Languages

To write a program for a computer, we must use a computer language. A computer language is a set of predefined words that are combined into a program according to predefined rules (***syntax***).

Computer understand its own language i.e 0 and 1 (Binary Language). It has two basic types:

Low Level Language

High Level Language

Machine languages

In the earliest days of computers, the only programming languages available were **machine languages**. Each computer had its own machine language, which was made of streams of 0s and 1s.

Machine language is a low-level programming language that consists of binary bits i.e. only 0 and 1. The data present in binary form is the reason for its fast execution. In Machine language, instructions are directly executed by the CPU. Machine language is also known as object code or machine code. Machine language is binary language.

The only language understood by a computer is machine language.

Hexadecimal	Code in Machine Language
1FEF	0001 1111 1110 1111
240F	0010 0100 0000 1111
1FEF	0001 1111 1110 1111
241F	0010 0100 0001 1111
1040	0001 0000 0100 0000
1141	0001 0001 0100 0001
3201	0011 0010 0000 0001
2422	0010 0100 0010 0010
1F42	0001 1111 0100 0010
2FFF	0010 1111 1111 1111
0000	0000 0000 0000 0000

Features of Machine Language:

- Machine language is a low-level language.
- Machine language consist of only 0 and 1 bits.
- Machine languages are platform dependent.
- It is nearly impossible to learn machine language for humans because it requires a lot of memoization.
- Machine language is used to create and construct drivers as well.

Advantages of Machine Language

- Machine languages are faster in execution because they are in binary form.
- Machine language does not need to be translated, because it is already present in simple binary form.
- The CPU directly executes the machine language.
- The evolution of the computer system and operating system over the time period is due to machine language.

Disadvantages of Machine Language

- Machine language is complex to understand and memorize.
- Writing codes in machine language is time-consuming.
- It is very difficult to resolve bugs and errors present in the codes and programs.

- Codes written in machine languages are more prone to error.
- Machine languages are not easy to modify.
- Machine language is platform Independent.

Assembly languages/Symbolic Language

Replacing binary code for instruction and **addresses with symbols or mnemonics**. Assembly language used symbols; these languages were first known as symbolic languages.

Code in Assembly Language			Description
Load	RF	Keyboard	Load from keyboard controller to register F
Store	Number1	RF	Store register F into number 1
Load	RF	Keyboard	Load from keyboard controller to register F
Store	Number2	RF	Store register F into number 2
Load	R0	Number1	Load Number1 into register 0
Load	R1	Number2	Load Number2 into register 1
Add1 R1	R2	R0	Add register 0 and 1 with result in register 2
Store	Result	R2	Store register 2 into result
Load	RF	Result	Load Result into register F
Store	Monitor	RF	Store Register F into monitor controller
HALT			Stop

Advantages of Assembly Language

- It provides precise control over hardware and hence increased code optimization.
- It allows direct access to hardware components like registers, so it enables solutions for hardware issues.
- Efficient resource utilization because of low level control, optimized code, resource awareness, customization etc.
- It is ideal for programming microcontrollers, sensors and other hardware components.
- It is used in security researches for finding security vulnerabilities, reverse engineering software for system security.
- It is very essential for the making the operating systems, kernel and device controllers that requires hardware interaction for its functionality.

Disadvantages of Assembly Language

- Complex and very hard to learn the language especially for beginners.
- It is highly machine dependent. So, it limits portability.
- It is really hard to maintain the code, especially for large scale projects.
- It is very time consuming since it is really hard to understand and very length of code.
- Debugging is very challenging to programmers.

High Level Language

High level languages are programming languages which are used for writing programs or software which could be understood by the humans and computer. High level languages are easier to understand for humans because it uses lot of symbols letters phrases to represent logic and instructions in a program. It contains high level of abstraction compared to low level languages.

Over the years, various languages, BASIC, COBOL, Pascal, Ada, C, C++ and Java, were developed.

eg. program for adding two integers.

//addition of 2 numbers

```
#include<stdio.h>

main()

{

        int a,b,c;

        a=5,b=10;

        c=a+b;

        printf(“addition is \t%d”,c);

        return 0;

}
```

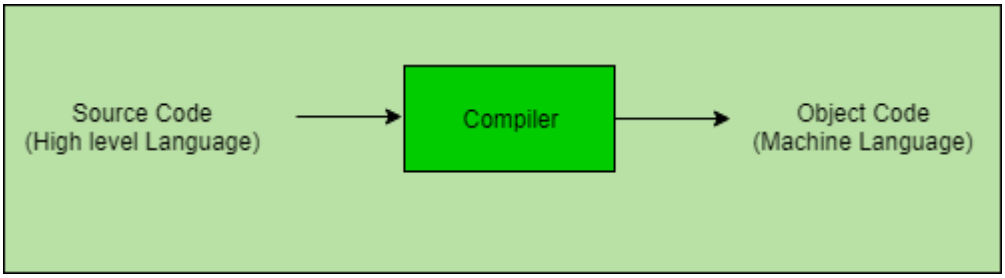
TRANSLATOR

A translator is a programming language processor that converts a computer program from one language to another. It takes a program written in source code and converts it into machine code. The program in a high-level language is called the source program. The translated program in machine language is called the object program. Two methods are used for translation: **compilation** and **interpretation**.

Compilation

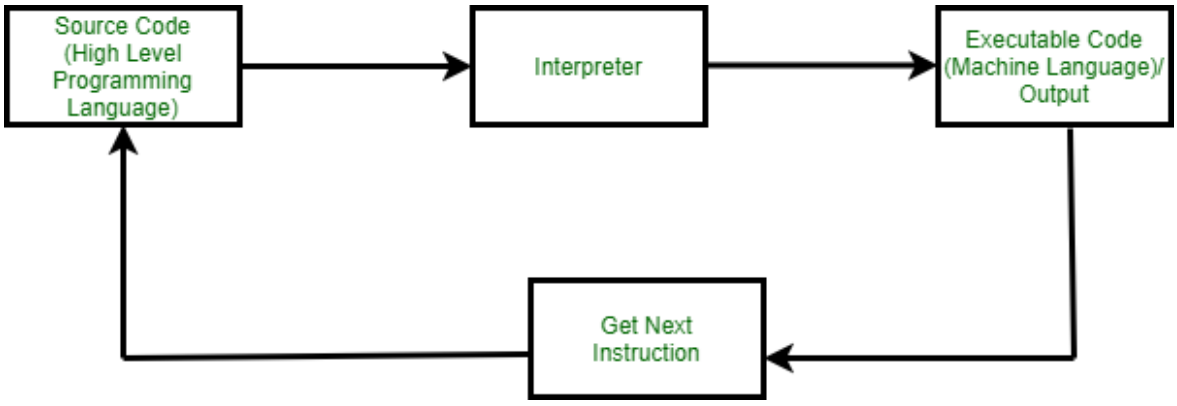
A Compiler is a type of language processor that **takes the entire program written in a high-level language** (like C, C++, C#) and translates it all at once into machine language. It’s like converting the whole code into a language that the computer can understand.

If the source code is free of errors, the compiler can successfully translate it into an object code. However, if there are any errors in the source code, the compiler will point them out at the end of the process, indicating the line numbers where the errors occur.



Interpretation

Interpretation takes **one line of code** at a time and quickly translates it into a language the computer can understand. If there’s a mistake in a line of code, the Interpreter stops and points it out, so we can fix it before moving forward.

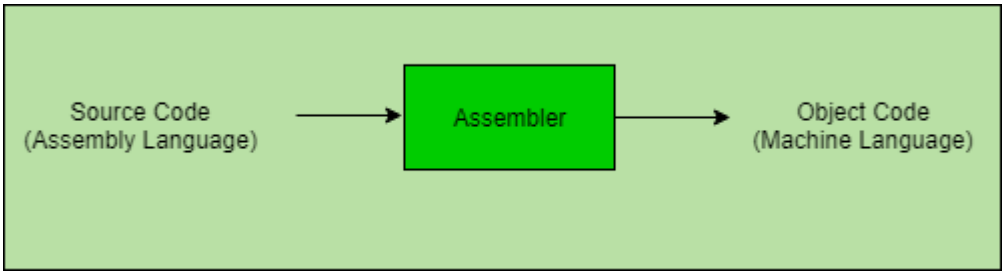


Compilation and interpretation differ in that the first translates the whole source code before executing it, while the second translates and executes the source code a line at a time.

Assembler

An Assembler is a special tool that translates programs **written in Assembly language into machine code,** which the computer can understand. In the source program, we write instructions using simple mnemonics like ADD, MUL, SUB, etc.

It takes the program written in Assembly language as input and produces the machine code as output.



Difference between Compiler and Interpreter

Compiler	**Interpreter**
Compiler Takes **Entire** program as input.	Interpreter Takes **Single** instruction as input.
Intermediate Object Code is **Generated** in case of compiler.	**No** Intermediate Object Code is **Generated.**
Processing is fast.	Processing is slow.
It takes less time to execute.	It takes more time to execute.
Memory Requirement is **More**	**Memory Requirement** is **Less**
Errors are displayed after **entire program** is checked	**Errors** are displayed for **every instruction** interpreted. An interpreter continues translating the program until the first error is met, in which case it stops.
debugging is comparatively hard.	debugging is easy.
Suitable for large programs.	Suitable for small programs.

C, C++ use compiler.	Python, Ruby use interpreter.

Difference between Assembler and Interpreter

Assembler	**Interpreter**
It is a system program that is utilized to convert assembly language code into machine code that can be relocated.	It is a language converter that converts high-level language source code to machine or system language code.
It is a low-level language program.	It is a high-level language program.
It generates an executable file.	It doesn't generate any executable file.
Assembler translates the complete program before execution.	The interpreter translates the complete program at execution time.
It utilizes the source code to generate an executable file.	It utilizes the source code every time during the program execution.
It is quicker because it doesn't fix any external references in the source code.	It is much slower than an assembler because it fixes all external references in the source code.
It needs memory space for its storage.	It doesn't need memory for its storage.
It is designed for specific hardware.	It is designed for a specific language.
It is utilized by assembly language.	It is utilized by Ruby, PHP, Perl, and Python.

Difference between Assembler and Compiler

Assembler	**Compiler**
The assembler converts assembly code into machine code.	Compiler converts program written in a high-level language to machine-level language.
Debugging is tough as compared to the compiler.	Debugging is easy in the case of the compiler.
The assembler is less intelligent as compared to the compiler.	The compiler is more intelligent than the assembler.
An assembler works in two phases over the given input: Analysis Phase and Synthesis Phase.	A compiler works in the following phases: lexical analyzer, semantic analyzer, syntax analyzer, intermediate code generator, code optimizer, symbol table, and error handle.
Assembler converts code into object code then it converts object code into machine code.	The compiler scans the entire program before converting it into machine code.
Examples of assemblers are GNU, GAS, etc.	Examples of compilers are GCC etc.

Linker and Loader

In the execution of the program, major role is played by two utility programs known as **Linker** and **Loader.**

1. **Linker:**
   A linker is a special program that combines the object files generated by a compiler or assembler, along with other pieces of code, to create an executable file with a .exe extension. In the object file, the linker searches for and appends all libraries needed for the execution of the file. It manages the memory space that will hold the code from each module and also merges two or more separate object programs while establishing links among them.

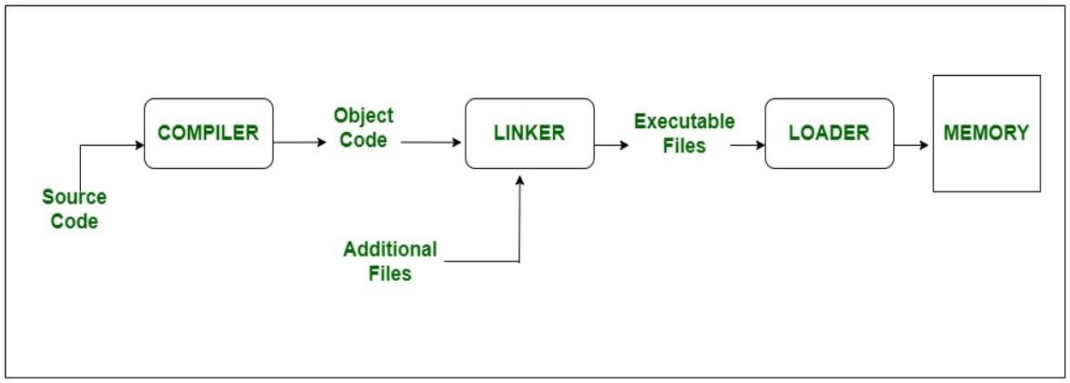
Generally, linkers are of two types :

a) Linkage Editor

b) Dynamic Linker

The linker performs several tasks, including:

- Symbol resolution: The linker resolves symbols in the program that are defined in one module and referenced in another.
- Code optimization: The linker optimizes the code generated by the compiler to reduce code size and improve program performance.
- Memory management: The linker assigns memory addresses to the code and data sections of the program and resolves any conflicts that arise.
- Library management: The linker can link external libraries into the executable file to provide additional functionality.



2. **Loader: It is special program that takes input of executable files from linker, loads it to main memory, and prepares this code for execution by computer. Loader allocates memory space to program**. Even it settles down symbolic reference between objects. It is in charge of loading programs and libraries in operating system. The embedded computer systems don't have loaders. In them, code is executed through ROM. There are following various loaders:

a) Absolute Loaders
b) Relocating Loaders
c) Direct Linking Loaders
d) Bootstrap Loaders

The loader performs several tasks, including:

- Loading: The loader loads the executable file into memory and allocates memory for the program.
- Relocation: The loader adjusts the program's memory addresses to reflect its location in memory.
- Symbol resolution: The loader resolves any unresolved external symbols that are required by the program.
- Dynamic linking: The loader can dynamically link libraries into the program at runtime to provide additional functionality.

Differences between Linker and Loader are as follows:

LINKER	LOADER
The main function of Linker is to generate executable files.	Whereas main objective of Loader is to load executable files to main memory.
The linker takes input of object code generated by compiler/assembler.	And the loader takes input of executable files generated by linker.
Linking can be defined as process of combining various pieces of codes and source code to obtain executable code.	Loading can be defined as process of loading executable codes to main memory for further execution.
Linkers are of 2 types: Linkage Editor and Dynamic Linker.	Loaders are of 4 types: Absolute, Relocating, Direct Linking, Bootstrap.
Another use of linker is to combine all object modules.	It helps in allocating the address to executable codes/files.
Linker is also responsible for arranging objects in program's address space.	Loader is also responsible for adjusting references which are used within the program.

Summary:

Linker and Loader are both important components in the software development process. **The Linker is used during the compilation process to link object files into a single executable file, while the Loader is used at runtime to load the executable file into memory and prepare it for execution.**