NAME

c_count − C-language line counter

USAGE

c_count *[options] [file-specifications]*

SYNOPSIS

C_count counts lines and statements in C-language source files. It provides related statistics on the amount of whitespace, comments and code. **C_count** also shows the presence of unbalanced (or nested) comments, unbalanced quotation marks and illegal characters.

DESCRIPTION

C_count reads one or more C-language source files and displays simple statistics about them. It counts statements (i.e., sequences of tokens terminated with semicolon) and measures the relative amount of commentary. **C_count** ignores semicolons where they appear in comments or in quoted literals.

A count of semicolons is a reasonable way of counting C statements. Note, however, that it does not count preprocessor definitions as C statements.

The statistics summary shows you the relative amount of commentary. This is the ratio of the alphanumeric characters in comments to the total of characters in the code (ignoring tabs and other whitespace which are not inside quotes). **C_count** only counts alphanumeric characters, thus ignoring punctuation used as fillers. It also suppresses RCS and DEC/CMS history comments from this ratio.

C_count counts the number of tokens (names and constants) in the source files and gives this total, as well as their average length.

C_count provides you not only with measurements, but also diagnostics in the form of a set of flags shown next to each file name:

" file contains an unbalanced quote (").

? file contains illegal characters (e.g., nonprinting, nonwhitespace characters such as an escape). **C_count** also flags tabs inside quotes. Depending on the context, these may indicate a problem with the source code.

***** file contains unterminated or nested comments. This usually indicates a problem with the source code.

\+ file contains unterminated curly-braces. This may be a problem with the source code, or a limitation of **c_count**. For example, the source may contain ifdef'd chunks with curly braces such as

```
#ifdef FOO
if (first_condition) {
#elif defined(BAR)
if (alternate_condition) {
#else
if (default_condition) {
#endif
```

If the closing curly brace is not ifdef'd to match, **c_count** will report the mismatch as an unterminated block.

\> file contains identifiers longer than specified by the **−w** option.

Unbalanced quotes may be legal. The C preprocessor permits you to define symbols which contain an unbalanced quote mark, for example:

```
#define     WARN  (void)printf("** warning:
      ...
      WARN item
```

Rather than duplicate all of the complexity of the C preprocessor, **c_count** permits you to specify symbols

which contain unbalanced quote marks.

OPTIONS

Command line options of **c_count** are:

−b display block-level statistics. The total number of top-level blocks (or statements), the maximum blocklevel, counting the top as 1, and the weighted average blocklevel for code.

−c display character-level statistics.

−d show each token as **c_count** parses it from the input stream.

−i display identifier-level statistics.

−j annotate summary in technical format (i.e., "physical source statements" and "logical source statements" for "lines" and "statements" respectively).

−l display line-level statistics.

−n suppress summary statistics.

−o *file* write the report to the specified file, rather than to the standard output.

−p display statistics on a per-file basis.

−q *define*

define tokens which may evaluate with an unbalanced quote mark '"'. For example,

```
−q WARN
```

tells **c_count** that the token "WARN" contains a quote mark.

−s display specialized statistics (e.g., code:comment ratio).

−t generate output in spreadsheet format (e.g., comma-separated columns). If you set any of the options "**−c**", "**−i**", "**−l**" or "**−s**", **c_count** generates these statistics. Otherwise it generates only the lines/statements.

−V print the version number.

−v direct **c_count** not only to print a summary line for each file, but also to print a running summary showing each source line, together with the current line and statement numbers, as well as the cumulative flags. Repeating the option causes **c_count** to also show block (curly-brace) levels.

−w *LEN*

specify the length for identifiers beyond which we should report an error. If this option is not given, **c_count** reports identifiers longer than 31 characters.

OPERATIONS

C_count reads one or more C language source files and writes its statistics to standard output. If you do not give any file names, **c_count** reads a list of file names from standard input.

The special filename "−" directs **c_count** to read the file itself from the standard input.

Following is an example of the use of **c_count**, showing the detailed types of information which it reports. The percentages add up to 100%, since overlapping data are discounted.

```
~/src/count (5) c_count *.[ch] */*.[ch]
  1165   418   |c_count.c
    17     0   |config.h
     1     0   |patchlev.h
   103     3   |system.h
    87    33   |porting/getopt.c
     8     4   |porting/getopt.h
   107    31   |porting/wildcard.c
     5     0?" |testing/test1.c
     6     2?  |testing/test2.c
    20     1   |testing/test3.c
----------------
  1519   492?"  total lines/statements

   228      lines had comments        15.0 %
     7      lines had history          0.5 %
    45      comments are inline       -3.0 %
   142      lines were blank           9.3 %
   170      lines for preprocessor    11.2 %
  1017      lines containing code     67.0 %
  1519      total lines              100.0 %

  6355      comment-chars             18.1 %
   105      history-chars              0.3 %
  1277      nontext-comment-chars      3.6 %
  7427      whitespace-chars          21.2 %
  2882      preprocessor-chars         8.2 %
 16984      statement-chars           48.5 %
 35030      total characters         100.0 %

  2698      tokens, average length 4.99

  0.32      ratio of comment:code
     3      ?:illegal characters found
     2      ":lines with unterminated quotes

    70      top-level blocks/statements
     7      maximum blocklevel
  2.67      ratio of blocklevel:code
```

If you use the "**−p**" option, **c_count** prints the detailed information for each file, as well as for all files together.

ENVIRONMENT

C_count runs in a POSIX environment. Execute it on VAX/VMS by defining it as a foreign command.

FILES

C_count is a single binary module, that uses no auxiliary files (e.g., **C_COUNT.EXE** on VAX/VMS).

AUTHOR

Thomas Dickey.

SEE ALSO

wc (1)