

Must declare function prototype in C? [duplicate]

[Ask Question](#)

This question already has an answer here:

[Are prototypes required for all functions in C89, C90 or C99?](#) 6 answers

I am kind of new to C (I have prior Java, C#, and some C++ experience). In C, is it necessary to declare a function prototype or can the code compile without it? Is it good programming practice to do so? Or does it just depend on the compiler? (I am running Ubuntu 9.10 and using the GNU C Compiler, or gcc, under the Code::Blocks IDE)

c function function-prototypes

asked Apr 4 '10 at 16:59



Mohit Deshpande

20k 62 172 237

marked as duplicate by [Jonathan Leffler](#) c Feb 27 '15 at 5:56

This question has been asked before and already has an answer. If those answers do not fully address your question, please [ask a new question](#).

Related: [Are prototypes required for all functions in C89, C90 or C99?](#) – [legends2k](#) Feb 3 '14 at 8:25

10 Answers

In ANSI C (meaning C89 or C90), you do not have to declare a function prototype; however, it is a best practice to use them. The only reason the standard allows you to not use them is for backward compatibility with very old code.

If you do not have a prototype, and you call a function, the compiler will infer a prototype from the parameters you pass to the function. If you declare the function later in the same compilation unit, you'll get a compile error if the function's signature is different from what the compiler guessed.

Worse, if the function is in another compilation unit, there's no way to get a compilation error, since without a a prototype there's no way to check. In that case, if the compiler gets it wrong, you could get undefined behavior if the function call pushes different types on the stack than the function expects.

Convention is to always declare a prototype in a header file that has the same name as the source file containing the function.

In C99 or C11, standard C requires a function declaration in scope before you call any function. Many compilers do not enforce this restriction in practice unless you force them to do so.

edited Feb 27 '15 at 5:52



Jonathan Leffler

529k 82 625 972

answered Apr 4 '10 at 17:08



user308405

895 7 10

- 1 You will not necessarily get a "compile error", if the actual function type is different from the inferred one. This is undefined behavior in C, but not a constraint violation. – [AnT](#) Apr 4 '10 at 17:18
- 1 You're right, should have been more specific, what I meant was that most compilers will throw a warning. – [user308405](#) Apr 4 '10 at 17:32
- 1 In C99, you do need a function declaration - though I suppose it does not have to be a prototyped declaration. C90 (C89) was a lot more lax about it. Note that you do need a prototype in scope if the function uses 'variable arguments' - even in C89/C90. – [Jonathan Leffler](#) Apr 4 '10 at 17:42
- 2 @JeremyP: the compilers are still (too?) forgiving by default, largely because of the legacy code bases that still have to be updated. The compilers don't enforce the letter of the standard unless you force their hand (-Werror, for example, and perhaps -pedantic or -Wold-style-definition or -Wold-style-declaration or -Wmissing-prototypes or -Wstrict-prototypes or all (or most) of the above). I usually use -std=c11 and all those except -pedantic, but I make sure that POSIX or related #define's are active. I don't write #define _GNU_SOURCE or #define _BSD_SOURCE very often. – [Jonathan Leffler](#) Jan 9 '14 at 17:45
- 1 @JonathanLeffler: I've seen some systems which use different default calling conventions for non-prototyped functions than are used for prototyped functions that take arguments, and *also* use different link-time naming. If a function has a prototype and isn't marked as using stack-based arguments, attempting to call it without a visible declaration will result in a link-time error. – [supercat](#) May 28 '15 at 16:48

It is never required to declare a *prototype* for a function in C, neither in "old" C (including C89/90) nor in new C (C99). However, there's a significant difference between C89/90 and C99 with regard to function declarations.

In C89/90 it was not necessary to declare a function at all. If the function is not declared at the point of the call, the compiler "guesses" (infers) the declaration implicitly from the types of the arguments passed in the call and assumes that the return type is `int`.

For example

```
int main() {
    int i = foo(5);
    /* No declaration for `foo`, no prototype for `foo`.
       Will work in C89/90. Assumes `int foo(int)` */

    return 0;
}

int foo(int i) {
    return i;
}
```

Join **Stack Overflow** to learn, share knowledge, and build your career.

Email Sign Up

OR SIGN IN WITH

 Google

Facebook



case), but parameter types can still be guessed from the argument types if function is declared without a prototype.

The previous example will not compile in C99, since `foo` is not declared at the point of the call. Yet, you can add a non-prototype declaration

```
int foo(); /* Declares `foo`, but still no prototype */

int main() {
    int i = foo(5);
    /* No prototype for `foo`, although return type is known.
       Will work in C99. Assumes `int foo(int)` */

    return 0;
}
...
```

and end up with valid C99 code.

Nevertheless, it is always a good practice to declare a prototype for the function before you call it.

An additional note: I said above that it is never required to declare a function prototype. In fact, for some functions it is a requirement. In order to properly call a *variadic* function in C (`printf` for example) the function must be declared *with a prototype* before the point of the call. Otherwise, the behavior is undefined. This applies to both C89/90 and C99.

edited Feb 27 '15 at 5:53



Jonathan Leffler

529k 82 625 972

answered Apr 4 '10 at 17:16



AnT

244k 31 379 629

2 +1 for a detailed answer with all the nuances explained well – legends2k Feb 3 '14 at 8:12

"will not compile in C99" -- no compiler errors (only warnings) on gcc and clang with `-std=c99` on my system. ideone also compiles it – jfs Oct 22 '17 at 17:01

it's not a must, if the function is defined before its use.

edited Apr 4 '10 at 17:07

anon

answered Apr 4 '10 at 17:05



Drakosha

9,353 3 30 46

Same in C++, of course. – anon Apr 4 '10 at 17:07

What's the point of defining it *after* its use? – endolith Feb 21 '12 at 20:45

1 Just the logical order of code in the file, or the fuction is defined in other file. – Drakosha Feb 22 '12 at 17:37

It is not required, but it is bad practice not to use prototypes.

With prototypes, the compiler can verify you are calling the function correctly (using the right number and type of parameters).

Without prototypes, it's possible to have this:

```
// file1.c
void doit(double d)
{
    ....
}

int sum(int a, int b, int c)
{
    return a + b + c;
}
```

and this:

```
// file2.c

// In C, this is just a declaration and not a prototype
void doit();
int sum();


int main(int argc, char *argv[])
{
    char idea[] = "use prototypes!";

    // without the prototype, the compiler will pass a char *
    // to a function that expects a double
    doit(idea);

    // and here without a prototype the compiler allows you to
    // call a function that is expecting three argument with just
    // one argument (in the calling function, args b and c will be
    // random junk)
    return sum(argc);
}
```

edited Apr 4 '10 at 17:13

answered Apr 4 '10 at 17:07

 **R Samuel Klatchko**
61.8k 10 106 171

@NeilButterworth: When compiling file2.c the definition of doit (which is in file1.c) is not visible. – sepp2k Apr 4 '10 at 17:09

@NeilButterworth - I'll reformat, but the comments // file1.c and // file2.c are meant to imply these are different files. When compiling file2.c, the compiler will not have seen the definition in file1.c – [R Samuel Klatchko](#) Apr 4 '10 at 17:10

My comment preceded your edit. I'll delete it. – anon Apr 4 '10 at 17:11

In C, if we do not declare a function prototype and use the function definition, there is no problem and the program compiles and generates the output if the return type of the function is "integer". In all other conditions the compiler error appears. The reason is, if we call a function and do not declare a function prototype then the compiler generates a prototype which returns an integer and it searches for the similar function definition. if the function prototype matches then it compiles successfully. If the return type is not integer then the function prototypes do not match and it generates an error. So it is better to declare function prototype in the header files.

edited Jan 9 '14 at 16:43



Benjamin
14.7k 26 112 214

answered May 20 '12 at 7:06



Raviteja
21 1

C allows functions to be called even though they have not previously been declared, but I strongly recommend you to declare a prototype for all functions before using them so that the compiler can save you if you use the wrong arguments.

answered Apr 4 '10 at 17:07



Anders Abel
53.7k 8 113 192

You should put function declarations in the header file (X.h) and the definition in the source file (X.c). Then other files can `#include "X.h"` and call the function.

answered Apr 4 '10 at 17:08



cpalmer

1,077 6 10

Why is this -1? This is what you should be doing. – [cpalmer](#) Apr 4 '10 at 17:28

Can someone explain why this is a bad idea, I was always under the impression this was standard practice? The only issue that occurs from this is circular inclusion. But, otherwise if you plan your includes you won't run into that problem. – [Nathan Adams](#) Apr 4 '10 at 17:32

- 2 Because not all functions are meant to be public. If you have static void foo() defined *after* static int bar(), and bar calls foo .. well :) Why would you prototype functions that aren't exposed in a public header? I'm not the one who down voted, though. – [Tim Post](#) ♦ Apr 4 '10 at 18:35

- 1 I assume the downvote (not from me) was because this doesn't address the question. – anon Apr 4 '10 at 18:41

@Tim, bar can call foo because the compiler will already know about it from the prototype. Unless this is untrue in C, it should be true in C++. Can you post an example so maybe we can get a better idea of what you mean? @Neil Even though it doesn't answer the question directly it does relate to a "best practice" that is related to the question which I don't see as a need to down vote. Maybe not an upvote, but it isn't something that I would want removed or is unnecessary. – [Nathan Adams](#) Apr 4 '10 at 22:15

Function prototype is not mandatory according to the `c99` standard.

answered Apr 4 '10 at 17:13



zoli2k

2,187 3 16 31

It's not absolutely necessary to declare a function for the calling code to compile. There are caveats though. Undeclared function is assumed to return `int` and compiler will issue warnings first about function being undeclared and then about any mismatches in return type and parameter types.

Having said that it's obvious that declaring functions properly with prototypes is a much better practice.

answered Apr 4 '10 at 17:10



Nikolai Fetissov

68.6k 8 85 146

Select 'Options' menu and then select 'Compiler | C++ Options'. In the dialog box that pops up, select 'CPP always' in the 'Use C++ Compiler' options. Again select 'Options' menu and then select 'Environment | Editor'. Make sure that the default extension is 'C' rather than 'CPP'.

answered Jun 10 '14 at 7:27



user3725015

1

- 1 This does not seem to answer the question that was asked but some loosely related question about a specific IDE. – [Jonathan Leffler](#) Feb 27 '15 at 5:58