C is a general-purpose, procedural computer programming language supporting structured programming, lexical variable scope, and recursion, with a static type system. By design, C provides constructs that map efficiently to typical machine instructions. It has found lasting use in applications previously coded in assembly language. Such applications include operating systems and various application software for computer architectures that range from supercomputers to PLCs and embedded systems.

A successor to the programming language B, C was originally developed at Bell Labs by Dennis Ritchie between 1972 and 1973 to construct utilities running on Unix. It was applied to re-implementing the kernel of the Unix operating system.[6] During the 1980s, C gradually gained popularity. It has become one of the most widely used programming languages,[7][8] with C compilers from various vendors available for the majority of existing computer architectures and operating systems. C has been standardized by the ANSI since 1989 (ANSI C) and by the International Organization for Standardization (ISO). As of September 2020, C is the most popular programming language.[9]

C is an imperative procedural language. It was designed to be compiled to provide low-level access to memory and language constructs that map efficiently to machine instructions, all with minimal runtime support. Despite its low-level capabilities, the language was designed to encourage cross-platform programming. A standards-compliant C program written with portability in mind can be compiled for a wide variety of computer platforms and operating systems with few changes to its source code.

The origin of C is closely tied to the development of the Unix operating system, originally implemented in assembly language on a PDP-7 by Dennis Ritchie and Ken Thompson, incorporating several ideas from colleagues. Eventually, they decided to port the operating system to a PDP-11. The original PDP-11 version of Unix was also developed in assembly language.[6]

Thompson desired a programming language to make utilities for the new platform. At first, he tried to make a Fortran compiler, but soon gave up the idea. Instead, he created a cut-down version of the recently developed BCPL systems programming language. The official description of BCPL was not available at the time,[11] and Thompson modified the syntax to be less wordy, producing the similar but somewhat simpler B.[6] However, few utilities were ultimately written in B because it was too slow, and B could not take advantage of PDP-11 features such as byte addressability.

In 1972, Ritchie started to improve B, which resulted in creating a new language C.[12] The C compiler and some utilities made with it were included in Version 2 Unix.[13]

At Version 4 Unix, released in November 1973, the Unix kernel was extensively re-implemented in C.[6] By this time, the C language had acquired some powerful features such as struct types.

Preprocessor was introduced around 1973 at the urging of Alan Snyder and also in recognition of the usefulness of the file-inclusion mechanisms available in BCPL and PL/I. Its original version provided only included files and simple string replacements: #include and #define of parameterless macros. Soon after that, it was extended, mostly by Mike Lesk and then by John Reiser, to incorporate macros with arguments and conditional compilation.[14]

Unix was one of the first operating system kernels implemented in a language other than assembly. Earlier instances include the Multics system (which was written in PL/I) and Master Control Program (MCP) for the Burroughs B5000 (which was written in ALGOL) in 1961. In around 1977, Ritchie and Stephen C. Johnson made further changes to the language to facilitate portability of the Unix operating system. Johnson's Portable C Compiler served as the basis for several implementations of C on new platforms.

During the late 1970s and 1980s, versions of C were implemented for a wide variety of mainframe computers, minicomputers, and microcomputers, including the IBM PC, as its popularity began to increase significantly.

In 1983, the American National Standards Institute (ANSI) formed a committee, X3J11, to establish a standard specification of C. X3J11 based the C standard on the Unix implementation; however, the non-portable portion of the Unix C library was handed off to the IEEE working group 1003 to become the basis for the 1988 POSIX standard. In 1989, the C standard was ratified as ANSI X3.159-1989 "Programming Language C". This version of the language is often referred to as ANSI C, Standard C, or sometimes C89.

In 1990, the ANSI C standard (with formatting changes) was adopted by the International Organization for Standardization (ISO) as ISO/IEC 9899:1990, which is sometimes called C90. Therefore, the terms "C89" and "C90" refer to the same programming language.

ANSI, like other national standards bodies, no longer develops the C standard independently, but defers to the international C standard, maintained by the working group ISO/IEC JTC1/SC22/WG14. National adoption of an update to the international standard typically occurs within a year of ISO publication.

One of the aims of the C standardization process was to produce a superset of K&R C, incorporating many of the subsequently introduced unofficial features. The standards committee also included several additional features such as function prototypes (borrowed from C++), void pointers, support for international character sets and locales, and preprocessor enhancements. Although the syntax for parameter declarations was augmented to include the style used in C++, the K&R interface continued to be permitted, for compatibility with existing source code.

C89 is supported by current C compilers, and most modern C code is based on it. Any program written only in Standard C and without any hardware-dependent assumptions will run correctly on any platform with a conforming C implementation, within its resource limits. Without such precautions, programs may compile only on a certain platform or with a particular compiler, due, for example, to the use of non-standard libraries, such as GUI libraries, or to a reliance on compiler- or platform-specific attributes such as the exact size of data types and byte endianness.

In cases where code must be compilable by either standard-conforming or K&R C-based compilers, the \_\_STDC\_\_ macro can be used to split the code into Standard and K&R sections to prevent the use on a K&R C-based compiler of features available only in Standard C.

After the ANSI/ISO standardization process, the C language specification remained relatively static for several years. In 1995, Normative Amendment 1 to the 1990 C standard (ISO/IEC 9899/AMD1:1995, known informally as C95) was published, to correct some details and to add more extensive support for international character sets.

Historically, embedded C programming requires nonstandard extensions to the C language in order to support exotic features such as fixed-point arithmetic, multiple distinct memory banks, and basic I/O operations.

In 2008, the C Standards Committee published a technical report extending the C language[20] to address these issues by providing a common standard for all implementations to adhere to. It includes a number of features not available in normal C, such as fixed-point arithmetic, named address spaces, and basic I/O hardware addressing.

C supports a rich set of operators, which are symbols used within an expression to specify the manipulations to be performed while evaluating that expression. C has operators for:

arithmetic: +, -, \*, /, %

assignment: =

augmented assignment: +=, -=, \*=, /=, %=, &=, |=, ^=, <<=, >>=

bitwise logic: ~, &, |, ^

bitwise shifts: <<, >>

boolean logic: !, &&, ||

conditional evaluation: ? :

equality testing: ==, !=

calling functions: ( )

increment and decrement: ++, --

member selection: ., ->

object size: sizeof

order relations: <, <=, >, >=

reference and dereference: &, \*, [ ]

sequencing: ,

subexpression grouping: ( )

type conversion: (typename)

C uses the operator = (used in mathematics to express equality) to indicate assignment, following the precedent of Fortran and PL/I, but unlike ALGOL and its derivatives. C uses the operator == to test for equality. The similarity between these two operators (assignment and equality) may result in the accidental use of one in place of the other, and in many cases, the mistake does not produce an error message (although some compilers produce warnings). For example, the conditional expression if (a == b + 1) might mistakenly be written as if (a = b + 1), which will be evaluated as true if a is not zero after the assignment.[26]

The C operator precedence is not always intuitive. For example, the operator == binds more tightly than (is executed prior to) the operators & (bitwise AND) and | (bitwise OR) in expressions such as x & 1 == 0, which must be written as (x & 1) == 0 if that is the coder's intent.

C is widely used for systems programming in implementing operating systems and embedded system applications,[39] because C code, when written for portability, can be used for most purposes, yet when needed, system-specific code can be used to access specific hardware addresses and to perform type punning to match externally imposed interface requirements, with a low run-time demand on system resources.

C can be used for website programming using the Common Gateway Interface (CGI) as a "gateway" for information between the Web application, the server, and the browser.[40] C is often chosen over interpreted languages because of its speed, stability, and near-universal availability.[41]

A consequence of C's wide availability and efficiency is that compilers, libraries and interpreters of other programming languages are often implemented in C. For example, the reference implementations of Python, Perl, and PHP are written in C.

C enables programmers to create efficient implementations of algorithms and data structures, because the layer of abstraction from hardware is thin, and its overhead is low, an important criterion for computationally intensive programs. For example, the GNU Multiple Precision Arithmetic Library, the GNU Scientific Library, Mathematica, and MATLAB are completely or partially written in C.

C is sometimes used as an intermediate language by implementations of other languages. This approach may be used for portability or convenience; by using C as an intermediate language, additional machine-specific code generators are not necessary. C has some features, such as line-number preprocessor directives and optional superfluous commas at the end of initializer lists, that support compilation of generated code. However, some of C's shortcomings have prompted the development of other C-based languages specifically designed for use as intermediate languages, such as C--.

C has also been widely used to implement end-user applications. However, such applications can also be written in newer, higher-level languages.