NAME

pretty - adjust C code to coding standard

SYNOPSIS

pretty [-a|b|c[n:n:n ...]] [-rin[+]] [file ...]

DESCRIPTION

Pretty takes C code, either from the standard input or from the listed files and reformats them to the coding standard specified by the **-a**, **-b**, or **-c** switch. If no coding standard is specified, **-a4** is assumed.

-r Output is normally directed to the standard output. The **-r** switch directs pretty to place the output temporarily in tmppretty????? and at the end of reformatting, rename original to old.original and tmppretty????? to original, thus preserving a copy of the original code.

-i Normally pretty makes an attempt to add four extra spaces to any lines that it thinks are continuations of previous lines. If a previous line did not end in ; : { or } and the new line does start with a {, then pretty thinks it is a continuation. The **-i** switch tells pretty not to worry about continuation lines and no extra spaces are added to such lines.

-n[+]

Pretty can handle the C formatting macros (Programmers Note #113) if the **-n** switch is specified. This switch tells pretty that comments can start with _C*, _B*, or _F* as well as the usual /* sequence. It also tells pretty that braces might be found as _{ and _} and that the occurance of __ should be treated as a complete statement. The "+" additionally specifies that all indents must be multiples of 8 and that when deindenting switch statement labels, they should be deindented by 8 instead of the normal 2. This is necessary for anyone using the _{ and _} in place of normal braces and the __ to specify that nroff deindent the switch statment labels. This is because the nroff formatting macros do not delete leading spaces in lines, though they do delete leading tabs. Without the "+" option, the final results after nroff'ing would not be what was desired. It is suggested that people use the commenting aids of the nroff macros, but allow pretty to handle the braces and indenting. If this is done, only the **n** option without the "+" is required. It should be noted that the "+" on the **n** switch overides any tab setting given with the **a** , **b** , or **c** switches.

The coding standards are:

```
-a   xxxxxxxxxxxxxxxxxxxxxxx
     {
         xxxxxxxxxxxxxxxxxx
         xxxxxxxxxxxxxxxxxx
     }

-b   xxxxxxxxxxxxxxxxxxxxxxx{
         xxxxxxxxxxxxxxxxxx
         xxxxxxxxxxxxxxxxxx
     }

-c   xxxxxxxxxxxxxxxxxxxxxxx
       {
         xxxxxxxxxxxxxxxxxx
         xxxxxxxxxxxxxxxxxx
       }
```

A coding standard switch may be followed by an optional **n:n:....**, where each **n** is the size of the next indent in the series. The default has all indents set to 4 for coding standards **a** and **b** and 2 for coding standard **c**. This means that each new section of code will begin four spaces further to the right than the previous section. In the following example

pretty -c2:6:2 file

produces code of type **c** with the first indent at 2, the second at 2+6, the third at 2+6+2, and all later indents at 2 further in from the previous indent. Sample code would appear as

```
subroutine()
  {
        xxxxxxxxxxxxxxx
          {
            xxxxxxxxxxxxxx
            xxxxxxxxxxxxxx
          }
        xxxxxxxxxxxxxxxxx
        xxxxxxxxxxxxxxxxx
  }
```

Pretty makes certain assumptions that the user should be aware of. Lines beginning with # and comments beginning in the leftmost column are not adjusted. All other comments are indented normally. If the user translates a file of C code from types **-a** or **-c** to **-b** it is possible to move a { from inside an #ifdef , #if or #else to the first statement preceding that conditional. Code must not appear in this form or the following can happen:

-a code

```
        if (a > b)
#ifdef TYPE
        {
          xxxxxxxxxxx ;
          xxxxxxxxxxx ;
        }
#else
            xxxxxxxxxxx ;
#endif
```

-->

-b code

```
        if (a > b){
#ifdef TYPE
            xxxxxxxxxxx ;
            xxxxxxxxxxx ;
        }
#else
            xxxxxxxxxxx ;
#endif
```

As long as the code inside condtionals is syntactically complete, this problem will not arise.

If code is translated to first one coding standard and then back, there is the possibility of minor differences between the original and final output, caused by movement of blank lines. No functional changes in code appearance will take place.

Labels are checked for as the first printing item on lines. If they are encountered, the label is printed left justified on a line by itself to make it stand out. The keywords case and default of the switch statement are printed with the indent at that point reduced by 2 spaces if possible. Lines that do not appear to be complete, that is not ending in a ; : { or } character, cause the next line of code to be indented an extra 4 spaces as long as the new line doesn't begin with a { character. This feature is turned off by the **-i** switch.

FILES

tmppretty?????