

A programming language is a language designed to communicate instructions to a computer. They are used to create programs that control the behavior of a machine.

- Syntax is a set of rules that define how the commands have to be arranged to make sense and to be correctly translatable to the machine code.
- Grammar is a set of rules of using different punctuation, quotation marks, semicolons, and other symbols to divide and clarify the syntax of a particular language
- Semantics is a set of meanings assigned to every command of the language and is used to properly translate the programme to machine code.

Machine Language: Machine language is a collection of binary digits or bits that the computer reads and interprets. Machine language is the only language a computer is capable of understanding

Assembly language: Assembly language is easier to use than machine language. An assembler is useful for detecting programming errors. Programmers do not have the absolute address of data items. Assembly language encourage modular programming

High level language High level language is a language that supports the human and the application sides of the programming. A line in a high-level language can execute powerful operations. and correspond to tens, or hundreds, of instructions at the machine level e BASIC, FORTRAN

Attribute of good Programming Language

Clarity, Simplicity and Unity: A Programming language provides both a conceptual framework for Algorithm planning and means of expressing them.

Orthogonality: Orthogonality is one of the most important features of PL. It is the property that says "Changing A does not change B" When the features of a language are orthogonal, language is easier to learn and programs are easier to write because only few exceptions and special cases to be remembered.

Support for Abstraction:

Ease of program verification/Reusability: The reusability of program written in a language is always a central concern. A program is checked by various testing technique like Formal verification method Desk checking Input output test checking e.t.c Simplicity of semantic and syntactic structure is a primary aspect that tends to simplify program verification.

Portability of programs: Programming language should be portable means it should be easy to transfer a program from which they are developed to the other computer.

Program performance and features of programming languages A programming language's features include orthogonality or simplicity, available control structures, data types and data structures, syntax

design, support for abstraction, expressiveness, type equivalence, and strong versus weak type checking, exception handling, and restricted aliasing. **While the performance of a program, including reliability, readability, writability, reusability, and efficiency, is largely determined by the way the programmer writes the algorithm and selects the data structures,**

Programming language evolution

1883: The Beginning ...!! 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.

1949: Assembly Language: a type of low-level language. It mainly consists of instructions (kind of symbols) that only machines could understand. In today's time, assembly language is used in real-time programs such as simulation flight navigation systems and medical equipment e.g. – Fly-by-wire (FBW) systems. It is also used to create computer viruses.

1954 - : FORTRAN: FORTRAN was developed in 1954 by 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. It is first high level language. It was developed using the first compiler and it is Machine Independent Language. FORTRAN 2nd version was developed which introduces subroutines, functions, loops and primitive for loop. It started as a Project and later renamed as ALGOL58

1958: ALGOL ALGOL stands for ALGOritmic Language, which is 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.

ALGOL brought concepts like Block structure code (Marked by BEGIN and END), Scope of variables (Scope of local variables inside blocks), BNF (Backus Naur Form), Notation for defining syntax, Dynamic Arrays, Reserved words and IF THEN ELSE, FOR, WHILE loop

1959: COBOL It stands for **CO**mmun **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 BASIC (Beginner's All-purpose Symbolic Instruction Code). It was designed as a teaching language in 1963 by John George Kemeny and Thomas Eugene Kurtz of Dartmouth college. Intended to make it easy to learn programming. In 1991 Microsoft released Visual Basic, an updated version of Basic but the first microcomputer version of Basic was co-written by Bill Gates, Paul Allen, and Monte Davidoff for their newly-formed company, Microsoft



1970: Pascal Pascal is named after a French religious fanatic and mathematician Blaise Pascal. It was Created in 1970 with the intension of replacing BASIC for teaching language. It was quickly developed as a general purpose language.

1972: C: C is a general-purpose, procedural programming language and the most popular till now. All the previous codes (like operating system and kernel) written in assembly language gets replaced by the C language. C can be used to implementing operating system, embedded system, and 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.

Programming Language Paradigm: A programming paradigm is an approach to programming a computer based on a coherent set of principles or mathematical theory.

- **different programming languages share the same mathematical foundation (e.g., Boolean algebra, logic);**
- they provide similar functionality (e.g., arithmetic, logic operations, and text processing);
- they are based on the same kind of hardware and instruction sets;
- they have common design goals: find languages that make it simple for humans to use and efficient for hardware to execute;
- designers of programming languages share their design experiences.

Categories of Programming Paradigm

- Imperative paradigm
- Functional paradigm,
- Logic paradigm
- Object-Oriented paradigm
- Visual paradigm
- Parallel/concurrent paradigms, • Constraint based paradigm • Dynamic paradigms.

There are four main programming paradigms which are **imperative paradigm functional paradigm, logical paradigm and object-oriented paradigm.**

Imperative Paradigm The imperative, also called the procedural programming paradigm expresses



computation by fully specified and controlled manipulation of named data in a stepwise fashion. Typical imperative programming languages include all assembly languages and earlier high-level languages like Fortran, Algol, Ada, Pascal, and C

Object-Oriented Paradigm The object-oriented programming paradigm is basically the same as the imperative paradigm, except that related variables and operations on variables are organized into classes of objects. Typical object-oriented programming languages include Smalltalk, C++, Python, Java, and C#

Functional (Application) Paradigm also called the applicative, programming paradigm expresses computation in terms of mathematical functions. Since we express computation in mathematical functions in many of the mathematics courses, functional programming is supposed to be easy to understand and simple to use. However, programmers find it difficult to switch because they are already familiar with the functional programming. **The main difference is that there is no concept of memory locations in functional programming languages.**

Logic Paradigm also called the declarative, programming paradigm expresses computation in terms of logic predicates. A logic program is a set of facts, rules, and questions. The execution process of a logic program is to compare a question to each fact and rule in the given fact and rulebase

Structured programming

(sometimes known as modular programming) is a programming paradigm that facilitates the creation of programs with readable code and reusable components. All modern programming languages support structured programming, but the mechanisms of support, like the syntax of the programming languages, varies. Structured programming encourages dividing an application program into a hierarchy of modules or autonomous elements, which may, in turn, contain other such elements. Within each element, code may be further structured using blocks of related logic designed to improve readability and maintainability.

- **Block:** It is a command or a set of commands that the program executes linearly. The sequence has a single point of entry (first line) and exit (last line).

- **Selection:** It is the branching of the flow of control based on the outcome of a condition.

Two sequences are specified: the 'if' block when the condition is true and the 'else' block when it is false. The 'else' block is optional and can be a no-op.

- **Iteration:** It is the repetition of a block as long as it meets a specific condition. The evaluation of the condition happens at the start or the end of the block. When the condition

results in false, the loop terminates and moves on to the next block.

- Nesting: The above building blocks can be nested because conditions and iterations, when encapsulated, have singular entry-exit points and behave just like any other block.
- Subroutines: Since entire programs now have singular entry-exit points, encapsulating them into subroutines allows us to invoke blocks by one identifier.

Difference between structured and unstructured programming languages

A structured programming language facilitates or enforces structured programming practices.

The theoretical basis for structured programming goes back to the 1950s, with the emergence of the ALGOL 58 and 60 languages

code clarity was reduced by the need to build condition/action tests by having programmers write linked tests and actions explicitly (using the goto statement or its equivalent), resulting in what was often called spaghetti code

Modular programming, which is today seen as synonymous with structured programming, emerged a decade later as it became clear that reuse of common code could improve developer productivity. In modular programming, a program is divided into semi-independent modules, each of which are called when needed

Types of structured programming

Procedural programming.

Defines modules as "procedures" or "functions" that are called with a set of parameters to perform

A task. A procedural language will begin a process, which is then given data. It is also the most common category and has recently been subdivided into the following:

- Service-oriented programming simply defines reusable modules as "services" with advertised interfaces.
- Microservice programming focuses on creating modules that do not store data internally, and so are scalable and resilient in cloud deployment.
- Functional programming, technically, means that modules are written from functions, and

that these functions' outputs are derived only from their inputs. Designed for server less computing, the definition of functional programming has since expanded to be largely synonymous with microservices

Object-oriented programming (OOP). Defines a program as a set of objects or resources to which commands are sent. An object-oriented language will define a data resource and send it to process commands.

Model-based programming. The most common example of this is database query languages. In database programming, units of code are associated with steps in database access and update or run when those steps occur.

Advantages of structured programming

- It encourages top-down implementation, which improves both readability and maintainability of code.
- It promotes code reuse, since even internal modules can be extracted and made independent, residents in libraries, described in directories and referenced by many other applications.
- It's widely agreed that development time and code quality are improved through structured programming.

Disadvantages of structured programming

The biggest disadvantage of structured programming is a reduction in execution efficiency greater memory usage.

Concept of Language Structure

The structures of programming languages are grouped into four structural layers which are **lexical, syntactic, contextual, and semantic**

Lexical structure defines the vocabulary of a language. Lexical units are considered the building blocks of programming languages. The lexical structures of all programming languages are similar and normally include the following kinds of units:

Keywords: Names reserved by the language designer and used to form the syntactic structure of the language.

- **Operators:** Symbols used to represent the operations.

Separators: Symbols used to separate lexical or syntactic units of the language. Space, comma, colon, semicolon, and parentheses are used as separators.

Literals: Values that can be assigned to variables of different types.

Comments:

Layout and spacing: Some languages are of free format such as C, C++, and Java. They use braces and parentheses for defining code blocks and separations.

Syntactic structure: Syntactic structure defines the grammar of forming sentences or statements using the lexical units. An imperative programming language normally offers the following basic kinds of statements:

- **Assignments:** An assignment statement assigns a literal value or an expression to a variable.
- **Conditional statements:** A conditional statement tests a condition and branches to a certain statement based on the test result (true or false). Typical conditional statements are if-then, if-then- else, and switch (case).
- **Loop statements:** A loop statement tests a condition and enters the body of the loop or exits the loop based on the test result (true or false). Typical loop statements are for-loop and while-loop.

Contextual structure : (also called static semantics) defines the program semantics before dynamic execution. It includes variable declaration, initialization, and type checking A statement that is lexically correct may not be contextually correct.

Semantic structure: Semantic structure describes the meaning of a program, or what the program does during the execution. The semantics of a language are often very complex. In most imperative languages, there is no formal definition of semantic structure.

Error types at different levels

Lexical errors: Errors at the lexical level. Compiler can detect all.

Syntactic errors: Errors at the syntactic level. Compiler can detect all of them

Contextual errors: Contextual errors are complex and compiler implementations may or may not detect all of the initialization errors, depending on whether they actually compute the initialization expression or not.

Semantic errors: Errors at the semantic level include all the errors in the statements that will be

executed after passing compilation. The compiler normally does not detect semantic errors.

BNF Notation **BNF (Backus-Naur Form)** is a meta language that can be used to define the lexical and syntactic structures of another language.

What is lexeme and token. A Lexeme is a string of characters that is a lowest-level syntactic unit in the programming language while a Token is a syntactic category that forms a class of lexemes

- How are programming languages formally defined? Languages can be formally described in two distinct ways which are by recognition and by generation.
- In which form is the programming language syntax commonly described Grammars are commonly used to describe the syntax of programming languages. The forms of the tokens of programming languages can be described by regular grammars while the syntax of whole programming languages, with minor exceptions, can be described by context-free grammars
- What is an ambiguous grammar? A grammar that generates a sentential form for which there are two or more distinct parse trees is said to be ambiguous.
- Explain the use of metasymbols in EBNFs. The brackets, braces, and parentheses in the EBNF extensions are metasymbols, which means they are notational tools and not terminal symbols in the syntactic entities they help describe. In cases where these metasymbols are also terminal symbols in the language being described, the instances that are terminal symbols can be underlined or quoted
- What is the purpose of a predicate function? . Predicate functions state the static semantic rules of the language which is used to check if the input meets some condition.
- What is the use of intrinsic attributes? Intrinsic attributes are synthesized attributes of leaf nodes whose values are determined outside the parse tree.

The syntax analyzer is the heart of a compiler, because several other important components, including the semantic analyzer and the intermediate code generator, are driven by the actions of the syntax analyzer. Syntax analyzers are based directly on the grammars

Lexical Analysis

A **lexical analyzer** is essentially a pattern matcher. A pattern matcher attempts to find a substring of a given string of characters that matches a given character pattern. Pattern matching is a traditional part of computing. One of the earliest uses of pattern matching was with text editors, such as the ed line editor, which was introduced in an early version of UNIX. **A lexical analyzer performs syntax analysis at the lowest level of program structure**

Building Lexical Analyzer

- Write a formal description of the token patterns of the language using a descriptive language related to regular expressions. These descriptions are used as input to a software tool that automatically generates a lexical analyzer. There are many such tools available for this. The oldest of these, named lex, is commonly included as part of UNIX systems.
- Design a state transition diagram that describes the token patterns of the language and write a program that implements the diagram.
- Design a state transition diagram that describes the token patterns of the language and hand construct a table-driven implementation of the state diagram.

The Parsing Problem

The part of the process of analyzing syntax that is referred to as syntax analysis is often called parsing.

Parsers for programming languages construct parse trees for given programs. In some cases, the parse tree is only implicitly constructed, meaning that perhaps only a traversal of the tree is generated.

The parse tree (or its trace) is used as the basis for translation.

Parsers are categorized according to the direction in which they build parse trees. The two broad classes of parsers are top-down, in which the tree is built from the root downward to the leaves, and bottom-up, in which the parse tree is built from the leaves upward to the root

set of notational conventions for grammar symbols and strings

- Terminal symbols—lowercase letters at the beginning of the alphabet (a, b, . . .)
- Nonterminal symbols—uppercase letters at the beginning of the alphabet (A, B, . . .)
- Terminals or nonterminals—uppercase letters at the end of the alphabet (W, X, Y, Z)
- Strings of terminals—lowercase letters at the end of the alphabet (w, x, y, z)
- Mixed strings (terminals and/or nonterminals)—lowercase Greek letters (α , β , δ , γ)

Top-Down Parsers

A top-down parser traces or builds a parse tree in preorder. A preorder traversal of a parse tree begins with the root. Each node is visited before its branches are followed. Branches from a particular node are followed in left-to-right order. This corresponds to a leftmost derivation.

Bottom-Up Parsers

A bottom-up parser constructs a parse tree by beginning at the leaves and progressing toward the root. This parse order corresponds to the reverse of a rightmost derivation. That is, the sentential forms of the derivation are produced in order of last to first.

The Recursive-Descent Parsing Process

A recursive-descent parser is so named because it consists of a collection of subprograms, many of which are recursive, and it produces a parse tree in top-down order. This recursion is a reflection of the nature of programming languages, which include several different kinds of nested structures.

Shift-Reduce Algorithms

Bottom-up parsers are often called shift-reduce algorithms, because shift and reduce are the two most common actions they specify. An integral part of every bottom-up parser is a stack. As with other parsers, the input to a bottom-up parser is the stream of tokens of a program and the output

is a sequence of grammar rules. Uses PDA (Push Down Automation)

LR Parsers

Many different bottom-up parsing algorithms have been devised. Most of them are variations of a process called LR. (1) They require far less computer resources to produce the required parsing table than the canonical LR algorithm, and (2) they work on smaller classes of grammars than the canonical LR algorithm.

There are three advantages to LR parsers:

- They can be built for all programming languages.
- They can detect syntax errors as soon as it is possible in a left-to-right scan.
- The LR class of grammars is a proper superset of the class parsable by LL parsers

Interpretation

an interpreter is a program that repeatedly executes the following sequence.

- Get the next statement;
- Determine the actions to be executed;
- Perform the actions

Translation

programs written in a high-level language are translated into an equivalent machine-language version before being executed. This translation is often performed in several steps

Structuring Data

A data structure is a way of organizing

information, so that it is easier to use. Instead of just the simple sequences of bits in the physical machine, a high level language provides complex structured data which easily lends itself to describe the structure of the problems that are to be solved.

Numeric types

- The integer data types, or "whole numbers". May be subtyped according to their ability to contain negative values (e.g. unsigned in C and C++). May also have a small number of predefined subtypes (such as short and long in C/C++); or allow users to freely define subranges such as 1..12 (e.g. Pascal/Ada).
- Floating point data types, sometimes misleadingly called reals, contain fractional values. They usually have predefined limits on both their maximum values and their precision. These are often represented as decimal numbers.
- Fixed point data types are convenient for representing monetary values. They are often implemented internally as integers, leading to predefined limits.
- Bignum or arbitrary precision numeric types lack predefined limits. They are not primitive types, and are used sparingly for efficiency reasons.

Composite / Derived data types

- An array stores a number of elements of the same type in a specific order. They are accessed using an integer to specify which element is required (although the elements may be of almost any type). Arrays may be fixed length or expandable.
- Record (also called tuple or struct) Records are among the simplest data structures. A record is a value that contains other values, typically in fixed number and sequence and typically indexed by names. The elements of records are usually called fields or members.
- Union. A union type definition will specify which of a number of permitted primitive types

may be stored in its instances, e.g. "float or long integer". Contrast with a record, which could be defined to contain a float and an integer; whereas, in a union, there is only one value at a time.

- A tagged union (also called a variant, variant record, discriminated union, or disjoint union) contains an additional field indicating its current type, for enhanced type safety.
- A set is an abstract data structure that can store certain values, without any particular order, and no repeated values. Values themselves are not retrieved from sets, rather one tests a value for membership to obtain a boolean "in" or "not in".
- An object contains a number of data fields, like a record, and also a number of program code fragments for accessing or modifying them. Data structures not containing code, like those above, are called plain old data structure.

Enumerated Type: This has values which are different from each other, and which can be compared and assigned, but which do not necessarily have any particular concrete representation in the computer's memory; compilers and interpreters can represent them arbitrarily. For example, the four suits in a deck of playing cards may be four enumerators named CLUB, DIAMOND, HEART, SPADE, belonging to an enumerated type named suit.

String and text types

- Alphanumeric character. A letter of the alphabet, digit, blank space, punctuation mark, etc.
- Alphanumeric strings, a sequence of characters. They are typically used to represent words and text.

Character and string

Character and string types can store sequences of characters from a character set such as ASCII.

Since most character sets include the digits, it is possible to have a numeric string, such as "1234"

Abstract data types

Any type that does not specify an implementation is an abstract data type.

an abstract data type (ADT) is a mathematical model for a certain class of data structures that have similar behavior; or for certain data types of one or more programming languages that have similar semantics. An abstract data type is defined indirectly, only by the operations that may be performed on it and by mathematical constraints on the effects (and possibly cost) of those operations.

an abstract stack could be defined by three

operations:

- push, that inserts some data item onto the structure,
- pop, that extracts an item from it (with the constraint that each pop always returns the most recently pushed item that has not been popped yet), and
- peek, that allows data on top of the structure to be examined without removal.

Data Structure A data structure is a collection of data type 'values' which are stored and organized in such a way that it allows for efficient access and modification

Data structures

can be grouped into four forms:

- Linear: arrays, lists
- Tree: binary, heaps, space partitioning etc.
- Hash: distributed hash table, hash tree etc.
- Graphs: decision, directed, acyclic etc

Array: An array is a finite group of data, which is allocated contiguous (i.e. sharing a common border) memory locations, and each element within the array is accessed via an index key (typically numerical, and zero based). The name assigned to an array is typically a pointer to the first item in the array.

Linked List

A linked list is different to an array in that the order of the elements within the list are not

determined by a contiguous memory allocation. Instead the elements of the linked list can be sporadically placed in memory due to its design, which is that each element of the list (also referred to as a 'node') consists of two parts:

- the data
- a pointer

Tree The concept of a 'tree' in its simplest terms is to represent a hierarchical tree structure, with a root value and subtrees of children (with a parent node), represented as a set of linked nodes

A tree contains "nodes" (a node has a value associated with it) and each node is connected by a line called an "edge"

The top level node is known as the "root" and a node with no children is a "leaf". If a node is connected to other nodes, then the proceeding node is referred to as the "parent", and nodes following it are "child" nodes.

Binary Tree

A binary tree is a 'rooted tree' and consists of nodes which have, at most, two children. This is as the name suggests (i.e. 'binary': 0 or 1), so two potential values/directions.

There three ways of traversing binary tree which are **preorder traversal, inorder traversal and postorder traversal**

Preorder Traversal (Root, leftnode, rightnode)

Inorder Traversal (Leftnode, root, rightnode)

Postorder Traversal (Leftnode, rightnode, Root)

B-tree

A B-tree is a sorted tree that is very similar in essence to a red-black tree in that it is self-balancing and as such can guarantee logarithmic time for search/insert/delete operations. A B-tree is useful for large read/writes of data and is commonly used in the design of databases and file systems, but it's important to note that a B-tree is not a binary search tree because it allows more than two child nodes.

Weight-balanced Tree

A weight-balanced tree is a form of binary search tree and is similar in spirit to a weighted graph, in that individual nodes are 'weighted' to indicate the more likely successful route with regards to searching for a particular value.

Binary Heap

A binary heap tree is a binary tree, not a binary search tree, and so it's not a sorted tree

- heap property: the node value is either greater (or lesser depending on the direction of the heap) or equal to the value of its parent.
- shape property: if the last level of the tree is incomplete, the missing nodes are filled.

Hash Table

A hash table is a data structure which is capable of mapping 'keys' to 'values', and you'll typically find this is abstracted and enhanced with additional behaviours by many high-level programming languages such that they behave like an 'associative array' abstract data type. In Python it's called a 'dictionary'

two solutions for avoiding hasvhing collisions

- Separate Chaining
- Linear Probing

Stack

A stack is a basic data structure that can be logically thought as linear structure represented by a real physical stack or pile, a structure where insertion and deletion of items takes place at one end called top of the stack.

There are basically three operations that can be performed on stacks. They are:

- inserting (—pushing||) an item into a stack
- deleting (—popping||) an item from the stack
- displaying the contents of the top item of the stack (—peeking||)

Queue

A queue is an abstract data type or a linear data structure, in which the first element is inserted from one end (the —tail||), and the deletion of existing element takes place from the other end (the—head||). A queue is a —First In First Out|| structure. The process of adding an element to a queue is called —enqueueing|| and the process of removing an element from a queue is called —dequeueing||.

Infix Notation

In this notation, a binary operation symbol is placed between the expressions representing its two

Operands. Infix notation is the one most commonly used in mathematics, and, as a consequence is the one used by most programming languages

Prefix Notation

Prefix notation is another type of notation. It is also called prefix Polish notation.¹ The symbol which represents the operation precedes the symbols representing the operands (written from left to right, in the same way as text).

Side effects: A side effect is an action that influences the result (partial or final) of a computation without otherwise explicitly returning a value in the context in which it is found.

Finite arithmetic: