# C Tokens

In a C source program, the basic element recognized by the compiler is the "token." A token is source-program text that the compiler does not break down into component elements.

[Syntax](javascript:void(0))

*token*:

*keyword*

*identifier*

*constant*

*string-literal*

*operator 运算符*

*punctuator 标点符号.*

## White-Space Characters

**Visual Studio 2013**

[Other Versions](javascript:;)

https://i-msdn.sec.s-msft.com/Areas/Epx/Content/Images/ImageSprite.png?v=635914614327034862

Space, tab, linefeed, carriage-return, formfeed, vertical-tab, and newline characters are called "white-space characters" because they serve the same purpose as the spaces between words and lines on a printed page — they make reading easier. Tokens are delimited (bounded) by white-space characters and by other tokens, such as operators and punctuation. When parsing code, the C compiler ignores white-space characters unless you use them as separators or as components of character constants or string literals. Use white-space characters to make a program more readable. Note that the compiler also treats comments as white space.

## C Comments

**Visual Studio 2013**

[Other Versions](javascript:;)

A "comment" is a sequence of characters beginning with a forward slash/asterisk combination (**/\***) that is treated as a single white-space character by the compiler and is otherwise ignored. A comment can include any combination of characters from the representable character set, including newline characters, but excluding the "end comment" delimiter (**\*/**). Comments can occupy more than one line but cannot be nested.

Comments can appear anywhere a white-space character is allowed. Since the compiler treats a comment as a single white-space character, you cannot include comments within tokens. The compiler ignores the characters in the comment.

Use comments to document your code. This example is a comment accepted by the compiler:

/\* Comments can contain keywords such as

for and while without generating errors. \*/

Comments can appear on the same line as a code statement:

printf( "Hello\n" ); /\* Comments can go here \*/

You can choose to precede functions or program modules with a descriptive comment block:

/\* MATHERR.C illustrates writing an error routine

\* for math functions.

\*/

Since comments cannot contain nested comments, this example causes an error:

/\* Comment out this routine for testing

/\* Open file \*/

fh = \_open( "myfile.c", \_O\_RDONLY );

.

.

.

\*/

The error occurs because the compiler recognizes the first \*/, after the words Open file, as the end of the comment. It tries to process the remaining text and produces an error when it finds the \*/ outside a comment.

While you can use comments to render certain lines of code inactive for test purposes, the preprocessor directives **#if** and **#endif** and conditional compilation are a useful alternative for this task. For more information, see [Preprocessor Directives](https://msdn.microsoft.com/en-us/library/3sxhs2ty(v=vs.120).aspx) in the *Preprocessor Reference*.

**Microsoft Specific**

The Microsoft compiler also supports single-line comments preceded by two forward slashes (**//**). If you compile with /Za (ANSI standard), these comments generate errors. These comments cannot extend to a second line.

// This is a valid comment

Comments beginning with two forward slashes (**//**) are terminated by the next newline character that is not preceded by an escape character. In the next example, the newline character is preceded by a backslash (**\**), creating an "escape sequence." This escape sequence causes the compiler to treat the next line as part of the previous line. (For more information, see [Escape Sequences](https://msdn.microsoft.com/en-us/library/h21280bw(v=vs.120).aspx).)

// my comment \

i++;

Therefore, the i++; statement is commented out.

The default for Microsoft C is that the Microsoft extensions are enabled. Use /Za to disable these extensions.

**END Microsoft Specific**

## Evaluation of Tokens

**Visual Studio 2013**

[Other Versions](javascript:;)

https://i-msdn.sec.s-msft.com/Areas/Epx/Content/Images/ImageSprite.png?v=635914614327034862

When the compiler interprets tokens, it includes as many characters as possible in a single token before moving on to the next token. Because of this behavior, the compiler may not interpret tokens as you intended if they are not properly separated by white space. Consider the following expression:



In this example, the compiler first makes the longest possible operator (++) from the three plus signs, then processes the remaining plus sign as an addition operator (+). Thus, the expression is interpreted as (i++) + (j), not (i) + (++j). In this and similar cases, use white space and parentheses to avoid ambiguity and ensure proper expression evaluation.

**Microsoft Specific**

The C compiler treats a CTRL+Z character as an end-of-file indicator. It ignores any text after CTRL+Z.

**END Microsoft Specific**

C Keywords

**Visual Studio 2013**

[Other Versions](javascript:;)

https://i-msdn.sec.s-msft.com/Areas/Epx/Content/Images/ImageSprite.png?v=635914614327034862

"Keywords" are words that have special meaning to the C compiler. In translation phases 7 and 8, an identifier cannot have the same spelling and case as a C keyword. (See a description of [translation phases](https://msdn.microsoft.com/en-us/library/bxss3ska(v=vs.120).aspx) in the *Preprocessor Reference*; for information on identifiers, see [Identifiers](https://msdn.microsoft.com/en-us/library/e7f8y25b(v=vs.120).aspx).) The C language uses the following keywords:

|  |  |  |  |
| --- | --- | --- | --- |
| **auto** | **double** | **int** | **struct** |
| **break** | **else** | **long** | **switch** |
| **case** | **enum** | **register** | **typedef** |
| **char** | **extern** | **return** | **union** |
| **const** | **float** | **short** | **unsigned** |
| **continue** | **for** | **signed** | **void** |
| **default** | **goto** | **sizeof** | **volatile** |
| **do** | **if** | **static** | **while** |

You cannot redefine keywords. However, you can specify text to be substituted for keywords before compilation by using C [preprocessor directives](https://msdn.microsoft.com/en-us/library/3sxhs2ty(v=vs.120).aspx).

**Microsoft Specific**

The ANSI C standard allows identifiers with two leading underscores to be reserved for compiler implementations. Therefore, the Microsoft convention is to precede Microsoft-specific keyword names with double underscores. These words cannot be used as identifier names. For a description of the ANSI rules for naming identifiers, including the use of double underscores, see [Identifiers](https://msdn.microsoft.com/en-us/library/e7f8y25b(v=vs.120).aspx).

The following keywords and special identifiers are recognized by the Microsoft C compiler:

|  |  |  |  |
| --- | --- | --- | --- |
| **\_\_asm** | **dllimport**2 | **\_\_int8** | **naked** 2 |
| **\_\_based** 1 | **\_\_except** | **\_\_int16** | **\_\_stdcall** |
| **\_\_cdecl** | **\_\_fastcall** | **\_\_int32** | **thread**2 |
| **\_\_declspec** | **\_\_finally** | **\_\_int64** | **\_\_try** |
| **dllexport** 2 | **\_\_inline** | **\_\_leave** |  |

1. The **\_\_based** keyword has limited uses for 32-bit and 64-bit target compilations.

2. These are special identifiers when used with **\_\_declspec**; their use in other contexts is not restricted.

Microsoft extensions are enabled by default. To ensure that your programs are fully portable, you can disable Microsoft extensions by specifying the /Za option (compile for ANSI compatibility) during compilation. When you do this, Microsoft-specific keywords are disabled.

When Microsoft extensions are enabled, you can use the keywords listed above in your programs. For ANSI compliance, most of these keywords are prefaced by a double underscore. The four exceptions, **dllexport**, **dllimport**, **naked**, and **thread**, are used only with **\_\_declspec** and therefore do not require a leading double underscore. For backward compatibility, single-underscore versions of the rest of the keywords are supported.

**END Microsoft Specific**

C Identifiers

**Visual Studio 2013**

[Other Versions](javascript:;)

https://i-msdn.sec.s-msft.com/Areas/Epx/Content/Images/ImageSprite.png?v=635914614327034862

"Identifiers" or "symbols" are the names you supply for variables, types, functions, and labels in your program. Identifier names must differ in spelling and case from any keywords. You cannot use keywords (either C or Microsoft) as identifiers; they are reserved for special use. You create an identifier by specifying it in the declaration of a variable, type, or function. In this example, result is an identifier for an integer variable, and main and printf are identifier names for functions.

#include <stdio.h>

int main()

{

int result;

if ( result != 0 )

printf\_s( "Bad file handle\n" );

}

Once declared, you can use the identifier in later program statements to refer to the associated value.

A special kind of identifier, called a statement label, can be used in **goto** statements. (Declarations are described in [Declarations and Types](https://msdn.microsoft.com/en-us/library/dsyxx40f(v=vs.120).aspx) Statement labels are described in [The goto and Labeled Statements](https://msdn.microsoft.com/en-us/library/2c002fdz(v=vs.120).aspx).)

[Syntax](javascript:void(0))

*identifier*:

*nondigit*

*identifier nondigit*

*identifier digit*

*nondigit*: one of

**\_ a b c d e f g h i j k l m n o p q r s t u v w x y z**

**A B C D E F G H I J K L M N O P Q R S T U V W X Y Z**

*digit*: one of

**0 1 2 3 4 5 6 7 8 9**

The first character of an identifier name must be a *nondigit* (that is, the first character must be an underscore or an uppercase or lowercase letter). ANSI allows six significant characters in an external identifier's name and 31 for names of internal (within a function) identifiers. External identifiers (ones declared at global scope or declared with storage class **extern**) may be subject to additional naming restrictions because these identifiers have to be processed by other software such as linkers.

**Microsoft Specific**

Although ANSI allows 6 significant characters in external identifier names and 31 for names of internal (within a function) identifiers, the Microsoft C compiler allows 247 characters in an internal or external identifier name. If you aren't concerned with ANSI compatibility, you can modify this default to a smaller or larger number using the /H (restrict length of external names) option.

**END Microsoft Specific**

The C compiler considers uppercase and lowercase letters to be distinct characters. This feature, called "case sensitivity," enables you to create distinct identifiers that have the same spelling but different cases for one or more of the letters. For example, each of the following identifiers is unique:

add

ADD

Add

aDD

**Microsoft Specific**

Do not select names for identifiers that begin with two underscores or with an underscore followed by an uppercase letter. The ANSI C standard allows identifier names that begin with these character combinations to be reserved for compiler use. Identifiers with file-level scope should also not be named with an underscore and a lowercase letter as the first two letters. Identifier names that begin with these characters are also reserved. By convention, Microsoft uses an underscore and an uppercase letter to begin macro names and double underscores for Microsoft-specific keyword names. To avoid any naming conflicts, always select identifier names that do not begin with one or two underscores, or names that begin with an underscore followed by an uppercase letter.

**END Microsoft Specific**

The following are examples of valid identifiers that conform to either ANSI or Microsoft naming restrictions:

j

count

temp1

top\_of\_page

skip12

LastNum

**Microsoft Specific**

Although identifiers in source files are case sensitive by default, symbols in object files are not. Microsoft C treats identifiers within a compilation unit as case sensitive.

The Microsoft linker is case sensitive. You must specify all identifiers consistently according to case.

The "source character set" is the set of legal characters that can appear in source files. For Microsoft C, the source set is the standard ASCII character set. The source character set and execution character set include the ASCII characters used as escape sequences. See [Character Constants](https://msdn.microsoft.com/en-us/library/fwa1sfwk(v=vs.120).aspx) for information about the execution character set.

**END Microsoft Specific**

An identifier has "scope," which is the region of the program in which it is known, and "linkage," which determines whether the same name in another scope refers to the same identifier. These topics are explained in [Lifetime, Scope, Visibility, and Linkage](https://msdn.microsoft.com/en-us/library/592xa3sk(v=vs.120).aspx).

## Multibyte and Wide Characters

**Visual Studio 2013**

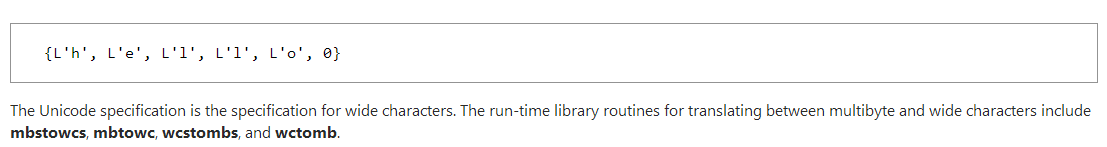
[Other Versions](javascript:;)

https://i-msdn.sec.s-msft.com/Areas/Epx/Content/Images/ImageSprite.png?v=635914614327034862

A multibyte character is a character composed of sequences of one or more bytes. Each byte sequence represents a single character in the extended character set. Multibyte characters are used in character sets such as Kanji.

Wide characters are multilingual character codes that are always 16 bits wide. The type for character constants is **char**; for wide characters, the type is**wchar\_t**. Since wide characters are always a fixed size, using wide characters simplifies programming with international character sets.

The wide-character-string literal L"hello" becomes an array of six integers of type **wchar\_t**.



## Trigraphs

**Visual Studio 2013**

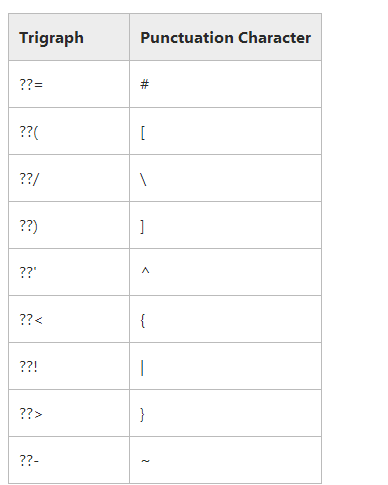
[Other Versions](javascript:;)

https://i-msdn.sec.s-msft.com/Areas/Epx/Content/Images/ImageSprite.png?v=635914614327034862

The source character set of C source programs is contained within the 7-bit ASCII character set but is a superset of the ISO 646-1983 Invariant Code Set. Trigraph sequences allow C programs to be written using only the ISO (International Standards Organization) Invariant Code Set. Trigraphs are sequences of three characters (introduced by two consecutive question marks) that the compiler replaces with their corresponding punctuation characters. You can use trigraphs in C source files with a character set that does not contain convenient graphic representations for some punctuation characters.

The following table shows the nine trigraph sequences. All occurrences in a source file of the punctuation characters in the first column are replaced with the corresponding character in the second column.

Trigraph Sequences



A trigraph is always treated as a single source character. The translation of trigraphs takes place in the first [translation phase](https://msdn.microsoft.com/en-us/library/bxss3ska(v=vs.120).aspx), before the recognition of escape characters in string literals and character constants. Only the nine trigraphs shown in the above table are recognized. All other character sequences are left untranslated.

The character escape sequence, **\?**, prevents the misinterpretation of trigraph-like character sequences. (For information about escape sequences, see [Escape Sequences](https://msdn.microsoft.com/en-us/library/h21280bw(v=vs.120).aspx).) For example, if you attempt to print the string What??! with this **printf** statement

printf( "What??!\n" );

the string printed is What| because ??! is a trigraph sequence that is replaced with the | character. Write the statement as follows to correctly print the string:

printf( "What?\?!\n" );

In this **printf** statement, a backslash escape character in front of the second question mark prevents the misinterpretation of ??! as a trigraph.

# C Constants

**Visual Studio 2013**

[Other Versions](javascript:;)

https://i-msdn.sec.s-msft.com/Areas/Epx/Content/Images/ImageSprite.png?v=635914614327034862

A "constant" is a number, character, or character string that can be used as a value in a program. Use constants to represent floating-point, integer, enumeration, or character values that cannot be modified.

[Syntax](javascript:void(0))

*constant*:

*floating-point-constant*

*integer-constant*

*enumeration-constant*

*character-constant*

Constants are characterized by having a value and a type. [Floating-point](https://msdn.microsoft.com/en-us/library/w9bk1wcy(v=vs.120).aspx), [integer](https://msdn.microsoft.com/en-us/library/2k2xf226(v=vs.120).aspx), and [character constants](https://msdn.microsoft.com/en-us/library/fwa1sfwk(v=vs.120).aspx) are discussed in the next three sections. Enumeration constants are described in [Enumeration Declarations](https://msdn.microsoft.com/en-us/library/whbyts4t(v=vs.120).aspx).

## C Floating-Point Constants

**Visual Studio 2013**

[Other Versions](javascript:;)

https://i-msdn.sec.s-msft.com/Areas/Epx/Content/Images/ImageSprite.png?v=635914614327034862

A "floating-point constant" is a decimal number that represents a signed real number. The representation of a signed real number includes an integer portion, a fractional portion, and an exponent. Use floating-point constants to represent floating-point values that cannot be changed.

[Syntax](javascript:void(0))

*floating-point-constant*:

*fractional-constant exponent-part*opt *floating-suffix*opt

*digit-sequence exponent-part floating-suffix*opt

*fractional-constant*:

*digit-sequence*opt **.***digit-sequence*

*digit-sequence***.**

*exponent-part*:

**e***sign*opt *digit-sequence*

**E***sign*opt *digit-sequence*

*sign*: one of

**+ –**

*digit-sequence*:

*digit*

*digit-sequence digit*

*floating-suffix*: one of

**f l F L**

You can omit either the digits before the decimal point (the integer portion of the value) or the digits after the decimal point (the fractional portion), but not both. You can leave out the decimal point only if you include an exponent. No white-space characters can separate the digits or characters of the constant.

The following examples illustrate some forms of floating-point constants and expressions:

15.75

1.575E1 /\* = 15.75 \*/

1575e-2 /\* = 15.75 \*/

-2.5e-3 /\* = -0.0025 \*/

25E-4 /\* = 0.0025 \*/

Floating-point constants are positive unless they are preceded by a minus sign (**–**). In this case, the minus sign is treated as a unary arithmetic negation operator. Floating-point constants have type **float**, **double**, or **long double**.

A floating-point constant without an **f**, **F**, **l**, or **L** suffix has type **double**. If the letter **f** or **F** is the suffix, the constant has type **float**. If suffixed by the letter **l**or **L**, it has type **long double**. For example:

100L /\* Has type long double \*/

100F /\* Has type float \*/

Note that the Microsoft C compiler maps **long double**to type **double**. See [Storage of Basic Types](https://msdn.microsoft.com/en-us/library/9c3yd98k(v=vs.120).aspx) for information about type **double**, **float**, and **long**.

You can omit the integer portion of the floating-point constant, as shown in the following examples. The number .75 can be expressed in many ways, including the following:

.0075e2

0.075e1

.075e1

75e-2

### Limits on Floating-Point Constants

**Visual Studio 2013**

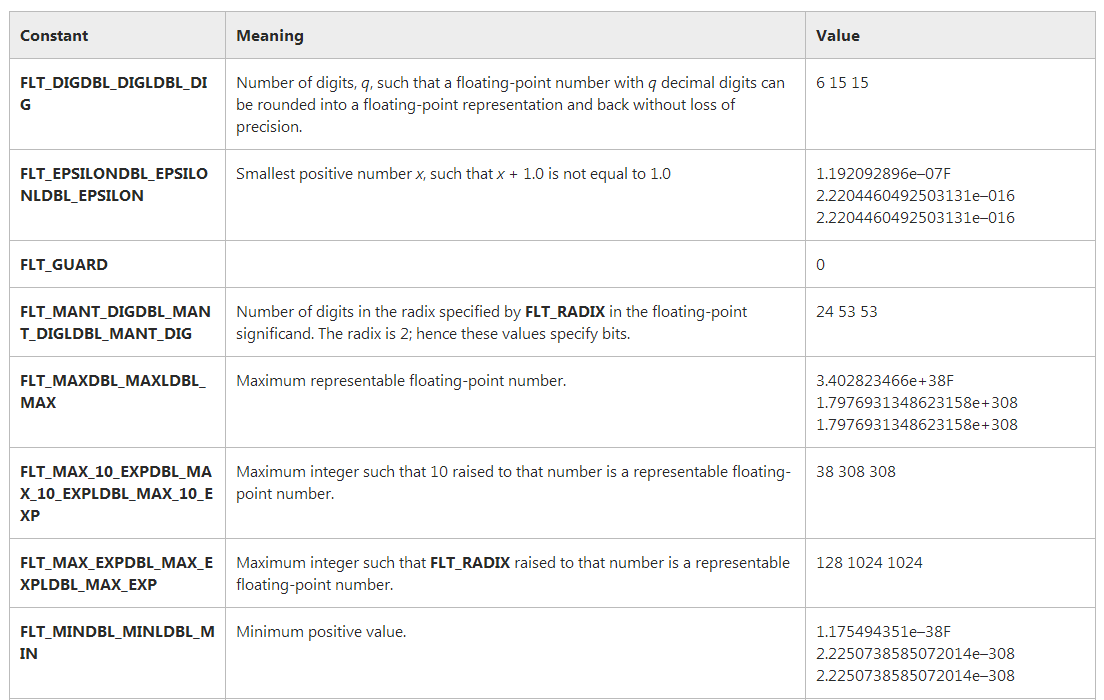
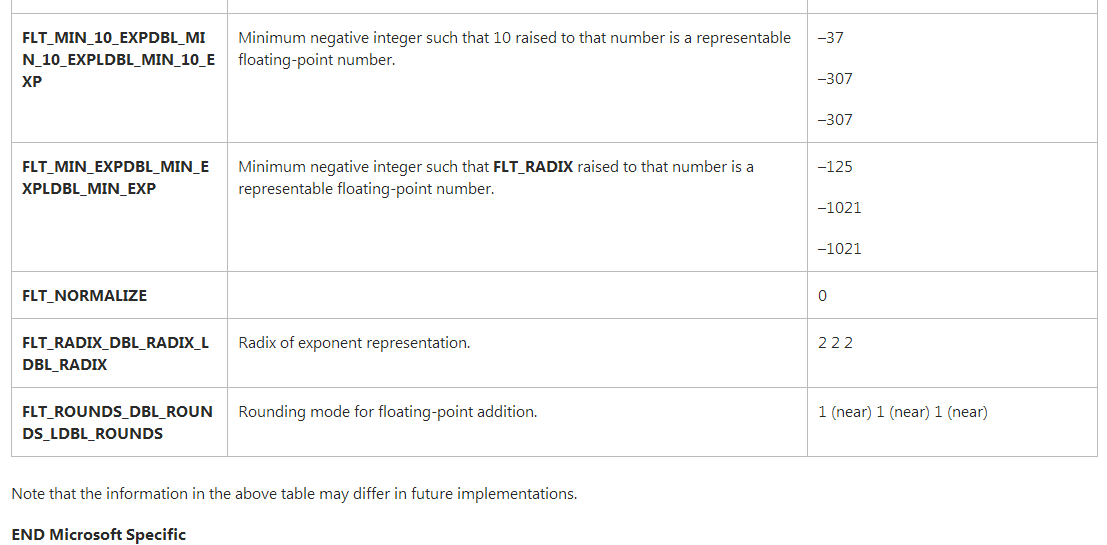
[Other Versions](javascript:;)

https://i-msdn.sec.s-msft.com/Areas/Epx/Content/Images/ImageSprite.png?v=635914614327034862

**Microsoft Specific**

Limits on the values of floating-point constants are given in the following table. The header file FLOAT.H contains this information.

Limits on Floating-Point Constants

## C Integer Constants

**Visual Studio 2013**

[Other Versions](javascript:;)

https://i-msdn.sec.s-msft.com/Areas/Epx/Content/Images/ImageSprite.png?v=635914614327034862

An "integer constant" is a decimal (base 10), octal (base 8), or hexadecimal (base 16) number that represents an integral value. Use integer constants to represent integer values that cannot be changed.

[Syntax](javascript:void(0))

*integer-constant*:

*decimal-constant integer-suffix*opt

*octal-constant integer-suffix*opt

*hexadecimal-constant integer-suffix*opt

*decimal-constant*:

*nonzero-digit*

*decimal-constant digit*

*octal-constant*:

**0**

*octal-constant octal-digit*

*hexadecimal-constant*:

**0x***hexadecimal-digit*

**0X***hexadecimal-digit*

*hexadecimal-constant hexadecimal-digit*

*nonzero-digit*: one of

**1 2 3 4 5 6 7 8 9**

*octal-digit*: one of

**0 1 2 3 4 5 6 7**

*hexadecimal-digit*: one of

**0 1 2 3 4 5 6 7 8 9**

**a b c d e f**

**A B C D E F**

*integer-suffix*:

*unsigned-suffix long-suffix*opt

*long-suffix unsigned-suffix*opt

*unsigned-suffix*: one of

**u U**

*long-suffix*: one of

**l L**

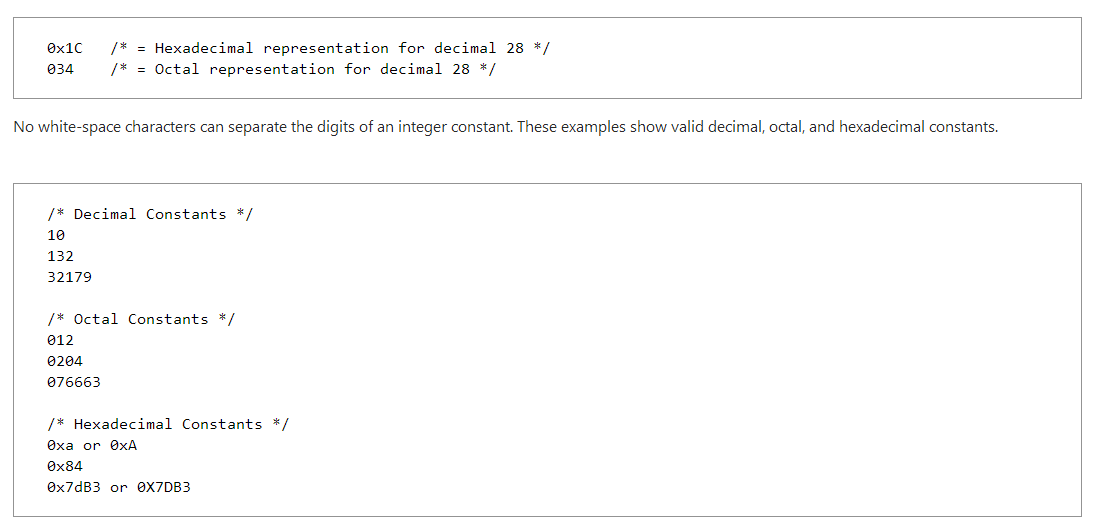
*64-bit integer-suffix*:

**i64**

Integer constants are positive unless they are preceded by a minus sign (**–**). The minus sign is interpreted as the unary arithmetic negation operator. (See[Unary Arithmetic Operators](https://msdn.microsoft.com/en-us/library/s50et82s(v=vs.120).aspx) for information about this operator.)

If an integer constant begins with **0x** or **0X**, it is hexadecimal. If it begins with the digit **0**, it is octal. Otherwise, it is assumed to be decimal.

The following lines are equivalent:



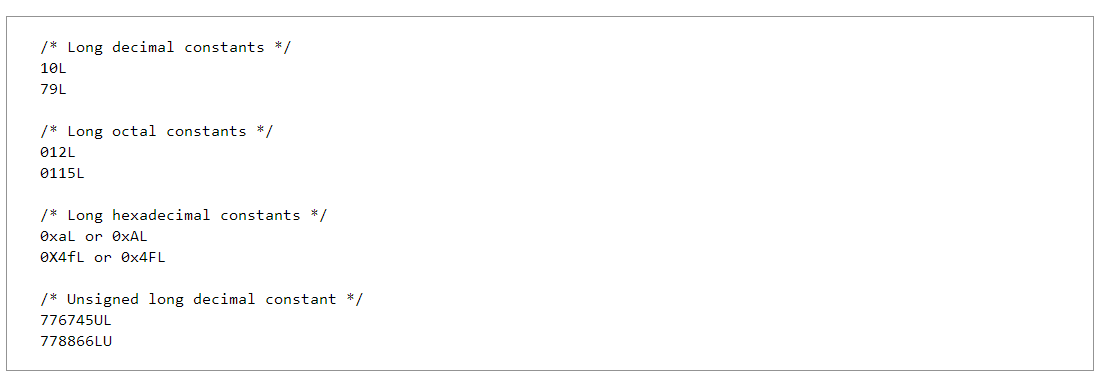
### Integer Types

**Visual Studio 2013**

[Other Versions](javascript:;)

https://i-msdn.sec.s-msft.com/Areas/Epx/Content/Images/ImageSprite.png?v=635914614327034862

Every integer constant is given a type based on its value and the way it is expressed. You can force any integer constant to type **long** by appending the letter **l**or **L** to the end of the constant; you can force it to be type **unsigned** by appending **u** or **U** to the value. The lowercase letter **l** can be confused with the digit 1 and should be avoided. Some forms of **long** integer constants follow:



The type you assign to a constant depends on the value the constant represents. A constant's value must be in the range of representable values for its type. A constant's type determines which conversions are performed when the constant is used in an expression or when the minus sign (**–**) is applied. This list summarizes the conversion rules for integer constants.

* The type for a decimal constant without a suffix is either **int**, **long int**, or **unsigned long int**. The first of these three types in which the constant's value can be represented is the type assigned to the constant.
* The type assigned to octal and hexadecimal constants without suffixes is **int**, **unsigned int**, **long int**, or **unsigned long int**depending on the size of the constant.
* The type assigned to constants with a **u** or **U** suffix is **unsigned int**or **unsigned long int**depending on their size.
* The type assigned to constants with an **l** or **L** suffix is **long int**or **unsigned long int**depending on their size.
* The type assigned to constants with a **u** or **U** and an **l** or **L** suffix is **unsigned long int**.

### C++ Integer Limits

**Visual Studio 2013**

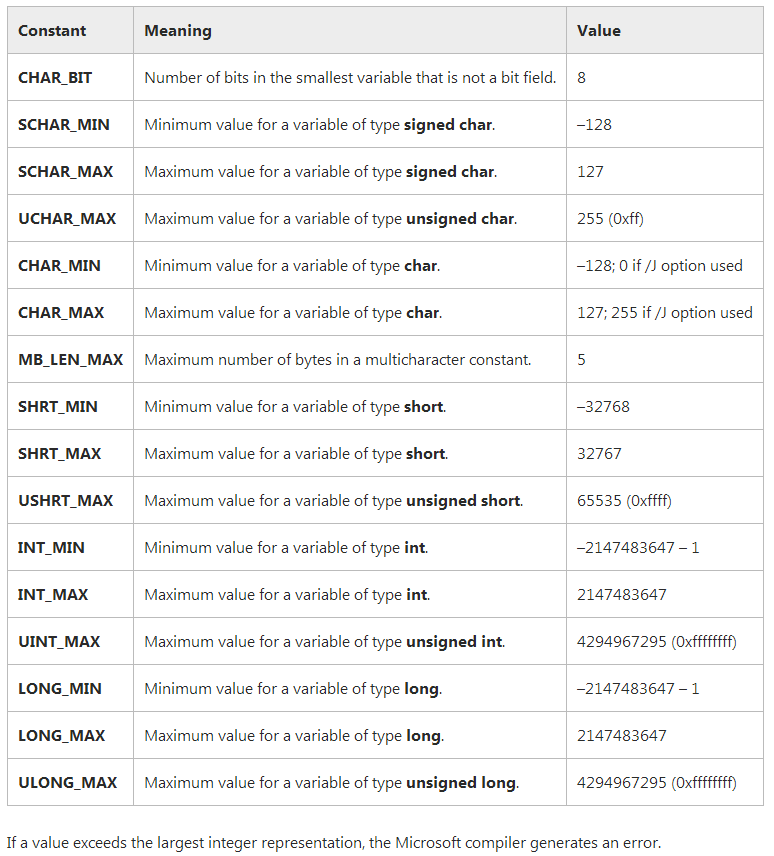
[Other Versions](javascript:;)

https://i-msdn.sec.s-msft.com/Areas/Epx/Content/Images/ImageSprite.png?v=635914614327034862

**Microsoft Specific**

The limits for integer types are listed in the following table. These limits are defined in the standard header file LIMITS.H. Microsoft C also permits the declaration of sized integer variables, which are integral types of size 8-, 16-, or 32-bits. For more information on sized integers, see [Sized Integer Types](https://msdn.microsoft.com/en-us/library/0b726ydb(v=vs.120).aspx).

Limits on Integer Constants



## C Character Constants

**Visual Studio 2013**

[Other Versions](javascript:;)

https://i-msdn.sec.s-msft.com/Areas/Epx/Content/Images/ImageSprite.png?v=635914614327034862

A "character constant" is formed by enclosing a single character from the representable character set within single quotation marks (**' '**). Character constants are used to represent characters in the [execution character set](https://msdn.microsoft.com/en-us/library/09k5ez9h(v=vs.120).aspx).

[**Syntax**](javascript:void(0))

character-constant:

**'** c-char-sequence **'**

**L'** c-char-sequence **'**

c-char-sequence:

c-char

c-char-sequence c-char

c-char:

Any member of the source character set except the single quotation mark (**'**), backslash (**\**), or newline character

escape-sequence

escape-sequence:

simple-escape-sequence

octal-escape-sequence

hexadecimal-escape-sequence

simple-escape-sequence: one of

**\a \b \f \n \r \t \v**

**\' \" \\ \?**

octal-escape-sequence:

**\**octal-digit

**\**octal-digit octal-digit

**\**octal-digit octal-digit octal-digit

hexadecimal-escape-sequence:

**\x**hexadecimal-digit

hexadecimal-escape-sequence hexadecimal-digit

### Character Types

**Visual Studio 2013**

[Other Versions](javascript:;)

https://i-msdn.sec.s-msft.com/Areas/Epx/Content/Images/ImageSprite.png?v=635914614327034862

An integer character constant not preceded by the letter **L** has type **int**. The value of an integer character constant containing a single character is the numerical value of the character interpreted as an integer. For example, the numerical value of the character a is 97 in decimal and 61 in hexadecimal.

Syntactically, a "wide-character constant" is a character constant prefixed by the letter **L**. A wide-character constant has type **wchar\_t**, an integer type defined in the STDDEF.H header file. For example:

# Wide-character constants are 16 bits wide and specify members of the extended execution character set. They allow you to express characters in alphabets that are too large to be represented by type char. See [Multibyte and Wide Characters](https://msdn.microsoft.com/en-us/library/z207t55f(v=vs.120).aspx) for more information about wide characters.

### Execution Character Set

**Visual Studio 2013**

[Other Versions](javascript:;)

https://i-msdn.sec.s-msft.com/Areas/Epx/Content/Images/ImageSprite.png?v=635914614327034862

This content often refers to the "execution character set." The execution character set is not necessarily the same as the source character set used for writing C programs. The execution character set includes all characters in the source character set as well as the null character, newline character, backspace, horizontal tab, vertical tab, carriage return, and escape sequences. The source and execution character sets may differ in other implementations.

### Escape Sequences

**Visual Studio 2013**

[Other Versions](javascript:;)

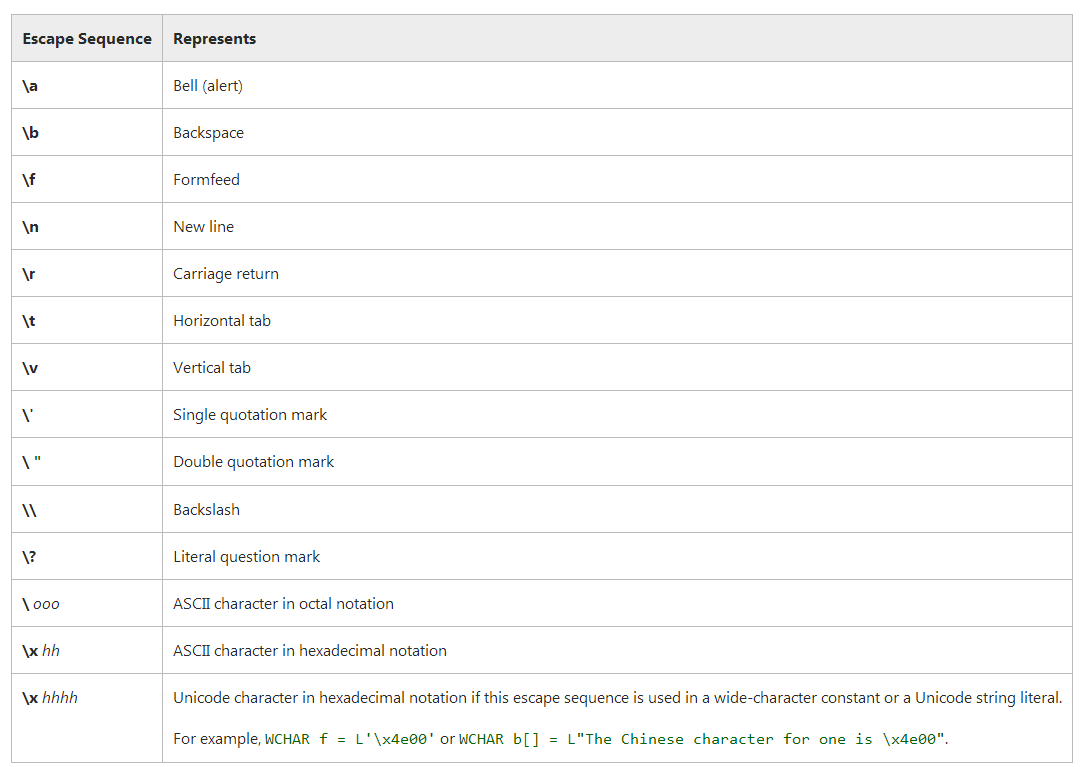
https://i-msdn.sec.s-msft.com/Areas/Epx/Content/Images/ImageSprite.png?v=635914614327034862

Character combinations consisting of a backslash (**\**) followed by a letter or by a combination of digits are called "escape sequences." To represent a newline character, single quotation mark, or certain other characters in a character constant, you must use escape sequences. An escape sequence is regarded as a single character and is therefore valid as a character constant.

Escape sequences are typically used to specify actions such as carriage returns and tab movements on terminals and printers. They are also used to provide literal representations of nonprinting characters and characters that usually have special meanings, such as the double quotation mark (**"**). The following table lists the ANSI escape sequences and what they represent.

Note that the question mark preceded by a backslash (**\?**) specifies a literal question mark in cases where the character sequence would be misinterpreted as a trigraph. See [Trigraphs](https://msdn.microsoft.com/en-us/library/bt0y4awe(v=vs.120).aspx) for more information.

Escape Sequences

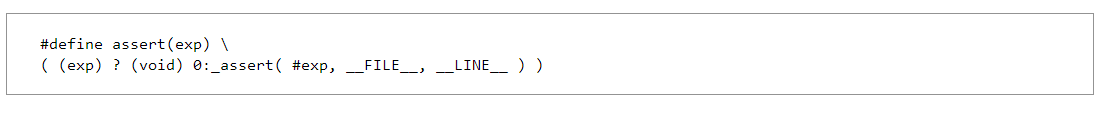
  
  
**Microsoft Specific**

If a backslash precedes a character that does not appear in the table, the compiler handles the undefined character as the character itself. For example, \c is treated as an c.

**END Microsoft Specific**

Escape sequences allow you to send nongraphic control characters to a display device. For example, the ESC character (**\033**) is often used as the first character of a control command for a terminal or printer. Some escape sequences are device-specific. For instance, the vertical-tab and formfeed escape sequences (**\v** and **\f**) do not affect screen output, but they do perform appropriate printer operations.

You can also use the backslash (**\**) as a continuation character. When a newline character (equivalent to pressing the RETURN key) immediately follows the backslash, the compiler ignores the backslash and the newline character and treats the next line as part of the previous line. This is useful primarily for preprocessor definitions longer than a single line. For example:



### Octal and Hexadecimal Character Specifications

**Visual Studio 2013**

[Other Versions](javascript:;)

https://i-msdn.sec.s-msft.com/Areas/Epx/Content/Images/ImageSprite.png?v=635914614327034862

The sequence **\**ooo means you can specify any character in the ASCII character set as a three-digit octal character code. The numerical value of the octal integer specifies the value of the desired character or wide character.

Similarly, the sequence **\x**hhh allows you to specify any ASCII character as a hexadecimal character code. For example, you can give the ASCII backspace character as the normal C escape sequence (**\b**), or you can code it as **\010** (octal) or **\x008** (hexadecimal).

You can use only the digits 0 through 7 in an octal escape sequence. Octal escape sequences can never be longer than three digits and are terminated by the first character that is not an octal digit. Although you do not need to use all three digits, you must use at least one. For example, the octal representation is **\10**for the ASCII backspace character and **\101** for the letter A, as given in an ASCII chart.

Similarly, you must use at least one digit for a hexadecimal escape sequence, but you can omit the second and third digits. Therefore you could specify the hexadecimal escape sequence for the backspace character as either **\x8**, **\x08**, or **\x008**.

The value of the octal or hexadecimal escape sequence must be in the range of representable values for type **unsigned char**for a character constant and type **wchar\_t** for a wide-character constant. See [Multibyte and Wide Characters](https://msdn.microsoft.com/en-us/library/z207t55f(v=vs.120).aspx) for information on wide-character constants.

Unlike octal escape constants, the number of hexadecimal digits in an escape sequence is unlimited. A hexadecimal escape sequence terminates at the first character that is not a hexadecimal digit. Because hexadecimal digits include the letters **a** through **f**, care must be exercised to make sure the escape sequence terminates at the intended digit. To avoid confusion, you can place octal or hexadecimal character definitions in a macro definition:

