LINT - C source code checker.

Syntax:

```
        tlint [file] [option]*

    (+|-)ControlComments (-)   (+|-)Declarations (-)
    (+|-)Keep (-)              (+|-)StandardLibrary (+)
    (+|-)Verbose (-)           (+|-)Wide (-)
    Configuration=file         CrossReference=keyword
    Define=name=value          Include=directory
    INSTallation=file          Library=lib
    libraryName=string         Output=file
    Output+=file               StandardInclude=directory
    StandardLibrary=name       Summary=file
    Target=keyword             indeX=file
```

Examples:

```
        tlint myfile
        tlint x=files o=incls ln=proj def=PROJECT=1
```

Options:

`file`

is a source file containing C code. The other options on the command line determine whether LINT scans this file for problem spots, or uses the file to create a summary or an entry in a LINT library.

`indeX=file`

gives the name of a text file. Each line in this text file should contain the name of one C source file. LINT checks each of these source files individually. In addition to a source file name, a line in an index file may specify `ControlComments`, `Configuration=`, `Include=` and `Define=` options. These options only apply to the source file named on the same line. For example, if the line

```
    myfile define=VERSION=1
```

appeared in an index file, LINT would use the given option when examining `myfile` but not for other files named in the index file. When processing an index line, LINT normally uses all the options appearing on the main command line, followed by the options specified on the index line. As a special case, however, `Include=` options on a line in an index file are used before any `Include=` options on the LINT command line. Only one `indeX=` option may appear on the command line.

`Configuration=file`

specifies a configuration file for LINT. For more information on configuration files, see below.

`+ControlComments`

indicates that control comments of the type

```
/*VARARGS*/
```

should be processed. The default is `-ControlComments`, in which case such comments are ignored. A `ControlComments` option on a line in an index file overrides any `ControlComments` option on the main LINT command line.

`CrossReference=keyword`

indicates that LINT output should include a cross-reference table that shows where symbols are defined, referenced, set, and used. (By default, cross-references are not provided.) Possible values for the `keyword` are:

`All` - cross-references all symbols;

`External` - only cross-references external symbols.

`+Declarations`

may be used when creating a LINT library or summary file. With this option, LINT treats every file scope declaration as if it were a definition for the variable, instead of a simple reference.

`Define=name=value`

has the same effect as

```
#define name value
```

in the C source code. The option indicates that `"name"` should be replaced with `"value"` (as text) wherever it appears in the source code being examined. If `"value"` contains blanks or tab characters, it should be enclosed in double or single quotes.

`Include=directory`

is the same as the `Include=` option for the C compiler. When LINT tries to find quoted include files, as in

```
#include "file"
```

it begins by searching the given directory for the file. If the file is not found there, LINT searches any directory named in `StandardInclude=` options, and finally searches the directory that contains the source file being examined. Any number of `Include=` options may be specified. Directories are searched in the order given on the command line.

`INSTallation=file`

tells LINT where to find the installation file. The installation file tells where various software components have been installed. For more information, see the section on *Installation Files* below.

If you do not specify an `INSTallation=` option on the command line, LINT checks for an environment variable named `LINT_INST` and uses its value as the name of the installation file. If this environment variable does not exist, LINT uses the default installation file.

`+Keep`
: does not delete intermediate files left by the preprocessor and the LINT steps.

`Library=lib`
: names a LINT library that should be used when checking source code. Any number of `Library=` options may appear on the command line. By default, LINT automatically includes LINT libraries of standard C functions (e.g. `printf`) so that such functions are always recognized.

`libraryName=string`
: is used when creating a LINT library. See below for more details.

`Output=file`
: is used when creating a LINT library or summary file (see below). The output overwrites the file's current contents.

`Output+=file`
: is the same as `Output=file`, except that output is written to the end of the current contents of the specified file, rather than overwriting what the file already holds.

`StandardInclude=directory`
: is the same as the `StandardInclude=` option for the C compiler. When LINT tries to find include files whose names are enclosed in angle brackets, as in

```
#include <file>
```

it begins by searching the given directory for the file. If the file is not found there, LINT searches directories named in `Include=` options, and finally searches the directory that contains the source file being examined. Any number of `StandardInclude=` options may be specified. Directories are searched in the order given on the command line.

`StandardLibrary=name`
: asks LINT to include the standard library indicated by "name". The following names are recognized:

`C` - the standard C library. This is included automatically, unless `-StandardLibrary` is specified.

`-StandardLibrary`
: does not include the standard `C` library routines.

`Summary=file`

states that the given file is a summary file that should be used as input to the LINT operation. See below for more on summary files.

`Target=keyword`

controls the kind of problem-checking you want to do. The following keywords are recognized.

`Host` - describes any problems that may arise if the program is run on the host machine (i.e. the machine where you are running LINT). This is the default.

`Extensions` - points out any extensions to C that might have been used, as well as any machine-dependencies in the code. It also describes the problems detected with `Target=Host`.

`Minimal` - points out any features that may cause problems if you port this program to a non-ANSI compiler. This will make note of all ANSI features used. It also describes the problems detected with `Target=Extensions`.

`+Verbose`

prints out the name of the file being examined. The default is `-Verbose`.

`+Wide`

prints output in a format that is 132 columns wide. The default is `-Wide`, which prints in an 80-column format.

Abbreviating Options:

The option keywords given above can be abbreviated by omitting any or all of the letters shown in lower case. For example, `CrossReference=All` may be abbreviated to

```
crossref=all
cref=all
cr=all
cr=a
```

and so on. When entering option keywords, you may type letters in upper, lower, or mixed case; the use of upper and lower case in this documentation is simply to show what letters are and aren't required.

Description:

LINT checks C source code for "irregularities": errors and non-portable constructions. Diagnostics from LINT normally take the form

```
filename,lineno: class: message
```

where "`filename`" is the name of the file that contained the problem and "`lineno`" is the number of the line within the file. The "class" describes what type of message this is: warning, error, etc. If a diagnostic refers to a function or external variable in a LINT library, the form of the diagnostic is

```
libname:message
```

where `"libname"` is the name of the LINT library.

Unless your program is very small, LINT will probably produce a great deal of output. You will usually find it convenient to redirect LINT's output into a file instead of letting it be displayed on the terminal screen. To do this, just put ">filename" on the LINT command line, as in

```
tlint myprog >output
```

This tells LINT to write its output into the given file.

Other Ways to Use LINT

In addition to checking C code for irregularities, LINT can:

(a) Extract a summary file of information that can be used in other LINT operations.

(b) Create a LINT library.

Summary files and LINT libraries are discussed below.

Summary Files

A summary file contains a "summary" of your source code: the names and types of all the external variables and functions defined or referenced in the code, plus any other information LINT may need when checking the code (such as the declared types of function parameters). Summary files also record any error messages that might be issued by the C compiler when parsing the source code. All of this information is stored in a special format that is not directly readable by humans.

To create a summary file from a normal source code file, use the `Output=file` option, as in

```
tlint src1.c output=summ
```

You can add more material to the same summary file using `Output+=file`, as in

```
tlint src2.c output+=summ
```

This appends new material to the existing contents of the summary file.

When LINT creates a Summary file, LINT only summarizes your source code and checks for syntax errors. It does not check for such problems as non-portable constructs or type mismatches. However, you can run a summary file through LINT again to do standard type-checking. For example,

```
tlint srcfile output=lf
tlint summ=lf
```

creates a summary file, then uses the information in the summary file to see if types match in symbol references and definitions.

The reason for creating summary files is the same as the reason for breaking up the source code of your program into several source files: it is easier to deal with source code in small pieces than in one big hunk. Some users may keep a summary file for every source file. If the code in one source file is changed, you can create a summary file from that source file, then run all the summary files of the program through LINT to see if the change has caused any problems. This is much faster than using LINT on all the raw source code.

LINT Libraries

A LINT library is similar to a summary file, in that it contains a summary of C source code. However, LINT libraries are intended to parallel the way that object libraries work.

To create a LINT library, specify both the `Output=` option and the `libraryName=` option on the LINT command line. For example,

```
tlint file output=mylib libname="abc"
```

creates a LINT library named `abc` in the file `mylib`. This name is used in LINT diagnostic messages related to symbols found in the LINT library. Using LINT in this way only generates minimal messages; full messages are printed when you actually use the LINT library.

To use a LINT library, specify the `Library=` option on the LINT command line, as in

```
tlint myfile library=mylib
```

When LINT finds that `myfile` contains a reference to an undefined symbol, LINT checks the information in the given LINT library to see if the symbol is defined there. This works just like compiling a module while referring to an object library. No error occurs if a symbol in the LINT library has the same name as a symbol in the source file; the source file symbol is the one that is used.

Functions should only be placed in a LINT library when you are sure they contain no errors. LINT does not look for errors when it is creating a LINT library.

Review of Inputs and Outputs

To specify a file as an input, simply give the file's name. To specify a summary file as input, use `Summary=file`. To specify a LINT library as input, use `Library=file`. LINT uses all of the contents of a source file or a summary file; it only uses library entries if they are referenced by source code or a summary file.

An index file contains partial command lines to be used by LINT in its operations. These command lines can name source files as input, but not summary files or LINT libraries.

When both an `Output=` and `libraryName=` option are specified, LINT creates a LINT library. If only `Output=` is specified, LINT creates a summary file. If `Output=` is not specified, LINT examines a source file or summary file for problem spots and writes its diagnostics to the standard output.

Multiple Definitions

If LINT finds two definitions for the same function or external variable, it always outputs a diagnostic message. If the two definitions are both in normal source code, LINT arbitrarily chooses the first definition it finds as the "correct" definition, and issues diagnostics for any later deviations from this definition. If one of the two definitions is in a LINT library and the other is in normal source code, LINT chooses the non-library version as the "correct" definition. In this way, a definition in normal source code overrides a library definition.

Configuration Files

A configuration file consists of a series of directives that control the behavior of LINT. The possible directives are explained below.

`define name string`

has the same format and purpose as a **`#define`** directive in normal C code. It creates a manifest or macro with the given value.

`inline includefile`

lets you simulate an include file. When LINT finds an `inline` directive, it begins to gather input lines up until the first line consisting of only a `'#'` character. For example, in

```
inline sim.h
extern int junk1;
extern int junk2;
#
```

LINT collects the two declarations and associates them with the name `"sim.h"`. If a program contains the directive,

```
#include <sim.h>
```

LINT takes the gathered text and includes it at that point in the program, as if it had come from an included file.

`map includename1 includename2`

says that all references of the form

```
#include includename1
```

should be converted to

```
#include includename2
```

`null includefile`

tells LINT to ignore all `#include` directives that attempt to include the specified file.

`search pathname`

is equivalent to `Include=pathname` on the LINT command line.

`system_search pathname`

is equivalent to `StandardInclude=pathname` on the LINT command line.

Installation Files:

An installation file specifies the pathnames for software and data files used by LINT. Installation files are text files made up of comment lines and option lines.

Comment lines:

Any line whose first non-blank character is `#` will be taken as a comment. Blank lines are also considered comments.

Option lines:

Option lines have the format

```
Keyword=pathname
```

In this documentation, keywords are written with some letters in upper case and some in lower case. You may abbreviate keywords by omitting any or all of the letters shown in lower case. The remaining letters may be entered in either upper or lower case; the documentation simply uses upper case to show which characters may not be omitted.

In this version of LINT, possible option lines are:

`Include=pathname`

gives the directory containing the LINT include files.

`Library=pathname`

gives the directory containing the LINT libraries.

`Program=pathname`

gives the directory containing the LINT executable files (CPP, and so on).