INTRODUCTION

Outline:-

What is a computer.?

Computer hardware & software/program

Computer programming

Programming language

Generations of programming languages

Assembler & Compiler & Interpreter

Program execution & Linkers & Errors

1. What is a computer.?

A computer is a programmable machine that can execute a programmed list of instructions and respond to new instructions that it is given.

A computer is a programmable electronic device that accepts raw data as input and processes it with a set of instructions (a program) to produce the result as output.

A computer has two main parts: hardware and software. The hardware is responsible for executing tasks, and the software provides the hardware with directions for what tasks to perform and how to perform them.

2. Computer hardware & software/program

Computer hardware refers to the physical parts of a computer that enable it to function. These are parts of a computer which we can touch. Hardware includes Input devices(Keyboard, Mouse, Touchscreen) and Output devices (Monitor,Speaker, Printer), Storage devices (Hard Disk,SSD), Memory (RAM), Processor.

A computer program is a set of instructions that tells the computer what to do. Even a single line of code can be a program. It is the basic unit of a software.

A software is a set of programs or instructions that performs a function. It controls the working of a computer and manages all the components.

Software is mainly classified into two types. They are System software and Application software.

System software includes Operating systems (Windows, Android), Device drivers, Utility softwares.

Application software includes web browsers (Chrome), word processors (Ms office), Graphics softwares (Adobe).

Another class is firmware.

3. What is computer programming?

Computer Programming is the process of developing and implementing various sets of instructions to enable a computer to do a certain task.

Computer programming is the process of writing instructions, code or programs, that computers can use to perform tasks and solve problems.

Computer programmers are professionals who designs, write, test, and maintain code to create and update software and applications. They also document programs for future reference. They can work independently or as team.

Programming paradigms:-

Programming paradigms are fundamental styles or approaches to programming that define how problems are conceptualized and solutions are implemented.

i. Structured programming

ii. Object-oriented programming (OOP)

i. Structured programming

A programming paradigm aimed at improving the clarity and quality of software by using a top-down approach, breaking a program into smaller, manageable modules/functions/procedures.

Some examples of structured/procedural programming languages:- C, ALGOL, Pascal.

Use case:- Suitable for smaller programs or applications where complex relationships between data and functionality arent needed.

Characteristics:-

Top-Down approach: Programs are designed starting from a high-level overview, breaking down into smaller, more manageable components (functions or procedures).

Control structures: Utilizes three primary control structures;- Sequential, Selection(conditions), Iteration (loops).

Modularity: Encourages the use of functions or procedures to encapsulate logic and enhance readability.

ii. Object-oriented programming (OOP)

A paradigm based on the concept of "objects," which can contain data and methods. OOP focuses on organizing code around these objects rather than functions or logic.

Some examples of object-oriented programming languages:- Java,C++, Python.

Use case:- Ideal for large-scale applications, game development, and systems that require complex data interactions.

Characteristics:-

Objects: Central to OOP; objects are instances of classes that encapsulate data (attributes) and behaviors (methods).

Encapsulation: Data and methods are bundled together, promoting data hiding and reducing complexity.

Inheritance: New classes can inherit properties and methods from existing classes, facilitating code reuse.

Polymorphism: Objects of different classes can be treated as objects of a common superclass, allowing for flexible code.

4. What is a programming language?

Programming language is a set of rules that tells a computer what operations to perform.

Programming language is a set of rules for communicating an algorithm. It provides a linguistic framework for describing computations.

Programming language is a notational system for describing computation in a machine-readable and human-readable form.

Algorithm -a finite set of precise instructions for performing a computation or for solving a problem.

A computer language typically consists of two main parts:-

i. Syntax

This refers to the rules and structure of the language, dictating how code should be written. Syntax includes the arrangement of keywords, operators, and symbols, determining whether a piece of code is valid. For example, in Python, indentation is significant, while in languages like C or Java, curly braces are used to define blocks of code.

ii. Semantics

This refers to the meaning of the statements and expressions in the language. While syntax governs how code is structured, semantics deals with what that code actually does when executed. For instance, the syntax might allow for adding two numbers, but the semantics determines how that addition is performed and what the result is.

Together, syntax and semantics define how programmers communicate instructions to the computer and what those instructions accomplish.

5. Generations of programming languages

With regard to the level of abstraction, computer programming languages can be divided into:-

Low-level languages

High-level languages

A low-level language is a programming language that's closer to a computer's hardware and machine code than high-level languages.

It allows programmers to directly control a computer's resources, such as memory, registers, and CPU operations. This gives programmers the ability to create highly efficient and specialized programs.

They are hardware specific. They are fast and memory efficient.

Examples: Machine language and assembly language (x86, ARM, MIPS).

A high-level language is a programming language that provides a significant level of abstraction from the details of the computer's hardware. These languages are designed to be more user-friendly and easier to read, write, and maintain, making them more accessible for programmers.

High-level languages typically handle complex tasks, such as memory management and system calls, behind the scenes.

Characteristics:- Abstraction, Readability, Portability, Rich libraries. Examples: Java, C#, Python.

Generations:-

First Generation Languages

Second Generation Languages

Third Generation Languages

Fourth Generation Languages

Fifth Generation Languages

i. First Generation Languages (1GLs)

These are known as machine languages or machine code, are the lowest-level programming languages. They consist of binary code (0s and 1s) that a computer's central processing unit (CPU) can directly execute.

Each instruction had two parts: Operation code and Operand

Opcode is the first part of an instruction that tells the computer what function to perform and is also called Operation code. Opcodes are the numeric codes that hold the instructions given to the computer system.

Operand is another second part of instruction, which indicates the computer system where to find the data or instructions or where to store the data or instructions.

Characteristics:- Instructions written in binary format, Hardware specific, No abstraction.

ii. Second Generation Languages (2GLs)

Also known as assembly languages. Programs are made up of instructions written in mnemonics.

Mnemonics: Uses convenient alphabetic abbreviations to represent operation codes, and abstract symbols to represent operands.

Each instruction had two parts: Operation code, Operand

Because programs are not written in 1s and 0s, the computer must first translate the program before it can be executed. They use assemblers.

Characteristics:- Mnemonic instructions, Hardware specific/dependent, They need assemblers

Examples: x86, ARM, MIPS

Example of 2GL code:-

```
READ num1
READ num2
LOAD num1
ADD num2
STORE sum
PRINT sum
STOP
```

iii. Third Generation Languages (3GLs)

These are high-level programming languages that provide a greater level of abstraction from machine code and hardware details compared to first and second-generation languages.

They are designed to be more user-friendly and allow programmers to write code that is easier to read, maintain, and understand.

Instructions are written using English-like words and are called statements.

Examples: Java, Python, C++.

Characteristics:- High level abstraction, Readable syntax, Portability, Rich libraries, support structured & object oriented programming.

Example of 3GL code:-

```
Program sum2(input,output);
var
  num1,num2,sum : integer;

begin
  read(num1,num2);
  sum:=num1+num2;
  writeln(sum)
end.
```

iv. Fourth Generation Languages (4GLs)

These are high-level programming languages that are designed to be even more abstract and user-friendly than third-generation languages (3GLs).

They aim to reduce the amount of code needed to perform specific tasks, making programming more accessible and efficient, often focusing on specific problem domains such as database management, report generation, and user interface design.

Characteristics:- High-level of abstraction, Declarative in nature, Less code, Rapid application development, Focus on business application.

Examples: SQL (Structured Query Language, RPG (Report Program Generator).

v. Fifth Generation Languages (5GLs)

These are a category of programming languages that focus on solving problems using constraints and logic rather than explicit programming instructions.

They are often associated with artificial intelligence (AI) and natural language and aim to enable machines to understand and solve complex problems more autonomously.

Characteristics:- Artificial intelligence, High-level abstraction, Integration of knowledge representation.

Examples: Prolog, LISP, Mercury.

6. Assembler & Compiler & Interpreter

i. Assembler

This is a type of computer program that translates assembly language code into machine code (binary code) that a computer's CPU can execute.

Key functions:- Translation, Symbol management, Address resolution, Error checking.

Types: One-pass & Two-pass assembler.

Examples: NASM commonly used for x86 architecture, or MASM , used for programming on Windows platforms.

ii. Compiler

This is a program that translates source code written in a high-level programming language into intermediate code then machine code.

This process enables the code to be executed by a computer's CPU or run on a specific platform.

Key functions:- Lexical, Syntax and Semantic analysis, Error reporting, Optimization.

Types of compilers:-

Native compilers: Generate machine code for the specific architecture they are targeting.

Cross compilers: Generate code for a different architecture than the one on which the compiler is running.

Just-In-Time (JIT) compilers: Compile code at runtime, typically used in environments like Java and .NET to improve performance.

Examples of compilers:-

GCC (GNU Compiler Collection): Supports multiple programming languages, including C, C++, and Fortran.

Clang: A compiler for C, C++, and Objective-C, known for its modular architecture and fast compilation.

Javac: The Java compiler that converts Java source code into bytecode for the Java Virtual Machine (JVM).

iii. Interpreter

An interpreter is a type of program that executes code written in a programming language directly, translating it into machine code or an intermediate form line by line or statement by statement.

Unlike compilers, which translate the entire source code into machine code before execution, interpreters process the code on-the-fly.

Examples of interpreted languages include Python, Ruby, and JavaScript.

Characteristics:- Immediate execution, Portability, Dynamic typing.

Examples of interpreters:- CPython, Ruby MRI, PHP interpreter, Perl interpreter.

7. Program execution & Linkers & Errors

A breakdown of source code, object code, and executable files:-

a) Source code

This is the human-readable code written by programmers in a high-level programming language (like Python, Java, or C++).

It consists of instructions, functions, and logic that define what the program is supposed to do.

Example: A .py file for Python or a .java file for Java is considered source code.

b) Object code

This is the machine-readable code generated by a compiler after translating the source code.

Object code is typically in binary format and contains instructions that the computers processor can execute.

It often contains unresolved references (e.g., to external functions or variables) and is usually stored in an object file with extensions like .o (for Unix/Linux) or .obj (for Windows).

c) Executable file

This is a compiled version of the program that can be run directly by the operating system.

The linker processes the object code, resolving any references and combining multiple object files (including libraries) into a single file, typically with an extension like .exe (for Windows) or no extension (for Unix/Linux).

An executable file contains all the necessary code and data needed to run the program without further compilation or linking.

What is a linker?

A linker is a software tool or program that plays a crucial role in the compilation process of a program. It takes the object code generated by the compiler and combines it with other necessary libraries and modules to create an executable file.

Role: Combining object files, Address allocation, Handling libraries, Creating the executable file.

Types: Static linker & Dynamic linker

Why do i need a linker?

You need a linker because it takes care of resolving references between different parts of your program. When you write code, you often divide it into multiple source files or modules.

The linker ensures that all the necessary functions and variables from different modules are correctly connected, allowing your program to run smoothly.

If the linker encounters unresolved references during the linking process, it will produce an error and fail to create the executable file.

Program execution is the process of running a program on a computer, where the instructions written in the source code are carried out by the computer's processor. Here are the typical steps involved in program execution:-

Writing the source code

Compilation

Linking

Loading

Execution

Runtime operations

Termination

i. Writing the source code

The programmer writes the source code using a text editor or an Integrated Development Environment (IDE).

ii. Compilation

The source code is compiled by a compiler, which translates it into object code (machine code). This step checks for syntax errors and converts high-level instructions into a lower-level format.

iii. Linking

The object code is passed to a linker, which combines it with other object files and libraries to resolve references and produce an executable file. This step ensures that all function calls and variable references are correctly mapped.

iv. Loading

The operating system's loader takes the executable file and loads it into memory. This involves allocating space in RAM and preparing the necessary environment for the program to run.

v. Execution

The CPU begins executing the instructions in the loaded program. It fetches, decodes, and executes each instruction sequentially or according to control structures (like loops and conditionals).

vi. Runtime operations

During execution, the program may perform various operations, such as reading/writing to files, interacting with the user, or making system calls. It may also handle runtime errors, which can occur due to unexpected conditions (like file not found or division by zero).

vii. Termination

Once the program completes its task, it terminates. The operating system may clean up resources allocated during execution and return control to the user or another program.

These steps encompass the entire lifecycle of a program from code writing to execution and termination.

Program errors

A program error refers to a mistake or flaw in a software application that causes it to produce incorrect results, behave unexpectedly, or crash. Program errors can be classified into several types:-

i. Syntax errors

ii. Runtime errors

iii. Logical errors

i. Syntax errors

These are mistakes in the code structure that prevent the program from compiling or running.

Examples include typos, missing semicolons, or mismatched parentheses. Syntax errors are typically caught by the compiler or interpreter.

ii. Runtime errors

These errors occur while the program is executing, often leading to crashes or unexpected behavior.

Common examples include:-

Division by Zero: Attempting to divide a number by zero.

Null Pointer Exception: Trying to access a method or property of a null object.

Out of Bounds Errors: Accessing elements outside the range of an array or collection.

iii. Logical errors

These occur when the program runs without crashing but produces incorrect results due to faulty logic. Examples include:

Incorrect algorithms or formulas.

Misplaced conditional statements.

Errors in loops or iterations.