INTERNATIONAL MULTIDISCIPLINARY BOOK SERIES
https://ibseries.com/
IMBS, Vol. 01, No. 1, (2023)
Open Access. Online & Print
ISSN:
Foundations of Computer Science and Information Technology, Edited By: Dr. Sunita Chaudhary

CHAPTER 5:
SOFTWARE SYSTEMS, OPERATING SYSTEMS, AND PROGRAMMING LANGUAGES: CONCEPTS AND APPLICATIONS

Mohammad Sharif
Independent Researcher, India

Abstract
This chapter presents a comprehensive overview of computer software, operating systems, and programming language concepts, forming a critical foundation for understanding modern computing systems. It begins with the definition and classification of software into system software and application software, highlighting their roles in managing hardware and supporting user-oriented tasks. Various types of application software, such as word processing, database management, spreadsheet, presentation, and multimedia tools, are discussed with practical examples.
The chapter further explores the concept of open-source software, emphasizing its collaborative development model and contrast with proprietary software. A detailed explanation of operating systems is provided, including their functions, user interfaces, and examples such as MS-DOS, Windows, Linux, and UNIX. The evolution and features of these operating systems are also examined.
Additionally, the chapter introduces fundamental programming concepts, including algorithms, variables, data types, loops, functions, and object-oriented principles. It also explains the classification of programming languages into low-level and high-level languages, along with their advantages, limitations, and real-world applications. Furthermore, the chapter covers language translators such as assemblers, compilers, and interpreters, explaining their roles in converting source code into executable programs. Overall, this chapter provides a structured understanding of software systems and their significance in computer science.

Keywords: Software, System Software, Application Software, Open Source Software, Operating System, MS-DOS, Windows, Linux, Programming Concepts, Algorithms, Compiler, Interpreter, Assembler, Programming Languages, High-Level Languages, Low-Level Languages

Software:- Computer software is the set of programs that makes the hardware perform a set of tasks in particular order. Hardware and software are complimentary to each other. Both have to work together to produce meaningful results.

Types of software:- Computer software is classified into two broad categories; system software and application software.

1.System Software:- System software consists of a group of programs that control the operations of a computer equipment including functions like managing memory, managing peripherals, loading, storing, and is an interface

ISBN (978-81-685212-8-5)

between the application programs and the computer. MS DOS (Microsoft's Disk Operating System), UNIX are examples of system software.

2. Application software:- Software that can perform a specific task for the user, such as word processing, accounting, budgeting or payroll, fall under the category of application software. Word processors, spreadsheets, database management systems are all examples of general purpose application software.

Types of application software are:-

1. Word processing software: The main purpose of this software is to produce documents. MS-Word, Word Pad, Notepad and some other text editors are some of the examples of word processing software.

2.Database software: Database is a collection of related data. The purpose of this software is to organize and manage data. The advantage of this software is that you can change the way data is stored and displayed. MS access, dBase, FoxPro, Paradox, and Oracle are some of the examples of database software.

3. Spread sheet software: The spread sheet software is used to maintain budget, financial statements, grade sheets, and sales records. The purpose of this software is organizing numbers. It also allows the users to perform simple or complex calculations on the numbers entered in rows and columns. MS-Excel is one of the examples of spreadsheet software.

4.Presentation software: This software is used to display the information in the form of slide show. The three main functions of presentation software is editing that allows insertion and formatting of text, including graphics in the text and executing the slide shows. The best example

for this type of application software is Microsoft PowerPoint.

5.Multimedia software: Media players and real players are the examples of multimedia software. This software will allow the user to create audio and videos. The different forms of multimedia software are audio converters, players, burners, video encoders and decoders.

6. Open source software:- Open source refers to a program or software in which the source code (the form of the program when a programmer writes a program in a particular programming language) is available to the general public for use and/or modification from its original design free of charge. Open source code is typically created as a collaborative effort in which programmers improve upon the code and share the changes within the community. The rationale for this movement is that a larger group of programmers not concerned with proprietary ownership or financial gain will produce a more useful and bug-free product for everyone to use.

The basics behind the Open Source Initiative is that when programmers can read, redistribute and modify the source code for a piece of software, the software evolves. Open source sprouted in the technological community as a response to proprietary software owned by corporations. Proprietary software is privately owned and controlled. In the computer industry, proprietary is considered the opposite of open. A proprietary design or technique is one that is owned by a company. It also implies that the company has not divulged specifications that would allow other companies to duplicate the product

OERATING SYSTEM :- An operating system is a software component of a computer system that is responsible for the management of various activities of the computer and the sharing of computer resources. It hosts several applications that run on a computer and handles the operations of computer hardware. Users and application programs access the services offered by the operating systems, by means of system calls and application programming interfaces. Users interact with a computer operating system through Command Line Interfaces (CLIs) or Graphical User Interfaces known as GUIs. In short, an operating system enables user interaction with computer systems by acting as an interface between users or application programs and the computer hardware. Some of the common operating systems are LINUX, Windows, DOS etc.

1.Introduction to MS-DOS:- An operating system is a set of interrelated programs that manage and control computer processing. The Microsoft Disk Operating System, MS-DOS, is a traditional microcomputer operating system that consists of four major components.:-

- MS-DOS Kernel,
- User Interface (shell),
- MS-DOS BIOS,
- Operating-system loader

Short for Microsoft Disk Operating System, MS-DOS is a non-graphical command line operating system derived from 86-DOS that was created for IBM compatible computers. MS-DOS originally written by Tim Paterson and introduced by Microsoft in August 1981 and was last updated in 1994 when MS-DOS 6.22 was released. MS-DOS allows the user to navigate, open, and otherwise manipulate files on their computer from a command line instead of

a GUI like Windows.

Today, MS-DOS is no longer used; however, the command shell, more commonly known as the Windows command line is still used by many users. The picture to the right, is an example of what an MS-DOS window more appropriately referred to as the Windows command line looks like running under Microsoft Windows.

Most computer users are only familiar with how to navigate Microsoft Windows using the mouse. Unlike Windows, MS-DOS is a command-line and is navigated by using MS-DOS commands. For example, if you wanted to see all the files in a folder in Windows you would double-click the folder to open the folder in Windows Explorer. In MS-DOS, to view that same folder you would navigate to the folder using the cd command and then list the files in that folder using the dir command.

2.Introduction to Windows :-Microsoft Windows is a series of operating systems and environments developed and marketed by Microsoft Corporation. The first version of Windows was released in 1985 as a graphical user interface to MS-DOS, providing multiple document support, mouse support, drop down menus, and color video drivers. Later versions gradually replaced many of MS-DOS's built-in hardware functions with their own enhanced functions, until Windows fully assimilated MS-DOS and became a full-fledged operating system. Microsoft Windows is now often referred to as an integrated operating system due to the high level of integration between the core kernel functions and other Microsoft software such as Outlook, Windows Explorer and Internet Explorer. Despite it's poor security record caused by this integration, Microsoft Windows is today the most widely used OS on personal home computers, laptop computers, and small business machines.

As an integrated operating system, all Microsoft Windows versions come with preinstalled software that is ready to use upon installation. Basic text editors and calculators have been available since the first versions of Windows. Windows 98 added Media Player, Internet Explorer and Outlook Express. Windows Vista expands this with the Windows Mobility Center, Photo Gallery, DVD Maker, and the Linux-like Windows Sidebar. Vista is also the first version of Windows to have built-in security features. Although the second Windows XP service pack added a firewall and anti-virus monitoring service, Vista implements these features and more at the kernel level. Individual programs are 'sandboxed' and cannot access each other's memory. Critical drivers are executed in user mode, so crashes and malicious behavior cannot cause system-wide instability or security breaches. Even third-party anti-virus and anti-spyware software runs outside the kernel, further protecting it from bugs and backdoors in those programs.

3.Introduction to Linux:- Two of the most popular variations of UNIX come in the form of Linux. A big advantage of both Linux is that they are both open-source, that is, any user can contribute to the development of the OS. Versions of both operating systems are completely free.

Linux can easily take the role of a server or client machine. However, they are considered to be more difficult to master as both utilize the command line rather than a user friendly GUI. There are several different distributions of Linux, but for each the underlying operating system remains the same. Apple Macintosh machines offer high performance sound and graphics editing and are therefore extremely popular in the design industry. Apple have developed their own operating system, the newest version of which is the Mac OS X, which is based on UNIX. Mac OS X is a very user friendly operating system and is increasingly popular for home PCs.

4.Other Operating Systems:-

UNIX:-A big advantage of UNIX is that it can be run on nearly every computer hardware platform including Apple Macintosh machines. The UNIX operating system is one of the oldest and most powerful operating systems. It was developed by Bell Laboratories. There are many variants of UNIX available.

Novell NetWare:-Novell NetWare is an advanced network operating system. It has an advanced directory service structure similar to Microsoft's Active Directory. Fortunately both directory services are interoperable as both directories use the x500 directory service standard.

Computer programming concepts:- Following concepts are present in the majority of computer programming languages and/or are a fundamental part of the programming process.

- **Algorithm:-** A set of steps for carrying out a specific task. Algorithms are used extensively in computer programming to arrive at a solution for a problem. The process of creating an algorithm involves documenting all the necessary steps needed to arrive at the solution and how to perform each step. A real world example of an algorithm would be a recipe. The instructions of a typical recipe (add ingredients, mix, stir, etc.) are an algorithm.
- Source code:- The actual text used to write the instructions for a computer program. This text is then translated into something meaningful the computer can understand.

- Compiler :- A software tool that translates source code into data that the computer can understand. Specifically, a compiler is used to turn source code into object code. The object code is then passed through a program called a linker which turns it into an executable program.
- Data type :- The classification of pieces of information in a program. The amount of different data types varies between languages. Typically, there are data types for integers (whole numbers), floating-point numbers (numbers with a decimal part), and single characters. To distinguish between different data types, a computer uses special internal codes.
- Variable :- A container which represents a value in a program. Variables can store different types of data including numeric values, single characters, and text strings. The value of a variable can change all throughout a program.
- Constant:- The same thing as a variable with one major difference - the value of a constant does not change, while the value of a variable can change all throughout a program.
- Conditional :- A set of code that will execute only if a certain condition is true. Conditionals are used to test expressions and perform certain operations accordingly. For example, you could test a number input by the user and if it is too high print the message "The number entered is too high" and the program exits. Thanks to conditionals, a program can work differently every time it runs.
- Array :- A special type of variable used in many programming and web languages including PHP, JavaScript, and Java that contains a list of related values. For example, a colors array would contain a list of colors.
- Loop:- A segment of code that executes repeatedly based on a certain condition. Loops are used to perform tasks repeatedly a certain amount of times. For example, if you needed to print the numbers 1 to 10. You can use a loop for this task instead of manually printing all the numbers.
- Function :- A set of code used to carry out specific tasks. A function can take parameters which will affect its output as well as return values. Functions prevent unnecessary redundancy because you can use them as much as needed instead of retyping some code over and over. For example, if you need to multiply two numbers, instead of doing the calculation manually every time, you can supply the data to a function through some parameters which will do it for you.
- Class :- A template for a real world object to be used in a program. For example, a programmer can create a car class which represents a car. This class can contain the properties of a car (color, model, year, etc.) and functions that specify what the car does (drive, reverse, stop, etc.). Classes are used in object-oriented programming.

Two Basic Types of Computer Language:- Different kinds of languages have been developed to perform different types of work on the computer. Basically, languages can be divided into two categories according to how the computer understands them.

- Low-Level Languages: A language that corresponds directly to a specific machine
- High-Level Languages: Any language that is independent of the machine
- There are also other types of languages, which include
- System languages: These are designed for low-level tasks, like memory and process management
- Scripting languages: These tend to be high-level and very powerful
- Domain-specific languages: These are only used in very specific contexts
- Visual languages: Languages that are not text-based
- Esoteric languages: Languages that are jokes or are not intended for serious use

These languages are not mutually exclusive, and some languages can belong to multiple categories. The terms low-level and high-level are also open to interpretation, and some languages that were once considered high-level are now considered low-level as languages have continued to develop.

Computer language or programming language is a coded syntax used by computer programmers to communicate with a computer. Computer language establishes a flow of communication between software programs. The language enables a computer user to dictate what commands the computer must perform to process data. These languages can be classified into following categories.

- Machine language
- Assembly language

- High level language

Low-Level Languages:

Low-level computer languages are either machine codes or are very close them. A computer cannot understand instructions given to it in high-level languages or in English. It can only understand and execute instructions given in the form of machine language i.e. binary. There are two types of low-level languages:

Machine Language: a language that is directly interpreted into the hardware. Machine language or machine code is the native language directly understood by the computer's central processing unit or CPU. This type of computer language is not easy to understand, as it only uses a binary system, an element of notations containing only a series of numbers consisting of one and zero, to produce commands.

Machine language is the lowest and most elementary level of programming language and was the first type of programming language to be developed. Machine language is basically the only language that a computer can understand and it is usually written in hex.

In fact, a manufacturer designs a computer to obey just one language, its machine code, which is represented inside the computer by a string of binary digits (bits) 0 and 1. The symbol 0 stands for the absence of an electric pulse and the 1 stands for the presence of an electric pulse. Since a computer is capable of recognizing electric signals, it understands machine language.

Advantages	Disadvantages
Machine language makes fast and efficient use of the computer.	All operation codes have to be remembered
It requires no translator to translate the code. It is directly understood by the computer.	All memory addresses have to be remembered.
	It is hard to amend or find errors in a program written in the machine language.

Assembly Language: Its a slightly more user-friendly language that directly corresponds to machine language Assembly Level Language is a set of codes that can run directly on the computer's processor. This type of language is most appropriate in writing operating systems and maintaining desktop applications. With the assembly level language, it is easier for a programmer to define commands. It is easier to understand and use as compared to machine language.

Assembly language was developed to overcome some of the many inconveniences of machine language. This is another low-level but very important language in which operation codes and operands are given in the form of alphanumeric symbols instead of 0's and l's.

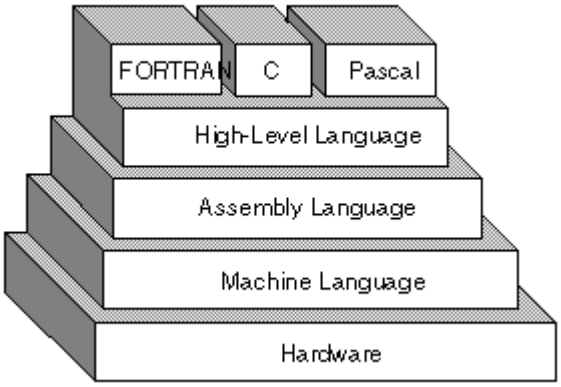
These alphanumeric symbols are known as mnemonic codes and can combine in a maximum of five-letter combinations e.g. ADD for addition, SUB for subtraction, START, LABEL etc. Because of this feature, assembly language is also known as 'Symbolic Programming Language.'

This language is also very difficult and needs a lot of practice to master it because there is only a little English support in this language. Mostly assembly language is used to help in compiler orientations. The instructions of the assembly language are converted to machine codes by a language translator and then they are executed by the computer.

Advantages	Disadvantages
Assembly language is easier to understand and use as compared to machine language.	Like machine language, it is also machine dependent/specific.
It is easy to locate and correct errors.	Since it is machine dependent, the programmer also needs to understand the hardware.
It is easily modified.	

High-Level Languages:

High Level Languages are user-friendly languages which are similar to English with vocabulary of words and symbols. These are easier to learn and require less time to write. They are problem oriented rather than 'machine' based. Program written in a high-level language can be translated into many machine languages and therefore can run on any computer for which there exists an appropriate translator.



High-level computer languages use formats that are similar to English. The purpose of developing high-level languages was to enable people to write programs easily, in their own native language environment (English).

High-level languages are basically symbolic languages that use English words and/or mathematical symbols rather than mnemonic codes. Each instruction in the high-level language is translated into many machine language instructions that the computer can understand.

Advantages	Disadvantages
High-level languages are user-friendly	A high-level language has to be translated into the machine language by a translator, which takes up time
They are similar to English and use English vocabulary and well-known symbols	The object code generated by a translator might be inefficient compared to an equivalent assembly language program
They are easier to learn	
They are easier to maintain	
They are problem-oriented rather than 'machine'-based	
A program written in a high-level language can be translated into many machine languages and can run on any computer for which there exists an appropriate translator	
The language is independent of the machine on which it is used i.e. programs developed in a high-level language can be run on any computer text	

Types of High-Level Languages

Many languages have been developed for achieving a variety of different tasks. Some are fairly specialized, and others are quite general.

These languages, categorized according to their use, are:

1) Algebraic Formula-Type Processing

These languages are oriented towards the computational procedures for solving mathematical and statistical problems.

Examples include:

- BASIC (Beginners All Purpose Symbolic Instruction Code)
- FORTRAN (Formula Translation)

- PL/I (Programming Language, Version 1)
- ALGOL (Algorithmic Language)
- APL (A Programming Language)

2. Business Data Processing

These languages are best able to maintain data processing procedures and problems involved in handling files. Some examples include:

- COBOL (Common Business Oriented Language)
- RPG (Report Program Generator)

3. String and List Processing

These are used for string manipulation, including search patterns and inserting and deleting characters. Examples are:

- LISP (List Processing)
- Prolog (Program in Logic)
- Object-Oriented Programming Language
- In OOP, the computer program is divided into objects. Examples are:
- C++
- Java

5. Visual Programming Language

These programming languages are designed for building Windows-based applications. Examples are:

- Visual Basic
- Visual Java
- Visual C

Some of the High level languages are as followes:-

FORTRAN:- The first important algorithmic language was FORTRAN (formula translation), designed in 1957 by an IBM team led by John Backus. It was intended for scientific computations with real numbers and collections of them organized as one- or multidimensional arrays. Its control structures included conditional IF statements, repetitive loops (so-called DO loops), and a GOTO statement that allowed non sequential execution of program code. FORTRAN made it convenient to have subprograms for common mathematical operations, and built libraries of them. FORTRAN was also designed to translate into efficient machine language. It was immediately successful and continues to evolve.

ALGOL:- ALGOL (algorithmic language) was designed by a committee of American and European computer scientists during 1958–60 for publishing algorithms, as well as for doing computations. ALGOL had recursive subprograms—procedures that could invoke themselves to solve a problem by reducing it to a smaller problem of the same kind. ALGOL introduced block structure, in which a program is composed of blocks that might contain both data and instructions and have the same structure as an entire program. Block structure became a powerful tool for building large programs out of small components.

ALGOL contributed a notation for describing the structure of a programming language, Backus–Naur Form, which in some variation became the standard tool for stating the syntax (grammar) of programming languages. ALGOL was widely used in Europe, and for many years it remained the language in which computer algorithms were published. Many important languages, such as Pascal and Ada, are its descendants.

LISP:- LISP (list processing) was developed about 1960 by John McCarthy at the Massachusetts Institute of Technology (MIT) and was founded on the mathematical theory of recursive functions (in which a function appears in its own definition). A LISP program is a function applied to data, rather than being a sequence of procedural steps as in FORTRAN and ALGOL. LISP uses a very simple notation in which operations and their operands are given in a parenthesized list. For example, (+ a (* b c)) stands for a + b*c. Although this appears awkward, the notation works well for computers. LISP also uses the list structure to represent data, and, because programs and data use the same structure, it is easy for a LISP program to operate on other programs as data. LISP became a common language for artificial intelligence (AI) programming, partly owing to the confluence of LISP and AI work at MIT and partly because AI programs capable of "learning" could be written in LISP as self-modifying programs. LISP has evolved through numerous dialects, such as Scheme and Common LISP.

C Language:- The C programming language was developed in 1972 by Dennis Ritchie and Brian Kernighan at the AT&T Corporation for programming computer operating systems. Its capacity to structure data and programs through the composition of smaller units is comparable to that of ALGOL. It uses a compact notation and provides the programmer with the ability to operate with the addresses of data as well as with their values. This ability is important in systems programming, and C shares with assembly language the power to exploit all the features of a computer's internal architecture. C, along with its descendant C++, remains one of the most common languages.

Business-oriented languages

COBOL:-COBOL (common business oriented language) has been heavily used by businesses since its inception in 1959. A committee of computer manufacturers and users and U.S. government organizations established CODASYL (Committee on Data Systems and Languages) to develop and oversee the language standard in order to ensure its portability across diverse systems. COBOL uses an English-like notation—novel when introduced. Business computations organize and manipulate large quantities of data, and COBOL introduced the record data structure for such tasks. A record clusters heterogeneous data such as a name, ID number, age, and address into a single unit. This contrasts with scientific languages, in which homogeneous arrays of numbers are common. Records are an important example of "chunking" data into a single object, and they appear in nearly all modern languages.

SQL:- SQL (structured query language) is a language for specifying the organization of databases (collections of records). Databases organized with SQL are called relational because SQL provides the ability to query a database for information that falls in a given relation. For example, a query might be "find all records with both last_name Smith and city New York." Commercial database programs commonly use a SQL-like language for their queries.

Education-oriented languages

BASIC:- BASIC (beginner's all-purpose symbolic instruction code) was designed at Dartmouth College in the mid-1960s by John Kemeny and Thomas Kurtz. It was intended to be easy to learn by novices, particularly non-computer science majors, and to run well on a time-sharing computer with many users. It had simple data structures and notation and it was interpreted: a BASIC program was translated line-by-line and executed as it was translated, which made it easy to locate programming errors. Its small size and simplicity also made BASIC a popular language for early personal computers. Its recent forms have adopted many of the data and control structures of other contemporary languages, which makes it more powerful but less convenient for beginners.

Pascal:- About 1970 Niklaus Wirth of Switzerland designed Pascal to teach structured programming, which emphasized the orderly use of conditional and loop control structures without GOTO statements. Although Pascal resembled ALGOL in notation, it provided the ability to define data types with which to organize complex information, a feature beyond the capabilities of ALGOL as well as FORTRAN and COBOL. User-defined data types allowed the programmer to introduce names for complex data, which the language translator could then check for correct usage before running a program. During the late 1970s and '80s, Pascal was one of the most widely used languages for programming instruction. It was available on nearly all computers, and, because of its familiarity, clarity, and security, it was used for production software as well as for education.

Logo:- Logo originated in the late 1960s as a simplified LISP dialect for education; Seymour Papert and others used it at MIT to teach mathematical thinking to schoolchildren. It had a more conventional syntax than LISP and featured "turtle graphics," a simple method for generating computer graphics. (The name came from an early project to program a turtle like robot.) Turtle graphics used body-centered instructions, in which an object was moved around a screen by commands, such as "left 90" and "forward," that specified actions relative to the current position and orientation of the object rather than in terms of a fixed framework. Together with recursive routines, this technique made it easy to program intricate and attractive patterns.

Hypertalk:- Hypertalk was designed as "programming for the rest of us" by Bill Atkinson for Apple's Macintosh. Using a simple English-like syntax, Hypertalk enabled anyone to combine text, graphics, and audio quickly into "linked stacks" that could be navigated by clicking with a mouse on standard buttons supplied by the program. Hypertalk was particularly popular among educators in the 1980s and early '90s for classroom multimedia presentations. Although Hypertalk had many features of object-oriented languages (described in the next section), Apple did not develop it for other computer platforms and let it languish; as Apple's market share declined in the 1990s, a new cross-platform way of displaying multimedia left Hypertalk all but obsolete (see the section World Wide Web display languages).

Object-oriented languages:-

Object-oriented languages help to manage complexity in large programs. Objects package data and the operations on them so that only the operations are publicly accessible and internal details of the data structures are hidden. This

information hiding made large-scale programming easier by allowing a programmer to think about each part of the program in isolation. In addition, objects may be derived from more general ones, "inheriting" their capabilities. Such an object hierarchy made it possible to define specialized objects without repeating all that is in the more general ones. Object-oriented programming began with the Simula language (1967), which added information hiding to ALGOL. Another influential object-oriented language was Smalltalk (1980), in which a program was a set of objects that interacted by sending messages to one another.

C++:- The C++ language, developed by Bjarne Stroustrup at AT&T in the mid-1980s, extended C by adding objects to it while preserving the efficiency of C programs. It has been one of the most important languages for both education and industrial programming. Large parts of many operating systems, such as the Microsoft Corporation's Windows 98, were written in C++.

Ada:- Ada was named for Augusta Ada King, countess of Lovelace, who was an assistant to the 19th-century English inventor Charles Babbage, and is sometimes called the first computer programmer. Ada, the language, was developed in the early 1980s for the U.S. Department of Defense for large-scale programming. It combined Pascal-like notation with the ability to package operations and data into independent modules. Its first form, Ada 83, was not fully object-oriented, but the subsequent Ada 95 provided objects and the ability to construct hierarchies of them. While no longer mandated for use in work for the Department of Defense, Ada remains an effective language for engineering large programs.

Java:- In the early 1990s, Java was designed by Sun Microsystems, Inc., as a programming language for the World Wide Web (WWW). Although it resembled C++ in appearance, it was fully object-oriented. In particular, Java dispensed with lower-level features, including the ability to manipulate data addresses, a capability that is neither desirable nor useful in programs for distributed systems. In order to be portable, Java programs are translated by a Java Virtual Machine specific to each computer platform, which then executes the Java program. In addition to adding interactive capabilities to the Internet through Web "applets," Java has been widely used for programming small and portable devices, such as mobile telephones.

Visual Basic:- Visual Basic was developed by Microsoft to extend the capabilities of BASIC by adding objects and "event-driven" programming: buttons, menus, and other elements of graphical user interfaces (GUIs). Visual Basic can also be used within other Microsoft software to program small routines.

Declarative languages:-

Declarative languages, also called nonprocedural or very high level, are programming languages in which (ideally) a program specifies what is to be done rather than how to do it. In such languages there is less difference between the specification of a program and its implementation than in the procedural languages described so far. The two common kinds of declarative languages are logic and functional languages.

Logic programming languages, of which PROLOG (programming in logic) is the best known, state a program as a set of logical relations (e.g., a grandparent is the parent of a parent of someone). Such languages are similar to the SQL database language. A program is executed by an "inference engine" that answers a query by searching these relations systematically to make inferences that will answer a query. PROLOG has been used extensively in natural language processing and other AI programs.

Functional languages have a mathematical style. A functional program is constructed by applying functions to arguments. Functional languages, such as LISP, ML, and Haskell, are used as research tools in language development, in automated mathematical theorem provers, and in some commercial projects.

Scripting languages:-

Scripting languages are sometimes called little languages. They are intended to solve relatively small programming problems that do not require the overhead of data declarations and other features needed to make large programs manageable. Scripting languages are used for writing operating system utilities, for special-purpose file-manipulation programs, and, because they are easy to learn, sometimes for considerably larger programs.

PERL (practical extraction and report language) was developed in the late 1980s, originally for use with the UNIX operating system. It was intended to have all the capabilities of earlier scripting languages. PERL provided many ways to state common operations and thereby allowed a programmer to adopt any convenient style. In the 1990s it became popular as a system-programming tool, both for small utility programs and for prototypes of larger ones. Together with other languages discussed below, it also became popular for programming computer Web "servers."

Document formatting languages:-

Document formatting languages specify the organization of printed text and graphics. They fall into several classes:

text formatting notation that can serve the same functions as a word processing program, page description languages that are interpreted by a printing device, and, most generally, markup languages that describe the intended function of portions of a document.

TeX:- TeX was developed during 1977–86 as a text formatting language by Donald Knuth, a Stanford University professor, to improve the quality of mathematical notation in his books. Text formatting systems, unlike WYSIWYG ("What You See Is What You Get") word processors, embed plain text formatting commands in a document, which are then interpreted by the language processor to produce a formatted document for display or printing. TeX marks italic text, for example, as {\it this is italicized}, which is then displayed as this is italicized.

TeX largely replaced earlier text formatting languages. Its powerful and flexible abilities gave an expert precise control over such things as the choice of fonts, layout of tables, mathematical notation, and the inclusion of graphics within a document. It is generally used with the aid of "macro" packages that define simple commands for common operations, such as starting a new paragraph; LaTeX is a widely used package. TeX contains numerous standards "style sheets" for different types of documents, and these may be further adapted by each user. There are also related programs such as BibTeX, which manages bibliographies and has style sheets for all of the common bibliography styles, and versions of TeX for languages with various alphabets.

PostScript:- PostScript is a page-description language developed in the early 1980s by Adobe Systems Incorporated on the basis of work at Xerox PARC (Palo Alto Research Center). Such languages describe documents in terms that can be interpreted by a personal computer to display the document on its screen or by a microprocessor in a printer or a typesetting device. PostScript commands can, for example, precisely position text, in various fonts and sizes, draw images that are mathematically described, and specify colour or shading. PostScript uses postfix, also called reverse Polish notation, in which an operation name follows its arguments. Thus, "300 600 20 270 arc stroke" means: draw ("stroke") a 270-degree arc with radius 20 at location (300, 600). Although PostScript can be read and written by a programmer, it is normally produced by text formatting programs, word processors, or graphic display tools. The success of PostScript is due to its specification's being in the public domain and to its being a good match for high-resolution laser printers. It has influenced the development of printing fonts, and manufacturers produce a large variety of PostScript fonts.

SGML:- SGML (standard generalized markup language) is an international standard for the definition of markup languages; that is, it is a meta language. Markup consists of notations called tags that specify the function of a piece of text or how it is to be displayed. SGML emphasizes descriptive markup, in which a tag might be "<emphasis>." Such a markup denotes the document function, and it could be interpreted as reverse video on a computer screen, underlining by a typewriter, or italics in typeset text. SGML is used to specify DTDs (document type definitions). A DTD defines a kind of document, such as a report, by specifying what elements must appear in the document—e.g., <Title>—and giving rules for the use of document elements, such as that a paragraph may appear within a table entry but a table may not appear within a paragraph. A marked-up text may be analyzed by a parsing program to determine if it conforms to a DTD. Another program may read the markups to prepare an index or to translate the document into PostScript for printing. Yet another might generate large type or audio for readers with visual or hearing disabilities.

World Wide Web display languages:

HTML:- The World Wide Web is a system for displaying text, graphics, and audio retrieved over the Internet on a computer monitor. Each retrieval unit is known as a Web page, and such pages frequently contain "links" that allow related pages to be retrieved. HTML (hypertext markup language) is the markup language for encoding Web pages. It was designed by Tim Berners-Lee at the CERN nuclear physics laboratory in Switzerland during the 1980s and is defined by an SGML DTD. HTML markup tags specify document elements such as headings, paragraphs, and tables. They mark up a document for display by a computer program known as a Web browser. The browser interprets the tags, displaying the headings, paragraphs, and tables in a layout that is adapted to the screen size and fonts available to it.

HTML documents also contain anchors, which are tags that specify links to other Web pages. An anchor has the form <A HREF= "http://www.britannica.com"> Encyclopedia Britannica</A>, where the quoted string is the URL (universal resource locator) to which the link points (the Web "address") and the text following it is what appears in a Web browser, underlined to show that it is a link to another page. What is displayed as a single page may also be formed from multiple URLs, some containing text and others graphics.

XML:- HTML does not allow one to define new text elements; that is, it is not extensible. XML (extensible markup language) is a simplified form of SGML intended for documents that are published on the Web. Like SGML, XML uses DTDs to define document types and the meanings of tags used in them. XML adopts conventions that make it easy to parse, such as that document entities are marked by both a beginning and an ending

tag, such as <BEGIN>…</BEGIN>. XML provides more kinds of hypertext links than HTML, such as bidirectional links and links relative to a document subsection. Because an author may define new tags, an XML DTD must also contain rules that instruct a Web browser how to interpret them—how an entity is to be displayed or how it is to generate an action such as preparing an e-mail message.

Web scripting:- Web pages marked up with HTML or XML are largely static documents. Web scripting can add information to a page as a reader uses it or let the reader enter information that may, for example, be passed on to the order department of an online business. CGI (common gateway interface) provides one mechanism; it transmits requests and responses between the reader's Web browser and the Web server that provides the page. The CGI component on the server contains small programs called scripts that take information from the browser system or provide it for display. A simple script might ask the reader's name; determine the Internet address of the system that the reader uses, and print a greeting. Scripts may be written in any programming language, but, because they are generally simple text-processing routines, scripting languages like PERL are particularly appropriate.

Another approach is to use a language designed for Web scripts to be executed by the browser. JavaScript is one such language, designed by the Netscape Communications Corp., which may be used with both Netscape's and Microsoft's browsers. JavaScript is a simple language, quite different from Java. A JavaScript program may be embedded in a Web page with the HTML tag <script language="JavaScript">. JavaScript instructions following that tag will be executed by the browser when the page is selected. In order to speed up display of dynamic (interactive) pages, JavaScript is often combined with XML or some other language for exchanging information between the server and the client's browser. In particular, the XMLHttpRequest command enables asynchronous data requests from the server without requiring the server to resend the entire Web page. This approach, or "philosophy," of programming is called Ajax (asynchronous JavaScript and XML).

VB Script is a subset of Visual Basic. Originally developed for Microsoft's Office suite of programs, it was later used for Web scripting as well. Its capabilities are similar to those of JavaScript, and it may be embedded in HTML in the same fashion.

Behind the use of such scripting languages for Web programming lies the idea of component programming, in which programs are constructed by combining independent previously written components without any further language processing. JavaScript and VB Script programs were designed as components that may be attached to Web browsers to control how they display information.

Language translator:- A computer language translator is a program that translates a set of code written in one programming language into a functional equivalent of the code in another programming language. The different types of computer translators are interpreters, source-to-source compilers, standard compilers, de compilers, assemblers and dis-assemblers.

Computer language translators are the programs that execute instructions written in a high-level language. There are two ways to run programs written in a high-level language. The most common is to compile the program; the other method is to pass the program through an interpreter.

1. Assembler:- An assembler translates assembly language into machine code. Assembly language consists of mnemonics for machine opcodes so assemblers perform a 1:1 translation from mnemonics to a direct instruction. For example:

LDA #4 converts to 0001001000100100

Conversely, one instruction in a high level language will translate to one or more instructions at machine level.

Advantages of using an Assembler:

1. Very fast in translating assembly language to machine code as 1 to 1 relationship.
2. Assembly code is often very efficient (and therefore fast) because it is a low level language.
3. Assembly code is fairly easy to understand due to the use of English-like mnemonics.

Disadvantages of using Assembler:

- Assembly language is written for a certain instruction set and/or processor.
- Assembly tends to be optimized for the hardware it's designed for, meaning it is often incompatible with different hardware.
- Lots of assembly code is needed to do relatively simple tasks, and complex programs require lots of programming time.

2.Compiler:- A Compiler is a computer program that translates code written in a high level language to a lower

level language, object/machine code. The most common reason for translating source code is to create an executable program (converting from a high level language into machine language).

In other words we can say, A compiler is a special program that processes statements written in a particular programming language called as source code and converts them into machine language or "machine code" that a computer's processor uses. Compiler translates high level language programs directly into machine language program. This process is called compilation.

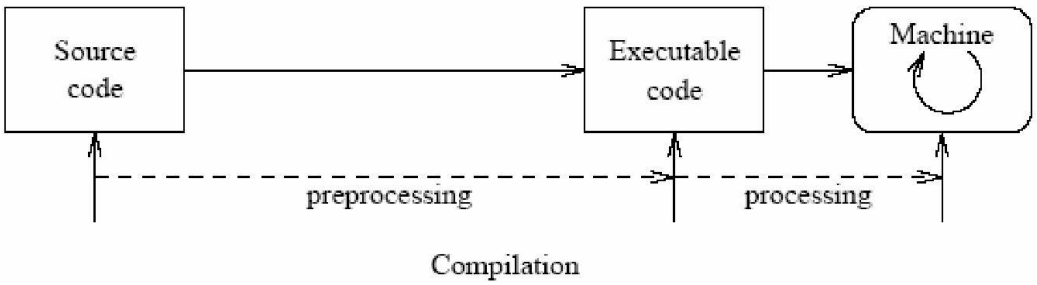


Fig.:- Compilation

Advantages of using a compiler

1. Source code is not included; therefore compiled code is more secure than interpreted code
2. Tends to produce faster code than interpreting source code
3. Produces an executable file, and therefore the program can be run without need of the source code

Disadvantages of using a compiler

1. Object code needs to be produced before a final executable file, this can be a slow process
2. The source code must be 100% correct for the executable file to be produced

3.Interpreter:- An interpreter program executes other programs directly, running through program code and executing it line-by-line. As it analyses every line, an interpreter is slower than running compiled code but it can take less time to interpret program code than to compile and then run it — this is very useful when prototyping and testing code. Interpreters are written for multiple platforms, this means code written once can be run immediately on different systems without having to recompile for each. Examples of this include flash based web programs that will run on your PC, MAC, games console and Mobile phone.

In other words we can say, An interpreter translates high-level instructions into an intermediate form, which it then executes. Compiled programs generally run faster than interpreted programs. The advantage of an interpreter, however, 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.

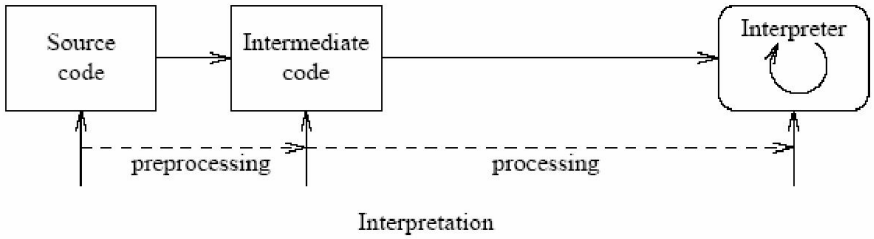


Fig:-Interpretation

Advantages of using an Interpreter

1.Easier to debug(check errors) than a compiler.

2.Easier to create multi-platform code, as each different platform would have an interpreter to run the same code.

3.Useful for prototyping software and testing basic program logic.

Disadvantages of using an Interpreter

1.Source code is required for the program to be executed, and this source code can be read making it insecure.

2.Interpreters are generally slower than compiled programs due to the per-line translation method.

References

1. Silberschatz, A., Galvin, P. B., & Gagne, G. (2018). Operating system concepts (10th ed.). Wiley.
2. Stallings, W. (2018). Operating systems: Internals and design principles (9th ed.). Pearson.
3. Tanenbaum, A. S., & Bos, H. (2015). Modern operating systems (4th ed.). Pearson.
4. Sebesta, R. W. (2016). Concepts of programming languages (11th ed.). Pearson.
5. Backus, J. (1978). Can programming be liberated from the von Neumann style? Communications of the ACM, 21(8), 613–641.
6. Dijkstra, E. W. (1968). Go to statement considered harmful. Communications of the ACM, 11(3), 147–148.
7. McIlroy, M. D. (1968). Mass produced software components. NATO Software Engineering Conference.
8. Open Source Initiative. (n.d.). The open source definition. Retrieved from https://opensource.org
9. IEEE Computer Society. (n.d.). Software engineering and operating systems. Retrieved from https://www.computer.org
10. ACM Digital Library. (n.d.). Programming languages and software systems research. Retrieved from https://dl.acm.org