

# User-Defined Literals: Raw and Cooked

In this lesson, we will study raw and cooked forms of user-defined literals.

## WE'LL COVER THE FOLLOWING ^

- The Four User-Defined Literals
- Raw and Cooked Forms

## The Four User-Defined Literals #

C++11 has user-defined literals for characters, C strings, integers, and floating point numbers. For integers and floating point numbers, they are available in raw and cooked form. Due to C++14, we have built-in literals for binary numbers, C++ strings, complex numbers, and time units.

To clarify, below are the raw and cooked variations of the literal types.

Data Type	Syntax	Example	Argument type	Literal Operator
Character	Character_suffix	's'_c	char	operator""_c('s')
C string	C_string_suffix	"hi"_il8n	(const char*, std::size_t)	operator""_il8n("hi",2)
Integer (Raw Form)	Integer_suffix	11_u	const char*	operator""_u("11")
Integer (Cooked Form)	Integer_suffix	11_k	unsigned long long int	operator""_u(11)
Floating point number (Raw Form)	FloatingPointNumber_suffix	1.1_lm	const char*	operator""_lm("1.1")
Floating point number (Cooked Form)	FloatingPointNumber_suffix	1.1_kd	long double	operator""_kd(1.1)

How should we read the table? The data type character has the form of character\_suffix, such as the example at 's'\_c. The compiler tries to invoke the literal operator operator""\_c('s'). The character, in this case, is of the type char. C++ also supports the data type char, the data type wchar\_t, char16\_t, and char32\_t. We can use these types as a base for our C string. In the table, we used a char. The table shows that the compiler maps the C string "hi"\_il8n to the literal operator operator""\_il8n("hi",2). The length of the C string is 2.

## Raw and Cooked Forms #

# Raw and Cooked Forms

The compiler can map integers or floating point numbers to integers (`unsigned long long int`) or floating point numbers (`long double`), but the compiler can also map them to C strings.

The first variant is called **cooked form**, and the second variant is called **raw form**. The compiler will use the raw form if the literal operator wants its arguments as C string. If not, the compiler uses the cooked form. If we implement both versions, the compiler will choose the cooked form since it has *higher priority*.

Admittedly, the last lines have a lot of potential for confusion. To solve this, we sum up all the information from the perspective of the signatures in the following table. The first column has the signature of the literal operator. The second column has the type of the user-defined literal, and the last column is an example of a user-defined literal that fits the signature of the literal operator.

Signatur of the Literal Operator	User-defined Literal	Example
(const char*)	Raw Form for integers or floating point numbers	11_s or 1.1_km
(unsigned long long int)	Cooked Form for integers	11_s
(long double)	Cooked Form for floating point numbers	1.1_km
(char)	Character literal	's'_c
(wchar_t)	Character literal	L's'_c
(char16_t)	Character literal	u's'_c
(char32_t)	Character literal	U's'_c
(const char* , std::size_t)	String literal	"hi"_i18n
(const wchar_t* , std::size_t)	String literal	L"hi"_i18n
(const char16_t* , std::size_t)	String literal	u"hi"_i18n
(const char32_t* , std::size_t)	String literal	U"hi"_i18n

Let's look at an example of this topic in the next lesson.