Raw and Cooked

Let's look at the two different types of literal operators.

WE'LL COVER THE FOLLOWING

- Raw
- Cooked
- Further information

The literal operator is available in two types:

1. Raw
2. Cooked

Raw

The raw form accepts its arguments as `(const char*, size_t)`, `(const char*)` or `const char`:

```
1.45_km => operator "" _km("1.45")
```

The raw string literals we talked about earlier fall under this category.

Cooked

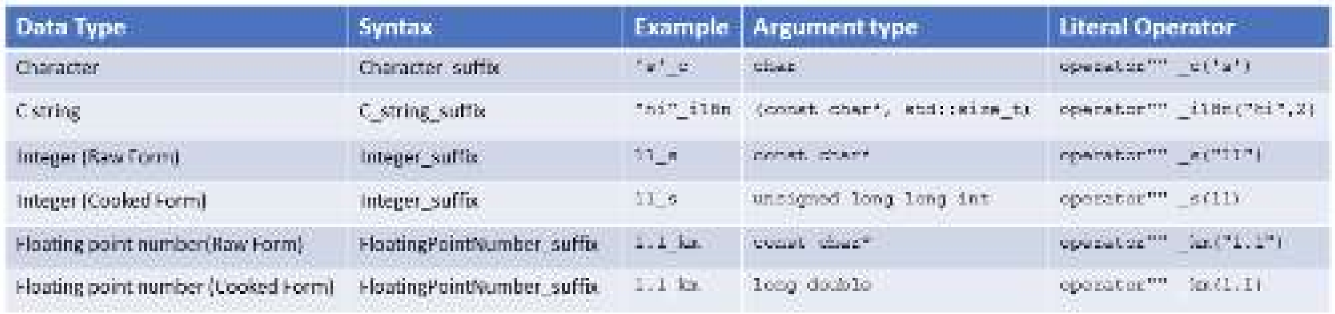
Accepts its arguments as `long double` or `unsigned long long int`:

```
1.45_km => operator "" _km(1.45)
```

> One thing to keep in mind is that there has to be a space between `""` and `_km`. Also, user-defined literals should start with an underscore (`_km`) to distinguish them from the built-in literals.

The cooked and raw forms are available for natural numbers and floating point numbers. However, only raw literal operators work with C-string literals and character literals.

Before we get into detail, here are the literal types including the raw and cooked variations:



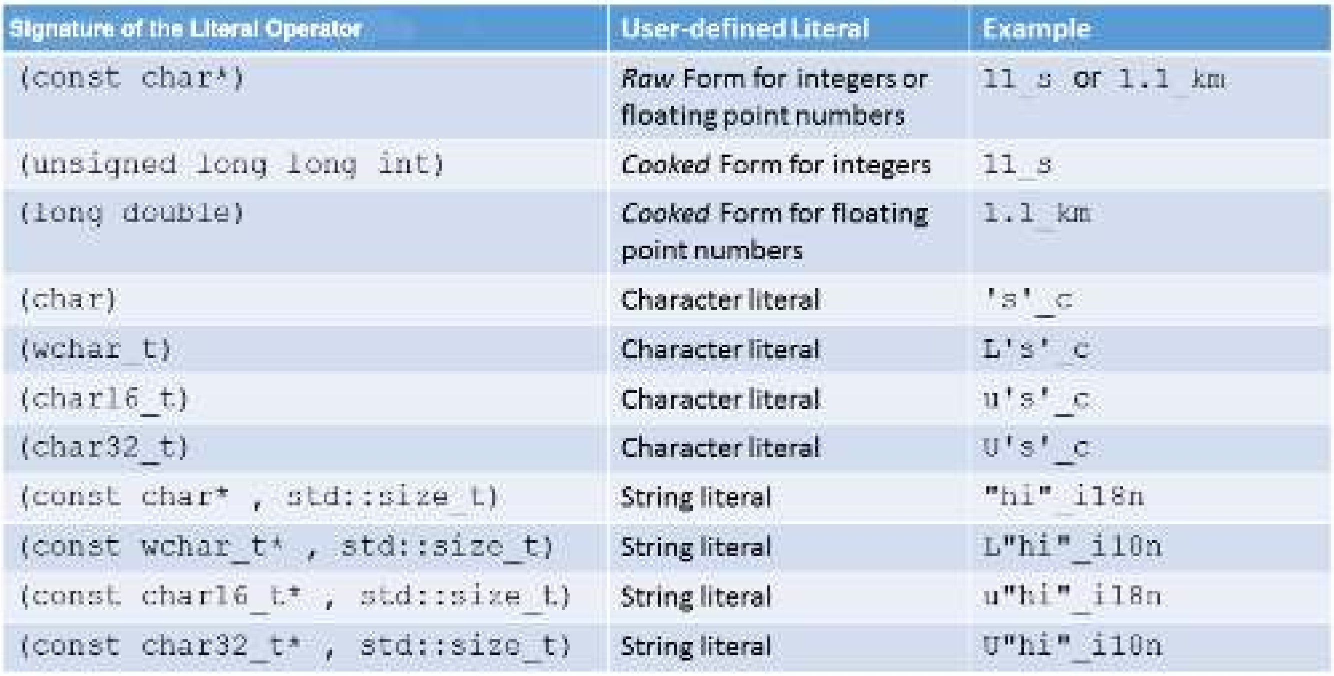
Data Type	Syntax	Example	Argument type	Literal Operator
Character	Character_suffix	'a'_c	char	operator"" _c('a')
C string	C_string_suffix	"hi"_i18n	(const char*, std::size_t)	operator"" _i18n("hi",2)
Integer (Raw Form)	Integer_suffix	11_s	const char*	operator"" _s("11")
Integer (Cooked Form)	Integer_suffix	11_s	unsigned long long int	operator"" _s(11)
Floating point number(Raw Form)	FloatingPointNumber_suffix	1.1_km	const char*	operator"" _km("1.1")
Floating point number (Cooked Form)	FloatingPointNumber_suffix	1.1_km	long double	operator"" _km(1.1)

How should we read the table? The data type `character` has the form `character_suffix`. An example is `'s'_c`. The compiler tries to invoke the literal operator `operator"" _c('s')`. The character in this case is of the type `char`.

In addition to the `char` data type, C++ supports the data types `wchar_t`, `char16_t`, and `char32_t`. We can use these types as the base for our C string. I used a `char` in the table. The table shows that the compiler maps the C string `"hi"_i18n` to the literal operator `operator"" _i18n("hi",2)`. `2` is the length of the C string.

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

Let's sum it all up from the perspective of the signatures in the following table:



Signature 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"_i10n
(const char16_t* , std::size_t)	String literal	u"hi"_i18n
(const char32_t* , std::size_t)	String literal	U"hi"_i10n

Further information

- raw string literals

Next, we'll look at examples for user-defined and built-in literals.