CS 317: Concepts of programming languages

By Ensaf Hussein

Assistant professor,

Computer Science Department

Introduction

Lecture 1

Chapter 1 Topics

- ❑ ~~Reasons for Studying Concepts of Programming Languages~~
- ❑ ~~Programming Domains~~
- ❑ ~~Language Evaluation Criteria~~
- ❑ Influences on Language Design
- ❑ Language Categories
- ❑ Language Design Trade-Offs
- ❑ Implementation Methods
- ❑ Programming Environments

Influences on Language Design

Other factors have had a strong influence on programming language design:

- Computer Architecture

Languages are developed around the prevalent computer architecture, known as the *von Neumann* architecture

- Programming Methodologies

  New software development methodologies (e.g., object-oriented software development) led to new programming paradigms and by extension, new programming languages

Copyright © 2009 Addison-Wesley. All rights reserved.

Computer Architecture Influence

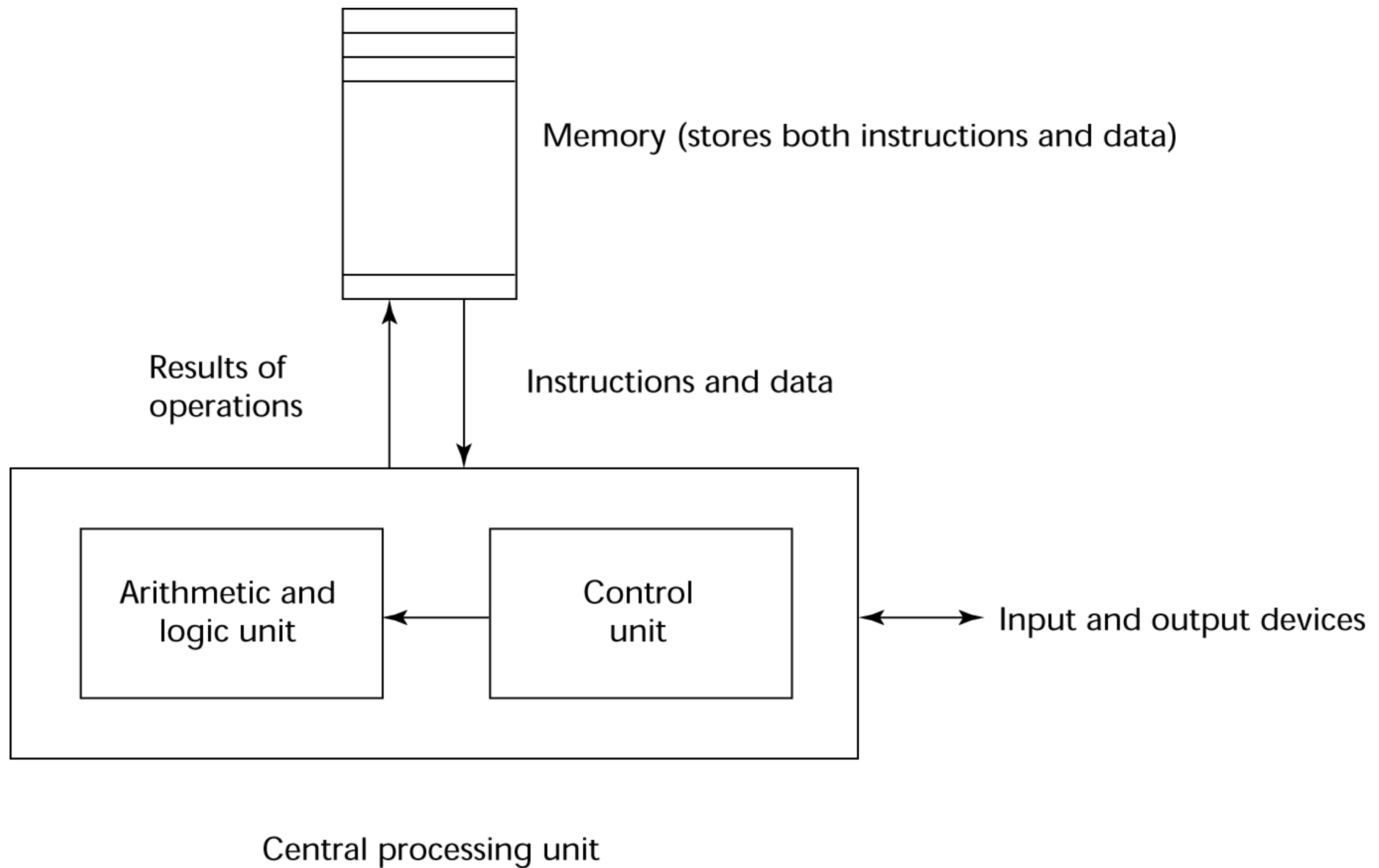
Well-known computer architecture: Von Neumann

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*

Copyright © 2009 Addison-Wesley. All rights reserved.

The von Neumann Architecture



Copyright © 2009 Addison-Wesley. All rights reserved.

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
```

Copyright © 2009 Addison-Wesley. All rights reserved.

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

Copyright © 2009 Addison-Wesley. All rights reserved.

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
- 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

Copyright © 2009 Addison-Wesley. All rights reserved.

Imperative languages

Imperative languages are command-driven or statement oriented languages.

- The basic concept is the machine state (the set of all values for all memory locations).
- A program consists if a sequence of statements and the execution of each statement changes the machine state.
- Programs take the form:

  statement1;

  statement2;

  … …
- FORTRAN, COBOL, C, Pascal, PL/I are all imperative languages.

Copyright © 2009 Addison-Wesley. All rights reserved.

Functional languages

An functional programming language looks at the function that the program represents rather than the state changes as each statement is executed.

- The key question is: What function must be applied to our initial machine and our data to produce the final result?
- Statements take the form:

function_n(function1, function2, . . . (data)) . . .)

- ML, Scheme and LISP are examples of functional languages.

Copyright © 2009 Addison-Wesley. All rights reserved.

Example GCD in Scheme

```
;; A Scheme version of Greatest
;; Common divisor
(define (gcd u v)
  (if (= v 0) u
      (gcd v (modulo u v))))
```

Copyright © 2009 Addison-Wesley. All rights reserved.

A Function GCD in C++

```
//gcd() - A version of greatest common
//        divisor written in C++ in
//        function style
int  gcd(int u, int v)
{
  if (v == 0)
     return(u);
  else
     return(v, u % v);
}
```

Copyright © 2009 Addison-Wesley. All rights reserved.

Rule-Based Languages

Rule-based or ***declarative*** languages execute checking to see if a particular condition is true and if so, perform the appropriate actions.

- The enabling conditions are usually written in terms of predicate calculus and take the form:

  Condition1 $\longrightarrow$ action1

  Condition2 $\longrightarrow$ action2

  … … …

- Prolog is the best know example of a declarative language.

Copyright © 2009 Addison-Wesley. All rights reserved.

GCD in Prolog

```
gcd(U, V, U) :-  V = 0.
gcd(U, V, X) :-  not (V = 0),
                 Y is U mod V.
                 gcd(V, Y, X).
```

means "if"

clauses in Prolog

Copyright © 2009 Addison-Wesley. All rights reserved.

Object-Oriented Languages

- In object-oriented languages, data structures and algorithms support the abstraction of data and endeavor to allow the programmer to use data in a fashion that closely represents its real world use.

- Data abstraction is implemented by use of
  - ***Encapsulation*** – data and procedures belonging to a class can only be accessed by that classes (with noteworthy exceptions).
  - ***Polymorphism*** – the same functions and operators can mean different things depending on the parameters or operands,
  - ***Inheritance*** – New classes may be defined in terms of other, simpler classes.

Copyright © 2009 Addison-Wesley. All rights reserved.

GCD in Java

```
public class IntWithGcd
{ public IntWithGcd( int val ){ value = val; }
  public int intValue() { return value; }
  public int gcd( int val );
  { int z= value;
    int y = v;
    while (y != 0)
    {
      int t = y;
      y = z % y;
      z = t;
    }
    return z;
  }
  private int value;
}
```

Copyright © 2009 Addison-Wesley. All rights reserved.

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

Writability (flexibility) vs. reliability

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

Copyright © 2007 Addison-Wesley. All rights reserved.

Implementation Methods

- Compilation
  - Programs are translated into machine language
- Pure Interpretation
  - Programs are interpreted by another program known as an interpreter
- Hybrid Implementation Systems
  - A compromise between compilers and pure interpreters

Copyright © 2009 Addison-Wesley. All rights reserved.

Compilation

Translate high-level program (source language) into machine code (machine language)

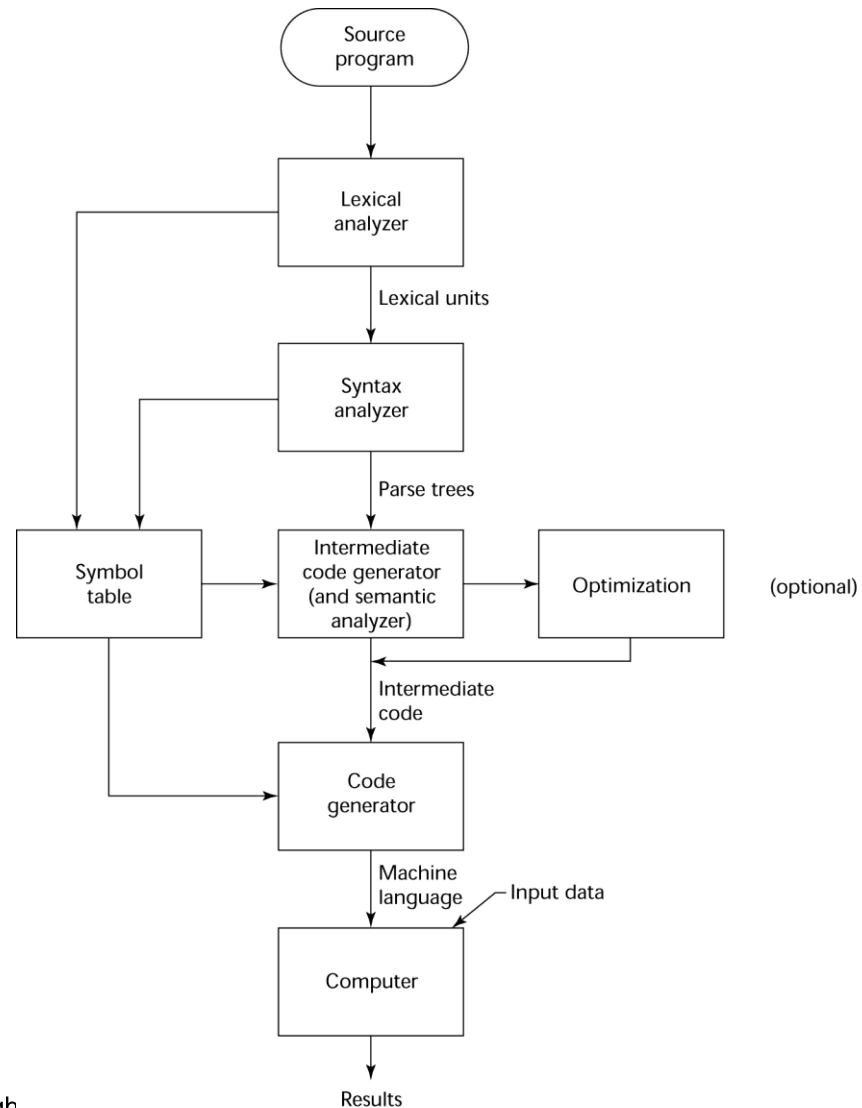
Slow translation, fast execution

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

Copyright © 2009 Addison-Wesley. All rights reserved.

The Compilation Process



Copyright © 2009 Addison-Wesley. All rights reserved.

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

Copyright © 2009 Addison-Wesley. All rights reserved.

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

Copyright © 2009 Addison-Wesley. All rights reserved.

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)

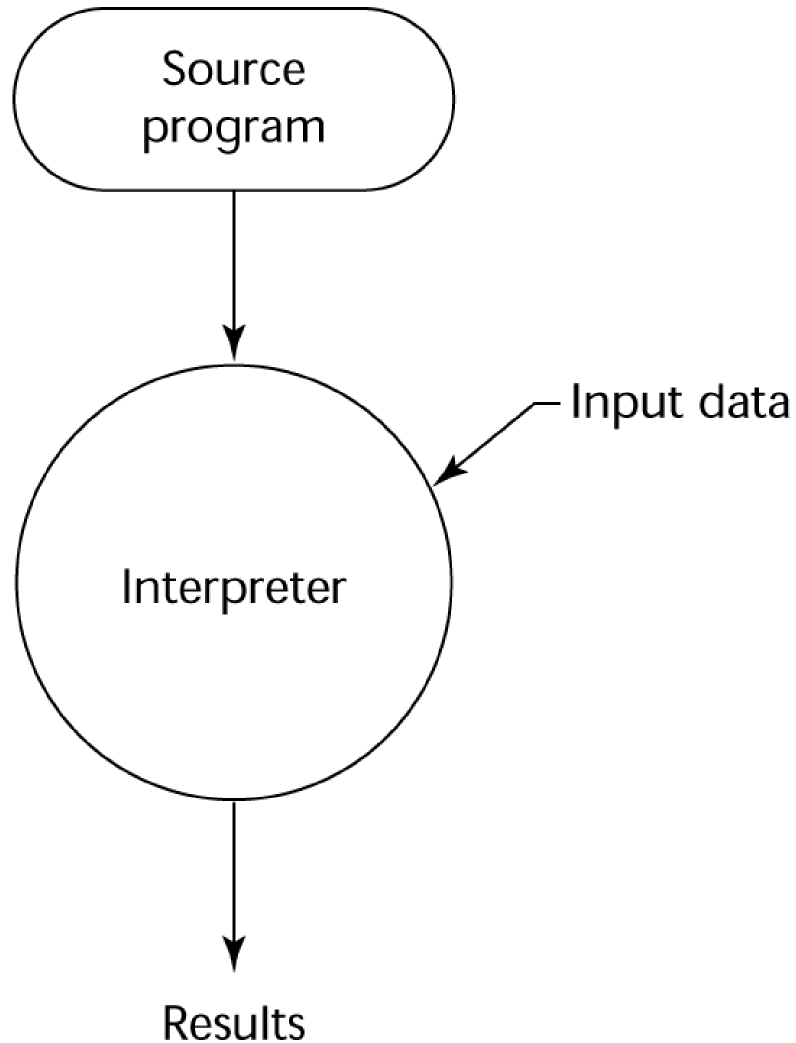
Often requires more space

Now rare for traditional high-level languages

Significant comeback with some Web scripting languages (e.g., JavaScript, PHP)

Copyright © 2009 Addison-Wesley. All rights reserved.

Pure Interpretation Process



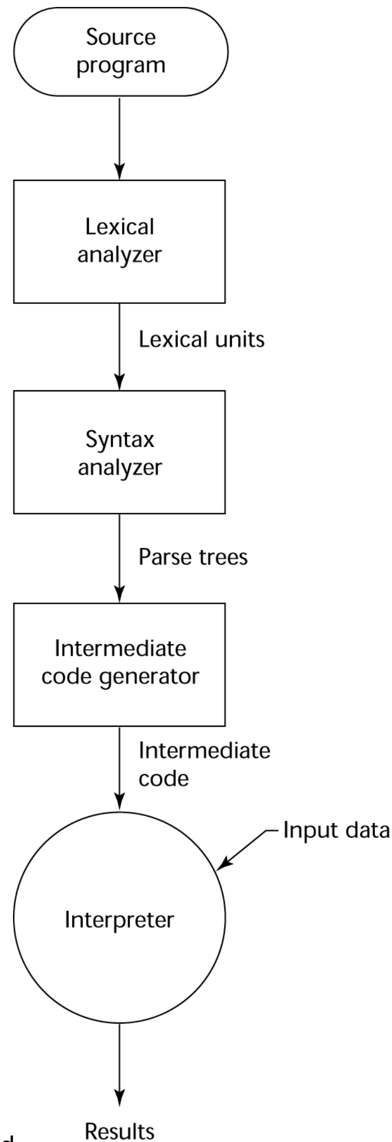
Copyright © 2009 Addison-Wesley. All rights reserved.

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*)

Copyright © 2009 Addison-Wesley. All rights reserved.

Hybrid Implementation Process



Copyright © 2009 Addison-Wesley. All rights reserved.

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

Copyright © 2009 Addison-Wesley. All rights reserved.

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

Copyright © 2009 Addison-Wesley. All rights reserved.

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 Web applications in Java

Copyright © 2009 Addison-Wesley. All rights reserved.

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

Copyright © 2009 Addison-Wesley. All rights reserved.