```
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.

[1] '`+`' is being superseded by '`--`' to maintain consistency with the POSIX standard.

## 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:

```
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

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

```
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:

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));
```

'`-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 9.

'`-cli`*n*'
'`--case-indentation`*n*'

Case label indent of *n* spaces.
See Section 1.6 [Statements], page 9.

'`-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 9.

'`-bs`'
'`--blank-before-sizeof`'

Put a space between `sizeof` and its argument.
See Section 1.6 [Statements], page 9

'`-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.

'-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 9.

'-ncs'
'--no-space-after-casts'

Do not put a space after cast operators.
See Section 1.6 [Statements], page 9.

'-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.

'-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 9.

'-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 9.

'-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 12.

'-v'
'--verbose'
: Enable verbose mode.
See Section 1.9 [Miscellaneous options], page 13.