SUBJECT-C-PROGRAMMING [UNIT-I]

C (programming language)

C (*pronounced* /ˈsiː/ – *like the letter* c)[6] is a general-purpose programming language. It was created in the 1970s by Dennis Ritchie and remains very widely used and influential. By design, C's features cleanly reflect the capabilities of the targeted CPUs. It has found lasting use in operating systems code (especially in kernels[7]), device drivers, and protocol stacks, but its use in application software has been decreasing.[8] C is commonly used on computer architectures that range from the largest supercomputers to the smallest microcontrollers and embedded systems.

A successor to the programming language B, C was originally developed at Bell Labs by Ritchie between 1972 and 1973 to construct utilities running on Unix. It was applied to re-implementing the kernel of the Unix operating system.[9] During the 1980s, C gradually gained popularity. It has become one of the most widely used programming languages,[10][11] with C compilers available for practically all modern computer architectures and operating systems. The book *The C Programming Language*, co-authored by the original language designer, served for many years as the *de facto* standard for the language.[12][1] C has been standardized since 1989 by the American National Standards Institute (ANSI) and, subsequently, jointly by the International Organization for Standardization (ISO) and the International Electrotechnical Commission (IEC).

C is an imperative procedural language, supporting structured programming, lexical variable scope, and recursion, with a static type system. 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.

Since 2000, C has consistently ranked among the top four languages in the TIOBE index, a measure of the popularity of programming languages.[13]

Overview

Dennis Ritchie (right), the inventor of the C programming language, with Ken Thompson

C is an imperative, procedural language in the ALGOL tradition. It has a static type system. In C, all executable code is contained within subroutines (also called "functions", though not in the sense of functional programming). Function parameters are passed by value, although arrays are passed as pointers, i.e. the address of the first item in the array. *Pass-by-reference* is simulated in C by explicitly passing pointers to the thing being referenced.

C program source text is free-form code. Semicolons terminate statements, while curly braces are used to group statements into blocks.

The C language also exhibits the following characteristics:

- The language has a small, fixed number of keywords, including a full set of control flow primitives: `if/else`, `for`, `do/while`, `while`, and `switch`. User-defined names are not distinguished from keywords by any kind of sigil.
- It has a large number of arithmetic, bitwise, and logic operators: `+`,`+=`,`++`,`&`,`||`, etc.
- More than one assignment may be performed in a single statement.
- Functions:
  - Function return values can be ignored, when not needed.
  - Function and data pointers permit *ad hoc* run-time polymorphism.
  - Functions may not be defined within the lexical scope of other functions.
  - Variables may be defined within a function, with scope.
  - A function may call itself, so recursion is supported.
- Data typing is static, but weakly enforced; all data has a type, but implicit conversions are possible.
- User-defined (typedef) and compound types are possible.
  - Heterogeneous aggregate data types (`struct`) allow related data elements to be accessed and assigned as a unit. The contents of whole structs cannot be compared using a single built-in operator (the elements must be compared individually).
  - Union is a structure with overlapping members; it allows multiple data types to share the same memory location.
  - Array indexing is a secondary notation, defined in terms of pointer arithmetic. Whole arrays cannot be assigned or compared using a single built-in operator. There is no "array" keyword in use or definition; instead, square brackets indicate arrays syntactically, for example `month[11]`.

    - Enumerated types are possible with the `enum` keyword. They are freely interconvertible with integers.
    - Strings are not a distinct data type, but are conventionally implemented as null-terminated character arrays.
- Low-level access to computer memory is possible by converting machine addresses to pointers.
- Procedures (subroutines not returning values) are a special case of function, with an empty return type `void`.
- Memory can be allocated to a program with calls to library routines.
- A preprocessor performs macro definition, source code file inclusion, and conditional compilation.
- There is a basic form of modularity: files can be compiled separately and linked together, with control over which functions and data objects are visible to other files via `static` and `extern` attributes.
- Complex functionality such as I/O, string manipulation, and mathematical functions are consistently delegated to library routines.
- The generated code after compilation has relatively straightforward needs on the underlying platform, which makes it suitable for creating operating systems and for use in embedded systems.

While C does not include certain features found in other languages (such as object orientation and garbage collection), these can be implemented or emulated, often through the use of external libraries (e.g., the GLib Object System or the Boehm garbage collector).

Relations to other languages

: List of C-family programming languages**Many later languages have** borrowed directly or indirectly from C, including C++, C#, Unix's C shell, D, Go, Java, JavaScript (including transpilers), Julia, Limbo, LPC, Objective-C, Perl, PHP, Python, Ruby, Rust, Swift, Verilog and SystemVerilog (hardware description languages).[5] These languages have drawn many of their control structures and other basic features from C. Most of them also express highly similar syntax to C, and they tend to combine the recognizable expression and statement syntax of C with underlying type systems, data models, and semantics that can be radically different.

History

]

Early developments

Timeline of C language

Year	Informal name	Official standard
1972	first release	—

1978	K&R C	—
1989, 1990	ANSI C, C89, ISO C, C90	ANSI X3.159-1989 ISO/IEC 9899:1990
1999	C99, C9X	ISO/IEC 9899:1999
2011	C11, C1X	ISO/IEC 9899:2011
2018	C17	ISO/IEC 9899:2018
2024	C23, C2X	ISO/IEC 9899:2024
Future	C2Y	—

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.[9]

B

Thompson wanted a programming language for developing utilities for the new platform. At first he tried to write a Fortran compiler, but he soon gave up the idea. Instead, he created a cut-down version of the recently developed systems programming language called BCPL. The official description of BCPL was not available at the time,[14] and Thompson modified the syntax to be less wordy and similar to a simplified ALGOL known as SMALGOL.[15] Thompson called the result *B*.[9] He described B as "BCPL semantics with a lot of SMALGOL syntax".[15] Like BCPL, B had a bootstrapping compiler to facilitate porting to new machines.[15] However, few utilities were ultimately written in B because it was too slow and could not take advantage of PDP-11 features such as byte addressability.

New B and first C release

In 1971 Ritchie started to improve B, to use the features of the more-powerful PDP-11. A significant addition was a character data type. He called this *New B* (NB).[15] Thompson started to use NB to write the Unix kernel, and his requirements shaped the direction of the language development.[15][16] Through to 1972, richer types were added to the NB language: NB had arrays of `int` and `char`. Pointers, the ability to generate pointers to other types, arrays of all types, and types to be returned from functions were all also added. Arrays within expressions became pointers. A new compiler was written, and the language was renamed C.[9]

The C compiler and some utilities made with it were included in Version 2 Unix, which is also known as Research Unix.[17]

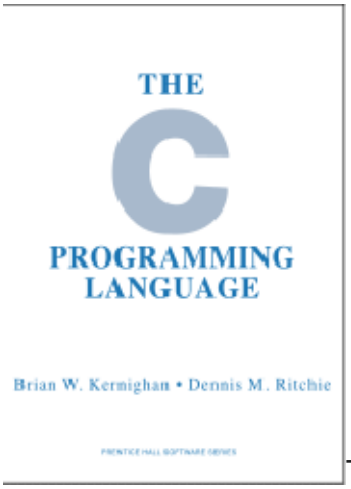
Structures and Unix kernel re-write

At Version 4 Unix, released in November 1973, the Unix kernel was extensively re-implemented in C.[9] By this time, the C language had acquired some powerful features such as `struct` types.

The 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.[9]

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.[16]

K&R C



The cover of the book *The C Programming Language*, first edition, by Brian Kernighan and Dennis Ritchie

In 1978 Brian Kernighan and Dennis Ritchie published the first edition of *The C Programming Language*.[18] Known as *K&R* from the initials of its authors, the book served for many years as an informal specification of the language. The version of C that it describes is commonly referred to as "**K&R C**". As this was released in 1978, it is now also referred to as *C78*.[19] The second edition of the book[20] covers the later ANSI C standard, described below.

K&R introduced several language features:

- Standard I/O library
- `long int` data type
- `unsigned int` data type

- Compound assignment operators of the form `=op` (such as `=-`) were changed to the form `op=` (that is, `-=`) to remove the semantic ambiguity created by constructs such as `i=-10`, which had been interpreted as `i =- 10` (decrement `i` by 10) instead of the possibly intended `i = -10` (let `i` be −10).

Even after the publication of the 1989 ANSI standard, for many years K&R C was still considered the "lowest common denominator" to which C programmers restricted themselves when maximum portability was desired, since many older compilers were still in use, and because carefully written K&R C code can be legal Standard C as well.

In early versions of C, only functions that return types other than `int` must be declared if used before the function definition; functions used without prior declaration were presumed to return type `int`.

For example:

```
long some_function(); /* This is a function declaration, so the compiler can
know the name and return type of this function. */
/* int */ other_function(); /* Another function declaration.  Because this is
an early version of C, there is an implicit 'int' type here.  A comment shows
where the explicit 'int' type specifier would be required in later versions.
*/

/* int */ calling_function() /* This is a function definition, including the
body of the code following in the { curly brackets }. Because no return type
is specified, the function implicitly returns an 'int' in this early version
of C. */
{
    long test1;
    register /* int */ test2; /* Again, note that 'int' is not required here.
The 'int' type specifier */
                               /* in the comment would be required in later
versions of C. */
                               /* The 'register' keyword indicates to the
compiler that this variable should */
                               /* ideally be stored in a register as opposed
to within the stack frame. */
    test1 = some_function();
    if (test1 > 1)
          test2 = 0;
    else
          test2 = other_function();
    return test2;
}
```

The `int` type specifiers which are commented out could be omitted in K&R C, but are required in later standards.

Since K&R function declarations did not include any information about function arguments, function parameter type checks were not performed, although some compilers would issue a warning message if a local function was called with the wrong number of arguments, or if different calls to an external function used different numbers or types of arguments. Separate tools such as Unix's lint utility were developed that

(among other things) could check for consistency of function use across multiple source files.

In the years following the publication of K&R C, several features were added to the language, supported by compilers from AT&T (in particular PCC[21]) and some other vendors. These included:

- `void` functions (i.e., functions with no return value)
- functions returning `struct` or `union` types (previously only a single pointer, integer or float could be returned)
- assignment for `struct` data types
- enumerated types (previously, preprocessor definitions for integer fixed values were used, e.g. `#define GREEN 3`)

The large number of extensions and lack of agreement on a standard library, together with the language popularity and the fact that not even the Unix compilers precisely implemented the K&R specification, led to the necessity of standardization.[22]

ANSI C and ISO C

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.[23]

C99

The C standard was further revised in the late 1990s, leading to the publication of ISO/IEC 9899:1999 in 1999, which is commonly referred to as "C99". It has since been amended three times by Technical Corrigenda.[24]

C99 introduced several new features, including inline functions, several new data types (including `long long int` and a `complex` type to represent complex numbers), variable-length arrays and flexible array members, improved support for IEEE 754 floating point, support for variadic macros (macros of variable arity), and support for one-line comments beginning with `//`, as in BCPL or C++. Many of these had already been implemented as extensions in several C compilers.

C99 is for the most part backward compatible with C90, but is stricter in some ways; in particular, a declaration that lacks a type specifier no longer has `int` implicitly assumed. A standard macro `__STDC_VERSION__` is defined with value `199901L` to indicate that C99 support is available. GCC, Solaris Studio, and other C compilers nowsupport many or all of the new features of C99. The C compiler in Microsoft Visual C++, however, implements the C89 standard and those parts of C99 that are required for compatibility with C++11. In addition, the C99 standard requires support for identifiers using Unicode in the form of escaped characters (e.g. `\u0040` or `\U0001f431`) and suggests support for raw Unicode names.

C11

In 2007 work began on another revision of the C standard, informally called "C1X" until its official publication of ISO/IEC 9899:2011 on 2011-12-08. The C standards committee adopted guidelines to limit the adoption of new features that had not been tested by existing implementations.

The C11 standard adds numerous new features to C and the library, including type generic macros, anonymous structures, improved Unicode support, atomic operations, multi-threading, and bounds-checked functions. It also makes some portions of the existing C99 library optional, and improves compatibility with C++. The standard macro `__STDC_VERSION__` is defined as `201112L` to indicate that C11 support is available.

C17

Published in June 2018 as ISO/IEC 9899:2018, C17 is the current standard for the C programming language. It introduces no new language features, only technical corrections, and clarifications to defects in C11. The standard macro `__STDC_VERSION__` is defined as `201710L` to indicate that C17 support is available.

C23

C23 is the informal name for the next (after C17) major C language standard revision. It was informally known as "C2X" through most of its development. C23 is expected to be published in early 2024 as ISO/IEC 9899:2024.[26] The standard macro `__STDC_VERSION__` is defined as `202311L` to indicate that C23 support is available.

C2Y

C2Y is a temporary informal name for the next major C language standard revision, after C23 (C2X), that is hoped to be released later in the 2020s decade, hence the '2' in "C2Y". An early working draft of C2Y was released in February 2024 as N3220 by the working group ISO/IEC JTC1/SC22/WG14.[27]

Embedded C

Historically, embedded C programming requires nonstandard extensions to the C language 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[28] 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.

Syntax

C has a formal grammar specified by the C standard.[29] Line endings are generally not significant in C; however, line boundaries do have significance during the preprocessing phase. Comments may appear either between the delimiters `/*` and `*/`, or (since C99)

following `//` until the end of the line. Comments delimited by `/*` and `*/` do not nest, and these sequences of characters are not interpreted as comment delimiters if they appear inside string or character literals.[30]

C source files contain declarations and function definitions. Function definitions, in turn, contain declarations and statements. Declarations either define new types using keywords such as `struct`, `union`, and `enum`, or assign types to and perhaps reserve storage for new variables, usually by writing the type followed by the variable name. Keywords such as `char` and `int` specify built-in types. Sections of code are enclosed in braces (`{` and `}`, sometimes called "curly brackets") to limit the scope of declarations and to act as a single statement for control structures.

As an imperative language, C uses *statements* to specify actions. The most common statement is an *expression statement*, consisting of an expression to be evaluated, followed by a semicolon; as a side effect of the evaluation, functions may be called and variables may be assigned new values. To modify the normal sequential execution of statements, C provides several control-flow statements identified by reserved keywords. Structured programming is supported by `if` ... [`else`] conditional execution and by `do` ... `while`, `while`, and `for` iterative execution (looping). The `for` statement has separate initialization, testing, and reinitialization expressions, any or all of which can be omitted. `break` and `continue` can be used within the loop. Break is used to leave the innermost enclosing loop statement and continue is used to skip to its reinitialisation. There is also a non-structured `goto` statement which branches directly to the designated label within the function. `switch` selects a `case` to be executed based on the value of an integer expression. Different from many other languages, control-flow will fall through to the next `case` unless terminated by a `break`.

Expressions can use a variety of built-in operators and may contain function calls. The order in which arguments to functions and operands to most operators are evaluated is unspecified. The evaluations may even be interleaved. However, all side effects (including storage to variables) will occur before the next "sequence point"; sequence points include the end of each expression statement, and the entry to and return from each function call. Sequence points also occur during evaluation of expressions containing certain operators (`&&`, `||`, `?:` and the comma operator). This permits a high degree of object code optimization by the compiler, but requires C programmers to take more care to obtain reliable results than is needed for other programming languages.

Kernighan and Ritchie say in the Introduction of *The C Programming Language*: "C, like any other language, has its blemishes. Some of the operators have the wrong precedence; some parts of the syntax could be better."[31] The C standard did not attempt to correct many of these blemishes, because of the impact of such changes on already existing software.

Character set

The basic C source character set includes the following characters:

- Lowercase and uppercase letters of ISO Basic Latin Alphabet: `a–z A–Z`
- Decimal digits: `0–9`
- Graphic characters: `! " # % & ' ( ) * + , - . / : ; < = > ? [ \ ] ^ _ { | } ~`
- Whitespace characters: *space*, *horizontal tab*, *vertical tab*, *form feed*, *newline*

Newline indicates the end of a text line; it need not correspond to an actual single character, although for convenience C treats it as one.

Additional multi-byte encoded characters may be used in string literals, but they are not entirely portable. The latest C standard (C11) allows multi-national Unicode characters to be embedded portably within C source text by using `\uXXXX` or `\UXXXXXXXX` encoding (where the `X` denotes a hexadecimal character), although this feature is not yet widely implemented.

The basic C execution character set contains the same characters, along with representations for alert, backspace, and carriage return. Run-time support for extended character sets has increased with each revision of the C standard.

Reserved words

The following reserved words are case sensitive.

C89 has 32 reserved words, also known as keywords, which are the words that cannot be used for any purposes other than those for which they are predefined:

- `auto`
- `break`
- `case`
- `char`
- `const`
- `continue`
- `default`
- `do`
- `double`
- `else`
- `enum`
- `extern`
- `float`
- `for`
- `goto`
- `if`
- `int`
- `long`
- `register`
- `return`
- `short`

- signed
- sizeof
- static
- struct
- switch
- typedef
- union
- unsigned
- void
- volatile
- while

C99 reserved five more words: (‡ is an alternative spelling alias for a C23 keywo