Complexity - Measure complexity of C source

For version 0.4, updated May 2011

Bruce Korb
bkorb@gnu.org

Table of Contents

1 Introduction **1**

1.1 Code Length 1
1.2 Switch Statement 1
1.3 Logic Conditions 1
1.4 Personal Experience 2
1.5 Rationale Summary 2

2 Complexity Computation **4**

2.1 Parsing Method 4
2.1.1 Complexity Measurement Parsing 4
2.1.2 Post-PreProcessing Parsing 4
2.1.3 During PreProcessing Parsing 5
2.1.4 `pmccabe` Parsing 5
2.2 Complexity Measurement Algorithm 5
2.3 Complexity Scores 5
2.4 Complexity Statistics 6
2.5 Scoring Adjustments 6

3 Example Output **8**

4 Invoking complexity **10**

4.1 complexity usage help (-?) 10
4.2 demi-nesting-penalty option 11
4.3 histogram option (-h) 11
4.4 horrid-threshold option 12
4.5 ignore option (-I) 12
4.6 input option (-i) 12
4.7 nesting-penalty option (-n) 12
4.8 no-header option (-H) 12
4.9 scale option (-s) 12
4.10 scores option (-c) 13
4.11 threshold option (-t) 13
4.12 trace option 13
4.13 unif-exe option 13
4.14 unifdef option (-u) 13
4.15 presetting/configuring complexity 14
4.16 complexity exit codes 15
4.17 complexity Description 15
4.18 complexity Bugs 15

Appendix A Copying This Manual **16**

Concept Index **24**

1 Introduction

Complexity measurement tools provide several pieces of information. They help to:

1. locate suspicious areas in unfamiliar code
2. get an idea of how much effort may be required to understand that code
3. get an idea of the effort required to test a code base
4. provide a reminder to yourself. You may see what you've written as obvious, but others may not. It is useful to have a hint about what code may seem harder to understand by others, and then decide if some rework may be in order.

But why another complexity analyzer? Even though the McCabe analysis tool already exists (`pmccabe`), I think the job it does is too rough for gauging complexity, though it is ideal for gauging the testing effort. Each code path should be tested and the `pmccabe` program provides a count of code paths. That, however, is not the only issue affecting human comprehension. This program attempts to take into account other factors that affect a human's ability to understand.

1.1 Code Length

Since `pmccabe` does not factor code length into its score, some folks have taken to saying either long functions or a high McCabe score find functions requiring attention. But it means looking at two factors without any visibility into how the length is obfuscating the code.

The technique used by this program is to count 1 for each line that a statement spans, plus the complexity score of control expressions (`for`, `while`, and `if` expressions). The value for a block of code is the sum of these multiplied by a nesting factor (see Section 4.7 [complexity nesting-penalty], page 12). This score is then added to the score of the encompassing block. With all other things equal, a procedure that is twice as long as another will have double the score. `pmccabe` scores them identically.

1.2 Switch Statement

`pmccabe` has changed the scoring of `switch` statements because they seemed too high. `switch` statements are now "free" in this new analysis. That's wrong, too. The code length needs to be counted and the code within a `switch` statement adds more to the difficulty of comprehension than code at a shallower logic level.

This program will multiply the score of the `switch` statement content by the See Section 4.7 [complexity nesting-penalty], page 12.

1.3 Logic Conditions

'`pmccabe`' does not score logic conditions very well. It overcharges for simple logical operations, it doesn't charge for comma operators, and it undercharges for mixing assignment operators and relational operators and the `and` and `or` logical operators.

For example:

```
xx = (A && B) || (C && D) || (E && F);
```

scores as `6`. Strictly speaking, there are, indeed, six code paths there. That is a fairly straight forward expression that is not nearly as complicated as this:

```
if (A) {
  if (B) {
    if (C) {
      if (D)
        a-b-c-and-d;
    } else if (E) {
        a-b-no_c-and-e;
    }
  }
}
```

and yet this scores exactly the same. This program reduces the cost to very little for a sequence of conditions at the same level. (That is, all `and` operators or all `or` operators.) so the raw score for these examples are 4 and 35, respectively (1 and 2 after scaling, see Section 4.9 [complexity scale], page 12).

If you nest boolean expressions, there is a little cost, assuming you parenthesize grouped expressions so that `and` and `or` operators do not appear at the same parenthesized level. Also assuming that you do not mix assignment and relational and boolean operators all together. If you do not parenthesize these into subexpressions, their small scores get multiplied in ways that sometimes wind up as a much higher score.

The intent here is to encourage easy to understand boolean expressions. This is done by,

- not combining them with assignment statements
- canonicalizing them (two level expressions with all `&&` operators at the bottom level and all `||` operators in the nested level -\- or vice versa)
- parenthesizing for visual clarity (relational operations parenthesized before being joined into larger `&&` or `||` expressions)
- breaking them up into multiple `if` statements, if convenient.

1.4 Personal Experience

I have used `pmccabe` on a number of occasions. For a first order approximation, it does okay. However, I was interested in zeroing in on the modules that needed the most help and there were a lot of modules needing help. I was finding I was looking at some functions where I ought to have been looking at others. So, I put this together to see if there was a better correlation between what seemed like hard code to me and the score derived by an analysis tool.

This has worked much better. I ran `complexity` and `pmccabe` against several million lines of code. I correlated the scores. Where the two tools disagreed noticeably in relative ranking, I took a closer look. I found that '`complexity`' did, indeed, seem to be more appropriate in its scoring.

1.5 Rationale Summary

Ultimately, complexity is in the eye of the beholder and, even, the particular mood of the beholder, too. It is difficult to tune a tool to properly accommodate these variables.

`complexity` will readily score as zero functions that are extremely simple, and code that is long with many levels of logic nesting will wind up scoring much higher than with `pmccabe`, barring extreme changes to the default values for the tunables.

I have included several adjustments so that scores can be tweaked to suit personal taste or gathered experience. (See Section 4.7 [complexity nesting-penalty], page 12, and Section 4.2 [complexity demi-nesting-penalty], page 11, but also See Section 4.9 [complexity scale], page 12, to adjust scores to approximate scores rendered by `pmccabe`).

2 Complexity Computation

The principal goal Fundamentally, this program counts lines of non-comment source lines, multiplies by a "nesting factor" for each level of logic nesting and divides by a scaling factor so that the typical results lie roughly in the same range as `pmccabe` results. That happens to be approximately 20.

2.1 Parsing Method

The method chosen for parsing the source has an effect on what gets seen (scored) by the program.

2.1.1 Complexity Measurement Parsing

This program examines the actual source a human looks at when the file is opened, provided it is not pre-processed by `unifdef`, See Section 4.14 [complexity unifdef], page 13. This was chosen because uncompiled code adds to the complexity of what a human must understand. However, sometimes the source will contain unbalanced braces a la:

```
#if FOO
  for (int ix = foo;;) {
#else
  for (int ix = bar;;) {
#endif
    code...
  }
```

rendering code that cannot be parsed correctly. `unifdef`-ing makes it parsable. Unfortunately, because the practice of `ifdef`-ing unbalanced curly braces is so common, this program cannot rely on finding the correct closing brace.

CAVEAT: for the purposes of this program, procedures end when either a matching closing brace is found *or* a closing curly brace is found in column 1, whichever comes first. If the closing brace in column one does not match the procedure opening brace, the procedure is considered unscorable.

Fortunately, unscorable procedures are relatively unusual.

CAVEAT2: K&R procedure headers are not recognized. If anything other than an opening curly brace appears after the parameter list will cause the code recognizer to go back into "look for a procedure header" mode. K&R procedures are not just not scored, they are completely ignored.

This should probably get fixed, though.

2.1.2 Post-PreProcessing Parsing

Another approach would be to use the `C` compiler and analize the tokens coming out of the preprocessor. The drawbacks are that macro expansions will add to the complexity, even though they do not add to human perceived complexity, and uncompiled code do not add to the complexity measure. The benefit, of course, is that you know for certain where a procedure body starts and ends.

2.1.3 During PreProcessing Parsing

This would require going into the C preprocessor code and cause macros to not be expanded. Again, the great benefit is that you know for certain you can find the starting and ending braces for every procedure body. The downsides are the extra work and, again, the uncompiled code won't get counted in the complexity measure.

This might be a useful exercise to do some day, just to see how helpful it might be. Being able to recognize all procedure bodies without fail would be a good thing.

2.1.4 `pmccabe` Parsing

The `pmccabe` parsing actually inspired the method for this program. Thd difference is that `pmccabe` will always keep scanning until a procedure body's closing curly brace is found, even if that means counting the code from several following procedure definitions. The consequence of this is that this program's code will see some procedures that `pmccabe` will not, and vice versa.

2.2 Complexity Measurement Algorithm

Fundamentally, this program counts non-comment source lines and examines elements of parenthesized expressions. This score is multiplied by a nesting scoring factor for each layer of code nesting.

A parenthesized expression is scanned for operators. If they are all arithmetic operators, or all arithmetic and one relational operator, the score is zero. If all the operators are boolean `and`s or they are all `or`s, then the score is one. An assignment operator with arithmetic operators also scores one. If you mix relational operators and all `and`s or all `or`s, the score is the number of boolean elements. If you mix `and`s and `or`s at the same parenthetical level, the two counts are multiplied, unless the boolean element count is higher.

Fundamentally, do not use multiple relational or boolean operators at the same parenthetical level, unless they are all boolean `and`s or they are all boolean `or`s. If you use boolean operators and relational operators in one expression, you are charged one statement for each boolean element.

After scoring each statement and any parenthesized expressions, the score is multiplied by any encompassing controlled block and added to the score of that block. A "controlled block" is a curly-braced collection of statements controlled by one of the statement controlling statements `do`, `for`, `else`, `if`, `switch`, or `while`. Stand alone blocks for scoping local variables do not trigger the multiplier.

You may trace the scores of parenthesized expressions and code blocks (see Section 4.12 [complexity trace], page 13). You will see the raw score of the code block or expression.

The final score is the outermost score divided by the "scaling factor", See Section 4.9 [complexity scale], page 12.

2.3 Complexity Scores

The "Complexity Scores" table shows the score of each procedure identified that also exceeded the threshold score, See Section 4.11 [complexity threshold], page 13. The entries on each line are:

- The computed score

- The number of lines between the opening and closing curly braces
- The number of non-comment, non-blank lines found there
- The name of the source file
- The line number of the opening curly brace
- The name of the procedure

The output is sorted by the score and then the number of non-comment lines. Procedures with scores below the threshold are not displayed.

2.4 Complexity Statistics

The statistics are displayed both as a table and as a histogram, See Chapter 3 [Example Output], page 8. It is under the control of the Section 4.3 [complexity histogram], page 11 option. The statistics are for each non-comment source line and each source line is given the score of its encompassing procedure. This way, larger procedures are given proportionally more weight than one line procedures.

The histogram is broken up into three ranges. Scores of 0 through 99 are displayed in 10 point groupings, 100 through 999 in 100 point groupings and 1000 and above (good grief!!, but they exist) are in 1000 point groupings. The number of asterisks represent the number of lines of code that are in procedures that score in the specified range.

The tabular statistics are also based on lines, not procedures.

'`Average line score`'
: This is the procedure score times the non-comment line count, all added up and divided by the total non-comment source lines found.

'`25%-ile score`'
'`50%-ile score`'
'`75%-ile score`'
'`Highest score`'
: Since the distribution of scores is nothing like a bell curve, the mean and standard deviation do not give a very clear picture of the distribution of the scores. Typically, the standard deviation is larger than the average score. So, instead the program prints the the four quartile scores. The score for which 25, 50, and 75 percent of code is scored less than, plus the highest scoring procedure (100 percent of code scores less than or equal to that score).

'`Unscored procedures`'
: If any procedures were found that could not be scored, the number of such procedures is printed.

2.5 Scoring Adjustments

Scores can be adjusted with three different options:

'`nesting-penalty`'
: See Section 4.7 [complexity nesting-penalty], page 12.

'`demi-nesting-penalty`'
: See Section 4.2 [complexity demi-nesting-penalty], page 11.

'`scale`' See Section 4.9 [complexity scale], page 12.

The raw score is the number of lines or statements, whichever is greater, adjusted by a factor for the depth of the logic. Statements are nested when they are inside of a block of statements for a "block" statement (viz., "do", "for", "if", "switch" or "while"). Statements within blocks used to constrain the scope of variables (not controlled by a block statement) are not multiplied by this factor.

Expressions are nested when contained within parentheses. The *cost* of these is different. Block level nesting multiplies the score for the block by the `--nesting-penalty` factor (2.0 by default). Nested expressions are multiplied by the `--demi-nesting-penalty`, the square root of `--nesting-penalty` by default.

Some attempt is made to judge the complexity of an expression. A complicated expression is one that contains an assignment operator, more than one relation operator, or a mixture of "and" and "or" operators with any other different kind of non-arithmetic operator. Expression scores are minimized by:

- Doing assignments outside of boolean expressions, or at least parenthesizing them.
- Parenthesizing each relationship operation in an expression of multiple "and" and/or "or" operations. Yes, precedence parses them correctly, but it is less clear.
- Parenthesizing groups of "and" and "or" operations so that operators of only one type appear at one level. For example, the first expression below instead of the second. Yes, precedence means the effect is the same, but we're after code clarity so that correctness is more obvious.

  ```
  1: ((a && b) || (c && d))
  2: (a && b || c && d)
  ```

  The first adds 2 to the raw score (before dividing by the scaling factor). The latter will add 5, assuming a `demi-nesting-penalty` of `1.41`.

3 Example Output

This is a self-referential example. This output was obtained by going into the complexity source directory and running the command:

```
complexity --histogram --score --thresh=3 '*.c'
```

The `--threshold` is set to three because all of the functions score below the default threshold of 30. It is not zero because there are too many trivial (0, 1 or 2) functions for a short example.

This results in:

```
Complexity Scores
Score | ln-ct | nc-lns| file-name(line): proc-name
    3      13      12   tokenize.c(507): skip_params
    3      15      13   score.c(121): handle_stmt_block
    3      22      17   tokenize.c(64): check_quote
    3      34      27   score.c(736): score_proc
    3      37      27   complexity.c(72): initialize
    3      43      35   score.c(513): handle_TKN_KW_DO
    4      15      13   tokenize.c(28): skip_comment
    4      20      16   tokenize.c(405): next_nonblank
    4      22      18   tokenize.c(528): skip_to_semi
    4      35      28   tokenize.c(335): keyword_check
    4      52      40   complexity.c(360): load_file
    5      33      28   score.c(466): handle_TKN_KW_CASE
    5      35      30   score.c(315): handle_parms
    5      58      41   complexity.c(297): popen_unifdef
    5      59      48   score.c(160): fiddle_subexpr_score
    5      78      58   score.c(561): handle_TKN_KW_IF
    6      45      30   opts.c(776): translate_option_strings
    6      67      44   opts.c(670): main
    7      49      40   score.c(644): handle_TKN_KW_FOR
   10      84      65   complexity.c(441): complex_eval
   11      57      50   tokenize.c(555): find_proc_start
   11      88      65   complexity.c(123): print_histogram
   12      81      60   score.c(224): handle_subexpr
   12      92      68   score.c(355): handle_expression
   15      64      51   tokenize.c(103): hash_check
   25      72      60   tokenize.c(430): next_token

Complexity Histogram
Score-Range  Lin-Ct
    0-9         565 **********************************************
   10-19        359 **************************************
   20-29         60 ******

Scored procedure ct:       26
Non-comment line ct:      984
```

```
Average line score:          8
25%-ile score:               4 (75% in higher score procs)
50%-ile score:               6 (half in higher score procs)
75%-ile score:              11 (25% in higher score procs)
Highest score:              25 (next_token() in tokenize.c)
```

4 Invoking complexity

Compute the complexity of source code not just with a path-through-the-code count, but also amplifying line counts by logic level nesting. `complexity` ignores all cpp preprocessor directives - calculating the complexity of the appearance of the code, rather than the complexity after the preprocessor manipulates the code. `getchar(3)`, for example, will expand into quite complicated code.

This chapter was generated by **AutoGen**, using the `agtexi-cmd` template and the option descriptions for the `complexity` program.

This software is released under the GNU General Public License.

4.1 complexity usage help (-?)

This is the automatically generated usage text for complexity:

```
complexity (GNU Complexity) - Measure complexity of C source - Ver. 0.4
USAGE:  complexity [ -<flag> [<val>] | --<name>[{=| }<val>] ]... \
                [ <file-name> ... ]

   -t, --threshold=num        Reporting threshold
       --horrid-threshold=num zero exit threshold
   -n, --nesting-penalty=str  score multiplier for nested code
       --demi-nesting-penalty=str score multiplier for nested expressions
   -s, --scale=num            complexity scaling factor
   -h, --histogram            Display histogram of complexity numbers
                                - disabled as --no-histogram
                                - may not be preset
   -c, --scores               Display the score for each procedure
                                - disabled as --no-scores
                                - may not be preset
   -I, --ignore=str           procedure name to be ignored
                                - may appear multiple times
   -H, --no-header            do not print scoring header
                                - may not be preset
   -u, --unifdef=str          Run the source(s) through unifdef(1BSD)
                                - may appear multiple times
       --unif-exe=str         Specify the unifdef program
   -i, --input=str            file of file list
       --trace=str            trace output file
   -v, --version[=arg]        Output version information and exit
   -?, --help                 Display extended usage information and exit
   ->, --save-opts[=arg]      Save the option state to a config file
   -<, --load-opts=str        Load options from a config file
                                - disabled as --no-load-opts
                                - may appear multiple times

Options are specified by doubled hyphens and their name or by a single
hyphen and the flag character.
```

```
Compute the complexity of source code not just with a
path-through-the-code count, but also amplifying line counts by logic
level nesting.
If no arguments are provided, input arguments are read from stdin,
one per line; blank and '#'-prefixed lines are comments.
'stdin' may not be a terminal (tty).

The following option preset mechanisms are supported:
 - reading file $@/complex.conf
 - reading file $HOME/.complexityrc
 - reading file $PROJECT_ROOT/complex.conf
 - reading file ./.complexityrc
 - examining environment variables named COMPLEXITY_*

``complexity'' ignores all cpp preprocessor directives - calculating
the complexity of the appearance of the code, rather than the complexity
after the preprocessor manipulates the code.  ``getchar(3)'', for example,
will expand into quite complicated code.

please send bug reports to:  bkorb@gnu.org
```

4.2 demi-nesting-penalty option

This is the "score multiplier for nested expressions" option. By default, this value is halfway between 1.0 and the nesting penalty (specifically, the square root of the nesting penalty). It refers to a parenthesized sub-expression. e.g.

```
((a > b) && (c > d))
```

contains two parenthesized sub-expressions. This would count 3.5 points. On the other hand, this:

```
(a > b && c > d)
```

contains two relation operators and a logical operator at the same level. These nested counts will be multiplied together and yield `2.5 * 2.5`, or `6.25`. Don't do that. It gets even worse if you have logical ands and ors at the same level.

4.3 histogram option (-h)

This is the "display histogram of complexity numbers" option.

This option has some usage constraints. It:

- may not be preset with environment variables or configuration (rc/ini) files.

Instead of printing out each function's score, a summary is printed at the end showing how many functions had particular ranges of scores. Unless `--scores` is specifically called out, the scores will not print with this option specified. The minimum scoring threshold will also be reduced to zero (0), unless `--threshold` is specified.

4.4 horrid-threshold option

This is the "zero exit threshold" option. If any procedures score higher than this threshold, then the program will exit non-zero. (`4/COMPLEX_EXIT_HORRID_FUNCTION`, if no other problems are encountered.) By default, this program exits zero unless one function exceeds the horrid score of 100.

4.5 ignore option (-I)

This is the "procedure name to be ignored" option.

This option has some usage constraints. It:

- may appear an unlimited number of times.

Some code has macros defined that confuse the lexical analysis. This will cause them to be ignored. Other ways to cause functions to be ignored are:

1. Use K&R syntax for a procedure header.
2. Use a preprocessing macro to assemble the procedure header.
3. Simplify your code.

Generally speaking, anything you do that alters normal C syntax will confuse the lexical analysis. If a procedure is not seen, then it will not get counted. If code within a procedure is incomprehensible, you will likely get inappropriate results.

4.6 input option (-i)

This is the "file of file list" option. Instead of either a command line list of input files or reading them from standard input, read the list of files from this file.

4.7 nesting-penalty option (-n)

This is the "score multiplier for nested code" option. Linguistic constructs weigh more heavily the more deeply nested they are. By default, each layer penalizes by a factor of 2.0. The option argument is a floating point number. The penalty may be 1, but not less.

4.8 no-header option (-H)

This is the "do not print scoring header" option.

This option has some usage constraints. It:

- may not be preset with environment variables or configuration (rc/ini) files.

If a script is going to process the scoring output, parsing is easier without a header. The histogram output will always have a header.

4.9 scale option (-s)

This is the "complexity scaling factor" option. By default, the scaling is 20 which divides the raw score by 20. This was normalized to roughly correspond to the `pmccabe` scores:

'`0-9`' Easily maintained code.

'`10-19`' Maintained with little trouble.

'20-29' Maintained with some effort.

'30-39' Difficult to maintain code.

'40-49' Hard to maintain code.

'50-99' Unmaintainable code.

'100-199' Crazy making difficult code.

'200+' I only wish I were kidding.

```
Score | ln-ct | nc-lns| file-name(line): proc-name
 4707    3815    2838   lib/vasnprintf.c(1747): VASNPRINTF
```

4.10 scores option (-c)

This is the "display the score for each procedure" option.

This option has some usage constraints. It:

- may not be preset with environment variables or configuration (rc/ini) files.

If you specify `--histogram`, individual scores will not be displayed, unless this option is specified.

4.11 threshold option (-t)

This is the "reporting threshold" option. Ignore any procedures with a complexity measure below this threshold. By default, a complexity score of under 30 is not printed. However, if a histogram and statistics are to be printed, but not individual procedure scores, then the default is set to zero. Procedures below this limit are not counted in the statistics.

4.12 trace option

This is the "trace output file" option. Print intermediate scores to a trace file.

4.13 unif-exe option

This is the "specify the unifdef program" option. Alternate program to use for unifdef-ing the input.

4.14 unifdef option (-u)

This is the "run the source(s) through unifdef(1bsd)" option.

This option has some usage constraints. It:

- may appear an unlimited number of times.

Strip out sections of code surrounded by `#if/#endif` directives. The option argument is passed as an argument to the '`unifdef(1BSD)`' program. For example:

```
complexity -u-Dsymbol
```

would cause `symbol` to be defined and remove sections of code preceded by `#ifndef symbol` directives.

Please see the '`unifdef`' documentation for more information.

4.15 presetting/configuring complexity

Any option that is not marked as *not presettable* may be preset by loading values from configuration ("rc" or "ini") files, and values from environment variables named COMPLEXITY and COMPLEXITY_<OPTION_NAME>. "<OPTION_NAME>" must be one of the options listed above in upper case and segmented with underscores. The COMPLEXITY variable will be tokenized and parsed like the command line. The remaining variables are tested for existence and their values are treated like option arguments.

libopts will search in 4 places for configuration files:

- $(pkgdatadir)/complex.conf
- $HOME
- $PROJECT_ROOT/complex.conf
- $PWD

The value for $(pkgdatadir) is recorded at package configure time and replaced by 'libopts' when 'complexity' runs. The environment variables HOME, PROJECT_ROOT, and PWD are expanded and replaced when 'complexity' runs. For any of these that are plain files, they are simply processed. For any that are directories, then a file named '.complexityrc' is searched for within that directory and processed.

Configuration files may be in a wide variety of formats. The basic format is an option name followed by a value (argument) on the same line. Values may be separated from the option name with a colon, equal sign or simply white space. Values may be continued across multiple lines by escaping the newline with a backslash.

Multiple programs may also share the same initialization file. Common options are collected at the top, followed by program specific segments. The segments are separated by lines like:

```
[COMPLEXITY]
```

or by

```
<?program complexity>
```

Do not mix these within one configuration file.

Compound values and carefully constructed string values may also be specified using XML syntax:

```
<option-name>
   <sub-opt>...&lt;...&gt;...</sub-opt>
</option-name>
```

yielding an option-name.sub-opt string value of

```
"...<...>..."
```

AutoOpts does not track suboptions. You simply note that it is a hierarchicly valued option. AutoOpts does provide a means for searching the associated name/value pair list (see: optionFindValue).

4.16 complexity exit codes

One of the following exit values will be returned:

'0' Successful program execution.

'1' The operation failed or the command syntax was not valid.

'3' insufficient memory to run program

'4' One or more functions scored over 100

'5' No qualifying procedures were found.

'32' one or more input files were unreadable or empty.

4.17 complexity Description

The weight of each statement is the number of lines the statement uses. This value is multiplied by the nested logic weighting (2.0 by default) for each layer of logic. For example, this snippet:

```
if (foo) {
  if (bar) {
    bumble; baz;
  }
}
```

will score 11. This score is then scaled to approximate `pmccabe` results by dividing by 20 and rounding. This scores "1" at the end. `pmccabe` scores higher on simple procedures and `complexity` scores higher with more deeply nested logic.

The scoring can be tweaked by adjusting the `--nesting-penalty` and `--scale`-ing factors. The default values were calibrated by comparing the average results of millions of lines of code with the results of `pmccabe`.

For the purposes of this program, a procedure is identified by a name followed by a parenthesized expression and then an opening curly brace. It ends with a closing curly brace in column 1.

4.18 complexity Bugs

This program does not recognize K&R procedure headers.

Some procedures still get missed. Usually, these are procedures that use the C preprocessor to extend the C language in some way.

Appendix A Copying This Manual

You may copy this manual under the terms of the FDL (the GNU Free Documentation License).

0. PREAMBLE

   The purpose of this License is to make a manual, textbook, or other functional and useful document *free* in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

   This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

   We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

   This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you". You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

   A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

   A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

The "publisher" means any person or entity that distributes copies of the Document to the public.

A section "Entitled XYZ" means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as "Acknowledgements", "Dedications", "Endorsements", or "History".) To "Preserve the Title" of such a section when you modify the Document means that it remains a section "Entitled XYZ" according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

2. VERBATIM COPYING

   You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

   You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

   If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

   If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

   If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

   It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

   You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.
B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.
C. State on the Title page the name of the publisher of the Modified Version, as the publisher.
D. Preserve all the copyright notices of the Document.
E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.
F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.
G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.
H. Include an unaltered copy of this License.
I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.
J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.
K. For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.
L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.
M. Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version.
N. Do not retitle any existing section to be Entitled "Endorsements" or to conflict in title with any Invariant Section.
O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section Entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties—for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

   You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

   The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

   In the combination, you must combine any sections Entitled "History" in the various original documents, forming one section Entitled "History"; likewise combine any sections Entitled "Acknowledgements", and any sections Entitled "Dedications". You must delete all sections Entitled "Endorsements."

6. COLLECTIONS OF DOCUMENTS

   You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

   You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

   A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an "aggregate" if the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

   If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

8. TRANSLATION

   Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

   If a section in the Document is Entitled "Acknowledgements", "Dedications", or "History", the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

9. TERMINATION

   You may not copy, modify, sublicense, or distribute the Document except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, or distribute it is void, and will automatically terminate your rights under this License.

   However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

   Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

   Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, receipt of a copy of some or all of the same material does not give you any rights to use it.

10. FUTURE REVISIONS OF THIS LICENSE

    The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See http://www.gnu.org/copyleft/.

    Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation. If the Document specifies that a proxy can decide which future versions of this License can be used, that proxy's public statement of acceptance of a version permanently authorizes you to choose that version for the Document.

11. RELICENSING

    "Massive Multiauthor Collaboration Site" (or "MMC Site") means any World Wide Web server that publishes copyrightable works and also provides prominent facilities for anybody to edit those works. A public wiki that anybody can edit is an example of such a server. A "Massive Multiauthor Collaboration" (or "MMC") contained in the site means any set of copyrightable works thus published on the MMC site.

    "CC-BY-SA" means the Creative Commons Attribution-Share Alike 3.0 license published by Creative Commons Corporation, a not-for-profit corporation with a principal place of business in San Francisco, California, as well as future copyleft versions of that license published by that same organization.

    "Incorporate" means to publish or republish a Document, in whole or in part, as part of another Document.

    An MMC is "eligible for relicensing" if it is licensed under this License, and if all works that were first published under this License somewhere other than this MMC, and subsequently incorporated in whole or in part into the MMC, (1) had no cover texts or invariant sections, and (2) were thus incorporated prior to November 1, 2008.

    The operator of an MMC Site may republish an MMC contained in the site under CC-BY-SA on the same site at any time before August 1, 2009, provided the MMC is eligible for relicensing.

ADDENDUM: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

```
Copyright (C)  year  your name.
Permission is granted to copy, distribute and/or modify this document
under the terms of the GNU Free Documentation License, Version 1.3
or any later version published by the Free Software Foundation;
with no Invariant Sections, no Front-Cover Texts, and no Back-Cover
Texts.  A copy of the license is included in the section entitled ``GNU
Free Documentation License''.
```

If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts, replace the "with. . . Texts." line with this:

```
with the Invariant Sections being list their titles, with
the Front-Cover Texts being list, and with the Back-Cover Texts
being list.
```

If you have Invariant Sections without Cover Texts, or some other combination of the three, merge those two alternatives to suit the situation.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.

Concept Index

C

`complexity` 10
Complexity Computation 4
complexity usage 10
complexity-demi-nesting-penalty 11
complexity-histogram 11
complexity-horrid-threshold 12
complexity-ignore 12
complexity-input 12
complexity-nesting-penalty 12
complexity-no-header 12
complexity-scale 12
complexity-scores 13
complexity-threshold 13
complexity-trace 13
complexity-unif-exe 13
complexity-unifdef 13

E

Example Output 8

F

FDL, GNU Free Documentation License 16

I

Introduction 1

M

Measure complexity of C source 10

P

parsing 4

S

scores 5, 6
statistics 6

T

tuning 6