Indent style

From Wikipedia, the free encyclopedia

In computer programming, an **indent style** is a convention governing the indentation of blocks of code to convey the program's structure. This article largely addresses the C programming language and its descendants, but can be (and frequently is) applied to most other programming languages (especially those in the curly bracket family). Indent style is just one aspect of programming style.

Indentation is not a requirement of most programming languages. Rather, programmers indent to better convey the structure of their program. In particular, indentation is used to show the relationship between control flow constructs such as conditions or loops and code contained within and outside them. However, some programming languages (such as Python and Occam) use the indentation to determine the structure instead of using braces or keywords.

The size of the indent is usually independent of the style. Many early programs used tab characters for indentation, for simplicity and to save on source file size. Unix editors generally view tabs as equivalent to eight characters, while Macintosh environments would set them to four, creating confusion when code was transferred back and forth. Modern programming editors are now often able to set arbitrary indentation sizes, and will insert the appropriate combination of spaces and tabs. They can also help with the cross-platform tab confusion by being configured to insert only spaces.

There are a number of computer programs that automatically correct indent styles as well as the length of tabs. A famous one among them is `indent`, a program included with many Unix-like operating systems. These programs work best for those who use an indent style close to that considered "proper" by their programmers; those who use other styles will more likely become frustrated.

Contents

K&R style

The K&R style, so-called because it was used in Kernighan and Ritchie's book *The C Programming Language*, is commonly used in C. It is less common for Objective C, C++, C#, and others. It keeps the first opening brace on the same line as control statement, indents the statements within the braces, and puts the closing brace on the same indentation level as the control statement (on a line of its own).

```
while (x == y) {
    something();
    somethingelse();
}
finalthing();
```

Advocates of this style sometimes refer to it as "The One True Brace Style" or "The One Original True Brace Style" (abbreviated as 1TBS or TOOTBS) because of the precedent set by C (although advocates of other styles have been known to use similarly strong language). The source code of the UNIX kernel and Linux kernel is written in this style.

Advantages of this style are that the beginning brace does not require an extra line by itself; and the ending brace lines up with the statement it conceptually belongs to. Disadvantages of this style are that the beginning brace can be difficult to locate when scanning lines of programming code (the eye must repeatedly jump from the beginning of the line to the end and back again) and the ending brace takes up an entire line by itself. The latter can be partially resolved in if/else blocks and do/while blocks:

```
if (x < 0) {
    printf("Negative");
    negative(x);
} else {
    printf("Positive");
    positive(x);
}
```

The motive of this style is apparently to conserve screen real estate (and keep more code visible at once) by avoiding the need to dedicate an entire line to a single character (at the time this style was originated, a typical terminal had only 25 lines visible). This is most visibly demonstrated in the preceding example, as a chain of if...else if...[etc.]...else statements will display more code in a given number of lines than a style that places braces on new lines.

While Java is often written in BSD/Allman or other styles, a significant body of Java code uses the K&R style, largely because Sun's original style guides (see [here (http://developers.sun.com/prodtech/cc/products/archive/whitepapers/java-style.pdf)] and [here (http://java.sun.com/docs/codeconv/CodeConventions.pdf)]) used K&R, and as a result most of the standard source code for the Java API is written in K&R.

BSD/Allman style

The BSD/Allman style is common. It puts the brace associated with a control statement on the next line, indented to the same level as the control statement. Statements within the braces are indented to the next level.

```
while(x == y)
{
    something();
    somethingelse();
}
finalthing();
```

This style is similar to the standard indentation used by the Pascal programming language and Transact-SQL, where the braces are equivalent to the "begin" and "end" keywords.

Advantages of this style are that the indented code is clearly set apart from the containing statement by lines that are almost completely whitespace, improving readability; the ending brace lines up in the same column as the beginning brace; and the braces are consistently at the beginning of their lines. A disadvantage of this style is that each of the enclosing braces occupies an entire line by itself without adding any actual code. This once was an important consideration when programs were usually edited on terminals that displayed only 24 lines, but is less significant with larger resolutions.

The motive of this style is apparently to promote code readability through visually separating blocks from their control statements, deeming screen real estate a secondary concern.

This style is enforced by default in Microsoft's Visual Studio 2005.

Whitesmiths style

The Whitesmiths style is relatively uncommon compared to the prior two. It puts the brace associated with a control statement on the next line, indented. Statements within the braces are indented to the same level as the braces.

```
while (x == y)
    {
    something();
    somethingelse();
    }
finalthing();
```

The advantages of this style are similar to those of the BSD style in that blocks are clearly set apart from control statements. Another advantage is that the alignment of the braces with the block emphasizes the fact that the entire block is conceptually (as well as programmatically) a single compound statement. Furthermore, indenting the braces emphasizes the fact that they are not syntactically part of the 'while' statement, but rather the delineations of the subordinate compound statement.

A disadvantage of this style could be that the braces do not stand out as well. However, this is largely a matter of opinion, because the braces occupy an entire line to themselves, even if they are indented to the same level as the block.

GNU style

Like the BSD and Whitesmiths styles, GNU style puts braces on a line by themselves. The braces are indented by 2 spaces, and the contained code is indented by a further 2 spaces. Popularised by Richard Stallman, the layout may be influenced by his background of writing Lisp code. Although not directly related to indentation, GNU coding style also includes a space before the bracketed list of arguments to a function.

```
while (x == y)
  {
    something ();
    somethingelse ();
  }
finalthing ();
```

This style combines the advantages of BSD/Allman and Whitesmiths, thereby removing the possible Whitesmiths disadvantage of braces not standing out from the block. A disadvantage of this style is that more typing is required, since both the braces and the block must be indented.

The GNU Emacs text editor and the GNU systems' indent command indent code according to this style by default. It is mandated by nearly all maintainers of GNU project software, but is rarely used outside of the GNU community.

Those who do not use GNU Emacs, or similarly extensible/customisable editors, may find that the automatic indenting settings of their editor are unhelpful for this style which uses two levels of indentation.

Link: GNU Formatting (http://www.gnu.org/prep/standards/html_node/Formatting.html)

Pico style

The style used most commonly in the Pico programming language by its designers is different from the aforementioned styles. The lack of return statements and the fact that semicolons are used in Pico as statement separators, instead of

terminators, leads to the following syntax:

```
stuff(n):
{ x: 3 * n;
  y: doStuff(x);
  y + x }
```

The advantages and disadvantages are similar to those of saving screen real estate with K&R style. One additional advantage is that the beginning and closing braces are consistent in application (both share space with a line of code), as opposed to K&R style where one brace shares space with a line of code and one brace has a line to itself.

Banner style

The banner style makes visual scanning easier for some, since the "headers" of any block are the only thing exdented at that level (the theory being that the closing control of the previous block interferes with the header of the next block in the K&R and BSD/Allman styles). In this style, the closing control is indented as the last item in the list (and thus appropriately loses salience).

```
function1 () {
  dostuff
  do more stuff
  }

function2 () {
  etc
  }
```

or, in a markup language...

```
<table>
  <tr>
    <td> lots of stuff...
      more stuff
      </td>
    <td> alternate for short lines </td>
    <td> etc. </td>
    </tr>
  </table>

<table>
  <tr ... etc>
  </table>
```

Other considerations

Losing track of braces

In certain situations, there is a risk of losing track of which braces belong to which control statements. This is often seen in large sections of code containing many compound statements nested to many levels of indentation - by the time the programmer scrolls to the bottom of a huge set of nested statements, he may have lost track of which control statements go where. However this is also an inherent risk of certain indentation styles such as K&R, where the beginning brace can be much harder to find than the ending brace.

There are two ways to avoid losing track of control statements such as for. One way is to use a large indent, such as an 8-unit wide hard tab, along with breaking up large functions into smaller and more readable functions. Linux is done this way. Another way is to use inline comments added after the closing brace:

```
for ( int i = 0 ; i < total ; i++ ) {
    foo(bar);
} //for
```

```
if (x < 0) {
    bar(foo);
} //if
```

See also

- Indentation
- Programming style
- Kernel Normal Form

References

- Jargon File article on indent style (http://www.catb.org/~esr/jargon/html/I/indent-style.html)

Retrieved from "http://en.wikipedia.org/wiki/Indent_style"

Categories: Curly bracket programming languages | Text editor features

---