IEEE P1003.2 Draft 11.2 – September 1991

Copyright © 1991 by the
Institute of Electrical and Electronics Engineers, Inc.
345 East 47th Street
New York, NY 10017, USA
All rights reserved as an unpublished work.

This is an unapproved and unpublished IEEE Standards Draft, subject to change. The publication, distribution, or copying of this draft, as well as all derivative works based on this draft, is expressly prohibited except as set forth below.

Permission is hereby granted for IEEE Standards Committee participants to reproduce this document for purposes of IEEE standardization activities only, and subject to the restrictions contained herein.

Permission is hereby also granted for member bodies and technical committees of ISO and IEC to reproduce this document for purposes of developing a national position, subject to the restrictions contained herein.

Permission is hereby also granted to the preceding entities to make limited copies of this document in an electronic form only for the stated activities.

The following restrictions apply to reproducing or transmitting the document in any form: 1) all copies or portions thereof must identify the document's IEEE project number and draft number, and must be accompanied by this entire notice in a prominent location; 2) no portion of this document may be redistributed in any modified or abridged form without the prior approval of the IEEE Standards Department.

Other entities seeking permission to reproduce this document, or any portion thereof, for standardization or other activities, must contact the IEEE Standards Department for the appropriate license.

Use of information contained in this unapproved draft is at your own risk.

IEEE Standards Department
Copyright and Permissions
445 Hoes Lane, P.O. Box 1331
Piscataway, NJ 08855-1331, USA
+1 (908) 562-3800
+1 (908) 562-1571 [FAX]

7.2.1 Access Environment Variables Rationale. *(This subclause is not a part of P1003.2)*

This facility is required in POSIX. 2 so that applications can obtain values of exported shell variables.

7.3 Regular Expression Matching

Any language binding to Language-Independent System Services shall include a facility to interpret regular expressions as described in 2.8.

The language-independent specification for this facility has not been developed. The C binding is the *regcomp* (), *regexec* (), and *regfree* () functions described in B.5.

7.3.1 Regular Expression Matching Rationale. *(This subclause is not a part of P1003.2)*

This service is important enough that it should be required by any language binding to POSIX. 2.

Regular expression parsing and pattern matching are listed separately, since they are different services. A language binding could provide different functions to support regular expressions and patterns, or could combine them into a single function.

7.4 Pattern Matching

Any language binding to Language-Independent System Services shall include a facility to interpret patterns as described in 3.13.1 and 3.13.2. This facility shall allow the application to specify whether a slash character in the string to be matched will be treated as a regular character, or must be explicitly matched against a slash in the pattern.

The language-independent specification for this facility has not been developed. The C binding is the *fnmatch* () function described in B.6.

7.5 Command Option Parsing

Any language binding to Language-Independent System Services shall include a facility to parse the options and operands from the command line that invoked the application.

The language-independent specification for this facility has not been developed. The C binding for this facility is the *getopt* () function described in B.7.

Copyright © 1991 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

7.6 Generate Pathnames Matching a Pattern

Any language binding to Language-Independent System Services shall include a facility to generate pathnames matching a pattern as described in 3.13.

The language-independent specifications for this facility has not been developed. The C binding is the *glob* () and *globfree* () functions described in B.8.

7.7 Perform Word Expansions

Any language binding to Language-Independent System Services shall include a facility to do shell word expansions as described in 3.6.

The language-independent specification for this facility has not been developed. The C binding is the *wordexp* () and *wordfree* () functions described in B.9.

7.7.1 Perform Word Expansions Rationale. *(This subclause is not a part of P1003.2)*

See the rationale for this function in B.9.

7.8 Get POSIX Configurable Variables

7.8.1 Get String-Valued Configurable Variables

Any language binding to Language-Independent System Services shall include a facility to obtain string configurable variables.

The language-independent specification for this facility has not been developed. The C binding for this facility is the *confstr* () function described in B.10.1.

7.8.2 Get Numeric-Valued Configurable Variables

Any language binding to Language-Independent System Services shall include facilities to determine the current values of system and pathname limits or options (*variables*), as specified by POSIX .1 {8}. The configurable variables listed in Table 7-1, which are defined in POSIX .1 {8}, shall be available in any POSIX. 2 language-dependent binding, with minimum values as given in POSIX .1 {8}. Other POSIX .1 {8} configurable variables may be supported, but are not required by POSIX. 2. This facility shall also make available current values for all system limits defined in 2.13.

The language-independent specifications for these facilities have not been developed. The C bindings are the *sysconf* () function described in POSIX .1 {8}4.8, and the *pathconf* () and *fpathconf* () functions defined in POSIX .1 {8}5.7.

Copyright © 1991 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

7.8.2.1 Get Numeric-Valued Configurable Variables Rationale. *(This subclause is not a part of P1003.2)*

This description calls out specific values that *sysconf* (), *pathconf* (), and *fpathconf* () are required to support. Some of the POSIX .1 {8} values are excluded from this list because they are not relevant in a POSIX. 2-only environment. Currently, only {CLK _TCK } is not required by POSIX. 2.

This description does not specify the *name* values for the arguments to the various functions. This is because different language bindings might use different naming conventions, or might use a completely different scheme for obtaining the required configurable values. Specific names for the *name* values for the C language binding are given in B.10.2.

7.9 Locale Control

Any language binding to Language-Independent System Services shall include a facility to set locale control information.

The language-independent specification for this facility has not been developed. The C binding for this facility is described in B.11.

7.9.0.1 Locale Control Rationale. *(This subclause is not a part of P1003.2)*

This facility is required in POSIX. 2 so that applications can control the locale, which affects the operation of POSIX. 2 utilities.

Table 7-1 – POSIX .1 Numeric-Valued Configurable Variables

{ARG_MAX}	{NAME_MAX}	{_POSIX _CHOWN _RESTRICTED }
{CHILD _MAX}	{NGROUPS _MAX}	{_POSIX _JOB _CONTROL }
{LINK _MAX}	{OPEN _MAX}	{_POSIX _NO_TRUNC }
{MAX_CANON }	{PATH _MAX}	{_POSIX _SAVED _IDS }
{MAX_INPUT }	{PIPE _BUF }	{_POSIX _VDISABLE }

Copyright © 1991 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

Annex A
(norm ative)

C Language Developm ent Utilities Option

This annex describes utilities used for the development of C language applications, including compilation or translation of C source code and complex program generators for simple lexical tasks and processing of context-free grammars.

The utilities described in this annex may be provided by the conforming system; however, any system claiming conformance to the **C Language Development Utilities Option** shall provide all of the utilities described here. The utilities described in Section 6 are prerequisites to this annex.

A.0.1 C Language Development Utilities Option Rationale. *(This subclause is not a part of P1003.2)*

The portions of this standard that concern specific languages—currently C and FORTRAN —have been collected to the rear of the document as Normative Annexes. For purposes of conformance, they are no less a part of the standard than one of the numbered sections. They were grouped as Annexes to illustrate that the base standard is [planned to be] language independent, giving a small degree of separation. The working group also wished to send a message to those groups planning other language bindings: the standard is not C-oriented, and there's plenty of room to add more annexes for your languages as you develop them, right alongside C and FORTRAN.

Copyright © 1991 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

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.

— 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:

Copyright © 1991 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

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

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

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

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

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

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

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

89 −L *directory*
90 Change the algorithm of searching for the libraries named in the
91 −l objects to look in the directory named by the *directory* path-
92 name before looking in the usual places. Directories named in −L
93 options shall be searched in the order specified. Implementations
94 shall support at least ten instances of this option in a single c89
95 command invocation. If a directory specified by a −L option con-
96 tains files named libc.a, libm.a, libl.a, or liby.a, the
97 results are unspecified.

Copyright © 1991 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

98 `-O` Optimize. The nature of the optimization is unspecified.

99 `-U` *name* Remove any initial definition of *name* .

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

101 ### A.1.4 Operands

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

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

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

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

114 The processing of other files is implementation defined.

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

116 `lib`*library*`.a`

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

122 ### A.1.5 External Influences

123 #### A.1.5.1 Standard Input

124 None.

125 #### A.1.5.2 Input Files

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

Copyright © 1991 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

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 represents the results of the preprocessing stage of the language; it may contain extra information appropriate for subsequent compilation passes.

Copyright © 1991 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

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.

Copyright © 1991 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

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

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

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

208 #### A.1.7.2 External Symbols

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

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

216 ### A.1.8 Exit Status

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

218 0 Successful compilation or link edit.

219 >0 An error occurred.

220 ### A.1.9 Consequences of Errors

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

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

227 **Examples, Usage**

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

Copyright © 1991 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

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

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

237 **[–W***c*,*arg1* **[,***arg2 ...* **]]**
238 Hand off the argument(s) *argi* to phase *c* where *c* is one of [**p02al**] indi-
239 cating preprocessor, compiler, optimizer, assembler, or link editor,
240 respectively. For example, **–Wa,–m** passes **–m** to the assembler phase.
241 (Note the rationale concerning **–W** in 2.10.1.1.)

242 The **–fpq** options have been excluded, since they use features that are not in this
243 standard.

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

248 The following are examples of usage:

249 **c89 -o foo foo.c** Compiles **foo.c** and creates the executable **foo**.

250 **c89 -c foo.c** Compiles **foo.c** and creates the object file **foo.o**.

251 **c89 foo.c** Compiles **foo.c** and creates the executable **a.out**.

252 **c89 foo.c bar.o** Compiles **foo.c**, links it with **bar.o**, and creates the
253 executable **a.out**. Also creates and leaves **foo.o**.

254 The following examples clarify the use and interactions of **–L** options and **–l**
255 operands:

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

260 ```
c89 -L /a/b/c main.o a.c -l Q b.c -l p
```

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

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

266 ```
c89 -L /a/a/a -L /a/b/c main.o a.c -l Q b.c -l p
```

267 In this example, the linker searches the **–L** options in the order specified,
268 and finds **/a/a/a/libp.a** before **/a/b/c/libp.a** when resolving refer-
269 ences for **b.c**. The order of the **–l** operands is still important, however.

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

Copyright © 1991 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

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

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

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

287 **History of Decisions Made**

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

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

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

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

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

Copyright © 1991 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

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

Copyright © 1991 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

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.

Copyright © 1991 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

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:

Copyright © 1991 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

file A pathname of an input file. If more than one such *file* is specified, all files shall be concatenated to produce a single `lex` program. If no *file* operands are specified, or if a *file* operand is –, the standard input shall be used.

A.2.5 External Influences

A.2.5.1 Standard Input

The standard input shall be used if no *file* operands are specified, or if a *file* operand is –. See Input Files.

A.2.5.2 Input Files

The input files shall be text files containing `lex` source code, as described in A.2.7.

A.2.5.3 Environment Variables

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

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_COLLATE This variable shall determine the locale for the behavior of ranges, equivalence classes, and multicharacter collating elements within regular expressions. If this variable is not set to the POSIX Locale, the results are unspecified.

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) and the behavior of character classes within extended regular expressions. If this variable is not set to the POSIX Locale, the results are unspecified.

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

Copyright © 1991 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

A.2.5.4 Asynchronous Events

Default.

A.2.6 External Effects

A.2.6.1 Standard Output

If the `-t` option is specified, the text file of C source code output of `lex` shall be written to standard output.

If the `-t` option is not specified:

(1) Implementation-defined informational, error, and warning messages concerning the contents of `lex` source code input shall be written to either the standard output or standard error.

(2) If the `-v` option is specified and the `-n` option is not specified, `lex` statistics shall also be written to either the standard output or standard error, in an implementation-defined format. These statistics may also be generated if table sizes are specified with a `%` operator in the *Definitions* section (see A.2.7), as long as the `-n` option is not specified.

A.2.6.2 Standard Error

If the `-t` option is specified, implementation-defined informational, error, and warning messages concerning the contents of `lex` source code input shall be written to the standard error.

If the `-t` option is not specified:

(1) Implementation-defined informational, error, and warning messages concerning the contents of `lex` source code input shall be written to either the standard output or standard error.

(2) If the `-v` option is specified and the `-n` option is not specified, `lex` statistics shall also be written to either the standard output or standard error, in an implementation-defined format. These statistics may also be generated if table sizes are specified with a `%` operator in the *Definitions* section (see A.2.7), as long as the `-n` option is not specified.

A.2.6.3 Output Files

A text file containing C source code shall be written to `lex.yy.c`, or to the standard output if the `-t` option is present.

Copyright © 1991 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

A.2.7 Extended Description

Each input file contains `lex` source code, which is a table of regular expressions with corresponding actions in the form of C program fragments.

When `lex.yy.c` is compiled and linked with the `lex` library (using the `-l l` operand with `c89`), the resulting program reads character input from the standard input and partitions it into strings that match the given expressions.

When an expression is matched, these actions shall occur:

— The input string that was matched is left in *yytext* as a null-terminated string; *yytext* is either an external character array or a pointer to a character string. As explained in A.2.7.1, the type can be explicitly selected using the `%array` or `%pointer` declarations, but the default is implementation defined.

— The external *int yyleng* is set to the length of the matching string.

— The expression's corresponding program fragment, or action, is executed.

During pattern matching, `lex` shall search the set of patterns for the single longest possible match. Among rules that match the same number of characters, the rule given first shall be chosen.

The general format of `lex` source is:

Definitions
%%
Rules
%%
User Subroutines

The first `%%` is required to mark the beginning of the rules (regular expressions and actions); the second `%%` is required only if user subroutines follow.

Any line in the *Definitions* section beginning with a `<blank>` shall be assumed to be a C program fragment and shall be copied to the external definition area of the `lex.yy.c` file. Similarly, anything in the *Definitions* section included between delimiter lines containing only `%{` and `%}` shall also be copied unchanged to the external definition area of the `lex.yy.c` file.

Any such input (beginning with a `<blank>` or within `%{` and `%}` delimiter lines) appearing at the beginning of the *Rules* section before any rules are specified shall be written to `lex.yy.c` after the declarations of variables for the *yylex* () function and before the first line of code in *yylex* (). Thus, user variables local to *yylex* () can be declared here, as well as application code to execute upon entry to *yylex* ().

The action taken by `lex` when encountering any input beginning with a `<blank>` or within `%{` and `%}` delimiter lines appearing in the *Rules* section but coming after one or more rules is undefined. The presence of such input may result in an erroneous definition of the *yylex* () function.

Copyright © 1991 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

A.2.7.1 `lex` Definitions

Definitions appear before the first `%%` delimiter. Any line in this section not contained between `%{` and `%}` lines and not beginning with a `<blank>` shall be assumed to define a `lex` substitution string. The format of these lines is:

name substitute

If a *name* does not meet the requirements for identifiers in the C Standard {7},the result is undefined. The string *substitute* shall replace the string {*name* } when it is used in a rule. The *name* string shall be recognized in this context only when the braces are provided and when it does not appear within a bracket expression or within double-quotes.

In the *Definitions* section, any line beginning with a `%` (percent-sign) character and followed by an alphanumeric word beginning with either `s` or `S` shall define a set of start conditions. Any line beginning with a `%` followed by a word beginning with either `x` or `X` shall define a set of exclusive start conditions. When the generated scanner is in a `%s` state, patterns with no state specified shall be also active; in a `%x` state, such patterns shall not be active. The rest of the line, after the first word, shall be considered to be one or more `<blank>`-separated names of start conditions. Start condition names shall be constructed in the same way as definition names. Start conditions can be used to restrict the matching of regular expressions to one or more states as described in the section A.2.7.4.

Implementations shall accept either of the following two mutually exclusive declarations in the *Definitions* section:

`%array` Declare the type of *yytext* to be a null-terminated character array.

`%pointer` Declare the type of *yytext* to be a pointer to a null-terminated character string.

The default type of *yytext* is implementation defined. If an application refers to *yytext* outside of the scanner source file (i.e., via an `extern`), the application shall include the appropriate `%array` or `%pointer` declaration in the scanner source file.

Implementations shall accept declarations in the *Definitions* section for setting certain internal table sizes. The declarations are shown in Table A-1. In the table, *n* represents a positive decimal integer, preceded by one or more `<blank>`s. The exact meaning of these table size numbers is implementation defined. The implementation shall document how these numbers affect the `lex` utility and how they are related to any output that may be generated by the implementation should space limitations be encountered during the execution of `lex`. It shall be possible to determine from this output which of the table size values needs to be modified to permit `lex` to successfully generate tables for the input language. The values in the column Minimum Value represent the lowest values conforming implementations shall provide.

Copyright © 1991 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.