Concepts of Programming Languages

Robert W. Sebesta

Chapter 1

Preliminaries

Reasons for studying the underlying concepts of programming languages…

The Study of Programming Languages

- Increases our ability to express ideas through programs
  - **Thinking Through *Language**** *www.yale.edu/cogdevlab/aarticles/bloom%20and%20keil.pdf*
- Enables us to choose the most appropriate language for a project based on its strengths and weaknesses.
  - http://www.wired.com/wiredenterprise/2012/06/beard-gallery/?pid=87
- Facilitates the learning of new languages.
  - TIOBE Programming Community http://www.tiobe.com/tiobe_index/index.htm
- Helps us to be better code writers and debuggers by giving us a better understanding of the implementation level of a program language.
- Helps us make better use of languages that are we are already using.
- Drives the advancement of computing
  - Example: ALGOL 60 vs. Fortran
  - http://softtalkblog.com/2012/04/23/istep-2012-why-fortran/

Classifying Languages by Use

Programming Domains

- The design and evaluation of a particular language is highly dependent on the domain in which it is to be used.
  - Scientific Applications - Large numbers of floating point computations; use of arrays
    - Fortran, ALGOL 60, MatLab, Numerical Python, etc…
  - Business Applications - Produce reports, use decimal numbers and characters
    - COBOL, RPG, etc..
  - Artificial Intelligence - Symbols rather than numbers are manipulated; use of linked lists
    - LISP, Prolog, Scheme, etc…
  - Systems Programming - Need efficiency because of continuous use
    - IBM's PL/S, Digital's BLISS, UNIX's C, etc…
  - Browser Software - Eclectic collection of languages from markup (e.g., XHTML) to scripting to general-purpose
    - PHP, JavaScript, Java Applets, etc…

Considerations when Choosing or Designing a Language

Language Evaluation Criteria

- **Readability**: the ease with which programs can be read and understood.
- **Writability**: expressivity, simplicity, orthogonality, support for abstraction...
- **Reliability**: conformance to specifications (i.e., performs to its specifications) support for type checking, exception handling, aliasing,
  - readable, maintainable, writable
  - “A language that does not support “natural” ways of expressing an algorithm will require the use of “unnatural” approaches, and hence reduced reliability.”
- **Cost**: - training, coding, compiler & execution, implementation system, legal, maintenance…
- **Other:** portability, generality, well-definedness…

Overlapping of criteria…

Table 1.1 Language evaluation criteria and the characteristics that affect them.

	CRITERIA		
Characteristic	READABILITY	WRITABILITY	RELIABILITY
Simplicity/orthogonality	•	•	•
Control structures	•	•	•
Data types and structures	•	•	•
Syntax design	•	•	•
Support for abstraction		•	•
Expressivity		•	•
Type checking			•
Exception handling			•
Restricted aliasing			•

Tradeoffs

Reliability vs. cost of execution

Example: Java demands all references to array elements be checked for proper indexing, which leads to increased execution costs

Readability vs. Writability

Example: APL provides many powerful operators (and a large number of new symbols), allowing complex computations to be written in a compact program but at the cost of poor readability

Writability (flexibility) vs. reliability

Example: C++ pointers are powerful and very flexible but are unreliable

Zoom in on aspects of Readability

Evaluation Criteria: Readability

the ease with which programs can be read and understood

- Overall simplicity - A *manageable* set of features and constructs
  - Minimal feature multiplicity – Example: count++ etc.
  - Minimal operator overloading – Example: + symbol
  - Assembly languages vs. HLLs
- Orthogonality
  - A relatively small set of primitive constructs can be combined in a relatively small number of ways where every possible combination is legal.
  - For Example: Add two 32 bit integers and replace one of the two with the sum.
    - IBM mainframe two instructions required–
      - A Reg1, memory_cell and AR Reg1, Reg2
    - VAX one instruction –
      - ADDL operand_1, operand_2, where either or both operands can be a register or a memory cell.
  - C – structs can be returned from functions but arrays cannot; the parameter passing mechanism is different for arrays.
    - Disadvantage – computational complexity
  - Functional languages offer balance by using a single construct – the function call

Evaluation Criteria: Readability

the ease with which programs can be read and understood

- Control statements
  - The presence of well-known **and reliable** control structures
  - Research of the 70s led to the desire for language constructs that made “goto-less” programming possible.
    - “A program that can be read from top to bottom is much easier to understand than a program that requires the reader to jump from one statement to some nonadjacent statement in order to follow the order of execution.” Sebesta .
    - For example, what output is generated by this code segment?

```
inum1 = 1;
loop1:
   if(inum1 > 10) goto end;
   inum2 = 1;
loop2:
   if (inum2 > 10) goto next
    print (inum1 + " * " + inum2 + " = " + inum1 * inum2);
    inum2++;
    goto loop2;
next:
    inum1++;;
   goto loop1;
}
end:
```

Evaluation Criteria: Readability

the ease with which programs can be read and understood

- Equivalent code in java for the Nested loop

```
inum1 = 1;
while (inum1 <= 10)
{
   inum2 = 1;
   while (inum2 <= 10)
   {
          print (inum1 + " * " + inum2 + " = " + inum1 * inum2);
          inum2++;
   }
   inum1++;
}
```

Evaluation Criteria: Readability

the ease with which programs can be read and understood

- Data types and structures
  - Adequate predefined data types, pointers
    - Example: numeric types vs. Boolean type for indicator variables.
  - Adequate structures, such as arrays, pointers
  - The presence of adequate facilities for defining programmer-defined data structures, such as records
    - Example: Using a record structure or a class vs. parallel arrays

```
Record
    Character (Len=30):: Name
    Integer:: Age
    Integer:: Employee_Number
    Real:: Salary
End_Record
```

    - In contrast to:

```
    Character (Len=30):: Name(100)
    Integer:: Age (100)
    Integer:: Employee_Number (100)
    Real:: Salary (100)
```

Evaluation Criteria: Readability

the ease with which programs can be read and understood

- Syntax considerations
  - Identifier forms: flexible composition
    - Example: Fortran 77 limits identifiers to 6 characters
    - Example: Original ANSI BASIC in 1978 limited identifiers to a single letter or a single letter followed by a single digit
  - Special words
    - Should denote usage such as if, while, or class.
    - Methods of forming compound statements: braces vs. end if and end loop.
      - Fewer reserved words vs. more readable code.
    - Reserved or not – Example: Fortran 95 allowed Do and End as legal variable names in addition to their keyword meanings.
  - Form and meaning
    - Self-descriptive constructs – “Semantics should follow directly from syntax..” Sebesta
    - Conflict when two language constructs are similar, but have different meanings depending on context.
      - Example: In C the reserved word static has different meaning depending on the context
        - If applied to variables in methods vs. if applied to variables outside of all methods
      - Example: Unix shell command grep –
        - root is in the ed editor command g/regular_expression/p

The Influence of Computer Architecture on Language Design

The Influence of Computer Architecture on Language Design

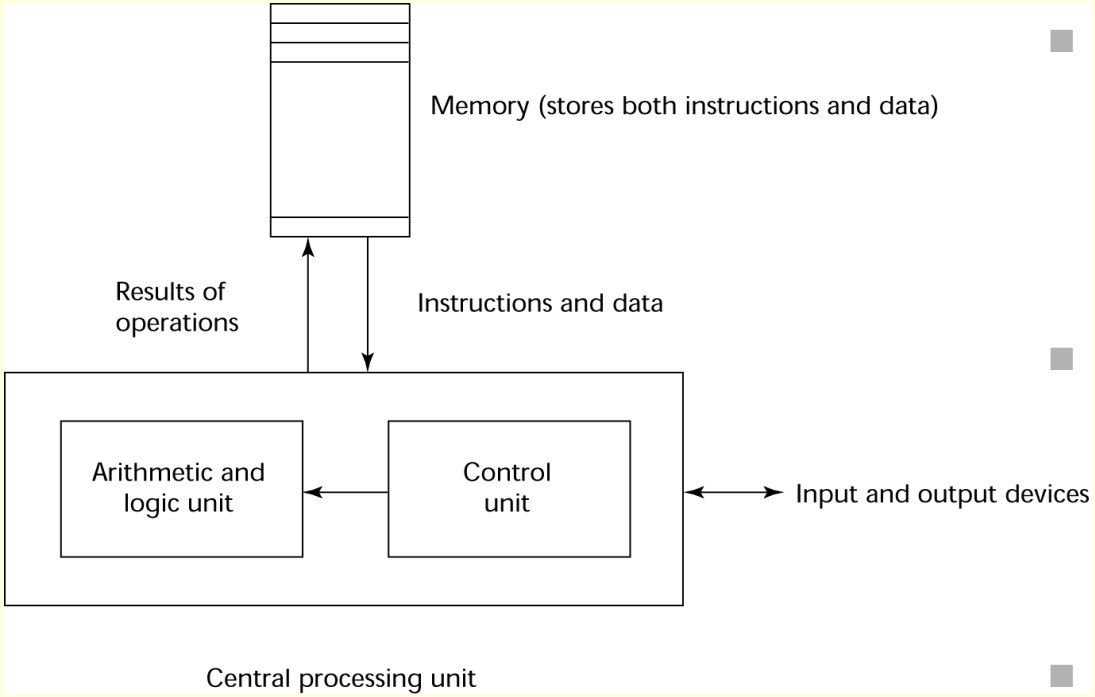
The Influence of Computer Architecture on Language Design

The von Neumann Architecture

- Fetch-execute-cycle (on a von Neumann architecture computer)

```
initialize the program counter
repeat forever
  fetch the instruction pointed by the
  counter
  increment the counter
  decode the instruction
  execute the instruction
   store the result
end repeat
```

The von Neumann Architecture



- Connection speed between a computer's memory and its processor determines the speed of a computer
- Program instructions often can be executed much faster than the speed of the connection resulting in a *bottleneck*
- Known as the *von Neumann bottleneck*; it is the primary limiting factor in the speed of computer programming.

Computer Architecture Influence

- Von Neumann computer architecture is the basis for imperative languages.
  - Data and programs are stored in memory
    - Variables model memory cells
  - Memory is separate from the CPU
  - Instructions and data are piped from memory to CPU
    - Assignment statements model piping
  - Ineffective for functional (applicative) languages, such as Scheme, where computation occurs by applying functions. "Can Programming be Liberated from the von Neumann Style? A Functional Style and Its Algebra of Programs." by John Backus. 1978 Comm. ACM, Vol. 21, No. 8, pp. 613-641.

The Influence of Programming Methodologies on Language Design

Programming Methodologies Influence

- 50s and early 60s:
  - Simple applications
  - Concerns were about machine efficiency
- Late 60s, early 70s:
  - Programming problems more complex.
  - Cost of hardware reduced.
  - Cost of software development increased.
    - **People** efficiency became important.
      - Structured Programming Movement led to Top-Down-Stepwise Refinement. Readability, Better control structures (“gotoless” programming)
- Late 70s: shift from procedure-oriented to data-oriented design
  - Data abstraction to encapsulate processing with data
    - SIMULA 67
- Middle 80s: OOP
  - Data abstraction plus Inheritance and Dynamic method binding (polymorphism)
    - Smalltalk, Ada 95, Java, C++.
- More recently procedure oriented programming applied to concurrency
  - Ada, Java, C# have capabilities to control concurrent program units.

Other Effects on Language Design

- Difficulty of implementing the various constructs and features.
- Politics
- Economics
- Advances in research

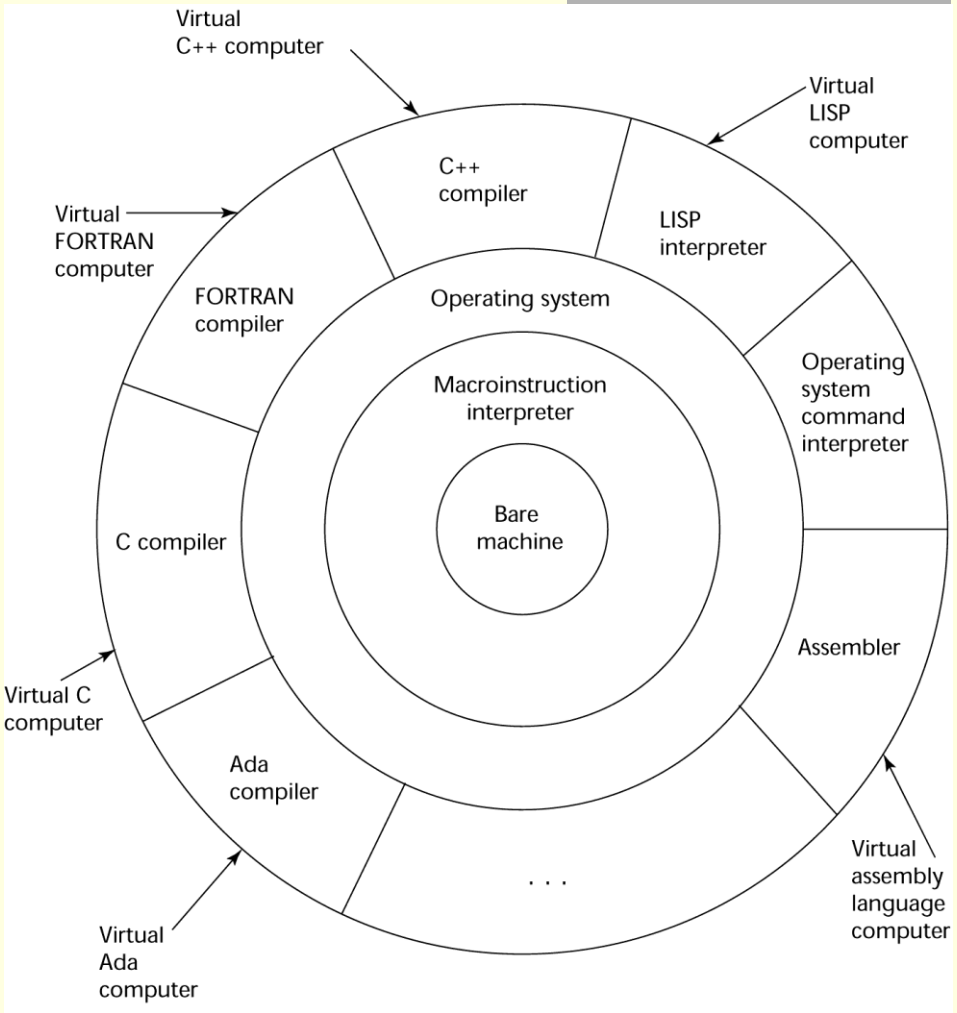
Types of Languages

Language Types

- Imperative
  - Central features are variables, assignment statements, and iteration
  - Include languages that support object-oriented programming
  - Include scripting and visual languages
  - Examples: C, Java, Perl, JavaScript, Visual BASIC .NET, C++
- Functional
  - Main means of making computations is by applying functions to given parameters
  - Examples: LISP, Scheme
- Logic
  - Rule-based (rules are specified in no particular order)
  - Example: Prolog
- Markup/programming hybrid
  - Markup languages extended to support some programming
  - Examples: JSTL, XSLT

Layered View of Computer

The operating system and language implementation are layered over machine interface of a computer.



Methods of Implementation

- Compilation
- Pure Interpretation
- Hybrid Interpretation

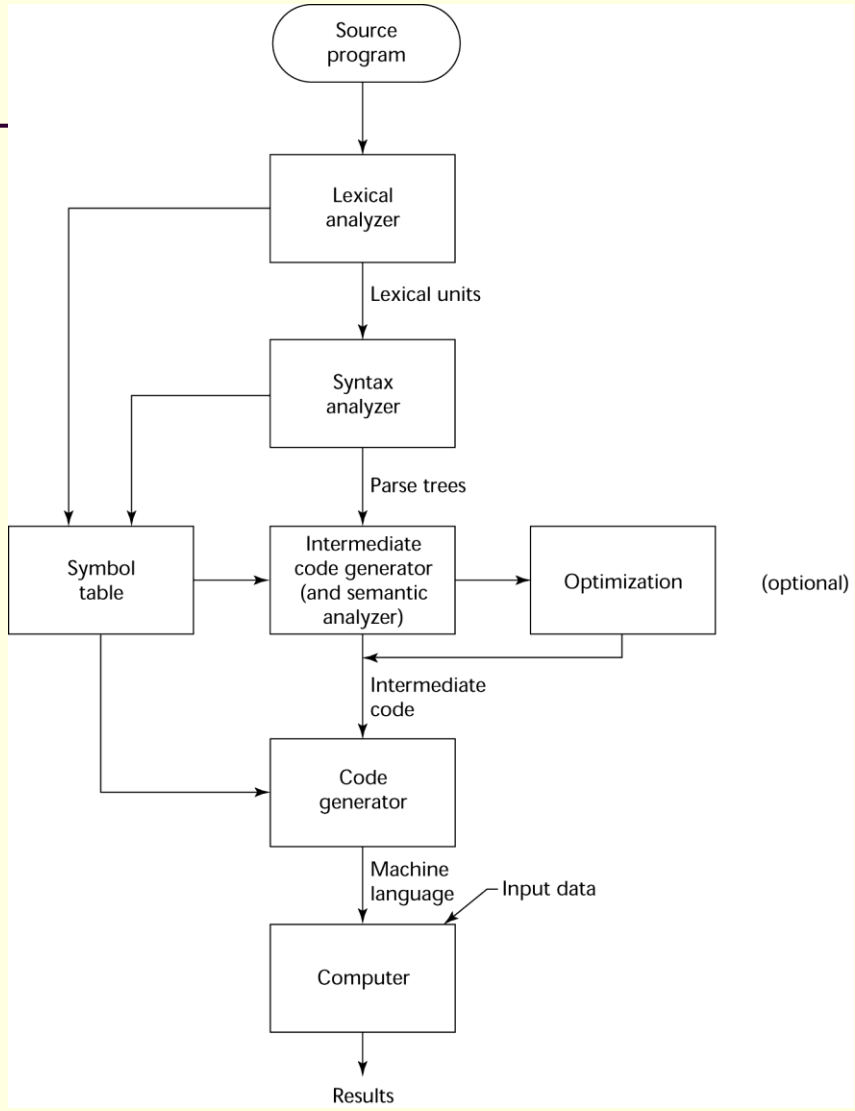
Compilation Process Phases

Source code is translated into equivalent machine code as a unit and stored into a file that has to be executed in a separate step.

Yields faster execution times;

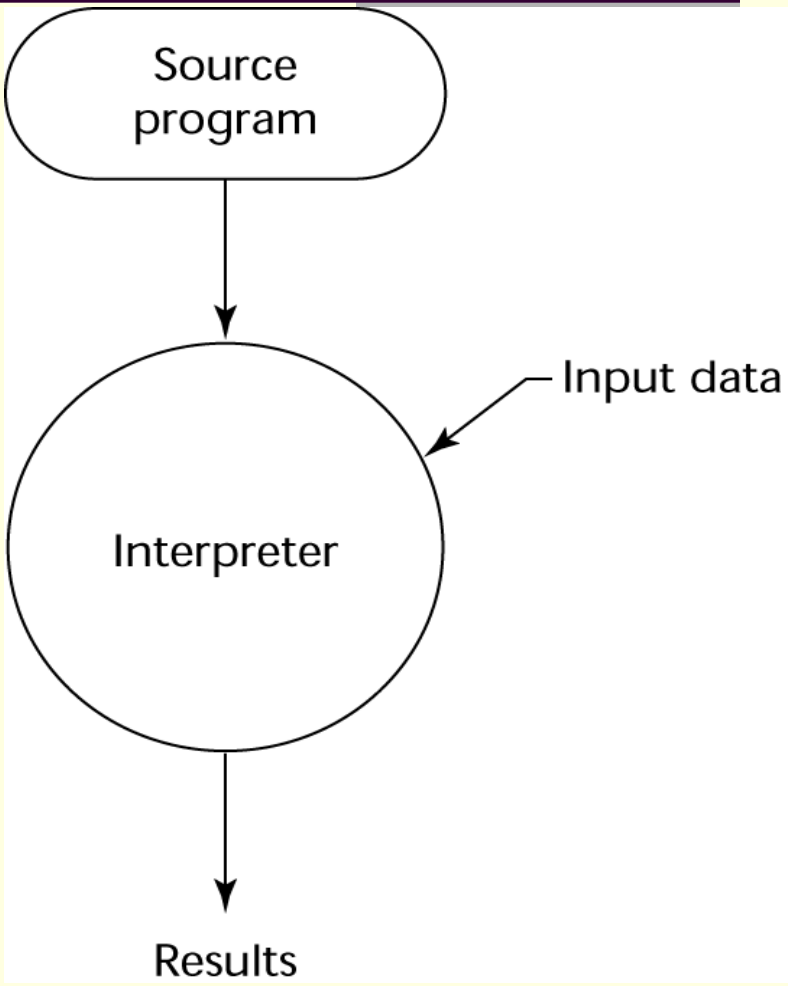
Examples: C/C++, Pascal, COBOL, Ada

- Lexical analysis:
  - extracts a sequence of tokens (lexical units) from source code
  - The symbol table contains the definitions of the identifiers
- Syntax analysis (i.e. parsing) :
  - transforms tokens (lexical units) into *parse trees* which represent the syntactic structure of program.
- Semantics analysis:
  - generate intermediate code
- Code generation:
  - machine code is generated



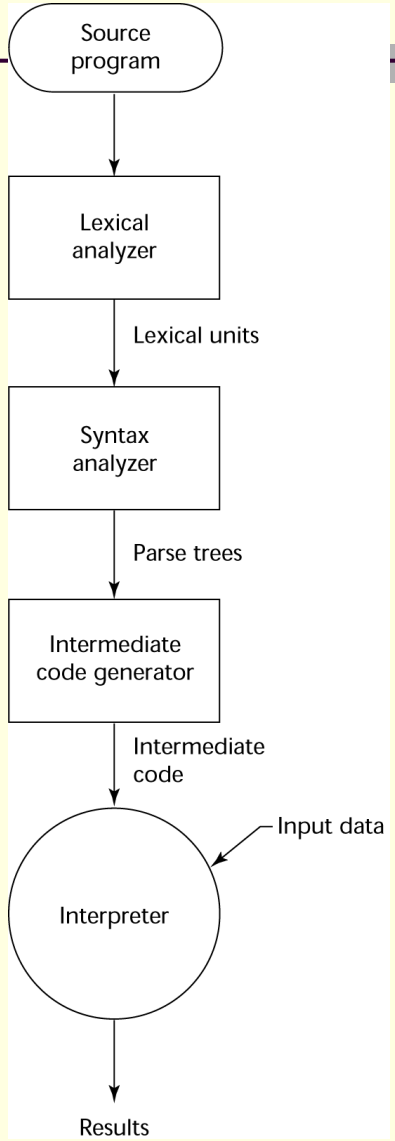
Pure Interpretation Process

- Pure interpretation – source code is translated to machine code and executed immediately.
  - Advantage – run-time errors can refer to source level units such as array index out of bounds errors
  - Disadvantage
    - 10 to 100 times slower execution time
    - Often requires more space
  - Significant comeback with some Web scripting languages (e.g., JavaScript, PHP)



Hybrid Implementation Process

- Hybrid interpretation - A compromise between compilers and pure interpreters
  - A high-level language program is translated to an intermediate language that allows easy interpretation
  - Faster than pure interpretation since source language statements decoded only once.
  - Examples
    - Perl programs are partially compiled to detect errors before interpretation
    - Just-in-Time system compiles intermediate language methods into machine code when they are initially called. This machine code is kept so that if they are called again the code does not have to be re-interpreted. - Java



Additional Compilation Terminologies

- **Linking and loading**:
  - the process of collecting system program units and linking them to a user program
- **Load module** (executable image):
  - the user and system code together
- **Preprocessor**
  - Preprocessor macros (instructions) are commonly used to specify that code from another file is to be included
  - A preprocessor processes a program immediately before the program is compiled to expand embedded preprocessor macros
  - A well-known example: C preprocessor expands `#include, #define`, and similar macros

Programming Environments

- The collection of tools used in software development
- Simple – file system, text editor, compiler, interpreter or linker.
- Extensive – rich set of tools
  - Borland JBuilder
    - An integrated development environment for Java
  - Microsoft Visual Studio.NET
    - A large, complex visual environment
    - Used to program in C#, Visual BASIC.NET, Jscript, J#, and C++

Summary

- The study of programming languages is valuable for a number of reasons:
  - Increase our capacity to use different constructs
  - Enable us to choose languages more intelligently
  - Makes learning new languages easier
- Most important criteria for evaluating programming languages include:
  - Readability, writability, reliability, cost
- Major influences on language design have been machine architecture and software development methodologies
- The major methods of implementing programming languages are: compilation, pure interpretation, and hybrid implementation

supplements

www.aw.com/sebesta - This site contains mini-manuals (approximately 100-page tutorials) on a handful of languages. Currently the site includes manuals for C++, C, Java, and Smalltalk.

Language Processor Availability-Processors for and information about some of the programming languages discussed in this book can be found at the following Web sites:

C, C++, Fortran, and Ada	*gcc.gnu.org*
C# and F#	*microsoft.com*
Java	*java.sun.com*
Haskell	*haskell.org*
Lua	*www.lua.org*
Scheme	*www.plt-scheme.org/software/drscheme*
Perl	*www.perl.com*
Python	*www.python.org*
Ruby	*www.ruby-lang.org*

- JavaScript is included in virtually all browsers; PHP is included in virtually all Web servers.