Chapter 1

Continued

Evaluation Criteria: Readability

- Overall simplicity
  - A manageable set of features and constructs
  - Minimal feature multiplicity
  - Minimal operator overloading
- Orthogonality
  - A relatively small set of primitive constructs can be combined in a relatively small number of ways
  - Every possible combination is legal
- Data types
  - Adequate predefined data types
- Syntax considerations
  - Identifier forms: flexible composition
  - Special words and methods of forming compound statements
  - Form and meaning: self-descriptive constructs, meaningful keywords

  - *boolean A = ((0xa | 0xb) > 0x9) ? !c : !d;*

Feature Multiplicity

```
count = count + 1
count += 1
count++
++count
```

```
int count = 6;
System.out.println(++count);
System.out.println(count++);
```

Feature Multiplicity (Orthogonality)

- Which of these two are better, easier to read, write and more realiable?

```
for(int i = 0; i<array.length; i++)
```

```
for(int i : array)
```

Operator Overloading (Orthogonality)

```
void operator = (const Distance &D)
{
        //Stuff
}
```

https://www.tutorialspoint.com/cplusplus/assignment_operators_overloading.htm

Context Dependence (Orthogonality)

- Pass by Reference and pass by value (arrays are different than primitivies, and objects are different)

A + B

Data Types (Orthogonality)

- flag = 1 //(0 is false in C, everything else is true)

- flag = true // because we have Boolean data types

Syntax design

- Think about Java:
  - For
  - While
  - Do/while
  - If
  - If/else
  - Switch
- Now Python

Evaluation Criteria: Writability

- Simplicity and orthogonality
  - Few constructs, a small number of primitives, a small set of rules for combining them
  - Too much orthogonality can be a detriment to writeability, Errors can go undetected when any combo of primitivies is allowed.

- Support for abstraction
  - The ability to define and use complex structures or operations in ways that allow details to be ignored

- Expressivity
  - A set of relatively convenient ways of specifying operations
  - Strength and number of operators and predefined functions

Evaluation Criteria: Reliability

- Type checking
  - Testing for type errors, runtime is expensive(Python), compile time is desirable.
  - C does prototyping now.
- Exception handling (Python???? Java????, C?????)
  - Intercept run-time errors and take corrective measures, C++, Java, C#
  - None in C
- Aliasing ---- my rants on understanding references
  - Presence of two or more distinct referencing methods for the same memory location
- Readability and writability
  - A language that does not support “natural” ways of expressing an algorithm will require the use of “unnatural” approaches, and hence reduced reliability

Evaluation Criteria: Cost (a function of below)

- Training programmers to use the language
  - 4 year degree????
- Writing programs (closeness to particular applications)
  - Constantly looking up API
- Executing programs
  - Cost has gone way down with CPU advancement
  - Optimization (Java vs C)
- Reliability: poor reliability leads to high costs
- Maintaining programs
  - (Become much more important)

Evaluation Criteria: Others

- Portability
  - The ease with which programs can be moved from one implementation to another
    - Tools such as Unity or Xamirin or even Mono
    - Windows historical dominance of 95% of market, no portability.
- Generality
  - The applicability to a wide range of applications
- Well-definedness
  - The completeness and precision of the language's official definition

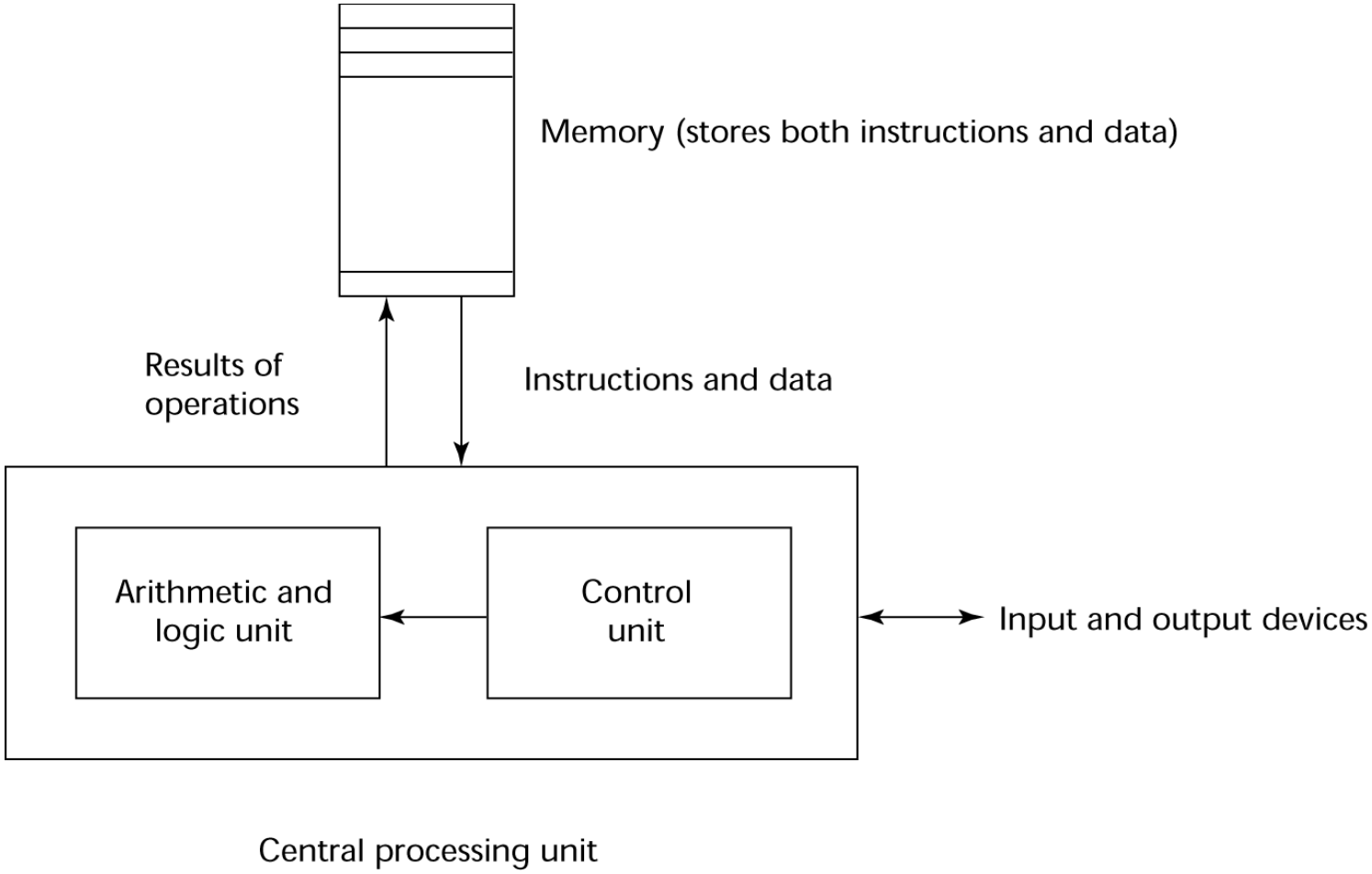
Influences on Language Design

- Computer Architecture
  - Languages are developed around the prevalent computer architecture, known as the *von Neumann* architecture
- Program Design Methodologies
  - New software development methodologies (e.g., object-oriented software development) led to new programming paradigms and by extension, new programming languages

Computer Architecture Influence

- Well-known computer architecture: Von Neumann
- Imperative languages, most dominant, because of von Neumann computers
  - Data and programs stored in memory
  - Memory is separate from CPU
  - Instructions and data are piped from memory to CPU
  - Basis for imperative languages
    - Variables model memory cells
    - Assignment statements model piping
    - Iteration is efficient

The von Neumann Architecture



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
end repeat
```

Cache

- Cache speeds things up
- Language results on cache

Programming Methodologies Influences

- 1950s and early 1960s: Simple applications; worry about machine efficiency
- Late 1960s: People efficiency became important; readability, better control structures
  - structured programming
  - top-down design and step-wise refinement
- Late 1970s: Process-oriented to data-oriented
  - data abstraction
- Middle 1980s: Object-oriented programming
  - Data abstraction + inheritance + polymorphism
  - Reuseable code

Language Categories

- Imperative
  - Central features are variables, assignment statements, and iteration
  - Include languages that support object-oriented programming
  - Include scripting languages
  - Include the 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, ML, F#
- 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

Language Design Trade-Offs

- 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
  - *A* = (*a | b == False*) ? !c : *d*
- Writability (flexibility) vs. reliability
  - Example: C++ pointers are powerful and very flexible but are unreliable

Implementation Methods

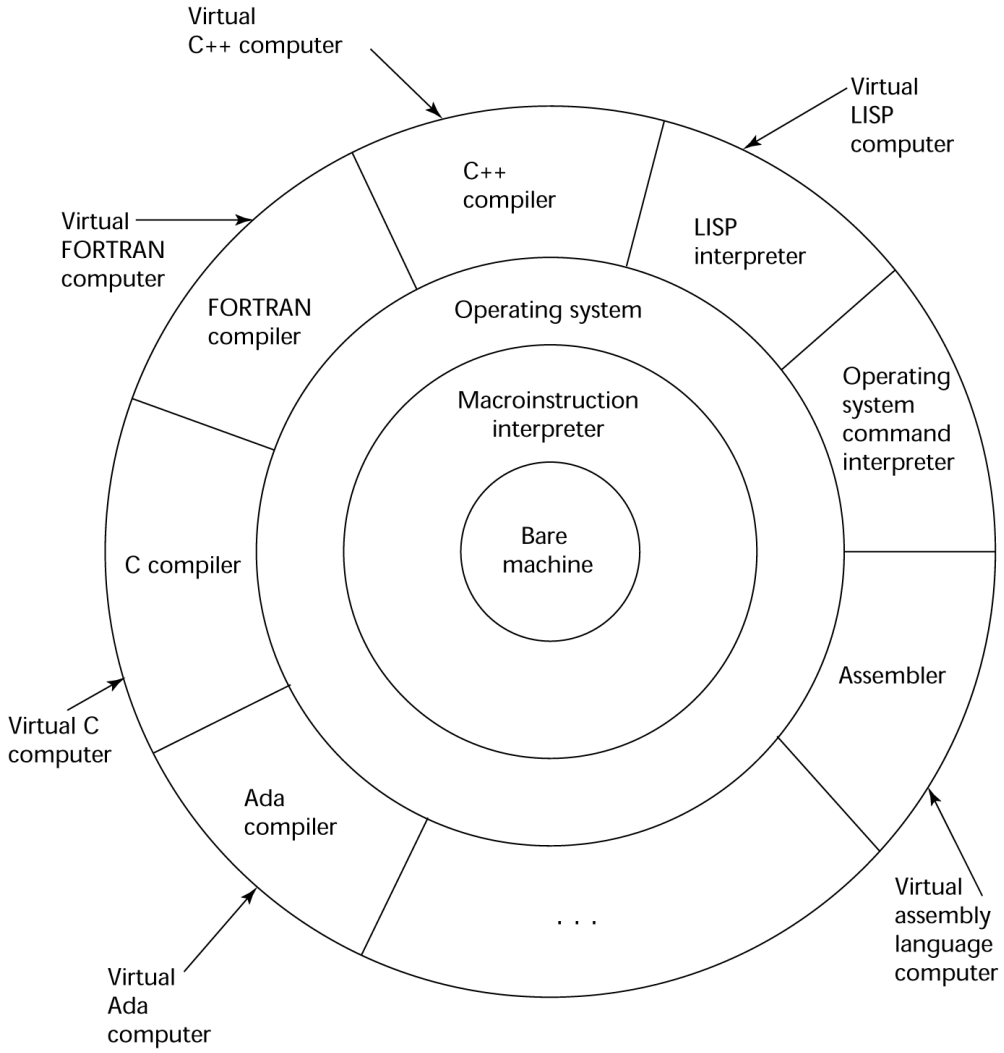
- Compilation
  - Programs are translated into machine language; includes JIT systems
  - Use: Large commercial applications

- Pure Interpretation
  - Programs are interpreted by another program known as an interpreter
  - Use: Small programs or when efficiency is not an issue

- Hybrid Implementation Systems
  - A compromise between compilers and pure interpreters
  - Use: Small and medium systems when efficiency is not the first concern

Layered View of Computer

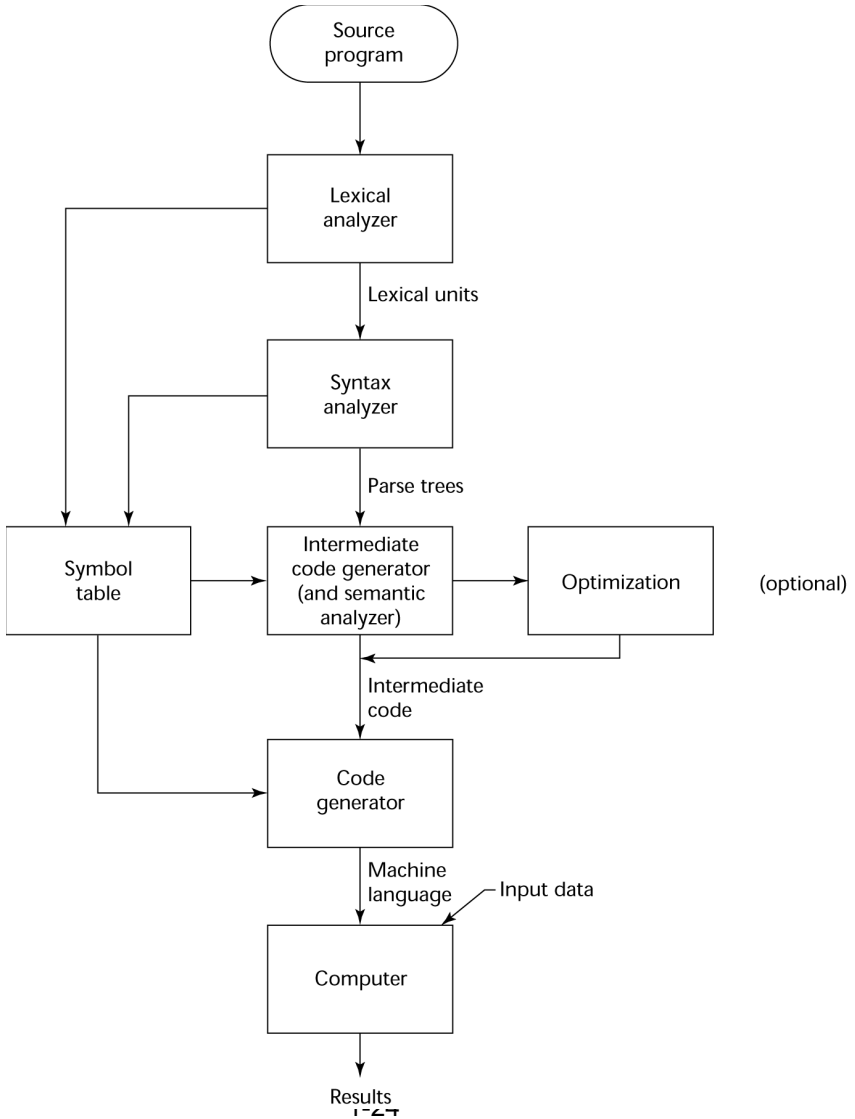
The operating system and language implementation are layered over machine interface of a computer



Compilation

- Translate high-level program (source language) into machine code (machine language)
- Slow translation, fast execution
- Compilation process has several phases:
  - lexical analysis: converts characters in the source program into lexical units
  - syntax analysis: transforms lexical units into *parse trees* which represent the syntactic structure of program
  - Semantics analysis: generate intermediate code
  - code generation: machine code is generated

The Compilation Process



Additional Compilation Terminologies

- **Load module** (executable image): the user and system code together
- **Linking and loading**: the process of collecting system program units and linking them to a user program

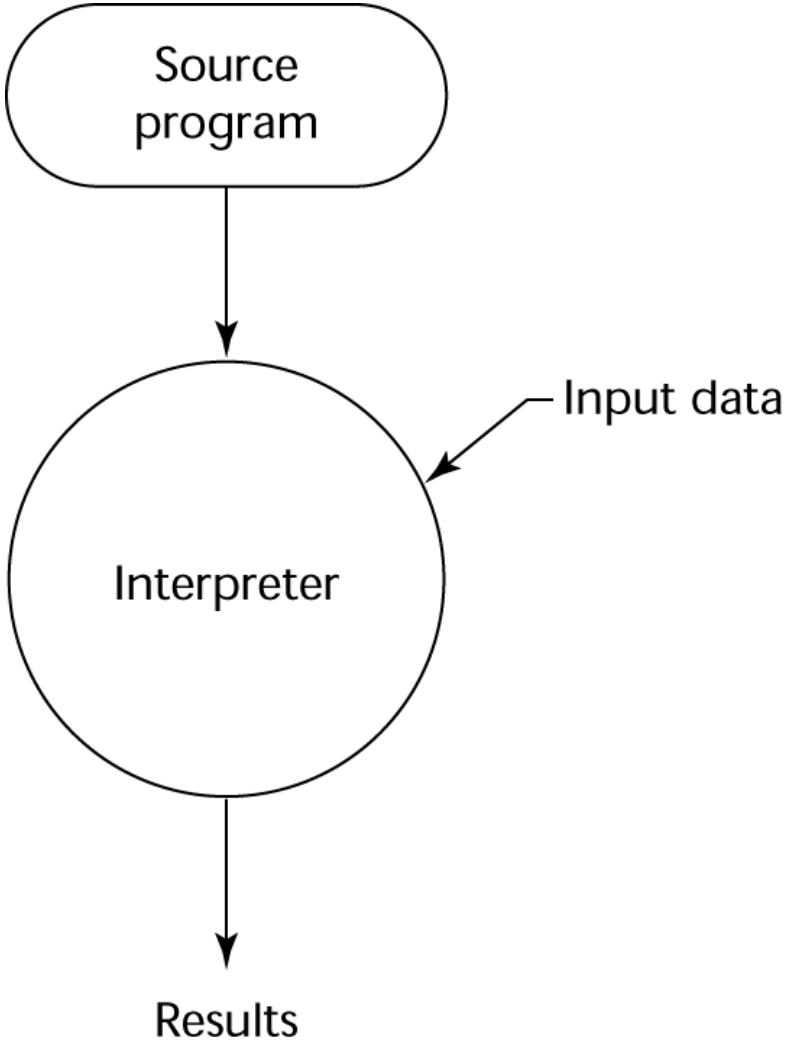
Von Neumann Bottleneck

- 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; the connection speed thus results in a *bottleneck*
- Known as the *von Neumann bottleneck*; it is the primary limiting factor in the speed of computers

Pure Interpretation

- No translation
- Easier implementation of programs (run-time errors can easily and immediately be displayed)
- Slower execution (10 to 100 times slower than compiled programs)
- Often requires more space
- Now rare for traditional high-level languages
- Significant comeback with some Web scripting languages (e.g., JavaScript, PHP)

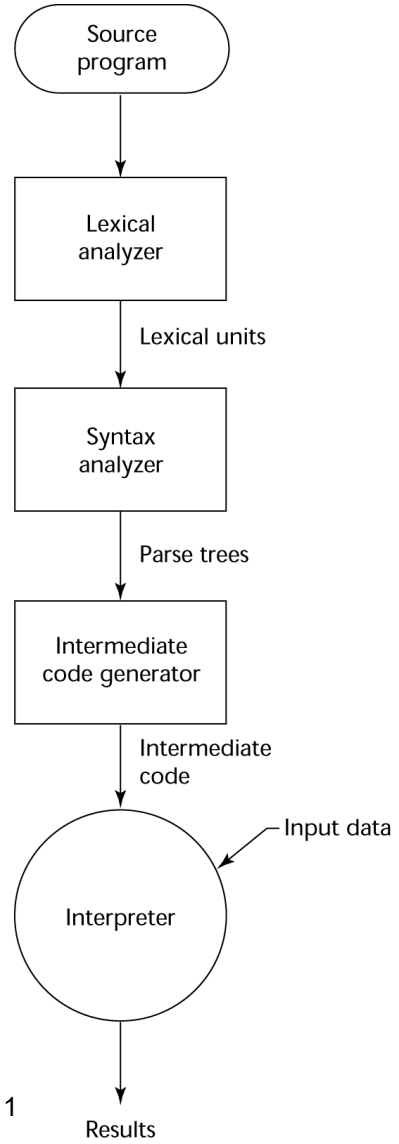
Pure Interpretation Process



Hybrid Implementation Systems

- 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
- Examples
  - Perl programs are partially compiled to detect errors before interpretation
  - Initial implementations of Java were hybrid; the intermediate form, *byte code*, provides portability to any machine that has a byte code interpreter and a run-time system (together, these are called *Java Virtual Machine*)

Hybrid Implementation Process



Just-in-Time Implementation Systems

- Initially translate programs to an intermediate language
- Then compile the intermediate language of the subprograms into machine code when they are called
- Machine code version is kept for subsequent calls
- JIT systems are widely used for Java programs
- .NET languages are implemented with a JIT system
- In essence, JIT systems are delayed compilers

Preprocessors

- 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

- A collection of tools used in software development
- UNIX
  - An older operating system and tool collection
  - Nowadays often used through a GUI (e.g., CDE, KDE, or GNOME) that runs on top of UNIX
- Microsoft Visual Studio.NET
  - A large, complex visual environment
- Used to build Web applications and non-Web applications in any .NET language
- NetBeans
  - Related to Visual Studio .NET, except for applications in Java

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