

Primitive data type

In [computer science](#), **primitive data type** is either of the following:

- a *basic type* is a [data type](#) provided by a [programming language](#) as a basic building block. Most languages allow more complicated [composite types](#) to be recursively constructed starting from basic types.
- a *built-in type* is a data type for which the programming language provides built-in support.

In most programming languages, all basic data types are built-in. In addition, many languages also provide a set of composite data types.

Depending on the language and its implementation, primitive data types may or may not have a one-to-one correspondence with objects in the computer's memory. However, one usually expects operations on basic primitive data types to be the fastest language constructs there are. Integer addition, for example, can be performed as a single machine instruction, and some [processors](#) offer specific instructions to process sequences of characters with a single instruction. In particular, the [C](#) standard mentions that "a 'plain' int object has the natural size suggested by the architecture of the execution environment." This means that `int` is likely to be 32 bits long on a 32-bit architecture. Basic primitive types are almost always [value types](#).

Most languages do not allow the behavior or capabilities of primitive (either built-in or basic) data types to be modified by programs. Exceptions include [Smalltalk](#), which permits all data types to be extended within a program, adding to the operations that can be performed on them or even redefining the built-in operations.

Contents

[Overview](#)

[Specific primitive data types](#)

[Integer numbers](#)

[Floating-point numbers](#)

[Fixed-point numbers](#)

[Booleans](#)

[Characters and strings](#)

[Numeric data type ranges](#)

[See also](#)

[References](#)

[External links](#)

Overview

The actual range of primitive data types that is available is dependent upon the specific programming language that is being used. For example, in [C#](#), [strings](#) are a composite but built-in data type, whereas in modern dialects of [BASIC](#) and in [JavaScript](#), they are assimilated to a primitive data type that is both basic and built-in.

^[1] ^[2]

Classic basic primitive types may include:

- Character (character, char);
- Integer (integer, int, short, long, byte) with a variety of precisions;
- Floating-point number (float, double, real, double precision);
- Fixed-point number (fixed) with a variety of precisions and a programmer-selected scale.
- Boolean, logical values **true** and **false**.
- Reference (also called a *pointer* or *handle* or *descriptor*), a value referring to another object. The reference can be a memory address, or an index to a collection of values.

The above primitives are generally supported more or less directly by computer hardware, except possibly for floating point, so operations on such primitives are usually fairly efficient. Some programming languages support text strings as a primitive (e.g. BASIC) while others treat a text string as an array of characters (e.g. C). Some computer hardware (e.g. x86) has instructions which help in dealing with text strings, but complete hardware support for text strings is rare.

Strings could be any series of characters in the used encoding. To separate strings from code, most languages enclose them by single or double quotes. For example "Hello World" or 'Hello World'. Note that "200" could be mistaken for an integer type but is actually a string type because it is contained in double quotes.

More sophisticated types which can be built-in include:

- Tuple in Standard ML, Python, Scala, Swift, Elixir
- List in Common Lisp, Python, Scheme, Haskell
- Complex number in C99, Fortran, Common Lisp, Python, D, Go
- Rational number in Common Lisp, Haskell
- Associative array in Perl, Python, Ruby, JavaScript, Lua, D, Go
- First-class function, in all functional languages, JavaScript, Lua, D, Go, and in newer standards of C++, Java, C#, Perl

Specific primitive data types

Integer numbers

An integer data type represents some range of mathematical integers. Integers may be either signed (allowing negative values) or unsigned (non-negative integers only). Common ranges are:

Size (bytes)	Size (bits)	Names	Signed range (assuming two's complement for signed)	Unsigned range
1 byte	8 bits	Byte, octet, minimum size of char in C99(see limits.h CHAR_BIT)	−128 to +127	0 to 255
2 bytes	16 bits	x86 word, minimum size of short and int in C	−32,768 to +32,767	0 to 65,535
4 bytes	32 bits	x86 double word, minimum size of long in C, actual size of int for most modern C compilers, ^[3] pointer for IA-32-compatible processors	−2,147,483,648 to +2,147,483,647	0 to 4,294,967,295
8 bytes	64 bits	x86 quadruple word, minimum size of long long in C, actual size of long for most modern C compilers, ^[3] pointer for x86-64-compatible processors	−9,223,372,036,854,775,808 to +9,223,372,036,854,775,807	0 to 18,446,744,073,709,551,615
unlimited/8	unlimited	Bignum	−2 ^{unlimited} /2 to +(2 ^{unlimited} /2 − 1)	0 to 2 ^{unlimited} − 1

Literals for integers can be written as regular [Arabic numerals](#), consisting of a sequence of digits and with negation indicated by a [minus sign](#) before the value. However, most programming languages disallow use of commas or spaces for [digit grouping](#). Examples of integer literals are:

- 42
- 10000
- -233000

There are several alternate methods for writing integer literals in many programming languages:

- Most programming languages, especially those influenced by C, prefix an integer literal with 0X or 0x to represent a [hexadecimal](#) value, e.g. 0xDEADBEEF. Other languages may use a different notation, e.g. some [assembly languages](#) append an H or h to the end of a hexadecimal value.
- Perl, Ruby, Java, Julia, D, Rust and Python (starting from version 3.6) allow embedded [underscores](#) for clarity, e.g. 10_000_000, and fixed-form Fortran ignores embedded spaces in integer literals.
- In C and C++, a leading zero indicates an octal value, e.g. 0755. This was primarily intended to be used with [Unix modes](#); however, it has been criticized because normal integers may also lead with zero.^[4] As such, Python, Ruby, Haskell, and OCaml prefix octal values with 0O or 0o, following the layout used by hexadecimal values.
- Several languages, including Java, C#, Scala, Python, Ruby, and OCaml, can represent binary values by prefixing a number with 0B or 0b.

Floating-point numbers

A [floating-point](#) number represents a limited-precision [rational number](#) that may have a fractional part. These numbers are stored internally in a format equivalent to [scientific notation](#), typically in [binary](#) but sometimes in [decimal](#). Because floating-point numbers have limited precision, only a subset of [real](#) or [rational](#) numbers are

exactly representable; other numbers can be represented only approximately.

Many languages have both a single precision (often called "float") and a double precision type.

Literals for floating point numbers include a decimal point, and typically use e or E to denote scientific notation. Examples of floating-point literals are:

- 20.0005
- 99.9
- -5000.12
- 6.02e23

Some languages (e.g., Fortran, Python, D) also have a complex number type comprising two floating-point numbers: a real part and an imaginary part.

Fixed-point numbers

A fixed-point number represents a limited-precision rational number that may have a fractional part. These numbers are stored internally in a scaled-integer form, typically in binary but sometimes in decimal. Because fixed-point numbers have limited precision, only a subset of real or rational numbers are exactly representable; other numbers can be represented only approximately. Fixed-point numbers also tend to have a more limited range of values than floating point, and so the programmer must be careful to avoid overflow in intermediate calculations as well as the final result.

Booleans

A boolean type, typically denoted "bool" or "boolean", is typically a *logical type* that can have either the value "true" or the value "false". Although only one bit is necessary to accommodate the value set "true" and "false", programming languages typically implement boolean types as one or more bytes.

Many languages (e.g. Java, Pascal and Ada) implement booleans adhering to the concept of *boolean* as a distinct logical type. Languages, though, may implicitly convert booleans to *numeric types* at times to give extended semantics to booleans and boolean expressions or to achieve backwards compatibility with earlier versions of the language. For example, early versions of the C programming language that followed ANSI C and its former standards did not have a dedicated boolean type. Instead, numeric values of zero are interpreted as "false", and any other value is interpreted as "true".^[5] The newer C99 added a distinct boolean type that can be included with stdbool.h,^[6] and C++ supports `bool` as a built-in type and "true" and "false" as reserved words.^[7]

Characters and strings

A character type (typically called "char") may contain a single letter, digit, punctuation mark, symbol, formatting code, control code, or some other specialized code (e.g., a byte order mark). In C, `char` is defined as the smallest addressable unit of memory. On most systems, this is 8 bits; Several standards, such as POSIX, require it to be this size. Some languages have two or more character types, for example a single-byte type for ASCII characters and a multi-byte type for Unicode characters. The term "character type" is normally used even for types whose values more precisely represent code units, for example a UTF-16 code unit as in Java (support limited to 16-bit characters only ^[8]) and JavaScript.

Characters may be combined into strings. The string data can include numbers and other numerical symbols but is treated as text. For example, the mathematical operations that can be performed on a numerical value (e.g. 200) generally cannot be performed on that same value written as a string (e.g. "200").

Strings are implemented in various ways, depending on the programming language. The simplest way to implement strings is to create them as an array of characters, followed by a delimiting character used to signal the end of the string, usually NUL. These are referred to as null-terminated strings, and are usually found in languages with a low amount of hardware abstraction, such as C and Assembly. While easy to implement, null terminated strings have been criticized for causing buffer overflows. Most high-level scripting languages, such as Python, Ruby, and many dialects of BASIC, have no separate character type; strings with a length of one are normally used to represent single characters. Some languages, such as C++ and Java, have the capability to use null-terminated strings (usually for backwards-compatibility measures), but additionally provide their own class for string handling (std::string and java.lang.String, respectively) in the standard library.

There is also a difference on whether or not strings are mutable or immutable in a language. Mutable strings may be altered after their creation, whereas immutable strings maintain a constant size and content. In the latter, the only way to alter strings are to create new ones. There are both advantages and disadvantages to each approach: although immutable strings are much less flexible, they are simpler and completely thread-safe. Some examples of languages that use mutable strings include C++, Perl and Ruby, whereas languages that do not include JavaScript, Lua, Python and Go. A few languages, such as Objective-C, provide different types for mutable and immutable strings.

Literals for characters and strings are usually surrounded by quotation marks: sometimes, single quotes (') are used for characters and double quotes (") are used for strings. Python accepts either variant for its string notation.

Examples of character literals in C syntax are:

- 'A'
- '4'
- '\$'
- '\t' (tab character)

Examples of string literals in C syntax are:

- "A"
- "Hello World"
- "There are 4 cats."

Numeric data type ranges

Each numeric data type has its maximum and minimum value known as the range. Attempting to store a number outside the range may lead to compiler/runtime errors, or to incorrect calculations (due to truncation) depending on the language being used.

The range of a variable is based on the number of bytes used to save the value, and an integer data type is usually able to store 2^n values (where n is the number of bits that contribute to the value). For other data types (e.g. floating-point values) the range is more complicated and will vary depending on the method used to store it. There are also some types that do not use entire bytes, e.g. a boolean that requires a single bit, and represents a binary value (although in practice a byte is often used, with the remaining 7 bits being redundant).

Some programming languages (such as Ada and Pascal) also allow the opposite direction, that is, the programmer defines the range and precision needed to solve a given problem and the compiler chooses the most appropriate integer or floating-point type automatically.


See also

- Language primitive
- List of data structures § Data types
- Object type
- Primitive wrapper class
- Variable (computer science)

References

1. "Primitive Data Types (The Java™ Tutorials > Learning the Java Language > Language Basics)" (<https://docs.oracle.com/javase/tutorial/java/nutsandbolts/datatypes.html>). *docs.oracle.com*. Retrieved 2020-05-01.
2. "Data Types in C" (<https://www.geeksforgeeks.org/data-types-in-c/>). *GeeksforGeeks*. 2015-06-30. Retrieved 2020-05-01.
3. Fog, Agner (2010-02-16). "Calling conventions for different C++ compilers and operating systems: Chapter 3, Data Representation" (http://www.agner.org/optimize/calling_conventions.pdf) (PDF). Retrieved 2010-08-30.
4. ECMAScript 6th Edition draft: <https://people.mozilla.org/~jorendorff/es6-draft.html#sec-literals-numeric-literals> Archived (<https://web.archive.org/web/20131216202526/https://people.mozilla.org/~jorendorff/es6-draft.html>) 2013-12-16 at the Wayback Machine
5. Kernighan, Brian W; Ritchie, Dennis M (1978). *The C Programming Language* (1st ed.). Englewood Cliffs, NJ: Prentice Hall. p. 41 (<https://archive.org/details/cprogramminglang00kern/page/41>). ISBN 0-13-110163-3.
6. "Boolean type support library" (<https://devdocs.io/c/types/boolean>). *devdocs.io*. Retrieved October 15, 2020.
7. "Bool data type in C++" (<https://www.geeksforgeeks.org/bool-data-type-in-c/>). *GeeksforGeeks*. Retrieved October 15, 2020.
8. Mansoor, Umer. "The char Type in Java is Broken" (<https://codeahoy.com/2016/05/08/the-char-type-in-java-is-broken/>). *CodeAhoy*. Retrieved 10 February 2020.

External links

-  Media related to Primitive types at Wikimedia Commons

Retrieved from "https://en.wikipedia.org/w/index.php?title=Primitive_data_type&oldid=983689161"

This page was last edited on 15 October 2020, at 17:40 (UTC).

Text is available under the Creative Commons Attribution-ShareAlike License; additional terms may apply. By using this site, you agree to the Terms of Use and Privacy Policy. Wikipedia® is a registered trademark of the Wikimedia Foundation, Inc., a non-profit organization.