NAME

lcc − ANSI/ISO C compiler

SYNOPSIS

lcc [*option* | *file*]. . .

DESCRIPTION

lcc is an ANSI/ISO C compiler for a variety of platforms.

Arguments whose names end with '.c' (plus '.C' under Windows) are taken to be C source programs; they are preprocessed, compiled, and each object program is left on the file whose name is that of the source with '.o' (UNIX) or '.obj' (Windows) substituted for the extension. Arguments whose names end with '.i' are treated similarly, except they are not preprocessed. In the same way, arguments ending with '.s' (plus '.S', '.asm', and '.ASM', under Windows) are taken to be assembly source programs and are assembled, producing an object file. If there are no arguments, **lcc** summarizes its options on the standard error.

lcc deletes an object file if and only if exactly one source file is mentioned and no other file (source, object, library) or **−l** option is mentioned.

If the environment variable **LCCINPUTS** is set, **lcc** assumes it gives a semicolon- or colon-separated list of directories in which to look for source and object files whose names do not begin with '/'. These directories are also added to the list of directories searched for libraries. If **LCCINPUTS** is defined, it must contain '.' in order for the current directory to be searched for input files.

lcc uses ANSI/ISO Standard header files (see 'FILES' below). Include files not found in the ANSI/ISO header files are taken from the normal default include areas, which usually includes */usr/include*. Under Windows, if the environment variable **INCLUDE** is defined, it gives a semicolon-separated list of directories in which to search for header files.

OPTIONS

lcc interprets the following options; unrecognized options are taken as loader options (see **ld**(1)) unless **−c**, **−S**, or **−E** precedes them. Except for **−l**, all options are processed before any of the files and apply to all of the files. Applicable options are passed to each compilation phase in the order given.

GNU/POSIX style **−−option** syntax is recognized as equivalent to **−option**.

−c Suppress the loading phase of the compilation, and force an object file to be produced even if only one program is compiled.

−copyright
Display copyright information on the standard output, and exit immediately with a success status code.

This option may be abbreviated to any unique prefix at least as long as **−co**.

−g Produce additional symbol table information for the local debuggers. **lcc** warns when **−g** is unsupported.

−Wf,−g*n*,*x*
Set the debugging level to *n* and emit source code as comments into the generated assembly code; *x* must be the assembly language comment character. If *n* is omitted, it defaults to 1, which is similar to **−g**. Omitting **,***x* just sets the debugging level to *n*.

−w Suppress warning diagnostics, such as those announcing unreferenced statics, locals, and parameters. The line *#pragma ref id* simulates a reference to the variable *id*.

−d *f* Generate jump tables for switches whose density is at least *f*, a floating-point constant between zero and one. The default is 0.5.

−A Warn about declarations and casts of function types without prototypes, assignments between pointers to **int**s and pointers to **enum**s, and conversions from pointers to smaller integral types.

A second **−A** warns about unrecognized control lines, nonANSI/ISO language extensions and source characters in literals, unreferenced variables and static functions, declaring arrays of incomplete types, and exceeding *some* ANSI/ISO environmental limits, such as more than 257 cases in

switches. It also arranges for duplicate global definitions in separately compiled files to cause loader errors.

−P Writes declarations for all defined globals on standard error. Function declarations include prototypes; editing this output can simplify conversion to ANSI/ISO C. This output may not correspond to the input when there are several **typedef**s for the same type.

−n Arrange for the compiler to produce code that tests for dereferencing zero pointers. The code reports the offending file and line number and calls **abort**(3).

−O This option is ignored (with a warning): **lcc** does not provide additional optimization levels.

−S Compile the named C programs, and leave the assembler-language output on corresponding files suffixed '.s' or '.asm'.

−E Run only the preprocessor on the named C programs and unsuffixed file arguments, and send the result to the standard output.

−o *output*
Name the output file *output*. If **−c** or **−S** is specified and there is exactly one source file, this option names the object or assembly file, respectively. Otherwise, this option names the final executable file generated by the loader, and '*a.out*' (UNIX) or '*a.exe*' (Windows) is left undisturbed. **lcc** warns if **−o** and **−c** or **−S** are given with more than one source file and ignores the **−o** option.

−D*name*=*def*
Define the *name* to the preprocessor, as if by '#define'. If =*def* is omitted, the name is defined as "1".

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

−I*dir* '#include' files whose names do not begin with '/' are always sought first in the directory of the *file* arguments, then in directories named in **−I** options, then in directories on a standard list.

−L*dir* Add directory *dir* to the library search path.

−l*name* Add library *name* to the list of libraries to be searched. The library is searched before all default libraries. On UNIX systems, this usually maps to the file *libname.a* (static linking), or *libname.so* (dynamic linking), found first in the library search path.

−dynamic
Request dynamic linking with shared libraries, instead of static linking. This is the default. This option is recognized on all platforms, but may be silently ignored on some.

−static Request static linking, instead of the default dynamic linking with shared libraries. This option is recognized on all platforms, but may be silently ignored on some.

−M Suppress normal preprocessor output, compilation, and linking, and produce *Makefile* object file dependencies instead.

If **−M** is given at least once, generate a list of dependencies on quoted header files (**#include** *"myfile.h"*), and write them on the standard output unit.

If **−M** is given more than once, the output also includes dependencies on angle-bracketed header files (**#include** <*sysfile.h*>).

−N Do not search *any* of the standard directories for '#include' files. Only those directories specified by subsequent explicit **−I** options will be searched, in the order given.

−B*str* Use the compiler component *str***rcc** instead of the default version. Note that *str* often requires a trailing slash.

This option is deprecated, and may disappear in a future release, since its job can be done by the **−Wo,−rcc=***path-to-alternate-rcc* option, one of four that permit changing any of the compiler's internally-invoked components.

On Sun Solaris systems only, **−Bstatic** and **−Bdynamic** are recognized and passed to the loader; see **ld**(1). They are provided only for compatibility with native compilers; the **lcc**-standard options **−dynamic** or **−static** should normally be used in their place.

−W*x**arg*
−W*x**,arg*

Pass argument *arg* to the program indicated by *x*; *x* can be one of **p**, **f** (or **0**), **a**, **l**, or **o**, which refer, respectively, to the preprocessor, the compiler proper, the assembler, the loader, or a system-specific option. *arg* is passed as given; if a − is expected, it must be given explicitly.

POSIX uses a comma in the fourth character; its use with **lcc** is optional. Documentation of other **−W** options here includes that comma.

−Wf,−a

Read a *prof.out* file from a previous execution and use the data therein to compute reference counts (see **−b**).

lcc assigns the most frequently-referenced scalar parameters and locals to registers whenever possible. For each block, explicit register declarations are obeyed first; remaining registers are assigned to automatic locals if they are 'referenced' at least 3 times. Each top-level occurrence of an identifier counts as 1 reference. Occurrences in a loop, either of the **then**/**else** arms of an **if** statement, or a **case** in a **switch** statement each count, respectively, as 10, 1/2, or 1/10 references. These values are adjusted accordingly for nested control structures.

−Wf,−C

Produce code to count the number of function calls. See also **−b**.

−Wf,−html

When used with **−Wf,−target=symbolic**, this option causes the text rendition to be emitted as strictly grammar-conformant HTML, complete with hypertext links for cross references in the code!

−Wf,−target=*architecture/os*

lcc is also a cross compiler; this option causes it to generate code for *architecture* running the operating system denoted by *os*. The supported *architecture*/*os* combinations may include

alpha/linux	Compaq/DEC Alpha, GNU/Linux
alpha/osf	Compaq/DEC Alpha, OSF/1 3.2, 4.x
mips/irix	big-endian MIPS Rx000, IRIX 5.2, 6.x
mips/linux	big-endian MIPS Rx000, GNU/Linux
mips/ultrix	little-endian MIPS Rx000, ULTRIX 4.3
null	no output
sparc/linux	Sun SPARC, GNU/Linux
sparc/solaris	Sun SPARC, Solaris 2.x
sparc/sun	Sun SPARC, SunOS 4.x
symbolic	text rendition of the generated code
symbolic/osf	text rendition of the generated code for Compaq/DEC OSF/1
symbolic/irix	text rendition of the generated code for SGI IRIX
x86/freebsd	Intel x86, FreeBSD
x86/linux	Intel x86, GNU/Linux
x86/solaris	Intel x86, Sun Solaris 2.x
x86/win32	Intel x86, Windows NT 4.0/Windows 95/98/2000

Additional combinations that may be supported in the future, if code-generation support is completed, include

ia64/freebsd	HP/Intel IA-64, FreeBSD
ia64/linux	HP/Intel IA-64, GNU/Linux
ia64/win32	HP/Intel IA-64, Windows NT 4.0/Windows 95/98/2000
ia64/win64	HP/Intel IA-64, Windows-64

parisc/hpux	HP PA-RISC, HP-UX 10.x, 11.x
parisc/linux	HP PA-RISC, GNU/Linux
ppc/aix	PowerPC, IBM AIX 4.x
ppc/linux	PowerPC, GNU/Linux
ppc/macosx	PowerPC, Mac OS X (Darwin, Rhapsody)

lcc can be built even on platforms for which no code generator is yet available. In this case, you can use it for fast syntax checking, e.g., with **−Wf,−target=null −S**, and for cross assembly. You can also use it to link existing object files produced by another C compiler.

For user convenience, when a **−Wf,−target=***xxx* option requests nonnative code generation, assembly and linking are automatically suppressed, as if an explicit **−S** option had been given. With **−Wf,−target=null**, no *.s* file is produced at all.

−Wf,−unsigned_char=1
−Wf,−unsigned_char=0

Make plain **char** an unsigned (1, or nonzero) or signed (0) type.

By default, **char** is signed on all platforms on which **lcc** runs. Note that this may differ from the choice made by other C compilers on the same platform.

You can test at runtime for the signedness of **char**: the expression *(***char***)(-1)* will be negative if **char** is signed, and positive if it is signed.

An alternate test, using the sign of the value of **SCHAR_MIN** from <*limits.h*>, is known to fail with other compilers on a few platforms because of implementation errors. **lcc** handles that test correctly.

Since preprocessor symbol values in <*limits.h*>, and possibly **typedef**s as well, may depend on the signedness of **char**, care should be taken to ensure that the same value of this option is used if preprocessing is done separately from compilation.

−Wf,−wchar_t=unsigned_char
−Wf,−wchar_t=unsigned_short
−Wf,−wchar_t=unsigned_int

Makes wide characters the type indicated.

By default, for **lcc**, wide characters are **unsigned short int**, and **wchar_t** is a typedef defined in <*stddef.h*>. The definition for **wchar_t** in <*stddef.h*> changes according to the setting of this option. For that reason, care should be taken to ensure that the same value of this option is used if preprocessing is done separately from compilation.

−Wo,−o32
−Wo,−32
−Wo,−n32
−Wo,−64

This option applies only to SGI IRIX systems, and specifies one of the four memory models supported. IRIX 5.x has only **−o32** and **−n32**, while IRIX 6.x treats **−32** as equivalent to **−o32**, and adds **−64**.

The default for both IRIX 5.x and 6.x is **−o32**.

lcc support for the **−n32** model is incomplete, so although the option is recognized, and accepted, the resulting code is likely to fail at assembly or link time.

lcc does not yet have 64-bit code-generation support, so although **−Wo,−64** is recognized, it raises an error.

For further description of these memory models, see **cc**(1).

−Wo,−as=*path-to-alternate-assembler*

Use the specified assembler, instead of the **lcc** default one.

The assembler can also be defined by setting the **LCCAS** environment variable, but any such value is overridden by an explicit command-line **−Wo,−as=***xxx* option.

−Wo,−cpp=*path-to-alternate-cpp*
Use the specified C preprocessor, instead of the **lcc** default one.

On many systems, the C preprocessor is not expected to be invoked directly by users, so it is not in the default *PATH*. Consequently, this option normally requires the full path to the C preprocessor.

The default **lcc** preprocessor strictly conforms to the ANSI/ISO C89 Standard, but some platforms have preprocessor language extensions in system header files, preventing their use by Standard-conformant preprocessors. The proper way to handle this is to write **lcc**-specific versions of those header files for installation in **lcc**'s own *include* directory. However, until that can be done, during installation and bootstrapping on a new system with extended header files, this option provides a way to use an extended preprocessor. Once **lcc** has been properly installed, ordinary user programs should *never* require this option.

The preprocessor can also be defined by setting the **LCCCPP** environment variable, but any such value is overridden by an explicit command-line **−Wo,−cpp=***xxx* option.

−Wo,−lccdir=*dir*
Find the preprocessor, compiler proper, and include directory in the directory *dir/* or *dir*\. If the environment variable **LCCDIR** is defined, it gives this directory. **lcc** warns when this option is unsupported.

−Wo,−ld=*path-to-alternate-ld*
Use the specified loader, instead of the **lcc** default one.

The preprocessor can also be defined by setting the **LCCLD** environment variable, but any such value is overridden by an explicit command-line **−Wo,−ld=***xxx* option.

Because **ld**(1) implementations vary so widely in their command-line options, and because **lcc** has a long built-in list of options to pass to the loader, it is unlikely that this option will work, unless you choose a loader implementation that is command-line compatible with the default one, which you can determine by examining the output from the **−v** option.

−Wo,−rcc=*path-to-alternate-rcc*
Use the specified lexer, parser and code-generator component, **rcc**(1), instead of the **lcc** default one.

The component can also be defined by setting the **LCCRCC** environment variable, but any such value is overridden by an explicit command-line **−Wo,−rcc=***xxx* option.

−v Print commands as they are executed; some of the executed programs are directed to print their version numbers. More than one occurrence of **−v** causes the commands to be printed, but *not* executed.

−version
Display version information on the standard output, and exit immediately with a success status code.

This option may be abbreviated to any unique prefix at least as long as **−ve**.

−help or **−?**
Print a message on the standard error summarizing *lcc*'s options and giving the values of the environment variables **LCCINPUTS** and **LCCDIR**, if they are defined. Under Windows, the values of **INCLUDE** and **LIB** are also given, if they are defined.

−b Produce code that counts the number of times each expression is executed. If loading takes place, arrange for a *prof.out* file to be written when the object program terminates. A listing annotated with execution counts can then be generated with **bprint**(1). **lcc** warns when **−b** is unsupported. **−Wf,−C** is similar, but counts only the number of function calls.

−p Produce code that counts the number of times each function is called. If loading takes place, replace the standard startup function by one that automatically calls **monitor**(3) at the start and arranges to write a *mon.out* file when the object program terminates normally. An execution profile can then be generated with **prof**(1). **lcc** warns when **−p** is unsupported.

−pg Causes the compiler to produce counting code like **−p**, but invokes a run-time recording mechanism that keeps more extensive statistics and produces a *gmon.out* file at normal termination. Also, a profiling library is searched, in lieu of the Standard C library. An execution profile can then be generated with **gprof**(1). **lcc** warns when **−pg** is unsupported.

−t*name*

−t Produce code to print the name of the function, an activation number, and the name and value of each argument at function entry. At function exit, produce code to print the name of the function, the activation number, and the return value. By default, *printf* does the printing; if *name* appears, it does. For null **char*** values, "(null)" is printed. **−target** *name* is accepted, but ignored.

−tempdir=*dir*

Store temporary files in the directory *dir/* or *dir*\. The default is usually */tmp*.

Other arguments are taken to be either loader option arguments, or C-compatible object programs, typically produced by an earlier **lcc** run, or perhaps libraries of C-compatible routines. Duplicate object files are ignored. These programs, together with the results of any compilations specified, are loaded (in the order given) to produce an executable program with name *a.out* (UNIX) or *a.exe* (Windows).

LIMITATIONS

lcc accepts the C programming language as described in the ANSI/ISO Standard. If **lcc** is used with the GNU C preprocessor, the **−Wp,−trigraphs** option is required to enable trigraph sequences.

Plain int bit fields are signed. Bit fields are aligned like unsigned integers but are otherwise laid out as by most Standard C compilers. Some compilers, such as the GNU C compiler, may choose other, incompatible layouts.

Likewise, calling conventions are intended to be compatible with the host C compiler, except possibly for passing and returning structures. Specifically, **lcc** passes and returns structures like host ANSI/ISO C compilers on most targets, but some older host C compilers use different conventions. Consequently, calls to/from such functions compiled with older C compilers may not work. Calling a function that returns a structure without declaring it as such violates the ANSI/ISO Standard and may cause a fault.

ENVIRONMENT VARIABLES

The behavior of **lcc** can be influenced by these environment variables:

LCCAS Path to alternate assembler. This can be overridden by an explicit command-line **−Wo,−as=***xxx* option.

LCCCPP Path to an alternate C preprocessor. This can be overridden by an explicit command-line **−Wo,−cpp=***xxx* option.

LCCDIR Alternate directory tree in which to find the preprocessor, compiler, and include directory. To see the default directory tree, run **lcc** with the **−v** option: **cpp**(1) and **rcc**(1) will share a common path that is the default.

LCCDIR is used before **LCCAS**, **LCCCPP**, **LCCLD**, and **LCCRCC**, so that they can override it.

LCCINPUTS Directory search path, separated by colons or semicolons, in which to look for source and object files whose names do not begin with '/'. These directories are also added to the list of directories searched for libraries.

LCCLD Path to an alternate loader. This can be overridden by an explicit command-line **−Wo,−ld=***xxx* option.

LCCRCC Path to an alternate lexer, parser, and code generator. This can be overridden by an explicit command-line **−Wo,−rcc=***xxx* option.

TEMP Directory in which temporary files should be written. If specified, this variable overrides the settings of any **TMPDIR** variable, but is itself overridden by **TMP**.

TMP Directory in which temporary files should be written. If specified, this variable overrides the settings of any **TEMP** or **TMPDIR** variables.

TMPDIR Directory in which temporary files should be written. This variable is overridden by both **TMP** and **TEMP**.

Under Microsoft Windows only, **lcc** recognizes two additional variables:

INCLUDE
Semicolon-separated list of directories in which to search for header files.

LIB Semicolon-separated list of libraries to search.

FILES

The file names listed below are *typical*, but vary among installations; installation-dependent variants can be displayed by running **lcc** with the **−v** option.

file.{c,C}	input file
file.{s,asm}	assembly-language file
file.{o,obj}	object file
a.{out,exe}	loaded output
*/tmp/lcc**	temporary files
$LCCDIR/cpp	preprocessor
$LCCDIR/rcc	compiler (lexer, parser, and code generator)
$LCCDIR/liblcc.{a,lib}	*lcc*-specific library
/lib/crt0.o	runtime startup (UNIX)
/lib/[gm]crt0.o	startups for profiling (UNIX)
/lib/libc.a	standard library (UNIX)
$LCCDIR/include	ANSI/ISO Standard headers
/usr/local/include	local headers
/usr/include	traditional headers
prof.out	file produced for **bprint**(1)
mon.out	file produced for **prof**(1)
gmon.out	file produced for **gprof**(1)

lcc predefines the macro **__LCC__** on all systems. It may also predefine some installation-dependent symbols; option **−v** exposes them. For a nice sorted list, try

csh/tcsh:
lcc -v -v -E foo.c |& tr ' ' '\n' | grep -e - | sort
sh/ksh/bash:
lcc -v -v -E foo.c 2>&1 | tr ' ' '\n' | grep -e - | sort

SEE ALSO

as(1), **bprint**(1), **c89**(1), **cc**(1), **collect2**(1), **cpp**(1), **gas**(1), **gcc**(1), **gprof**(1), **lcc-cpp**(1), **ld**(1), **pgcc**(1), **prof**(1), **rcc**(1).

C. W. Fraser and D. R. Hanson, *A Retargetable C Compiler: Design and Implementation*, Addison-Wesley, 1995. ISBN 0-8053-1670-1.

The World-Wide Web page at `http://www.cs.princeton.edu/software/lcc/`.

S. P. Harbison and G. L. Steele, Jr., *C: A Reference Manual*, 4th ed., Prentice-Hall, 1995.

B. W. Kernighan and D. M. Ritchie, *The C Programming Language*, 2nd ed., Prentice-Hall, 1988.

American National Standards Inst., *American National Standard for Information Systems—Programming Language—C*, ANSI X3.159-1989, New York, 1990.

International Organization for Standardization, *ISO/IEC 9899:1990: Programming languages — C*, Geneva, Switzerland, 1990.

BUGS

Mail bug reports along with the shortest preprocessed program that exposes them and the details reported by **lcc**'s **−v** option to `lcc-bugs@princeton.edu`. The World-Wide Web page at URL `http://www.cs.princeton.edu/software/lcc/` includes detailed instructions for reporting bugs.

The ANSI/ISO Standard headers conform to the specifications in the Standard, which may be too restrictive for some applications, but necessary for portability. Functions given in the ANSI headers may be missing from some local C libraries (e.g., wide-character functions) or may not correspond exactly to the local versions; for example, the ANSI/ISO Standard <*stdio.h*> specifies that *printf*, *fprintf*, and *sprintf* return the number of characters written to the file or array, but some existing libraries don't implement this convention.

On the MIPS and SPARC, old-style variadic functions must use <*varargs.h*> from MIPS or Sun. New-style is recommended.

With **−b**, files compiled *without* **−b** may cause *bprint* to print erroneous call graphs. For example, if **f** calls **g** calls **h** and **f** and **h** are compiled with **−b**, but **g** is not, **bprint**(1) will report that **f** called **h**. The total number of calls is correct, however.