What are literals?

Literals are the constant values assigned to the constant variables. We can say that the literals represent the fixed values that cannot be modified. It also contains memory but does not have references as variables. For example, const int =10; is a constant integer expression in which 10 is an integer literal.

Types of literals

**There are four types of literals that exist in**[**C programming**](https://www.javatpoint.com/c-programming-language-tutorial)**:**

* **Integer literal**
* **Float literal**
* **Character literal**
* **String literal**

Integer literal

It is a numeric literal that represents only integer type values. It represents the value neither in fractional nor exponential part.

It can be specified in the following three ways:

Decimal number (base 10)

It is defined by representing the digits between 0 to 9. For example, 45, 67, etc.

Octal number (base 8)

It is defined as a number in which 0 is followed by digits such as 0,1,2,3,4,5,6,7. For example, 012, 034, 055, etc.

Hexadecimal number (base 16)

It is defined as a number in which 0x or 0X is followed by the hexadecimal digits (i.e., digits from 0 to 9, alphabetical characters from (a-z) or (A-Z)).

**An integer literal is suffixed by following two sign qualifiers:**

**L or l:** It is a size qualifier that specifies the size of the integer type as long.

**U or u:** It is a sign qualifier that represents the type of the integer as unsigned. An unsigned qualifier contains only positive values.

Note: The order of the qualifier is not considered, i.e., both lu and ul are the same.

**Let's look at a simple example of integer literal.**

1. #include <stdio.h>
2. **int** main()
3. {
4. **const** **int** a=23;  // constant integer literal
5. printf("Integer literal : %d", a);
6. **return** 0;
7. }

**Output**

Integer literal : 23

Float literal

It is a literal that contains only floating-point values or real numbers. These real numbers contain the number of parts such as integer part, real part, exponential part, and fractional part. The floating-point literal must be specified either in decimal or in exponential form. Let's understand these forms in brief.

Decimal form

The decimal form must contain either decimal point, exponential part, or both. If it does not contain either of these, then the compiler will throw an error. The decimal notation can be prefixed either by '+' or '-' symbol that specifies the positive and negative numbers.

**Examples of float literal in decimal form are:**

1. 1.2, +9.0, -4.5

**Let's see a simple example of float literal in decimal form.**

1. #include <stdio.h>
2. **int** main()
3. {
4. **const** **float** a=4.5; // constant float literal
5. **const** **float** b=5.6; // constant float literal
6. **float** sum;
7. sum=a+b;
8. printf("%f", sum);
9. **return** 0;
10. }

**Output**

10.100000

Exponential form

The exponential form is useful when we want to represent the number, which is having a big magnitude. It contains two parts, i.e., mantissa and exponent. For example, the number is 2340000000000, and it can be expressed as 2.34e12 in an exponential form.

**Syntax of float literal in exponential form**

1. [+/-] <Mantissa> <e/E> [+/-] <Exponent>

Examples of real literal in exponential notation are:

1. +1e23, -9e2, +2e-25

**Rules for creating an exponential notation**

**The following are the rules for creating a float literal in exponential notation:**

* In exponential notation, the mantissa can be specified either in decimal or fractional form.
* An exponent can be written in both uppercase and lowercase, i.e., e and E.
* We can use both the signs, i.e., positive and negative, before the mantissa and exponent.
* Spaces are not allowed

Character literal

A character literal contains a single character enclosed within single quotes. If multiple characters are assigned to the variable, then we need to create a character array. If we try to store more than one character in a variable, then the warning of a **multi-character character constant** will be generated. Let's observe this scenario through an example.

1. #include <stdio.h>
2. **int** main()
3. {
4. **const** **char** c='ak';
5. printf("%c",c);
6. **return** 0;
7. }

In the above code, we have used two characters, i.e., 'ak', within single quotes. So, this statement will generate a warning as shown below.

**Warning generated:**

1. main.c:6:18: warning: multi-character character constant
2. [-Wmultichar]
3. **const** **char** c='ak';
4. main.c:6:18: warning: implicit conversion from 'int' to 'char'
5. changes value from 24939 to 107 [-Wconstant-conversion]
6. **const** **char** c='ak';
7. ~ ^~~~
8. 2 warnings generated.
9. ? ./main

**Representation of character literal**

**A character literal can be represented in the following ways:**

* It can be represented by specifying a single character within single quotes. For example, 'a', 'b', etc.
* We can specify the escape sequence character within single quotes to represent a character literal. For example, '\n', '\a', '\b'.
* We can also use the ASCII in integer to represent a character literal. For example, the ascii value of 65 is 'A'.
* The octal and hexadecimal notation can be used as an escape sequence to represent a character literal. For example, '\023', '\0x12'.

String literal

A string literal represents multiple characters enclosed within double-quotes. It contains an additional character, i.e., '\0' (null character), which gets automatically inserted. This null character specifies the termination of the string. We can use the '+' symbol to concatenate two strings.

For example,

String1= "javatpoint";

String2= "family";

To concatenate the above two strings, we use '+' operator, as shown in the below statement:

"javatpoint " + "family"= javatpoint family

Note: If we represent a single character, i.e., 'b', then this character will occupy a single byte as it is a character literal. And, if we represent the character within double quotes "b" then it will occupy more bytes as it is a string literal.