Lecture 2

Introduction to Programming

Shibo Li

shiboli@cs.fsu.edu

Department of Computer Science
Florida State University

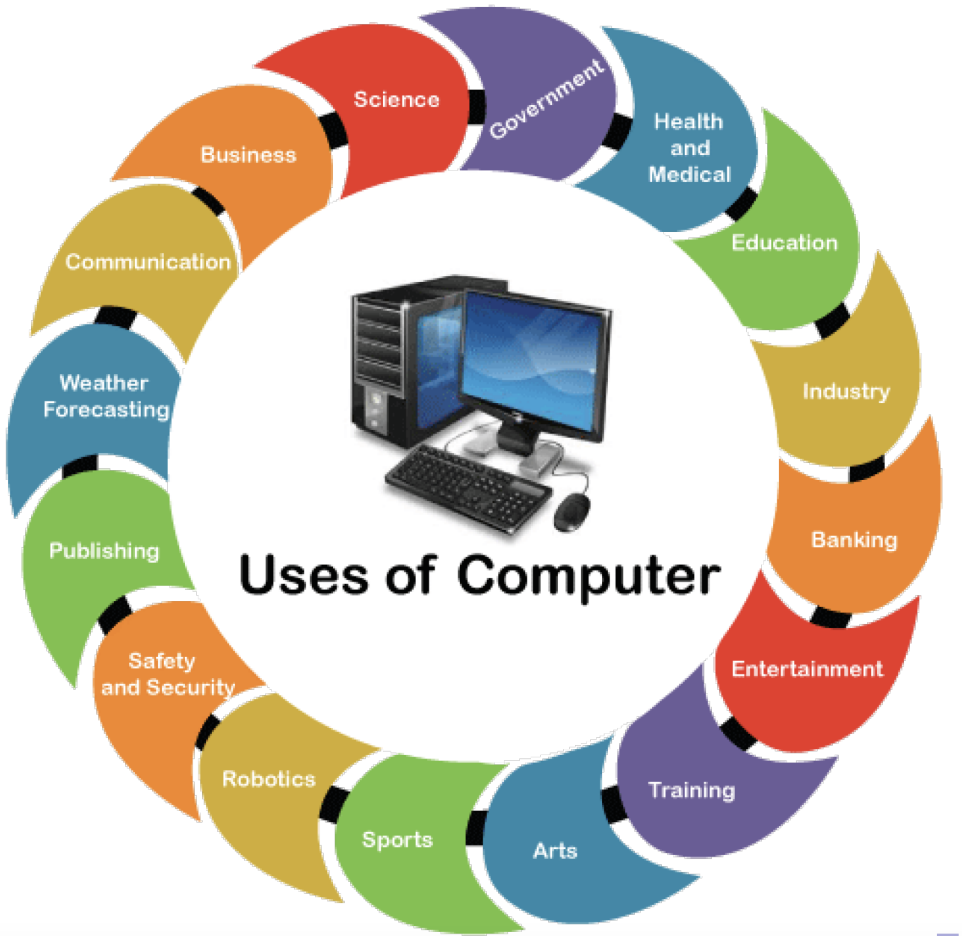
The slides are mainly from Sharanya Jayaraman

Why study Computer Programming

- **Problem Solving Skills:** break-and conquer
- **Automation and Efficiency:** automate repetive tasks, increasing efficiency
- **Empowerment:** create your own solutions rather than relying on others
- **Innovation of Technology:** As technology becomes more integral to our daily lives, programming allows you to create new tools, websites, apps and even art!
- **Career Opportunities:** The demand for programming skills is high across various industries.

Programming is a Ubiquitous Discipline

- Computing systems are used in nearly every field. The understanding of how to effective use a computer is becoming essential in almost every diciplines.



What Makes Computer Programming Different

- Almost everyone uses a computer.
- The most common uses of computers include end-user-applications (Apps) that do not involve any modifications of the programs. These are "shipped" to the user as a finished product.
- Some applications are "tools". These allow some freedom to put the processing power of the computer to use data and some direction from the user to accomplish a task.
- For example, we can use Microsoft Excel application to generate Reports. This involves some knowledge of what the underlying system/data is capable of doing.

What Makes Computer Programming Different

- Computer Programming can be done on two levels.
- **Application Development** using mostly existing software products and libraries to put together a product for a specific area of use. This requires domain knowledge as well as a general understanding of Computer Science.
- **Software Development** using basic programming language constructs to develop the software and libraries used in Application Development. This requires deeper knowledge of Computer Science, including theory and hardware.
- Both of the above can be considered **Software Engineering.**

Main Components of a Computer

CPU - Central Processing Unit: the "brain" of the computer

- Processing instructions: x86/x86-64, ARM, MIPS, etc.
- Arithmetic and logic operators: Add/Sub, Mul/Div, AND/OR/NOT/XOR, Bit-Shifting
- Fetching, Decoding, Executing: CPU follows a cycle where to fetch instruction from memory, decodes what needs to be done and executes the instruction:

```
int a = 3; int b =4; c = a + b;
cout << c;
```

- Manage Multitasking

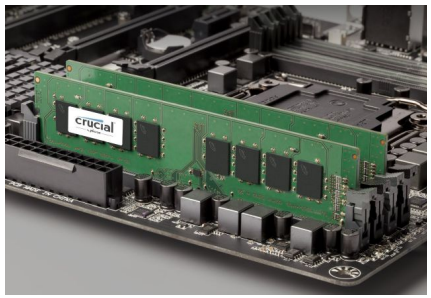
Main Components of a Computer

Memory(RAM) - Random Access Memory

- ***Faster*** to access than hard disk
- Store instructions and data that the CPU needs while performing tasks.

Storage Devices - *Log-term* storage of data, files, and applications

- HDD, SSD, Flash Drives, Optical Drives(CD/DVD/Blu-Ray for reaching and writing), Tapes, etc.



(a) RAM

(b) Hard Drive

(c) CD/DVD

(d) Tape Cartridge

Main Components of a Computer

User's Devices

- *Input:* mouse, keyborad, scanner, microphone, etc.
- *Output:* monitor, printer, speaker, VR goggle, etc.

Motherboard

- The main circuit board that connects all components of the computer
- Include sockets for CPU, RAM, storage devices, power supply, other peripherals, and expansion slots, etc.

Main Components of a Computer

Network Interface Card (NIC)

- Allows the computer to connect to a network (Ethernet or Wi-Fi), transimitte and receive packages

Graphics Processig Unit (GPU)

- Specialized processor designed for rendering, processing images

Audio Card

- Converting digial audio data into analog signals and vice versa.

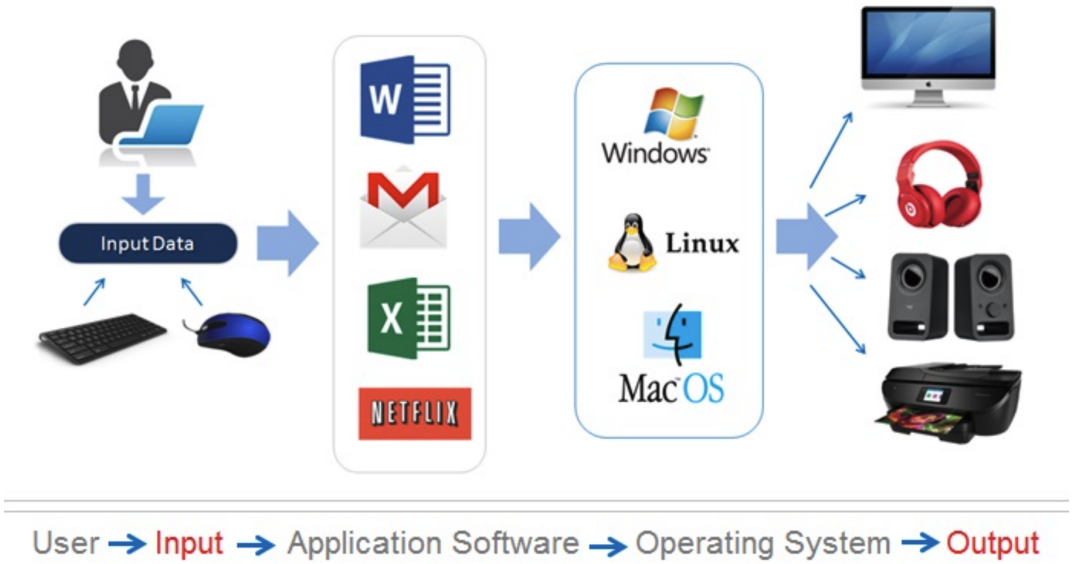
(a) Entertainment

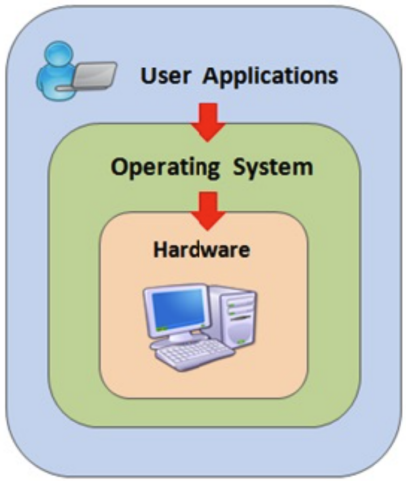
(b) AI and Machine Learning

Software Components

- Operating Systems (OS): The most important software that manages the hardware resouces and provide services to the user applications (e.g., macOS, Windows, Linux, etc)
- Application: Programs that perform specific tasks for users (e.g., Microsoft Office, browsers, games, etc.)
- Drivers: Software that allows the OS to communicate with hardware components

A general computing system

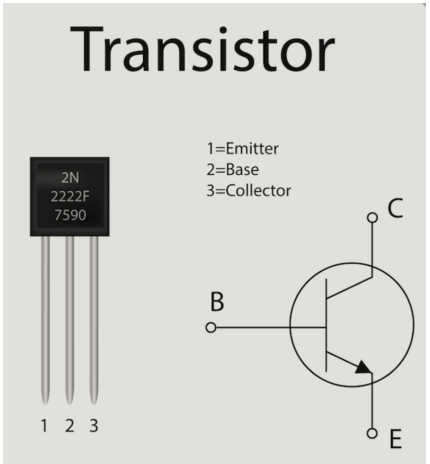




Memory Concepts

Bit: A binary digit

- Stores the value of **0** or **1**
- Smallest unit of storage in a computer



Memory Concepts

Byte: 8 bits

- Smallest addressable unit of storage in a computer
- Storage units (variables) in a program are 1 or more bytes, for example, `int8`, `int64`, `float32`, `cfloat64`
- Each byte in memory has an address: a number that identifies the location, constant time access

Address	Value
0x00	01001010
0x01	10111010
0x02	01011111
0x03	00100100
0x04	01000100
0x05	10100000
0x06	01110100
0x07	01101111
0x08	10111011
...	...
0xFE	11011110
0xFF	10111011

What is A Program?

A set of instructions for a computer to execute

```
section .data
    num1 db 10      ; First number (10)
    num2 db 20      ; Second number (20)
    num3 db 30      ; Third number (30)
    result db 0     ; Memory location to store the result

section .text
    global _start   ; Entry point for the program

_start:
    ; Load the first number into the AL register
    mov al, [num1]

    ; Add the second number to AL
    add al, [num2]

    ; Add the third number to AL
    add al, [num3]

    ; Store the result in the 'result' memory location
    mov [result], al

    ; Exit the program
    mov eax, 60     ; syscall number for exit
    xor edi, edi    ; return code 0
    syscall
```

(a) Assembly

```
#include <iostream>

int main() {
    // Declare three integers to hold the numbers
    int num1, num2, num3;

    // Ask the user to input the three numbers
    std::cout << "Enter three numbers: ";
    std::cin >> num1 >> num2 >> num3;

    // Calculate the sum of the three numbers
    int sum = num1 + num2 + num3;

    // Output the result
    std::cout << "The sum of " << num1 << ", " << num2

    return 0;
}
```

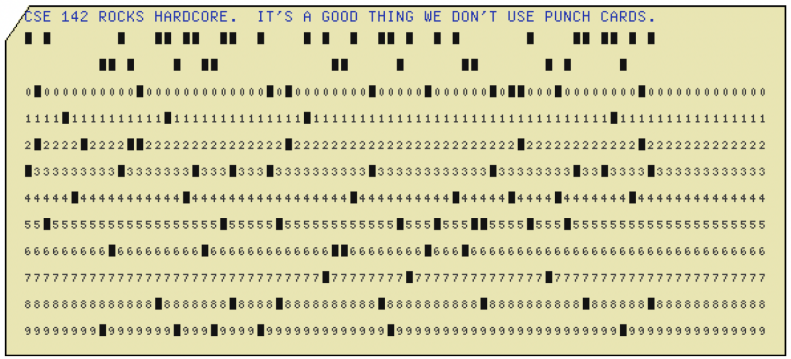
(b) C++

```
# Prompt the user to enter three numbers
num1 = float(input("Enter the first number: "))
num2 = float(input("Enter the second number: "))
num3 = float(input("Enter the third number: "))

# Calculate the sum of the three numbers
sum_of_numbers = num1 + num2 + num3

# Print the result
print(f"The sum of {num1}, {num2}, and {num3} is {sum_
```

(c) Python

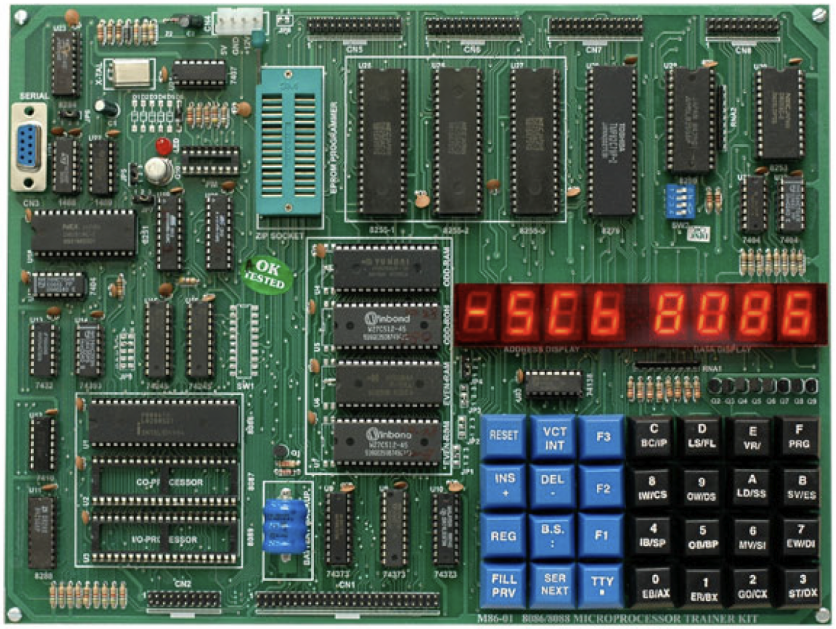
Machine Language

- Based on machine's core instruction set
- Binary code is fundamental to how computers process information, as it represents the most basic form of data that computers understand.
- Example: 11101101010101100011101010



Assembly Language

- translation of machine instructions to symbols, slightly easier for human to read
- `ADD $R1, $R2, $R3`



High-Level Procedural Languages

- Abstraction of concepts into more human-readable terms
- Closer to "natural language" (i.e. what we speak)
- Easy to write and design, but must be translated for computer
- Examples include C, Pascal, Fortran

Object-oriented languages

- Abstraction taken farther than procedural languages
- Objects model real-world objects, not only storing data(attributes), but having inherent behaviors (operations,functions)
- Easier to design and write good, portable, maintainable code
- Examples include C++, Java

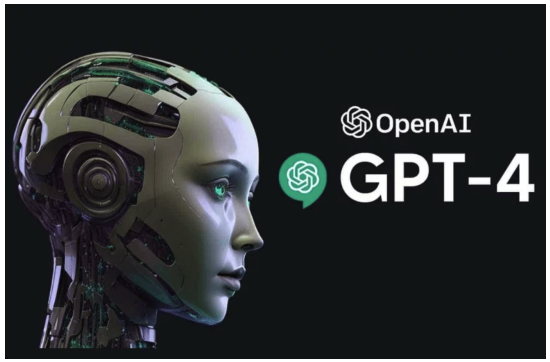
Interpreted Languages

- Line-by-line execution
- No need for Compilation
- Platform independence
- Slower execution speed
- Easy to debug
- Examples: Python, JavaScript, Ruby, PHP, R, MATLAB

Execution Efficiency *v.s.* Developing Efficiency

(a) Punch Card

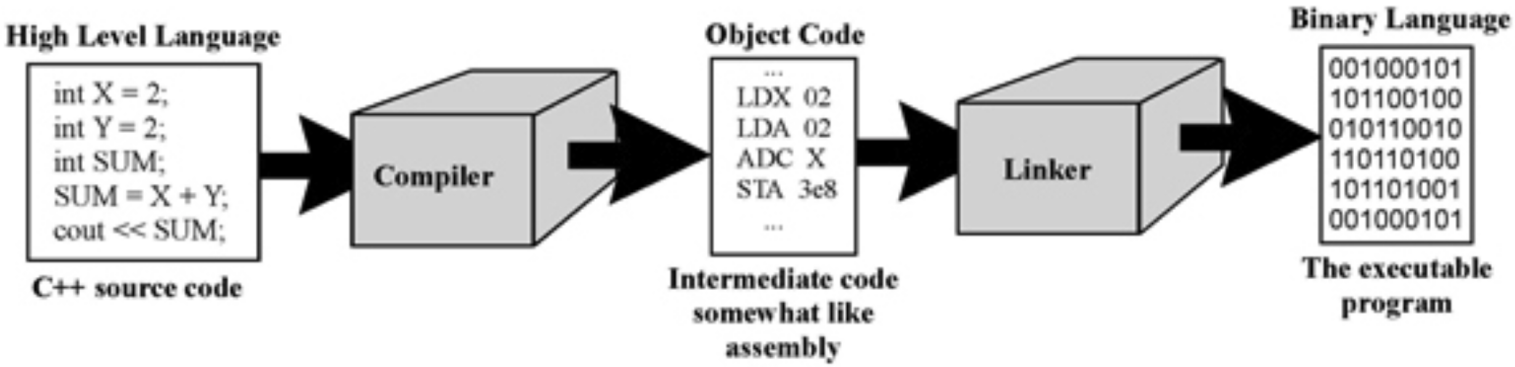
(b) Margaret Hamilton and her code for NASA's Apollo Program



(c) Era of AI and Cloud

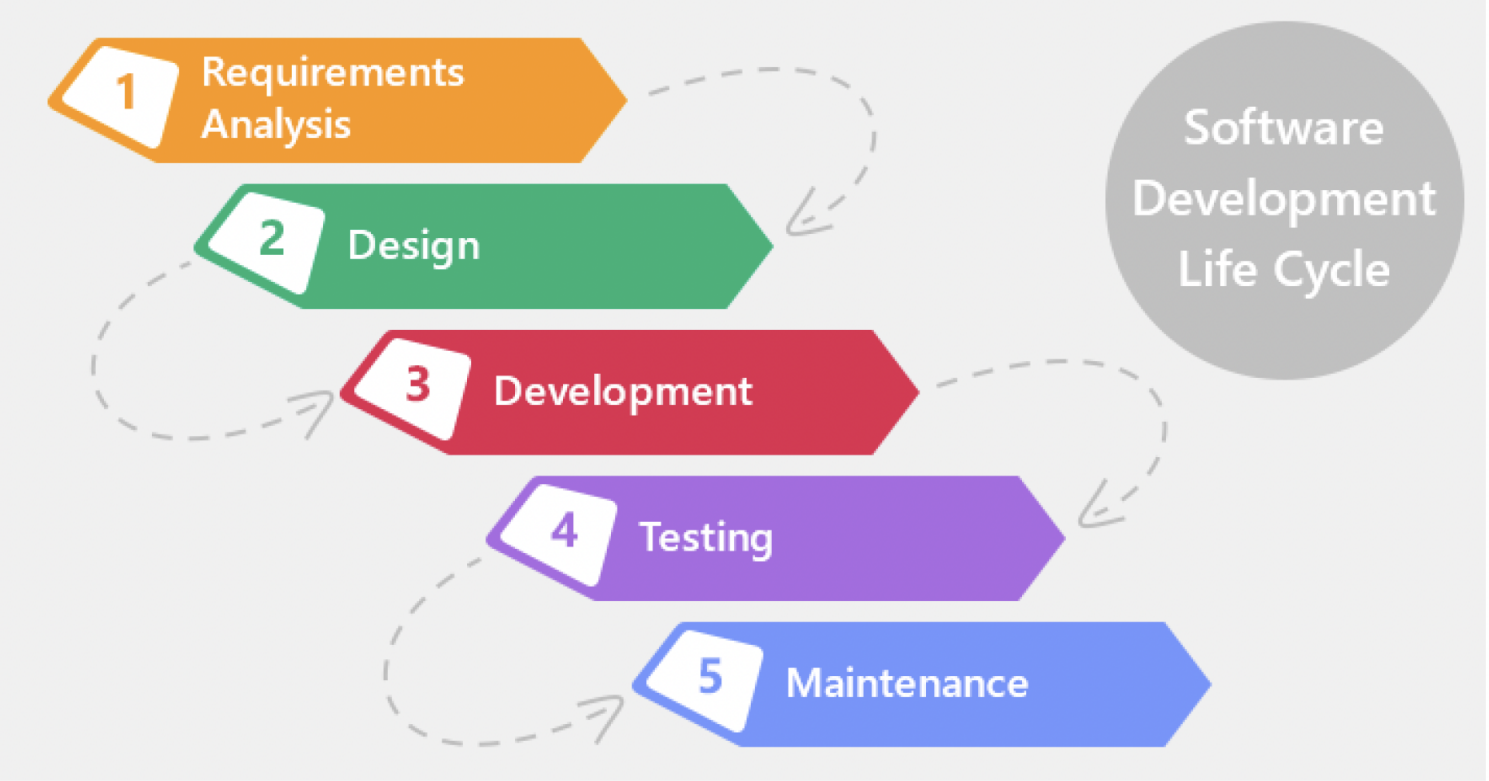
How your code is being executed?

- **Interpreted languages:** source code is directly run on an interpreter, there is a program that runs the code statements.
- **Compiled Languages:**
  - Compiler: translates source code (your code) to machine language (objecte code)
  - Linker: puts various object code files together into an executable program
  - Examples: `C`, `C++`, `Java`



Software Development

Involves more than just writing code



Software Development

- Analysis and problem denition
- Design - includes design of program or system structure, algorithms, user-interfaces, and more
- Implementation (coding)
- Testing - can be done during design, during implementation,and after implementation
- Maintenance - usually the major cost of a software system. Not part of "development", but denitely part of the software life cycle

Programming is about Problem Solving

- Algorithm - a finite sequence of steps to perform a specific task
  - To solve a problem, you have to come up with the necessary step-by-step process before you can code it
  - This is often the trickiest part of programming
- Some useful tools and techniques for formulating an algorithm
  - Top-down Refinement: Decomposing a task into smaller and simpler steps, then breaking down each step in to smaller steps, etc.
  - Pseudocode: Writing algorithms informally in a mixture of natural language and general types of code statements
  - Flowcharting: If you can visualize it, it's often easier to follow and understand.

Programming is about Problem Solving

- Testing - algorithms must also be tested!
  - Does it do what is required?
  - Does it handle all possible situations?
- Syntax *vs.* Semantics
  - Syntax - the grammar of a language "I is a Pikachu"
  - Semantics - the meaning of language constructs "The elevator ate my phone."

Basic Creation and Execution of a C++ program

FSU

- Create **source code** with a text editor
  - Source code is just a plain text file, usually given a filename extension to identify the programming language (like .c for C, or .cpp for C++)
- **Preprocessor** - Part of compiler process, performs any pre-processing tasks on source code.
- **Compilation** - syntax checking, creation of object code. Object code is the machine code translation of the source code.
- **Linking** - Final stage of the creation of an executable program. Linking of object code files together with any necessary libraries (also already compiled).
- **Execution of program** - Program loaded into memory, usually RAM, CPU executes code instructions.