What is the difference between "long", "long long", "long int", and "long long int" in C++?

Ask Question

Asked 11 years, 3 months ago Modified 2 years, 10 months ago Viewed 626k times

317

I am transitioning from Java to C++ and have some questions about the `long` data type. In Java, to hold an integer greater than 2^{32}, you would simply write `long x;`. However, in C++, it seems that `long` is both a data type and a modifier.

There seems to be several ways to use `long`:

```
long x;
long long x;
long int x;
long long int x;
```

Also, it seems there are things such as:

```
long double x;
```

and so on.

What is the difference between all of these various data types, and do they all have the same purpose?

c++ long-integer

Share Improve this question Follow

edited Nov 22, 2015 at 18:09
Jamal
771 ● 7 ● 22 ● 32

asked Sep 24, 2013 at 1:49
1110101001
5,087 ● 10 ● 27 ● 48

2 stackoverflow.com/q/6462439/139010 – Matt Ball Sep 24, 2013 at 1:53

2 @user2612743 - to be safe, think about what your requirements are and use the appropriate type. `long long` might be slower than `long`, which might be slower than `int`. – Pete Becker Sep 24, 2013 at 13:51

1 msdn.microsoft.com/en-us/library/s3f49ktz.aspx – 1110101001 Nov 13, 2014 at 23:58

5 No, "to be safe, use long long" is the same as saying "to be safe, just give everyone one earth AIDS, so we don't have to worry about safe sex, we've all already got it anyhow!" Silly, no? Think about the data and what possible values it can have, and use the best-fitting type. This also helps the compiler make additional optimizations without breaking the original code intent, such as if it has to load additional libraries to handle numbers larger than the natural bit-width of the target platform. – C. M. Jan 27, 2015 at 23:52

Using long long with win32 listboxes and the like is wont to play havoc with your memory and other variables- even if the limits aren't breached. Even with the innocuous looking C4244 warnings it's not easy to detect. – Laurie Stearn Feb 26, 2016 at 9:56

Add a comment

7 Answers

Sorted by: Highest score (default)

270

`long` and `long int` are identical. So are `long long` and `long long int`. In both cases, the `int` is optional.

As to the difference between the two sets, the C++ standard mandates minimum ranges for each, and that `long long` is at *least* as wide as `long`.

The controlling parts of the standard (C++11, but this has been around for a long time) are, for one, `3.9.1 Fundamental types`, section 2 (a later section gives similar rules for the unsigned integral types):

> There are five standard signed integer types : signed char, short int, int, long int, and long long int. In this list, each type provides at least as much storage as those preceding it in the list.

There's also a table 9 in `7.1.6.2 Simple type specifiers`, which shows the "mappings" of the specifiers to actual types (showing that the `int` is optional), a section of which is shown below:

```
Specifier(s)         Type
-------------    -------------
long long int    long long int
long long        long long int
long int         long int
long             long int
```

Note the distinction there between the specifier and the type. The specifier is how you tell the compiler what the type is but you can use different specifiers to end up at the same type.

Hence `long` on its own is neither a type *nor* a modifier as your question posits, it's simply a specifier for the `long int` type. Ditto for `long long` being a specifier for the `long long int` type.

Although the C++ standard itself doesn't specify the minimum ranges of integral types, it does cite C99, in `1.2 Normative references`, as applying. Hence the minimal ranges as set out in `C99 5.2.4.2.1 Sizes of integer types <limits.h>` are applicable.

In terms of `long double`, that's actually a floating point value rather than an integer. Similarly to the integral types, it's required to have at least as much precision as a `double` and to provide a superset of values over that type (meaning *at least* those values, not necessarily *more* values).

Share Improve this answer Follow edited Dec 15, 2013 at 7:19 answered Sep 24, 2013 at 1:53

paxdiablo
880k ●241 ●1.6k ●2k

10 same thing with `unsigned` and `unsigned int` – Kal Sep 24, 2013 at 1:54

7 I'm pretty sure `long` is at least 32 bits (2^31-1 on either side of zero) and `long long` is at least 64 (2^63-1 on either side). – Qaz Sep 24, 2013 at 1:55

2 And `long double` is guaranteed to have at least the range of `double`, but it may be the same. It depends on the computer. Some FPUs have extended precision; the x87 chips had 32-bit single precision, 64-bit double precision, and 80-bit extended precision. – Eric Jablow Sep 24, 2013 at 1:56

As for the range requirements, the C++11 standard references the C11 standard, which has actual ranges. – Qaz Sep 24, 2013 at 2:05

1 @AbhishekMane: I believe the problem with `long int(b)` may simply be that `long` is being considered a separate thing from the rest of the cast, `int(b)`. If you use `long(b)` or `(long int)(b)` or even one of `typedef long int LI;` or `using LI = long int;` with `LI(b)`, it works fine. I'm finding it hard to generate a lot of interest though, since C++ developers should probably be avoiding the old style casts as much as possible. By all means use `(long int)b` if you're incorporating legacy C code but there's no `<someType>(var)` in C :-) – paxdiablo Apr 14, 2021 at 11:33

Show **4** more comments

83

Long and long int are at least 32 bits.

long long and long long int are at least 64 bits. You must be using a c99 compiler or better.

long doubles are a bit odd. Look them up on Wikipedia for details.

Share Improve this answer Follow edited Sep 9, 2018 at 10:00 answered Sep 24, 2013 at 1:57

MBT
24k ● 22 ● 94 ● 110

Eric Lippert
660k ● 183 ● 1.3k ● 2.1k

Add a comment

32

`long` is equivalent to `long int`, just as `short` is equivalent to `short int`. A `long int` is a signed integral type that is *at least* 32 bits, while a `long long` or `long long int` is a signed integral type is *at least* 64 bits.

This doesn't necessarily mean that a `long long` is wider than a `long`. Many platforms / ABIs use the `LP64` model - where `long` (and pointers) are 64 bits wide. Win64 uses the `LLP64`, where `long` is still 32 bits, and `long long` (and pointers) are 64 bits wide.

There's a good summary of 64-bit data models here.

`long double` doesn't guarantee much other than it will be *at least* as wide as a `double`.

Share Improve this answer Follow

answered Sep 24, 2013 at 2:00
Brett Hale
22.3k ● 2 ● 64 ● 99

2 Is there any platform where all 4 of the integer types: short, int, long and long long have different size?
– Sapien2 Mar 13, 2023 at 13:54

Add a comment

28

While in Java a `long` is always 64 bits, in C++ this *depends on computer architecture and operating system*. For example, a `long` is 64 bits on Linux and 32 bits on Windows (this was done to keep backwards-compatability, allowing 32-bit programs to compile on 64-bit Windows without any changes). `long int` is a synonym for `long`.

Later on, `long long` was introduced to mean "long (64 bits) on Windows for real this time". `long long int` is a synonym for this.

It is considered good C++ style to **avoid** `short`, `int`, `long` etc. and instead use:

```
std::int8_t   # exactly  8 bits
std::int16_t  # exactly 16 bits
std::int32_t  # exactly 32 bits
std::int64_t  # exactly 64 bits

std::size_t   # can hold all possible object sizes, used for indexing
```

You can use these `int*_t` types by including the `<cstdint>` header. `size_t` is in `<stdlib.h>`.

Share Improve this answer Follow   edited Mar 5, 2022 at 17:20   answered Jun 1, 2020 at 12:39

xjcl
15.2k ● 8 ● 82 ● 104

Add a comment

12

This looks confusing because you are taking `long` as a datatype itself.

`long` is nothing but just the shorthand for `long int` when you are using it alone.

`long` is a modifier, you can use it with `double` also as `long double`.

`long` == `long int`.

Both of them take 4 bytes.

Share Improve this answer Follow

answered Jun 1, 2016 at 8:30
Siraj Alam
10k ● 10 ● 59 ● 72

2 Long taking 4 bytes is only valid on Win64, it's platform dependent – Prodigle Jun 12, 2019 at 9:11

2 Yes obviously...not only long...int takes 4 bytes and 2 bytes depend on the platform, no doubt. – Siraj Alam Jun 12, 2019 at 9:13

Add a comment

3

Historically, in early C times, when processors had 8 or 16 bit wordlength, `int` was identical to todays `short` (16 bit). In a certain sense, int is a more abstract data type than `char`, `short`, `long` or `long long`, as you cannot be sure about the bitwidth.

When defining `int n;` you could translate this with "give me the best compromise of bitwidth and speed on this machine for n". Maybe in the future you should expect compilers to translate `int` to be 64 bit. So when you want your variable to have 32 bits and not more, better use an explicit `long` as data type.

[Edit: `#include <stdint.h>` seems to be the proper way to ensure bitwidths using the int##_t types, though it's not yet part of the standard.]

Share Improve this answer Follow

edited Nov 20, 2014 at 11:25

answered Aug 13, 2014 at 11:23
thomiel
2,937 ● 24 ● 41

3 That last part, using a "long" to get 32-bits when int is 64-bits, is incorrect. If int is 64-bits, then long will be *at least* 64-bits, as well. Long is guaranteed to be *at least* as large as int, although it may be larger, but never smaller. Most compilers have various methods which allow the programmer to be more specific, such as a (non-portable) __int32 type that is exactly 32-bits, and so on. – C. M. Jan 27, 2015 at 23:43

2 As defined in the C standard, a `long` is guaranteed to be *at least* 32 bits. (Standards may change tough.) Current C++14 draft just says: @C.M. "Plain ints have the natural size suggested by the

architecture of the execution environment the other signed integer types are provided to meet special needs" (section 3.9.1). I found no word about the length relations of various ints in it. __int32 isn't really part of the standard, but since C++11 there are typedefs available like int_fast32_t or int_least32_t available to get you exactly what you want. – thomiel Aug 19, 2015 at 17:06

1 I'd say that on twentieth-century implementations for general-purpose reprogrammable microcomputers, `char` was almost unanimously 8 bits, `short` was 16, and `long` was 32; `int` could either be 16 or 32. Note for some platforms (esp. the 68000) both 16-bit and 32-bit `int` were quite common, and indeed some compilers had options to support either. Code which needed to be portable was thus expected to use `short` or `long` in preference to `int`. – supercat Jun 23, 2016 at 14:30

Add a comment

-2

There is no deffirence, (long long x) is equivalent to (long long int x), but the second confirms that variable x is integer

Share Improve this answer Follow

answered Mar 23, 2021 at 23:51

Mohanad Talat
7 ●3

3 What does this answer add to the previous ones? – zkoza Mar 24, 2021 at 0:25

This answer is short and straight to the reason as the answer need – Mohanad Talat May 7, 2022 at 14:13

No, a variable of type `long long int` is not "integer", because only `int` can be "integer". A variable of type `long long int` has an integral type though, and I believe that's what you meant. But same is for `long long` type. The `int` in `long long int` doesn't add anything but confusion IMO. – Eric Nov 24, 2023 at 12:12

Add a comment

Not the answer you're looking for? Browse other questions tagged c++ long-integer or ask your own question.