Good Style

This handout was written by Julie Zelenski.

Developing a good sense of style in a particular programming language takes practice and work. And, as with any complex activity, there is no one "right" way that can easily described, nor is there an easy-to-follow checklist of do's and don'ts. There's a lot of variation allowable in C coding style. If nothing else, try to be consistent. Here are a few style tips on the various C constructs based on my ideas of what makes good sense.

Break

Using `break` is ok in loops. Ideally, the loop should be structured to iterate in the most straightforward way. The `break` in the body can detect the exceptional case that comes up during the iteration.

```
for (i = 0; i < length; i++ ) {
   if (<found>) break;
   ...
}
```

or

```
while (current != NULL) {
   if (<found>) break;
   ...
}
```

While

`while (TRUE)` types loops are fine if they are really necessary. Often you need this for the loop-and-a-half-type situation where you need to first do some processing before you are able to test whether you need to exit the loop. If the first statement of a `while (TRUE)` loop is the test, then you should just put the test directly into the `while` statement. If the bounds of iteration are known, then a `for` loop is preferable.

Return

A `return` in a place other than the end of a function body is potentially vulgar. The early `return` can be used nicely if it detects and immediately exits on an exceptional case—for example, a recursive base case or an error case right at the beginning of a function. Sometimes `return` can be used like a `break` inside a loop when some condition becomes true. Be careful with `return` in the bodies of your functions— experience shows they are responsible for a disproportionate number of bugs. The programmer forgets about the early-`return` case and assumes the function runs all the way to its end.

++, --

Nice obvious uses of these are fine, but nesting it inside something complicated is just asking for trouble. Find a more useful outlet for your cleverness.

```
for (i = 0; i < length; i++) {   // ok
...

while (--length) {         // ok
...

while (*t++ = *s++)        // ick!
...
```

Switch

If you ever exploit the fall-through property of cases within a switch, your documentation should definitely say so. It's pretty unusual.

!=, ==

This is just a minor readability issue, but it's nice to put in what you are really testing for, rather than rely on the anything-non-zero-is-true property. Even though the code may compile to exactly the same thing, it reads a little nicer.

`if (*current)`	**more clearly is**	`if (*current != '\0')`
`if (current)`	**more clearly is**	`if (current != NULL)`
`if (!count)`	**more clearly is**	`if (count == 0)`

Boolean Values

Boolean expressions and variables seem to be prone to redundancy and awkwardness. Replace repetitive constructions with the more concise and direct alternatives. If you have a boolean value, don't use additional `!=` or `==` operators on it to test its value. Just use its value directly or inverted. Also, watch for if-else statements that assign a variable to `true` or `false`. The result from evaluating the test can go right into the variable. A few examples:

```
if (flag == TRUE)          more succinctly is      if (flag)

if (matches > 0)           more succinctly is      found = (matches > 0);
 found = TRUE;
else
 found = FALSE;

if (hadError == FALSE)     more succinctly is      return !hadError;
   return TRUE;
else
   return FALSE;
```

Constants

`#define`-d constants should be independent; that is, you should only need to change one `#define` to change something about a program. For example,

```
#define RectWidth       3
#define RectHeight      2                   /* WARNING: problem */
#define RectPerimeter   10
```

is not so hot, because if you wanted to change `RectWidth` or `RectHeight`, you would also have to remember to change `RectPerimeter`. A better way is:

```
#define RectWidth       3
#define RectHeight      2
#define RectPerimeter   (2*RectWidth + 2*RectHeight)
```

Finis

This, of course, isn't the final word on all issues of C style. If you've got a question or an issue you'd like input on, don't be afraid to ask!