In computer programming, a **string** is traditionally a sequence of characters, either as a literal constant or as some kind of variable. The latter may allow its elements to be mutated and the length changed, or it may be fixed (after creation). A string is generally considered as a data type and is often implemented as an array data structure of bytes (or words) that stores a sequence of elements, typically characters, using some character encoding. *String* may also denote more general arrays or other sequence (or list) data types and structures.

Depending on the programming language and precise data type used, a variable declared to be a string may either cause storage in memory to be statically allocated for a predetermined maximum length or employ dynamic allocation to allow it to hold a variable number of elements.

When a string appears literally in source code, it is known as a string literal or an anonymous string.[1]

In formal languages, which are used in mathematical logic and theoretical computer science, a string is a finite sequence of symbols that are chosen from a set called an alphabet.

String datatypes

A **string datatype** is a datatype modeled on the idea of a formal string. Strings are such an important and useful datatype that they are implemented in nearly every programming language. In some languages they are available as primitive types and in others as composite types. The syntax of most high-level programming languages allows for a string, usually quoted in some way, to represent an instance of a string datatype;

such a meta-string is called a *literal* or *string literal*.

String length

Although formal strings can have an arbitrary finite length, the length of strings in real languages is often constrained to an artificial maximum. In general, there are two types of string datatypes: *fixed-length strings*, which have a fixed maximum length to be determined at compile time and which use the same amount of memory whether this maximum is needed or not, and *variable-length strings*, whose length is not arbitrarily fixed and which can use varying amounts of memory depending on the actual requirements at run time (see Memory management). Most strings in modern programming languages are variable-length strings. Of course, even variable-length strings are limited in length – by the size of available computer memory. The string length can be stored as a separate integer (which may put another artificial limit on the length) or implicitly through a termination character, usually a character value with all bits zero such as in C programming language. See also "Null-terminated" below.

Character encoding

String datatypes have historically allocated one byte per character, and, although the exact character set varied by region, character encodings were similar enough that programmers could often get away with ignoring this, since characters a program treated specially (such as period and space and comma) were in the same place in all the encodings a program would encounter. These character sets were typically based on ASCII or EBCDIC. If text in one encoding was displayed on a system using a different encoding, text was often mangled, though often somewhat readable and some computer users learned to read the mangled text.

Logographic languages such as Chinese, Japanese, and Korean (known collectively as CJK) need far more than 256 characters (the limit of a one 8-bit byte per-character encoding) for reasonable representation. The normal solutions involved keeping single-byte representations for ASCII and using two-byte representations for CJK ideographs. Use of these with existing code led to problems with matching and cutting of strings, the severity of which depended on how the character encoding was designed.

Some encodings such as the EUC family guarantee that a byte value in the ASCII range will represent only that ASCII character, making the encoding safe for systems that use those characters as field separators. Other encodings such as ISO-2022 and Shift-JIS do not make such guarantees, making matching on byte codes unsafe. These encodings also were not "self-synchronizing", so that locating character boundaries required backing up to the start of a string, and pasting two strings together could result in corruption of the second string.

Unicode has simplified the picture somewhat. Most programming languages now have a datatype for Unicode strings. Unicode's preferred byte stream format UTF-8 is designed not to have the problems described above for older multibyte encodings. UTF-8, UTF-16 and UTF-32 require the programmer to know that the fixed-size code units are different than the "characters", the main difficulty currently is incorrectly designed APIs that attempt to hide this difference (UTF-32 does make *code points* fixed-sized, but these are not "characters" due to composing codes).

Implementations

Some languages, such as C++ and Ruby, normally allow the contents of a string to be changed after it has been created; these are termed *mutable* strings. In other languages, such as Java and Python, the value is fixed and a new string must be created if any alteration is to be made; these are termed *immutable* strings (some of these languages also provide another type that is mutable, such as Java and .NET `StringBuilder`, the thread-safe Java `StringBuffer`, and the Cocoa `NSMutableString`).

Strings are typically implemented as arrays of bytes, characters, or code units, in order to allow fast access to individual units or substrings—including characters when they have a fixed length. A few languages such as Haskell implement them as linked lists instead.

Some languages, such as Prolog and Erlang, avoid implementing a dedicated string datatype at all, instead adopting the convention of representing strings as lists of character codes.

Representations

Representations of strings depend heavily on the choice of character repertoire and the method of character encoding. Older string implementations were designed to work with repertoire and encoding defined by ASCII, or more recent extensions like the ISO 8859 series. Modern implementations often use the extensive repertoire defined by Unicode along with a variety of complex encodings such as UTF-8 and UTF-16.

The term *byte string* usually indicates a general-purpose string of bytes, rather than strings of only (readable) characters, strings of bits, or such. Byte strings often imply that bytes can take any value and any data can be stored as-is, meaning that there should be no value interpreted as a termination value.

Most string implementations are very similar to variable-length arrays with the entries storing the character codes of corresponding characters. The principal difference is that, with certain encodings, a single logical character may take up more than one entry in the array. This happens for example with UTF-8, where single codes (UCS code points) can take anywhere from one to four bytes, and single characters can take an arbitrary number of codes. In these cases, the logical length of the string (number of characters) differs from the physical length of the array (number of bytes in use). UTF-32 avoids the first part of the problem.

Null-terminated

The length of a string can be stored implicitly by using a special terminating character; often this is the null character (NUL), which has all bits zero, a convention used and perpetuated by the popular C programming language.[2] Hence, this representation is commonly referred to as a **C string**. This representation of an *n*-character string takes *n* + 1 space (1 for the terminator), and is thus an implicit data structure.

In terminated strings, the terminating code is not an allowable character in any string. Strings with *length* field do not have this limitation and can also store arbitrary binary data.

An example of a *null-terminated string* stored in a 10-byte buffer, along with its ASCII (or more modern UTF-8) representation as 8-bit hexadecimal numbers is:

F	R	A	N	K	NUL	k	e	f	w
46_{16}	52_{16}	41_{16}	$4E_{16}$	$4B_{16}$	00_{16}	$6B_{16}$	65_{16}	66_{16}	77_{16}

The length of the string in the above example, "`FRANK`", is 5 characters, but it occupies 6 bytes. Characters after the terminator do not form part of the representation; they may be either part of other data or just garbage. (Strings of this form are sometimes called *ASCIZ strings*, after the original assembly language directive used to declare them.)

Byte- and bit-terminated

Using a special byte other than null for terminating strings has historically appeared in both hardware and software, though sometimes with a value that was also a printing character. `$` was used by many assembler systems, `:` used by CDC systems (this character had a value of zero), and the ZX80 used `"`[3] since this was the string delimiter in its BASIC language.

Somewhat similar, "data processing" machines like the IBM 1401 used a special word mark bit to delimit strings at the left, where the operation would start at the right. This bit had to be clear in all other parts of the string. This meant that, while the IBM 1401 had a seven-bit word, almost no-one ever thought to use this as a feature, and override the assignment of the seventh bit to (for example) handle ASCII codes.

Early microcomputer software relied upon the fact that ASCII codes do not use the high-order bit, and set it to indicate the end of a string. It must be reset to 0 prior to output.[4]

Length-prefixed

The length of a string can also be stored explicitly, for example by prefixing the string with the length as a byte value. This convention is used in many Pascal dialects; as a consequence, some people call such a string a **Pascal string** or **P-string**. Storing the string length as byte limits the maximum string length to 255. To avoid such limitations, improved implementations of P-strings use 16-, 32-, or 64-bit words to store the string length. When the *length* field covers the address space, strings are limited only by the available memory.

If the length is bounded, then it can be encoded in constant space, typically a machine word, thus leading to an implicit data structure, taking *n* + *k* space, where *k* is the number of characters in a word (8 for 8-bit ASCII on a 64-bit machine, 1 for

32-bit UTF-32/UCS-4 on a 32-bit machine, etc.). If the length is not bounded, encoding a length *n* takes log(*n*) space (see fixed-length code), so length-prefixed strings are a succinct data structure, encoding a string of length *n* in log(*n*) + *n* space.

In the latter case, the length-prefix field itself doesn't have fixed length, therefore the actual string data needs to be moved when the string grows such that the length field needs to be increased.

Here is a Pascal string stored in a 10-byte buffer, along with its ASCII / UTF-8 representation:

length	F	R	A	N	K	k	e	f	w
05_{16}	46_{16}	52_{16}	41_{16}	$4E_{16}$	$4B_{16}$	$6B_{16}$	65_{16}	66_{16}	77_{16}

Strings as records

Many languages, including object-oriented ones, implement strings as records with an internal structure like:

```
class string {
  size_t length;
  char *text;
};
```

However, since the implementation is usually hidden, the string must be accessed and modified through member functions. `text` is a pointer to a dynamically allocated memory area, which might be expanded as needed. See also string (C++).

Other representations

Both character termination and length codes limit strings: For example, C character arrays that contain null (NUL) characters cannot be handled directly by C string library functions: Strings using a length code are limited to the maximum value of the length code.

Both of these limitations can be overcome by clever programming.

It is possible to create data structures and functions that manipulate them that do not have the problems associated with character termination and can in principle overcome length code bounds. It is also possible to optimize the string represented using techniques from run length encoding (replacing repeated characters by the character value and a length) and Hamming encoding.

While these representations are common, others are possible. Using ropes makes certain string operations, such as insertions, deletions, and concatenations more efficient.

The core data structure in a text editor is the one that manages the string (sequence of characters) that represents the current state of the file being edited. While that state could be stored in a single long consecutive array of characters, a typical text editor instead uses an alternative representation as its sequence data structure—a gap buffer, a linked list of lines, a piece table, or a rope—which makes certain string operations, such as insertions, deletions, and undoing previous edits, more efficient.[5]

Security concerns

The differing memory layout and storage requirements of strings can affect the security of the program accessing the string data. String representations requiring a terminating character are commonly susceptible to buffer overflow problems if the terminating character is not present, caused by a coding error or an attacker deliberately altering the data. String representations adopting a separate length field are also susceptible if the length can be manipulated. In such cases, program code accessing the string data requires bounds checking to ensure that it does not inadvertently access or change data outside of the string memory limits.

String data is frequently obtained from user input to a program. As such, it is the responsibility of the program to validate the string to ensure that it represents the expected format. Performing limited or no validation of user input can cause a program to be vulnerable to code injection attacks.

Literal strings

Sometimes, strings need to be embedded inside a text file that is both human-readable and intended for consumption by a machine. This is needed in, for example, source code of programming languages, or in configuration files. In this case, the NUL character doesn't work well as a terminator since it is normally invisible (non-printable) and is difficult to input via a keyboard. Storing the string length would also be inconvenient as manual computation and tracking of the length is tedious and error-prone.

Two common representations are:

- Surrounded by quotation marks (ASCII 0x22 double quote or ASCII 0x27 single quote), used by most programming languages. To be able to include special characters such as the quotation mark itself, newline characters, or non-printable characters, escape sequences are often available, usually prefixed with the backslash character (ASCII 0x5C).
- Terminated by a newline sequence, for example in Windows INI files.

Non-text strings

While character strings are very common uses of strings, a string in computer science may refer generically to any sequence of homogeneously typed data. A bit string or byte string, for example, may be used to represent non-textual binary data retrieved from a communications medium. This data may or may not be represented by a string-specific datatype, depending on the needs of the application, the desire of the programmer, and the capabilities of the programming language being used. If the programming language's string implementation is not 8-bit clean, data corruption may ensue.

C programmers draw a sharp distinction between a "string", aka a "string of characters", which by definition is always null terminated, vs. a "byte string" or "pseudo string" which may be stored in the same array but is often not null terminated. Using C string handling functions on such a "byte string" often seems to work, but later leads to security problems.[6][7][8]

String processing algorithms

There are many algorithms for processing strings, each with various trade-offs. Competing algorithms can be analyzed with respect to run time, storage requirements, and so forth.

Some categories of algorithms include:

- String searching algorithms for finding a given substring or pattern
- String manipulation algorithms
- Sorting algorithms
- Regular expression algorithms
- Parsing a string
- Sequence mining

Advanced string algorithms often employ complex mechanisms and data structures, among them suffix trees and finite-state machines.

The name *stringology* was coined in 1984 by computer scientist Zvi Galil for the issue of algorithms and data structures used for string processing.[9]

Character string-oriented languages and utilities

Character strings are such a useful datatype that several languages have been designed in order to make string processing applications easy to write. Examples include the following languages:

- awk
- Icon
- MUMPS
- Perl

- Rexx
- Ruby
- sed
- SNOBOL
- Tcl
- TTM

Many Unix utilities perform simple string manipulations and can be used to easily program some powerful string processing algorithms. Files and finite streams may be viewed as strings.

Some APIs like Multimedia Control Interface, embedded SQL or printf use strings to hold commands that will be interpreted.

Recent scripting programming languages, including Perl, Python, Ruby, and Tcl employ regular expressions to facilitate text operations. Perl is particularly noted for its regular expression use,[10] and many other languages and applications implement Perl compatible regular expressions.

Some languages such as Perl and Ruby support string interpolation, which permits arbitrary expressions to be evaluated and included in string literals.

Character string functions

String functions are used to create strings or change the contents of a mutable string. They also are used to query information about a string. The set of functions and their names varies depending on the computer programming language.

The most basic example of a string function is the string length function – the function that returns the length of a string (not counting any terminator characters or any of the string's internal structural information) and does not modify the string. This function is often named `length` or `len`. For example, `length("hello world")` would return 11. Another common function is concatenation, where a new string is created by appending two strings, often this is the + addition operator.

Some microprocessor's instruction set architectures contain direct support for string operations, such as block copy (e.g. In intel x86m `REPNZ MOVSB`).[11]

Formal theory

Let Σ be a finite set of symbols (alternatively called characters), called the alphabet. No assumption is made about the nature of the symbols. A **string** (or **word**) over Σ is any finite sequence of symbols from Σ.[12] For example, if Σ = {0, 1}, then *01011* is a string over Σ.

The *length* of a string *s* is the number of symbols in *s* (the length of the sequence) and can be any non-negative integer; it is often denoted as |*s*|. The *empty string* is the unique string over Σ of length 0, and is denoted *ε* or *λ*.[12][13]

The set of all strings over Σ of length *n* is denoted Σ^n. For example, if Σ = {0, 1}, then Σ^2 = {00, 01, 10, 11}. Note that Σ^0 = {ε} for any alphabet Σ.

The set of all strings over Σ of any length is the Kleene closure of Σ and is denoted Σ^*. In terms of Σ^n,

$$\Sigma^* = \bigcup_{n \in \mathbb{N} \cup \{0\}} \Sigma^n$$

For example, if Σ = {0, 1}, then Σ^* = {ε, 0, 1, 00, 01, 10, 11, 000, 001, 010, 011, ...}. Although the set Σ^* itself is countably infinite, each element of Σ^* is a string of finite length.

A set of strings over Σ (i.e. any subset of Σ^*) is called a *formal language* over Σ. For example, if Σ = {0, 1}, the set of strings with an even number of zeros, {ε, 1, 00, 11, 001, 010, 100, 111, 0000, 0011, 0101, 0110, 1001, 1010, 1100, 1111, ...}, is a formal language over Σ.

Concatenation and substrings

Concatenation is an important binary operation on Σ^*. For any two strings *s* and *t* in Σ^*, their concatenation is defined as the sequence of symbols in *s* followed by the sequence of characters in *t*, and is denoted *st*. For example, if Σ = {a, b, ..., z}, *s* = bear, and *t* = hug, then *st* = bearhug and *ts* = hugbear.

String concatenation is an associative, but non-commutative operation. The empty string ε serves as the identity element; for any string *s*, ε*s* = *s*ε = *s*. Therefore, the set Σ* and the concatenation operation form a monoid, the free monoid generated by Σ. In addition, the length function defines a monoid homomorphism from Σ* to the non-negative integers (that is, a function $L : \Sigma^* \mapsto \mathbb{N} \cup \{0\}$, such that $L(st) = L(s) + L(t) \quad \forall s, t \in \Sigma^*$).

A string *s* is said to be a *substring* or *factor* of *t* if there exist (possibly empty) strings *u* and *v* such that *t* = *usv*. The relation "is a substring of" defines a partial order on Σ*, the least element of which is the empty string.

Prefixes and suffixes

A string *s* is said to be a prefix of *t* if there exists a string *u* such that *t* = *su*. If *u* is nonempty, *s* is said to be a *proper* prefix of *t*. Symmetrically, a string *s* is said to be a suffix of *t* if there exists a string *u* such that *t* = *us*. If *u* is nonempty, *s* is said to be a *proper* suffix of *t*. Suffixes and prefixes are substrings of *t*. Both the relations "is a prefix of" and "is a suffix of" are prefix orders.

Reversal

The reverse of a string is a string with the same symbols but in reverse order. For example, if *s* = abc (where a, b, and c are symbols of the alphabet), then the reverse of *s* is cba. A string that is the reverse of itself (e.g., *s* = madam) is called a palindrome, which also includes the empty string and all strings of length 1.

Rotations

A string *s* = *uv* is said to be a rotation of *t* if *t* = *vu*. For example, if Σ = {0, 1} the string 0011001 is a rotation of 0100110, where *u*

= 00110 and *v* = 01. As another example, the string abc has three different rotations, viz. abc itself (with *u*=abc, *v*=ε), bca (with *u*=bc, *v*=a), and cab (with *u*=c, *v*=ab).

Lexicographical ordering

It is often useful to define an ordering on a set of strings. If the alphabet Σ has a total order (cf. alphabetical order) one can define a total order on Σ^* called lexicographical order. For example, if Σ = {0, 1} and 0 < 1, then the lexicographical order on Σ^* includes the relationships ε < 0 < 00 < 000 < ... < 0001 < 001 < 01 < 010 < 011 < 0110 < 01111 < 1 < 10 < 100 < 101 <

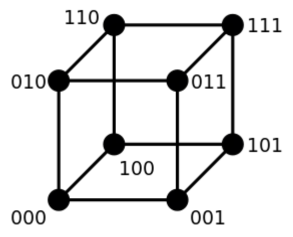
111 < 1111 < 11111 ... The lexicographical order is total if the alphabetical order is, but isn't well-founded for any nontrivial alphabet, even if the alphabetical order is.

See Shortlex for an alternative string ordering that preserves well-foundedness.

String operations

A number of additional operations on strings commonly occur in the formal theory. These are given in the article on string operations.

Topology



(Hyper)cube of binary strings of length 3

Strings admit the following interpretation as nodes on a graph, where *k* is the number of symbols in Σ:

- Fixed-length strings of length *n* can be viewed as the integer locations in an *n*-dimensional hypercube with sides of length *k*-1.
- Variable-length strings (of finite length) can be viewed as nodes on a perfect *k*-ary tree.

- Infinite strings (otherwise not considered here) can be viewed as infinite paths on a *k*-node complete graph.

The natural topology on the set of fixed-length strings or variable-length strings is the discrete topology, but the natural topology on the set of infinite strings is the limit topology, viewing the set of infinite strings as the inverse limit of the sets of finite strings. This is the construction used for the *p*-adic numbers and some constructions of the Cantor set, and yields the same topology.

Isomorphisms between string representations of topologies can be found by normalizing according to the lexicographically minimal string rotation.

See also

- Binary-safe — a property of string manipulating functions treating their input as raw data stream
- Bit array — a string of binary digits
- C string handling — overview of C string handling
- C++ string handling — overview of C++ string handling

- Comparison of programming languages (string functions)
- Connection string — passed to a driver to initiate a connection (e.g., to a database)
- Empty string — its properties and representation in programming languages
- Incompressible string — a string that cannot be compressed by any algorithm
- Rope (data structure) — a data structure for efficiently manipulating long strings
- String metric — notions of similarity between strings

References

Wikimedia Commons has media related to ***Strings***.

1. *"Introduction To Java - MFC 158 G" . Archived from the original on 2016-03-03. "String literals (or constants) are called 'anonymous strings'"*
2. *Bryant, Randal E.; David, O'Hallaron (2003), Computer Systems: A Programmer's Perspective (2003 ed.), Upper Saddle River, NJ: Pearson Education, p. 40, ISBN 0-13-034074-X, archived from the original on 2007-08-06*

3. *Wearmouth, Geoff. "An Assembly Listing of the ROM of the Sinclair ZX80" . Archived from the original on August 15, 2015.*
4. *Allison, Dennis. "Design Notes for Tiny BASIC" . Archived from the original on 2017-04-10.*
5. *Charles Crowley. "Data Structures for Text Sequences" Archived 2016-03-04 at the Wayback Machine. Section "Introduction" Archived 2016-04-04 at the Wayback Machine.*

6. *"strlcpy and strlcat - consistent, safe, string copy and concatenation." Archived 2016-03-13 at the Wayback Machine*
7. *"A rant about strcpy, strncpy and strlcpy." Archived 2016-02-29 at the Wayback Machine*
8. *Keith Thompson. "No, strncpy() is not a "safer" strcpy()". 2012.*
9. *"The Prague Stringology Club" . stringology.org. Archived from the original on 1 June 2015. Retrieved 23 May 2015.*

10. *"Essential Perl" . Archived from the original on 2012-04-21. "Perl's most famous strength is in string manipulation with regular expressions."*
11. *"x86 string instructions" . Archived from the original on 2015-03-27.*
12. *Barbara H. Partee; Alice ter Meulen; Robert E. Wall (1990). Mathematical Methods in Linguistics. Kluwer.*
13. *John E. Hopcroft, Jeffrey D. Ullman (1979). Introduction to Automata Theory, Languages, and Computation . Addison-Wesley. ISBN 0-201-02988-X. Here: sect.1.1, p.1*