B

B A typeless language invented by Ken Thompson at Bell Labs that was an ancestor of C.

backslash The character used to introduce an escape sequence (as in `\f` for form-feed and `\n` for new-line) in string literals and character constants. *See also* backslash escape sequence; backslash/new-line sequence.

backslash escape sequence An escape sequence, `\\`, that allows the backslash character to be represented in a string literal or character constant.

backslash/new-line sequence A source line terminated with a backslash immediately followed by a new-line. This sequence deems the source line to be continued on the following source line. Traditionally, this capability was only permitted in cases where you wanted to continue long macro definitions or long string literals. However, Standard C permits this notion by allowing any token to be broken across multiple source lines. Note there must be no characters between the backslash and new-line, not even white-space or comments. For example, the following is permissible:

```
#  \
def\
in\
e MAX  \
10\
00
in\
t i;
```

and is equivalent to

```
#define MAX  1000
int i;
```

See also phases of translation.

backspace escape sequence An escape sequence, `\b`, which represents the backspace character.

base documents Documents from which substantial parts of C89 were derived. Specifically, the language section was derived from "The C Reference Manual" by Dennis M. Ritchie, a version of which was published as Appendix A of *The C Programming Language*, first edition, by Brian W. Kernighan and Dennis M. Ritchie. *See also* K&R.

The library section was based on the 1984 */usr/group Standard* by the /usr/group Standards Committee, Santa Clara, California (November 14, 1984).

benign redefinition[C89] A method in which an object-like or function-like macro can be redefined multiple times in the same translation, provided the token sequence in each definition exactly matches that in all other definitions. All white-space character sequences are treated as being equivalent. *See also* macro, redefinition of.

binary operator *See* operator, binary.

binary stream *See* stream, binary.

`bitand`[C95] A macro, defined in iso646.h, that expands to the token `&`. It allows programmers using source character sets (such as ISO 646) that are missing certain characters necessary for writing C programs to enter those characters using identifiers instead. Note that in C++, this name is a keyword.

bit-field A structure or union member whose size is specified in bits. Bit-fields may have signed or unsigned `int` types. (C99 also allows bit-fields of type `_Bool` as well as other implementation-defined types.) It is implementation defined as to whether a plain `int` bit-field is signed. You cannot take the address of a bit-field (and therefore can't have an array of bit-fields) or find its size using `sizeof`. Standard C permits a bit-field to be a member of a union. Bit-fields are packed in an implementation-defined size storage unit. The order in which bit-fields are packed, and whether or not they may span storage unit boundaries, is implementation-defined.

bit-field, plain `int` A bit-field specified as having type `int`. When an object has type `int`, it is implied that this means it is signed. However, a bit-field having type `int` without either of the modifiers `signed` or `unsigned`, is an exception; such a bit-field might or might not be signed, as the implementation chooses.

`bitor`[C95] A macro, defined in iso646.h, that expands to the token `|`. It allows programmers using source character sets (such as ISO 646) that are missing certain characters necessary for writing C programs to enter those characters using identifiers instead. Note that in C++, this name is a keyword.

bitwise operators The operators `&` (*See* AND operator, bitwise), `|` (*See* OR operator, bitwise inclusive), `^` (*See* OR operator, bitwise exclusive), `<<` (*See* left-shift operator), `>>` (*See* right-shift operator), and `~` (*See* complement operator).

blank character *See* `isblank`; `iswblank`.

block That part of a function definition delimited by a matching pair of braces (excluding those braces delimiting initializer lists, structure or union layouts, enumeration definitions, or compound literals). A block defines the scope of locally declared identifiers, and it can be used to delimit the scope of a statement such as `if`/`else`, `while`, and `for`. Blocks may be nested and optionally may contain declarations and/or statements. A block is also referred to as a compound statement. The empty block ({}) is equivalent to the null statement `;` except that the body of a function cannot be a null statement.

`bool`[C99] A macro, defined in stdbool.h, that expands to the keyword `_Bool`. For normal usage of the boolean type, it is strongly recommended you include stdbool.h and use this macro rather than using the `_Bool` keyword directly. In C++, `bool` is a keyword.

`_Bool`[C99] A keyword that provides support for a boolean type. Since it was invented by C99, by which time many programs already contained type synonyms or macros named using some form of the word bool, this keyword was spelled using one of the forms reserved for implementers. Unless you must mix both existing homegrown boolean machinery and this new keyword in the same source file, it is strongly recommended that you include stdbool.h and use its macro `bool` instead.

`__bool_true_false_are_defined`[C99] A macro, defined in stdbool.h, that expands to the integer constant 1. It is intended for supporting backwards compatibility by allowing code to determine whether the standard boolean machinery is supported. *See also* `bool`; `_Bool`; `false`; `true`.

braces The characters { and } that delimit a compound statement (block), initializer list, structure or union layout, enumeration definition, or a compound literal. Braces are always used in matching pairs, except perhaps in comments, string literals, and character constants.

`break` A keyword that causes termination of the innermost current `while`, `for`, or `do` loop, or `switch` statement. It may not be used in any other context. Control is transferred to the statement immediately following that being terminated. It is subtly different from `continue`. `break` only permits breaking out one level. To break out of more levels, use `goto`. It is used as follows:

```
break;
```

broken-down time The components of a calendar time contained in an object of type `struct tm` defined in time.h.

`bsearch` A function that searches an array of `nmemb` objects, the initial member of which is pointed to by `base`, for a member that matches the object pointed to by `key`.

```
#include <stdlib.h>
void *bsearch(const void *key, const void *base,
    size_t nmemb, size_t size,
    int (*compar)(const void *, const void *));
```

`size` specifies the size of each member in the array. The members of the array are assumed to be sorted in an ascending order corresponding to that expected by the comparison function pointed to by `compar`. `compar` is passed two arguments, the first pointing to the key object, and the second pointing to the array member. Based on the comparison, a negative, zero, or positive value is returned from `compar`. If no match is found, `NULL` is returned by `bsearch`; otherwise, a pointer to the matching member in the array is returned. If two members compare as equal, it is unspecified as to which member is matched.

`btowc`[C95] A function that indicates whether its argument `c` is a valid single-byte character in the initial shift state, and if so, converts that character to a wide character.

```
#include <stdio.h>
#include <wchar.h>
wint_t btowc(int c);
```

If `c` has the value EOF, or its value cast to type `unsigned char` does not constitute a valid single-byte character in the initial shift state, `WEOF` is returned; otherwise, the wide character representation of that character is returned.

`BUFSIZ` A macro, defined in stdio.h, that expands to an integer constant expression that is the size of the buffer to be used by `setbuf`. Standard C requires it to be at least 256.

♢ ♢ ♢