (MIT)

Domain: Artificial intelligence; symbolic computing

Features:

• List processing

• “First-class” functions

• Extremely simple program syntax; programs can easily manipulate programs

• Dynamic typing

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


• Static but flexible typing

• Rich, orthogonal type system

• Module support

Haskell 1987- Academic Committee

• Lazy (demand-driven) evaluation

• Pure functions
Visual Basic

Domain: Customizing and extending MS Windows-based office and COM applications.

- Very loosely based on BASIC language developed in early ’60’s.
- Programs (especially user-interfaces) usually built using interactive visual program development environment.
- Often used to “glue” together existing code components.
- Supports rapid, “one-off” prototyping and large-scale system development.
- Conventional procedural language.
- Dynamic typing.
- Supports COM objects.

Used very widely in Microsoft environments.