Tab-width Independent C-mode Indentation for Emacs

I think that the cc-mode indentation engine for Emacs is the best around; it is extremely customizable, and has features found in no other. But, to me, there is one problem with it : the way it distributes tabs and spaces when it indents.

I have been able, through deliberation and trial and error, to find a way to indent C and C++ code in such a way that the overall layout of the code is not affected by the tab size. I used to do this manually in vi, but read about CC-Mode in emacs, including its vaunted flexibility. I immediately thought that there would be a way to configure CC-Mode to indent the way I wanted it to. But, alas, CC-Mode uses emacs indent-to to indent the code, based only upon a character count of how much indentation is needed, and this either

- fills it with tabs, according to the current tab-width, until it cannot insert any more and fills the remainder with spaces, or
- it just fills it with spaces.

depending on whether indent-tabs-mode is nil. The former will misalign code if tab-width is changed, and the latter doesn't allow one to change the indentation size.

In the ideal, one should be able to change the amount of (visual) space by which the code in a block, say, is indented, and one should be able to *align* code that should begin at the same column in a way that is not affected by tab-width. Changing the indentation (visual) space without misaligning code can be done without modifying the file if the tabs and spaces are inserted meticulously.

Unfortunately, CC-Mode cannot be configured to do this automaticly.
I seek to change that. Here are some examples of what I mean. Given :

```
int
func()
{
some_function_with_long_name( arg1,
arg2,
arg3 );
}
```

cc-mode indents this to

```
int
func()
{
----some_function_with_long_name( arg1,
--------------------------------  arg2,
--------------------------------  arg3 );
}
```

with the ---- representing tabs and __ the spaces. It does it this way because emacs' indent-to does. If I change the tab-width to 8, I get

```
int
func
{
--------some_function_with_long_name( arg1,
                                       arg2,
                                       arg3
);
}
```

and the relative alignment is off. It should indent this as

```
int
func
{
----some_function_with_long_name( arg1,
----                              arg2,
----                              arg3 );
}
```

with (= tab-width 4) or

```
int
func
{
--------some_function_with_long_name( arg1,
--------                               arg2,
--------                               arg3 );
}
```

with (= tab-width 8) so that the relative alignment is not affected by the tab stops.
You could, e.g., for GNU style, indent like this with (= tab-width 4) :

```
int
func
{
----if(pred1)
----  {
------    action();
----  }
}
```

Or you can do GNU style indented like this :

```
int
func
{
--if(pred)
----{
------action();
```

```
----}
}
```

if you want to set your tab-width to 2 and use the gnu style that way. Both are preferable to

```
int
func
{
----if(pred)
------{
--------action();
------}
}
```

which would misalign the code if the tab-width changed.
I think of this as follows. When laying out code, there are two different things to be done.

- Indent code. Code is formatted with extra space before it to show that it is a sub-part of something else, e.g. a block, function body, etc.
- Align code. Code is put on the same column to show a relationship, e.g. args in the same function call, equivalent elements of an arithmetic or boolean expression, cases in the ?: operator, etc.

The alignment should be done, character for character, with spaces, so that it does not depend on the tab-width. The indentation should be done with indentation characters, i.e. tabs, so it can be easily globally modified to the preference of the reader without misaligning the aligned code.

The way I have sought to do this is to not use indent-to, but use a new elisp function I created, c-indent-to. This function accepts numbers, in which case it just calls indent-to. It also accepts, however, a list, whose elements prescribe actions for indenting. These can be

- A number. This calls indent-to, which indents in the normal way from the current column.
- A cons cell of the form (tab) or (tab . n), where n is a number. This inserts one or n tabs, resp.
- A cons cell of the form (space) or (space . n), where n is a number. This inserts one or n spaces, resp.
- A cons cell of the form (flush), which deletes all indentation inserted thus far

I have tried to be careful to not break any existing cc-mode configurations, so my changes can be incorporated into cc-mode without bothering those who couldn't care less about changing tab-widths.
Thus (c-indent-to '((flush) (tab . 2) (space . 2) (tab . 2))) indents

```
    action();
```

to

```
--------  --------action();
```

Also, c-collect-indentations has been modified to look around the indentation point and find out where spaces and tabs should be inserted and return the list for passing to c-indent-to, which will indent and align the code properly.

You can download what I've done so far : cc-mode-twi.tgz. This includes a file called cc-mode-dev.el which sets c-offsets-alist to *my* model setup.

It works for me and my setup, but there are some things that don't yet work, like using ++ or -- as the indentation in c-offsets-alist. It is unclear how -- should be handled. I've made some modifications to some functions in cc-indent.el to indent the way I like. Some of these really show off the appropriate using of tabs and spaces for indenting.

But: There is one problem. Sometimes an indentation character (tab) will begin in a column that is not a multiple of tab-width. This causes the indentation character to be displayed as less than tab-width characters.

E.g. Suppose I want to set a variable to the result of a function call, and want to start the argument list of the function on a separate line, and add a separate line for each arguement. Suppose, too, that one of the arguments is a function call whose argument list I want to treat in a similar way. I want to start the argument list indented, with beginning of the indentation aligned with the beginning of the function name. I set cc-mode up to do that. Emacs displays the following :

```
12345678var = function_call
12345678        12(arg1,
12345678        12 arg2_funcall
12345678        12 1234567(arg1,
12345678        12 1234567 arg2),
12345678        12 arg3 );
```

with (= tab-width 8) and

```
1234var = function_call
1234        12(arg1,
1234        12 arg2_funcall
1234        12 123(arg1,
1234        12 123 arg2),
1234        12 arg3 );
```

with (= tab-width 4). The tabs aligned with the f in function_call are only displayed as 2 characters wide, and the tabs aligned with the a in arg2_funcall are displayed as 7 and 3 characters wide, resp.
We can fix this. To make (X)Emacs display the tabs as tab-width characters wide, like this :

```
12345678var = function_call
12345678        12345678(arg1,
12345678        12345678 arg2_funcall
12345678        12345678 12345678(arg1,
12345678        12345678 12345678 arg2),
12345678        12345678 arg3 );
```

and thus make the indentation consistent, use the following function :

- for **XEmacs** :

```
(defun display-tabs-everywhere-as-tab-width (&optional win)
  "Makes all tabs display as tab-width characters.
This means that all tabs  will be displayed tab-width characters wide,
regardless of the column in which they begin; e.g. if (eq tab-width 8),
a tab starting at column 3 will be 8 instead of 5 characters wide."
  (interactive)
  (let* ((win (or win (selected-window)))
         (disptab (make-display-table))
         (tabvec (make-vector tab-width ?\ )))
    (aset disptab ?\t tabvec)
    (set-specifier current-display-table disptab)))
```

- for **GNU Emacs** :

```
(defun display-tabs-everywhere-as-tab-width (&optional win)
  "Makes all tabs display as tab-width characters.
This means that all tabs  will be displayed tab-width characters wide,
regardless of the column in which they begin; e.g. if (eq tab-width 8),
a tab starting at column 3 will be 8 instead of 5 characters wide."
  (interactive)
  (let* ((win (or win (selected-window)))
         (disptab (or (window-display-table win)
                      (make-display-table)))
         (tabvec (make-vector tab-width ?\ )))
    (aset disptab ?\t tabvec)
    (set-window-display-table win disptab)))
```

The following will undo this effect :

- for **Xemacs**

```
(defun display-tabs-normally (&optional win)
  (interactive)
  (let* ((win (or win (selected-window)))
  (disptab (window-display-table win)))
    (and disptab (aset disptab ?\t nil))))
```

- for **GNU Emacs** :

```
(defun display-tabs-normally (&optional win)
  (interactive)
  (let* ((win (or win (selected-window)))
         (disptab (window-display-table win)))
    (and disptab (aset disptab ?\t nil))))
```

Note: Unfortunately, in the version of XEmacs that I use, this evokes a display bug. When the cursor is on a tab, the cursor is tab-width characters wide (not necessarily a bug), and when the tab character has moved, sometimes tabs are still displayed with the cursor color. But, then, *it does* say that the display table code is immature.

Mail me with thoughts!

Note: This has recently changed. If you have sent me mail and not received a response, try sending it to this new address. Sorry about any frustration this might have caused.


W3C
HTML
4.0