Next: Comments, Up: Writing C [Contents][Index]

5.1 Formatting Your Source Code

Please keep the length of source lines to 79 characters or less, for maximum readability in the widest range of environments.

It is important to put the open-brace that starts the body of a C function in column one, so that they will start a defun. Several tools look for open-braces in column one to find the beginnings of C functions. These tools will not work on code not formatted that way.

Avoid putting open-brace, open-parenthesis or open-bracket in column one when they are inside a function, so that they won't start a defun. The open-brace that starts a `struct` body can go in column one if you find it useful to treat that definition as a defun.

It is also important for function definitions to start the name of the function in column one. This helps people to search for function definitions, and may also help certain tools recognize them. Thus, using Standard C syntax, the format is this:

```
static char *
concat (char *s1, char *s2)
{
  …
}
```

or, if you want to use traditional C syntax, format the definition like this:

```
static char *
concat (s1, s2)        /* Name starts in column one here */
     char *s1, *s2;
{                     /* Open brace in column one here */
  …
}
```

In Standard C, if the arguments don't fit nicely on one line, split it like this:

```
int
lots_of_args (int an_integer, long a_long, short a_short,
              double a_double, float a_float)
…
```

For `struct` and `enum` types, likewise put the braces in column one, unless the whole contents fits on one line:

```
struct foo
{
  int a, b;
}
```

or

```
struct foo { int a, b; }
```

The rest of this section gives our recommendations for other aspects of C formatting style, which is also the default style of the `indent` program in version 1.2 and newer. It corresponds to the options

```
-nbad -bap -nbc -bbo -bl -bli2 -bls -ncdb -nce -cp1 -cs -di2
-ndj -nfc1 -nfca -hnl -i2 -ip5 -lp -pcs -psl -nsc -nsob
```

We don't think of these recommendations as requirements, because it causes no problems for users if two different programs have different formatting styles.

But whatever style you use, please use it consistently, since a mixture of styles within one program tends to look ugly. If you are contributing changes to an existing program, please follow the style of that program.

For the body of the function, our recommended style looks like this:

```
if (x < foo (y, z))
  haha = bar[4] + 5;
else
  {
    while (z)
      {
        haha += foo (z, z);
        z--;
      }
    return ++x + bar ();
  }
```

We find it easier to read a program when it has spaces before the open-parentheses and after the commas. Especially after the commas.

When you split an expression into multiple lines, split it before an operator, not after one. Here is the right way:

```
if (foo_this_is_long && bar > win (x, y, z)
    && remaining_condition)
```

Try to avoid having two operators of different precedence at the same level of indentation. For example, don't write this:

```
mode = (inmode[j] == VOIDmode
        || GET_MODE_SIZE (outmode[j]) > GET_MODE_SIZE (inmode[j])
        ? outmode[j] : inmode[j]);
```

Instead, use extra parentheses so that the indentation shows the nesting:

```
mode = ((inmode[j] == VOIDmode
         || (GET_MODE_SIZE (outmode[j]) > GET_MODE_SIZE (inmode[j])))

        ? outmode[j] : inmode[j]);
```

Insert extra parentheses so that Emacs will indent the code properly. For example, the following indentation looks nice if you do it by hand,

```
v = rup->ru_utime.tv_sec*1000 + rup->ru_utime.tv_usec/1000
    + rup->ru_stime.tv_sec*1000 + rup->ru_stime.tv_usec/1000;
```

but Emacs would alter it. Adding a set of parentheses produces something that looks equally nice, and which Emacs will preserve:

```
v = (rup->ru_utime.tv_sec*1000 + rup->ru_utime.tv_usec/1000
     + rup->ru_stime.tv_sec*1000 + rup->ru_stime.tv_usec/1000);
```

Format do-while statements like this:

```
do
  {
    a = foo (a);
  }
while (a > 0);
```

Please use formfeed characters (control-L) to divide the program into pages at logical places (but not within a function). It does not matter just how long the pages are, since they do not have to fit on a printed page. The formfeeds should appear alone on lines by themselves.

Next: Comments, Up: Writing C [Contents][Index]