RDES

E-Library
C – Programming

HISTORY:

Early developments

Timeline of C language

Year	Informal name	C Standard
1972	Birth	—
1978	K&R C	—
1989/1990	ANSI C, ISO C	ISO/IEC 9899:1990
1999	C99	ISO/IEC 9899:1999
2011	C11, C1x	ISO/IEC 9899:2011
2018	C17	ISO/IEC 9899:2018
2023*	C23, C2x	TBA

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.

B

Thompson wanted a programming language for developing utilities for the new platform. At first, he tried to write 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 and Thompson modified the syntax to be less wordy, and similar to a simplified ALGOL known as SMALGOL. Thompson called the result *B*. He described B as "BCPL semantics with a lot of SMALGOL syntax". Like BCPL, B had a bootstrapping compiler to facilitate porting to new machines. 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 utilise the features of the more-powerful PDP-11. A significant addition was a character data type. He called this *New B*. Thompson started to use NB to write the Unix kernel, and his requirements shaped the direction of the language development. 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.

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

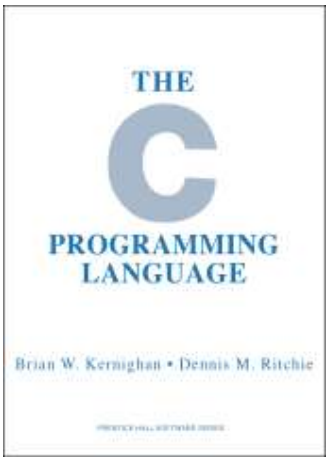
Structures and the Unix kernel re-write

At Version 4 Unix, released in November 1973, the was extensively re-implemented in C. 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 less and then by John Reiser, to incorporate macros with arguments and conditional compilation.

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.

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. This book, known to C programmers as *K&R*, 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 also referred to as *C78*. The second edition of the book covers the later ANSI C standard, described below.

K&R introduced several language features:

- Standard I/0 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. There is an implicit 'int' type here since we're talking about early version of C.  It's commented out here to show where it could go in later variants. */

/* int */ calling_function() /* this is a function definition, including the body of the code following in the { curly brackets } the return type is 'int', but this is implicit so no need to state 'int' when using this early version of C */

{
    long test1;
    register /* int */ test2; /* again, note the 'int' is not required here, and shown as */
```

```
                    /* a comment just to illustrate where it
would be required in later variants 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 multiple 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) 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.

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.

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.

C99 introduced several new features, including inline functions, several new data types (including 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 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 now support many or all of the new features of C99. The C compiler in Microsoft Visual C99, however, implements the C89 standard and those parts of C99 that are required for compatibility with C++11.

In addition, the standard requires support for Unicode identifiers (variable / function names) in the form of escaped characters (e.g. \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.

C2x

C2x is an informal name for the next (after C17) major C language standard revision. It is expected to be voted on in 2023 and would therefore be called C23.

Embedded C

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 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.