



Vishwendu Vidya Prasarak Mandal's

(Regd. No: MAH 1906 / F-1614; Dt.5/3/1987)

Abhinav Vidyalay & Junior College

(Govt. Regd. No. Prim.Edu. 6-970,90-91 Dtd:16-8-90; Index No: Sec:16.17.019 / 020; H.Sec: J16.17.24)

Contact: +91(0)251 2472232 admin@abhinav.ac.in; www.abhinav.ac.in

XI IT Programming-Languages

Need for Programming Languages

Machine computations are low level , full of details about the inner workings of the machine . machine language is a collection of very detailed instructions that control the computer's internal circuitry.

Machine language is the native language of the computer ; it is the notation to which the computer responds directly. For example a machine language code might look like this

```
000000101111001010
```

```
00000010111110010
```

```
000000110011101010
```

The above code adds the numbers in locations 10 and 11 and stores the result in location 12. this is exactly what the machine understands. Each different type of CPU has its own unique machine language.

Programs in machine language are usually unintelligible at the lowest level , since they consist only of 0s and 1s.

Programming languages were invented to make machines easier to use. They are tools for instructing machines. Programming languages are used for specifying , organizing , and reasoning about computations.

What is a Programming Language

A programming language is a notation for specifying a sequence of operations to be carried out by a computer.

A programming language has a vocabulary and set of grammatical rules for instructing a computer to perform specific tasks.

The term programming language usually refers to high-level languages, such as BASIC, C, C++, COBOL, FORTRAN, Ada, and Pascal.

Types of Programming Languages: There are many different languages that can be used to program a computer. Computer languages can be classified as first , second and third generation languages.

Usually, a program will be written in some language whose instruction set is more compatible with human languages and human thought processes. Such a language is called a high-level language for e.g. Pascal, Visual Basic, C++, and Java etc.

As a rule, a single instruction in a high-level language will be equivalent to several instructions in machine language. This greatly simplifies the task of writing complete, correct programs.

Lying between machine languages and high-level languages are languages called assembly languages.

Assembly language is a variant of machine language in which names and symbols take the place of the actual codes for machine operations , values and storage locations , making individual instructions more readable

```
mov ax, WORD PTR Long1[0] ; AX = low word, long1
mov dx, WORD PTR Long1[2] ; DX = high word, long1
add ax, WORD PTR Long2[0] ; Add low word, long2
adc dx, WORD PTR Long2[2] ; Add high word, long2
ret ; Result returned as DX:AX
```

Higher level languages have replaced machine and assembly language in all areas of programming. Programming languages were designed to be high level .

A language is high level if it is independent of the underlying machine. Portability is another term for machine independence; a language is portable if programs in the language can be run on different machines with little or no change. Furthermore, the rules for programming in a particular high-level language are much the same for all computers, so that a program written for one computer can generally be run on many different computers with little or no alteration. Thus, we see that a high-level language offers three significant advantages over machine language: simplicity, uniformity and portability.

Lying above high-level languages are languages called fourth-generation languages (usually abbreviated 4GL).

4GLs are far removed from machine languages and represent the class of computer languages closest to human languages.

- The question of which language is best is one that consumes a lot of time and energy among computer professionals. Every language has its strengths and weaknesses. For example, FORTRAN is a particularly good language for processing numerical data, but it does not lend itself very well to organizing large programs. Pascal is very good for writing well-structured and readable programs, but it is not as flexible as the C programming language. C++ embodies powerful object-oriented features, but it is complex and difficult to learn.
- The choice of which language to use depends on the type of computer the program is to run on, what sort of program it is, and the expertise of the programmer.

Regardless of what language you use, you eventually need to convert your program into machine language so that the computer can understand it.

Need for Translator : A program that is written in a high level language must, however, be translated into machine language before it can be executed. So a small program (which comes with every programming Language) comes into picture which is called as translator which converts High level language into low level language and vice – a - versa.

Types Of Translator: There are two types of translator: Interpreter and Compiler.

A Translator is itself a computer program. It accepts a program written in a high-level language as input, and generates a corresponding machine-language program as output. The original high-level program is called the source program.

Interpreters, proceed through a program by translating and then executing single instructions or small group of instructions.

An Interpreter takes a program and its input at the same time as shown in the fig below .it translates the program , implementing operations as it encounters them , and doing input / output as necessary.

One main advantage of an interpreter is that execution as well as syntax errors are detected as each statement is encountered .thus debugging is easier in interpreted languages.

Examples of Interpreted Languages are QBasic, Fortran

Compilers translate the entire program into machine language before executing any of the instructions. Compilers translate source code into machine oriented target code called **object code**

After source code is compiled into object code , no futher references is made to the source language.

Libraries

If you had to create everything from scratch every time you wrote a program, it would be tedious indeed. The same kind of functionality is often required in many programs—reading data from the keyboard, for example, or displaying information on the screen. To address this, programming languages tend to come supplied with considerable quantities of pre-written code that provides standard facilities such as these, so you don't have to write the code for them yourself.

Standard code intended for use in any program is kept in a library. The library that comes with a particular programming language is as important as the language itself, as the quality and scope of the library can have a significant effect on how long it will take you to complete a given programming task.

The object file produced by the compiler cannot be executed .If the source contains library functions the code for these functions are not included in the object file. It is the job of the Linker to integrate the code of the library functions with the object code into a single executable file.

Executable File: It is a file in a format that the computer can directly execute. Unlike source file, executable files cannot be read by humans. To transform a source file into an executable file you need to pass it through a compiler or assembler

Example of Compiled Languages are C, C++ .

The translation of a source program into object code is said to occur at translation time. Once translation is complete the object code is run at a later time called **run time**

Program Listing of a small program in C to explain compilation

The object code created by the compiler takes longer to run and occupies more space than machine code .Thus as you have seen the size of the object file is quite larger than the original source file.

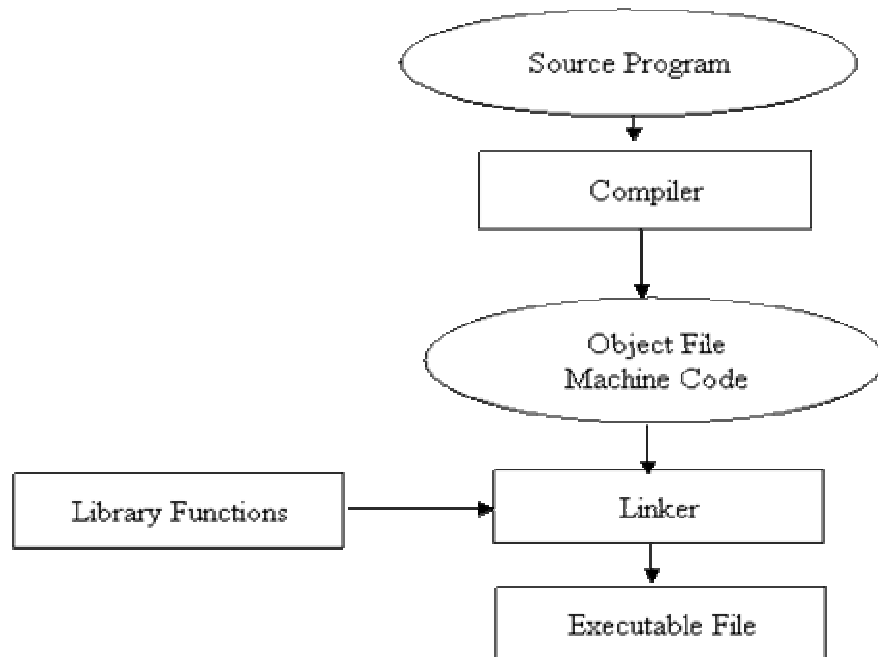
Interpretation can be more flexible than compilation .

The repeated examination of the source program by an interpreter allows interpretation to be more flexible than compilation.

An interpreter directly runs the source program, so it can allow program to be changed whenever required, to add features or correct errors.

Futhermore an interpreter works with the source text , so it can pinpoint an error in the source text and report it accurately.

With a compiler all tranalation is completed before the object code is run, which prevents the object file from being readily adapted as it runs.



Each programming language has a unique set of keywords (words that it understands) and a special syntax for organizing program instructions.

Language Elements

There are a number of basic programming elements found in most all programming languages.

Comments	// /* .. */
Data Types	int long string
Custom Data Types	Type TCustom ... End Type
Data Conversion	Convert a variable to a different type
Type Casting	Force the program to treat a variable as a different type
Pointers	Getting the address of a variable
Arrays	a(i), a[i]
Flow Control	If .. Then .. Else - Case - While
Catching Errors	How to trap errors so that your program won't crash
String Functions	compare copy substring
CR/LF	How to add CR/LF to a string
Functions and Subroutines	
Objects	
With	A shortcut for accessing objects
Program Exit	Exiting the Program
Naming Conventions	Some ideas on how to name variables
Elements	Old - do not use
Blank Template	Used to create new pages with separate VB, Delphi, C++, and Windows API sections

The syntax of a language specifies how programs in the language are built

The syntax of a programming language describes the structure of programs without any consideration of their meaning.

Semantics :Semantics is concerned with the interpretation or understanding of programs and how to predict the outcome of program execution .

semantics' is what programs mean.

Algorithm:An Algorithm is a stepwise and logical solution to a problem, taking into account all the possible alternatives. An Algorithm can be defined as writing of the plans in simple English what the computer is required to do before writing the actual program.