An Overview of Computers and Programming Languages

Chapter # 1

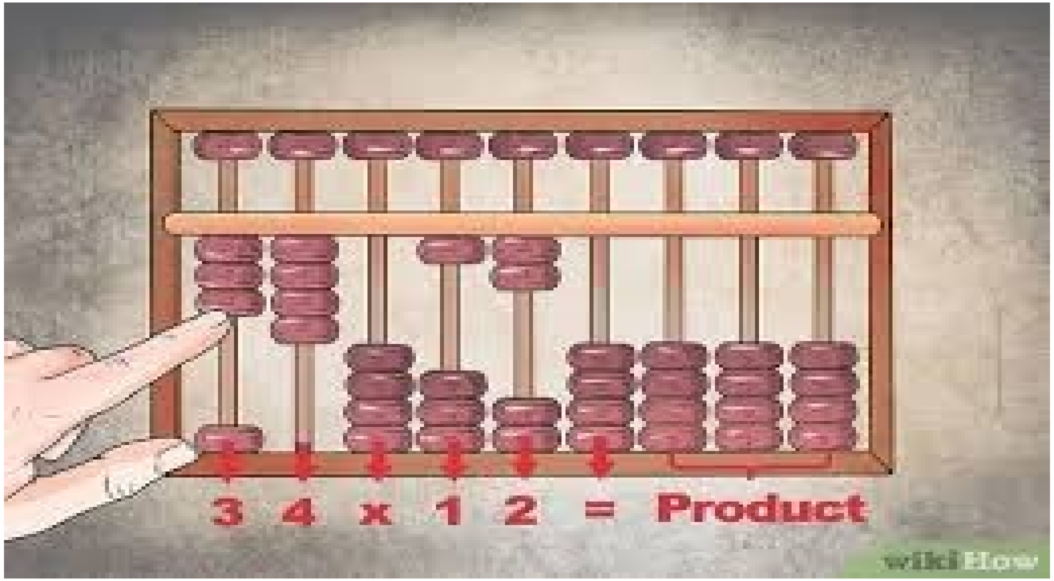
Instructor: Sadullah Karimi, Msc in CSE

Today Agenda

- Learn about different types of computers
- Explore the hardware and software components of a computer system
- Learn about the language of a computer
- Learn about the evolution of programming languages
- Examine high-level programming languages
- Discover what a compiler is and what it does
- Examine a C++ program
- Explore how a C11 program is processed
- Learn what an algorithm is and explore problem-solving techniques
- Become aware of structured-design and object-oriented design programming methodologies
- Become aware of ANSI/ISO Standard C11, C1111, C1114

The Abacus

- Origins: Invented in Asia, used in Babylon, China, and Europe.
- Function: Utilizes sliding beads on a rack.
- Operations: Performs addition and subtraction.

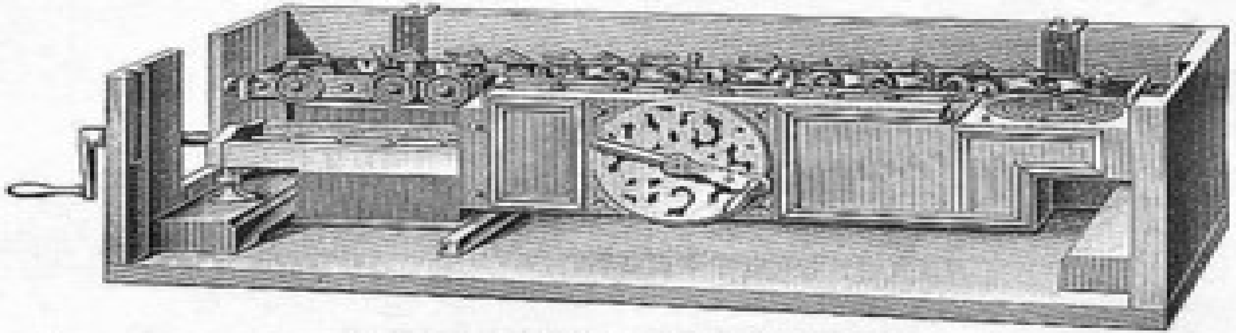


The Pascaline

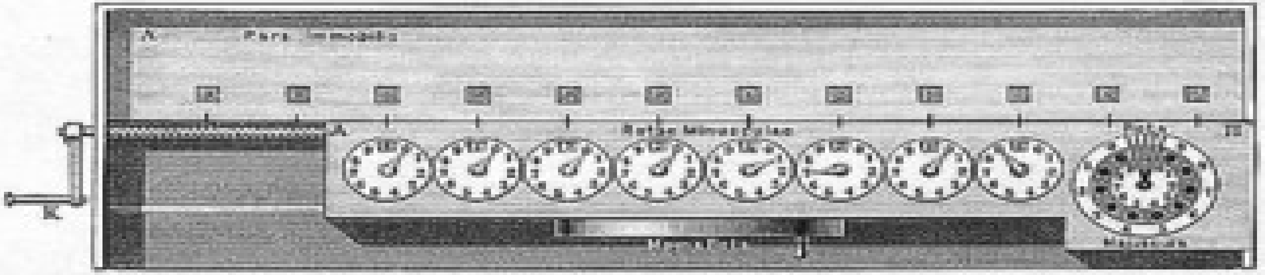
- Inventor: Blaise Pascal (1642)
- Features: Eight movable dials on wheels.
- Function: Calculates sums up to eight figures long.
- Operations: Addition and subtraction.

Leibniz's Calculator

- Inventor: Gottfried von Leibniz (17th Century)
- Capabilities: Addition, subtraction, multiplication, and division.



2. Rechenmaschine von Leibniz (1673, Hannover).



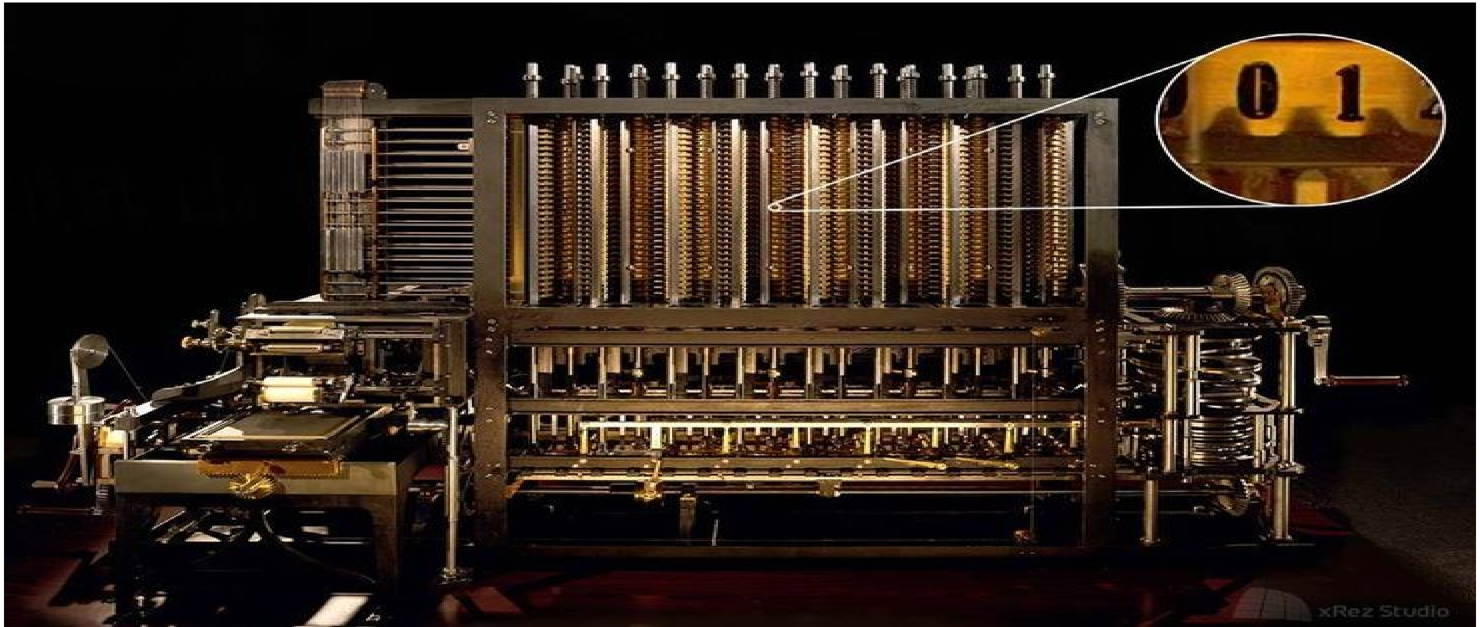
3. Leibnizsche Rechenmaschine, geometrische Zeichnung.

Punched Card Looms

- Inventor: Joseph Jacquard (1819)
- Function: Weaving instructions stored on punched cards.
- Impact: Concept of storing information on punched cards.

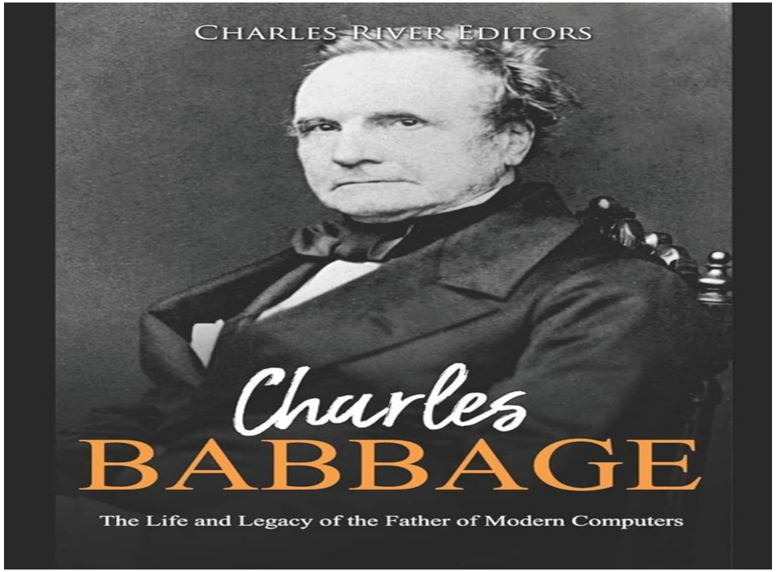
Charles Babbage’s Designs

- Inventions: Difference Engine and Analytical Engine (Early-Mid 1800s)
- Difference Engine: Performs complex operations like squaring numbers.
- Completion: First complete Difference Engine finished in 2002.



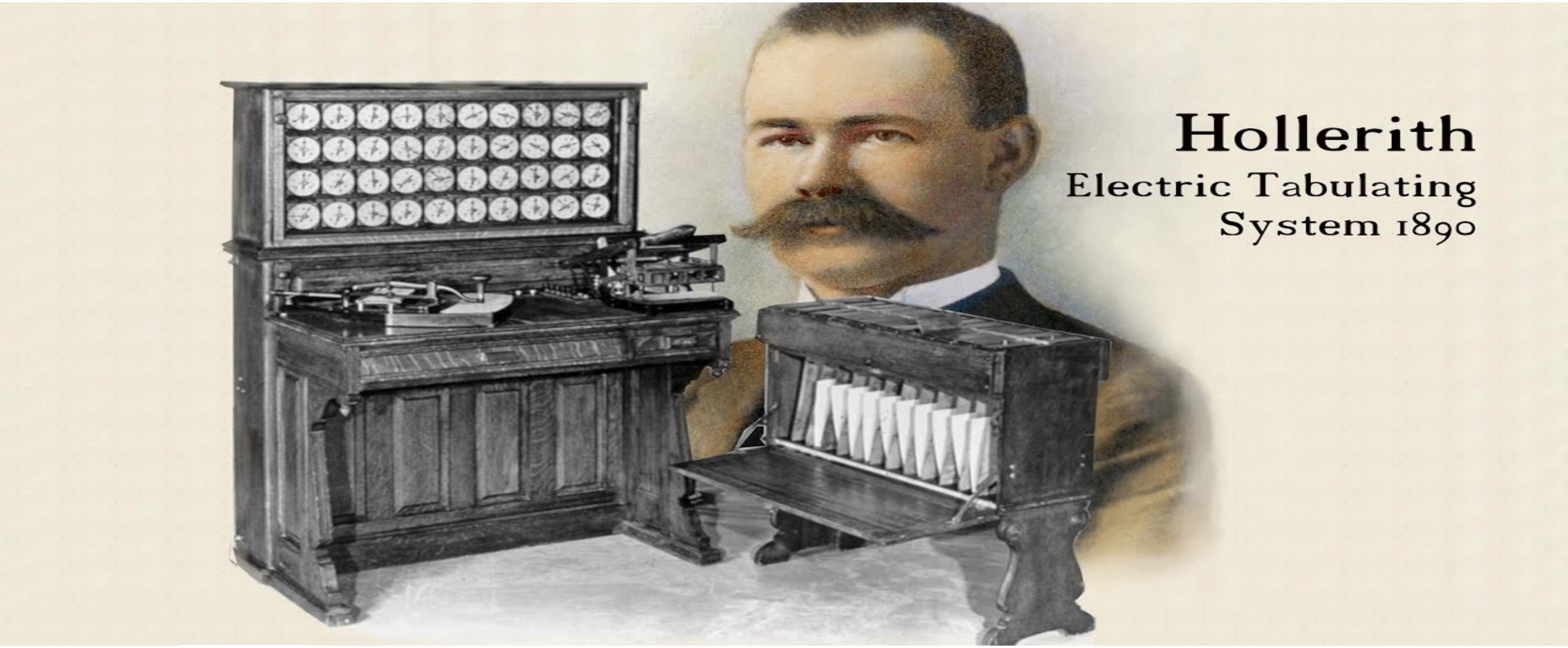
Legacy of Babbage

- Collaborator: Ada Augusta, Countess of Lovelace
- Contribution: Considered the first computer programmer.
- Museum: Replica of the Difference Engine displayed at the Computer History Museum, California.



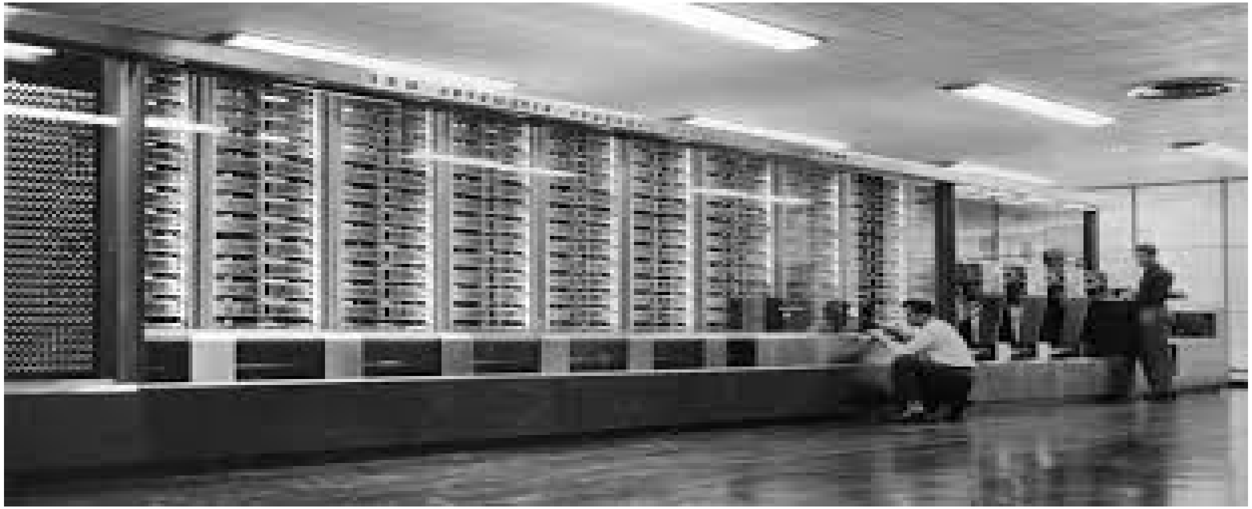
Herman Hollerith and Early Computers

- Invention: Calculating machine using punched cards.
- Impact: Helped accurately tabulate census data.
- Legacy: Founded the Tabulating Machine Company, now IBM.



The Mark I

- Year: 1944
- Developers: IBM and Harvard University (Howard Aiken)
- Specifications: 52 feet long, 50 tons, 750,000 parts.
- Data Input: Punched cards.

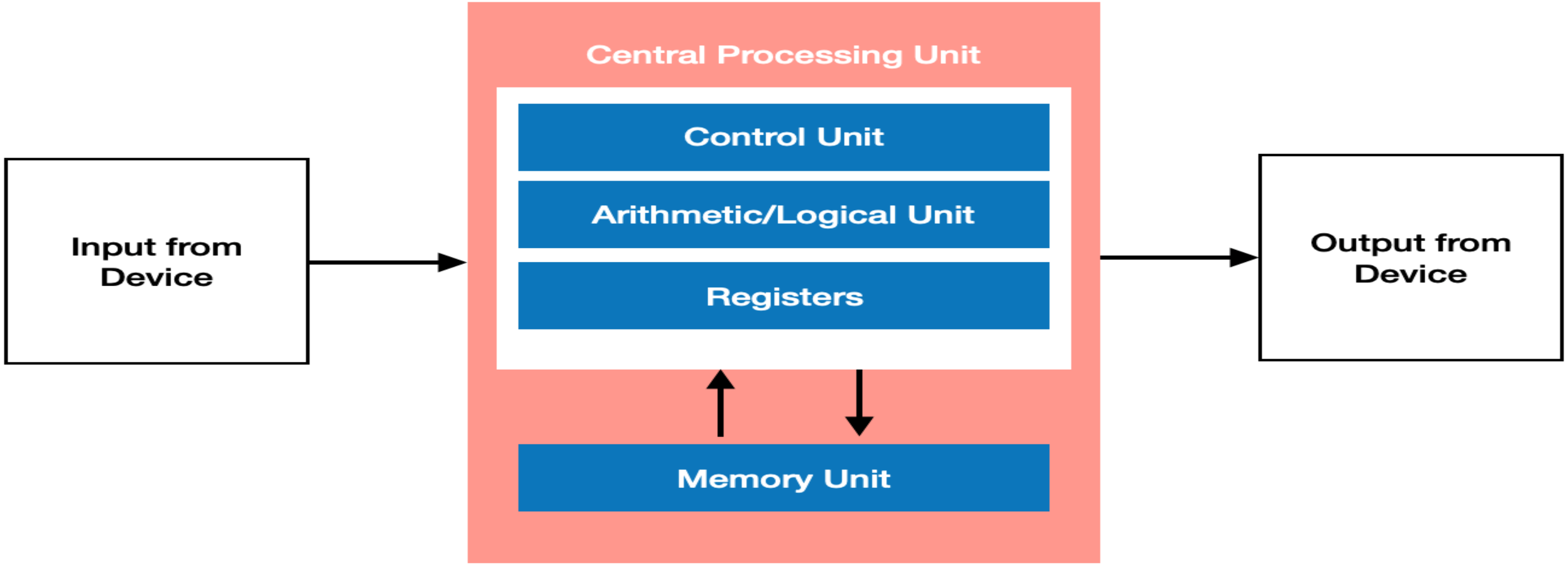


ENIAC

- Year: 1946
- Location: University of Pennsylvania
- Specifications: 18,000 vacuum tubes, 30 tons.
- Significance: Early electronic general-purpose computer.

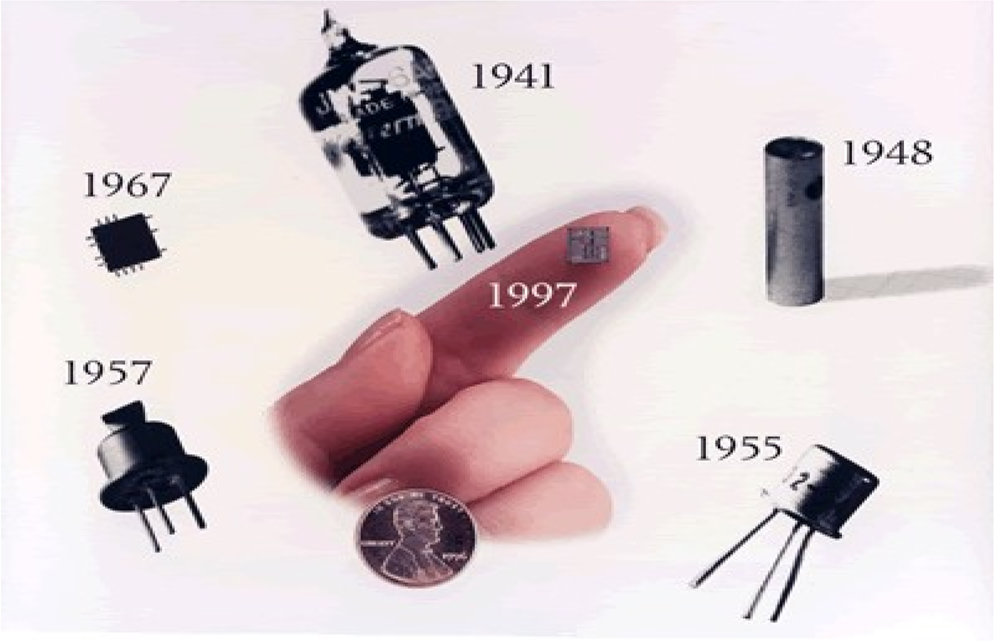
John von Neumann’s Architecture

- Design (Late 1940s): Arithmetic logic unit, control unit, memory, input/output devices.
- Key Innovation: Stored-program concept.
- First Computer Built: UNIVAC (1951), sold to U.S. Census Bureau.



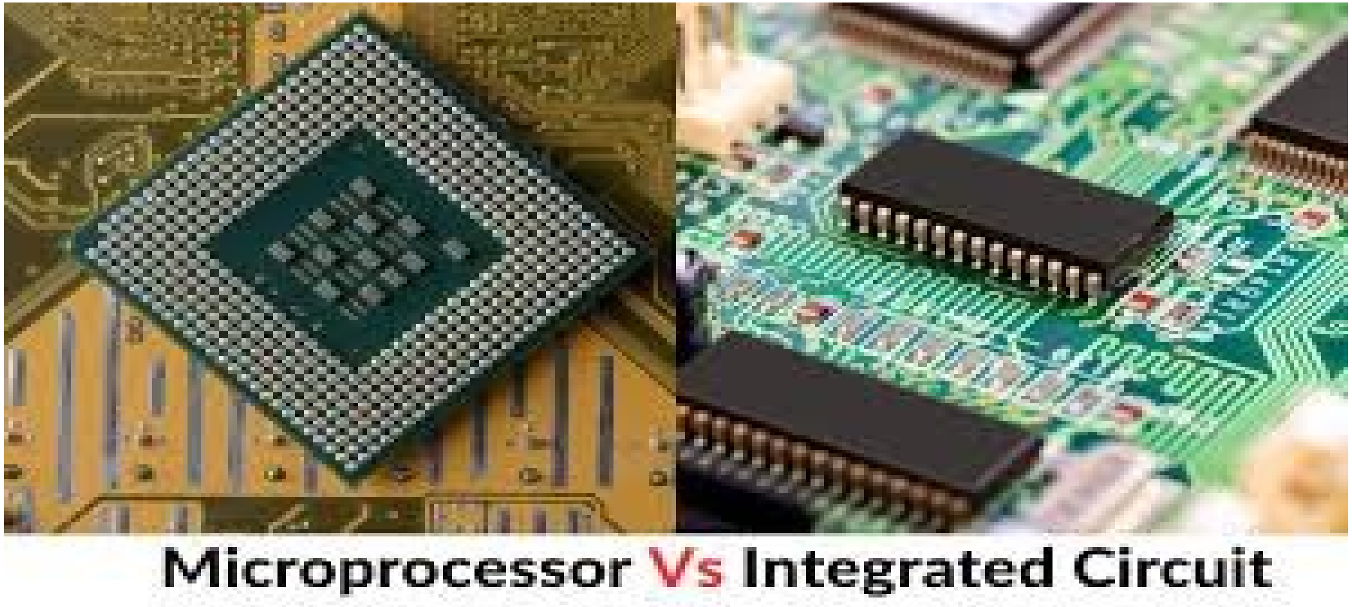
The Era of Transistors

- Year: 1956
- Impact: Smaller, faster, more reliable, and energy-efficient computers.
- Software Development: Introduction of FORTRAN and COBOL.



Integrated Circuits and Microprocessors

- Development: Transistors replaced by integrated circuits (chips).
- Advantage: Chips are smaller and more efficient; contain thousands of circuits.
- Year of Microprocessor Invention: 1970



The Personal Computer Revolution

- First Apple Computer: Designed by Stephen Wozniak and Steven Jobs (1977).
- IBM PC Introduction: 1981.
- Impact: Affordable personal computing, widespread adoption by mid-1990s.

Modern Computing Advances

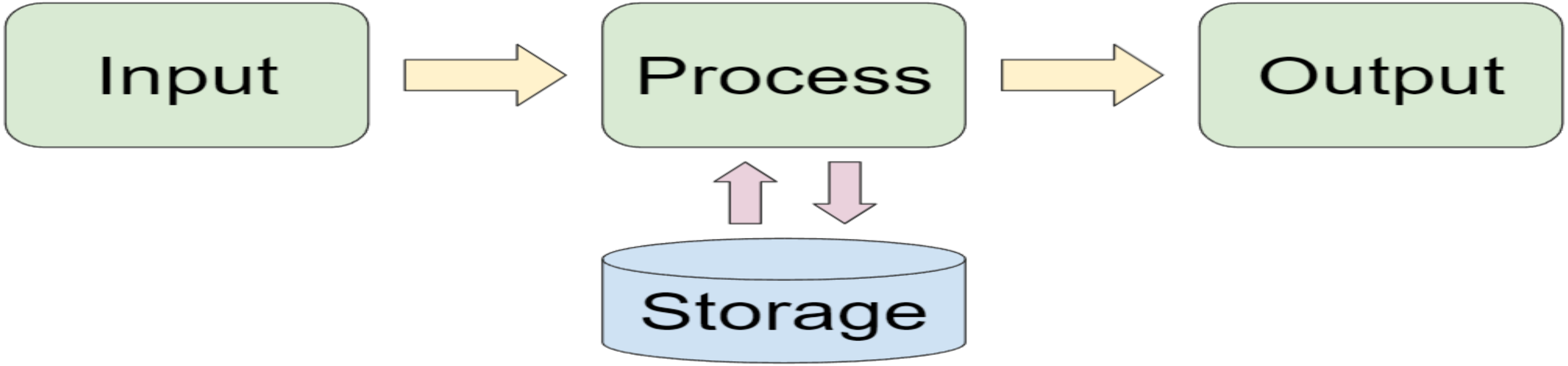
- Technological Improvements: Faster, more affordable, and powerful computers.
- Features: Spoken-word instructions, artificial intelligence, expert systems.
- Applications: Mobile computing, GPS, social media, banking.

Categories of Computers

- Types: Mainframe, midsize, and microcomputers.
- Basic Elements: Shared across categories (e.g., processors, memory).

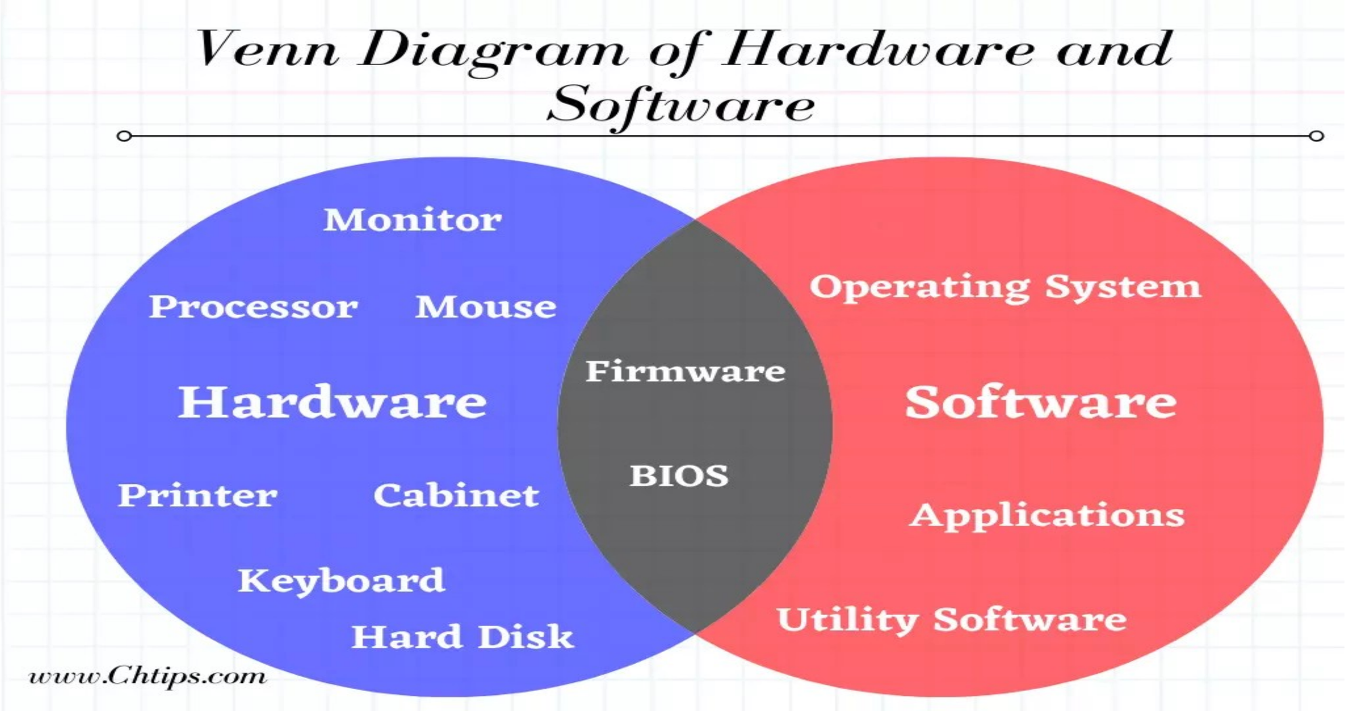
Elements of a Computer System

- Definition: An electronic device capable of performing commands.
- Basic Commands:
- Input: Receive data.
- Output: Display results.
- Storage: Save data.
- Arithmetic and Logical Operations: Perform calculations and logical decisions.



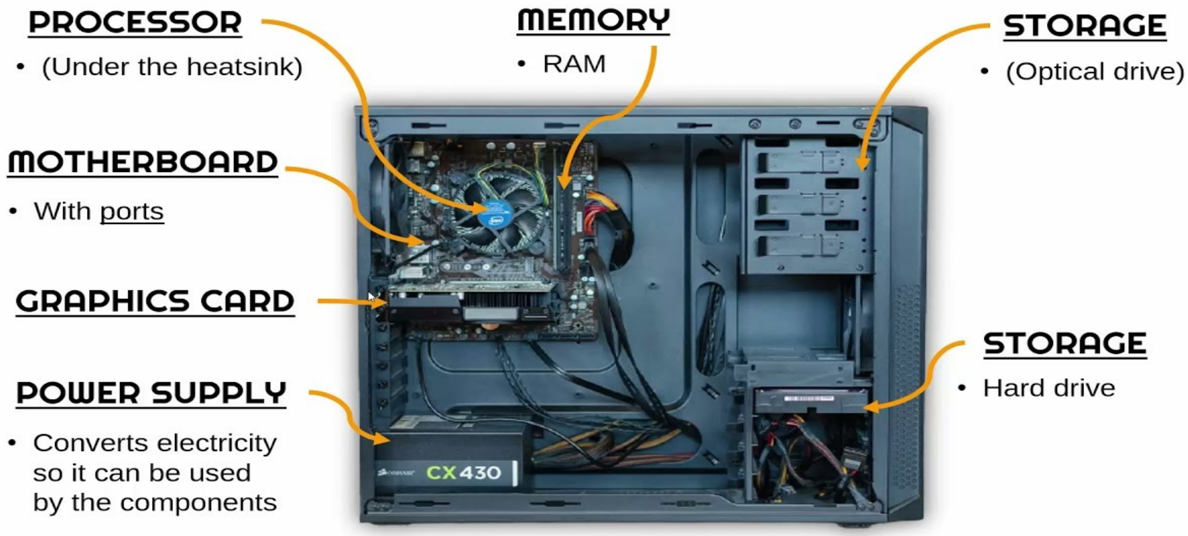
Components of a Computer System

- Main Components:
- Hardware: Physical parts of the computer.
- Software: Programs and applications.



Overview of Hardware

- Definition: The physical parts of a computer system.
- Components Include:
- Central Processing Unit (CPU): The brain of the computer.
- Memory: RAM (Random Access Memory), cache.
- Storage Devices: Hard drives, SSDs.
- Input Devices: Keyboard, mouse.
- Output Devices: Monitor, printer.



PROCESSOR
- (Under the heatsink)

MEMORY
- RAM

STORAGE
- (Optical drive)

MOTHERBOARD
- With ports

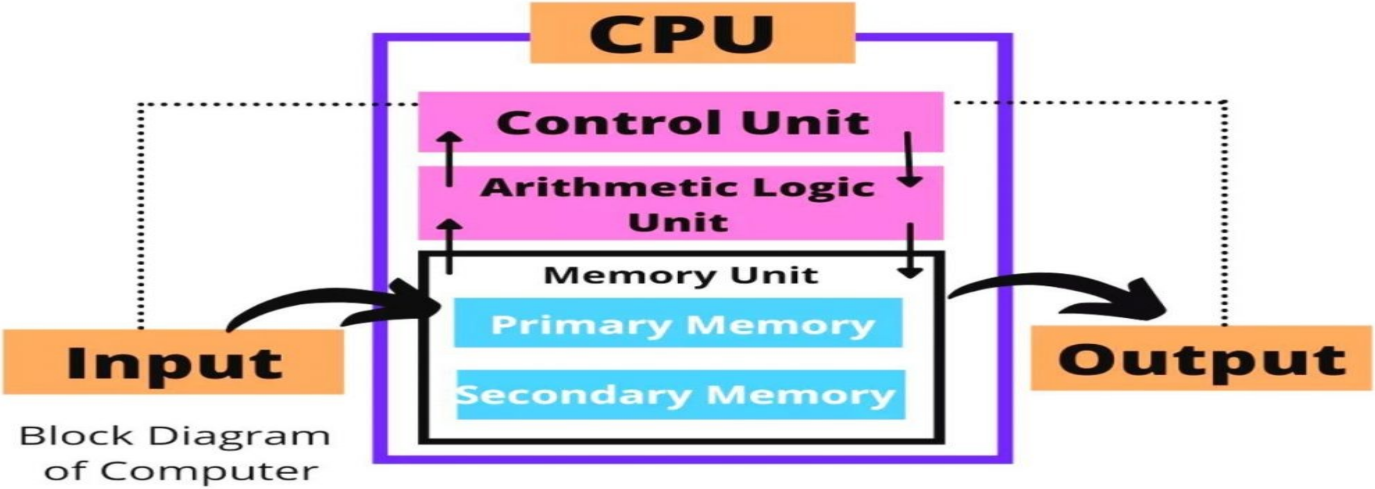
GRAPHICS CARD

STORAGE
- Hard drive

POWER SUPPLY
- Converts electricity so it can be used by the components

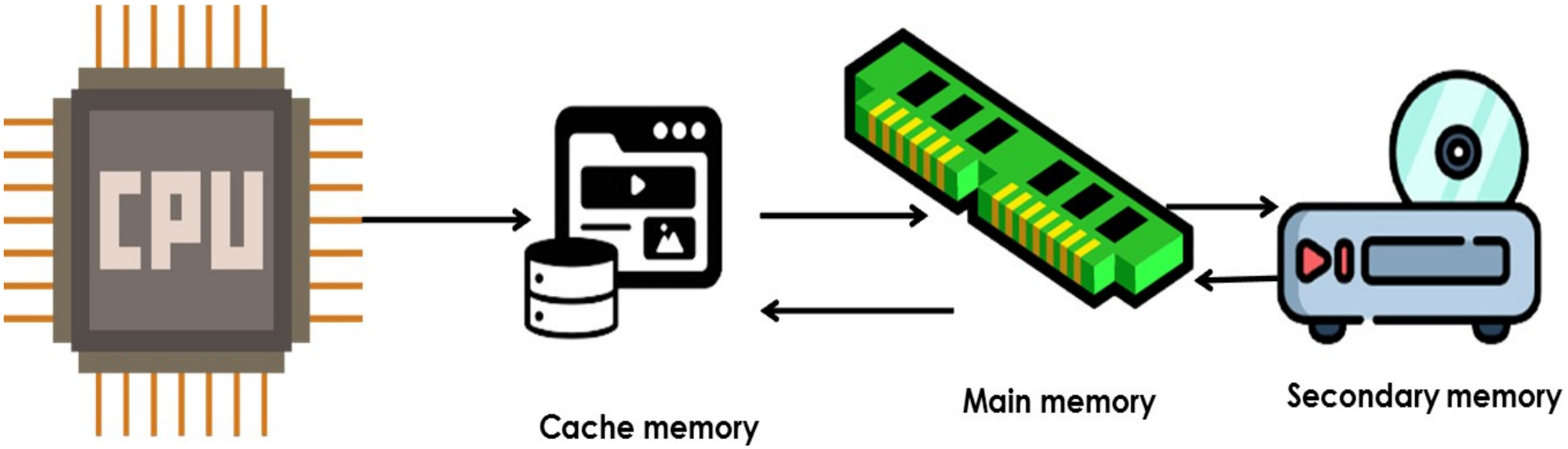
Central Processing Unit (CPU)

- Function: Executes instructions from software.
- Components:
- Arithmetic Logic Unit (ALU): Performs arithmetic and logical operations.
- Control Unit (CU): Directs operations and processes instructions.



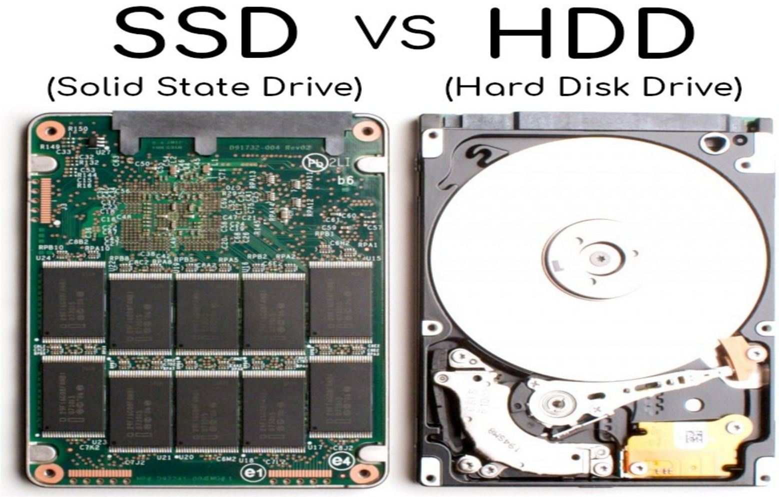
Memory

- Types of Memory:
- RAM (Random Access Memory): Temporary storage used for current tasks.
- Cache: Fast, small memory that speeds up access to frequently used data.



Storage Devices

- Types of Storage:
- Hard Disk Drives (HDD): Traditional spinning disks.
- Solid State Drives (SSD): Faster, newer technology with no moving parts.

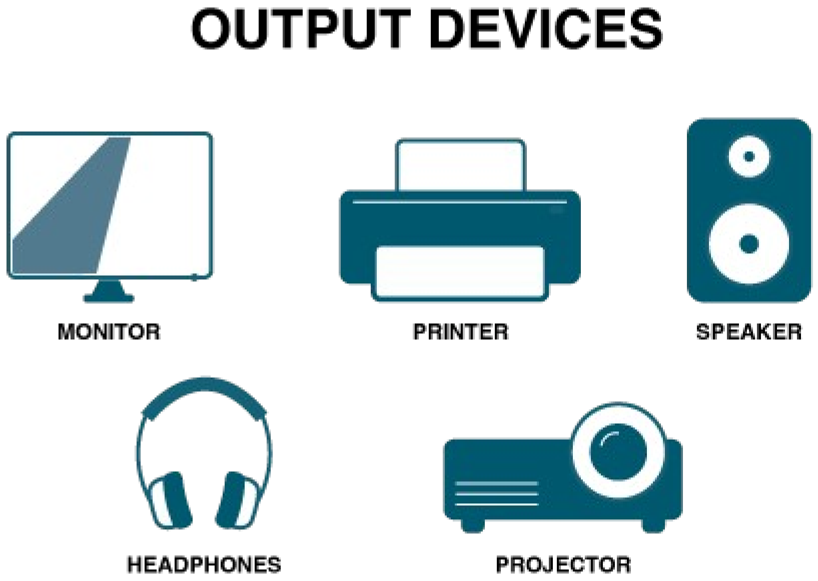


Input Devices

- Examples:
- Keyboard: For typing data.
- Mouse: For pointing and selecting.
- Others: Scanner, microphone.

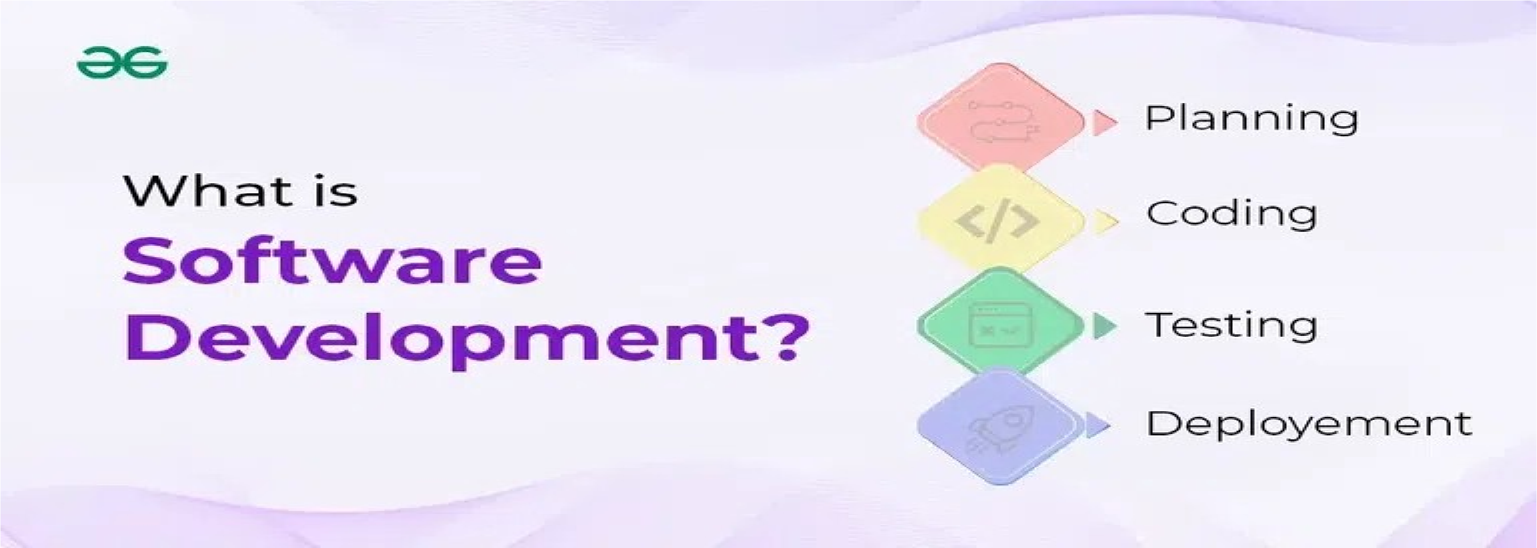
Output Devices

- Examples:
- Monitor: Displays visual output.
- Printer: Produces physical copies of digital documents.
- Speakers: For audio output.



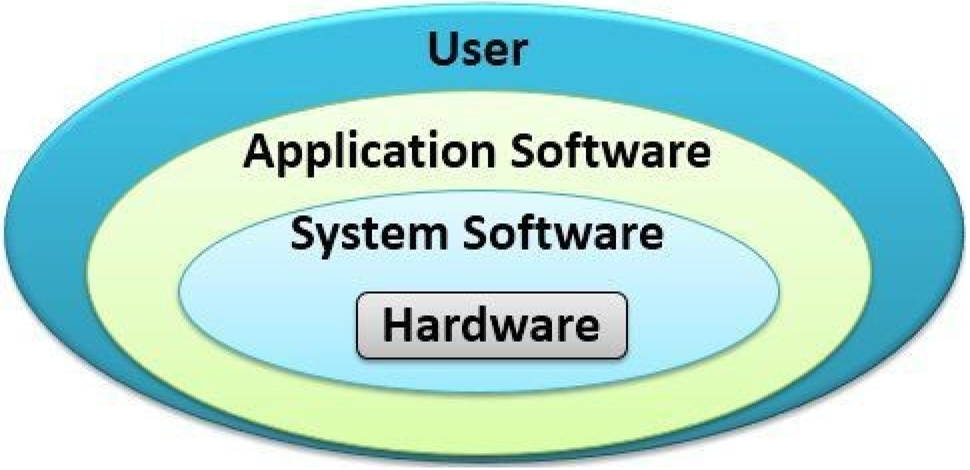
What is Software?

- Definition: Programs written to perform specific tasks.
- Example: Word processors for writing letters, papers, books.
- Written In: Programming languages.



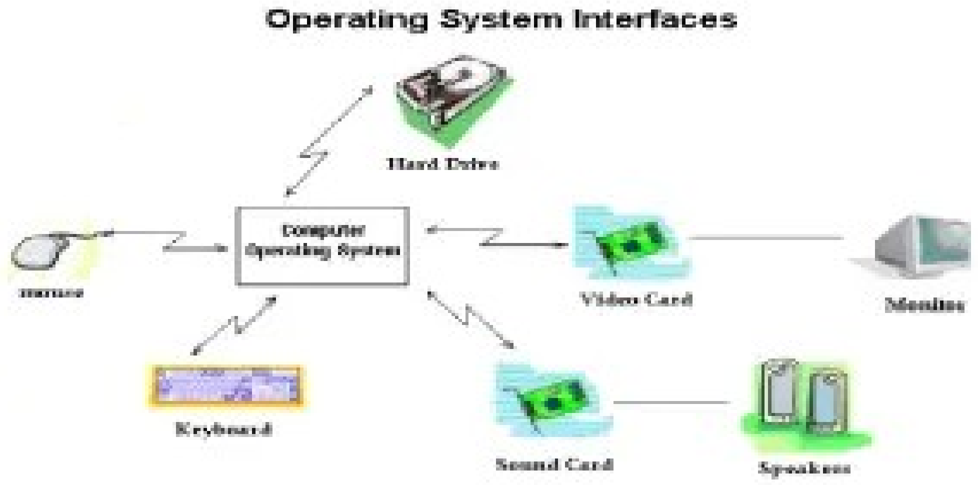
Types of Programs

- System Programs: Control and manage the computer.
- Application Programs: Perform specific tasks for users.



System Programs

- Definition: Programs that control and manage computer hardware and software.
- Key Component: Operating System (OS).
- Function:
- Memory management.
- Input/output activities.
- Storage management.

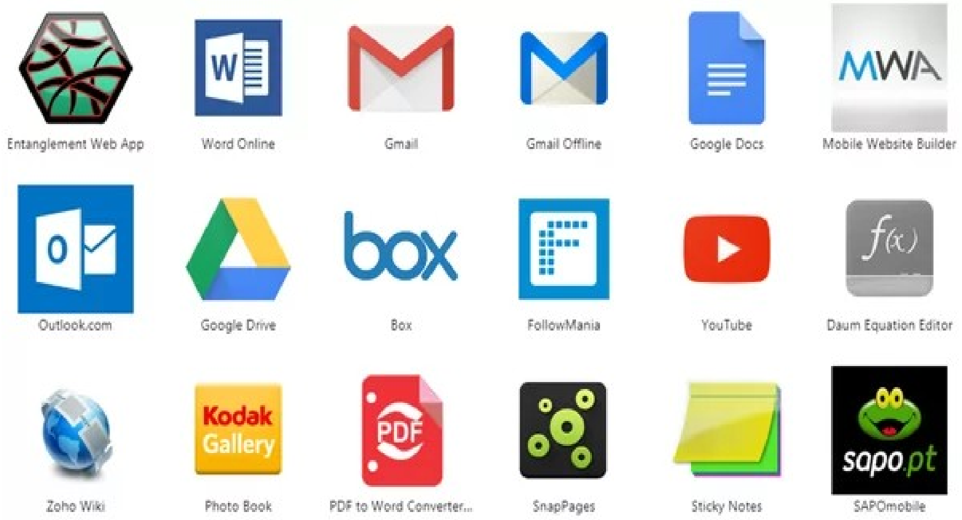


Operating System

- Definition: The system program that loads first when the computer starts.
- Importance: Without an OS, the computer is non-functional.
- Functions:
- Manages hardware resources.
- Provides user services.
- Organizes secondary storage.
- Examples: Windows 10, Mac OS X, Linux, Android.

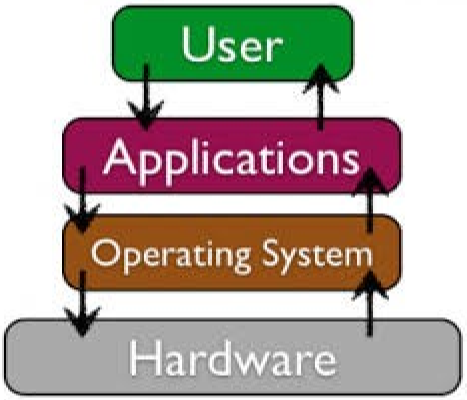
Application Programs

- Definition: Programs designed to perform specific tasks for users.
- Examples:
- Word Processors: Write letters, papers.
- Spreadsheets: Perform calculations and manage data.
- Games: Entertainment.
- Role: Run on top of the operating system.



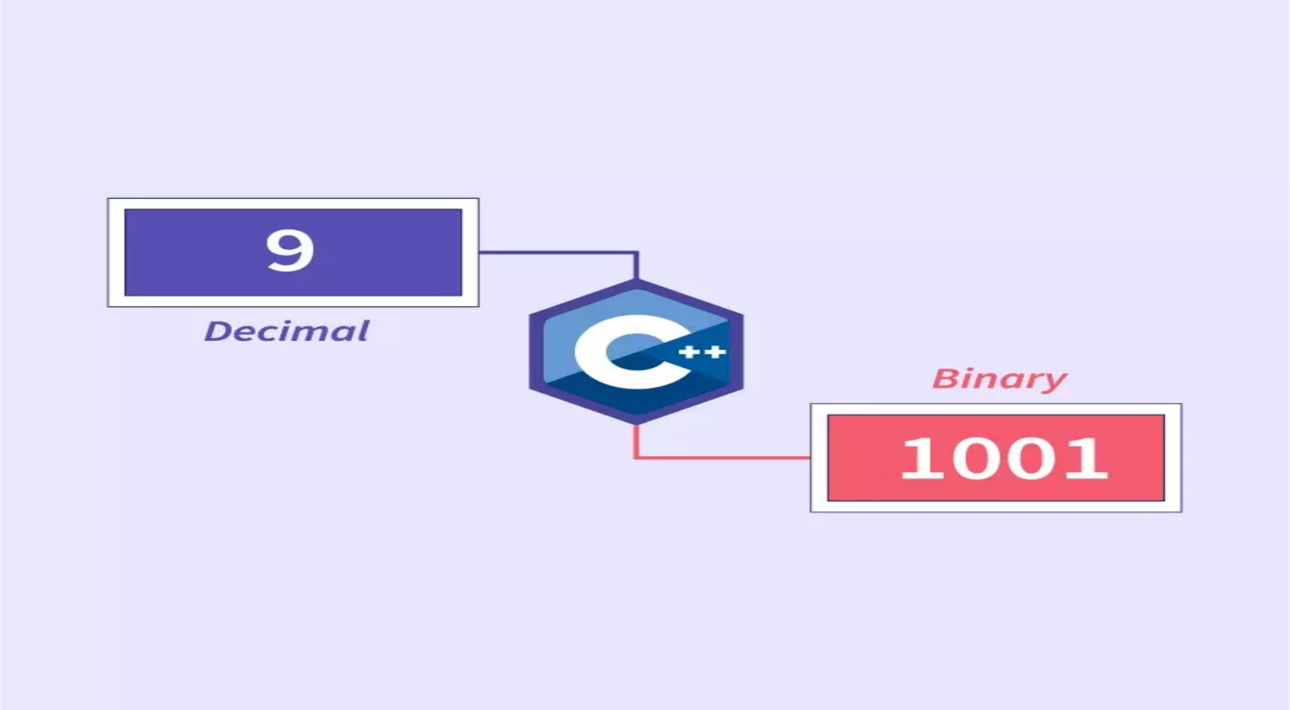
The Relationship Between OS and Applications

- Operating System: Manages hardware and provides a platform for applications.
- Applications: Utilize OS services to perform specific tasks.



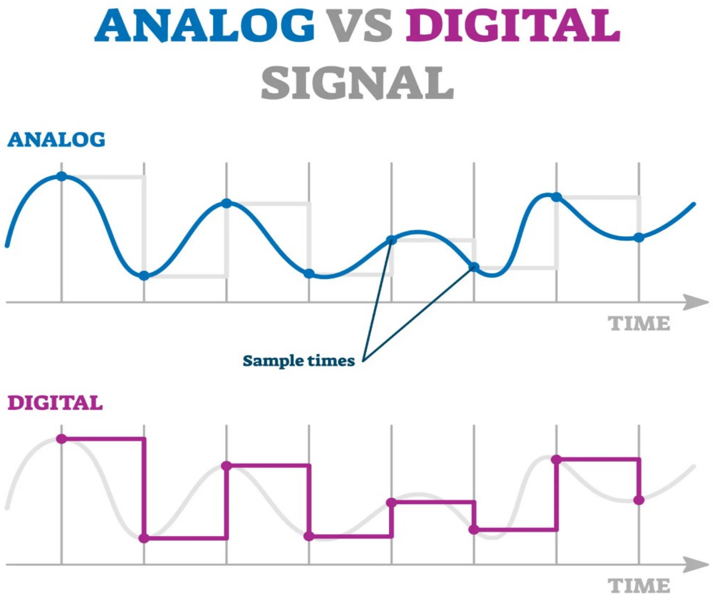
Computer Language Basics

- What You See: Pressing 'A' on the keyboard displays 'A' on the screen.
- What Happens Inside: Stored as binary data in the computer's memory.



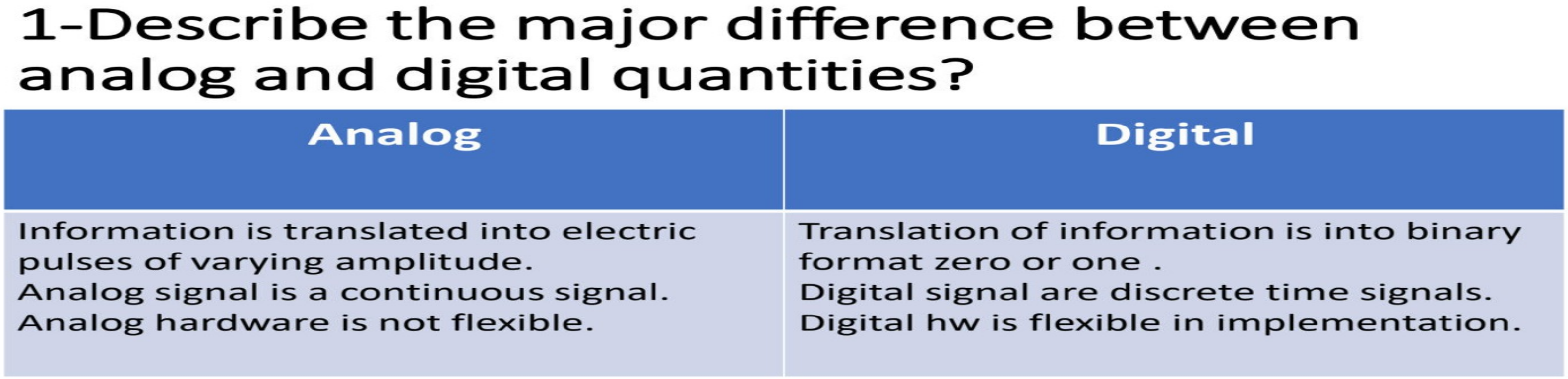
Electrical Signals

- Analog Signals:
- Continuously varying waveforms.
- Represent continuous data like sound.
- Example: Audio tapes.
- Digital Signals:
- Represent information as sequences of 0s and 1s.
- 0 = Low voltage, 1 = High voltage.
- More reliable and precise.



Why Digital Signals?

- Advantages of Digital Signals:
- Reliable transmission of information.
- Exact copies can be made (e.g., CD vs. audio tape).

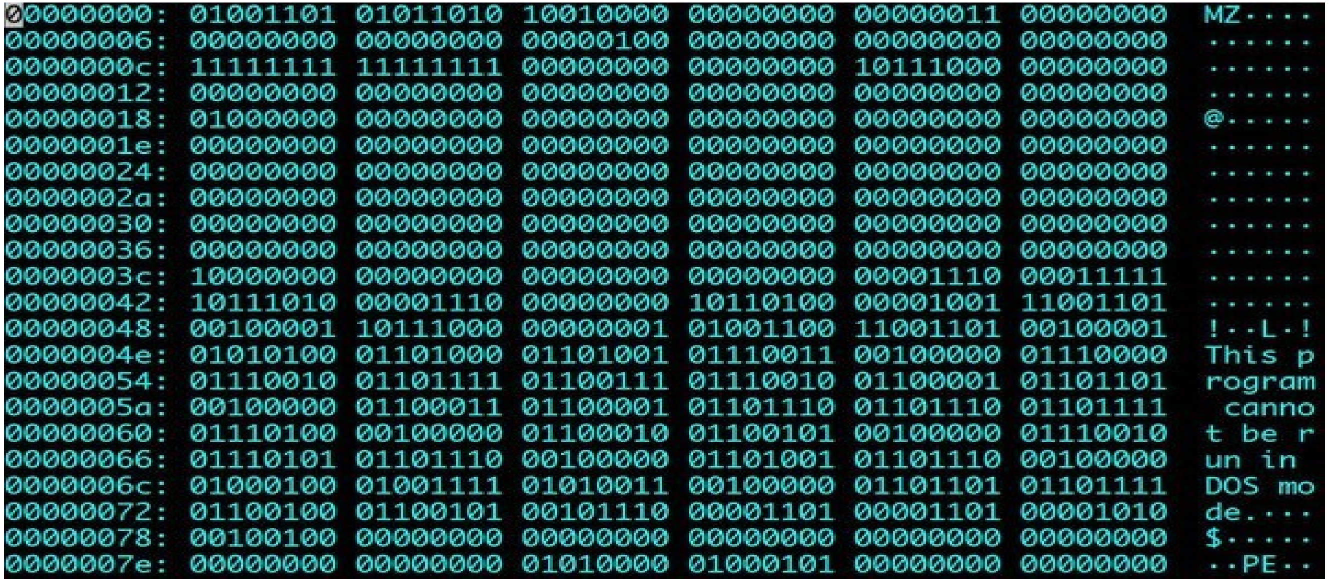


1-Describe the major difference between analog and digital quantities?

Analog	Digital
Information is translated into electric pulses of varying amplitude. Analog signal is a continuous signal. Analog hardware is not flexible.	Translation of information is into binary format zero or one . Digital signal are discrete time signals. Digital hw is flexible in implementation.

Machine Language

- Definition: The language of a computer, consisting of sequences of 0s and 1s.
- Binary Digit (Bit): The basic unit of data (0 or 1).
- Binary Code: Sequence of bits used to represent data.



```
00000000: 01001101 01011010 10010000 00000000 00000011 00000000  MZ····
00000006: 00000000 00000000 00000100 00000000 00000000 00000000  ······
0000000c: 11111111 11111111 00000000 00000000 10111000 00000000  ······
00000012: 00000000 00000000 00000000 00000000 00000000 00000000  ······
00000018: 01000000 00000000 00000000 00000000 00000000 00000000  @·····
0000001e: 00000000 00000000 00000000 00000000 00000000 00000000  ······
00000024: 00000000 00000000 00000000 00000000 00000000 00000000  ······
0000002a: 00000000 00000000 00000000 00000000 00000000 00000000  ······
00000030: 00000000 00000000 00000000 00000000 00000000 00000000  ······
00000036: 00000000 00000000 00000000 00000000 00000000 00000000  ······
0000003c: 10000000 00000000 00000000 00000000 00001110 00011111  ······
00000042: 10111010 00001110 00000000 10110100 00001001 11001101  ······
00000048: 00100001 10111000 00000001 01001100 11001101 00100001  !··L·!
0000004e: 01010100 01101000 01101001 01110011 00100000 01110000  This p
00000054: 01110010 01101111 01100111 01110010 01100001 01101101  rogram
0000005a: 00100000 01100011 01100001 01101110 01101110 01101111   canno
00000060: 01110100 00100000 01100010 01100101 00100000 01110010  t be r
00000066: 01110101 01101110 00100000 01101001 01101110 00100000  un in
0000006c: 01000100 01001111 01010011 00100000 01101101 01101111  DOS mo
00000072: 01100100 01100101 00101110 00001101 00001101 00001010  de.···
00000078: 00100100 00000000 00000000 00000000 00000000 00000000  $·····
0000007e: 00000000 00000000 01010000 01000101 00000000 00000000  ··PE··
```

Bits and Bytes

- Bit: A binary digit (0 or 1).
- Byte: A sequence of 8 bits.
- Kilobyte (KB): 1,024 bytes (2^10 bytes).

Summary of Byte Units

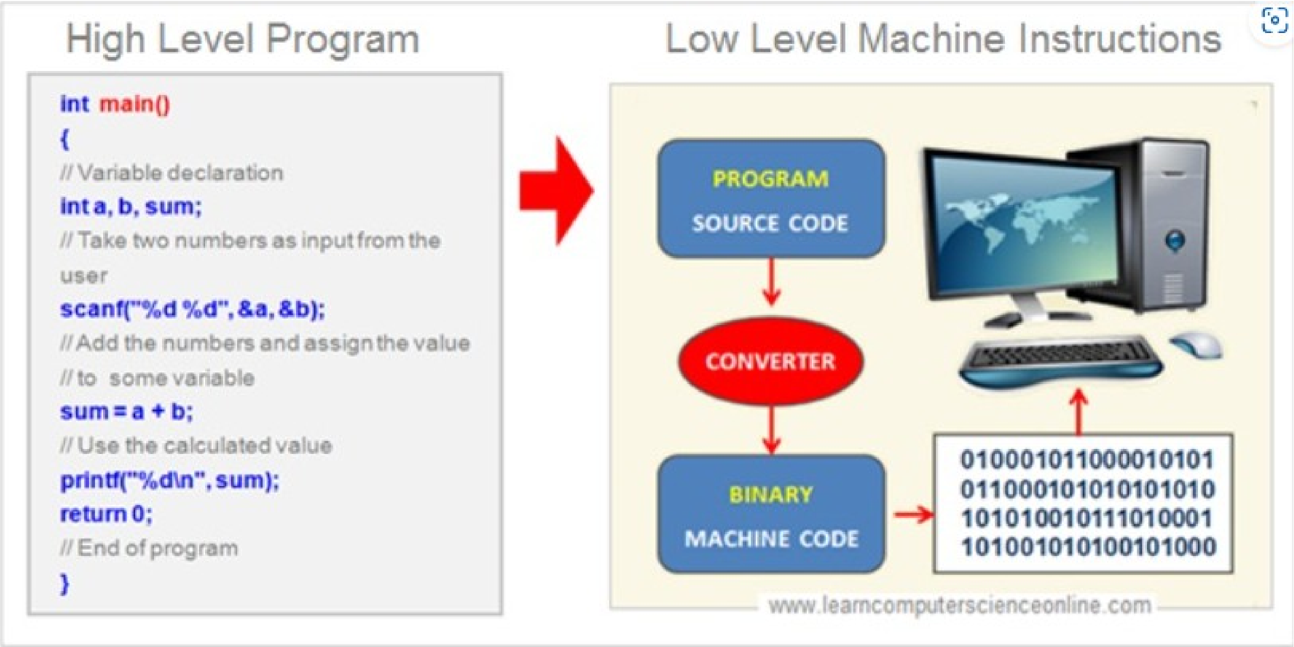
- Bits: 0 or 1.
- Bytes: 8 bits.
- Kilobyte (KB): 1,024 bytes.

TABLE 1-1 Binary Units

Unit	Symbol	Bits/Bytes
Byte		8 bits
Kilobyte	KB	2^{10} bytes = 1024 bytes
Megabyte	MB	1024 KB = 2^{10} KB = 2^{20} bytes = 1,048,576 bytes
Gigabyte	GB	1024 MB = 2^{10} MB = 2^{30} bytes = 1,073,741,824 bytes
Terabyte	TB	1024 GB = 2^{10} GB = 2^{40} bytes = 1,099,511,627,776 bytes
Petabyte	PB	1024 TB = 2^{10} TB = 2^{50} bytes = 1,125,899,906,842,624 bytes
Exabyte	EB	1024 PB = 2^{10} PB = 2^{60} bytes = 1,152,921,504,606,846,976 bytes
Zettabyte	ZB	1024 EB = 2^{10} EB = 2^{70} bytes = 1,180,591,620,717,411,303,424 bytes

Machine Language

- Definition: The most basic language of a computer, consisting of binary code (0s and 1s).
- Characteristics: Each computer may have different binary codes for operations.
- Example Instruction:
- wages = rate · hours
- Binary codes:
- Load: 100100
- Multiply: 100110
- Store: 100010



High Level Program

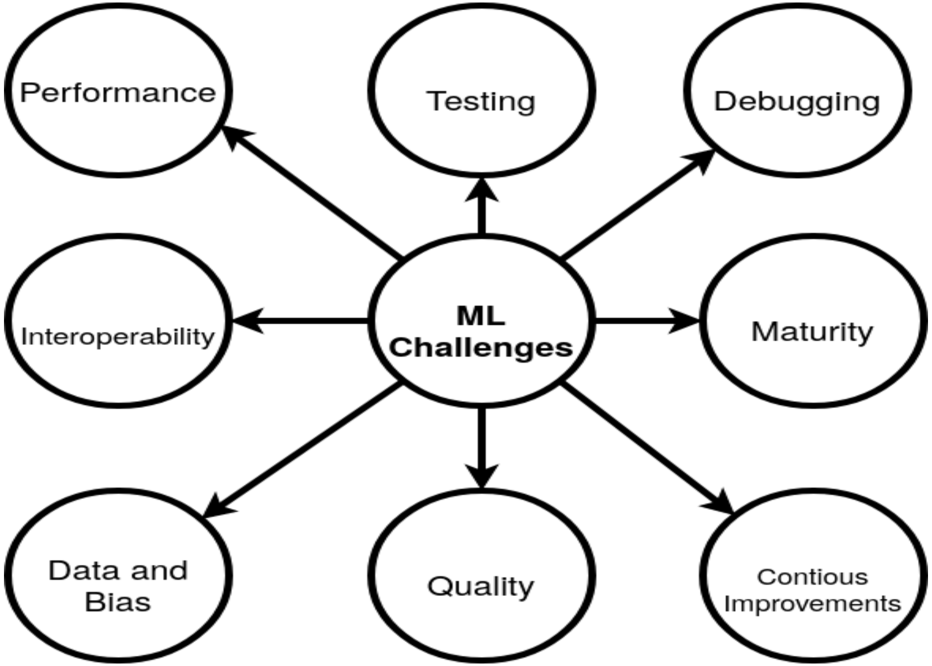
```
int main()
{
// Variable declaration
int a, b, sum;
// Take two numbers as input from the
user
scanf("%d %d", &a, &b);
// Add the numbers and assign the value
// to  some variable
sum = a + b;
// Use the calculated value
printf("%d\n", sum);
return 0;
// End of program
}
```

Low Level Machine Instructions

www.learncomputerscienceonline.com

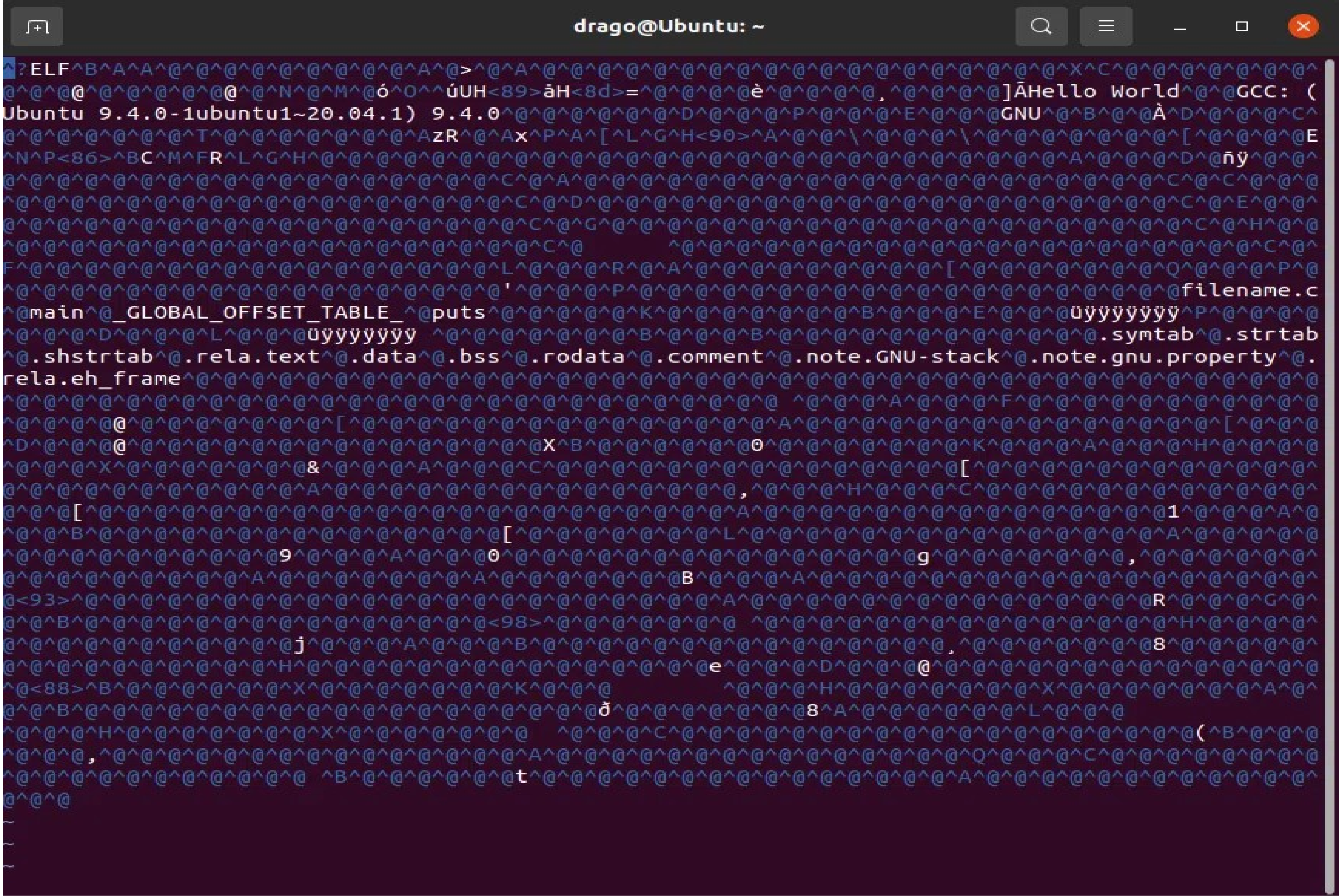
Challenges of Machine Language

- Complexity: Programmers had to remember binary codes for operations.
- Error-Prone: Difficulty in remembering specific codes and memory locations.



Assembly Language

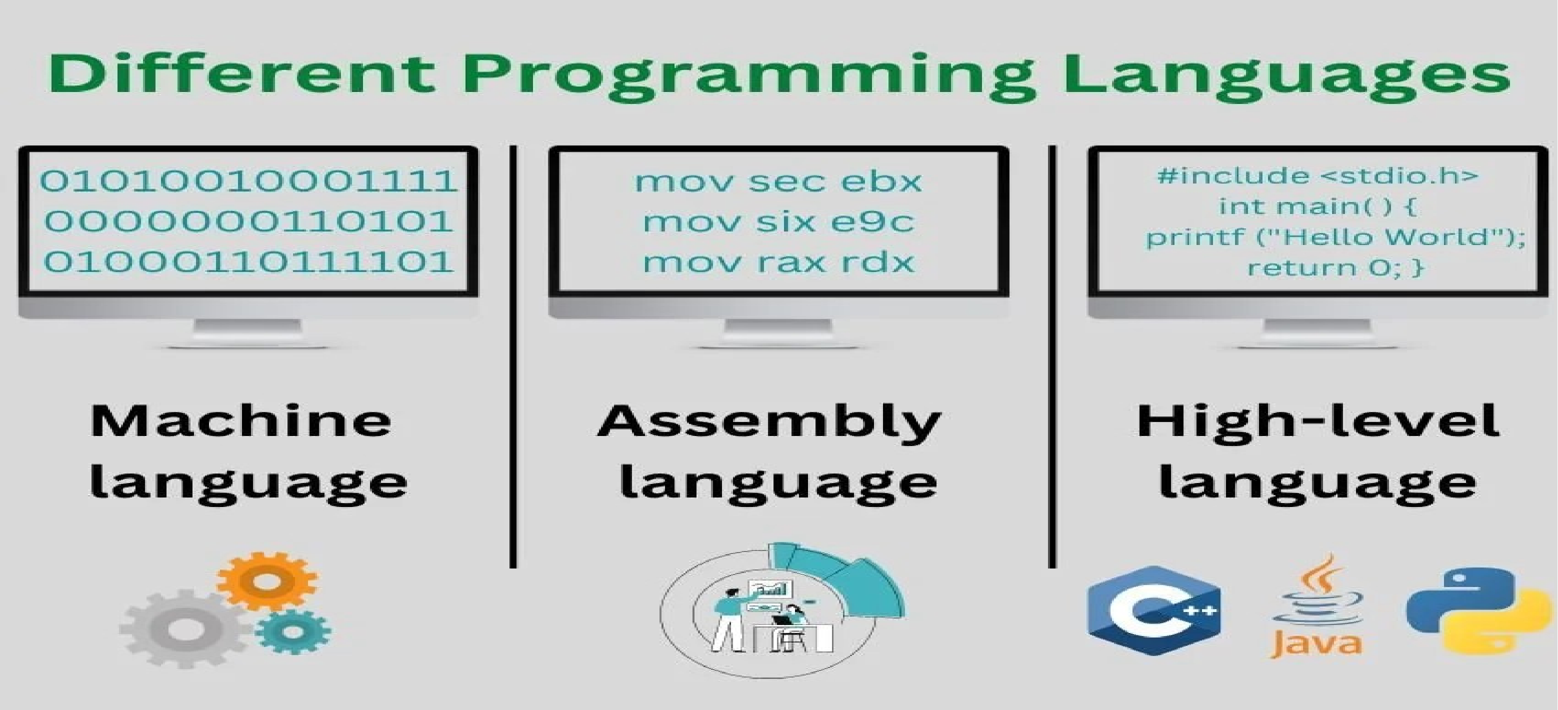
- Definition: A step above machine language, using mnemonics instead of binary codes.
- Example Instructions:
- LOAD rate
- MULT hours
- STOR wages



```
^?ELF^B^A^A^@^@^@^@^@^@^@^@^@^A^@>^@^A^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^X^C^@^@^@^@^@^@^@^
@^@^@@^@^@^@^@^@@^@^N^@^M^@ó^O^^úUH<89>åH<8d>=^@^@^@^@è^@^@^@^@¸^@^@^@^@]ÃHello World^@^@GCC: (
Ubuntu 9.4.0-1ubuntu1~20.04.1) 9.4.0^@^@^@^@^@^@^D^@^@^@^P^@^@^@^E^@^@^@GNU^@^B^@^@À^D^@^@^@^C^
@^@^@^@^@^@^@^T^@^@^@^@^@^@^@^AzR^@^Ax^P^A^[^L^G^H<90>^A^@^@^\^@^@^@^\^@^@^@^@^@^@^@^[^@^@^@^@E
^N^P<86>^BC^M^FR^L^G^H^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^A^@^@^@^D^@ñÿ^@^@^
@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^C^@^A^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^C^@^C^@^@^@
^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^C^@^D^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^C^@^E^@^@^
@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^C^@^G^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^C^@^H^@^@
^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^C^@      ^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^C^@^
F^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^L^@^@^@^R^@^A^@^@^@^@^@^@^@^@^@^[^@^@^@^@^@^@^@^Q^@^@^@^P^@
^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@'^@^@^@^P^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@filename.c
^@main^@_GLOBAL_OFFSET_TABLE_^@puts^@^@^@^@^@^K^@^@^@^@^@^@^@^B^@^@^@^E^@^@^@üÿÿÿÿÿÿÿ^P^@^@^@^@
^@^@^@^D^@^@^@^L^@^@^@üÿÿÿÿÿÿÿ ^@^@^@^@^@^@^@^B^@^@^@^B^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@.symtab^@.strtab
^@.shstrtab^@.rela.text^@.data^@.bss^@.rodata^@.comment^@.note.GNU-stack^@.note.gnu.property^@.
rela.eh_frame^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@
^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@ ^@^@^@^A^@^@^@^F^@^@^@^@^@^@^@^@^@^@^@
^@^@^@^@@^@^@^@^@^@^@^@^[^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^A^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^[^@^@^@
^D^@^@^@@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@X^B^@^@^@^@^@^@0^@^@^@^@^@^@^@^K^@^@^@^A^@^@^@^H^@^@^@^@
^@^@^@^X^@^@^@^@^@^@^@&^@^@^@^A^@^@^@^C^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@[^@^@^@^@^@^@^@^@^@^@^@^@^
@^@^@^@^@^@^@^@^@^@^A^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@,^@^@^@^H^@^@^@^C^@^@^@^@^@^@^@^@^@^@^@^@^
@^@^@[^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^A^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@1^@^@^@^A^@
^@^@^B^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@[^@^@^@^@^@^@^@^L^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^A^@^@^@^@^@
^@^@^@^@^@^@^@^@^@^@9^@^@^@^A^@^@^@0^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@g^@^@^@^@^@^@^@,^@^@^@^@^@^@^
@^@^@^@^@^@^@^@^@^A^@^@^@^@^@^@^@^A^@^@^@^@^@^@^@B^@^@^@^A^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^
@<93>^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^A^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@R^@^@^@^G^@^
@^@^B^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@<98>^@^@^@^@^@^@^@ ^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^H^@^@^@^@^
@^@^@^@^@^@^@^@^@^@^@j^@^@^@^A^@^@^@^B^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@¸^@^@^@^@^@^@^@8^@^@^@^@^@^
@^@^@^@^@^@^@^@^@^@^H^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@e^@^@^@^D^@^@^@@^@^@^@^@^@^@^@^@^@^@^@^@^@^@
^@<88>^B^@^@^@^@^@^@^X^@^@^@^@^@^@^@^K^@^@^@        ^@^@^@^H^@^@^@^@^@^@^@^X^@^@^@^@^@^@^@^A^@^
@^@^B^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@ð^@^@^@^@^@^@^@8^A^@^@^@^@^@^@^L^@^@^@
^@^@^@^H^@^@^@^@^@^@^@^X^@^@^@^@^@^@^@  ^@^@^@^C^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@(^B^@^@^@
^@^@^@,^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^A^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^Q^@^@^@^C^@^@^@^@^@^@^@^@
^@^@^@^@^@^@^@^@^@^@^@ ^B^@^@^@^@^@^@t^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^A^@^@^@^@^@^@^@^@^@^@^@^@^
@^@^@
~
~
~
```

Advantages of Assembly Language

- Ease of Use: Easier to write and understand than machine language.
- Translation: Requires an assembler to convert to machine language.



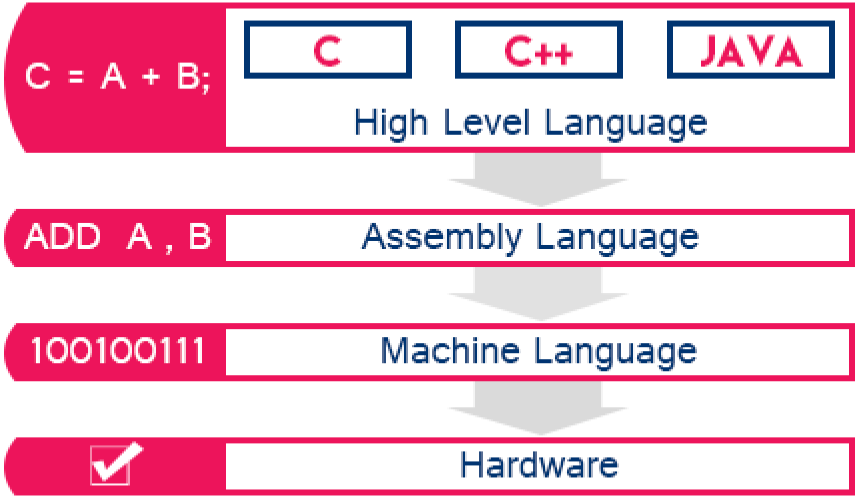
```
01010010001111
0000000110101
01000110111101
```

```
mov sec ebx
mov six e9c
mov rax rdx
```

```
#include <stdio.h>
int main( ) {
printf ("Hello World");
return 0; }
```

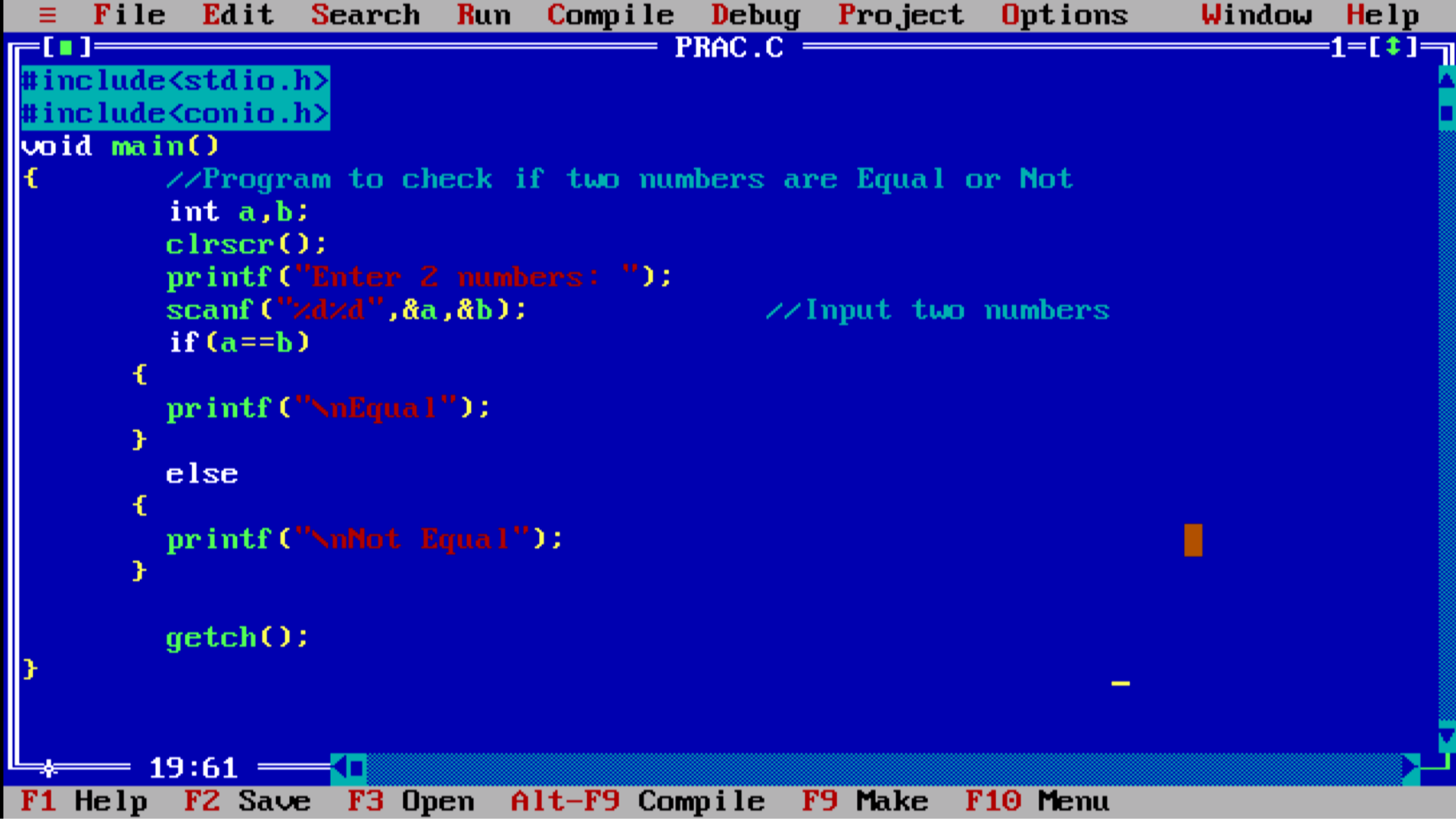
High-Level Languages

- Definition: Languages closer to human languages, making programming more intuitive.
- Examples: Basic, FORTRAN, COBOL, C, C++, C#, Java, Python.
- High-Level Language Instruction:
- wages = rate * hours; (in C++)



Benefits of High-Level Languages

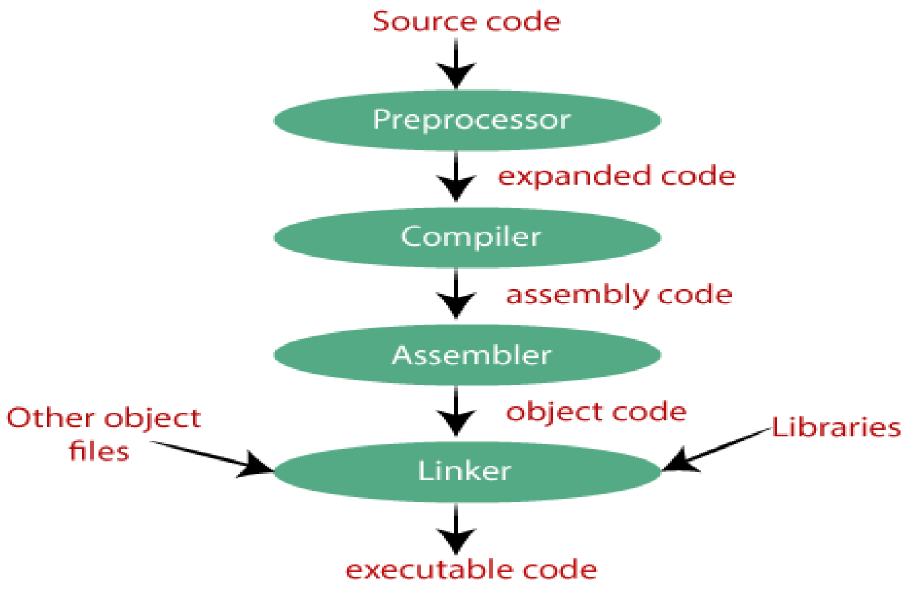
- Readability: Easier to understand and write, closer to natural language.
- Translation: Requires a compiler to convert into machine language.


≡ File Edit Search Run Compile Debug Project Options Window Help
[■]
PRAC.C
1=[↕]
19:61
F1 Help F2 Save F3 Open Alt-F9 Compile F9 Make F10 Menu

```c
#include<stdio.h>
#include<conio.h>
void main()
{       //Program to check if two numbers are Equal or Not
        int a,b;
        clrscr();
        printf("Enter 2 numbers: ");
        scanf("%d%d",&a,&b);             //Input two numbers
        if(a==b)
      {
        printf("\nEqual");
      }
        else
      {
        printf("\nNot Equal");
      }

        getch();
}
```

Compilers

- Definition: Programs that translate high-level language instructions into machine code.
- Function: Converts easier-to-write code into executable machine language.



Example C11 Program

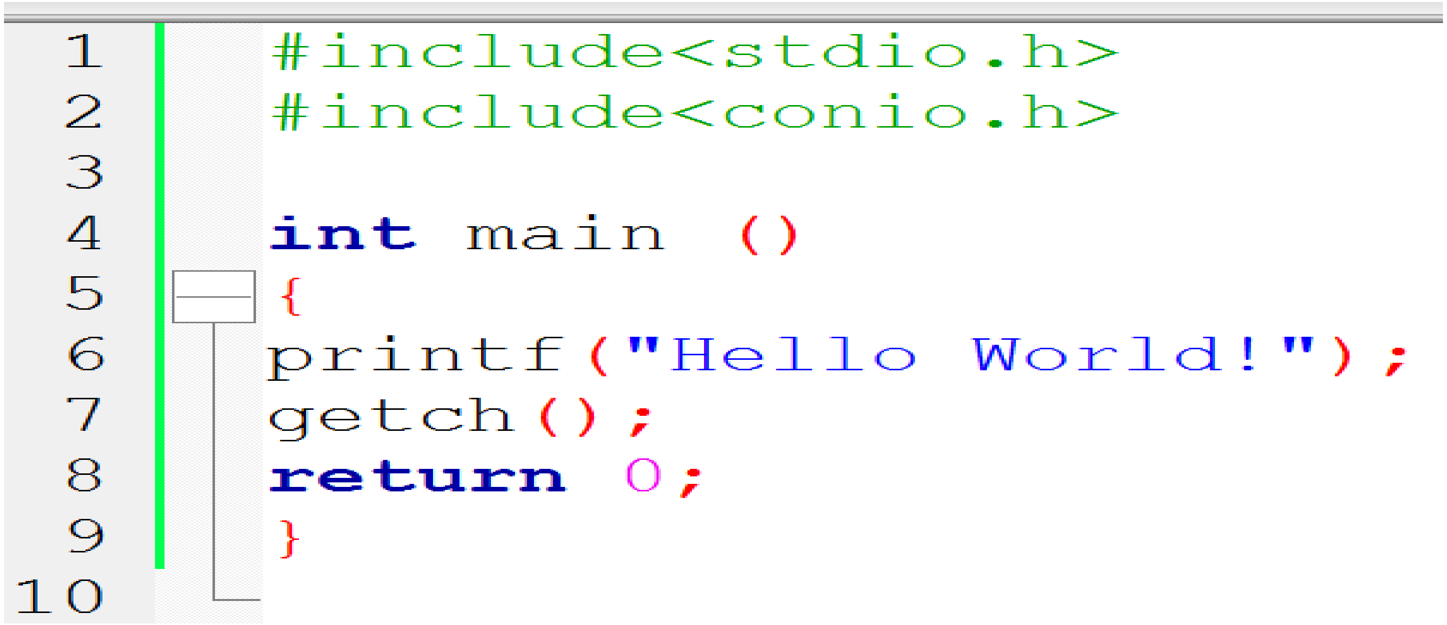
```
#include <iostream>
using namespace std;

int main() {
    cout << "My first C++ program." << endl;
    return 0;
}
```

Purpose: Displays "My first C++ program." on the screen.

Note: This code must be translated into machine language to be executed.

Writing the Source Code

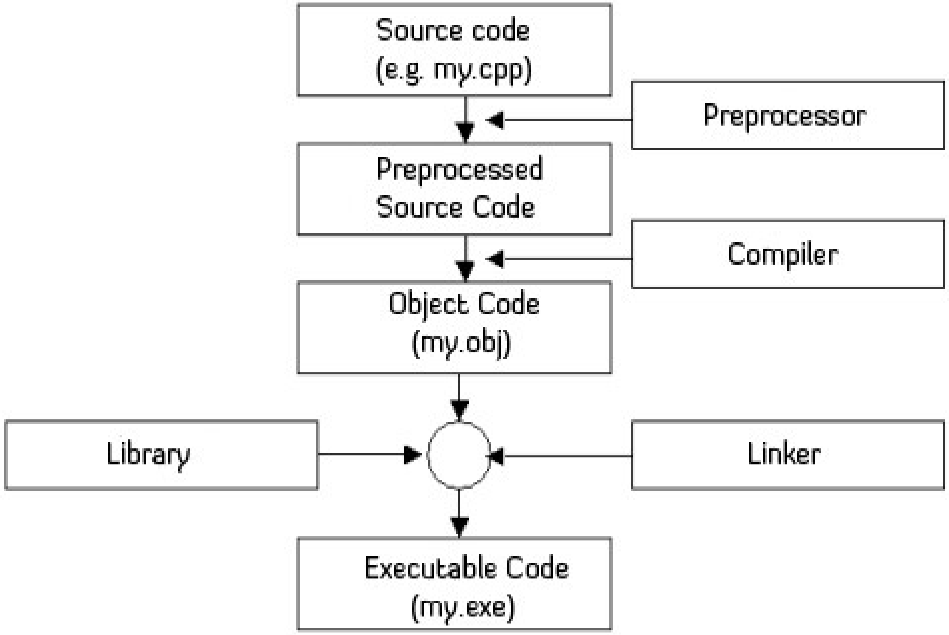
- Task: Use a text editor to create the C11 program.
- File Extension: Save the file with a .cpp extension (e.g., FirstCPPProgram.cpp).
- Source Program: The high-level language code you write.



```
#include<stdio.h>
#include<conio.h>

int main ()
{
printf("Hello World!");
getch();
return 0;
}

```

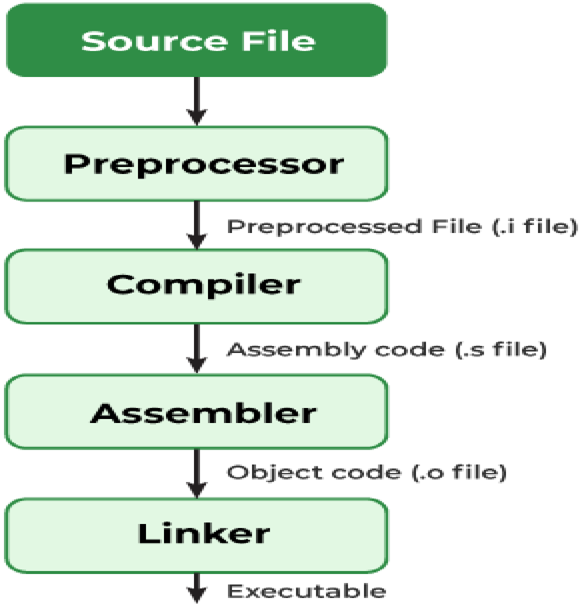
Compressor Directives

- Definition: Statements that begin with #, like #include <iostream>.
- Process: Handled by the preprocessor before actual compilation.



Compilation

- Task: The compiler checks the source code for syntax errors and translates it into machine language.
- Output: Object Program (machine language version of the source code).



Source File

Preprocessor

Preprocessed File (.i file)

Compiler

Assembly code (.s file)

Assembler

Object code (.o file)

Linker

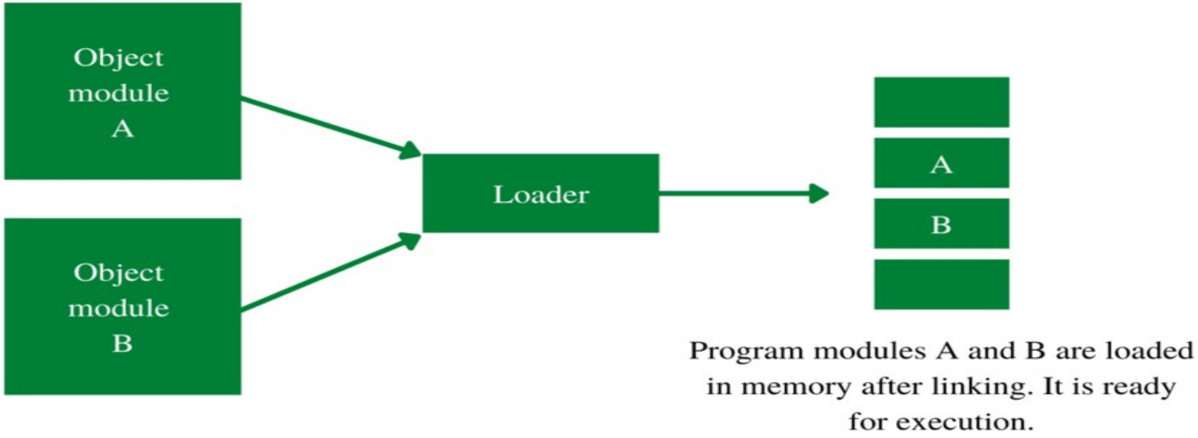
Executable

Linking

- Task: The linker combines the object program with prewritten code from libraries.
- Purpose: Creates an executable program by integrating code resources.
- Linker: A program that performs this task.

Loading

- Task: Load the executable program into main memory.
- Loader: A program responsible for this task.

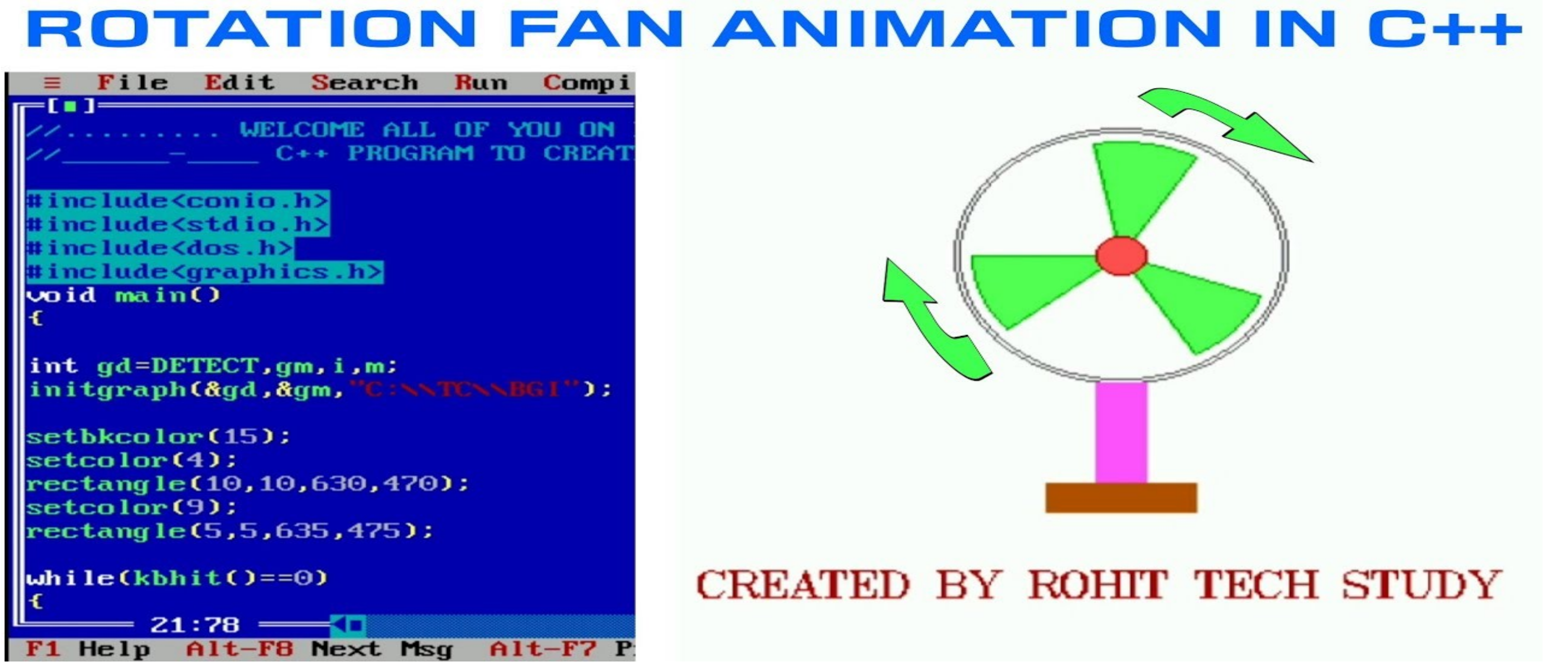


Program modules A and B are loaded in memory after linking. It is ready for execution.

General loading scheme

Execution

- Task: Run the program to perform its intended function.
- Outcome: The program displays the result or performs the action as coded.


ROTATION FAN ANIMATION IN C++
≡ File Edit Search Run Compi
[■]
21:78
F1 Help Alt-F8 Next Msg Alt-F7 P
CREATED BY ROHIT TECH STUDY

```
//.......... WELCOME ALL OF YOU ON
//________-_____ C++ PROGRAM TO CREAT

#include<conio.h>
#include<stdio.h>
#include<dos.h>
#include<graphics.h>
void main()
{

int gd=DETECT,gm,i,m;
initgraph(&gd,&gm,"C:\\TC\\BGI");

setbkcolor(15);
setcolor(4);
rectangle(10,10,630,470);
setcolor(9);
rectangle(5,5,635,475);

while(kbhit()==0)
{
```

Summary of Processing Steps

- Write Source Code
- Process Preprocessor Directives
- Compile to Object Program
- Link to Create Executable
- Load into Main Memory
- Execute Program

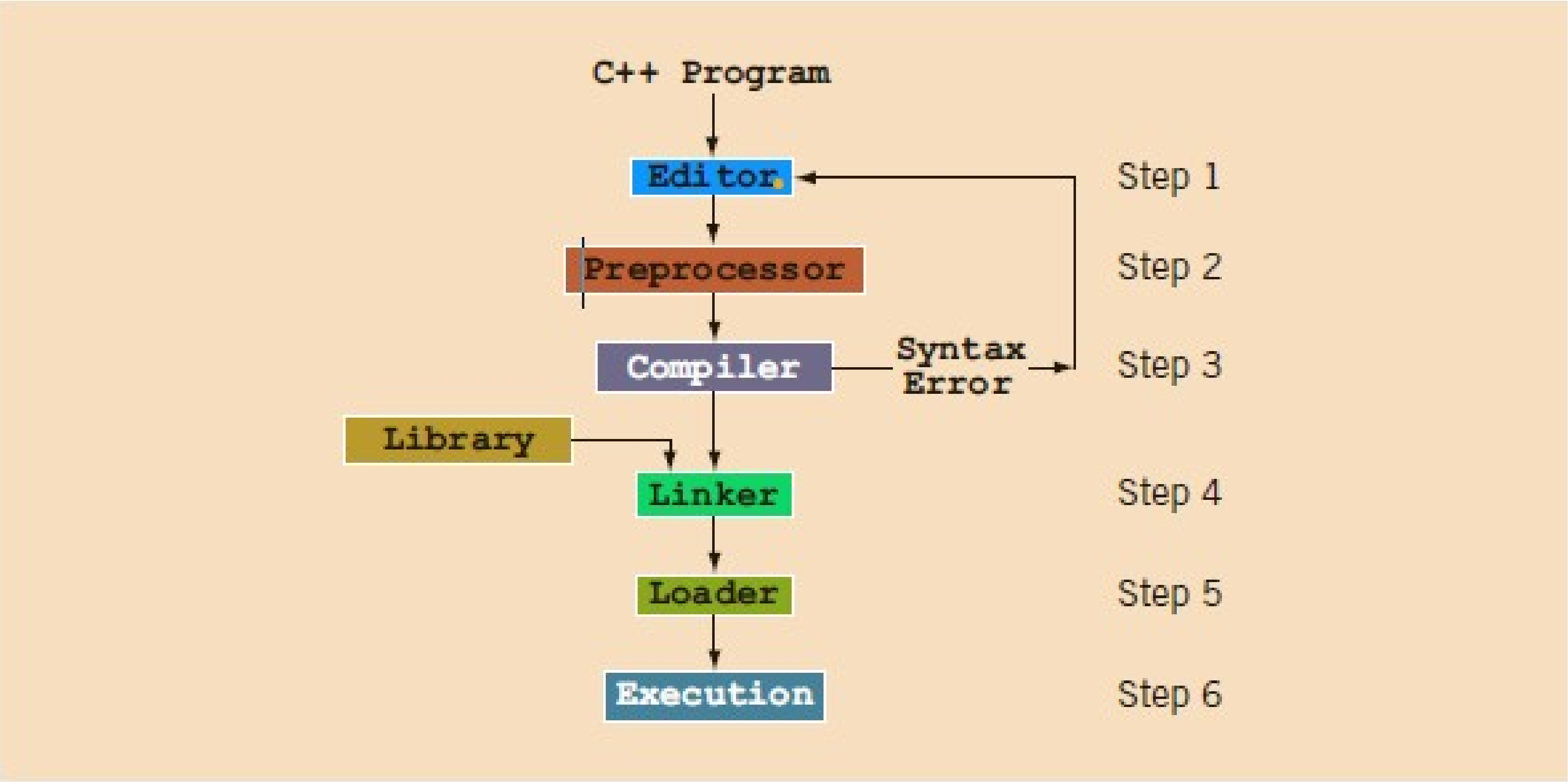


FIGURE 1-2 Processing a C++ program

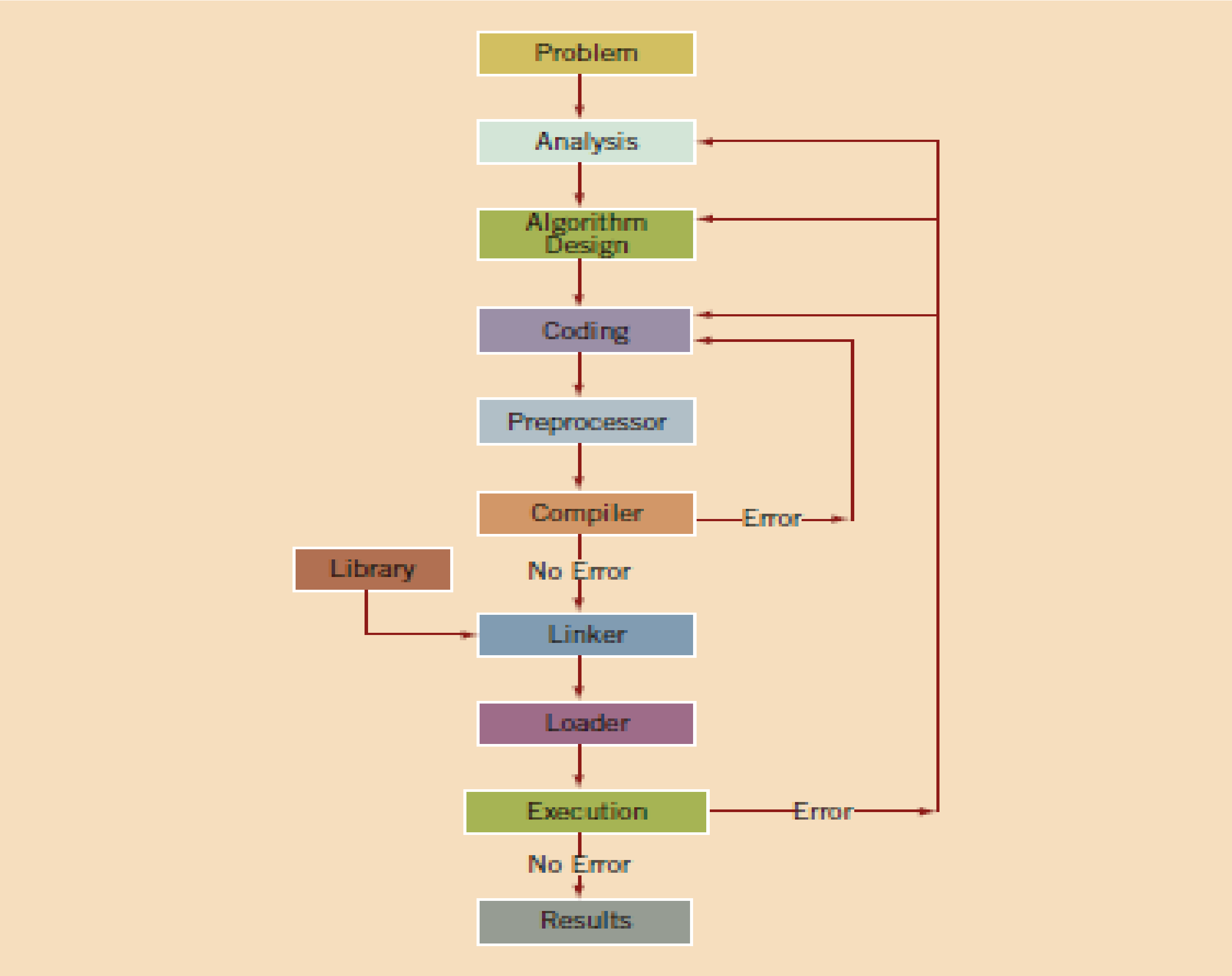


FIGURE 1-3 Problem analysis–coding–execution cycle

- Any question are appreciating