

fd**No table of contents entries found.**

# Getting started with C Language

|  |  |  |
| --- | --- | --- |
| **Version** | **Standard** | **Publication Date** |
| K&R | n/a | 1978-02-22 |
| C89 | ANSI X3.159-1989 | 1989-12-14 |
| C90 | ISO/IEC 9899:1990 | 1990-12-20 |
| C95 | ISO/IEC 9899/AMD1:1995 | 1995-03-30 |
| C99 | ISO/IEC 9899:1999 | 1999-12-16 |
| C11 | ISO/IEC 9899:2011 | 2011-12-15 |

## Hello World

To create a simple C program which prints *"Hello, World"* on the screen, use a [text editor](https://en.wikipedia.org/wiki/Text_editor) to create a new ﬁle (e.g. hello.c — the ﬁle extension must be .c) containing the following source code:

hello.c

|  |
| --- |
| #include <stdio.h>  int main(void)  {  puts("Hello, World"); return 0;  } |

[Live demo on Coliru](http://coliru.stacked-crooked.com/a/263e35298419ef1d)

|  |
| --- |
| #include <stdio.h> |

### Let's look at this simple program line by line

This line tells the compiler to include the contents of the standard library header ﬁle stdio.h in the program. Headers are usually ﬁles containing function declarations, macros and data types, and you must include the header ﬁle before you use them. This line includes stdio.h so it can call the function puts().

See more about headers.

|  |
| --- |
| int main(void) |

This line starts the deﬁnition of a function. It states the name of the function (main), the type and number of arguments it expects (void, meaning none), and the type of value that this function returns (int). Program execution starts in the main() function.

|  |
| --- |
| {  …  } |

The curly braces are used in pairs to indicate where a block of code begins and ends. They can be used in a lot of ways, but in this case they indicate where the function begins and ends.

|  |
| --- |
| puts("Hello, World"); |

This line calls the puts() function to output text to standard output (the screen, by default), followed by a newline.

The string to be output is included within the parentheses.

"Hello, World" is the string that will be written to the screen. In C, every string literal value must be inside the double quotes "…".

See more about strings.

In C programs, every statement needs to be terminated by a semi-colon (i.e. ;).

|  |
| --- |
| return 0; |

When we deﬁned main(), we declared it as a function returning an int, meaning it needs to return an integer. In this example, we are returning the integer value 0, which is used to indicate that the program exited successfully. After the return 0; statement, the execution process will terminate.

### Editing the program

Simple text editors include [vim](http://www.vim.org/) or [gedit](https://wiki.gnome.org/Apps/Gedit) on Linux, or [Notepad](https://en.wikipedia.org/wiki/Microsoft_Notepad) on Windows. Cross-platform editors also include

[Visual Studio Code](https://code.visualstudio.com/) or [Sublime Text](https://www.sublimetext.com/).

The editor must create plain text ﬁles, not RTF or other any other format.

### Compiling and running the program

To run the program, this source ﬁle (hello.c) ﬁrst needs to be compiled into an executable ﬁle (e.g. hello on Unix/Linux system or hello.exe on Windows). This is done using a compiler for the C language.

See more about compiling

### Compile using GCC

[GCC](https://gcc.gnu.org/) (GNU Compiler Collection) is a widely used C compiler. To use it, open a terminal, use the command line to navigate to the source ﬁle's location and then run:

|  |
| --- |
| gcc hello.c -o hello |

If no errors are found in the the source code (hello.c), the compiler will create a **binary ﬁle**, the name of which is given by the argument to the -o command line option (hello). This is the ﬁnal executable ﬁle.

We can also use the warning options -Wall -Wextra -Werror, that help to identify problems that can cause the program to fail or produce unexpected results. They are not necessary for this simple program but this is way of adding them:

|  |
| --- |
| gcc -Wall -Wextra -Werror -o hello hello.c |

### Using the clang compiler

To compile the program using [clang](https://llvm.org/) you can use:

|  |
| --- |
| clang -Wall -Wextra -Werror -o hello hello. |

By design, the clang command line options are similar to those of GCC.

### Using the Microsoft C compiler from the command line

If using the Microsoft cl.exe compiler on a Windows system which supports [Visual Studio](https://www.visualstudio.com/downloads/) and if all environment variables are set, this C example may be compiled using the following command which will produce an executable hello.exe within the directory the command is executed in (There are warning options such as /W3 for cl, roughly analogous to -Wall etc for GCC or clang).

|  |
| --- |
| cl hello.c |

### Executing the program

Once compiled, the binary ﬁle may then be executed by typing ./hello in the terminal. Upon execution, the compiled program will print Hello, World, followed by a newline, to the command prompt.

## Original "Hello, World!" in K&R C

The following is the original "Hello, World!" program from the book [The C Programming Language](https://en.wikipedia.org/wiki/The_C_Programming_Language) by Brian Kernighan and Dennis Ritchie (Ritchie was the original developer of the C programming language at Bell Labs), referred to as "K&R":

Version = K&R

|  |
| --- |
| #include <stdio.h>  main()  {  printf("hello, world**\n**");  } |

Notice that the C programming language was not standardized at the time of writing the ﬁrst edition of this book (1978), and that this program will probably not compile on most modern compilers unless they are instructed to accept C90 code.

This very ﬁrst example in the K&R book is now considered poor quality, in part because it lacks an explicit return type for main() and in part because it lacks a return statement. The 2nd edition of the book was written for the old C89 standard. In C89, the type of main would default to int, but the K&R example does not return a deﬁned value to the environment. In C99 and later standards, the return type is required, but it is safe to leave out the return statement of main (and only main), because of a special case introduced with C99 5.1.2.2.3 — it is equivalent to returning 0, which indicates success.

The recommended and most portable form of main for hosted systems is int main (void) when the program does not use any command line arguments, or int main(int argc, char \*\*argv) when the program does use the command line arguments.

C90 §5.1.2.2.3 **Program termination**

A return from the initial call to the main function is equivalent to calling the exit function with the value returned by the main function as its argument. If the main function executes a return that speciﬁes no value, the termination status returned to the host environment is undeﬁned.

C90 §6.6.6.4 **The return statement**

If a return statement without an expression is executed, and the value of the function call is used by the caller, the behavior is undeﬁned. Reaching the } that terminates a function is equivalent to executing a return statement without an expression.

C99 §5.1.2.2.3 **Program termination**

If the return type of the main function is a type compatible with int, a return from the initial call to the main function is equivalent to calling the exit function with the value returned by the main function as its argument; reaching the } that terminates the main function returns a value of 0. If the return type is not compatible with int, the termination status returned to the host environment is unspeciﬁed.

# Comments

Comments are used to indicate something to the person reading the code. Comments are treated like a blank by the compiler and do not change anything in the code's actual meaning. There are two syntaxes used for comments in C, the original ***/\* \*/*** and the slightly newer //. Some documentation systems use specially formatted comments to help produce the documentation for code.

### Commenting using the preprocessor

Large chunks of code can also be "commented out" using the preprocessor directives #if 0 and #endif. This is useful when the code contains multi-line comments that otherwise would not nest.

|  |
| --- |
| #if 0 /\* Starts the "comment", anything from here on is removed by preprocessor \*/  ***/\* A large amount of code with multi-line comments \*/***  int foo()  {  ***/\* lots of code \*/***  ...  ***/\* ... some comment describing the if statement ... \*/***  if (someTest) {  ***/\* some more comments \*/***  return 1;  }  return 0;  }  #endif /\* 0 \*/  ***/\* code from here on is "uncommented" (included in compiled executable) \*/*** |

### /\* \*/ delimited comments

A comment starts with a forward slash followed immediately by an asterisk (/\*), and ends as soon as an asterisk immediately followed by a forward slash (\*/) is encountered. Everything in between these character combinations is a comment and is treated as a blank (basically ignored) by the compiler.

|  |
| --- |
| ***/\* this is a comment \*/*** |

The comment above is a single line comment. Comments of this /\* type can span multiple lines, like so:

|  |
| --- |
| ***/\* thi is a multi-line comment \*/*** |

Though it is not strictly necessary, a common style convention with multi-line comments is to put leading spaces and asterisks on the lines subsequent to the ﬁrst, and the /\* and \*/ on new lines, such that they all line up:

|  |
| --- |
| ***/\****   * ***this is a*** * ***multi-line*** * ***comment***   ***\*/*** |

The extra asterisks do not have any functional eﬀect on the comment as none of them have a related forward slash.

These /\* type of comments can be used on their own line, at the end of a code line, or even within lines of code:

|  |
| --- |
| ***/\* this comment is on its own line \*/***  if (x && y) { ***/\*this comment is at the end of a line \*/***  if ((complexCondition1) ***/\* this comment is within a line of code \*/***  && (complexCondition2)) {  ***/\* this comment is within an if, on its own line \*/***  }  } |

Comments cannot be nested. This is because any subsequent /\* will be ignored (as part of the comment) and the ﬁrst \*/ reached will be treated as ending the comment. The comment in the following example *will not work*:

|  |
| --- |
| ***/\* outer comment, means this is ignored => /\* attempted inner comment \*/*** <= ends the comment, not this one => \*/ |

To comment blocks of code that contain comments of this type, that would otherwise be nested, see the Commenting using the preprocessor example below

### // delimited comments

Version ≥ C99

C99 introduced the use of C++-style single-line comments. This type of comment starts with two forward slashes and runs to the end of a line:

|  |
| --- |
| ***// this is a comment*** |

This type of comment does not allow multi-line comments, though it is possible to make a comment block by adding several single line comments one after the other:

|  |
| --- |
| ***// each of these lines are a single-line comment***  ***// note how each must start with***  ***// the double forward-slash*** |

This type of comment may be used on its own line or at the end of a code line. However, because they run *to the end of the line*, they may *not* be used within a code line

|  |
| --- |
| ***// this comment is on its own line***  if (x && y) { ***// this comment is at the end of a line***  ***// this comment is within an if, on its own line***  } |

### Possible pitfall due to trigraphs

Version ≥ C99

While writing // delimited comments, it is possible to make a typographical error that aﬀects their expected operation. If one types: