Chapter 2: Introduction to Programming Languages

2.1 Programming Languages

The computer system is simply a machine and hence it cannot perform any work; therefore, in order to make it functional different languages are developed, which are known as programming languages or simply computer languages.

Over the last two decades, dozens of computer languages have been developed. Each of these languages comes with its own set of vocabulary and rules, better known as syntax. Furthermore, while writing the computer language, syntax has to be followed literally, as even a small mistake will result in an error and not generate the required output.

Following are the major categories of Programming Languages −

- Machine Language
- Assembly Language
- High Level Language
- System Language
- Scripting Language

Let us discuss the programming languages in brief.

Machine Language or Code

This is the language that is written for the computer hardware. Such language is effected directly by the central processing unit (CPU) of a computer system.

Assembly Language

It is a language of an encoding of machine code that makes simpler and readable.

High Level Language

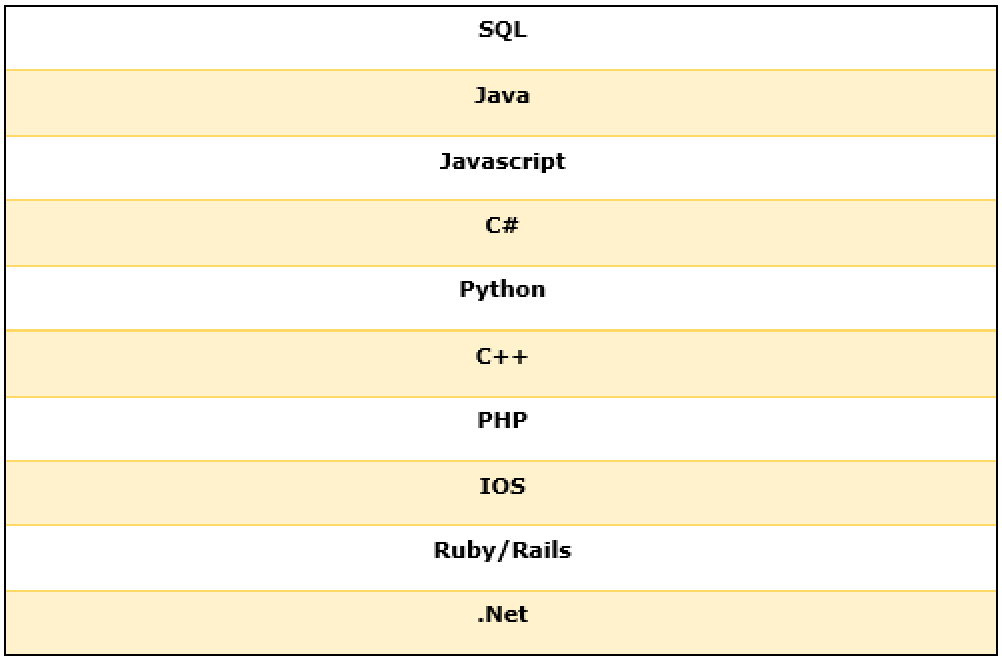
The high level language is simple and easy to understand and it is similar to English language. For example, COBOL, FORTRAN, BASIC, C, C+, Python, etc.

High-level languages are very important, as they help in developing complex software and they have the following advantages −

- Unlike assembly language or machine language, users do not need to learn the high-level language in order to work with it.
- High-level languages are similar to natural languages, therefore, easy to learn and understand.
- High-level language is designed in such a way that it detects the errors immediately.
- High-level language is easy to maintain and it can be easily modified.
- High-level language makes development faster.
- High-level language is comparatively cheaper to develop.
- High-level language is easier to document.

Although a high-level language has many benefits, yet it also has a drawback. It has poor control on machine/hardware.

The following table lists down the frequently used languages −



SQL
Java
Javascript
C#
Python
C++
PHP
IOS
Ruby/Rails
.Net

2.2 The evolution of Programming Languages

Programming Language is indeed the fundamental unit of today's tech world. It is considered as the set of commands and instructions that we give to the machines to perform a particular task. For example, if you give some set of instructions to add two numbers then the machine will do it for you and tell you the correct answer accordingly. But do you know that Programming Languages are having a long and rich history of their evolution? And with a similar concern, here in this article, we'll take a look at the evolution of Programming Languages over the period.

In the computer world, we have about 500+ programming languages with having their own syntax and features. And if you type who's the father of the computer, then the search engine will show you the result as to Charles Babbage but the father of the computer didn't write the first code. It was Ada Lovelace who has written the first-ever computer programming language and the year was 1883.

1883: The Journey starts from here…!!

- In the early days, Charles Babbage had made the device, but he was confused about how to give instructions to the machine, and then Ada Lovelace wrote the instructions for the analytical engine.
- The device was made by Charles Babbage and the code was written by Ada Lovelace for computing Bernoulli's number.
- First time in history that the capability of computer devices was judged.

1949: Assembly Language

- It is a type of low-level language.
- It mainly consists of instructions (kind of symbols) that only machines could understand.
- In today’s time also assembly language is used in real-time programs such as simulation flight navigation systems and medical equipment eg – Fly-by-wire (FBW) systems.
- It is also used to create computer viruses.

1952: Autocode

- Developed by Alick Glennie.
- The first compiled computer programming language.
- COBOL and FORTRAN are the languages referred to as Autocode.

1957: FORTRAN

- Developers are John Backus and IBM.
- It was designed for numeric computation and scientific computing.
- Software for NASA probes voyager-1 (space probe) and voyager-2 (space probe) was originally written in FORTRAN 5.

1958: ALGOL

- ALGOL stands for **ALGO**rithmic **L**anguage.
- The initial phase of the most popular programming languages of C, C++, and JAVA.
- It was also the first language implementing the nested function and has a simple syntax than FORTRAN.

- The first programming language to have a code block like “begin” that indicates that your program has started and “end” means you have ended your code.

1959: COBOL

- It stands for **CO**mmon **B**usiness-**O**riented **L**anguage.
- In 1997, 80% of the world’s business ran on Cobol.
- The US internal revenue service scrambled its path to COBOL-based IMF (individual master file) in order to pay the tens of millions of payments mandated by the coronavirus aid, relief, and economic security.

1964: BASIC

- It stands for beginners All-purpose symbolic instruction code.
- In 1991 Microsoft released Visual Basic, an updated version of Basic
- The first microcomputer version of Basic was co-written by Bill Gates, Paul Allen, and Monte Davidoff for their newly-formed company, Microsoft.

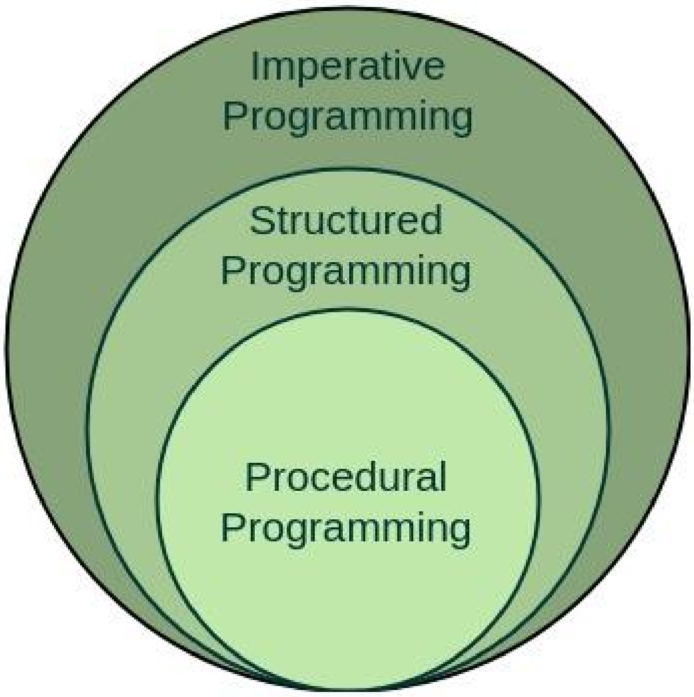
1972: C

- It is a general-purpose, procedural programming language and the most popular programming language till now.
- All the code that was previously written in assembly language gets replaced by the C language like operating system, kernel, and many other applications.
- It can be used in implementing an operating system, embedded system, and also on the website using the Common Gateway Interface (CGI).
- C is the mother of almost all higher-level programming languages like C#, D, Go, Java, JavaScript, Limbo, LPC, Perl, PHP, Python, and Unix’s C shell.

Structured Programming: Structured Programming Approach, as the word suggests, can be defined as a programming approach in which the program is made as a single structure. It means that the code will execute the instruction by instruction one after the other. It doesn't support the possibility of jumping from one instruction to some other with the help of any statement like GOTO, etc. Therefore, the instructions in this approach will be executed in a serial and structured manner. The languages that support Structured programming approach are:C

- C++
- Java
- C#

  ..etc



On the contrary, in the Assembly languages like Microprocessor 8085, etc, the statements do not get executed in a structured manner. It allows jump statements like GOTO. So the program flow might be random.

The structured program mainly consists of three types of elements:

- Selection Statements
- Sequence Statements
- Iteration Statements

The structured program consists of well structured and separated modules. But the entry and exit in a Structured program is a single-time event. It means that the program uses single-entry and single-exit elements. Therefore a structured program is well maintained, neat and clean program. This is the reason why the Structured Programming Approach is well accepted in the programming world.

Advantages of Structured Programming Approach:

1. Easier to read and understand
2. User Friendly
3. Easier to Maintain
4. Mainly problem based instead of being machine based
5. Development is easier as it requires less effort and time
6. Easier to Debug
7. Machine-Independent, mostly.

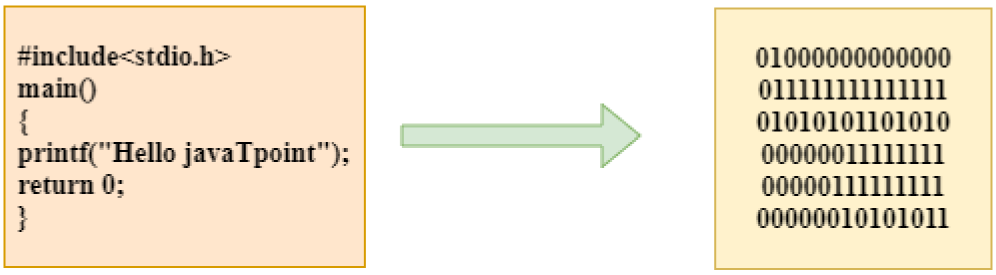
Disadvantages of Structured Programming Approach:

1. Since it is Machine-Independent, So it takes time to convert into machine code.
2. The converted machine code is not the same as for assembly language.

3. The program depends upon changeable factors like data-types. Therefore it needs to be updated with the need on the go.
4. Usually the development in this approach takes a longer time as it is language-dependent. Whereas in the case of assembly language, the development takes less time as it is fixed for the machine.

What is a compilation?

The compilation is a process of converting the source code into object code. It is done with the help of the compiler. The compiler checks the source code for the syntactical or structural errors, and if the source code is error-free, then it generates the object code.



```
#include<stdio.h>
main()
{
printf("Hello javaTpoint");
return 0;
}
```

```
01000000000000
0111111111111111
01010101101010
00000011111111
00000111111111
00000010101011
```

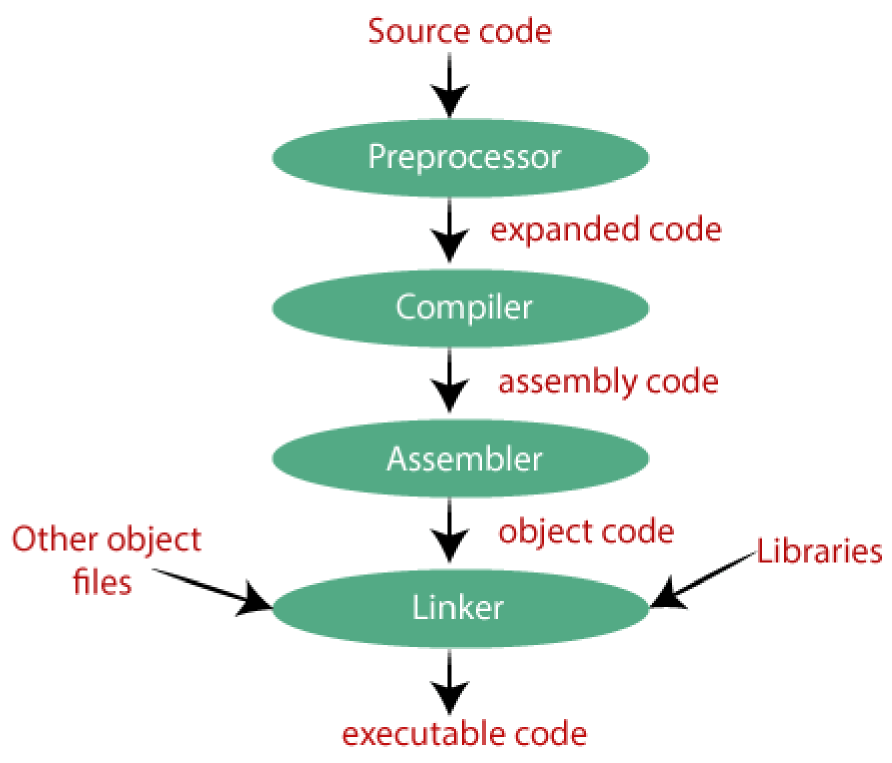
The c compilation process converts the source code taken as input into the object code or machine code. The compilation process can be divided into four steps, i.e., Pre-processing, Compiling, Assembling, and Linking.

The preprocessor takes the source code as an input, and it removes all the comments from the source code. The preprocessor takes the preprocessor directive and interprets it. For example, if **<stdio.h>,** the directive is available in the program, then the preprocessor interprets the directive and replace this directive with the content of the **'stdio.h'** file.

The following are the phases through which our program passes before being transformed into an executable form:

- **Preprocessor**
- **Compiler**

- **Assembler**
- **Linker**



Preprocessor

The source code is the code which is written in a text editor and the source code file is given an extension ".c". This source code is first passed to the preprocessor, and then the preprocessor expands this code. After expanding the code, the expanded code is passed to the compiler.

Compiler

The code which is expanded by the preprocessor is passed to the compiler. The compiler converts this code into assembly code. Or we can say that the C compiler converts the pre-processed code into assembly code.

Assembler

The assembly code is converted into object code by using an assembler. The name of the object file generated by the assembler is the same as the source file. The extension of the object file in DOS is '.obj,' and in UNIX, the extension is 'o'. If the name of the source file is **'hello.c',** then the name of the object file would be 'hello.obj'.

Linker

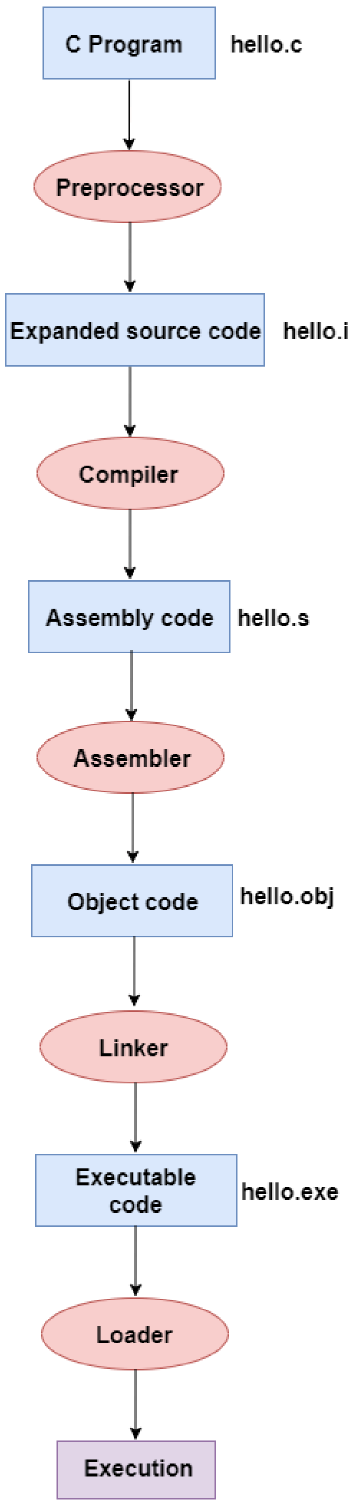
Mainly, all the programs written in C use library functions. These library functions are pre-compiled, and the object code of these library files is stored with '.lib' (or '.a') extension. The main working of the linker is to combine the object code of library files with the object code of our program. Sometimes the situation arises when our program refers to the functions defined in other files; then linker plays a very important role in this. It links the object code of these files to our program. Therefore, we conclude that the job of the linker is to link the object code of our program with the object code of the library files and other files. The output of the linker is the executable file. The name of the executable file is the same as the source file but differs only in their extensions. In DOS, the extension of the executable file is '.exe', and in UNIX, the executable file can be named as 'a.out'. For example, if we are using printf() function in a program, then the linker adds its associated code in an output file.

Let's understand through an example.

hello.c

```c
#include <stdio.h>
int main()
{
   printf("Hello javaTpoint");
   return 0;
}
```

Now, we will create a flow diagram of the above program:



In the above flow diagram, the following steps are taken to execute a program:

- Firstly, the input file, i.e., hello.c, is passed to the preprocessor, and the preprocessor converts the source code into expanded source code. The extension of the expanded source code would be hello.i.

- The expanded source code is passed to the compiler, and the compiler converts this expanded source code into assembly code. The extension of the assembly code would be hello.s.
- This assembly code is then sent to the assembler, which converts the assembly code into object code.
- After the creation of an object code, the linker creates the executable file. The loader will then load the executable file for the execution.

Source Code :

Source code refers to high level code or assembly code which is generated by a human/programmer. Source code is easy to read and modify. It is written by a programmer using any High Level Language or Intermediate language which is human-readable. Source code contains comments that programmers puts for better understanding.

Source code is provided to language translator which converts it into machine understandable code which is called machine code or object code. Computer can not understand direct source code, computer understands machine code and executes it. It is considered as fundamental component of computer. In simple we can say source code is a set of instructions/commands and statements which is written by a programmer by using a computer programming language like C, C++, Java, Python, Assembly language etc. So statements written in any programming language is termed as source code.

2. Object Code :

Object code refers to low level code which is understandable by machine. Object code is generated from source code after going through compiler or other translator. It is in executable machine code format. Object code contains a sequence of machine understandable instructions to which Central Processing Unit understands and executes.

Object file contains object code. It is considered as one more of machine code. Some object file examples are common object file format (COFF), COM files and “.exe” files. It is the output of a compiler or other translator. We can understand source code but we can not understand object code as it is not in plain text like source code rather it is in binary formats.

Difference between Source Code and Object Code :

S.No.	SOURCE CODE	OBJECT CODE
01.	Source code is generated by human or programmer.	Object code is generated by compiler or other translator.
02.	Source code is high level code.	Object code is low level code.

03.	Source code is written in plain text by using some high level programming language.	Object code is translated code of source code. It is in binary format.
04.	Source code is human understandable.	Object code is not human understandable.
05.	Source code is not directly understandable by machine.	Object code is machine understandable and executable.
06.	It is written in high level language like C, C++, Java, Python etc or assembly language.	It is written in machine language through compiler or assembler or other translator.

07.	It can be easily modified.	It can not be modified.
08.	It contains comments for better understanding by programmer.	It does not contain comments for understanding by machine.
09.	It contains less number of statements than object code.	It contains more number of statements than source code.
10.	It is less close. towards machine.	It is more close towards machine.

11.	Performance of source code is less than object code as it is less close towards machine.	Performance of object code is more than source code as it is more close towards machine.
12.	Source code is input to compiler or any other translator.	Object code is output of compiler or any other translator.
13.	Source code is not system specific.	Object code is system specific.
14.	It can be changed over time.	Source code needs to be compiled or translated by any other translator to get modified object code.

Executable File :

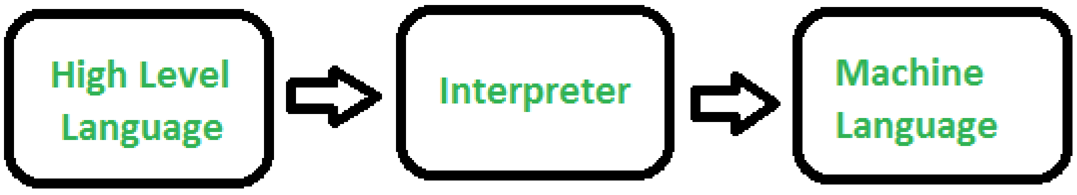
Executable File, as name suggests, are computer files that contain encoded sequence of instructions and is particular a kind of file that is capable of being executed or run as a program in computer.

Introduction to Interpreters

All high level languages need to be converted to machine code so that the computer can understand the program after taking the required inputs.

The software by which the conversion of the high level instructions is performed line-by-line to machine level language, other than compiler and assembler, is known as INTERPRETER. If an error is found on any line, the execution stops till it is corrected. This process of correcting errors is easier as it gives line-by-line error but the program takes more time to execute successfully. Interpreters were first used in 1952 to ease programming within the limitations of computers at the time.

It translates source code into some efficient intermediate representation and immediately execute this.



Source programs are compiled ahead of time and stored as machine independent code, which is then linked at run-time and executed by an interpreter. An Interpreter is generally used in micro computer. It helps the programmer to find out the errors and to correct them before control moves to the next statement.

Interpreter system performs the actions described by the high level program. For interpreted programs, the source code is needed to run the program every time. Interpreted programs run slower than the compiled programs.

Self-Interpreter is a programming language interpreter which is written in a language which can interpret itself.

For Example- **BASIC** interpreter written in **BASIC**. They are related to self-hosting compilers. Some languages have an elegant and self-interpreter such as Lisp and Prolog.

Need of an Interpreter :

The first and vital need of an interpreter is to translate source code from high-level language to machine language. However, for this purpose Compiler is also there to satisfy this condition.

The compiler is a very powerful tool for developing programs in high-level language. However, there are several demerits associated with the compiler. If the source code is huge in size, then it might take hours to compile the source code, which will significantly increase the compilation duration. Here, Interpreter plays its role. They can cut this huge compilation duration. They are designed to translate single instruction at a time and execute them immediately.

Advantage and Disadvantage of Interpreter :

- **Advantage of interpreter** is that it is executed line by line which helps users to find errors easily.
- **Disadvantage of interpreter** is that it takes more time to execute successfully than compiler.

Applications of Interpreters :

- Each operator executed in a command language is usually an invocation of a complex routine, such as an editor or compiler so they are frequently used to command languages and glue languages.
- Virtualization is often used when the intended architecture is unavailable.
- Sand-boxing

- Self-modifying code can be easily implemented in an interpreted language.
- Emulator for running Computer software written for obsolete and unavailable hardware on more modern equipment.

Some examples of programming languages that use interpreters are Phyton, Ruby, Perl, PHP and Matlab.

Top Interpreters according to the computer languages –

- Phyton- CPhyton, PyPy, Stackless Phyton, IronPhyton
- Ruby- YARV, Ruby MRI (CRuby)
- JAVA- HotSpot, OpenJ9, JRockIt
- Kotlin- JariKo

Loader :Loader is the **program of the operating system** which loads the executable from the disk into the primary memory(RAM) for execution. It allocates the memory space to the executable module in main memory and then transfers control to the beginning instruction of the program .

What is Algorithm? Algorithm Basics

The word **Algorithm** means "a process or set of rules to be followed in calculations or other problem-solving operations". Therefore Algorithm refers to a set of rules/instructions that step-by-step define how a work is to be executed upon in order to get the expected results.

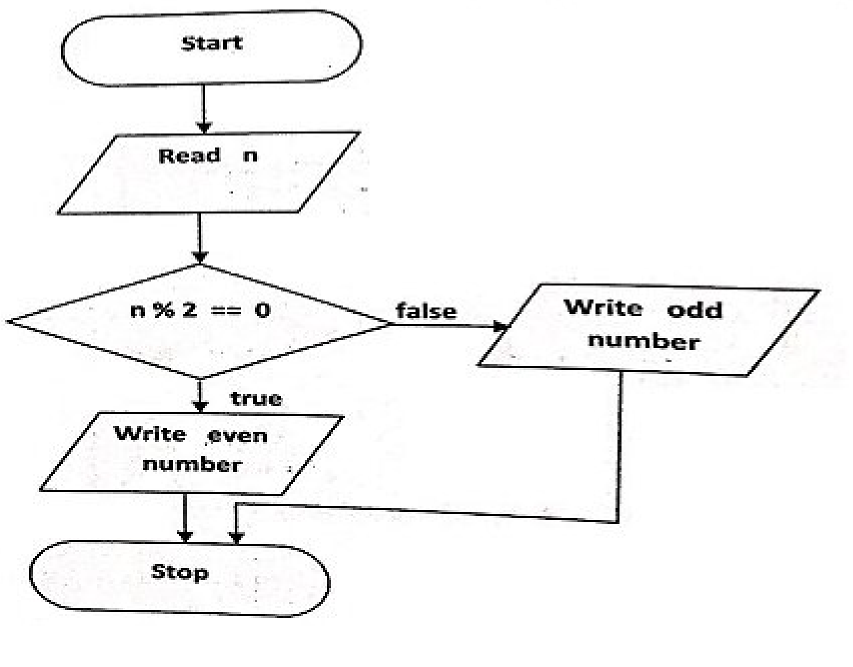
It can be understood by taking an example of cooking a new recipe. To cook a new recipe, one reads the instructions and steps and execute them one by one, in the given sequence. The result thus obtained is the new dish cooked perfectly. Similarly, algorithms help to do a task in programming to get the expected output.

The Algorithm designed are language-independent, i.e. they are just plain instructions that can be implemented in any language, and yet the output will be the same, as expected.

Algorithm to check whether the given number is even or odd

Algorithm

```
Step 1: Start
Step 2: [ Take Input ] Read: Number
Step 3: Check: If Number%2 == 0 Then
Print : N is an Even Number.
Else
Print : N is an Odd Number.
Step 4: Exit
```



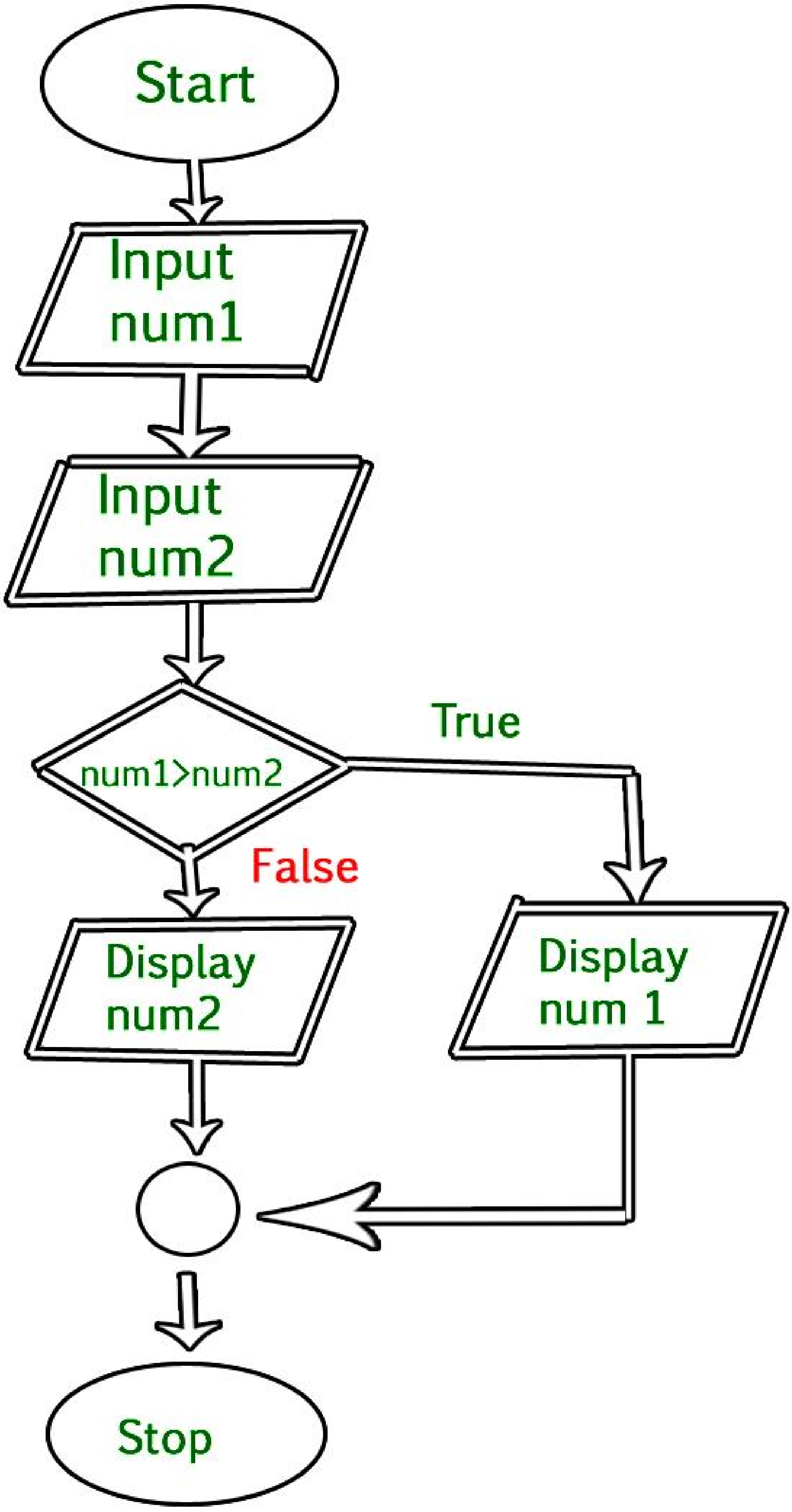
What is a Flowchart?

Flowchart is a graphical representation of an algorithm. Programmers often use it as a program-planning tool to solve a problem. It makes use of symbols which are connected among them to indicate the flow of information and processing.

The process of drawing a flowchart for an algorithm is known as “flowcharting”.

Basic Symbols used in Flowchart Designs

Example : Draw a flowchart to input two numbers from user and display the largest of two numbers


Start
Input num1
Input num2
num1>num2
True
False
Display num2
Display num 1
Stop