Chapter 1

Introduction

Computer Components

- Hardware
  - Physical Components
  - Keyboard, Mouse, Screen, Hard Disk. etc
- Software
  - System SW
    - Programs written for computer systems
      - Compilers, operating systems, …
  - Application SW
    - Programs written for computer users
      - Word-processors, spreadsheets, & other application packages

Programs

- Programs (software) are written in programming languages
  - PL = programming language
  - Pieces of the same program can be written in different PLs
- A PL is
  - A special purpose and limited language
  - A set of rules and symbols used to construct a computer program
  - A language used to interact with the computer

Programming Languages

- Programming languages allow programmers to code software.
- The three major families of languages are:
  - Machine languages
  - Assembly languages
  - High-Level languages

Machine Languages

- Comprised of 1s and 0s
- The “native” language of a computer
- Difficult to program – one misplaced 1 or 0 will cause the program to fail.
- Example of code:

```
1110100010101      111010101110
1011110110100      1010001110111
```

Assembly Languages

- Assembly languages are a step towards easier programming.
- Assembly languages are comprised of a set of elemental commands which are tied to a specific processor.
- Assembly language code needs to be translated to machine language before the computer processes it.
- Example:

```
ADD  1001010, 1011010
```

High-Level Languages

- High-level languages represent a giant leap towards easier programming.
- The syntax of HL languages is similar to English.
- Historically, we divide HL languages into two groups:
  - Procedural languages
  - Object-Oriented languages (OOP)

Procedural Languages

- Early high-level languages are typically called procedural languages.
- Procedural languages are characterized by sequential sets of linear commands. The focus of such languages is on *structure*.
- Examples include C, COBOL, Fortran, LISP, Perl, HTML, VBScript

Object-Oriented Languages

- Most object-oriented languages are high-level languages.
- The focus of OOP languages is not on structure, but on *modeling data*.
- Programmers code using “blueprints” of data models called *classes*.
- Examples of OOP languages include C++, Visual Basic.NET and Java.

Compilers & Programs

- **Compiler**
  - A program that **converts another program** from some source language (or high-level programming language / HLL) to machine language (object code).
  - Some compilers output assembly language which is then converted to machine language by a separate assembler.
  - Is distinguished from an assembler by the fact that each input statement, in general, correspond to **more than one machine instruction**.

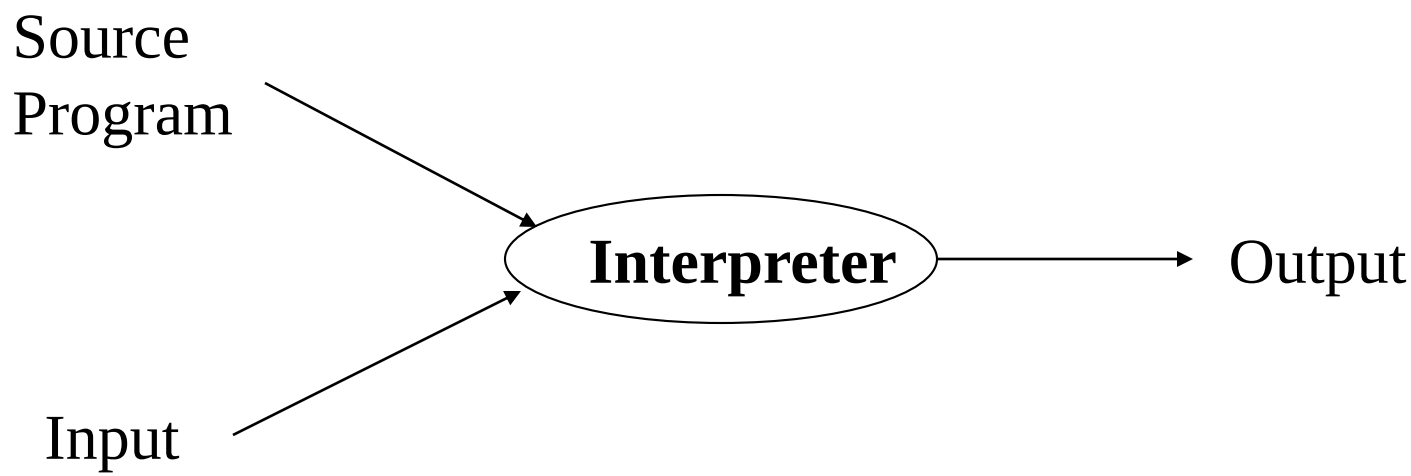
Compilers & Programs

- **Source program**
  - The form in which a computer program, written in some formal programming language, is **written by the programmer**.
  - Can be compiled automatically into object code or machine code or executed by an interpreter.

Compilers & Programs

- **Object program**
  - Output from the compiler
  - Equivalent machine language translation of the source program
  - Files usually have extension ‘.obj’
- **Executable program**
  - Output from linker/loader
  - Machine language program linked with necessary libraries & other files
  - Files usually have extension ‘.exe’

Interpretation

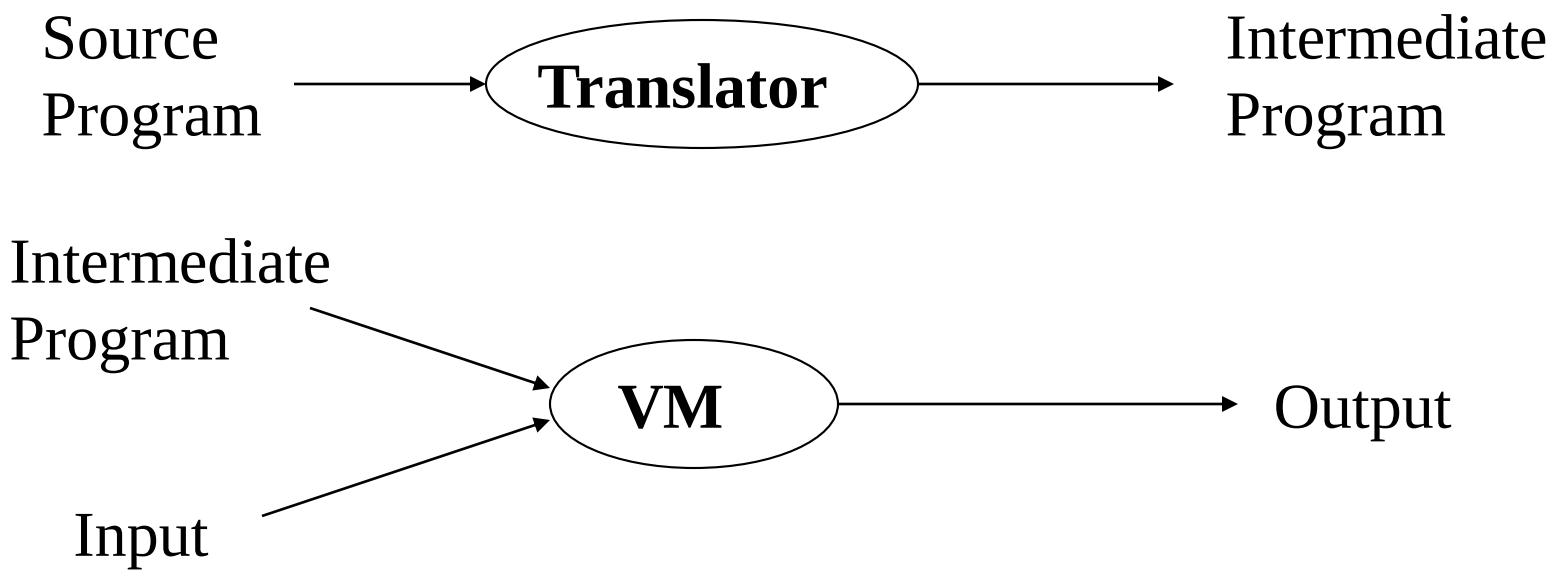


- The interpreter **stays around during execution**
- It reads and executes statements one at a time

Compilation vs. Interpretation

- Compilation:
  - Syntax errors caught before running the program
  - Better performance
  - Decisions made once, at compile time
- Interpretation:
  - Better diagnostics (error messages)
  - More flexibility
  - Supports **late binding** (delaying decisions about program implementation until runtime)
    - Can better cope with PLs where type and size of variables depend on input
  - Supports creation/modification of program code on the fly (e.g. Lisp, Prolog)

Mixture of Compiler & Interpreter



- Many programming languages implement this
- Interpreter implements a Virtual Machine (VM).

Program Development Cycle

Programming as Problem Solving

- Problem solving principles:
  1. Completely understand the problem
  2. Devise a plan to solve it
  3. Carry out the plan
  4. Review the results
- Developing a Program:
  1. Analyze the problem
  2. Design the program
  3. Code the program
  4. Test the program

1) Analyze the Problem

- Mary's Thousands
  - The problem: Mary wants to invest money at a local bank. There are many options such as interest rates, terms of deposit, compounding frequencies. She needs a program to compute, for any given initial investment, the final maturity (value) of the deposit.
    - What are the inputs? (given data)
    - What are the outputs? (required data)
    - How will we calculate the required outputs from the given inputs?

2) Design the Program

- Create an outline of the program
- An **algorithm –** a step by step procedure that will provide the required results from the given inputs.
- Algorithm Examples: Instructions on how to make a cake, use the bank’s ATM, etc.

3) Code the Program

- Once the design is completed, write the program **code**.
- Code is written in some programming language such as BASIC, Pascal, C++, Java, etc.

4) Testing the program

- Locate any errors (bugs)
- Testing is done throughout the development cycle
- Code **walkthrough** is performed to locate errors in the code.
- Ultimate test is to run the program to see if the outputs are correct for the given inputs.

Modular Programming

- Determine the major **tasks** that the program must accomplish.
- Each of these tasks will be a **module**.
- Some modules will be complex themselves, and they will be broken into ***sub-modules***, and those sub-modules may also be broken into even smaller modules.
- This is called **top-down design**