**Introduction to C**

C is a general-purpose, **procedural**, and**middle-level**programming language used for developing **computer software**, **system programming**, **applications**, **games**, and more. Known for its simplicity and efficiency, C is an excellent choice for beginners as it provides a strong foundation in programming concepts. C was developed by Dennis M. Ritchie at Bell Laboratories in 1972. Initially, it was created for programming the **UNIX operating system**.

**Types-of-programming-languages**

* Machine languages
* Assembly languages
* Procedural programming languages
* Functional programming languages
* Object-oriented programming languages
* Scripting languages
* Logic programming languages

**What Is a Programming Language?**

A programming language is a system of notation that instructs a computer on how to execute specific tasks. Made up of symbols, punctuation and keywords, they use syntax to dictate how code is written and structured. Over time, programming languages have moved away from machine-level instructions and have incorporated more natural language and human-readable elements, creating a more intuitive development process.

**Types of Programming Languages**

All of the programming languages known today can be funneled down into seven distinct categories:

**1. Machine Languages**

Machine language, or machine code, is the lowest-level programming language. Because it is written in [binary code](https://builtin.com/software-engineering-perspectives/binary), it’s the only language a computer can directly execute without the need for translation. It’s written in numeric strings of zeros and ones, where zeros represent an “off” state absent of an electrical signal while one indicates an electrified “on” state. These inputs directly manipulate a computer’s operations at the [hardware](https://builtin.com/hardware) level, including data movement, arithmetic and logic. Machine language is near impossible for a human to read and write, and is tailored to the architecture of each computer.

* **Examples of instruction set architectures for machine language:**ARM, x86, RISC-V

**2. Assembly Languages**

Assembly programming language is a low-level language that is closely related to machine code, providing a way to write instructions that a computer can directly execute. It’s essentially a human-readable version of machine language that’s made up of built-in or user-defined mnemonic codes. Because [assembly](https://builtin.com/articles/how-write-hello-world-asm) languages specifically correspond to a particular computer architecture, they offer more control and speed compared to high-level languages but require detailed knowledge of the underlying hardware in order to write them.

* **Examples:**ARM A64 assembly language, x86 assembly language, MIPS assembly language

**3. Procedural Programming Languages**

A procedural programming language follows a series of functions to perform actions or solve a specific problem. In this low-level style, developers write procedures — or reusable blocks of code — that contain step-by-step instructions that [manipulate data](https://builtin.com/data-science/data-management) in a systematic way. It is commonly used in medium-sized applications and embedded systems where straightforward logic and [control flow](https://builtin.com/software-engineering-perspectives/control-flow) are essential.

* **Examples**: C, C++, Java, Pascal, BASIC

**4. Functional Programming Languages**

A [functional programming](https://builtin.com/software-engineering-perspectives/functional-programming) language uses functions to perform tasks, treating everything like a math problem without changing values or states. It encourages writing predictable, maintainable code that is easy to understand by using pure functions and immutable data. It is often used for projects that need to handle [massive volumes of data](https://builtin.com/big-data) or run multiple tasks at once, as seen in [data science](https://builtin.com/data-science), [financial modeling](https://builtin.com/articles/quantitative-modeling) and real-time [distributed systems](https://builtin.com/articles/distributed-computing).

* **Examples**: [Scala](https://builtin.com/software-engineering-perspectives/scala), Erlang, Haskell, Elixir, F#

**5. Object-Oriented Programming Languages**

An object-oriented programming (OOP) language turns everything into objects. Each object — which is a unique piece of data — is defined by its attributes and methods, and assigned to a particular class. Rather than designing a language around functions and logic, object-oriented programming allows developers to directly manipulate data. These languages use principles like inheritance, encapsulation, polymorphism and abstraction to model real-world entities and relationships, making code more modular, reusable and easier to maintain. It’s frequently used in software, web and [game development](https://builtin.com/gaming) as well as [simulation](https://builtin.com/hardware/computer-simulation) modeling and [artificial intelligence](https://builtin.com/artificial-intelligence) applications.

* **Examples:** Java, Python, PHP, C++, Ruby

**6. Scripting Languages**

A scripting language is designed to automate repetitive tasks, manage dynamic content and support processes in larger applications. Unlike traditional programming languages, scripting languages are often interpreted rather than compiled, making them ideal for writing quick, small scripts for specific tasks. They have a simplified syntax, which makes them easy to learn. Due to their ease of use and flexibility, scripting languages are commonly featured in web development, system administration and software automation.

* **Examples**: PHP, Ruby, Python, bash, [Perl](https://builtin.com/software-engineering-perspectives/perl), [Node.js](https://builtin.com/software-engineering-perspectives/nodejs)

**7. Logic Programming Languages**

A logic programming language creates a paradigm that is based on formal logic. By presenting a system of facts and rules, it lets computers solve computations on their own without the need for step-by-step instructions. In these languages, developers define rules and relationships typically in the form of clauses. The program then performs computations based on those rules using logical inference. This allows programmers to define problems using logical relationships and constraints rather than direct commands. Logic programming languages are often used in fields like artificial intelligence and computational linguistics for tasks that involve pattern matching and knowledge representation.

* **Examples**: [Prolog](https://builtin.com/software-engineering-perspectives/prolog" \t "_blank), Absys, Datalog, Alma-0