Strings

In some languages, a string is simply an array of characters. In other languages, strings have special status, with operations that are not available for arrays of other sorts. Scripting languages like Perl, Python, and Ruby have extensive suites of built-in string operators and functions, including sophisticated pattern matching facilities based on regular expressions.

Almost all programming languages allow literal strings to be specified as a sequence of characters, usually enclosed in a single or double quote marks. Most languages distinguish between literal characters (often delimited with single quotes) and literal strings (often delimited with double quotes). A few languages make no distinction, defining a character as simply a string of length one. Most languages also provided **escape sequences** that allow nonprinting characters and quote marks appear inside literal strings.

C and C++ provide a very rich set of escape sequences. An arbitrary character can be represented by a backslash followed by:

(a) 1to 3 octal (base 8) digits,

(b) an `x` and one or more hexadecimal (base 16) digits,

(c) a `u` and exactly four hexadecimal digits, or

(d) a `U` and exactly eight hexadecimal digits.

The \U notation is meant to capture the four-byte (32-bit) Unicode character set. The \u notation is for characters in the Basic Multillingual Plane. Many of the common control characters also have single-character escape sequences, many of which have been adopted by other languages as well.

The set of operations provided for strings is strongly tied to the implementation envisioned by the language desiner(s). Several languages that do not in general allow arrays to change size dynamically do provide this flexibility for strings. Manipulation of variable-length strings is fundamental to a huge number of computer applications. The fact that strings are one-dimensional, have one-bye elements, and never contain references to anything else makes dynamic size strings easier to implement than general dynamic arrays.

Some languages require that the length of a string-valued variable be bound no later than elaboration time, allowing the variable to be implemented as a contiguous array of characters in the current stack frame. C provided only the ability to create a pointer to a string literal. because of C's unification of arrays and pointers, even assignment is not supported. Given the declaration `char *s`, the statement `s = "abc"` makes `s` point to the constant `"abc"` in static storage. If `s` is declared as an array, rather than a pointer `char s`[4], then the statement will trigger an error message from the compiler. To assign one array into another in C, the progrm must copy the characters individually.

Other languages allow the length of a string-valued variable to change over its lifetime, requiring that the variable be implemented as a block or chain of blocks in the heap. Ml and Lisp provide strings as a built-in type. C++, Java, and C# provide them as predefined classes of object, in the formal, object-oriented sense. In all these languages a string is a **reference** to a string. Assigning a new value to such a variable makes it refer to a different object—each such object is immutable. Concatenation and other string operators implicitly create new objects. The space used by objects that are no longer reachable from any variable is reclaimed automatically.