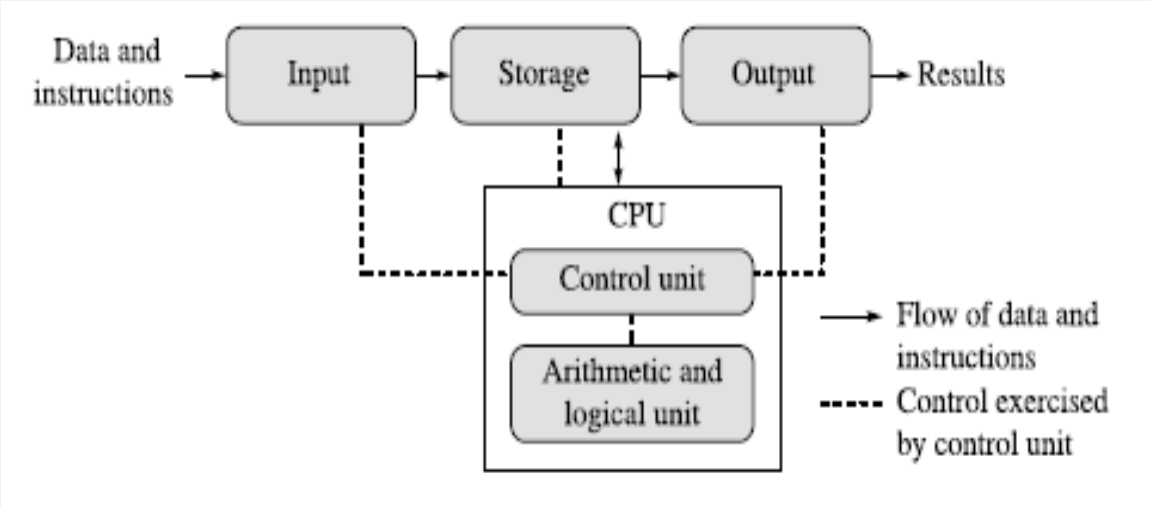
Introduction to Programming

What is a Computer?

- A computer, in simple terms, can be defined as an electronic device that is designed to accept data, perform the required mathematical and logical operations at high speed, and output the result. A computer accepts data, processes it, and produces information.

Basic Computer Organization

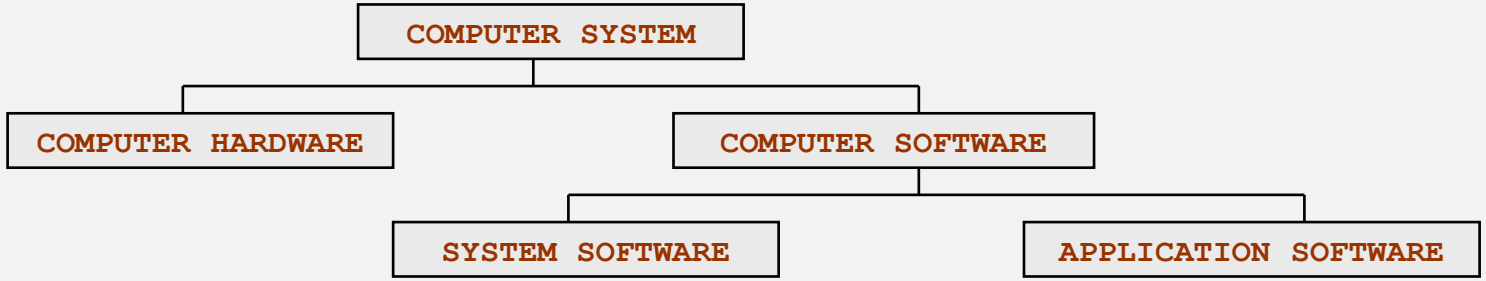
- A computer is an electronic device that basically performs five major operations:



Block diagram of a computer

Components of a Computer System

- When we talk about a computer, we actually mean two things. The first is the computer hardware which does all of the physical work computers are known for.
- The second part is computer software which tells the hardware what to do and how to do it.



Components of a Computer System

- Hardware of a system includes:
  - Memory
  - Disks
  - Processor
  - Peripheral Devices/Input and Output Devices

Computer Programming

- The computer hardware cannot think and make decisions on its own. So, it cannot be used to analyze a given set of data and find a solution on its own. The hardware needs a software (a set of programs) to instruct what has to be done. A program is a set of instructions that is arranged in a sequence to guide a computer to find a solution for the given problem. The process of writing a program is called *programming.*

Computer Programming

- Computer software is written by computer programmers using a programming language.
- The programmer writes a set of instructions (program) using a specific programming language. Such instructions are known as the s*ource code*.
- Another computer program called a *compiler* is then used on the source code, to transform the instructions into a language that the computer can understand. The result is an executable computer program, which is another name for software.
- Examples of computer software include:

➢ *Computer Games*

➢ *Driver Software*

➢ *Educational Software*

➢ *Media Players and Media Development Software*

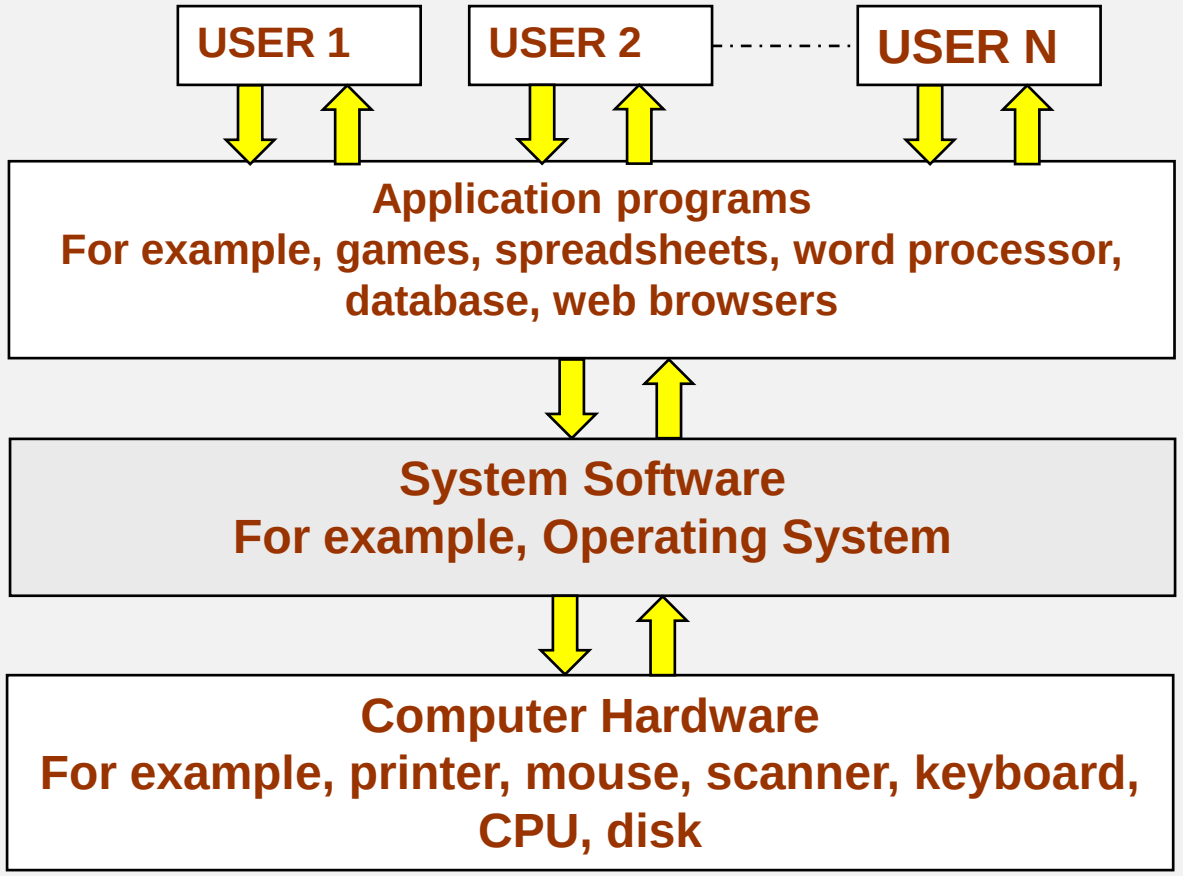
➢ *Productivity Software*

➢ *Operating Systems* software

Classification of Computer Software

- Computer software can be broadly classified into two groups: system software and application software.
- Application software is designed to solve a particular problem for users. It is generally what we think of when we say the word computer programs. Examples of application software include spreadsheets, database systems, desktop publishing systems, program development software, games, web browser, so on and so forth. Simply put, application software represents programs that allow users to do something besides simply run the hardware.
- On the contrary, system software provides a general programming environment in which programmers can create specific applications to suit their needs. This environment provides new functions that are not available at the hardware level and performs tasks related to executing the application program. System software represents programs that allow the hardware to run properly.

Classification of Computer Software



Computer Bios

- BIOS or the basic input/output system is a de facto standard defining a firmware interface.
- The BIOS is built into the computer and is the first code run by the computer when it is switched on. The key role of the BIOS is to load and start the operating system.
- When the computer starts, the first function that BIOS performs is to initialize and identify system devices such as the video display card, keyboard and mouse, hard disk, CD/DVD drive and other hardware. In other words, the code in the BIOS chip runs a series of tests called POST (Power On Self Test) to ensure that the system devices are working correctly.

Computer Bios

- BIOS performs the following functions:

➢Initializes the system hardware

➢The BIOS Menu Initializes system registers

➢Initializes power management system

➢Tests RAM

➢Test all the serial and parallel ports

➢Initializes floppy disk drive and hard disk controllers

➢Displays system summary information

Operating System

- The primary goal of an operating system is to make the computer system c*onvenient and efficient to use.* The operating system offers generic services to support user applications.
- From the users point of view the primary consideration is always the convenience. Users should find it easy to launch an application and work on it. For example, we use icon which gives us a clue about which application it is.
- An operating system ensures that the system resources (like CPU, memory, I/O devices, etc.) are utilized efficiently. For example, there may be many service requests on a web server and each user request need to be serviced. Similarly, there may be many programs residing in the main memory. Therefore, the system needs to determine which programs are active and which need to wait for some I/O operation. Since, the programs that need to wait can be suspended temporarily from engaging the processor. Hence, it is important for an operating system to have a control policy and algorithm to allocate the system resources.

Utility Software

- Utility software is used to analyze, configure, optimize and maintain the computer system. Utility programs may be requested by application programs during their execution. for multiple purposes. Some of them are listed below.

➢ *Disk defragmenters*	*Disk checkers*	*Disk cleaners*
➢ *Disk space analyzers*	*Disk partitions*	*Backup* utilities
➢ *Disk compression*	*File managers*	*System profilers*
➢ *Anti-virus* utilities	*Data compression* utilities	*Cryptographic* utilities
➢ *Launcher applications*	*Registry cleaners*	*Network utilities*
➢ *Command line interface (CLI)*	*Graphical user interface (GUI)*	

Compiler and Interpreter

- A **compiler** is a special type of program that transforms source code written in a programming language (the *source language*) into machine language comprising of just two digits- 1s and 0s (the *target language)*. The resultant code in 1s and 0s is known as the object *code*. The object code is the one which will be used to create an executable program.
- If the source code contains errors then the compiler will not be able to its intended task. Errors that limit the compiler in understanding a program are called *syntax errors*. Syntax errors are like spelling mistakes, typing mistakes, etc. Another type of error is logic error which occurs when the program does not function accurately. Logic errors are much harder to locate and correct.

Compiler and Interpreter

- **Interpreter:** Like the compiler, the interpreter also executes instructions written in a high-level language.
- While the compiler translates instructions written in high level programming language directly into the machine language; the interpreter on the other hand, translates the instructions into an intermediate form, which it then executes.
- Usually, a compiled program executes faster than an interpreted program. However, the big advantage of an interpreter is that it does not need to go through the compilation stage during which machine instructions are generated. This process can be time-consuming if the program is long. Moreover, the interpreter can immediately execute high-level programs.

Linker and Loader

- **Linker:** Also called *link editor* and *binder,* a linker is a program that combines object modules to form an executable program.
- Generally, in case of a large program, the programmers prefer to break a code into smaller modules as this simplifies the programming task. Eventually, when the source code of all the modules has been converted into object code, you need to put all the modules together. This is the job of the linker. Usually, the compiler automatically invokes the linker as the last step in compiling a program.
- **Loader:** A loader is a special type of program that copies programs from a storage device to main memory, where they can be executed. Most loaders are transparent to the users.

Application Software

- Application software is a type of computer software that employs the capabilities of a computer directly to perform a user-defined task. This is in contrast with system software which is involved in integrating a computer's capabilities, but typically does not directly apply them in the performance of tasks that benefit the user.
- To better understand application software consider an analogy where hardware would depict the relationship of an electric light bulb (an application) to an electric power generation plant (a system).
- Typical examples of software applications are word processors, spreadsheets, media players, education software, CAD, CAM, data communication software, statistical and operational research software, etc. Multiple applications bundled together as a package are sometimes referred to as an application suite.

Stored Program Concept

All digital computers are based on the principle of stored program concept, which was introduced by Sir John von Neumann in the late 1940s. The following are the key characteristic features of this concept:

- Before any data is processed, instructions are read into memory.
- Instructions are stored in the computer's memory for execution.
- Instructions are stored in binary form (using binary numbers—only 0s and 1s).
- Processing starts with the first instruction in the program, which is copied into a control unit circuit. The control unit executes the instructions.
- Instructions written by the users are performed sequentially until there is a break in the current flow.

Stored Program Concept

- Input/Output and processing operations are performed simultaneously. While data is being read/written, the central processing unit (CPU) executes another program in the memory that is ready for execution.

Note: A stored program architecture is a fundamental computer architecture wherein the computer executes the instructions that are stored in its memory.

Programming Languages

- A **programming language** is a language specifically designed to express computations that can be performed the computer. Programming languages are used to express algorithms or as a mode of human communication.
- While high-level programming languages are easy for the humans to read and understand, the computer actually understands the machine language that consists of numbers only.
- In between the machine languages and high-level languages, there is another type of language known as assembly language. Assembly languages are similar to machine languages, but they are much easier to program in because they allow a programmer to substitute names for numbers.
- However, irrespective of what language the programmer use, the program written using any programming languages has to be converted into machine language so that the computer can understand it. There are two ways to do this: compile the program or i*nterpret* the program.

Programming Languages

- The question of which language is best depends on the following factors:

➢ The type of computer on which the program has to be executed

➢ The type of program

➢ The expertise of the programmer

➢ For ex, FORTRAN is a good language for processing numerical data, but it does not lend itself very well to organizing large programs. Pascal can be used for writing well-structured and readable programs, but it is not as flexible as the C programming language. C++ goes one step ahead of C by incorporating powerful object-oriented features, but it is complex and difficult to learn.

First Generation: Machine Language

- Machine language is the lowest level of programming language. It is the only language that the computer understands. All the commands and data values are expressed using 1s and 0s.
- In the 1950s each computer had its own native language. Although there were similarities between each of the machine language but a computer could not understand programs written in another machine language.
- The main advantage of machine language is that the code can run very fast and efficiently, since it is directly executed by the CPU.

First Generation: Machine Language

- However, on the down side, the machine language is difficult to learn and is far more difficult to edit if errors occur. Moreover, if you want to add some instructions into memory at some location, then all the instructions after the insertion point would have to be moved down to make room in memory to accommodate the new instructions.
- Last but not the least, code written in machine language is not portable and to transfer code to a different computer it needs to be completely rewritten. Architectural considerations make portability a tough issue to resolve.

Second Generation: Assembly Language

- Assembly languages are symbolic programming languages that use mnemonics (symbols) to represent machine-language instructions. Since assembly language is close to the machine, it is also called low-level language.
- Basically, an assembly language statement consists of a label, an operation code, and one or more **operands**.
- Labels are used to identify and reference instructions in the program. The operation code (opcode) is a mnemonic that specifies the operation that has to be performed, such as *move, add, subtract,* or *compare*. The operand specifies the register or the location in main memory from where the data to be processed is located.
- Assembly language is machine dependent. This makes the code written in assembly language less portable as the code written to be executed on one machine will not run on machines from a different or sometimes even the same manufacturer.

Second Generation: Assembly Language

- No doubt, the code written in assembly language will be very efficient in terms of execution time and main memory usage as the language is also close to the computer.
- Programs written in assembly language need a translator often known as the assembler to convert them into machine language. This is because the computer will understand only the language of 1s and 0s. it will not understand mnemonics like ADD and SUB.
- The following instructions are a part of assembly language code to illustrate addition of two numbers
- MOV AX,4 Stores the value 4 in the AX register of CPU
  MOV BX,6 Stores the value 6 in the BX register of CPU
  ADD AX,BX Add the contents of AX and BX register. Store the result in AX register

Third Generation Programming Language

- The third generation was introduced to make the languages more programmer-friendly.
- 3GLs spurred the great increase in data processing that occurred in the 1960s and 1970s. in these languages, the program statements are not closely related to the internal characteristics of the computer and is therefore often referred to has high-level languages.
- 3GLs made programming easier, efficient and less prone to errors.
- Programs were written in an English-like manner, making them more convenient to use and giving the programmer more time to address a client's problems.

Third Generation Programming Language

- Most of the programmers preferred to use general purpose high level languages like BASIC (Beginners' All-purpose Symbolic Instruction Code), FORTRAN, PASCAL, COBOL, C++ or Java to write the code for their applications.
- Again, a translator is needed to translate the instructions written in high level language into computer-executable machine language. Such translators are commonly known as interpreters and compilers.
- 3GLs makes it easier to write and debug a program and gives the programmer more time to think about its overall logic. The programs written in such languages are portable between machines.

Fourth Generation: Very High Level Languages

- 4GLs is a little different from its prior generation because they are basically nonprocedural so the programmers define only what they want the computer to do, without supplying all the details of how it has to be done.

  Characteristics of such language include:

➢ the code comprising of instructions are written in English-like sentences;

➢ they are nonprocedural

➢ the code is easier to maintain

➢ 4GL code enhances the productivity of the programmers as they have to type fewer lines of code to get something done. It is said that a programmer become 10 times more productive when he writes the code using a 4GL than using a 3GL.

Fourth Generation: Very High Level Languages

- A typical example of a 4GL is the query language that allows a user to request information from a database with precisely worded English-like sentences.
- Let us take an example in which a report has to be generated that displays the total number of students enrolled in each class and in each semester. Using a 4GL, the request would look similar to this:

➢**TABLE FILE ENROLLMENT**

➢**SUM STUDENTS BY SEMESTER BY CLASS**

➢The only down side of a 4GL is that it does not make efficient use of machine's resources. However, the benefit of executing a program fast and easily far outweighs the extra costs of running it.

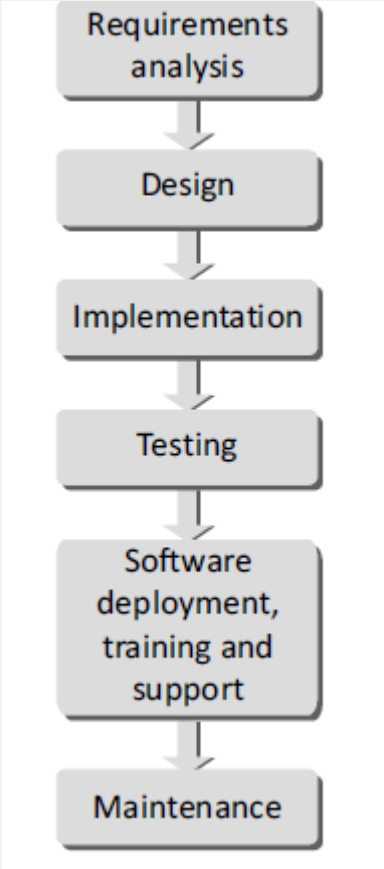
Fifth Generation Programming Language

- 5GLs are centered on solving problems using constraints given to the program, rather than using an algorithm written by a programmer.
- Most constraint-based and logic programming languages and some declarative languages form a part of the fifth-generation languages.
- 5GLs are widely used in artificial intelligence research.
- Typical examples of a 5GL include Prolog, OPS5, and Mercury.
- Another aspect of a 5GL is that it contains visual tools to help develop a program. A good example of a fifth generation language is Visual Basic.
- With 5GL, the programmer only needs to worry about what problems need to be solved and what conditions need to be met, without worrying about how to implement a routine or algorithm to solve them.

Design and Implementation of Efficient Programs

- The design and development of correct, efficient, and maintainable programs depends on the approach adopted by the programmer to perform various activities that need to be performed during the development process.
- The entire program or software (collection of programs) development process is divided into a number of phases where each phase performs a well-defined task. Moreover, the output of one phase provides the input for its subsequent phase.

Phases in Software Development Life Cycle



Phases in software development life cycle

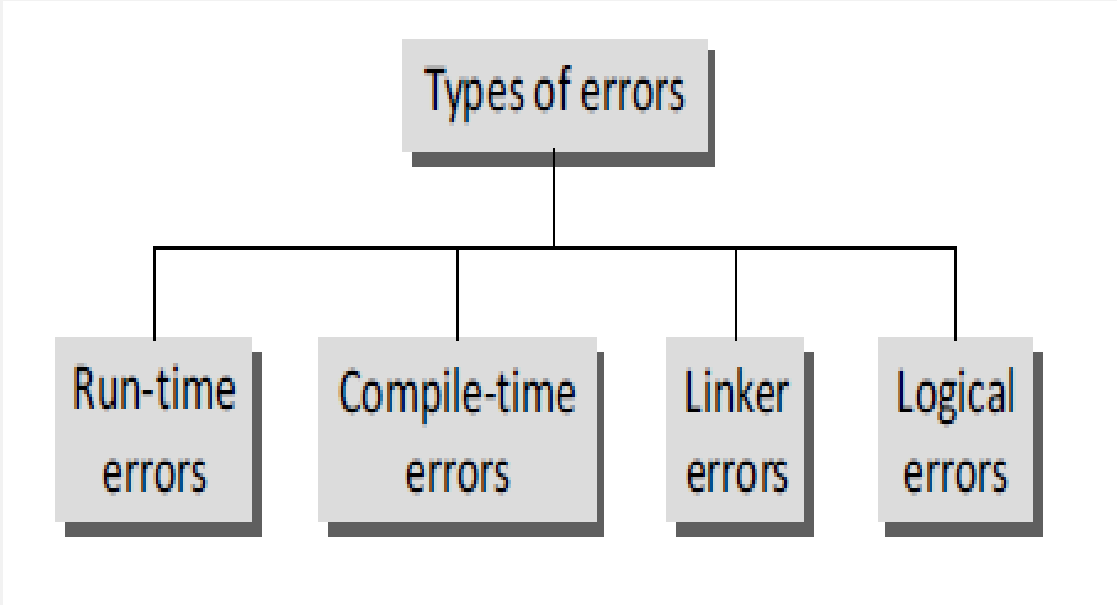
Program Design Tools: Algorithms, Flowcharts, Pseudocodes

- In general terms, an algorithm provides a blueprint to writing a program to solve a particular problem. It is considered to be an effective procedure for solving a problem in a finite number of steps. That is, a well-defined algorithm always provides an answer, and is guaranteed to terminate.
- A flowchart is a graphical or symbolic representation of a process. It is basically used to design and document virtually complex processes to help the viewers to visualize the logic of the process, so that they can gain a better understanding of the process and find flaws, bottlenecks, and other less obvious features within it. When designing a flowchart, each step in the process is depicted by a different symbol and is associated with a short description.

Program Design Tools: Algorithms, Flowcharts, Pseudocodes

- Pseudocode is a compact and informal high-level description of an algorithm that uses the structural conventions of a programming language.

Types of Errors



Testing and Debugging Approaches

- Unit Tests
- Integration Tests
- System Tests

END