-- --

IEEE P1003.2 Draft 11.2 − September 1991

A.1 `c89` — Compile Standard C programs

A.1.1 Synopsis

`c89` `[−c]` `[−D` *name* `[=`*value*`]]` ... `[−E]` `[−g]` `[−I` *directory* `]` ... `[−L` *directory* `]` ...
`[−o` *outfile* `]` `[−O]` `[−s]` `[−U` *name* `]` ... *operand* ...

A.1.2 Description

The `c89` utility is the interface to the standard C compilation system; it shall accept source code conforming to the C Standard {7}. The system conceptually consists of a compiler and link editor. The files referenced by *operand* s shall be compiled and linked to produce an executable file. (It is unspecified whether the linking occurs entirely within the operation of `c89`; some systems may produce objects that are not fully resolved until the file is executed.)

If the `−c` option is specified, for all pathname operands of the form *file* `.c`, the files

`$(basename` *pathname* `.c).o`

shall be created as the result of successful compilation. If the `−c` option is not specified, it is unspecified whether such `.o` files are created or deleted for the *file* `.c` operands.

If there are no options that prevent link editing (such as `−c` or `−E`), and all operands compile and link without error, the resulting executable file shall be written according to the `−o` *outfile* option (if present) or to the file `a.out`.

The executable file shall be created as specified in 2.9.1.4, except that the file permissions shall be set to

S_IRWXO | S_IRWXG | S_IRWXU

(see 5.6.1.2 in POSIX.1 {8})and that the bits specified by the *umask* of the process shall be cleared.

A.1.3 Options

The `c89` utility shall conform to the utility argument syntax guidelines described in 2.10.2, except that:

— The `−l` *library* operands have the format of options, but their position within a list of operands affects the order in which libraries are searched.

— The order of specifying the `−I` and `−L` options is significant. 1

— Conforming applications shall specify each option separately; that is, grouping option letters (e.g., `−cO`) need not be recognized by all implementations.

The following options shall be supported by the implementation:

`-c` Suppress the link-edit phase of the compilation, and do not remove any object files that are produced.

`-g` Produce symbolic information in the object or executable files; the nature of this information is unspecified, and may be modified by implementation-defined interactions with other options.

`-s` Produce object and/or executable files from which symbolic and other information not required for proper execution using *exec* (see POSIX.1 {8} 3.1.2) has been removed (stripped). If both `-g` and `-s` options are present, the action taken is unspecified.

`-o` *outfile* Use the pathname *outfile*, instead of the default `a.out`, for the executable file produced. If the `-o` option is present with `-c` or `-E`, the result is unspecified.

`-D` *name* [=*value*]

Define *name* as if by a C-language `#define` directive. If no =*value* is given, a value of 1 shall be used. The `-D` option has lower precedence than the `-U` option. That is, if *name* is used in both a `-U` and a `-D` option, *name* shall be undefined regardless of the order of the options. Additional implementation-defined *names* may be provided by the compiler. Implementations shall support at least 2048 bytes of `-D` definitions and 256 *names*.

`-E` Copy C-language source files to the standard output, expanding all preprocessor directives; no compilation shall be performed. If any operand is not a text file, the effects are unspecified.

`-I` *directory*

Change the algorithm for searching for headers whose names are not absolute pathnames to look in the directory named by the *directory* pathname before looking in the usual places. Thus, headers whose names are enclosed in double-quotes (`""`) shall be searched for first in the directory of the file with the `#include` line, then in directories named in `-I` options, and last in the usual places. For headers whose names are enclosed in angle brackets (`<>`), the header shall be searched for only in directories named in `-I` options and then in the usual places. Directories named in `-I` options shall be searched in the order specified. Implementations shall support at least ten instances of this option in a single `c89` command invocation.

`-L` *directory*

Change the algorithm of searching for the libraries named in the `-l` objects to look in the directory named by the *directory* pathname before looking in the usual places. Directories named in `-L` options shall be searched in the order specified. Implementations shall support at least ten instances of this option in a single `c89` command invocation. If a directory specified by a `-L` option contains files named `libc.a`, `libm.a`, `libl.a`, or `liby.a`, the results are unspecified.

`−O` Optimize. The nature of the optimization is unspecified.

`−U` *name* Remove any initial definition of *name* .

Multiple instances of the `−D`, `−I`, `−U`, and `−L` options can be specified.

A.1.4 Operands

An *operand* is either in the form of a pathname or the form `−l` *library* . At least one operand of the pathname form shall be specified. The following operands shall be supported by the implementation:

file.`c` A C-language source file to be compiled and optionally linked. The operand shall be of this form if the `−c` option is used.

file.`a` A library of object files typically produced by `ar` (see 6.1), and passed directly to the link editor. Implementations may recognize implementation-defined suffixes other than `.a` as denoting object file libraries.

file.`o` An object file produced by `c89 −c`, and passed directly to the link editor. Implementations may recognize implementation-defined suffixes other than `.o` as denoting object files.

The processing of other files is implementation defined.

`−l` *library* (The letter ell.) Search the library named:

`lib`*library* `.a`

A library shall be searched when its name is encountered, so the placement of a `−l` operand is significant. Several standard libraries can be specified in this manner, as described in A.1.7. Implementations may recognize implementation-defined suffixes other than `.a` as denoting libraries.

A.1.5 External Influences

A.1.5.1 Standard Input

None.

A.1.5.2 Input Files

The input file shall be one of the following: a text file containing a C-language source program; an object file in the format produced by `c89 −c`; or a library of object files, in the format produced by archiving zero or more object files, using `ar`. Implementations may supply additional utilities that produce files in these formats. Additional input file formats are implementation defined.

A.1.5.3 Environment Variables

The following environment variables shall affect the execution of `c89`:

LANG This variable shall determine the locale to use for the locale categories when both **LC_ALL** and the corresponding environment variable (beginning with **LC_**) do not specify a locale. See 2.6.

LC_ALL This variable shall determine the locale to be used to override any values for locale categories specified by the settings of **LANG** or any environment variables beginning with **LC_**.

LC_CTYPE This variable shall determine the locale for the interpretation of sequences of bytes of text data as characters (e.g., single- versus multibyte characters in arguments and input files).

LC_MESSAGES This variable shall determine the language in which messages should be written.

TMPDIR This variable shall be interpreted as a pathname that should override the default directory for temporary files, if any.

A.1.5.4 Asynchronous Events

Default.

A.1.6 External Effects

A.1.6.1 Standard Output

If more than one file operand ending in `.c` (or possibly other unspecified suffixes) is given, for each such file:

`"%s:\n"`, <*file*>

may be written. These messages, if written, shall precede the processing of each input file; they shall not be written to standard output if they are written to standard error, as described in A.1.6.2.

If the `−E` option is specified, the standard output shall be a text file that 1
represents the results of the preprocessing stage of the language; it may contain 1
extra information appropriate for subsequent compilation passes. 1

A.1.6.2 Standard Error

Used only for diagnostic messages. If more than one file operand ending in `.c` (or possibly other unspecified suffixes) is given, for each such file:

`"%s:\n"`, <*file*>

may be written to allow identification of the diagnostic and warning messages with the appropriate input file. These messages, if written, shall precede the processing of each input file; they shall not be written to the standard error if they are written to the standard output, as described in A.1.6.1.

This utility may produce warning messages about certain conditions that do not warrant returning an error (nonzero) exit value.

A.1.6.3 Output Files

Object files or executable files or both are produced in unspecified formats.

A.1.7 Extended Description

A.1.7.1 Standard Libraries

The `c89` utility shall recognize the following `−l` operands for standard libraries:

`−l c` This library contains all library functions referenced in `<stdlib.h>`, `<stdio.h>`, `<time.h>`, `<setjmp.h>`, `<signal.h>`, `<unistd.h>`, `<sys/types.h>`, `<string.h>`, and `<ctype.h>`, except for those functions referenced in `<math.h>`. If an invocation of

```
getconf _POSIX_VERSION
```

exits with a status of zero, the library searched also shall include all functions defined by POSIX.1 {8}; if the status is nonzero, it is unspecified whether these functions are available. If an invocation of

```
getconf _POSIX2_C_BIND
```

exits with a status of zero, the library searched also shall include all functions specified in Annex B; if the status is nonzero, it is unspecified whether these functions are available. An implementation shall not require this operand to be present to cause a search of this library.

`−l m` This library contains all functions referenced in `<math.h>`. An implementation may search this library in the absence of this operand.

`−l l` This library contains all functions required by the C-language output of `lex` (see A.2) that are not made available through the `−l c` operand.

`−l y` This library contains all functions required by the C-language output of `yacc` (see A.3) that are not made available through the `−l c` operand.

In the absence of options that inhibit invocation of the link editor, such as `−c` or `−E`, the `c89` utility shall cause the equivalent of a `−l c` operand to be passed to the link editor as the last `−l` operand, causing it to be searched after all other object files and libraries are loaded.

It is unspecified whether the libraries `libc.a`, `libm.a`, `libl.a`, and `liby.a` exist as regular files. The implementation may accept as `−l` operands names of objects that do not exist as regular files.

A.1.7.2 External Symbols

The C compiler and link editor shall support the significance of external symbols 1
up to a length of at least 31 bytes; the action taken upon encountering symbols exceeding the implementation-defined maximum symbol length is unspecified.

The compiler and link editor shall support a minimum of 511 external symbols per source or object file, and a minimum of 4095 external symbols total. A diagnostic message shall be written to the standard output if the implementation-defined limit is exceeded; other actions are unspecified.

A.1.8 Exit Status

The `c89` utility shall exit with one of the following values:

0 Successful compilation or link edit.

>0 An error occurred.

A.1.9 Consequences of Errors

When `c89` encounters a compilation error that causes an object file not to be created, it shall write a diagnostic to standard error and continue to compile other source code operands, but it shall not perform the link phase and shall return a nonzero exit status. If the link edit is unsuccessful, a diagnostic message shall be written to standard error and `c89` shall exit with a nonzero status.

A.1.10 Rationale. *(This subclause is not a part of P1003.2)*

Examples, Usage

Note that some implementations support a finer-grained model of compilation than the one described above. In this model, the following conceptual phases may exist: preprocessor, compiler, optimizer, assembler, link editor. Such implementations may support these additional options to the `c89` utility:

−P Preprocess, but do not compile, the named C programs and leave the result on corresponding files suffixed .i.

−S Compile the named C programs into assembly language, and leave the assembler-language output on corresponding files suffixed .s. No object files are created.

[−W*c*,*arg1* [,*arg2* . . .]]
Hand off the argument(s) *argi* to phase *c* where *c* is one of [p02al] indicating preprocessor, compiler, optimizer, assembler, or link editor, respectively. For example, −Wa,−m passes −m to the assembler phase. (Note the rationale concerning −W in 2.10.1.1.)

The −fpq options have been excluded, since they use features that are not in this standard.

In specifying that *file*.a operands are *typically* produced by ar, it is the intention of POSIX.2 to require that object libraries produced by ar be usable by c89, but not to preclude an implementation from supplying another utility that creates object library files.

The following are examples of usage:

c89 -o foo foo.c	Compiles foo.c and creates the executable foo.
c89 -c foo.c	Compiles foo.c and creates the object file foo.o.
c89 foo.c	Compiles foo.c and creates the executable a.out.
c89 foo.c bar.o	Compiles foo.c, links it with bar.o, and creates the executable a.out. Also creates and leaves foo.o.

The following examples clarify the use and interactions of −L options and −l operands:

Consider the case in which module a.c calls function *f*() in library libQ.a, and module b.c calls function *g*() in library libp.a. Assume that both libraries reside in /a/b/c. The command line to compile and link in the desired way is:

```
c89 -L /a/b/c main.o a.c -l Q b.c -l p
```

In this case the −l Q operand need only precede the first −l p operand, since both libQ.a and libp.a reside in the same directory.

Multiple −L operands can be used when library name collisions occur. Building on the previous example, suppose that we now want to use a new libp.a, in /a/a/a, but we still want *f*() from /a/b/c/libQ.a.

```
c89 -L /a/a/a -L /a/b/c main.o a.c -l Q b.c -l p
```

In this example, the linker searches the −L options in the order specified, and finds /a/a/a/libp.a before /a/b/c/libp.a when resolving references for b.c. The order of the −l operands is still important, however.

There is the possible implication that if a user supplies versions of the standard library functions (before they would be encountered by an implicit −l c or explicit

`-l m`), that those versions would be used in place of the standard versions. There are various reasons this might not be true (functions defined as macros, manipulations for clean namespace, etc.), so the existence of files named in the same manner as the standard libraries within the `-L` directories is explicitly stated to produce unspecified behavior.

Some historical implementations have permitted `-L` options to be interspersed with `-l` operands on the command line; with respect to POSIX, such behavior would be considered a vendor extension. For an application to compile consistently on systems that do not behave like this, it is necessary for a conforming application to supply all `-L` options before any of the `-l` options.

Some historical implementations have created `.o` files when `-c` is not specified and more than one source file is given. Since this area is left unspecified, the application cannot rely on `.o` files being created, but it also must be prepared for any related `.o` files that already exist being deleted at the completion of the link edit.

History of Decisions Made

The name of this utility differs from the historical `cc` name. The C Standard {7} document was approved during the development of POSIX.2, and it is clear that POSIX must support Standard C; there is no other good way of specifying a C language. The support of the C Standard {7} by `c89` also mandates the Standard C math libraries. An alternative approach was considered: provide an option to select the type of compilation required. However, it was found that all available option letters were already in use in the various historical `cc` utilities. Thus, this name change is being used essentially as a switch. There was some temptation to use the name change as an excuse to mandate a cleaner interface (e.g., conform to the utility syntax guidelines), but this was resisted; the majority of early `c89` implementations are expected to be satisfied with historical `cc`s with only minimal changes. This was decided more from the standpoint of existing applications and makefiles than for the implementors' sake.

The `-l` *library* operand must be capable of being interspersed with file name operands so that the order in which libraries are searched by the link editor can be specified.

The search algorithm for `-I` *directory* states that the directory of the file with the `#include` file is searched first, rather than being implementation defined. It is believed that this reflects most implementations, and it disallows variations on different implementations, since this would make it very difficult to distribute source code in a compatible form.

The `-I` options are searched in the order specified (which is left to right in English). This resolves the conflict of what header file is used if multiple files with the same name exist in different directories in the `include` path.

In a future extension or supplement to this standard, *should* will be changed to *shall* with respect to support for **TMPDIR** by applications.

It is unclear whether `c89` requires such a large number of file descriptors that its requirement should be documented here; POSIX.2 remains silent on the issue. It is also noted that an undocumented feature of some C compilers is that if file descriptor 9 is open, a linkage trace is written to it.

There is no pseudo-*printf*() specification for compile errors because no common format could be identified. As new C compilers are written, they are encouraged to use the following format:

> `"%s: %s: %d %s\n"`, *<compiler phase>*, *<file name>*, *<line number>*, *<explanation>*

The following option proposals were considered and rejected:

(1) The `−M` option in BSD does not exist in System V, and is not seen to enhance application portability.

(2) The `−S` option was not seen to enhance application portability, and makes assumptions about the underlying architecture.

Earlier drafts included a `−V` option to select a compiler version. Not only did this letter (and every other upper- and lowercase letter) collide with one historical implementation or another, but there was no agreement on how many compiler versions should be defined, or what they should mean. Another choice is to specify that the `cc` utility invoke a Standard C compiler. By specifying `c89` instead, an installation is able to link either a ‘‘common usage’’ or a Standard C compiler to the name `cc`. Implementors are free to select implementation-defined options to select (nonportable) extensions to their existing C compiler to aid the transition to Standard C.

The `−g` and `−s` options are not specified as mutually exclusive. Historically these two options have been mutually exclusive, but because both are so loosely specified, it seemed cleaner to leave their interaction unspecified.

The `−E` option was added because headers are not required to be separate files in a POSIX.1-conformant system; these values could be hard-coded into the compiler, or might only be accessible in a nonportable way. Hence, while not strictly required for application portability, this option is a practical necessity as a portable means for ascertaining the real effects of preprocessor statements.

In BSD systems, using `−c` and `−o` in the same command causes the object module to be stored in the specified file. In System V, this produces an error condition. Therefore, POSIX.2 indicates that this is an unspecified condition.

Reasonably precise specification of standard library access is required. Implementations are not required to have `/usr/lib/libc.a`, etc., as many historical implementations do, but if not they are required to recognize `c`, `m`, `l`, and `y` as tokens. Libraries `l` and `y` can be empty if the library functions specified for `lex` and `yacc` are accessible through the `−l c` operand. Historically, these libraries have been necessary, but they are not required for a conforming implementation.

External symbol size limits are in a normative subclause; portable applications need to know these limits. However, the minimum maximum symbol length should be taken as a constraint on a portable application, not on an implementation, and consequently the action taken for a symbol exceeding the limit is unspecified. The minimum size for the external symbol table was added for similar reasons.

The Consequences of Errors subclause clearly specifies the compiler's behavior when compilation or link-edit error occur. The behavior of several historical implementations was examined, and the choice was made to be silent on the status of the executable, or `a.out`, file in the face of compiler or linker errors. If a linker writes the executable file, then links it on disk with *lseek* ()s and *write* ()s, the partially-linked executable can be left on disk and its execute bits turned off if the link edit fails. However, if the linker links the image in memory before writing the file to disk, it need not touch the executable file (if it already exists) because the link edit fails. Since both approaches are existing practice, a portable application shall rely on the exit status of `c89`, rather than on the existence or mode of the executable file.

The requirement that portable applications specify compiler options separately is to reserve the multicharacter option namespace for vendor-specific compiler options, which are known to exist in many historical implementations. Implementations are not required to recognize, for example `−gc` as if it were `−g −c`; nor are they forbidden from doing so. The synopsis shows all of the options separately to highlight this requirement on applications.

Echoing filenames to standard error is considered a diagnostic message, because it might otherwise be difficult to associate an error message with the erring file. The text specifies either standard error or standard output for these messages because some historical practice uses standard output, but there was considerable sentiment expressed for allowing it to be on standard error instead. The rationale for using standard output is that these are not really error message headers, but a running progress report on which files have been processed. The messages are described as optional because there might be different ways of constructing the compiler's messages that should not be precluded.

A.2 `lex` — Generate programs for lexical tasks

A.2.1 Synopsis

`lex [−t] [ −n | −v ] [`*file* ... `]`

Obsolescent Version :

`lex −c [−t] [ −n | −v ] [`*file* ... `]`

A.2.2 Description

The `lex` utility shall generate C programs to be used in lexical processing of character input, and that can be used as an interface to `yacc` (see A.3). The C programs shall be generated from `lex` source code and conform to the C Standard {7}. Usually, the `lex` utility writes the program it generates to the file `lex.yy.c`; the state of this file is unspecified if `lex` exits with a nonzero exit status. See A.2.7 for a complete description of the `lex` input language.

A.2.3 Options

The `lex` utility shall conform to the utility argument syntax guidelines described in 2.10.2.

The following options shall be supported by the implementation:

`−c` (Obsolescent.) Indicate C-language action (default option).

`−n` Suppress the summary of statistics usually written with the `−v` option. If no table sizes are specified in the `lex` source code and the `−v` option is not specified, then `−n` is implied.

`−t` Write the resulting program to standard output instead of `lex.yy.c`.

`−v` Write a summary of `lex` statistics to the standard output. (See the discussion of `lex` table sizes in A.2.7.1.) If the `−t` option is specified and `−n` is not specified, this report shall be written to standard error. If table sizes are specified in the `lex` source code, and if the `−n` option is not specified, the `−v` option may be enabled.

A.2.4 Operands

The following operand shall be supported by the implementation: