**Fundamental data types.**

The values of variables are stored somewhere in an unspecified location in the computer memory as zeros and ones. Our program does not need to know the exact location where a variable is stored; it can simply refer to it by its name. What the program needs to be aware of is the kind of data stored in the variable. It's not the same to store a simple integer as it is to store a letter or a large floating-point number; even though they are all represented using zeros and ones, they are not interpreted in the same way, and in many cases, they don't occupy the same amount of memory.

Fundamental data types are basic types implemented directly by the language that represent the basic storage units supported natively by most systems. They can mainly be classified into:

* **Character types:** They can represent a single character, such as 'A' or '$'. The most basic type is char, which is a one-byte character. Other types are also provided for wider characters.
* **Numerical integer types:** They can store a whole number value, such as 7 or 1024. They exist in a variety of sizes, and can either be *signed* or *unsigned*, depending on whether they support negative values or not.
* **Floating-point types:** They can represent real values, such as 3.14 or 0.01, with different levels of precision, depending on which of the three floating-point types is used.
* **Boolean type:** The *boolean* type, known in C++ as bool, can only represent one of two states, true or false.

Here is the complete list of fundamental types in C++:

|  |  |  |
| --- | --- | --- |
| **Group** | **Type names\*** | **Notes on size / precision** |
| Character types | **char** | Exactly one byte in size. At least 8 bits. |
| **char16\_t** | Not smaller than char. At least 16 bits. |
| **char32\_t** | Not smaller than char16\_t. At least 32 bits. |
| **wchar\_t** | Can represent the largest supported character set. |
| Integer types (signed) | **signed char** | Same size as char. At least 8 bits. |
| *signed* **short** *int* | Not smaller than char. At least 16 bits. |
| *signed* **int** | Not smaller than short. At least 16 bits. |
| *signed* **long** *int* | Not smaller than int. At least 32 bits. |
| *signed* **long long** *int* | Not smaller than long. At least 64 bits. |
| Integer types (unsigned) | **unsigned char** | (same size as their signed counterparts) |
| **unsigned short** *int* |
| **unsigned** *int* |
| **unsigned long** *int* |
| **unsigned long long** *int* |
| Floating-point types | **float** |  |
| **double** | Precision not less than float |
| **long double** | Precision not less than double |
| Boolean type | **bool** |  |
| Void type | **void** | no storage |
| Null pointer | **decltype(nullptr)** |  |

\* The names of certain integer types can be abbreviated without their *signed* and *int* components - only the part not in italics is required to identify the type, the part in italics is optional. I.e., *signed* *short* *int* can be abbreviated as *signed short*, *short int*, or simply *short*; they all identify the same fundamental type.

Within each of the groups above, the difference between types is only their size (i.e., how much they occupy in memory): the first type in each group is the smallest, and the last is the largest, with each type being at least as large as the one preceding it in the same group. Other than that, the types in a group have the same properties.

Note in the panel above that other than *char* (which has a size of exactly one byte), none of the fundamental types has a standard size specified (but a minimum size, at most). Therefore, the type is not required (and in many cases is not) exactly this minimum size. This does not mean that these types are of an undetermined size, but that there is no standard size across all compilers and machines; each compiler implementation may specify the sizes for these types that fit the best the architecture where the program is going to run. This rather generic size specification for types gives the C++ language a lot of flexibility to be adapted to work optimally in all kinds of platforms, both present and future.

Type sizes above are expressed in bits; the more bits a type has, the more distinct values it can represent, but at the same time, also consumes more space in memory:

|  |  |  |
| --- | --- | --- |
| **Size** | **Unique representable values** | **Notes** |
| 8-bit | 256 | = 28 |
| 16-bit | 65 536 | = 216 |
| 32-bit | 4 294 967 296 | = 232 (~4 billion) |
| 64-bit | 18 446 744 073 709 551 616 | = 264 (~18 billion billion) |

For integer types, having more representable values means that the range of values they can represent is greater; for example, a 16-bit unsigned integer would be able to represent 65536 distinct values in the range 0 to 65535, while its signed counterpart would be able to represent, on most cases, values between -32768 and 32767. Note that the range of positive values is approximately halved in signed types compared to unsigned types, due to the fact that one of the 16 bits is used for the sign; this is a relatively modest difference in range, and seldom justifies the use of unsigned types based purely on the range of positive values they can represent.

For floating-point types, the size affects their precision, by having more or less bits for their significant and exponent.

If the size or precision of the type is not a concern, then char, int, and double are typically selected to represent characters, integers, and floating-point values, respectively. The other types in their respective groups are only used in very particular cases.

The properties of fundamental types in a particular system and compiler implementation can be obtained by using the [numeric\_limits](https://cplusplus.com/numeric_limits) classes (see standard header [<limits>](https://cplusplus.com/%3Climits%3E)). If for some reason, types of specific sizes are needed, the library defines certain fixed-size type aliases in header [<cstdint>](https://cplusplus.com/%3Ccstdint%3E).

The types described above (characters, integers, floating-point, and boolean) are collectively known as arithmetic types. But two additional fundamental types exist: void, which identifies the lack of type; and the type *nullptr*, which is a special type of pointer. Both types will be discussed further in a coming chapter about pointers.

C++ supports a wide variety of types based on the fundamental types discussed above; these other types are known as *compound data types*, and are one of the main strengths of the C++ language. We will also see them in more detail in future chapters.