# Integer and Floating Point Literals

This lesson will cover integer and floating point literals and their syntax rules.

WE'LL COVER THE FOLLOWING

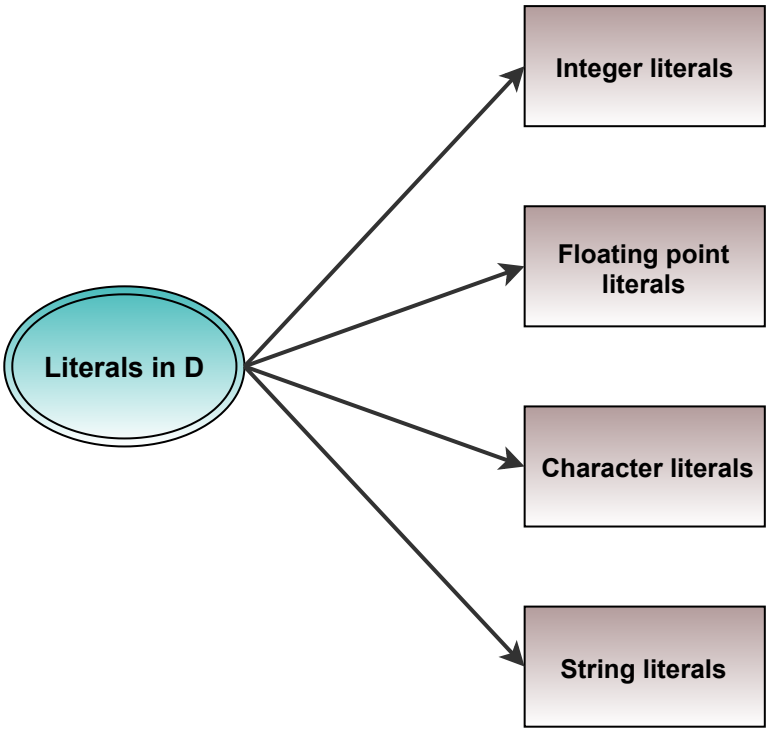


## Literals #

Programs achieve their tasks by manipulating the values of variables. They produce new values by using them with functions and operators.
Some values, however, need not be modified during the execution of the program. Instead, they are used as-is in the source code. For example, the floating point value 0.75 and the string value "Total price: " below are not calculated by the program:

```
discountedPrice = actualPrice * 0.75;
totalPrice += count * discountedPrice;
writeln("Total price: ", totalPrice);
```

Such values, which directly typed into the source code, are called **literals**. We have used string literals in the programs so far. Now, we will see other types of literals.




Literals in D

## Integer literals #

Integer literals can be written in one of four ways:

- **decimal system**, which we use in our daily lives
- **hexadecimal and binary systems**, which are more suitable for certain computing tasks
- **octal system**, which may be needed in very rare cases

In order to make the code more readable, it is possible to insert `_` characters anywhere after the first digit of integer literals. For example, we can use `_` to form groups of three digits, as in 1_234_567. Another example would be using it to separate some currency values in cents; _ separates the currency units from the cents, as in 199_99. These characters are optional; they are ignored by the compiler.

- **Decimal system:** Integer literals are specified using the decimal system in exactly the same way as numerals are usually used e.g., 12, 456, etc.

  When using the decimal system in D to write literals, the first digit cannot be 0. Such a leading zero digit is often used in other programming languages to indicate the octal system, so this constraint in D helps to

languages to indicate the octal system, so this constraint in D helps to prevent bugs that are caused by this easily overlooked difference. This does not preclude 0 on its own: 0 is zero.

- **Hexadecimal system:** These literals start with 0x or 0X and include the numerals of the hexadecimal system: “0123456789abcdef” and “ABCDEF,” as in 0x12ab00fe.
- **Octal system:** These literals are specified using the octal template from the `std.conv` module and include the numerals of the octal system: “01234567” as in octal!576.
- **Binary system:** These literals start with 0b or 0B and include the numerals of the binary system: 0 and 1, as in 0b01100011.

## The types of integer literals #

Just like every variable used in a program must have a type, literals also have a certain type. The types of literals are not specified explicitly as `int`, `double`, etc, the compiler infers the type from the value and syntax of the literal itself.

Although most of the time the types of literals are not important, sometimes the types may not match the expressions that they are used in. In such cases, the type must be *explicitly* specified.

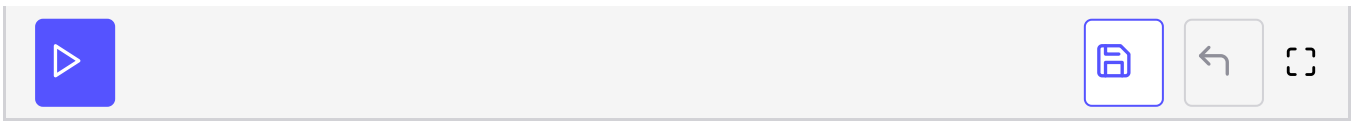
By default, integer literals are inferred to be of type `int`. When the value happens to be too large to fit an `int`, the compiler uses the following logic to decide the type of the literal:

- If the value of the literal does not fit an `int` and it is specified in the decimal system, then its type is `long`.
- If the value of the literal does not fit an `int` and it is specified in any other system, then its type becomes the first of the following types whichever can accommodate the literal’s value: `uint`, `long` and `ulong`.

To see this logic in action, let’s try the following program which takes advantage of `typeof` and `stringof`:

```
import std.stdio;
```

```
void main() {
    writeln("\n--- these are written in decimal ---");

    // fits an int, so the type is int
    writeln( 2_147_483_647, "\t\t", typeof(2_147_483_647).stringof);

    // does not fit an int and is decimal, so the type is long
    writeln( 2_147_483_648, "\t\t", typeof(2_147_483_648).stringof);

    writeln("\n--- these are NOT written in decimal ---");

    // fits an int, so the type is int
    writeln( 0x7FFF_FFFF, "\t\t", typeof(0x7FFF_FFFF).stringof);

    // does not fit an int and is not decimal, so the type is uint
    writeln( 0x8000_0000, "\t\t", typeof(0x8000_0000).stringof);

    // does not fit a uint and is not decimal, so the type is long
    writeln( 0x1_0000_0000, "\t\t", typeof(0x1_0000_0000).stringof);

    // does not fit a long and is not decimal, so the type is ulong
    writeln( 0x8000_0000_0000_0000, "\t", typeof(0x8000_0000_0000_0000).stringof);
}
```



Literals logic

### The L suffix #

Regardless of the magnitude of the value, if it ends with L as in 10L, the type is `long`.

### The U suffix #

If the literal ends with U as in 10U, then its type is an *unsigned type*. It depends on the value of the literal. Lowercase "u" can also be used.

## The floating point literals #

Floating point literals can be specified in either the decimal system, as in 1.234, or in the hexadecimal system, as in 0x9a.bc.

- **In the decimal system:** An exponent may be appended after the character `e` or `E`, meaning "*times 10 to the power of*." For example, 3.4e5 means "$3.5x10^5$," or 340000. The `-` character typed before the value of the exponent changes the meaning to be "*divided by 10 to the power of*." For example, 7.8e-3 means "*7.8 divided by 10 to the power of 3*." A `+` character may also be specified before the value of the exponent, but it

has no effect. For example, 5.6e2 and 5.6e+2 are the same.

- **In the hexadecimal system:** The value starts with either 0x or 0X and the parts before and after the point are specified in the numerals of the hexadecimal system. Since `e` and `E` are valid numerals in this system, the exponent is specified by `p` or `P`. Another difference is that the exponent does not mean "*10 to the power of*," but instead "*2 to the power of*." For example, the **P4** part in **0xabc.defP4** means "*2 to the power of 4*."

Floating point literals almost always have a point but it may be omitted if an exponent is specified. For example, 2e3 is a floating point literal with the value 2000. The value before the point may be omitted if zero. For example, .25 is a literal having the value "quarter."

The optional `_` characters may be used with floating point literals as well, as in 1_000.5.

## The types of floating point literals #

Unless explicitly specified, the type of a floating point literal is `double`. The `f` and `F` specifiers mean `float`, and the `L` specifier means real. For example, 1.2 is double, 3.4f is `float` and 5.6L is `real`.

---

In the next lesson, you will find a coding challenge related to the use of literals.