NAME

ppc − pretty printer for programs written in C.

SYNOPSIS

ppc [**−style** *nnn*] [**−v**] [**−nv**] [**−of** *output_file*] [**−nof**] [**−T** *name*] [**−d** *path*] [**−kwlc**] [**−kwuc**] [**−kwci**] [**−kwdk**] [**−nkwdk**] [**−ll** *nnn*] [**−mm** *nnn*] [**−sm**] [**−nsm**] [**−tabs**] [**−notabs**] [**−every** *nnn*] . . . [*input_files*]

DESCRIPTION

ppc checks the syntax of and pretty prints C programs found in *input_files*.

If no *input_files* argument is present, the standard input is read and the pretty printed program is written on the standard output. Otherwise, each *input_file* is analysed and, under default options, the corresponding pretty printed program is written on the same file if no error occurs.

OPTIONS

Options may appear in any order as long as they appear **before** the *input_files*. Only the **last** occurrence of a given option is taken into account. Default options are such that the command

ppc

is equivalent to

ppc −style 1 −v −kwlc −nkwdk −ll 122 −mm 60 −sm −tabs −every 8 −d ${HOME} −d .

−style *nnn*

Use output style *nnn*. Styles are not easy to describe in a few words, so the user is urged to try them all. Style 1 is at the moment the author's preferred one ; style 2 is intended for the nostalgics of usual Pascal styles (matching **{** and **}** are lined up, each on its own line) ; fervent readers of PCC source code may want to use style 3. (Default : *nnn* = 1)

−v, **−verbose**

Animate the user's screen by displaying cryptic information. (Default)

−nv, **−noverbose**

Execute silently.

−of *output_file*, **−output_file** *output_file*

Redirect program output. The default behaviour is to overwrite each *input_file*, or, if none is given, to write on the standard output. Note that this option should usually be regarded as a debugging option, as only the last pretty printed program will be found in *output_file*.

−nof, **−nooutput_file**

Redirect program output to the standard output. This is the default option when input is done from the standard input.

−T *name*, **−TYPE** *name*

Add *name* to the list of type key-words. You need to specify all the names that appear in your program as type names — whether they be defined by **typedef**s or by **#define** macros — ; if you do not, an error message will be output for each occurrence of such a type name. As some have already put it, ‘‘This sounds like a painful thing to have to do, but it´s really a symptom of a problem in C : **typedef** causes a syntactic change in the language . . .’’ ; furthermore *ppc* is neither the C preprocessor nor a C compiler, so it cannot do replacement on your text as they do.

−d *path*, **−directory** *path*

Immediately analyse the options file *path*/**.ppcrc**, if that file exists (the directory itself must exist).

−kwlc, **−key_words_in_lower_case**

Pretty print C key-words in lower case letters ; excludes -kwuc and -kwci. (Default)

−kwuc, **−key_words_in_upper_case**

Pretty print C key-words in upper case letters ; excludes -kwlc and -kwci. This option should not be used if the resulting program is intended to be compiled, as C key-words must be lower case to be recognised.

−kwci, **−key_words_capitalised_initial**
Pretty print C key-words with each leading letter capitalised and the others written in lower case ; excludes -kwuc and -kwlc. This option should not be used if the resulting program is intended to be compiled, as C key-words must be lower case to be recognised.

−kwdk, **−key_words_in_dark**
Attempt to make C key-words look darker than rest of text when displayed on a printer, by overstriking each letter twice (using backspaces). This option should not be used if the resulting program is intended to be compiled, as C key-words will no longer be recognised by any C compiler.

−nkwdk, **−nokey_words_in_dark**
Do not artificially embolden C key-words. (Default)

−ll *nnn*, **−line_length** *nnn*
Define the maximum length of an output line to be *nnn*. (Default : *nnn* = 122)

−mm *nnn*, **−max_margin** *nnn*
Specify that no output line should begin after column *nnn*. That value must be less than the line length. (Default : *nnn* = 60)

−sm, **−shift_margin**
Specify that when an output line should begin after the column defined by the **−mm** option, it is to be indented to the left, in order to preserve the structure of the program (more on this later). (Default)

−nsm, **−noshift_margin**
Specify that when an output line should begin after the column defined by the **−mm** option, it is to be output beginning in that column.

−tabs Replace sequences of spaces by tabulation characters wherever possible outside strings and comments. (Default)

−notabs
Do not produce tabulation characters.

−every *nnn*
Define the number of columns between two tabulation positions to be *nnn*. (Default : *nnn* = 8)

FURTHER DESCRIPTION

You may set up your own default options by creating a file named **.ppcrc** in your home directory, and including whatever options you like. You may also add further default options related to the C programs residing in the current directory by creating a **.ppcrc** file there also. Both your home and the current directory are always searched, in that order, for options files. This search occurs before any command line option analysis, and thus cannot be skipped. You may also specify other directories to be searched by using the **−directory** option, in options files as well as on the command line. The options must be separated by white space or comments ; a comment begins with a sharp character (‘#’), either at the beginning of a line or preceded by a tabulation character, and ends with the line.

Preprocessor lines

Any line that begins with a sharp character is left alone, as it is considered to be a comment. No attempt is made to understand conditional compilation nor to cope with syntactic variations introduced by the use of macros. For example, the text ``**forever {call proc}**´´ will give rise to a (im)pertinent error message, regardless of what this would be expanded into . . . (``**forever**´´ might expand to ``**for (;;)**´´, ``**call**´´ to nothing, and ``**proc**´´ to ``**func ();**´´, for example)

Line splitting

As all indenters, formatters, pretty-printers and other beautifiers, *ppc* has to meet the problem of physically bounded lines. The currently implemented way of resolving that stupid limitation (see **sxppp**(3)) is to put on a fresh line, indented by five spaces, the first lexical unit which cannot be output on the current line. This is not deemed to be satisfactory, and a new strategy is under study... Please come back later if you do not like the way your program has been laid out!

Indentation restriction

In order to minimise the effect of line splitting, *ppc* arbitrarily refuses to indent lines past a given column (defined by the **–max_margin** option). Its behaviour on a line which should begin far to the right of the page is governed by the **–shift_margin** option : when this option is set, such a line is shifted to the left, and the program text keeps its structure ; when the **–noshift_margin** option is in effect, the line is simply output beginning at the *max_margin* column. As a picture is always better than a long digression, here comes an example... Suppose you have the following program fragment deeply nested in your text :

```
if (conditional) {
    initialisations ();

    if (other_conditional) {
        do_work (some, arguments);
    }
    else
        do {
            if (nested_conditional) {
                do_complex_work (other, parameters);
            }
        } while (!finished);
}
```

Not shifting the margin might produce something like this :

```
            if (conditional) {
                initialisations ();

                if (other_conditional) {
                    do_work (some, arguments);
                }
                else
                    do {
                    if (nested_conditional) {
                    do_complex_work (other,
                          parameters);
                    }
                    } while (!finished);
            }
```

Shifting the margin would produce this kind of result :

```
            if (conditional) {
                initialisations ();

                if (other_conditional) {
                    do_work (some, arguments);
                }
                else
                    do {
if (nested_conditional) {
    do_complex_work (other, parameters);
}
                    } while (!finished);
            }
```

Choosing what to do here is mainly a matter of taste. Note that if you give to **–max_margin** the same value as the one you give to **–line_length**, only the process of line splitting as described earlier will be in effect.

FILES

~/.ppcrc	personal options file
./.ppcrc	options file
/dev/tty	for *verbose* output
/tmp/ppc*	temporary output file
/tmp/sx*	scratch files

SEE ALSO

The *SYNTAX Reference Manual* and the INRIA Report #455 entitled *Paradis, un Système de Paragraphage Dirigé par la Syntaxe*.

DIAGNOSTICS

When the input program is (syntactically) incorrect, error messages are issued and, unless the **–nof** option is in effect, the result is left in the temporary output file.

Exit status is 0 if everything was all right, 1 if only warnings were issued, 2 if error messages were issued, 3 for command line syntax errors.

BUGS

ppc is implemented using sxppp(3), and so benefits of all its bugs.

The meaning of a **–directory** option is to immediately replace this option by the contents of the specified options file, if it exists ; thus, recursion between options files leads to a loop, which is exited (gracefully) only when no more file can be opened by *ppc*.

Exotic options about key words case and emboldening should be replaced by a **–troff** option.