Faculty of Information Technology

Spring 2025

Concepts of Programming Languages

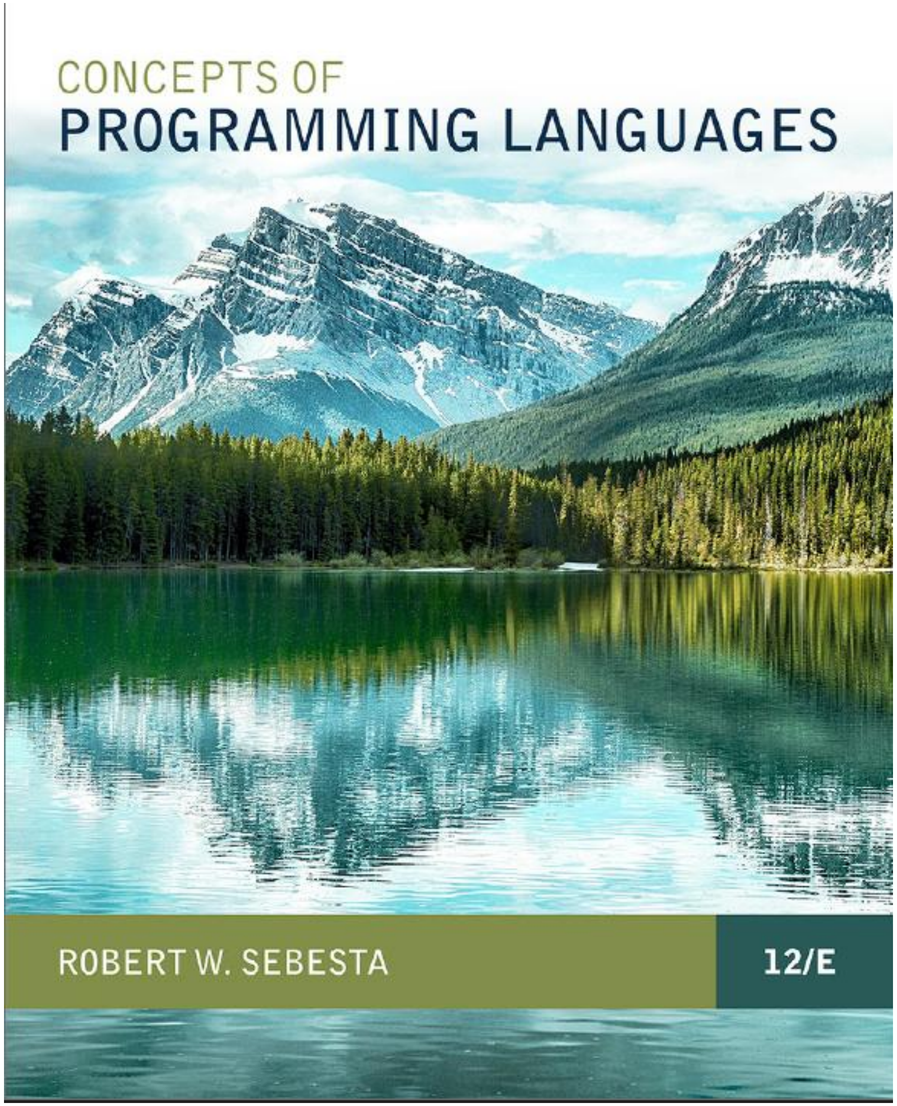
CS 211

Lecture (1)

Assessment Schedule

Assessment Method		Week	Weight
Semester Work	Exam (1)	Week#6	20%
	Exam (2)	Week#10	20%
Final Practical Exam		Week#14	20%
Final Written Exam		Week#15	40%

Textbook



Outline

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

Outline

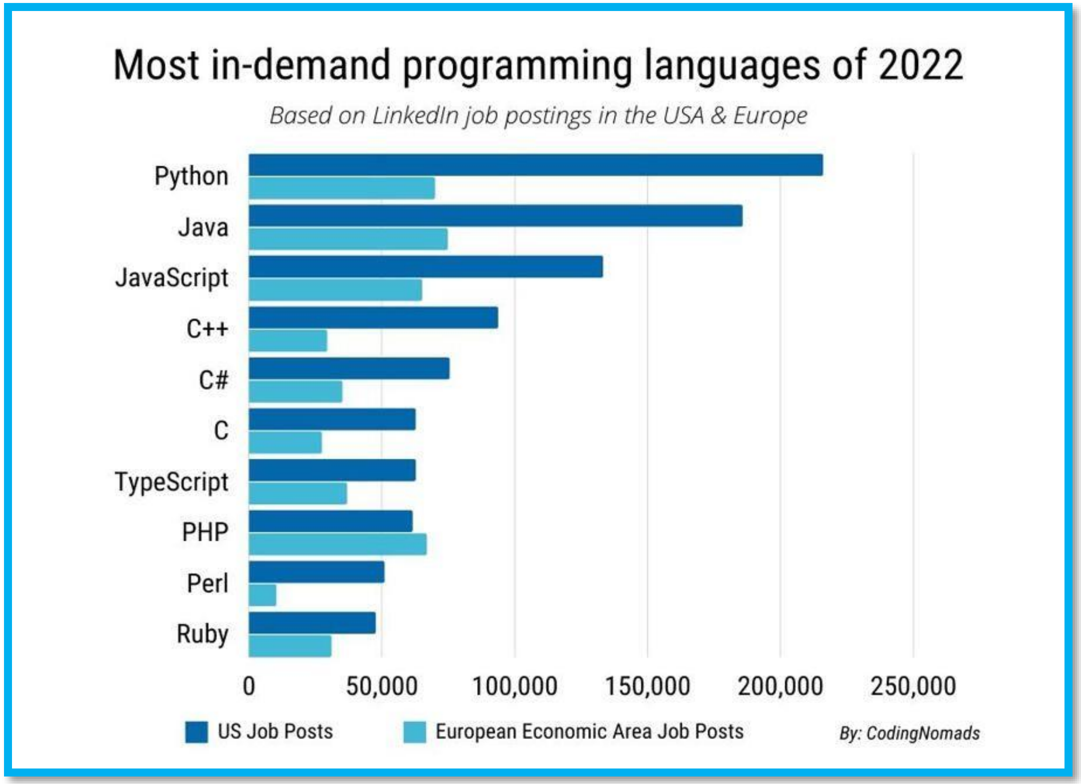
- Reasons for Studying Concepts of Programming Languages
- Programming Domains
- Language Evaluation Criteria
- Influences on Language Design
- Language Design Trade-Offs
- Language Categories
- Implementation Methods
- Programming Environments

Reasons for Studying Concepts of Programming Languages

- Increased ability to express ideas.

Reasons for Studying Concepts of Programming Languages (Cont.)

- Improved background for choosing appropriate languages.



Reasons for Studying Concepts of Programming Languages (Cont.)

- Increased ability to learn new languages.

Reasons for Studying Concepts of Programming Languages (Cont.)

- Better understanding of significance of implementation.

Reasons for Studying Concepts of Programming Languages (Cont.)

- Better use of languages that are already known.
- Overall advancement of computing.

Outline

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

Programming Domains

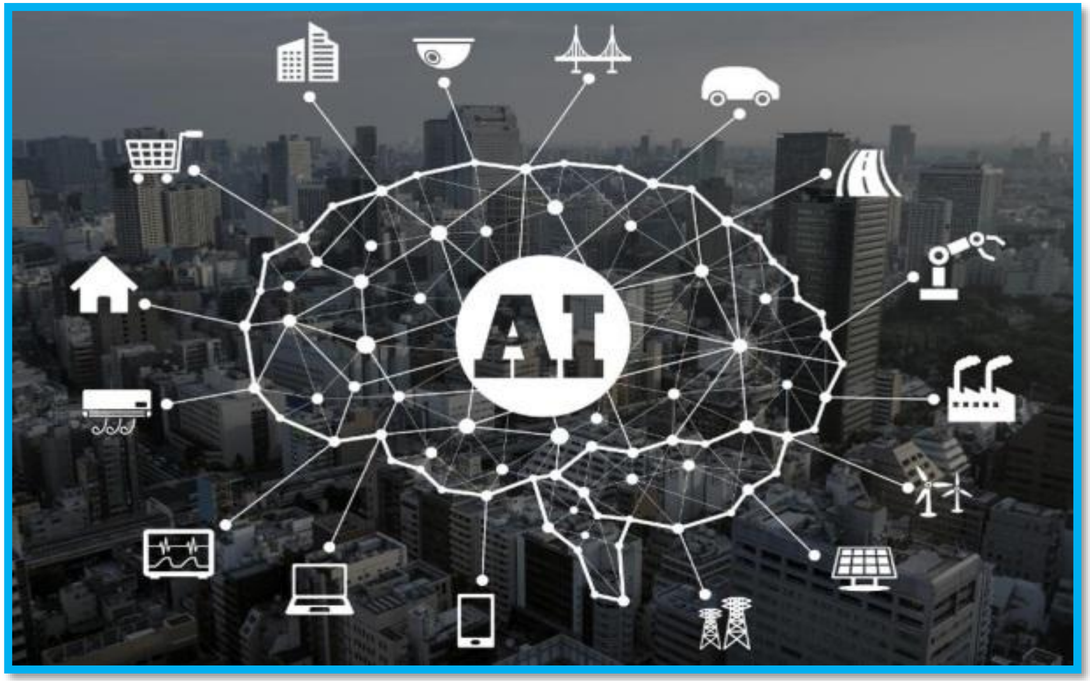
- Scientific Applications:-
  - Large numbers of floating-point computations; use of arrays.
  - Fortran

Programming Domains (Cont.)

- Business Applications:-
  - Produce reports, use decimal numbers and characters.
  - COBOL

Programming Domains (Cont.)

- Artificial Intelligence:-
  - Symbols rather than numbers manipulated; use of linked lists.
  - LISP



Programming Domains (Cont.)

- Systems Programming:-
  - Need efficiency because of continuous use.
  - C

Programming Domains (Cont.)

- Web Software:-
  - Eclectic collection of languages: markup (e.g., HTML), scripting (e.g., PHP), general-purpose (e.g., Java).

Outline

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

Language Evaluation Criteria

- Readability: the ease with which programs can be read and understood.

Language Evaluation Criteria (Cont.)

- 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.

Language Evaluation Criteria (Cont.)

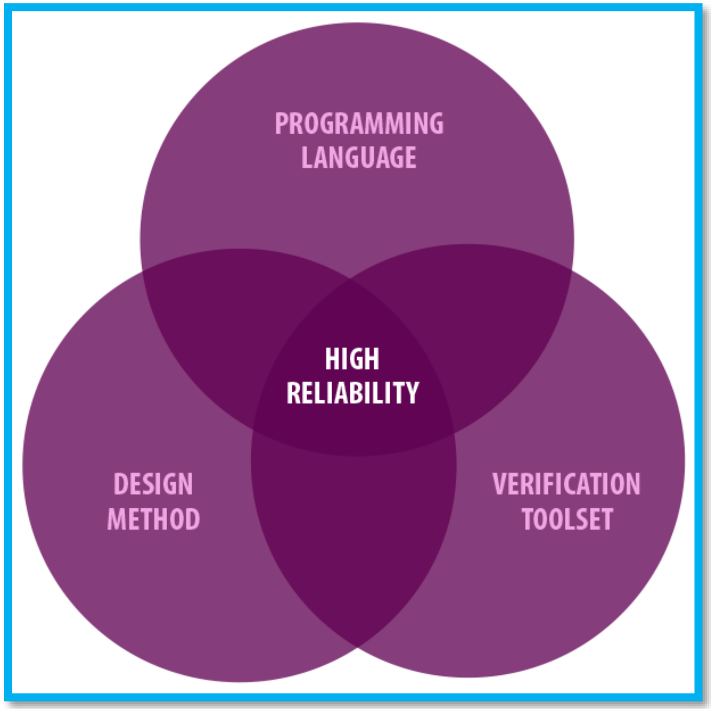
- Writability: the ease with which a language can be used to create programs.

Language Evaluation Criteria (Cont.)

- Simplicity and orthogonality:-
  - Few constructs, a small number of primitives, a small set of rules for combining them.
- 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.

Language Evaluation Criteria (Cont.)

- Reliability: conformance to specifications (i.e., performs to its specifications).



Language Evaluation Criteria (Cont.)

- Type checking:-
  - Testing for type errors.
- Exception handling:-
  - Intercept run-time errors and take corrective measures.
- Aliasing:-
  - 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.

Language Evaluation Criteria (Cont.)

- Cost: the ultimate total cost.

Language Evaluation Criteria (Cont.)

- Training programmers to use the language.
- Writing programs (closeness to applications).
- Executing programs.
- Reliability: poor reliability leads to high costs.
- Maintaining programs.

Evaluation Criteria: Others (Cont.)

- Portability:-
  - The ease with which programs can be moved from one implementation to another.
- Generality:-
  - The applicability to a wide range of applications.
- Well-Definedness:-
  - The completeness and precision of the language’s official definition.

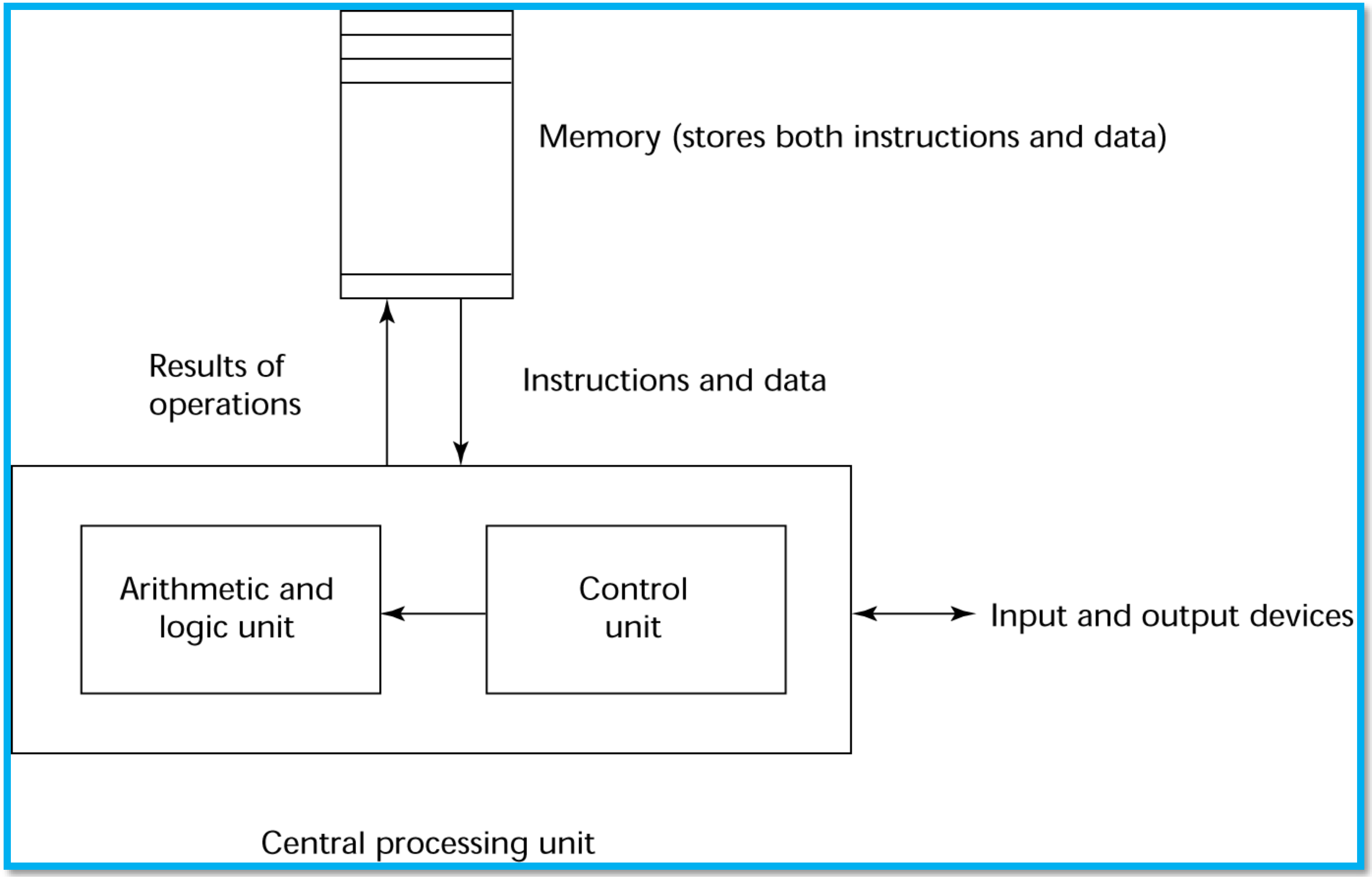
Outline

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

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.

The von Neumann Architecture



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.

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.

Outline

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

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.

Language Design Trade-Offs (Cont.)

- 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.

Language Design Trade-Offs (Cont.)

- Writability (flexibility) vs. reliability:-
  - Example: C++ pointers are powerful and very flexible but are unreliable.

Outline

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

Language Categories (Cont.)

- 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#

Language Categories (Cont.)

- 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

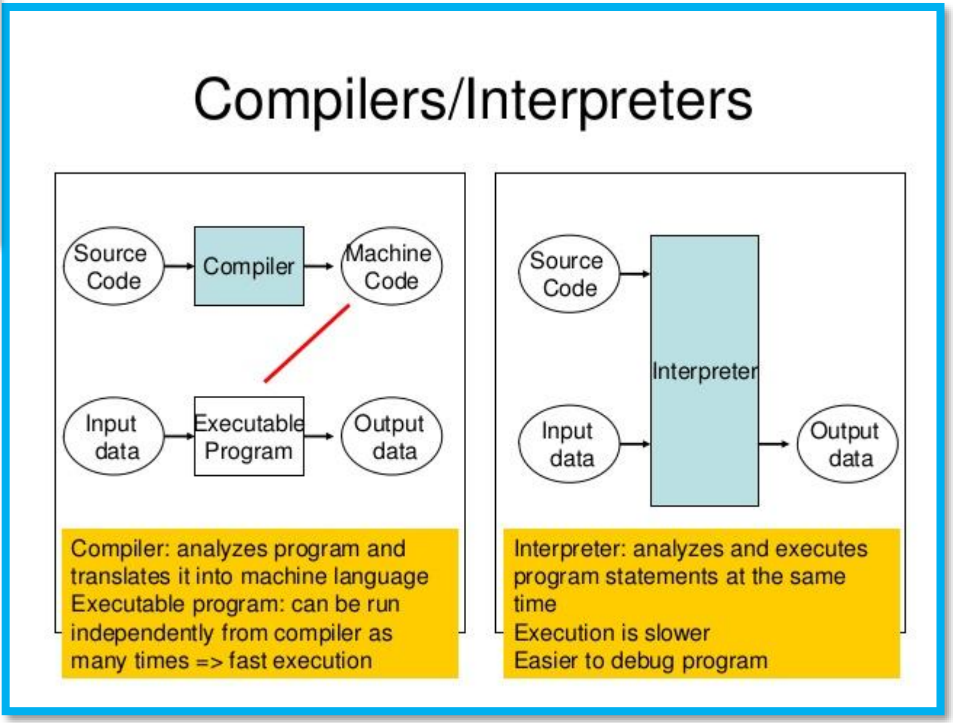
Outline

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

Implementation Methods

- Compilation:-
  - Programs are translated into machine language; includes Just-In-Time (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.

Implementation Methods (Cont.)



Compilers/Interpreters

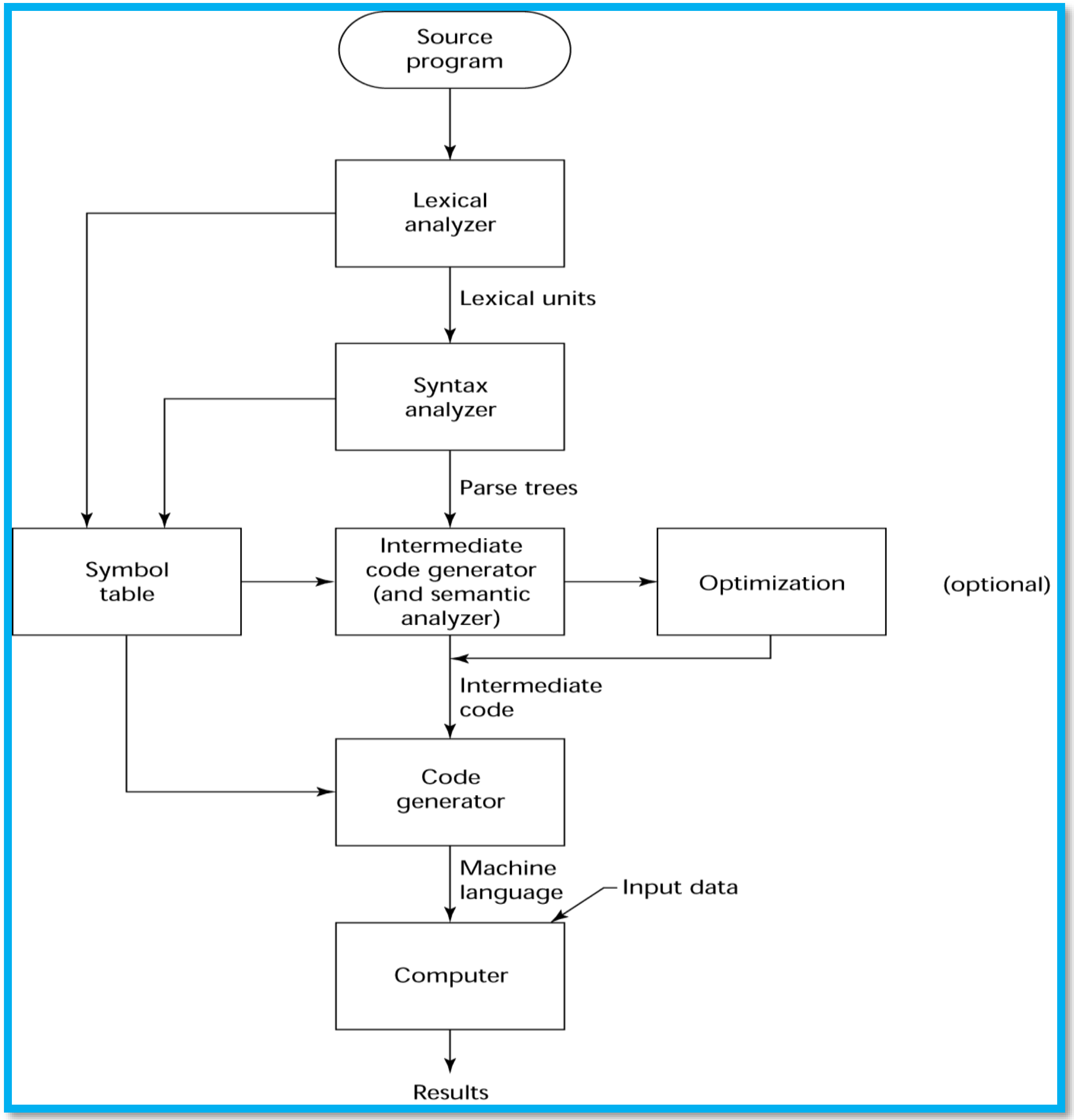
Compiler: analyzes program and translates it into machine language
Executable program: can be run independently from compiler as many times => fast execution

Interpreter: analyzes and executes program statements at the same time
Execution is slower
Easier to debug program

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.

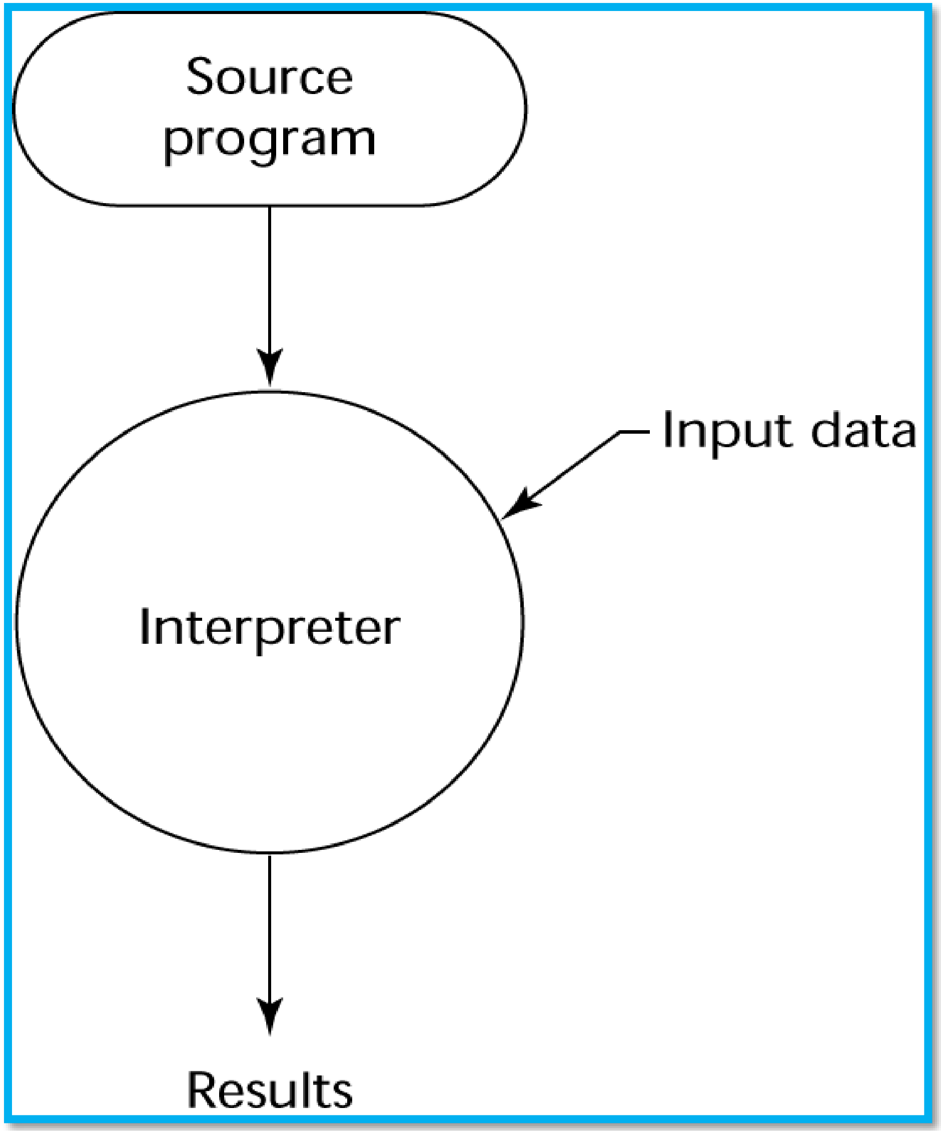
Compilation Process



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).

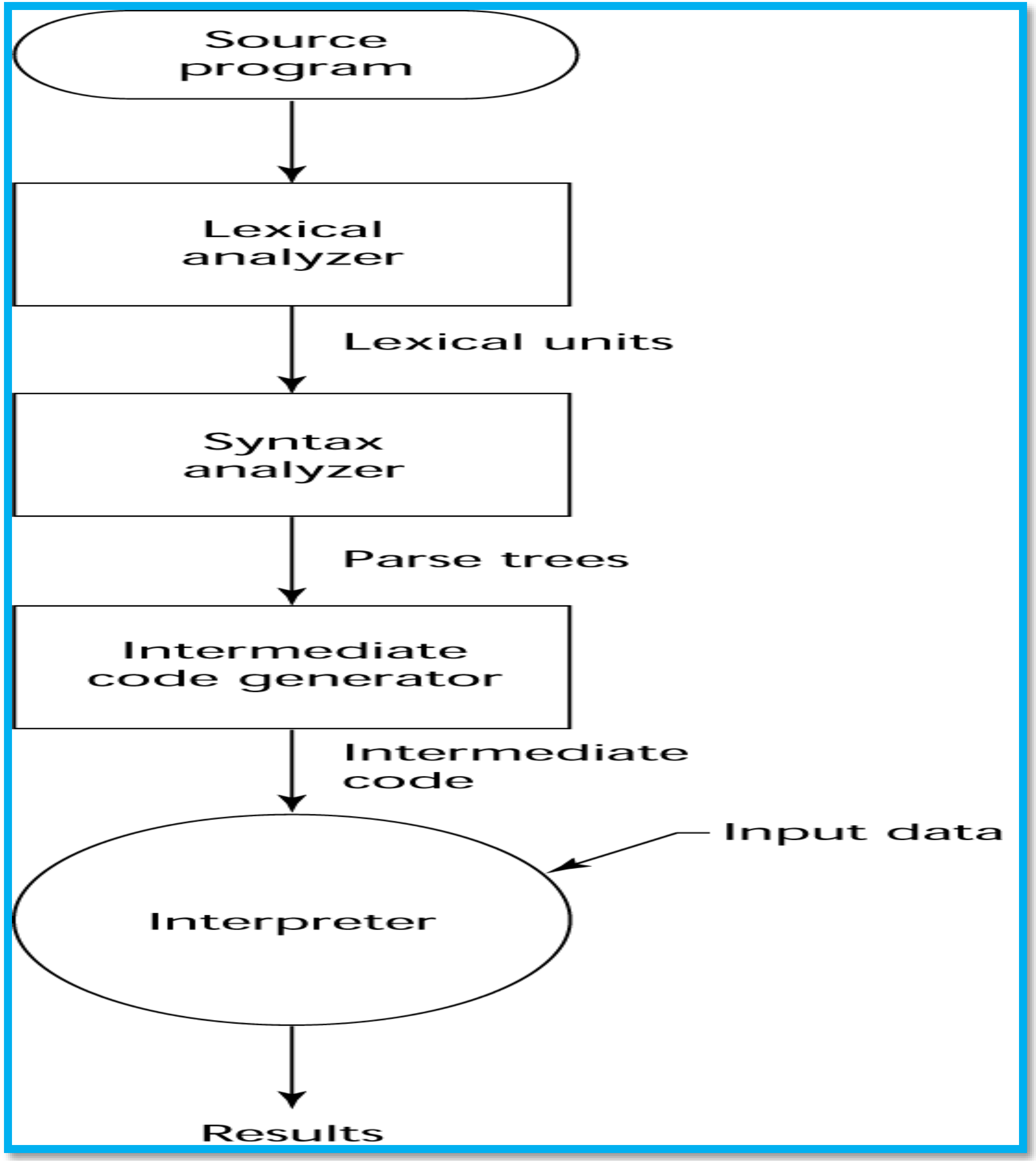
Pure Interpretation Process



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



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.

Outline

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

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

ANY QUESTIONS

Thank You!