Indent

Format C Code

Edition 0.02, for Indent Version 1.3
May 1992

Jim Kingdon
Joseph Arceneaux

1 The indent Program

The indent program changes the appearance of a C program by inserting or deleting whitespace. It can be used to make code easier to read. It can also convert from one style of writing C to another.

indent understands a substantial amount about the syntax of C, but it also attempts to cope with incomplete and misformed syntax.

In version 1.2 and more recent versions, the GNU style of indenting is the default.

1.1 Invoking indent

As of version 1.3, the format of the indent command is:

indent [*options*] [*input-files*]

indent [*options*] [*single-input-file*] [-o *output-file*]

This format is different from earlier versions and other versions of indent.

In the first form, one or more input files are specified. indent makes a backup copy of each file, and the original file is replaced with its indented version. See Section 1.2 [Backup files], page 3, for an explanation of how backups are made.

In the second form, only one input file is specified. In this case, or when the standard input is used, you may specify an output file after the '-o' option.

To cause indent to write to standard output, use the '-st' option. This is only allowed when there is only one input file, or when the standard input is used.

If no input files are named, the standard input is read for input. Also, if a filename named '-' is specified, then the standard input is read.

As an example, each of the following commands will input the program 'slithy_toves.c' and write its indented text to 'slithy_toves.out':

```
indent slithy_toves.c -o slithy_toves.out

indent -st slithy_toves.c > slithy_toves.out

cat slithy_toves.c | indent -o slithy_toves.out
```

Most other options to `indent` control how programs are formatted. As of version 1.2, `indent` also recognizes a long name for each option name. Long options are prefixed by either '`--`' or '`+`'.[1] In most of this document, the traditional, short names are used for the sake of brevity. See Appendix A [Option Summary], page 15, for a list of options, including both long and short names.

Here is another example:

```
indent -br test/metabolism.c -l85
```

This will indent the program '`test/metabolism.c`' using the '`-br`' and '`-l85`' options, write the output back to '`test/metabolism.c`', and write the original contents of '`test/metabolism.c`' to a backup file in the directory '`test`'.

Equivalent invocations using long option names for this example would be:

```
indent --braces-on-if-line --line-length185 test/metabolism.c

indent +braces-on-if-line +line-length185 test/metabolism.c
```

If you find that you often use `indent` with the same options, you may put those options into a file called '`.indent.pro`'. `indent` will first look for '`.indent.pro`' in the current directory and use that if found. Otherwise, `indent` will search your home directory for '`.indent.pro`' and use that file if it is found. This behaviour is different from that of other versions of `indent`, which load both files if they both exist.

Command line switches are handled *after* processing '`.indent.pro`'. Options specified later override arguments specified earlier, with one exception: Explicitly specified options always override background options (see Section 1.3 [Common styles], page 4). You can prevent `indent` from reading an '`.indent.pro`' file by specifying the '`-npro`' option.

---

[1] '`+`' is being superseded by '`--`' to maintain consistency with the POSIX standard.

1.2 Backup Files

As of version 1.3, GNU `indent` makes GNU–style backup files, the same way GNU Emacs does. This means that either *simple* or *numbered* backup filenames may be made.

Simple backup file names are generated by appending a suffix to the original file name. The default for the this suffix is the one-character string '`~`' (tilde). Thus, the backup file for '`python.c`' would be '`python.c~`'.

Instead of the default, you may specify any string as a suffix by setting the environment variable `SIMPLE_BACKUP_SUFFIX` to your preferred suffix.

Numbered backup versions of a file '`momewraths`' look like '`momewraths.c.~23~`', where 23 is the version of this particular backup. When making a numbered backup of the file '`src/momewrath.c`', the backup file will be named '`src/momewrath.c.~`*V*`~`', where *V* is one greater than the highest version currently existing in the directory '`src`'.

The type of backup file made is controlled by the value of the environment variable `VERSION_CONTROL`. If it is the string '`simple`', then only simple backups will be made. If its value is the string '`numbered`', then numbered backups will be made. If its value is '`numbered-existing`', then numbered backups will be made if there *already exist* numbered backups for the file being indented; otherwise, a simple backup is made. If `VERSION_CONTROL` is not set, then `indent` assumes the behaviour of '`numbered-existing`'.

Other versions of `indent` use the suffix '`.BAK`' in naming backup files. This behaviour can be emulated by setting `SIMPLE_BACKUP_SUFFIX` to '`.BAK`'.

Note also that other versions of `indent` make backups in the current directory, rather than in the directory of the source file as GNU `indent` now does.

1.3 Common styles

There are several common styles of C code, including the GNU style, the Kernighan & Ritchie style, and the original Berkeley style. A style may be selected with a single *background* option, which specifies a set of values for all other options. However, explicitly specified options always override options implied by a background option.

As of version 1.2, the default style of GNU `indent` is the GNU style. Thus, it is no longer neccessary to specify the option '`-gnu`' to obtain this format, although doing so will not cause an error. Option settings which correspond to the GNU style are:

```
-nbad -bap -nbbb -nbc -bl -bli2 -c33 -cd33 -ncdb -nce
-cli0 -cp1 -di0 -nfc1 -nfca -i2 -ip5 -lp -pcs -psl
-nsc -nsob -nss -ts8
```

The GNU coding style is that preferred by the GNU project. It is the style that the GNU Emacs C mode encourages and which is used in the C portions of GNU Emacs. (People interested in writing programs for Project GNU should get a copy of *The GNU Coding Standards*, which also covers semantic and portability issues such as memory usage, the size of integers, etc.)

The Kernighan & Ritchie style is used throughout their well-known book *The C Programming Language*. It is enabled with the '`-kr`' option. The Kernighan & Ritchie style corresponds to the following set of options:

```
-nbad -bap -nbbb -nbc -br -c33 -cd33 -ncdb -ce -ci4
-cli0 -cp33 -d0 -di1 -nfc1 -nfca -i4 -ip0 -l75 -lp
-npcs -npsl -nsc -nsob -nss -ts8
```

Kernighan & Ritchie style does not put comments to the right of code in the same column at all times (nor does it use only one space to the right of the code), so for this style `indent` has arbitrarily chosen column 33.

The style of the original Berkeley `indent` may be obtained by specifying '`-orig`' (or by specifyfying '`--original`', using the long option name). This style is equivalent to the following settings:

```
-nbap -nbad -nbbb -bc -br -c33 -cd33 -cdb -ce -ci4
-cli0 -cp33 -d4 -di16 -fc1 -fca -i4 -ip4 -l75 -lp
-npcs -psl -sc -nsob -nss -ts8
```

1.4 Blank lines

Various programming styles use blank lines in different places. `indent` has a number of options to insert or delete blank lines in specific places.

The '`-bad`' option causes `indent` to force a blank line after every block of declarations. The '`-nbad`' option causes `indent` not to force such blank lines.

The '`-bap`' option forces a blank line after every procedure body. The '`-nbap`' option forces no such blank line.

The '`-bbb`' option forces a blank line before every block comment. A block comment is one which starts in column one when formatting of such comments is disabled, or one with `-` or `*` immediately following the `/*`. The '`-nbbb`' option does not force such blank lines.

The '`-sob`' option causes `indent` to swallow optional blank lines (that is, any optional blank lines present in the input will be removed from the output). If the '`-nsob`' is specified, any blank lines present in the input file will be copied to the output file.

For example, given the input

```
char *foo;
char *bar;
/* This separates blocks of declarations.  */
int baz;
```

`indent -bad` produces

```
char *foo;
char *bar;

/* This separates blocks of declarations.  */
int baz;
```

and `indent -nbad` produces

```
char *foo;
char *bar;
/* This separates blocks of declarations.  */
int baz;
```

The '`-bap`' option forces a blank line after every procedure body.

For example, given the input

```
int
foo ()
{
  puts("Hi");
}
/* The procedure bar is even less interesting.  */
char *
bar ()
{
  puts("Hello");
}
```

`indent -bap` produces

```
int
foo ()
{
  puts ("Hi");
}

/* The procedure bar is even less interesting.  */
char *
bar ()
{
  puts ("Hello");
}
```

and `indent -nbap` produces

```
int
foo ()
{
  puts ("Hi");
}
/* The procedure bar is even less interesting.  */
char *
bar ()
{
  puts ("Hello");
}
```

No blank line will be added after the procedure `foo`.

1.5 Comments

Comments are no longer formatted by default as of version 1.2. This can be enabled with the '`-fca`' option. Doing so will cause newlines in the comment text to be ignored and the line will be filled up to the length of a line (which can be modified with '`-l`'). When formatting is enabled, blank lines indicate paragraph breaks.

The '`-fc1`' option enables the formatting of comments which begin in the first column. The '`-nfc1`' option disables the formatting of first column comments. When comment formatting is disabled, overall comment indentation may still be adjusted.

The indentation of comments which do not appear to the right of code is set by the '`-d`' option, which specifies the number of spaces to the left of the surrounding code that the comment appears. For example, '`-d2`' places comments two spaces to the left of code; '`-d0`' lines up comments with the code. The '`-cdb`' option controls whether the '`/*`' and '`*/`' are placed on blank lines. With '`-cdb`', comments look like this:

```
/*
 * this is a comment
 */
```

With '`-ncdb`', comments look like this:

```
/* this is a comment */
```

'`-cdb`' only affects block comments, not comments to the right of code. The default is '`-ncdb`'.

Comments which appear on the same line as code are placed to the right. The column in which comments on code start is controlled by the '`-c`' option. The column in which comments to the right of declarations start is controlled by the '`-cd`' option. By default, they start in the same column as comments to the right of code, which is column 33. The column number for comments to the right of '`#else`' and '`#endif`' statements is controlled by the '`-cp`' option. If the code on a line extends past the comment column, the comment starts further to the right, and the right margin may be automatically extended in extreme cases.

If the '`-sc`' option is specified, '`*`' is placed at the left edge of all comments. For example:

```
/* This is a comment which extends from one line
 * onto the next line, thus causing us to consider
 * how it should continue.  */
```

instead of

```
/* This is a comment which extends from one line
   onto the next line, thus causing us to consider
   how it should continue.  */
```

1.6 Statements

The '`-br`' or '`-bl`' option specifies how to format braces.

The '`-br`' option formats braces like this:

```
if (x > 0) {
  x--;
}
```

The '`-bl`' option formats them like this:

```
if (x > 0)
  {
    x--;
  }
```

If you use the '`-bl`' option, you may also want to specify the '`-bli`' option. This option specifies the number of spaces by which braces are indented. '`-bli2`', the default, gives the result shown above. '`-bli0`' results in the following:

```
if (x > 0)
{
  x--;
}
```

If you are using the '-br' option, you probably want to also use the '-ce' option. This causes the else in an if-then-else construct to cuddle up to the immediately preceding '}'. For example, with '-br -ce' you get the following:

```
if (x > 0) {
  x--;
} else {
  fprintf (stderr, "...something wrong?\n");
}
```

With '-br -nce' that code would appear as

```
if (x > 0) {
  x--;
}
else {
  fprintf (stderr, "...something wrong?\n");
}
```

The '-cli' option specifies the number of spaces that case labels should be indented to the right of the containing 'switch' statement.

If a semicolon is on the same line as a for or while statement, the '-ss' option will cause a space to be placed before the semicolon. This emphasizes the semicolon, making it clear that the body of the for or while statement is an empty statement. -nss disables this feature.

The '-pcs' option causes a space to be placed between the name of the procedure being called and the '(' (for example, puts ("Hi");. The '-npcs' option would give puts("Hi");).

If the '-cs' option is specified, indent puts a space after a cast operator.

The '-bs' option ensures that there is a space between the keyword sizeof and its argument. In some versions, this is known as the 'Bill_Shannon' option.

1.7 Declarations

By default indent will line up identifiers, in the column specified by the '-di' option. For example, '-di16' makes things look like:

```
int              foo;
char            *bar;
```

Using a small value (such as one or two) for the '`-di`' option can be used to cause the indentifiers to be placed in the first available position, for example

```
int foo;
char *bar;
```

The value given to the '`-di`' option will still affect variables which are put on separate lines from their types, for example '`-di2`' will lead to

```
int
  foo;
```

If the '`-bc`' option is specified, a newline is forced after each comma in a declaration. For example,

```
int a,
  b,
  c;
```

With the '`-nbc`' option this would look like

```
int a, b, c;
```

The '`-psl`' option causes the type of a procedure being defined to be placed on the line before the name of the procedure. This style is required for the `etags` program to work correctly, as well as some of the `c-mode` functions of Emacs.

If you are not using the '`-di1`' option to place variables being declared immediately after their type, you need to use the '`-T`' option to tell `indent` the name of all the typenames in your program that are defined by `typedef`. '`-T`' can be specified more than once, and all names specified are used. For example, if your program contains

```
typedef unsigned long CODE_ADDR;
typedef enum {red, blue, green} COLOR;
```

you would use the options '-T CODE_ADDR -T COLOR'.

1.8 Indentation

One issue in the formatting of code is how far each line should be indented from the left margin. When the beginning of a statement such as if or for is encountered, the indentation level is increased by the value specified by the '-i' option. For example, use '-i8' to specify an eight character indentation for each level. When a statement is continued from a previous line, it is indented by a number of additional spaces specified by the '-ci' option. '-ci' defaults to 0. However, if the '-lp' option is specified, and a line has a left parenthesis which is not closed on that line, then continuation lines will be lined up to start at the character position just after the left parenthesis. This processing also applies to '[' and applies to '{' when it occurs in initialization lists. For example, a piece of continued code might look like this with '-nlp -ci3' in effect:

```
p1 = first_procedure (second_procedure (p2, p3),
   third_procedure (p4, p5));
```

With '-lp' in effect the code looks somewhat clearer:

```
p1 = first_procedure (second_procedure (p2, p3),
                      third_procedure (p4, p5));
```

indent assumes that tabs are placed at regular intervals of both input and output character streams. These intervals are by default 8 columns wide, but (as of version 1.2) may be changed by the '-ts' option. Tabs are treated as the equivalent number of spaces.

The indentation of type declarations in old-style function definitions is controlled by the '-ip' parameter. This is a numeric parameter specifying how many spaces to indent type declarations. For example, the default '-ip5' makes definitions look like this:

```
char *
create_world (x, y, scale)
     int x;
     int y;
     float scale;
{
  . . .
}
```

For compatibility with other versions of indent, the option '`-nip`' is provided, which is equivalent to '`-ip0`'.

1.9 Miscellaneous options

To find out what version of `indent` you have, use the command `indent -version`. This will report the version number of `indent`, without doing any of the normal processing.

The '`-v`' option can be used to turn on verbose mode. When in verbose mode, `indent` reports when it splits one line of input into two more more lines of output, and gives some size statistics at completion.

1.10 Copyright

The following copyright notice applies to the `indent` program. The copyright and copying permissions for this manual appear near the beginning of this document.

Appendix A Option Summary

Here is a list of all the options for `indent`, alphabetized by short option. It is followed by a cross key alphabetized by long option.

'`-bad`'
'`--blank-lines-after-declarations`'

Force blank lines after the declarations.
See Section 1.4 [Blank lines], page 5.

'`-bap`'
'`--blank-lines-after-procedures`'

Force blank lines after procedure bodies.
See Section 1.4 [Blank lines], page 5.

'`-bbb`'
'`--blank-lines-after-block-comments`'

Force blank lines after block comments.
See Section 1.4 [Blank lines], page 5.

'`-bc`'
'`--blank-lines-after-commas`'

Force newline after comma in declaration.
See Section 1.7 [Declarations], page 10.

'`-bl`'
'`--braces-after-if-line`'

Put braces on line after `if`, etc.
See Section 1.6 [Statements], page 8.

'`-bli`*n*'
'`--brace-indent`*n*'

Indent braces *n* spaces.
See Section 1.6 [Statements], page 8.

'`-br`'
'`--braces-on-if-line`'

Put braces on line with `if`, etc.
See Section 1.6 [Statements], page 8.

'`-c`*n*'
'`--comment-indentation`*n*'

Put comments to the right of code in column *n*.
See Section 1.5 [Comments], page 7.

‘-cd*n*’
‘--declaration-comment-column*n*’

Put comments to the right of the declarations in column *n*.
See Section 1.5 [Comments], page 7.

‘-cdb’
‘--comment-delimiters-on-blank-lines’

Put comment delimiters on blank lines.
See Section 1.5 [Comments], page 7.

‘-ce’
‘--cuddle-else’

Cuddle else and preceeding ‘}’.
See Section 1.5 [Comments], page 7.

‘-ci*n*’
‘--continuation-indentation*n*’

Continuation indent of *n* spaces.
See Section 1.6 [Statements], page 8.

‘-cli*n*’
‘--case-indentation*n*’

Case label indent of *n* spaces.
See Section 1.6 [Statements], page 8.

‘-cp*n*’
‘--else-endif-column*n*’

Put comments to the right of ‘#else’ and ‘#endif’ statements in column *n*.
See Section 1.5 [Comments], page 7.

‘-cs’
‘--space-after-cast’

Put a space after a cast operator.
See Section 1.6 [Statements], page 8.

‘-bs’
‘--blank-before-sizeof’

Put a space between sizeof and its argument.
See Section 1.6 [Statements], page 8

‘-d*n*’
‘--line-comments-indentation*n*’

Set indentation of comments not to the right of code to *n* spaces.
See Section 1.5 [Comments], page 7.

‘`-di`*n*’
‘`--declaration-indentation`*n*’

Put variables in column *n*.
See Section 1.7 [Declarations], page 10.

‘`-fc1`’
‘`--format-first-column-comments`’

Format comments in the first column.
See Section 1.5 [Comments], page 7.

‘`-fca`’
‘`--format-all-comments`’

Do not disable all formatting of comments.
See Section 1.5 [Comments], page 7

‘`-gnu`’
‘`--gnu-style`’

Use GNU coding style. This is the default.
See Section 1.3 [Common styles], page 4.

‘`-i`*n*’
‘`--indent-level`*n*’

Set indentation level to *n* spaces.
See Section 1.8 [Indentation], page 11.

‘`-ip`*n*’
‘`--parameter-indentation`*n*’

Indent parameter types in old-style function definitions by *n* spaces.
See Section 1.8 [Indentation], page 11.

‘`-kr`’
‘`--k-and-r-style`’

Use Kernighan & Ritchie coding style.
See Section 1.3 [Common styles], page 4.

‘`-l`*n*’
‘`--line-length`*n*’

Set maximum line length to *n*.
See Section 1.5 [Comments], page 7.

‘`-lp`’
‘`--continue-at-parentheses`’

Line up continued lines at parentheses.
See Section 1.8 [Indentation], page 11.

‘-nbad’
‘--no-blank-lines-after-declarations’

Do not force blank lines after declarations.
See Section 1.4 [Blank lines], page 5.

‘-nbap’
‘--no-blank-lines-after-procedures’

Do not force blank lines after procedure bodies.
See Section 1.4 [Blank lines], page 5.

‘-nbbb’
‘--no-blank-lines-after-block-comments’

Do not force blank-lines after block comments.
See Section 1.4 [Blank lines], page 5.

‘-nbc’
‘--no-blank-lines-after-commas’

Do not force newlines after commas in declarations.
See Section 1.7 [Declarations], page 10.

‘-ncdb’
‘--no-comment-delimiters-on-blank-lines’

Do not put comment delimiters on blank lines.
See Section 1.5 [Comments], page 7.

‘-nce’
‘--dont-cuddle-else’

Do not cuddle `}` and `else`.
See Section 1.6 [Statements], page 8.

‘-ncs’
‘--no-space-after-casts’

Do not put a space after cast operators.
See Section 1.6 [Statements], page 8.

‘-nfc1’
‘--dont-format-first-column-comments’

Do not format comments in the first column as normal.
See Section 1.5 [Comments], page 7.

‘-nfca’
‘--dont-format-comments’

Do not format any comments.
See Section 1.5 [Comments], page 7.

‘`-nip`’
‘`--no-parameter-indentation`’

Zero width indentation for parameters.
See Section 1.8 [Indentation], page 11

‘`-nlp`’
‘`--dont-line-up-parentheses`’

Do not line up parentheses.
See Section 1.6 [Statements], page 8.

‘`-npcs`’
‘`--no-space-after-function-call-names`’

Do not put space after the function in function calls.
See Section 1.6 [Statements], page 8.

‘`-npsl`’
‘`--dont-break-procedure-type`’

Put the type of a procedure on the same line as its name.
See Section 1.7 [Declarations], page 10.

‘`-nsc`’
‘`--dont-star-comments`’

Do not put the ‘`*`’ character at the left of comments.
See Section 1.5 [Comments], page 7.

‘`-nsob`’
‘`--leave-optional-blank-lines`’

Do not swallow optional blank lines.
See Section 1.4 [Blank lines], page 5.

‘`-nss`’

‘`--dont-space-special-semicolon`’

Do not force a space before the semicolon after certain statements. Disables ‘`-ss`’.
See Section 1.6 [Statements], page 8.

‘`-nv`’
‘`--no-verbosity`’

Disable verbose mode. See Section 1.9 [Miscellaneous options], page 12.

‘`-orig`’
‘`--original`’

Use the original Berkeley coding style.
See Section 1.3 [Common styles], page 4.

‘-npro’
‘--ignore-profile’

Do not read ‘.indent.pro’ files.
See Section 1.1 [Invoking indent], page 1.

‘-pcs’
‘--space-after-procedure-calls’

Insert a space between the name of the procedure being called and the ‘(’.
See Section 1.6 [Statements], page 8.

‘-psl’
‘--procnames-start-lines’

Put the type of a procedure on the line before its name.
See Section 1.7 [Declarations], page 10.

‘-sc’
‘--start-left-side-of-comments’

Put the ‘*’ character at the left of comments.
See Section 1.5 [Comments], page 7.

‘-sob’
‘--swallow-optional-blank-lines’

Swallow optional blank lines.
See Section 1.4 [Blank lines], page 5.

‘-ss’
‘--space-special-semicolon’

On one-line `for` and `while` statments, force a blank before the semicolon.
See Section 1.6 [Statements], page 8.

‘-st’
‘--standard-output’

Write to standard output.
See Section 1.1 [Invoking indent], page 1.

‘-T’ Tell `indent` the name of typenames.
See Section 1.7 [Declarations], page 10.

‘-ts*n*’
‘--tab-size*n*’

Set tab size to *n* spaces.
See Section 1.8 [Indentation], page 11.

‘-v’
‘--verbose’

Enable verbose mode.
See Section 1.9 [Miscellaneous options], page 12.

‘`-version`’

Output the version number of `indent`.
See Section 1.9 [Miscellaneous options], page 12.

Options’ Cross Key

Here is a list of options alphabetized by long option, to help you find the corresponding short option.

Index

.

B

C

E

G

I

K

L

O

S

T

U

Table of Contents