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

5.3 Clean Use of C Constructs

Please explicitly declare the types of all objects. For example, you should explicitly declare all arguments to functions, and you should declare functions to return `int` rather than omitting the `int`.

Some programmers like to use the GCC '`-Wall`' option, and change the code whenever it issues a warning. If you want to do this, then do. Other programmers prefer not to use '`-Wall`', because it gives warnings for valid and legitimate code which they do not want to change. If you want to do this, then do. The compiler should be your servant, not your master.

Don't make the program ugly just to placate static analysis tools such as `lint`, `clang`, and GCC with extra warnings options such as `-Wconversion` and `-Wundef`. These tools can help find bugs and unclear code, but they can also generate so many false alarms that it hurts readability to silence them with unnecessary casts, wrappers, and other complications. For example, please don't insert casts to `void` or calls to do-nothing functions merely to pacify a lint checker.

Declarations of external functions and functions to appear later in the source file should all go in one place near the beginning of the file (somewhere before the first function definition in the file), or else should go in a header file. Don't put `extern` declarations inside functions.

It used to be common practice to use the same local variables (with names like `tem`) over and over for different values within one function. Instead of doing this, it is better to declare a separate local variable for each distinct purpose, and give it a name which is meaningful. This not only makes programs easier to understand, it also facilitates optimization by good compilers. You can also move the declaration of each local variable into the smallest scope that includes all its uses. This makes the program even cleaner.

Don't use local variables or parameters that shadow global identifiers. GCC's '`-Wshadow`' option can detect this problem.

Don't declare multiple variables in one declaration that spans lines. Start a new declaration on each line, instead. For example, instead of this:

```
int    foo,
       bar;
```

write either this:

```
int foo, bar;
```

or this:

```
int foo;
int bar;
```

(If they are global variables, each should have a comment preceding it anyway.)

When you have an `if-else` statement nested in another `if` statement, always put braces around the `if-else`. Thus, never write like this:

```
if (foo)
  if (bar)
    win ();
  else
    lose ();
```

always like this:

```
if (foo)
  {
    if (bar)
      win ();
    else
      lose ();
  }
```

If you have an `if` statement nested inside of an `else` statement, either write `else if` on one line, like this,

```
if (foo)
  …
else if (bar)
  …
```

with its `then`-part indented like the preceding `then`-part, or write the nested `if` within braces like this:

```
if (foo)
  …
else
  {
    if (bar)
      …
  }
```

Don’t declare both a structure tag and variables or typedefs in the same declaration. Instead, declare the structure tag separately and then use it to declare the variables or typedefs.

Try to avoid assignments inside `if`-conditions (assignments inside `while`-conditions are ok). For example, don’t write this:

```
if ((foo = (char *) malloc (sizeof *foo)) == NULL)
  fatal ("virtual memory exhausted");
```

instead, write this:

```
foo = (char *) malloc (sizeof *foo);
if (foo == NULL)
  fatal ("virtual memory exhausted");
```

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