Fundamental Types

This lesson explores in detail the fundamental types we have in D language and their properties.

WE'LL COVER THE FOLLOWING

The need for a data type

The smallest unit of data in a computer is called a **bit**. The value of a bit can be either **0** or **1**. Since a type of data that can hold only the values 0 and 1 would have very limited use, the CPU supports larger data types that are combinations of more than one bit. As an example, a byte usually consists of 8 bits. If an N-bit data type is the most efficient data type supported by a CPU, we consider it to be an N-bit CPU: as in 32-bit CPU, 64-bit CPU, etc.

The data types that the CPU supports are still not sufficient because they can't represent higher-level concepts like the *name of a student* or a *playing card*. Likewise, D's fundamental data types are not sufficient to represent many higher-level concepts because of the various requirements like memory, specific operations, etc. Such concepts must be defined by the programmer as *structs* and *classes*.

D's fundamental types

D is statically typed, meaning every expression has a type. D's fundamental types are very similar to the fundamental types of many other languages, as

seen in the following table.

D's Fundamental Data Types

Type	Definition	Initial Value
bool	Boolean type	false
byte	signed 8 bits	0
ubyte	unsigned 8 bits	0
short	signed 16 bits	0
ushort	unsigned 16 bits	0
int	signed 32 bits	0
uint	unsigned 32 bits	0
long	signed 64 bits	0L
ulong	unsigned 64 bits	0L
float	32-bit floating point	float.nan
double	64-bit floating point	double.nan
real	either the largest floating point type that the hardware supports, or double; whichever is larger	real.nan
ifloat	imaginary value type of float	float.nan * 1.0i
idouble	imaginary value type of double	double.nan * 1.0i
ireal	imaginary value type of real	real.nan * 1.0i
cfloat	complex number type made of two floats	float.nan + float.nan * 1.0i
cdouble	complex number type made of two doubles	double.nan + double.nan * 1.0i
creal	complex number type made of two reals	real.nan + real.nan * 1.0i
char	UTF-8 code unit	0xFF
wchar	UTF-16 code unit	0xFFFF
dchar	UTF-32 code unit and Unicode code point	0x0000FFFF

D's fundamental data types

> **Note:** In addition to the above, the keyword `void` represents having *no type*. The keywords `cent` and `ucent` are reserved for future use to represent signed and unsigned 128-bit values.

Whole numbers can be represented by `int`. To represent concepts that can have fractional values, consider `double`.

The following terms appear in the table above:

- **Boolean**: The type of logical expressions that has the value `true` for *truth* and `false` for *falsity*.
- **Signed type**: A type that can have both *negative* and *positive* values. For example, a byte can have values from -128 to 127. The names of the signed types come from the negative sign.
- **Unsigned type**: A type that can have only *positive* values. For example

- **Unsigned type**: A type that can have only *positive* values. For example, `ubyte` can have values from 0 to 255. The "u" at the beginning of the name of these types comes from unsigned.

- **Floating point**: The type that can represent values with decimal points as in 1.25. The precision of floating point calculations is directly related to the bit count of the type: the higher the bit count, the more precise the results are. Only *floating point* types can represent fractions; integer types like `int` can only represent whole values like 1 and 2.

- **Complex number type**: A type that can represent the complex numbers of mathematics having the form "a + bi".

- **Imaginary number type**: The type that represents **only** the *imaginary part* of complex numbers. The "i" that appears against this type in the "Initial Value" column of the table is called **iota**, which is equal to the square root of -1 in mathematics.

- **NaN**: Short for "not a number," represents an invalid floating- point value.

Properties of types

D types have properties. Properties are accessed with a dot after the name of the type. For example, the `sizeof` property of `int` is accessed as `int.sizeof`. We will explore some of the properties of types in this chapter:

- `.stringof` is the *name* of the type.

- `.sizeof` is the *length* of the type in terms of bytes. (In order to determine the bit count, this value must be multiplied by 8, the number of bits in a byte.)

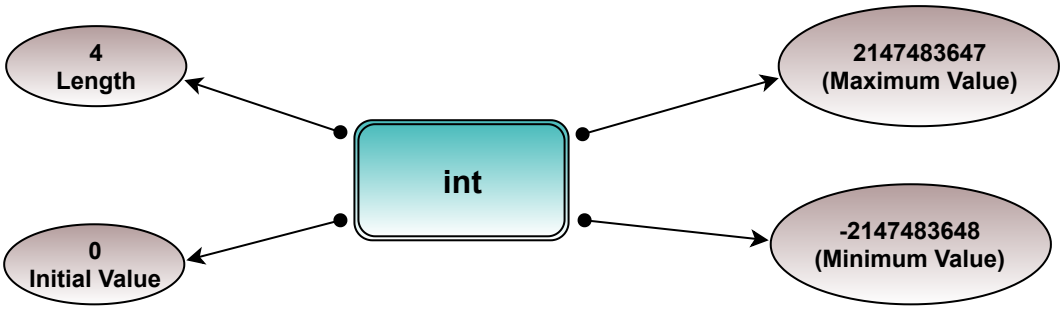
- `.min` is short for "*minimum*"; this is the smallest value that the type can have.

- `.max` is short for "*maximum*"; this is the largest value that the type can have.

- `.init` is short for "*initial value*" (default value); this is the value that D assigns to a type when an initial value is not specified.

Example: `int`

As an example, let's explore the properties of type `int`.



Properties of int

Here is a program that displays these properties for `int`:

```
import std.stdio;

void main() {
    writeln("Length in bytes: ", int.sizeof);
    writeln("Minimum value : ", int.min);
    writeln("Maximum value : ", int.max);
    writeln("Initial value : ", int.init);
}
```

Code to find properties of int

size_t

You will occasionally come across the `size_t` type, as well. `size_t` is not a separate type but an alias of an existing unsigned type. Its name comes from "size type." It is the most suitable type to represent concepts like the size of variables or count of array elements etc.

`size_t` is large enough to represent the number of bytes of memory that a program can potentially use. Its actual size depends on the system: `uint` on a 32-bit system and `ulong` on a 64-bit system. For this reason, `ulong` is larger than `size_t` on a 32-bit system.

You can use the `.stringof` property to see what `size_t` is an alias of on your system.

> **Note**: You can't use the reserved types `cent` and `ucent` in any program, and as an exception, `void` does not have the properties `.min`, `.max` and

`.init`.

Additionally, the `.min` property is deprecated for floating point types. If you use a floating point type, you would be warned by the compiler that `.min` is not valid for that type. Instead, as we will see later in the floating point types lesson, you must use the negative of the `.max` property e.g., as `-double.max`.

Quick quiz!

Try using data types in the code provided earlier in this lesson and then attempt the following quiz.

1 The length in bytes of `char` ____.

2 The initial value of `bool` is ____.

3 The minimum value of `short` is ____.

4 `.min` property can be used to find theinital value of a data type.

5 `size_t` is a data type.

Check Answers

In the next lesson, we will see how to assign values to variables.