Chapter One – Lecture 1

Introduction to Compilers

(A complete reference you can rely on without returning to the slides — A+ level, insha'Allah)

---

1. Introduction

Computers can understand **only machine language**, which consists of binary digits (0s and 1s).
This language is extremely difficult for humans to read, write, and debug.

To make programming practical, **high-level programming languages (HLLs)** were developed. These languages are:

- Easier for humans to understand
- Closer to natural and mathematical thinking
- Independent of specific hardware

However, computers **cannot directly execute high-level languages**.
Therefore, a **translation process** is required to convert high-level programs into machine language.

This translation is performed by **language processors**, mainly:

- Compilers
- Interpreters

---

2. Computer Organization

A computer system is organized into layers:

1. **Applications**
   Programs used by end users.
2. **Compiler**
   Translates high-level programs into low-level code.
3. **Operating System**
   Manages hardware resources.
4. **Hardware**
   Physical components (CPU, memory, I/O devices).

📌 The compiler plays a **critical role** as the bridge between software and hardware.

---

3. Evolution of Programming Languages

Programming languages evolved through several generations:

First Generation Languages (1GL)

- Machine languages (binary code)
- Very fast but extremely difficult to use

Second Generation Languages (2GL)

- Assembly languages
- Use mnemonic symbols instead of binary
- Still machine-dependent

Third Generation Languages (3GL)

- High-level languages such as:
  - Fortran, COBOL, Lisp
  - C, C++, C#, Java
- Portable and widely used

Fourth Generation Languages (4GL)

- Designed for specific applications
- Examples:
  - SQL (database queries)
  - PostScript (text formatting)
  - NOMAD (report generation)

Fifth Generation Languages (5GL)

- Logic- and constraint-based languages
- Used in Artificial Intelligence
- Examples: Prolog, OPS5

---

4. Why Use High-Level Languages?

High-level languages are preferred because:

1. Their notation is closer to human thinking.
2. Compilers can detect many programming errors.
3. Programs are shorter and easier to maintain.
4. Programs can be compiled for different machines (portability).

---

5. Characteristics of High-Level Languages

High-level languages provide:

- **Expressions**: arithmetic operations (+, −, ×, ÷)

- **Data Types**:
  - Primitive (int, float, boolean)
  - Composite (arrays, records)
- **Control Structures**:
  - Selection (if-else)
  - Iteration (loops)
- **Declarations**:
  - Introduction of variables, constants, procedures
- **Abstraction**:
  - Breaking complex problems into smaller parts
- **Encapsulation**:
  - Grouping data and hiding internal details (e.g., classes)

---

6. Why Study Principles of Programming Languages?

Studying programming languages helps to:

- Become a better software engineer
- Understand how language features are implemented
- Select appropriate languages for specific tasks
- Learn new languages more easily
- Understand advantages and disadvantages of languages
- Avoid repeating historical mistakes in language design

---

7. Language Processors

A **language processor** is a program that translates or executes source code.

Examples include:

- Editors
- Compilers
- Interpreters
- Assemblers
- Linkers
- Loaders
- Debuggers
- Profilers

---

8. Language Processing System

The translation of a program typically follows these stages:

1. **Source Program**
2. **Preprocessor**
   - Modifies source code
3. **Compiler**
   - Produces assembly or intermediate code
4. **Assembler**
   - Generates object code
5. **Linker**
   - Combines object files and libraries
6. **Target Machine Code**

---

9. Preprocessor

A preprocessor prepares the source program before compilation.
Its main functions include:

1. **Macro Processing**
   - Replacing macros with code
2. **File Inclusion**
   - Including header files
3. **Rational Preprocessing**
   - Adding structured control constructs
4. **Language Extensions**
   - Extending language capabilities

📌 The preprocessor **does not translate** the program; it only modifies it.

---

10. What Is a Compiler?

A **compiler** is a program that:

- Reads a program written in a **source language**
- Translates it into an equivalent **target language**
- Produces **error messages** during translation

Execution Process:

1. Compilation of source program
2. Generation of object program
3. Loading into memory

4. Execution

---

11. Compilation

Compilation is the process of translating a source program into a semantically equivalent target program.

Characteristics:

- Entire program is translated at once
- Errors are reported after compilation
- Execution occurs after successful translation

---

12. Interpreter

An **interpreter**:

- Executes the source program directly
- Translates and executes one statement at a time
- Does not produce a separate target program

Characteristics:

- Slower execution
- Easier debugging
- Errors are reported immediately

---

13. Hybrid Compiler

A **hybrid approach** combines compilation and interpretation.

Example: **Java**

- Source code is compiled into intermediate bytecode
- Bytecode is executed by a **Virtual Machine (Interpreter)**

---

14. Compiler vs. Interpreter

Compiler	Interpreter
Translates entire program	Translates one statement at a time

Compiler	Interpreter
Faster execution	Slower execution
Produces intermediate object code	No intermediate code
Requires more memory	Requires less memory
Errors shown after compilation	Errors shown immediately
Harder debugging	Easier debugging
Examples: C, C++	Examples: Python, Ruby

---

15. Debugging

Debugging is the process of:

- Finding errors (bugs)
- Fixing errors in source code

It is an essential part of software development.

---

16. Why Do We Need to Know Compilers?

Studying compilers:

- Explains how programs actually work
- Demonstrates the interaction between theory and practice
- Uses fundamental concepts such as:
  - Automata and regular expressions (lexical analysis)
  - Context-free grammars and parsing
  - Hash tables (symbol tables)
  - Graph coloring and optimization

---

17. Applications of Compiler Technology

Compiler technology is used in:

- Programming language implementation
- Document processing (Word → PDF)
- Natural language processing

- Hardware design
- Report generation
- Input/output translation tools

---

18. Issues Driving Compiler Design

Key design considerations include:

- **Correctness**
- **Execution speed**
- **Compilation time**
- **Memory usage**
- **Optimization techniques**
- **User feedback**
- **Debugging support**

Final Key Takeaway (Exam-Focused)

> A compiler is the fundamental link that transforms human-readable programs into machine-executable code.
> Understanding compilers means understanding how software truly works.