Full credit is given to the above companies including the OS that this PDF file was generated!

Red Hat Enterprise Linux Release 9.2 Manual Pages on 'fort77.1p' command

$ man fort77.1p

FORT77(1P) POSIX Programmer's Manual FORT77(1P)

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

fort77 ? FORTRAN compiler (FORTRAN)

SYNOPSIS

fort77 [-c] [-g] [-L directory]... [-O optlevel] [-o outfile] [-s] [-w] operand...

DESCRIPTION

The fort77 utility is the interface to the FORTRAN compilation system; it shall accept the full FORTRAN-77 language defined by the ANSI X3.9?1978 standard. The system conceptually consists of a compiler and link editor. The files referenced by operands are compiled and linked to produce an executable file. It is unspecified whether the linking occurs entirely within the operation of fort77; some implemen? tations may produce objects that are not fully resolved until the file is executed.

If the -c option is present, for all pathname operands of the form file.f, the files:

$(basename pathname.f).o

shall be created or overwritten as the result of successful compila? tion. If the -c option is not specified, it is unspecified whether such .o files are created or deleted for the file.f operands.

If there are no options that prevent link editing (such as -c) and all operands compile and link without error, the resulting executable file shall be written into the file named by the -o option (if present) or to the file a.out. The executable file shall be created as specified in the System Interfaces volume of POSIX.1?2017, except that the file permissions shall be set to: S_IRWXO | S_IRWXG | S_IRWXU and that the bits specified by the umask of the process shall be cleared.

OPTIONS

The fort77 utility shall conform to the Base Definitions volume of POSIX.1?2017, Section 12.2, Utility Syntax Guidelines, except that:

* The -l library operands have the format of options, but their posi? tion within a list of operands affects the order in which libraries are searched.
* The order of specifying the multiple -L options is significant.
* Conforming applications shall specify each option separately; that is, grouping option letters (for example, -cg) need not be recog? nized by all implementations.

The following options shall be supported:

-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 or executable files, or both, from which sym? bolic and other information not required for proper execution using the exec family of functions defined in the System In? terfaces volume of POSIX.1?2017 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, the result is unspecified.

-L directory

Change the algorithm of searching for the libraries named in -l operands 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 specified order. At least ten instances of this option shall be supported in a single fort77 command invocation. If a directory specified by a -L option contains a file named libf.a, the results are un? specified.

-O optlevel

Specify the level of code optimization. If the optlevel op? tion-argument is the digit '0', all special code optimiza? tions shall be disabled. If it is the digit '1', the nature of the optimization is unspecified. If the -O option is omit? ted, the nature of the system's default optimization is un? specified. It is unspecified whether code generated in the presence of the -O 0 option is the same as that generated when -O is omitted. Other optlevel values may be supported.

-w Suppress warnings.

Multiple instances of -L options can be specified.

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 fol? lowing operands shall be supported:

file.f The pathname of a FORTRAN source file to be compiled and op? tionally passed to the link editor. The filename operand shall be of this form if the -c option is used.

file.a A library of object files typically produced by ar, and

passed directly to the link editor. Implementations may rec?
ognize implementation-defined suffixes other than .a as de?
noting object file libraries.

file.o An object file produced by fort77 -c and passed directly to the link editor. Implementations may recognize implementa?
tion-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:

liblibrary.a

A library is 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 the EXTENDED DESCRIPTION section. Implementations may recog?
nize implementation-defined suffixes other than .a as denot?
ing libraries.

STDIN

Not used.

INPUT FILES

The input file shall be one of the following: a text file containing FORTRAN source code; an object file in the format produced by fort77 -c; or a library of object files, in the format produced by archiving zero or more object files, using ar. Implementations may supply addi?
tional utilities that produce files in these formats. Additional input files are implementation-defined.

A <tab> encountered within the first six characters on a line of source code shall cause the compiler to interpret the following character as if it were the seventh character on the line (that is, in column 7).

ENVIRONMENT VARIABLES

The following environment variables shall affect the execution of fort77:

LANG Provide a default value for the internationalization vari?
ables that are unset or null. (See the Base Definitions vol?

ume of POSIX.1?2017, Section 8.2, Internationalization Vari?
ables for the precedence of internationalization variables
used to determine the values of locale categories.)

LC_ALL If set to a non-empty string value, override the values of all the other internationalization variables.

LC_CTYPE Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments and input files).

LC_MESSAGES
Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard er?
ror.

NLSPATH Determine the location of message catalogs for the processing of LC_MESSAGES.

TMPDIR Determine the pathname that should override the default di?
rectory for temporary files, if any.

ASYNCHRONOUS EVENTS

Default.

STDOUT

Not used.

STDERR

The standard error shall be used only for diagnostic messages. If more than one file operand ending in .f (or possibly other unspecified suf?
fixes) is given, for each such file:

"%s:\n", <file>

may be written to allow identification of the diagnostic message with the appropriate input file.

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

OUTPUT FILES

Object files, listing files, and executable files shall be produced in unspecified formats.

EXTENDED DESCRIPTION

Standard Libraries

The fort77 utility shall recognize the following -l operand for the standard library:

-l f This library contains all functions referenced in the ANSI X3.9?1978 standard. This operand shall not be required to be present to cause a search of this library.

In the absence of options that inhibit invocation of the link editor, such as -c, the fort77 utility shall cause the equivalent of a -l f op? erand 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 library libf.a exists as a regular file.

The implementation may accept as -l operands names of objects that do not exist as regular files.

External Symbols

The FORTRAN compiler and link editor shall support the significance of external symbols up to a length of at least 31 bytes; case folding is permitted. The action taken upon encountering symbols exceeding the im? plementation-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 sym? bols total. A diagnostic message is written to standard output if the implementation-defined limit is exceeded; other actions are unspeci? fied.

EXIT STATUS

The following exit values shall be returned:

0 Successful compilation or link edit.

>0 An error occurred.

CONSEQUENCES OF ERRORS

When fort77 encounters a compilation error, it shall write a diagnostic to standard error and continue to compile other source code operands.

It shall return a non-zero exit status, but it is implementation-de?

fined whether an object module is created. If the link edit is unsuc?
cessful, a diagnostic message shall be written to standard error, and
fort77 shall exit with a non-zero status.

The following sections are informative.

APPLICATION USAGE

None.

EXAMPLES

The following usage example compiles xyz.f and creates the executable file foo:

```
fort77 -o foo xyz.f
```

The following example compiles xyz.f and creates the object file xyz.o:

```
fort77 -c xyz.f
```

The following example compiles xyz.f and creates the executable file a.out:

```
fort77 xyz.f
```

The following example compiles xyz.f, links it with b.o, and creates the executable a.out:

```
fort77 xyz.f b.o
```

RATIONALE

The name of this utility was chosen as fort77 to parallel the renaming of the C compiler. The name f77 was not chosen to avoid problems with historical implementations. The ANSI X3.9?1978 standard was selected as a normative reference because the ISO/IEC version of FORTRAN-77 has been superseded by the ISO/IEC 1539:1991 standard.

The file inclusion and symbol definition #define mechanisms used by the c99 utility were not included in this volume of POSIX.1?2017?even though they are commonly implemented?since there is no requirement that the FORTRAN compiler use the C preprocessor.

The -onetrip option was not included in this volume of POSIX.1?2017, even though many historical compilers support it, because it is derived from FORTRAN-66; it is an anachronism that should not be perpetuated.

Some implementations produce compilation listings. This aspect of FOR?TRAN has been left unspecified because there was controversy concerning

the various methods proposed for implementing it: a -V option over?
lapped with historical vendor practice and a naming convention of cre?
ating files with .I suffixes collided with historical lex file naming
practice.

There is no -I option in this version of this volume of POSIX.1?2017 to specify a directory for file inclusion. An INCLUDE directive has been a part of the Fortran-90 discussions, but an interface supporting that standard is not in the current scope.

It is noted that many FORTRAN compilers produce an object module even when compilation errors occur; during a subsequent compilation, the compiler may patch the object module rather than recompiling all the code. Consequently, it is left to the implementor whether or not an ob?
ject file is created.

A reference to MIL-STD-1753 was removed from an early proposal in re?
sponse to a request from the POSIX FORTRAN-binding standard developers. It was not the intention of the standard developers to require certifi?
cation of the FORTRAN compiler, and IEEE Std 1003.9?1992 does not spec?
ify the military standard or any special preprocessing requirements. Furthermore, use of that document would have been inappropriate for an international standard.

The specification of optimization has been subject to changes through early proposals. At one time, -O and -N were Booleans: optimize and do not optimize (with an unspecified default). Some historical practice led this to be changed to:

-O 0 No optimization.

-O 1 Some level of optimization.

-O n Other, unspecified levels of optimization.

It is not always clear whether ``good code generation'' is the same thing as optimization. Simple optimizations of local actions do not usually affect the semantics of a program. The -O 0 option has been in?
cluded to accommodate the very particular nature of scientific calcula?
tions in a highly optimized environment; compilers make errors. Some degree of optimization is expected, even if it is not documented here,

and the ability to shut it off completely could be important when port?
ing an application. An implementation may treat -O 0 as ``do less than
normal'' if it wishes, but this is only meaningful if any of the opera?
tions it performs can affect the semantics of a program. It is highly
dependent on the implementation whether doing less than normal is logi?
cal. It is not the intent of the -O 0 option to ask for inefficient
code generation, but rather to assure that any semantically visible op?
timization is suppressed.
The specification of standard library access is consistent with the C
compiler specification. Implementations are not required to have
/usr/lib/libf.a, as many historical implementations do, but if not they
are required to recognize f as a token.
External symbol size limits are in normative text; conforming applica?
tions need to know these limits. However, the minimum maximum symbol
length should be taken as a constraint on a conforming 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 section clearly specifies the behavior of
the compiler when compilation or link-edit errors 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 exe?
cutable file, then links it on disk with lseek()s and write()s, the
partially linked executable file 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 historical practice, a conforming application
shall rely on the exit status of fort77, rather than on the existence
or mode of the executable file.
The -g and -s options are not specified as mutually-exclusive. Histori?
cally, these two options have been mutually-exclusive, but because both

are so loosely specified, it seemed appropriate to leave their interac?
tion unspecified.

The requirement that conforming applications specify compiler options separately is to reserve the multi-character option name space for ven?
dor-specific compiler options, which are known to exist in many histor?
ical 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 would otherwise be difficult to associate an error message with the erring file. They are described with ``may'' to allow imple?
mentations to use other methods of identifying files and to parallel the description in c99.

FUTURE DIRECTIONS

Future versions of this standard may withdraw this utility. There are implementations of compilers that conform to much more recent versions of the FORTRAN programming language. Since there is no active FORTRAN binding to POSIX.1?2008, this standard does not need to specify any compiler.

SEE ALSO

ar, asa, c99, umask

The Base Definitions volume of POSIX.1?2017, Chapter 8, Environment Variables, Section 12.2, Utility Syntax Guidelines

The System Interfaces volume of POSIX.1?2017, exec

COPYRIGHT

is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.ker?nel.org/doc/man-pages/reporting_bugs.html .