1.7 — Keywords and naming identifiers

BY ALEX ON JUNE 5TH, 2007 | LAST MODIFIED BY NASCARDRIVER ON JANUARY 29TH, 2020

Keywords

C++ reserves a set of 84 words (as of C++17) for its own use. These words are called **keywords** (or reserved words), and each of these keywords has a special meaning within the C++ language.

Here is a list of all the C++ keywords (through C++17):

alignas (C++11)	enum	return
alignof (C++11)	explicit	short
and	export	signed
and_eq	extern	sizeof
asm	false	static
auto	float	static_assert (C++11)
bitand	for	static_cast
bitor	friend	struct
bool	goto	switch
break	if	template
case	inline	this
catch	int	thread_local (C++11)
char	long	throw
char16_t (C++11)	mutable	true
char32_t (C++11)	namespace	try
class	new	typedef
compl	noexcept (C++11)	typeid
const	not	typename
constexpr (C++11)	not_eq	union
const_cast	nullptr (C++11)	unsigned
continue	operator	using
decltype (C++11)	or	virtual
default	or_eq	void
delete	private	volatile
do	protected	wchar_t
double	public	while
dynamic_cast	register	xor
else	reinterpret_cast	xor_eq

The keywords marked (C++11) were added in C++11. If your compiler is not C++11 compliant (or does have C++11 functionality, but it's turned off by default), these keywords may not be functional.

C++11 also adds two special identifiers: *override* and *final*. These have a specific meaning when used in certain contexts but are not reserved.

You have already run across some of these keywords, including *int* and *return*. Along with a set of operators, these keywords and special identifiers define the entire language of C++ (preprocessor commands excluded). Because keywords and special identifiers have special meaning, your IDEs will likely change the text color of these words (often to blue) to make them stand out from other identifiers.

By the time you are done with this tutorial series, you will understand what almost all of these words do!

Identifier naming rules

As a reminder, the name of a variable (or function, type, or other kind of item) is called an identifier. C++ gives you a lot of flexibility to name identifiers as you wish. However, there are a few rules that must be followed when naming identifiers:

- The identifier can not be a keyword. Keywords are reserved.
- The identifier can only be composed of letters (lower or upper case), numbers, and the underscore character. That means the name can not contain symbols (except the underscore) nor whitespace (spaces or tabs).
- The identifier must begin with a letter (lower or upper case) or an underscore. It can not start with a number.
- C++ is case sensitive, and thus distinguishes between lower and upper case letters. `nvalue` is different than `nValue` is different than `NVALUE`.

Identifier naming best practices

Now that you know how you *can* name a variable, let's talk about how you *should* name a variable (or function).

First, it is a convention in C++ that variable names should begin with a lowercase letter. If the variable name is one word, the whole thing should be written in lowercase letters.

```
int value; // correct

int Value; // incorrect (should start with lower case letter)
int VALUE; // incorrect (should start with lower case letter)
int VaLuE; // incorrect (see your psychiatrist) ;)
```

Most often, function names are also started with a lowercase letter (though there's some disagreement on this point). We'll follow this convention, since function *main* (which all programs must have) starts with a lowercase letter, as do all of the functions in the C++ standard library.

Identifier names that start with a capital letter are typically used for user-defined types (such as structs, classes, and enumerations, all of which we will cover later).

If the variable or function name is multi-word, there are two common conventions: words separated by underscores, or intercapped (sometimes called camelCase, since the capital letters stick up like the humps on a camel).

```
int my_variable_name; // correct (separated by underscores)
int my_function_name(); // correct (separated by underscores)

int myVariableName; // correct (intercapped/CamelCase)
int myFunctionName(); // correct (intercapped/CamelCase)

int my variable name; // invalid (whitespace not allowed)
int my function name(); // invalid (whitespace not allowed)

int MyVariableName; // valid but incorrect (should start with lower case letter)
int MyFunctionName(); // valid but incorrect (should start with lower case letter)
```

In this tutorial, we will typically use the intercapped approach because it's easier to read (it's easy to mistake an underscore for a space in dense blocks of code). But it's common to see either -- the C++ standard library uses the underscore method for both variables and functions. Sometimes you'll see a mix of the two: underscores used for variables and intercaps used for functions.

It's worth noting that if you're working in someone else's code, it's generally considered better to match the style of the code you are working in than to rigidly follow the naming conventions laid out above.

Best practice

When working in an existing program, use the conventions of that program (even if they don't conform to modern best practices). Use modern best practices when you're writing new programs.

Second, you should avoid naming your identifiers starting with an underscore, as these names are typically reserved for OS, library, and/or compiler use.

Third, your identifiers should make clear what the value they are holding means (particularly if the units isn't obvious). Identifiers should be named in a way that would help someone who has no idea what your code does be able to figure it out as quickly as possible. In 3 months, when you look at your program again, you'll have forgotten how it works, and you'll thank yourself for picking variable names that make sense.

However, giving a trivial variable an overly complex name impedes overall understanding of what the program is doing almost as much as giving a widely used identifier an inadequate name. Therefore, a good rule of thumb is to make the length of a identifier proportional to how widely it is used. An identifier with a trivial use can have a short name (e.g. such as *x*). An identifier that is used more broadly (e.g. a function that is called from many different places in a program) should have a longer and more descriptive name (e.g. instead of *open*, try *openFileOnDisk*).

int ccount	Bad	What does the c before "count" stand for?
int customerCount	Good	Clear what we're counting
int i	Either	Okay if use is trivial, bad otherwise
int index	Either	Okay if obvious what we're indexing
int totalScore	Either	Okay if there's only one thing being scored, otherwise too ambiguous
int _count	Bad	Do not start variable names with underscore
int count	Either	Okay if obvious what we're counting
int data	Bad	What kind of data?
int time	Bad	Is this in seconds, minutes, or hours?
int minutesElapsed	Good	Descriptive
int value1, value2	Either	Can be hard to differentiate between the two
int numApples	Good	Descriptive
int monstersKilled	Good	Descriptive
int x, y	Either	Okay if use is trivial, bad otherwise

Finally, a clarifying comment can go a long way. For example, say we've declared a variable named *numberOfChars* that is supposed to store the number of characters in a piece of text. Does the text "Hello World!" have 10, 11, or 12 characters? It depends on whether we're including whitespace or punctuation. Rather than naming the variable *numberOfCharsIncludingWhitespaceAndPunctuation*, which is rather lengthy, a well placed comment on the declaration line should help the user figure it out:

```
1 // holds number of chars in a piece of text -- including whitespace and punctuation!
2 int numberOfChars;
```

Quiz time

Question #1

Based on how you *should* name a variable, indicate whether each variable name is correct (follows convention), incorrect (does not follow convention), or invalid (will not compile), and why.

int sum;
Show Solution

int _apples;
Show Solution

int VALUE;
Show Solution

int my variable name;
Show Solution

int TotalCustomers;
Show Solution

int void;
Show Solution

int numFruit;
Show Solution

int 3some;
Show Solution

int meters_of_pipe;
Show Solution


1.8 -- Introduction to literals and operators


Index


1.6 -- Uninitialized variables and undefined behavior

C++ TUTORIAL | PRINT THIS POST

181 comments to 1.7 — Keywords and naming identifiers

« Older Comments 1 2 3

Vitaliy Sh.
January 15, 2020 at 7:19 pm · Reply

Hi Sires! The humble list...

"Here is a list of all the C++ keywords (through C++17):"
** Is that feasible to do like in dictionaries?
** It could make this list friendlier:

```
...
<h4>
        L</h4>
<li>
        long</li>

<h4>
        M</h4>
<li>
        mutable</li>

<h4>
        N</h4>
...
```

"The keywords marked (C++11)..."
** "(C++11)" (quotes) || (C++11)

"nvalue is different than nValue is different than NVALUE."
** ``... than nValue and is ...``

"Most often, functions names"
** ``function names``

"If the variable or function name is multi-word, there are two common conventions: separated by underscores, or intercapped"
** ``... conventions: words separated ...``

Vitaliy Sh.
January 14, 2020 at 5:27 pm · Reply

FFFF:

```
false
  float
    for
      friend
// Read and sing the
// C++ alpha-bet...
```

fucitol
November 14, 2019 at 7:57 pm · Reply

Alex why don't you make a discord group .....where we all, who use this website to learn c++ can share knowledge and espertise over the language and you can be the admin.

Alex
November 15, 2019 at 4:04 pm · Reply

Interesting idea. I'll look into it.

alex
December 12, 2019 at 1:58 pm · Reply

seems like a good idea!

Petertheminer11
January 1, 2020 at 8:39 am · Reply

what is the name to the discord?
(If there is one)

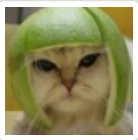
Alex
January 2, 2020 at 3:19 pm · Reply

There isn't one as of yet.

ren0d1
October 24, 2019 at 3:20 am · Reply

Firstly, I would like to say that it is a great tutorial series containing very well structured pages for each point of interest!
Then, the only thing I would propose here is to simply change your "CamelCase" to "camelCase" for consistency, to ease the understanding for readers unfamiliar with the term as well as to avoid potential confusion with the PascalCase notation.

Alex
October 25, 2019 at 2:10 pm · Reply



Lesson amended. Thanks!

Ayio
October 7, 2019 at 6:03 am · Reply

```
1 int 3some;
```

Xddddd

Megan
October 2, 2019 at 1:09 pm · Reply

Why aren't cin and cout keywords? Or is it cause they are a part of the iostream library?

nascardriver
October 3, 2019 at 12:14 am · Reply

They are variables. That might seem weird now, you'll understand it once you learn about classes and operators.

4STR4L_3N3MY
September 10, 2019 at 7:31 am · Reply

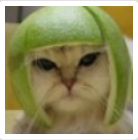
(this comment is not asking for help or advice): I heard mastering C++ will take a lifetime - I bet i can do it in half if i tried hard enough!

btw great tutorials. You are a really great teacher. in the future i will reference this website to anyone wanting to learn C++ , or generally wanting to start a programming language since the start is perfect for that.

the quizzes are pretty neat too.

Anurag Kashyap
August 27, 2019 at 9:20 am · Reply

♦Its written in the section that {The identifier must begin with a letter (lower or upper case) "or an underscore." It can not start with a number}.
But in one of the solutions you said that "_apple" is incorrect ??
HOW?
☻ BTW loved your work. ☻
♥ LOVE FROM INDIA ♥



Alex
August 27, 2019 at 1:34 pm · Reply

This is the difference between "can" and "should".

C++ will allow you to name your variables starting with an underscore. But you shouldn't.

Anurag Kashyap
August 27, 2019 at 6:23 pm · Reply

GOT YOUR POINT.
♥☻

Gerry
August 21, 2019 at 9:47 pm · Reply

The is a really well written and organised tutorial on C++!
Believe I have found a typo:
In the table at the end of this lesson, one row has the comment "Do not start veriable names with underscore"; but the associated identifier is int_count which does not start with underscore.

Alex
August 24, 2019 at 12:12 pm · Reply

I don't see a typo here: int is the type, and _count is the variable name, which does start with an underscore. It's a little hard to see the space between them, which might account for your confusion here.

Gerry
September 4, 2019 at 11:01 pm · Reply

Thanks. You are right - I missed the space and misread the line.
Gerry

calamity
August 21, 2019 at 8:42 am · Reply

int 3some;//Hmmm

Waheedullah Hanifi
August 6, 2019 at 8:33 am · Reply

Hi there
These information are more easy to understand the concept of c++.
I'm study computer since at university. the teacher has never told us these important information.
we hope that you make more advance c++ tutorials, like how can we create a GUI in c++ and ETC.
Thanks Regard
Waheedullah hanifi,
from Afghanistan.

Phil J
July 15, 2019 at 3:10 am · Reply

In Identifier naming best practices you have given a list of variable names just after you introduce the word CamelCase. I believe the second one should have a semi-colon.

```
1 int my_variable_name; // correct (separated by underscores)
```

```
2 int my_function_name() // correct (separated by underscores
```

I hope this helps, and thanks for this tutorial.

Nirbhay
July 9, 2019 at 4:59 am · Reply

"See your psychiatrist" hahahaah! Nice one :)

Alireza
June 19, 2019 at 8:16 am · Reply

Hello author,
Is it important to memorize all of keywords or not ?
I'm studying this tutorial until lesson 8.5b and you haven't covered some of these keywords(such as and, or, xor ,...).
I know I didn't finish these lessons who you will cover some other keywords later(such as virtual, ...).

nascardriver
June 19, 2019 at 9:01 am · Reply

You don't need to memorize all keywords. More will be covered later.

Lisa
June 3, 2019 at 7:38 am · Reply

are keywords and data types the same thing?

Alex
June 11, 2019 at 3:08 pm · Reply

No. Keywords are words that are reserved by the language and may not be used as identifiers. A data type determines what type of information an object stores (e.g. a number, a letter, etc...)

"int" is both a keyword and a data type, whereas "for" is a keyword and not a data type.

Lisa
June 17, 2019 at 10:07 pm · Reply

So, in your previous examples...
int x; // define a variable named x, of type int
int y, z; // define two integer variables, named y and z
int width{ 5 }; // brace (uniform) initialization of value 5 into variable width
double width; // define a variable named width, of type double
void my_function_name() // correct (separated by underscores)

the words - int, double, void are written.
Are these keywords or datatypes?

nascardriver

June 19, 2019 at 3:08 am · Reply

All data type names are keywords.
'int' is a data type. When writing it down, it's also a keyword.

Destiny
June 3, 2019 at 6:39 am · Reply

Hi Alex,
Reading through all the comments got me confused on understanding what a Function/ Functions in the C++ standard library and Function main really is. Would it be a possibility if you'd be able to clarify it for me?

Also... Have we learnt 'void' yet?
I understand 'int' very well, however, don't think I have learnt void yet.

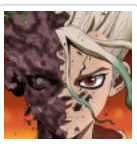
Lisa
June 4, 2019 at 3:47 am · Reply

Yeah! I'm a little confused on this as well.

Alex
June 11, 2019 at 3:05 pm · Reply

A large part of chapter 2 is devoted to functions -- so just keep reading. All you really need to understand at this point is that functions and variables have the same naming rules.

I replaced the use of "void" with "int" since we haven't covered void yet. Thanks for pointing that out.



Badreddine Boukheit
August 25, 2019 at 5:05 am · Reply

you didn't replace them in the quiz section x) I fell for it even though I know the keyword void x)

Alex
August 27, 2019 at 10:32 am · Reply

I did replace them. The only occurrence left uses "void" as a identifier, not as a type. That's fair game for this lesson.

oxygène
April 10, 2019 at 9:18 am · Reply

int 3some;
Invalid -- variable names can not start with a number.
int threeSome;
Invalid -- too explicit.

Gamef
April 27, 2019 at 1:43 pm · Reply

Lol

Vitaliy Sh.
January 15, 2020 at 6:53 pm · Reply

What do we get if an novice programmer, an source code file, and an compiler meet?
Show Solution...

Bastian Jerome
March 22, 2019 at 10:14 am · Reply

Why do variables need full definitional names if you label them anyways? I use shorthand names for variables, because to type out every single letter of every single name I use would be rather time consuming, and the shorthand is still rather obvious, especialy when you read the comment where they are declaired (when I feel the comment is neccisary).

For example, in the games I have made with Java, I use velX and velY intsead of velocityX and velocityY or velocityAlongXAxis and velocityAlongYAxis, but then I put that they are the velocities along their respective axes in a comment above their declairation. Or with a class that moves the camera, I have shortend it to Cam and put a comment at the top explaing that it is the class that controlls the camera. We already use shorthand for things like Enumeration, integer, boolean, Console In, console out, standard, etc. so why not take cues from those that started that with our shorthand.

Alex
March 24, 2019 at 1:00 pm · Reply

That's fine, as long as it's clear what they are, and they can be reasonably disambiguated from other variables.

Walter
April 15, 2019 at 5:40 am · Reply

Most code editors have auto-complete, so the "longer time" it takes to write isn't a great argument in my eyes. When I code, I spend 5% of it actually typing and 95% of it thinking, so the few extra key strokes don't bother me much.

You read code more than you write it. The half second you gain by writing shorthand, you lose every time you read the variable and have to expand it in your head again. It is ultimately up to you, of course, but I feel there's good arguments for writing out a full and clear name.

Chris
March 6, 2019 at 8:32 am · Reply

If your compiler is not C++11 compliant (or does have C++11 functionality turned on by default), these keywords may not be functional.

I'd assume from the context:
(or does have C++11 functionality turned on by default)
is meant to be:

(or does have C++11 functionality, but it's turned off by default)
would that be right?

Alex
March 8, 2019 at 3:53 pm · Reply

Correct, text amended. Thanks for pointing out the error.

Louis Cloete
February 7, 2019 at 7:51 am · Reply

In the green "Best practice" block: "Use best practices if its your program." "its" should be "it's"

Aurme
January 13, 2019 at 11:45 am · Reply

I was wondering if the definitions could be clarified slightly for me. In section 1.3 it states:
`A variable in C++ is simply an object that has a name.`

Then in this chapter, 1.4c:

`The name of a variable, function, type, or other kind of object in C++ is called an identifier.`

I am confused by this statement saying the name of a variable is an identifier, and the previous comment saying an object with a name is a variable, as well as then the statement saying the name of other kinds of objects is called an identifier.

so if an object with a name is a variable, then how does the variable part of that object also have a name? I believe I have confused my definitions somewhere.

It is my understanding that a variable is simply a named reference to a memory location, while an identifier is a word identifying any entity.

I would call

```
1 int x;
```

A variable declaration

```
1 void foo() { ...
```

a function declaration with an identifier of foo. (or simply call it its name when casually speaking)

Is this incorrect?

or perhaps a variable is just a specific type of identifier?

Alex
January 14, 2019 at 10:27 pm · Reply

An object is a region of memory that can store a value.
A function is not an object.
Both an object and a function can have a name, which is called an identifier.
An object that has a name is called a variable. A function that has a name is called a named function.

That's really all there is to it. There is no such thing as "the variable part of that object" -- a variable is an object.

Rancorous
November 26, 2018 at 4:20 am · Reply

I know you get this all the time but THANK YOU for keeping this website up and running. I've learned so much from you.

SCXRRADIATION
October 18, 2018 at 1:19 pm · Reply

hmm 3some hmm

vbot
September 25, 2018 at 8:37 am · Reply

Hi Alex,

I got used to (years of AS3 / JavaScript coding) adding a double underscore as a prefix to local variable names inside functions in order to keep them clearly separated from the global ones. When I look at the code inside one of my functions searching for global variables, my eye / brain just skips the ones beginning with a double underscore. This way I can quickly distinguish between global and local variables without having to actually read them. Other thing I like about this approach is that I don't need any editor-text-decorations like italic fonts or colors to "mark" them.

Question: How critical would it be (for me) to still use double underscores for local variable names inside function-scope in C++ and why exactly?

```
1  int globalVar = 0;
2
3  int exampleFunction(int a, int b) {
4
5  int __multiplied = a * b;
6  int __added = a + b;
7  int __substracted = a - b;
8
9  int __result = __multiplied + __added + __substracted;
10
11 globalVar = __result;
12 }
```

Thanks!

nascardriver
September 25, 2018 at 8:49 am · Reply

Hi vbot!

Quoting the standard N4762 § 5.10 (3.1)
"Each identifier that contains a double underscore __ or begins with an underscore followed by an uppercase letter is reserved to the implementation for any use."
Your code might work with your compiler, but it might not work with another compiler because you're using a reserved identifier.
In C++ prefixes like
"g_" (global)
"m_" (member)
"k_" (constant)
"s_" (static)
are common when naming variables.

```
int g_iMyGlobalInt;
int m_iMyMemberInt;
// ...
```

Local variables (and parameters) don't have special prefixes.

vbot
September 25, 2018 at 12:22 pm · Reply

Hi nascardriver,

thanks for the fast reply! Ok, maybe then I should better drop this habbit, it was rather confusing to other programers anyway. ;)

DEBUG: my exampleFunction above has to be void! ;)

MrStark
September 15, 2018 at 3:58 am · Reply

In quiz, why second variable name is wrong, while a name can begin with either lowercase letter or underscore?

nascardriver
September 15, 2018 at 7:20 am · Reply

The quiz isn't about what you _can_ do, but about what you _should_ do.
In my opinion this shouldn't be part of this tutorial, since there are many naming conventions which are equally good.

MrStark
September 15, 2018 at 8:53 am · Reply

I brought this topic because in naming rules you have stated that an identifier must start with lower case letter or underscore but you contradict it in you quiz as improper and not according to standards. Either the rules for naming identifier in your articles are wrong or the quiz. (Not personal preference but according to standards)

nascardriver
September 15, 2018 at 8:57 am · Reply

These aren't my lessons, I'm not affiliated with learncpp.
Variable names must start with a letter or underscore, but they should start with a lower case letter. The quiz isn't about must, but about should.

MrStark
September 15, 2018 at 8:58 am · Reply

Ok. I was taking them as according to standard conventions.(Pick which variables are improperly named according to standard conventions).

nascardriver

September 15, 2018 at 9:01 am · Reply

Yep, seems like you mixed up conventions and rules. Rules are what causes your code to not compile and conventions are what people like to do without breaking the rules.

« Older Comments 1 2 3