## **Programming Language Support — C**

Sopic Index	Programming Language Support — C			
<u>Description</u>	<u>Keystroke</u>	Function	<u>Note</u>	
Editing C Files  O Help & Customization  cc-mode learn/mod  O cc-mode set/help  O C keyword help  eldoc-mode  electric keys  electric-pair mode  linux-kernel style  insert new line(s)  Comments  delete whitespace  indentation  o indent rigidly  unindent  O c skeletons  insert ()  mark function  show function name  search support  highlighting blocks  navigate in C code  by xref  by call graph  by statement  O by block  O by preprocessor  Expand pre-processor  Insert/align eol \  Hide preprocessor  Preview UML	Emacs supports C natively via  PEL activates support for  Supported file extensions:  If pel-use-bison-mode  When pel-use-speedbar is  Important aspects of C sou  PEL customization for C:  Emacs customization gro  pel-c-indent-width: Ic  pel-c-tab-width: The  This concept differs  will indent to a colur  For most uses it  the indentation leve  pel-c-use-tabs: Whet  C code style sub-grou  pel-c-fill-column:  When pel-c-fill-column:  When pel-c-fill-column:  When pel-c-fill-column:  The values for those user  buffer and view their curr  None of the commands by	the built-in c-mode which extra C when one or both of the pel-code files: .c, .lex, .y, s is set, the .y and .yacc files set all these extensions are resurce code syntax controlled by Simplifies configuration for equip: pel-pkg-for-c (access with lentifies the number of columns width of a tab used for c-mode from indentation: you can have from indentation: you can have from indentation: you can have from that is a multiple of 3. PEL is best to set both values to the column where line-wrapping or obtain user option is nil, c-mod column user option is nil, c-mod is some Emacs Lisp code. Defare used for controlling C code up: pel-pkg-for-cc. Applies to Whether automatic newline mootion variables can also be sent settings using the comman elow change PEL default; they	ends the CC Mode that support the curly-bracket programming languages like C.  use-c or pel-use-bison user-option variables is set. Type C-h o pel-use-c RET to set it.  yacc, header files: h, i Add more in pel-auto-mode-alist user option.  les are associated with bison-mode, the l and lex with flex-mode and lison with jison-mode ecognized by speedbar, otherwise only the main ones are recognized.  the CC Mode are customizable with PEL user option variables.  diting C source code.  h <f12> <f2>):  sue used for indentation. Defaults to 3.  e an indentation of 3 and tab width of 8: M-i will move point to columns that are multiple of 8 <tab> stores this value inside the tab-width variable for c-mode buffers.  we width of your needed indentation level. This way you can use commands that use either to control attation or not: t: tabs are used, nil: only spaces are used. Default: nil.  ccurs: maximum line length (defaults to 80). You can change the value or set it nil.  be buffers use the Emacs fill-column value like other major modes.  le supported by the electric keys. You can select one of the values supported by Emacs or define</tab></f2></f12>	
o C code search/fix Programming help Info on C  See: Seeedbar Tree-Sitter (@GitHub)	Tree-Sitter Support  when the pel-use-tree-sitter user-option is set to t, PEL provides tree-sitter support via:  Tree-sitter and tree-sitter-langs external packages and Emacs built-in support.  Emacs with dynamic module loading, and built with tree-sitter support. tree-sitter library must be installed separately.  See: How to Get Started with Tree-Sitter  Emacs must find the tree-sitter language dynamic library files that have a name similar to 'libtree-sitter-python.so' (for Linux) or .dylib (for macOS).  Identify the relevant directory in the pel-treesit-load-path. See the docstring of that user-option for further instructions.  As of Emacs 30 tree-sitter c-ts-mode for C does not provide the flexibility of the c-mode to control the C source code style.  It's recommended to use c-mode to edit, navigate and manipulate C source code. At least until proper style control is added into c-ts-mode.			
Last updated on:	2025-04-28	PEL provides the following	g set of mode-specific key prefixes: <f11> SPC c as well as <f12> and M-<f12></f12></f12></f11>	
Open this PDF file. See also: <u>∑ Help/Info</u>	<f11> SPC c <f1> <f12> <f1></f1></f12></f1></f11>	(pel-help-pdf &optional OPEN-WEB-PAGE)	Open the <u>\$\text{\$\}\$}}\$}\$\text{\$\text{\$\text{\$\text{\$\text{\$\text{\$\text{\$\text{\$\text{\$\text{\$\text{\$\text{\$\text{\$\text{\$\text{\$\text{\$\texitt{\$\text{\$\text{\$\text{\$\text{\$\text{\$\text{\$\text{\$\text{\$\te</u>	
© Customize PEL C support	• <f12> <f2> • <f21> # <f2></f2></f21></f2></f12>	(pel-customize-pel &optional OTHER-WINDOW)	Customize PEL C support.  • If OTHER-WINDOW is non-nil (use <b>C-u</b> ), display in another window.	
© Customize Emacs C support	<f12> <f3></f3></f12>	(pel-customize-library &optional OTHER-WINDOW)	Customize Emacs C support: c, c-macro, bison-mode, electricity  • If OTHER-WINDOW is non-nil (use <b>C</b> - <b>u</b> ), display in another window.	
© Customize Emacs C pre-processor support	<f12> # <f3></f3></f12>	(pel-customize-library &optional OTHER-WINDOW)	Customize Emacs C pre-processor support: hide-ifdef.  • If OTHER-WINDOW is non-nil (use <b>C</b> - <b>u</b> ), display in another window.	
CC Mode Style Management Learn/Modify style used in current buffer	You can impose an indentat You can also adjust the styl uses. It learns the style and Use this to adapt to sour	<ul> <li>Automatic indentation, brace format style and several other C stylistic elements are controlled by the CC Mode and the CC mode variables.</li> <li>You can impose an indentation style by customization.</li> <li>You can also adjust the style to what is used in the current buffer: Emacs provides the following commands to parse the source code and identify the style it uses. It learns the style and sets the style controlling variables from what it detects in the buffer.</li> <li>Use this to adapt to source code written by others and want to continue using the same style, or to modify the style.</li> <li>For the following commands all commands that use a key binding that ends with an upper case letter install the style.</li> </ul>		
Show/Modify syntactic context	С-с С-о	(c-set-offset SYMBOL OFFSET &optional IGNORED)	Change the value of a syntactic element symbol in 'c-offsets-alist'.  • SYMBOL is the syntactic element symbol to change and OFFSET is the new offset for that syntactic element.   • Use this to <b>modify</b> a specific style, like how something is indented.	
Show syntactic information for current line	C-c C-s	(c-show-syntactic- information ARG)	Show syntactic information for each syntactic element present on the current line.  Display the syntactic information list: style and position highlight the reference position(s) listed as argument to the syntactic list.  Each list starts with a <b>syntactic symbol</b> with zero or several reference positions.  With universal argument, inserts the analysis as a comment on that line.	
Guess the style used in the current buffer, do not install it	<f12> <f4> g g</f4></f12>	( <b>c-guess-buffer-no-install</b> &optional ACCUMULATE)	Guess the style on the whole current buffer; don't install it.  If given a prefix argument (or if the optional argument ACCUMULATE is non-nil) then the previous guess is extended, otherwise a new guess is made from scratch.	
Guess the style of the code in the buffer and install it.	<f12> <f4> g B</f4></f12>	(c-guess-buffer &optional ACCUMULATE)	Guess the style on the whole current buffer, and install it.  • The style is given a name based on the file's absolute file name.  • If given a prefix argument (or if the optional argument ACCUMULATE is non-nil) then the previous guess is extended, otherwise a new guess is made from scratch.	
<u>Guess style</u> in the region and install it.	<f12> <f4> g G</f4></f12>	( <b>c-guess</b> & optional ACCUMULATE)	Guess the style using the first 'c-guess-region-max' bytes of the file, and install it.  The c-guess-region-max user-option defaults to 50,000 bytes, nil means all buffer.  The style is given a name based on the file's absolute file name.  If given a prefix argument (or if the optional argument ACCUMULATE is non-nil) then the previous guess is extended, otherwise a new guess is made from scratch.	
Guess the style of a region and install it.	<f12> <f4> g R</f4></f12>	(c-guess-region START END &optional ACCUMULATE)	Guess the style on the region and install it.  The style is given a name based on the file's absolute file name.  If given a prefix argument (or if the optional argument ACCUMULATE is non-nil) then the previous guess is extended, otherwise a new guess is made from scratch.	
Set buffer style to guessed style and install it.	<f12> <f4> g I</f4></f12>	(c-guess-install &optional STYLE-NAME)	Install the latest guessed style into the current buffer.  This guessed style is a combination of 'c-guess-guessed-basic-offset', 'c-guess-guessed-offsets-alist' and 'c-offsets-alist'.  The style is entered into CC Mode's style system by 'c-add-style'. Its name is either STYLE-NAME, or a name based on the absolute file name of the file if STYLE-NAME is nil.	
View Guessed style as a set of Emacs Lisp statements	<f12> <f4> g ?</f4></f12>	( <b>c-guess-view</b> &optional WITH-NAME)	Emit emacs lisp code which defines the last guessed style, so you can put the code into .emacs if you prefer the guessed code.  • "STYLE NAME HERE" is used as the name for the style in the emitted code. If WITH-NAME is given, it is used instead. WITH-NAME is expected as a string but if this function called interactively with prefix argument, the value for WITH-NAME is asked to the user.	
Toggle preprocessor line indentation	<f12> <f4> #</f4></f12>	(c-toggle-cpp-indent-to- body &optional ARG)	Toggle the C preprocessor indent-to-body feature. When enabled, preprocessor directives which are words in 'c-cpp-indent-to-body-directives' are indented as if they were statements.	

**Description Function Keystroke** Note Use following commands to dynamically change the behaviour of important keys such as the return key, delete key, semi-colon, etc.. The CC Mode controls the **CC Mode support** dentation and bracket style and what happens when electric characters are typed (when electric mode is activated).

CC Mode state displayed in the mode line: \$C{...} where: Behaviour control 2 is the CC mode programming language name: C, C++, ObjC, etc... Use <f12> <f4> ? to • C is the C comment style: '\*' for block command (/\* \*/) and '/' for line comments (//)
• {...} are the other electric flags: '1' for electric mode, 'a' for auto-newline mode, 'h' for hungry mode, 'w' for subword mode display the current state Toggle the electric indentation feature done with the electric character keys. (c-toggle-electric-state Toggle Electric state Optional numeric ARG, if supplied, turns on electric indentation when positive, turns it off when &optional ARG) <f12> <f4> e negative, and just toggles it when zero or left out. (c-set-style STYLENAME Set indentation style Set the bracket/indentation style for the current buffer. &optional DONT-OVERRIDE) • <f12> <f4> s · Prompts for the name. Supports tab completion (so use tab to see the list). Can be one of the <u>values supported by Emacs</u> but you can also add your customized mode with some Emacs Lisp code. Change indentation width (pel-cc-set-indent-width Interactively change the Indentation with for current buffer to NEW-WIDTH. <f12> <f4> TAB for current buffer &optional NEW-WIDTH) • Prompt for new value. Use 0 to restore value specified by configuration (pel-c-indent-width). This can be used to change indentation several times in a file. Toggle syntactic indentation (c-toggle-syntactic-indentation & optional ARG) Toggle syntactic indentation. Toggle if no ARG or if ARG is 0.

• With positive ARG turn on syntactic indentation, turns it off when negative. <f12> <f4> i When syntactic indentation turned on (the default), the indentation functions and electric keys indent according to syntactic context keys, when applicable.
When it's turned off, the electric keys don't reindent, the indentation functions indents every new line to the same level as the previous nonempty line, and M-x c-indent-command adjusts the indentation in steps specified by 'c-basic-offset'. The indentation style has no effect in this mode, nor any of the indentation associated variables, e.g. 'c-special-indent-hook' Toggle Comment Style (c-toggle-comment-style Toggle the C comment style between block/C-style ( /\* \*/ ) and line/C++-style ( // ) comments. • C-c C-k With optional numeric ARG, switch to block comment style when positive, to line comment style
when negative, and just toggles it when zero or left out. &optional ARG) <f12> <f4> M-; modeline: C/\* or C//
for. C/block or line The C++ style // comments (also called line comments) is compatible with C since C-99. <f12> <f4> DEL Toggle hungry-delete-key feature. Affects < DEL> and C-d keys. **Toggle Hungry Delete** (c-toggle-hungry-state Optional numeric ARG, if supplied, turns on hungry-delete when positive, turns it off when &optional ARG) mode negative, and just toggles it when zero or left out.

When the hungry-delete-key feature is enabled (indicated by "/h" on the mode line after the mode name) the delete key gobbles all preceding whitespace in one fell swoop. Toggle variable *pel-newline-does-align* for the local buffer: toggles how 'pel-newline-and-indent-below' operates: If *pel-newline-does-align* is t, it aligns several syntactic element in the current Toggle text alignment on <f11> M-RET (pel-toggle-newlineindent-align) pel-newline-and-indentbelow See also: block: the comments, the assignments. • 🔡 Identify modes where *pel-newline-does-align* is automatically activated (set to t) by adding the ∑ Align major mode to the list in the **pel-modes-activating-align-on-return** user option. This affects the behaviour of the following commands: • <u>∑ Indentation</u> Ť • pel-cc-newline (assigned to **RET** in CC modes like c-mode, c++-mode and d-mode). pel-newline-and-indent-below (assigned the M-RET) Toggle <u>auto-newline</u> feature.

• Optional numeric ARG, if supplied, turns on auto-newline when positive, turns it off when negative, Toggle auto-newline (c-toggle-auto-newline • C-c C-a &optional ARG) insertion mode <f12> <f4> M-RET and just toggles it when zero or left out.
Turning on auto-newline automatically enables electric indentation. + When the auto-newline feature is enabled (indicated by "/la" on the mode line after the mode name) newlines are automatically inserted after special characters such as brace, comma, semicolon, and colon. Change RET key <f12> <f4> RET (pel-cc-change-newline-Change the way the RET key behaves in the CC modes and display the new mode in the echo area. behaviour: select return Changes from one mode to the next and then rotate to the first one. The modes are: mode) mode. 1. context-newline: the default: uses (c-context-line-break) with the extra ability to repeat its execution with an argument. t 2. newline-and-indent: uses (newline ARG t) to insert newline and indent. 3. just-newline-no-indent: uses (electric-indent-just-newline ARG) ■ Emacs default is to use newline. PEL sets the default to c-context-line-break which provides more functionality for CC modes. A mode change is local to the current buffer and does not affect RET key behaviour in the other buffers using the same mode. PEL user option pel-initial-c-newline-mode can be set to change the default for c-mode. **Display current Mode** <f12> <f4> ? (pel-cc-mode-info) Display information about current **<u>CC mode</u>** derivative for the current c-mode buffer. settings The information displayed in specialized help buffer includes the following: CC mode style currently active, along with a list of styles associated with current mode. Change it for the current buffer with C-c . or <f12> <f4> s. The Notes: Emacs the **c-default-style** user option defines associations between major modes and the style to use. PEL provides the **pel-c-backet-style** that is used to set the style for c-mode. Use <f12> from a c-mode buffer to access the customization buffer to change it. C Language Revisions
 C coding styles:
 GNU Coding Standard / C RET (return key) mode. Change with pel-cc-change-newline-mode (<f12> <f4> RET). Whether return performs alignment. Change that with pel-toggle-indent-align (<f11> M-RET). Linux kernel coding style State of <u>electric C characters</u> (toggle it on/off with c-toggle-electric-state (C-c C-1 or <f12> <f4> e): whether it is active or not, and when active what character(s) exhibit electric behaviour. Barr Group C if auto-newline on some characters (';' and some other based on style) is active. Toggle it with C-c C-a or <f12> <f4> M-RET. Coding Standard The fill column: the column where force line wrap is done when the auto-fill-mode is active. Toggle auto fill mode with <f11> RET. Tab width and whether hard tabs are used. These are set by the user options pel-c-tab-width and pel-c-use-tabs. Carnegie Mellon
 majerle C99/later In c-mode buffer use <f12> <f2> to open the appropriate customization buffer to change them. Remember that tab width does not identify the indentation. It controls the spacing used in some commands moving point to the next tab stop column. Formatter clang-format Indentation is controlled separately. See next line. Indentation width controlled by **c-basic-offset** normally set by **pel-c-indent-width** in PEL and whether syntactic indentation mode is active. Shows how it is set and whether it was override by executing the pel-cc-set-indent-width command for this buffer (use <f12> <f4> TAB) for that command. The style currently used for indentation and bracket positioning (they should have the same value). Emacs identifies several built-in styles but you can create your own. The example below shows "bsd" with is another name for the Allman style. You can dynamically change for the current buffer with c-set-style command (C-c . or <f12> <f4> s). d CC Mode styles identify everything, including the number of indentation columns. PEL configures the style from the requested pel-c-bracket-style and then updates the indentation and other settings from the PEL user option requested. This allows you to slightly modify an existing style without having to create a new style name for it. The comment style. Supports C-style (/\* \*/) and C++-style (//) comments since both are now accepted in C since C99. • This can be changed dynamically for the current buffer with the c-toggle-comment-style command (C-c C-k or <f12> <f4> M-; ). C comment continuation lines can use 1 or 2 star characters: if a second one is used on a comment continuation line the remainder of the comment continuation lines Example: Notice the used two stars, otherwise only one is used. name of the PEL user-Whether hungry delete is used by DEL and C-d. Toggle this for the current buffer with c-toggle-hungry-state (<f12> <f4> DEL). options that set the The file search methods and parameters used by pel-open-at-point (see sections below). significant feature c-mode state: controlling Emacs variables active style
RET mode
Electric characters bsd. c-default-style: (bsd)
context-newline, and aligns (comments, assignments, etc...)
active on: #\*/(){}:;, in the message More info is shown in Auto newline fill column that buffer as buttons that 80, auto-filling: off. provide access to more help and ability to when c-mode buffer is opened.

de( t) when c-mode buffer is opened.

t(4) when c-mode buffer is opened. Set via: pel-c-tab-width(4) ==> tab-width(4)
Set via: pel-c-use-tabs( t) ==> indent-tabs-mode(
Set via: pel-c-indent-width(4) ==> c-basic-offset(4) Tab width Tab width
Indentation chars
Indent width
Syntactic indent
c-indentation-style
PEL Bracket style hard-tabs & spaces customize the values. bsd Block comments: /\* \*/ , continued line start with \*
off, but the Fll-® and Fll-® keys are available.
None found, searching for files identified in pel-project-root-identifiers: (.git .hg .projectile .pel-project) Comment style Hungry delete

Project root : None
File finder method : gene
pel-ffind-executable: fd

Description	<u>Keystroke</u>	Function	<u>Note</u>
C Code Help	-	nsion packages that can help w	
Get man help about C	• <f11> ? m</f11>	(man MAN-ARGS)	Open a Man page inside an Emacs window. See <u>v Help/Info</u> for more info about man.
code See: <u>∑ Help/Info</u>	• M- <f8> • #-M</f8>		Inside a C buffer, you can use it to request man help about a C function or structure. A large amount of information about C library code is available in man form under the various Unix-like platforms.
Toggle <u>c-eldoc</u> mode	<f12> ? e</f12>	(pel-toggle-c-eldoc-mode)	Toggle c-eldoc mode on/off.  • The c-eldoc mode provides the C prototype information in the echo area for the function at point.
Display function doc under the cursor	<f11> SPC c ? e</f11>		It currently appears when typing a new function with its arguments inside the code.
		mproved into providing the info	pel-use-c-eldoc is set to t
Electric Keys	<u> </u>		when the electrical state is active in a buffer using c-mode.  toggle-electric-state (C-c C-1 or <f12> <f4> e).</f4></f12>
#		(c-electric-pound ARG)  dle it specially according to the a literal or a macro, nothing specially	Insert a "#".  e variable 'c-electric-pound-behavior', which can only be nil or 'alignleft'. If a numeric ARG is pecial happens.
()	• ( )	(c-electric-paren ARG)	Insert a parenthesis. on-nil, the line is reindented unless a numeric ARG is supplied, or the parenthesis is inserted inside a
			may get added or removed; see the variable 'c-cleanup-list'. nil, some newline cleanups are done if appropriate; see the variable 'c-cleanup-list'.
<b>{}</b>	• { }	(c-electric-brace ARG)	Insert a brace.
	a) If the auto-newline fear	ture is turned on (indicated by t'. indented. The original line is a	and a numeric ARG hasn't been supplied, the command performs several electric actions: "/la" on the mode line) newlines are inserted before and after the brace as directed by the settings in also reindented unless 'c-syntactic-indentation' is nil. s based on the settings of 'c-cleanup-list' are done.
:	:	(c-electric-colon ARG)	Insert a colon.
	a) If the auto-newline fear hanging-colons-alist'.     b) Any auto-newlines are	ture is turned on (indicated by indented. The original line is a	and a numeric ARG hasn't been supplied, the command performs several electric actions:  "/la" on the mode line) newlines are inserted before and after the colon based on the settings in 'c-  also reindented unless 'c-syntactic-indentation' is nil.  b colons will be "cleaned up" leaving a scope operator, if this action is set in 'c-cleanup-list'.
;,	• ; ,	(c-electric-semi,	Insert a comma or semicolon.
	a) When the auto-newline criteria' for how newling b) Any auto-newlines are	e feature is turned on (indicated e insertion is determined. indented. The original line is a	In numeric ARG hasn't been supplied, the command performs several electric actions: d by "/la" on the mode line) a newline might be inserted. See the variable 'c-hanging-semi&comma- also reindented unless 'c-syntactic-indentation' is nil. ace list or a semicolon following a defun might be cleaned up, depending on the settings of 'c-cleanup-
Electric pairs	It is also possible to control the  Type the first of a pair to ins	ert this one and its matching ch	activating the <b>electric-pair-mode</b> in the buffer.  haracter for (), [], $\{\}$ , "" and ".  line lighter set by the pel-electric-pair-lighter is shown. This defaults to $\{E(1)\}$
Toggle electric-pair- mode in current buffer † Lighter:= ε(Ι)	<f11> M-e</f11>	(electric-pair-local-mode &optional <u>ARG</u> )	Toggle automatic parens pairing (Electric Pair mode) and org-mode special pair electric keys only in this buffer. With this typing (inserts the matching). Same for other pairs.  • With a prefix argument ARG, enable Electric Pair mode if ARG is positive, and disable it otherwise.  • Electric Pair mode is a global minor mode. When enabled, typing an open parenthesis automatically inserts the corresponding closing parenthesis, and vice versa. (Likewise for brackets, etc.). If the region is active, the parentheses (brackets, etc.) are inserted around the region instead.
Use Linux Kernel code style	<f12> M-k</f12>	(pel-linux-kernel-code- style)	Manually activate Linux Kernel Coding Style regardless of file content.
Insert New Line(s)	the point also moves to the pri • With PEL the default behaving command (bound to <f12> • The pel-cc-newline command the pel-cc-newline command the pel-cc-newline command the price of th</f12>	oper indentation according to to our can be selected by custom M-RET) see the CC-Mode belond also aligns comments and a	Indode electric mode is active or not. When it is not active it simply inserts a new line. When it is active it syntactic context. The following commands can also be used.  Initiation and modified dynamically for the current buffer with the pel-cc-change-newline-mode naviour control section above.  Insignment in the code block if the pel-modes-activating-align-on-return user option list includes an also be modified by the pel-cc-change-newline-mode command ( <f11> M-RET).</f11>
Insert a new line and operate according to the currently active selected return mode.  With PEL, modify	RET	(pel-cc-newline &optional N)	Insert a newline and perhaps align. With argument N repeat N times.  • For newline insertion, operate according to the value of the variable 'pel-cc-newline-mode' which selects one of 3 commands (see the full description in the 3 row below):  • c-context-line-break (PEL default for RET)  • newline (Emacs default for RET)  • electric-indent-just-newline
behaviour with <f12> M- RET.</f12>			by the context.  In the context and indent according to the syntactic context, unless 'c-syntactic-indentation' is nil, in which apply line instead.  In the continuation backslash is inserted before the line break and aligned appropriately. The appropriate comment prefix (see the 'c-comment-prefix-regexp' and 'c-block-comment-prefix' mment doesn't count as inside it.
See also:  •   \( \tilde{\mathbb{Z}} \) Filling/Justification	Use: (newline &optional ARG INTERACTIVE): Insert a newline, and move to left margin of the new line if it's blank.  • With ARG, insert that many newlines.		is marked with the text-property 'hard'. inal new line that it adds, and reindents the preceding line. st-newline. r is greater than the value of 'fill-column' and ARG is nil.
	Use: (electric-indent-just • With ARG, insert that r		ewline, without any auto-indentation.
Insert an indented line below unbroken current line See also: Indentation	• M-RET • <fi1> <tab> RET</tab></fi1>	(pel-newline-and-indent-below)	Insert an indented line just below current line regardless of the position of point and move point to the beginning of the next line. Does not break current line.  For example if point is at the beginning, middle or end of the line it just insert a new line below the current one at the proper indentation.  If pel-newline-does-align is t, it aligns several syntactic element in the current block: the comments, the assignments.  You can toggle this on/off with <f11> M-RET.  But lentify modes where pel-newline-does-align is automatically activated (set to t) by adding the c-mode to the list in the pel-modes-activating-align-on-return user option.</f11>

Description	<u>Keystroke</u>	Function	Note Note
Insert a newline	С-ј	(electric-newline-and- maybe-indent)	Insert a newline.  • If 'electric-indent-mode' is enabled, that's that, but if it is *disabled* then:
	<ul> <li>In programming langua</li> </ul>	age modes, this is the same as	
On an Navy Line in	·		mmand indents to the column specified by the function 'current-left-margin'.
Open New Line in Context	C-O	(c-context-open-line)	Insert a line break suitable to the context and leave point before it.
See also:  • <u>S Whitespace</u>	<ul> <li>This is the 'c-context-line-break' equivalent to 'open-line', which is normally bound to C-o. See 'c-context-line-break' for the details.</li> <li>Normally C-o is bound to open-line. PEL rebinds it to c-context-open-line for the CC modes.</li> <li>If you want to open the line without indenting the next use open-line via <f12> C-o</f12></li> </ul>		
Open new line	• <f12> C-o • M-<f12> C-o</f12></f12>	(open-line N)	Insert a newline and leave point before it. With arg N, insert N newlines.  • If there is a fill prefix and/or a 'left-margin', insert them on the new line if the line would have been blank.
C Comments	2 more characters have electri C supports 2 types of comments '*': Block Comments '': Line Comments (	nents: : /* comment	
Comment characters, wrapping long comment lines		(c-electric-slash ARG)	<ul> <li>Insert a slash character.</li> <li>If the slash is inserted immediately after the comment prefix in a c-style comment, the comment might get closed by removing whitespace and possibly inserting a "*". See the variable 'c-cleanup-list'.</li> <li>Indent the line as a comment, if: <ol> <li>The slash is second of a "//" line oriented comment introducing token and we are on a comment-only-line, or</li> <li>The slash is part of a "*/" token that closes a block oriented comment.</li> <li>If a numeric ARG is supplied, point is inside a literal, or 'c-syntactic-indentation' is nil or 'c-electric-flag' is nil, indentation is inhibited.</li> </ol> </li></ul>
See also:  E Filling/Justification	*	(c-electric-star ARG)	Insert a star character.  If 'c-electric-flag' and 'c-syntactic-indentation' are both non-nil, and the star is the second character of a C style comment starter on a comment-only-line, indent the line as a comment.  If a numeric ARG is supplied, point is inside a literal, or 'c-syntactic-indentation' is nil, this indentation is inhibited.  With this key being electric it becomes easy to type the following two styles of multi-line block comment:  /* Two star  ** continuation  ** prefix for  ** multi-line  ** C comment.  */  /* Single star  * prefix for  * multi-line  * C comment.  */  * When typing the '*' at the beginning of the line, it indents automatically. If another '*' is typed, indentation is set to allow a two-star continuation, otherwise it is placed for a single star continuation.  When auto-fill-mode is active, when you type a comment that would be longer than the line,
Comment/un-comment	M-;	(pel-c-comment-dwim	the line is wrapped and the comment continuation string used is automatically inserted. (toggle it with <f11> RET)  Comment line or region with // or /* */ style comments depending on the comment style currently</f11>
★★ See also: <u>▼ Comments</u>	·	ÄRG)	used in the buffer.  • When no marked region and no comment:  • On empty line: insert comment starter at the proper indentation level.  • Typed again: move it toward end of line.  • On line with code: insert comment starter after the code for an end-of-line comment  • With marked un-commented region:  • ***  Comment region with style selected by pel-c-multiline-comments user-option:  • default (like comment-dwim): each line is commented with a /* * /  • 1: single start multi-line comment (see example in box above)  • 2: double star multi-line comment (see example in the box above)  • With marked commented region:  • removes the comment.  • When a prefix ARG is specified, call 'comment-kill'. Else, call 'comment-indent'.  • You can toggle between C-style /* */ and C++ style // comments (compatible with C since C-99) <f12> M-;</f12>
Comment/un-comment	C-c C-c		Comment or uncomment each line in the region.
See also: <u>▼ Comments</u>	<ul> <li>Numeric prefix ARG means use ARG comment characters. If ARG is negative, delete that many comment characters instead.</li> <li>The strings used as comment starts are built from 'comment-start' and 'comment-padding'; the strings used as comment ends are built from 'comment-end' and 'comment-padding'. By default, the 'comment-start' markers are inserted at the current indentation of the region, and comments are terminated on each line (even for syntaxes in which newline does not end the comment and blank lines do not get comments). This can be changed with 'comment-style'.</li> <li>If you try this when no region is marked and the /* */ style comments is active, the comment ends on the next space, which is probably not what you want. The command comment-dwim works better and pel-c-comment-dwim (above) even better.</li> </ul>		
Fill current paragraph See also:  ∑ Filling/Justification	• M-q • <f12> F • M-<f12> F</f12></f12>	( <b>c-fill-paragraph</b> &optional ARG)	Like <f11> t f p but handles // and /* */ style comments.  • If any of the current line is a comment or within a comment, fill the comment or the paragraph of it that point is in, preserving the comment indentation or line-starting decorations (see the 'c-comment-prefix-regexp' and 'c-block-comment-prefix' variables for details).  • If point is inside multiline string literal, fill it. This currently does not respect escaped newlines, except for the special case when it is the very first thing in the string. The intended use for this rule is in situations like the following:  char description[] = "\ A very long description of something that you want to fill to make nicely formatted output.";  • If point is in any other situation, i.e. in normal code, do nothing.  • Optional prefix ARG means justify paragraph as well.</f11>
Toggle subword-mode See also:  Text Modes	• <f11> t m b • <f12> M-b • M-<f12> M-b</f12></f12></f11>	(subword-mode &optional ARG)	Toggle subword-mode: a minor mode that treats sections of <a href="mailto:camelCase">camelCase</a> and <a href="mailto:PascalCase">PascalCase</a> as distinct words.  • With a prefix argument ARG, enable Subword mode if ARG is positive, and disable it otherwise.
Hide/Show comments See also:  • © Comments	<f11> ; ;</f11>	(hide/show-comments- toggle &optional START END)	Toggle hiding/showing of comments in the active region or whole buffer.  • If the region is active then toggle in the region. Otherwise, in the whole buffer.  in this requires the <a href="hide-comnt.el">hide-comnt.el</a> package (see <a href="Decomments">Decomments</a> ). <a href="Decomments">Decomments</a> ). <a href="Decomments">Decomments</a> ) PEL activates it when the <a href="pel-use-hide-comnt">pel-use-hide-comnt</a> user option is t.

Description	Warratura la a	F	M.L.
<u>Description</u>	Keystroke	Function "Section "Section "Section "Section "Section "Section "Section "Section Section Secti	Note ngry whitespace deletion" that can also be used in every mode.
Hungry Deletion of Whitespace	<ul> <li>EPEL provides the convenient keys with the <f11> prefix keys for those 2 commands, available in all modes.</f11></li> <li>In modes compatible with the CC Mode (e.g. for C, C++, D, Java, Pike, etc) it is also possible to activate the Hungry Delete Mode to modify the behaviour of the simple <del> and C-d, to perform hungry deletions. That's not currently supported in other modes.</del></li> <li>When the Hungry Delete Mode is on, the mode-line displays a 'h' to the right of the '//l' indication of electric mode.</li> <li>The Hungry Mode also activates the key prefixes below that start with C-c. They are listed but remember they are only available once the Hungry state mod is activated (and that can only be done in modes that are CC Mode compatible).</li> <li>In modes derived from CC Mode you can also activate the hungry state to make standard delete commands delete hungrily, but that does not work for other modes. PEL provides the <f12> M-DEL key for those modes (like C).</f12></li> <li>Toggle hurry deletion mode of the DEL and C-d key for the current buffer with c-toggle-hungry-state (<f12> M-DEL).</f12></li> </ul>		
Delete preceding char or all preceding whitespace.  See also: <u>See Also:</u> <u>Cut &amp; Paste</u>	• C-c DEL • C-c @ • C-c C-@ • C-c C-DEL • <f11> @ @ • <f11> DEL DEL</f11></f11>	(c-hungry-delete- backwards)	Delete the preceding character or all preceding whitespace back to the previous non-whitespace character.  In terminal mode, even though C-@, C- <backspace> and C-DEL are not available, they are mapped to the non-control key so attempting to type them end up invoking the command anyway because the first key bindings are recognized.  With PEL, the <f11> @ Dinding is always available, in all modes.  The other keys are only available in modes derived from the CC Mode. This prevents conflicts with</f11></backspace>
Delete next char or all following whitespace.  See also:  See Also: Cut & Paste	• C-c C-d • C-c D • C-c C-D • C-c C-C+delete> • <f11> D</f11>	(c-hungry-delete-forward)	other modes that may use the popular C-c bindings.  Delete the following character or all following whitespace up to the next non-whitespace character.  In terminal mode, even though C-w and C- <delete> are not available, they are mapped to the non-control key so attempting to type them end up invoking the command anyway because the first key bindings are recognized.  With PEL, the <f11> binding is always available, in all modes. The other keys are only available in modes derived from the CC Mode. This prevents conflicts with other modes that may use the popular C-c bindings.</f11></delete>
Indentation	behaviour control section abov You can also explicitly request  The first set of commands p	ve. indentation using the commar erform syntactic indentations	-Mode state, the style and whether electric mode for some characters is active. See CC Mode and below.
Indent current line or region	<tab></tab>	(c-indent-line-or-region &optional ARG REGION)	Indent active region, current line, or block starting on this line.
See also:  • ∑ Indentation	<ul> <li>Behaviour depends on syntactic-indentation mode (enabled by default but can be toggled on/off with the <f12> M-i key):</f12></li> <li>With syntactic-indentation on (the default):</li> <li>In Transient Mark mode, when the region is active, reindent the region.</li> <li>Otherwise, with a prefix argument, rigidly reindent the expression starting on the current line.</li> <li>Otherwise reindent just the current line.</li> <li>This might seem strange for new Emacs users, but it ends up being very useful. You can type <tab> anywhere in the line to adjust the indentation of the current line or everything in the marked area if a block is marked.</tab></li> <li>With syntactic-indentation off:</li> <li><tab> always indent current line by one level</tab></li> <li>C-u - <tab> or M <tab> always un-indent current line by one level.</tab></tab></li> <li>Indenting marked region is done without syntax knowledge and at the same level as previous line.</li> <li>If you want to indent rigidly, bound to C-x <tab> and to <f11> <tab><tab><tab><tab><tab< tr=""> </tab<></tab></tab></tab></tab></f11></tab></li></ul>		
Indent lines of list after point	C-M-q	d to <b>M-i</b> to insert spaces to the (indent-pp-sexp &optional ARG)	Indent each line of the list starting just after point, or pretty-print it.  • A prefix argument ( <b>C-u</b> ) specifies pretty-printing. Pretty-printing essentially uses more lines as it
See also:  • <u>Indentation</u>			places the beginning of each list on a new line.
Indent current function or class	C-c C-q	(c-indent-defun)	Indent the content of the current top-level function or class. Leaves point unchanged.
Indent a region	C-M-\	(indent-region START END &optional COLUMN)	Indent each nonblank line in the region.  • A numeric prefix argument specifies a column: indent each line to that column.  • With no prefix argument, the command chooses one of these methods and indents all the lines with it:  1. If 'fill-prefix' is non-nil, insert 'fill-prefix' at the beginning of each line in the region that does not already begin with it.  2. If 'indent-region-function' is non-nil, call that function to indent the region.  3. Indent each line via 'indent-according-to-mode'.  When a region is marked you can also use the simple <tab> to do the same when syntactic-indentation is active.</tab>
Non Syntactic	Emacs provides the following	command to indent without reg	gards to semantics. More information on indentation is available in the <u>Indentation</u> table.
Indentation	For most editing scenarios while the other 2 use pel-c-inc		th and pel-c-indent-width to the same value: the first 2 commands use the value of pel-c-tab-width
Insert spaces or tabs to next defined tab-stop column See also:      Indentation	M-i	(tab-to-tab-stop)	Insert spaces or tabs to next defined tab-stop column.  The exact location of the next tab stop is identified by the value of the tab-stop-list and tab-width for the current buffer.  With PEL, the tab-stop interval is controlled by the value of pel-c-tab-width.  PEL sets tab-width to the value of pel-c-tab-width for each c-mode buffer.
Indent/Unindent rigidly  See also:  • <u>E Indentation</u> • <u>E Key-Chords</u>	• C-x <tab> • <f11> <tab> <tab> • <tab>q</tab></tab></tab></f11></tab>	(pel-indent-rigidly &optional N)	<ul> <li>Indent rigidly the marked region or current line N times tab-width columns.</li> <li>If a region is marked, it uses 'indent-rigidly' and provides the same prompts to control indentation changes.</li> <li>If no region is marked, it operates on current line(s) identified by the numeric argument N (or if not specified N=1): <ul> <li>N = [-1, 0, 1] : operate on current line</li> <li>N &gt; 1 : operate on the current line and N-1 lines below.</li> <li>N &lt; -1 : operate on the current line and (abs N) -1 lines above.</li> </ul> </li> </ul>
	indent-rigidly Indent all lines s  If called interactively w <right>, S-<left< td="">   Both of these commands active capabilities are controlled by the second of the second of</left<></right>	starting in the region. ith no prefix argument, activate >, or S- <right>. rate a transient mode where Erne variable indent-rigidly-map virigidly-right-to-tab-stop rigidly-left-to-tab-stop rigidly-left es the transient mode. Left&gt; keys indent/de-indent terval is controlled by the value the value of pel-c-tab-width for the cua-mode uses C-x, to invo</right>	to the next tab-stop position, which is controlled by the <b>tab-width</b> user option.

<u>Description</u>	Keystroke	Function	Note	
Indent line(s) rigidly	• <f6> <tab></tab></f6>	(pel-indent-lines	Indent current or marked lines by N indentation levels controlled by <b>pel-c-indent-width</b> .	
See also:	• <f11> <tab> c</tab></f11>	&optional N)	Works with point anywhere on the line.  gument N can specify more than one indentation level. It defaults to 1. If a negative number is	
• <u>\(\tilde{\ti}</u>	<ul> <li>specified, 'pel-unindent-lines' is used. If a region is marked, the function does not deactivate it to allow repeated execution of the command. It also modifies the region to include all characters in all affected lines. Use C-g to de-activate the region.</li> <li>Handles presence of hard tabs:</li> <li>If indent-tabs-mode is non-nil the indentation is created with a mix of hard-tabs and space characters.</li> <li>If indent-tabs-mode is nil, any hard tab in the indentation of the marked lines is replaced by the proper number of spaces. Hard tabs after first non-whitespace</li> </ul>			
	character on the line are I			
Un-indent line(s) rigidly  See also:  Indentation	• <backtab> • <f6> <backtab> • <f11> <tab> C</tab></f11></backtab></f6></backtab>	( <b>pel-unindent-lines</b> &optional N)	<ul> <li>Un-indent current line or marked lines by N indentation levels controlled by pel-c-indent-width.</li> <li>Works with point is anywhere on the line.</li> </ul>	
	<ul> <li>All lines touched by the region are un-indented.</li> <li>If region was marked, the function does not deactivate it to allow repeated execution of the command.</li> <li>If a region was marked, the function does not deactivate it to allow repeated execution of the command. It also modifies the region to include all charact affected lines. Use C-g to de-activate the region.</li> <li>Handles presence of hard tabs: If indent-tabs-mode is: <ul> <li>non-nil the indentation is created with a mix of hard-tabs and space characters.</li> <li>nil, any hard tab in the indentation of the marked lines is replaced by the proper number of spaces. Keeps hard tabs after first non-whitespace characters.</li> </ul> </li> </ul>			
Open file at point  See also:  File mngt  Atter changing user- options you currently must restart PEL by executing pel-init or restarting  Emacs!	The following command allow opening files from the file name taken at point (the cursor location).  In a c-mode buffer the command is specialized to be more useful for C programming and has the extra capability of searching files where header files are stored. The search method is controlled by the following user-options:  pel-c-file-finder-method identifies one of 4 supported method of identifying the header files. See their descriptions below. identifies one of 4 supported method of identified by pel-ffind-executable user-option. it is of extra directory trees also searched by the tool identified by pel-ffind-executable user-option. it he pel.ini file, a INI-format configuration file. The value mapped to that key identifies the list of directories to search for that tool-chain. The name of the tool chain can be overridden by the value of the environment variable PEL_CC_FIND_TOOLCHAIN, which is read and used when Emacs starts up (or pel-init is executed). Use the command pel-cc-set-file-finder-ini-tool-name to change the currently used tool chain name.  Note that when using the Ido completion mode, it is possible to instruct Ido to use a file name at point as the basis for the file name to open.  This Ido behaviour is controlled by the ido-use-filename-at-point user-option. With PEL you can control it globally or locally with <f11> f M</f11>			
Show active file finder setup for current buffer	• <f12> <f4> f • <m-f12> <f4></f4></m-f12></f4></f12>	(pel-cc-find-show-status &optional APPEND)	Print C specific PEL file finding control user-options and variables info inside a *pel-cc-ffind-status*  help-mode buffer.  • Prints current state and values of relevant user-options and variables as buttons you can use to get more info and change the values of the user options.  • Clear previous buffer content by default. Use prefix arg (like C-u) to append instead.	
Set name of Tool-Chain specific include path.  (with pel-ini-file search method)	• <f12> <f4> M-<f6> • <f12> <f4> <f54></f54></f4></f12></f6></f4></f12>	(pel-cc-set-file-finder-ini- tool-name &optional TOOL- NAME)	Change activate value of tool-chain name key identified by value of pel-c-file-finder-ini-tool-name user-option. The change is not persistent.   Bequires pel-use-ini set to t.  Used when the pel-c-file-finder-method is set to pel-ini-file. In that case it effectively select a new set of tool-chain specific directories to search by pel-open-at-point. The directories are identified by the corresponding TTT-c-path key in the [file-finder] section of the pel.ini file.	
Open file or web-page whose name is at point ★★	• M-* • <f11> f . • <u>6y</u></f11>	(pel-open-at-point &optional N)	Open the file, library or the URL, named at point, with potential line & column #s.  • If necessary will search source code files in current project as specified by pel-filename-at-point-finders user-option. Type <f12> <f4>? to show current file search method.  Supports glob characters, partial directory path. When multiple files are found it prompts using the method selected by pel-prompt-read-method user-option.  The 6y key-chord is available if pel-use-key-chord is non-nil. See Key-Chords.</f4></f12>	
C Header File finding control	This command works generically but is also specialized for C major mode: it opens the header file identified by the #include statement.  Aside from generic method described below, the command searches for the header file to open using the method identified by the pel-c-file-finder-method and the pel-c-file-searched-extra-dir-trees user-options. The first one identifies one of the following search method, the other identifies extra directory tree(s) to search using the search tool identified by the pel-ffind-executable user-option:  • generic: the command searches, in current directory and its parents, for a file identifying the parent root directory; a file with a name identified in the pel-project-root-identifiers user-option. Something like .git, .hg, .project or .pel-project by default. Then searches for files inside that directory tree.			
☑ pel-use-ini 🔻	<ul> <li>pel-ini-file: the command searches inside directories identified by lists defined in the pel.ini file which PEL identifies for the project like it does for project marker. The pel.ini file is a .INI file format. When found, it is opened and information inside the file identifies where to search.</li> <li>The file must contain a [file-finder] section with:</li> <li>The project-path key. The value is a list of directories to search recursively.</li> </ul>			
Command is also specialized for:	The currently used t	<ul> <li>One or several TTT-c-path key(s), where TTT is a tool-chain name. The value is a list of directories to search recursively for that tool-chain.</li> <li>The currently used tool chain is identified by the following values in order (first one takes priority on startup):</li> <li>The content of the PEL CC FIND TOOLCHAIN environment variable, if it exists.</li> </ul>		
M reStructuredText     BI - C++	The content of th	e pel-c-file-finder-ini-tool-na	me user-option; which identifies the name of a TTT-c-path key.  noment variables inside the path strings. Use the \$VARNAME format to identify them.	
• \$\mathbb{Y} \cdot - Erlang • \$\mathbb{Y} \cdot - UNIX Shell	<ul> <li>You can modify this to</li> <li>With several TTT-c-</li> </ul>	ol chain name anytime during a <b>path</b> keys inside the <b>pel.ini</b> file	an editing session by typing <f12> <f4> M-<f6> and specifying another name.  e, you can adjust the include path dynamically for various tool chains.</f6></f4></f12>	
Generic Delimiting characters	Directories are not sea     explicit lists: two lists of identify directory names in general the command extraction.	<ul> <li>environment variable name: the name of an environment variable (like INCLUDE) that holds a list of directory names to search files in.</li> <li>Directories are not searched recursively for the last 2 options.</li> <li>explicit lists: two lists of directory names: one list holds the project directory names, the other hold the tool and library directory names. The lists may identify directory names indirectly via environment variables. The \$VARNAME format must be used. Directories are not searched recursively.</li> <li>In general the command extracts the file or directory name, and possibly line and column numbers, from text at point and tries to open the file or directory.</li> <li>The generic mode extraction works by identifying the beginning &amp; end of the file/directory/library/URL name string by delimiter characters, one of: tab, newline</li> </ul>		
The complete file detection heuristic is			n and prompts using the method selected by <b>pel-prompt-read-method</b> user-option.	
described in the ∑ File mngt description of the	The default is a very prim	itive function implemented by	PEL. You can select a more powerful <a href="https://www.nee.use-ivy">ivy</a> prompting instead. <b>use-ivy</b> to <b>t</b> and <b>lvy mode</b> will be installed automatically when you restart Emacs.	
same command.  Select target window	Note that the command s	shows all files found by the spe	ecified search method, it does not only use the first one found.	
	Select target window:     Without argument:     If file or directory is already.	<ul> <li>Without argument:</li> <li>If file or directory is already opened in a window, move point to that window and to the line column coordinates if specified following the file name at point.</li> </ul>		
N>20 : open the directory <b>▼</b>	<ul> <li>If no window holds that file, select the target window according to the number of editable windows in frame: if 1, split that window and use the new window, if 2: use the other window, if 3 or more, use the current window.</li> <li>With prefix numeric argument N:</li> <li>N &lt; 0: create a new window and use that.</li> <li>(abs N) &gt; 20: then open the directory instead of the file. Interpret the window position from the N value adjusted: N-20 (or N+20 if N is negative)</li> <li>N = 0: use the 'other' (the next) window.</li> <li>N = 1, 3, 7or above (excluding 8, 9 and 10): select the target window based on the number of editable windows in frame:</li> <li>if 1 window: split that window and use the new window.</li> </ul>			
See function docstring for more info.	<ul> <li>N is: 8: up, 2: down, 4</li> <li>N is 9: force opening macOS Finder, Window</li> </ul>	the file in the OS associated	l application (with N=29 or N=-29, open the file's directory with the OS associated application (eg. open it in the OS default web browser. not allowed.	
Open file with alternate extension Supports:  • $\Sigma$ File-mngt • $\Re I$ - C++	M- <f12> M-f</f12>	(pel-open-file-alternate)	Open a file with same name but an alternate extension.  The new extension depends on the current file extension.  The list of alternate extensions is currently very limited and restricted to C and C++. If the alternate file is not found, save the file basename in the kill ring and prompt for the file name to open.	

Description	<u>Keystroke</u>	Function	Note		
Tempo skeletons	-		ugh the Emacs built-in tempo skeleton mechanism.		
for C	PEL creates key bindings to	invoke the skeletons in the su	pported major modes, using the same key prefix sequence for each mode: <f12> <f12>, with the ader block) as much as possible.</f12></f12>		
See also:	Several aspects of the P	EL Emacs Lisp Source Code S	Style is controlled by the user options inside the <b>pel-c-code-style</b> group. This group can be edited with		
• C Code Templates	pel-c-skel-module-head		ing a user-define module-header comment block.		
<ul> <li><u>Inserting Text</u> for more info and</li> </ul>	<ul><li>pel-c-skel-comment-wi</li><li>pel-c-skel-insert-file-tin</li></ul>		nat of C-style continuation comments.  utomatically updated timestamp is inserted in the file header block.		
information about tempo skeleton and yasnippet	<ul> <li>pel-c-skel-use-separator</li> <li>pel-c-skel-doc-markup</li> </ul>		ks use horizontal separator lines. cumentation markup supported by the templates. Currently 'none' and 'Doxygen' are available.		
template-based text insertion	pel-c-skel-cfile-section	-titles : identifies docum	entation section titles inserted in code files.		
	pel-c-skel-hfile-section	and after the incl	entation section titles inserted in header files. A section titled "." split sections placed before ude guard. If not present all sections are placed after the include guard.		
	pel-c-skell-function-sec	ction-titles : identifies the title	nction templates are inserted in the function description comment.  of the C function templates sections inserted when pel-c-skel-insert-function-sections is t.		
	pel-c-skel-function-defi	- no special com			
		- a Man-page sty	rmat style to describe the function above its code. yle comment block with the sections identified by pel-c-skell-function-section-titles		
		ne-on-first-column : identifies	tempo skeleton loaded from a user specified file name. See the source code example.  whether return type is located on the same line as function name or just above.		
	pel-c-skel-with-license		copy right and code license is specified. An option provide ability to insert open source extract controlled by dice.		
	pel-c-use-include-guard		be of include guard is inserted in header files. The available choices are:		
			oragma once statement assic #ifdef/#define/#endif block using symbol created from file name		
	Emana upor antiona by daf	- use a	#ifdef/#define/#endif block using symbol created from file name and UUID for its uniqueness.		
	effect on a single file or all files	s inside a directory tree. So by	using file and directory variables (see <u>File/Directory Variables</u> ) they can also be used to take default, the user options that control the PEL tempo template take effect globally. If you want to		
	for all files inside a directory tr	ee create a dir-locals file and s	control block at the end of that file. If you want to control the behaviour of the PEL tempo templates store the values of the relevant options variables inside that file. This allows you to control the user		
			and does not affect what you actually type.  e pel-tempo-mode) you can move to the next or previous point of interest (so called <i>tempo-marks</i> ) with		
	· ·		or some other keys like C-c and C-c,.		
∑ Customize PEL C Skeletons layout	<f12> <f12> <f2></f2></f12></f12>	(pel-customize-pel &optional OTHER-WINDOW)	<ul> <li>Customize PEL C skeleton layout.</li> <li>If OTHER-WINDOW is non-nil (use C-u), display in another window.</li> </ul>		
Insert a file header	<f12> <f12> h</f12></f12>	(pel-elisp-file-header)	Insert a file description block. Distinguish between code files and header files.		
			ard is inserted if requested by customization.  It's possible to create a user-specified skeleton this command will use instead of the default.		
	, ,	· · · · · · · · · · · · · · · · · · ·	/templates/c/files repo directory. Access customization buffer with: <f12> <f12> <f2></f2></f12></f12>		
Insert #define	<f12> <f12> d</f12></f12>	(pel-c-define)	Insert a C pre-processor #define statement.  insert the statement on the next line, otherwise insert it on the current line, even if there is text after		
		efore the name of the symbol to			
Insert #include <.h>	<f12> <f12> i</f12></f12>	(pel-c-include-lib)	Insert a C pre-processor #include <> statement to include a library file.		
<ul> <li>Use C-c . to move point to tempo marks.</li> </ul>		peginning of the line and point, sert a new line to place that tex	insert the statement on the next line, otherwise insert it on the current line.  tt on the next line.		
			d point left right before the period. The next tempo mark is placed at the end of the line.		
Insert #include ".h"	<f12> <f12> I</f12></f12>	(pel-c-include-local)	Insert a C pre-processor #include "" statement to include a local file. insert the statement on the next line, otherwise insert it on the current line.		
<ul> <li>Use C-c . to move point to tempo marks.</li> </ul>	If there is text after point, in:	sert a new line to place that tex			
Insert a function	<f12> <f12> f</f12></f12>	(pel-c-function)	Insert a C function definition code and comment template.		
definition with comment block	The command prompts for	the function name and its purp	ose.		
	You can hit return both prompts to specify no text; in that case a tempo skeleton marker is left at the location where the text must be inserted and point is left at the first one.      If you enter a function name it must be a valid C function name (as far as the syntax is concerned). However leading and trailing whitespace is accented and				
	• If you enter a function name, it must be a valid C function name (as far as the syntax is concerned). However leading and trailing whitespace is accepted and trimmed and dash characters ('-') are automatically replaced by underscores ('-') for convenience.				
	<ul> <li>If an invalid name is specified it is erased and you are prompted again. Use M-p to bring the old value back.</li> <li>Prompts for function and purpose maintain separate histories. Use M-p and M-n to navigate in the histories at the prompt. You can also use the <up> and</up></li> </ul>				
	<down> keys. <ul><li>The style of the code inserted</li></ul></down>	ed is controlled by the user opt	ions inside the pel-c-code-style group and the various C style element controls of the CC-mode.		
	Use C-g to cancel at any p	·	e examples in the PEL manual.		
Toggle pel-tempo-mode	<f12> <f12> SPC</f12></f12>	( <b>pel-tempo-mode</b> & optional ARG)	Toggle PEL tempo mode on/off.  PEL tempo mode activates C-c . and C-c , as well as to C-c C and C-c C-, key		
			bindings to navigate across tempo mark hot-spots. When pel-tempo-mode is active the pel-tempo-mode lighter (‡) is shown on the status bar. The second set are only available when Emacs runs in		
			graphics mode.  When a skeleton is inserted via the execution of one of the pel-rst commands, the pel-tempo-		
			mode is automatically activated.		
Jump to next tempo mark	• C-c M-f • C-c .	(tempo-forward-mark)	Jump to the next mark in 'tempo-back-mark-list': the location where code must be updated inside the inserted skeleton.		
	• C-c C		These key key bindings are only available when pel-tempo-mode is active.		
Jump to previous tempo mark	• C-c M-b • C-c ,	(tempo-backward-mark)	Jump to the previous mark in 'tempo-back-mark-list': the location where code must be updated inside the inserted skeleton.		
	• C-c C-,		These key binding are only available when pel-tempo-mode is active.		
Tempo Template Tag Insertion	<f12> <f12> <f12></f12></f12></f12>	(tempo-complete-tag &optional SILENT)	Look for a tag and expand it.		
	Instead of using the <f12></f12>	,	you can type the template name (shown in the title column like "if", "case", etc) completely or partially		
	and then hit <f12> <f12></f12></f12>	<f12>. A completion buffer of</f12>	pens up if the template name is incomplete (or empty in which case the buffer lists all available		
	All the tags in the tag lists in		acs expands the template.  les 'tempo-tags') are searched for a match for the text before the point. The way the string to match natch-finder'. If 'tempo-match-finder' returns nil, then the results are the same as no match at all.		
	If a single match is found, the	ne corresponding template is ex	completion is found and 'tempo-show-completion-buffer' is non-nil, a buffer containing possible		
	completions is displayed.	ori wili give a signal. Il a partial	completion is found and tempo-show-completion-buller is non-fill, a buller containing possible		
Inserting code	Extra text insertion can be dor	ne with the following command	s. See also above: <f12> M-e activates electric pair: typing (inserts the matching)</f12>		
Insert Parentheses pair ()	М- (	(insert-parentheses &optional ARG)	For C: insert a parenthesis pair '()', leaving point after open-paren.  • A positive ARG encloses the following ARG sexps in parenthesis if they are balanced.		
		aoptional Al (d)	A positive ARG encloses the following ARG sexps in parentnesis if they are balanced.     A negative ARG encloses the preceding ARG sexps instead.		
		o zero: just insert '()' and leave -spaces buffer local and set it	point between. to nil in C mode buffers, allowing the use of this command to insert the argument parentheses		
	following a function (and wit		the function name and the opening parenthesis.		
	20 2 2 200.0, 110012 0110	5	and a same of control of the same of control		

Description	Keystroke	Function	<u>Note</u>	
Marking	Emacs provides the following	command to quickly mark the	whole content of the current function. More mark commands exists, see the <u>S Marking</u> table.	
Mark the complete function body	C-M-h	(c-mark-function)	Mark complete function.	
See also: <u>I Marking</u>	<ul> <li>Put mark at end of the current top-level declaration or macro, point at beginning.</li> <li>If point is not inside any then the closest following one is chosen. Each successive call of this command extends the marked region by one function.</li> <li>A mark is left where the command started, unless the region is already active (in Transient Mark mode).</li> <li>As opposed to C-M-a and C-M-e, this function does not require the declaration to contain a brace block.</li> </ul>			
Getting Syntactic Information	Use the following commands	to extract syntactic information	from the source code.	
Display name of current function	• C-c C-z • <f12> f • M-<f12> f</f12></f12>	(c-display-defun-name &optional ARG)	Display the name of the current CC mode defun and the position in it.  • With a prefix arg, push the name onto the kill ring too.	
Search Support			case is often used. Using superword-mode helps searching. o change this use the <f11> t <f2> to access the customize buffer.</f2></f11>	
Toggle superword-mode See also:  • ∑ Text Modes  • ∑ Search/Replace	• <f11> t m p • <f12> M-p</f12></f11>	(superword-mode &optional ARG)	Toggle superword-mode: a minor mode that treats <a href="mailto:snake_case">snake_case</a> as one word. In C '_' are treated as part of words.  • With a prefix argument ARG, enable superword mode if ARG is positive, and disable it otherwise.	
Highlighting blocks	show-paren-mode, which h	nighlights the parens that match	seful modes to highlight blocks of (), {}, and [].  the one before or after point.  the highlighted with the same colour.	
Toggle show-paren mode on/off	• <f12> M-9 • M-<f12> M-9</f12></f12>	(show-paren-mode &optional ARG)	Toggle visualization of matching parens (Show Paren mode).  • With prefix argument ARG, enable Show Paren mode if ARG is positive, disable it otherwise.  • Show Paren mode is a global minor mode. When enabled, any matching parenthesis is	
See also: <u>E Highlight</u>	• <f11> h (</f11>		highlighted in 'show-paren-style' after 'show-paren-delay' seconds of Emacs idle time.	
Enable/Disable coloured highlight of nested blocks (),{},[]	• <f12> M-r • M-<f12> M-r</f12></f12>	(rainbow-delimiters-mode &optional ARG)	Highlight nested parentheses, brackets, and braces with colours according to their depth.  • Customize the depth and colours with M-x customize-group rainbow-delimiters  Requires: rainbow-delimiters.el	
See also: <u>∑ Highlight</u>	• <f11> h R</f11>		PEL activates this when the <b>pel-use-rainbow-delimiters</b> user option is set to <b>t</b> .	
Navigation in C	This current list below describ	e the specialized commands o	nly. See the others inside <u>Σ <b>Navigation</b></u>	
By definitions	Move to the definition of func	tion or type at point. See <b>Xre</b>	for more information to activate the various engines that support cross referencing for C code.	
Find definition of identifier at point	м	(xref-find-definitions IDENTIFIER)	Grab symbol at point and move cursor to its definition.  If there are more than one match, prompt in the "xref" buffer.  To search for a symbol entered manually, type C-u M	
See also: <u>∑ Xref</u>			With dumb-jump this performs a search using ag, ripgrep or git grep if available.	
Go back to where M was last issued	М-,	(xref-pop-marker-stack)	<ul> <li>Pop back to where M was last invoked.</li> <li>Marker depth is controlled by the xref-marker-ring-length user option.</li> </ul>	
By call graph	Use the call-graph external pa	ackage to build a call-graph of a	a C function. Uses either GNU Global or Git grep as backend.	
Build call-graph of function at point/region	<f12> M-g</f12>	(call-graph &optional FUNC)	Generate 'call-graph' for FUNC / func-at-point / func-in-active-region.  With prefix argument, discard cached data and re-generate reference data.  Preliminary support: validity of the generated graph needs to be investigated.  Requires external call-graph package, activated by PEL when pel-use-call-graph is t.	
By statements	Move to beginning /end of sta	ntement or comment	nequires external <u>can-graph</u> package, was activated by FLL when per-use-can-graph is t.	
Go to beginning of	M-a	(c-beginning-of-statement	Go to the beginning of the innermost statement.	
statement (backward)		&optional COUNT LIM SENTENCE-FLAG)	<ul> <li>With prefix arg, go back N - 1 statements.</li> <li>If already at the beginning of a statement then go to the beginning of the closest preceding one, moving into nested blocks if necessary (use C-M-b to skip over a block).</li> <li>If within or next to a comment or multiline string, move by sentences instead of statements.</li> </ul>	
Go to the end of statement (forward)	м-е	(c-end-of-statement &optional COUNT LIM SENTENCE-FLAG)	Go to the end of the innermost statement.  • With prefix arg, go forward N - 1 statements.  • Move forward to the end of the next statement if already at end, and move into nested blocks (use C-M-f to skip over a block).  • If within or next to a comment or multiline string, move by sentences instead of statements.	
Go to start of current switch statement	<f6> t w s</f6>	(pel-cc-to-switch-begin)	Move point to the start { of current switch statement, if any.  • If point is inside switch statement, mark position before moving point. Move it back with M-\cdot\cdot.  • If point is not inside a switch statement, issue a user error.	
Go to end of current switch statement	<f6> t w e</f6>	(pel-cc-to-switch-end)	Move point just past the end } of current switch statement, if any • If point is inside switch statement, mark position before moving point. Move it back with <b>M</b> -\`. • If point is not inside a switch statement, issue a user error.	
Go to start of enum definition block	<f6> t e s</f6>	(pel-cc-to-enum-begin)	Move point to the start { of current enum definition block, if any.  If point is inside such a block, mark position before moving point. Move it back with M-`.  If point is not inside an appropriate block statement, issue a user error.	
Go to end of enum definition block	<f6> t e e</f6>	(pel-cc-to-enum-end)	Move point just past the end } of current enum definition block, if any.  • If point is inside such a block, mark position before moving point. Move it back with M-\[^\circ\].  • If point is not inside an appropriate block statement, issue a user error.	
Go to start of struct definition block	<f6> t s s</f6>	(pel-cc-to-struct-begin)	Move point to the start { of current struct definition block, if any.  • If point is inside such a block, mark position before moving point. Move it back with M-`.  • If point is not inside an appropriate block statement, issue a user error.	
Go to end of struct definition block	<f6> t s e</f6>	(pel-cc-to-struct-end)	Move point just past the end } of current structdefinition block, if any.  • If point is inside such a block, mark position before moving point. Move it back with M-\[\cdot\).  • If point is not inside an appropriate block statement, issue a user error.	
Go to start of union definition block	<f6> t u s</f6>	(pel-cc-to-union-begin)	Move point to the start { of current union definition block, if any.  • If point is inside such a block, mark position before moving point. Move it back with M-\`.  • If point is not inside an appropriate block statement, issue a user error.	
Go to end of union definition block	<f6> t u e</f6>	(pel-cc-to-union-end)	Move point just past the end } of current union definition block, if any.  • If point is inside such a block, mark position before moving point. Move it back with M-`.  • If point is not inside an appropriate block statement, issue a user error.	

<u>Description</u>	<u>Keystroke</u>	Function	<u>Note</u>
By blocks     functions     structures	Move across C++ statemen     When point is located be	efore opening brace or right aft	ny group of (), [], {} or <> blocks. Jump over comments.  er closing brace and <b>show-paren-mode</b> is on, the matching parentheses are highlighted.
Move block forward See also:  ■ Navigation  ■ Use this to move to end of next syntax element or to end of block when already outside the block.  ■ Use C-M-u to exit a block (see below).	• <f12> <right> • <f12> <right> • <m-f12> <right> • C-M-f • C-M-<right> • C-[ C-f • Esc C-f • Esc C-<right></right></right></right></m-f12></right></f12></right></f12>	(forward-sexp &optional ARG)	Move forward across one balanced expression (sexp).  • With ARG, do it that many times. Negative arg -N means move backward across N balanced expressions. This command assumes point is not in a string or comment.  • C-M-f : ▼ Shift marking is available in graphics mode, not in terminal mode.  • C-M- <right> : ▼ Shift marking works with this command.  • With PEL: if you want to use Esc C-<right> binding you must ensure that pel-windmove-on-esc-cursor user option is set to nil, otherwise it does something else.  • C-M-<right> does not work on Windows, but H-<right> does.  • Several Linux distros map C-M-<right> to desktop workspace operation. In that case you can either use another key binding or change Linux key binding in Systems-&gt;settings-&gt;keyboard-&gt;shortcuts to prevent it from using that key sequence.</right></right></right></right></right>
Forward block/list See also: Navigation	C-M-n	(forward-list &optional ARG)	Move forward across one balanced group of parentheses.  This command will also work on other parentheses-like expressions defined by the current language mode.  With ARG, do it that many times.  Negative arg -N means move backward across N groups of parentheses.  This command assumes point is not in a string or comment.  C-M-n : Shift marking is available in graphics mode, not in terminal mode.
Move block backward  See also:  ■ Navigation	• <f12> <left> • <m-f12> <left> • C-M-b • C-M-<left> • C-[ C-b • Esc C-b • Esc C-<left></left></left></left></m-f12></left></f12>	(backward-sexp &optional ARG)	Move backward across one balanced expression (sexp).  • With ARG, do it that many times. Negative arg -N means move forward across N balanced expressions. This command assumes point is not in a string or comment.  • C-M-b : ► Shift marking is available in graphics mode, not in terminal mode.  • C-M-<1eft>: ► Shift marking works with this command.  • ⚠ With PEL: if you want to use Esc C-<1eft> binding you must ensure that pel-windmove-on-esc-cursor user option is set to nil, otherwise it does something else.  • C-M-<1eft> does not work on Windows, but H-<1eft> works.  ⑤ Several Linux distros map C-M-<1eft> to desktop workspace operation. In that case you can either use another key binding or change Linux key binding in Systems->settings->keyboard->shortcuts to prevent it from using that key sequence.
Backward block/list See also: Navigation	С-М-р	(backward-list &optional ARG)	Move backward across one balanced group of parentheses.  This command will also work on other parentheses-like expressions defined by the current language mode.  With ARG, do it that many times.  Negative arg -N means move forward across N groups of parentheses.  This command assumes point is not in a string or comment.  C-M-p: Shift marking is available in graphics mode, not in terminal mode.
Backward to beginning of current top-level function or struct	С-м-а	( <b>c-beginning-of-defun</b> &optional ARG)	Move backward to the beginning of a function or type definition.  With a positive argument, move backward that many functions or structures. A negative argument  N means move forward to the Nth following beginning.
	• <f12> <up> • <m-f12> <up> C-M-<home></home></up></m-f12></up></f12>	(beginning-of-defun &optional ARG)	Move backward to the beginning of function or type definition.  • Move point before the function type or the struct or typedef keyword.  • With ARG, do it that many times. Negative ARG means move forward to the ARGth following beginning of defun.  • Shift marking is available. With <f6> and <f12> hit Shift after function key, before cursor key.  • This command moves to the beginning go the next function or of the same nesting level of the current location. It skips the functions that are more deeply nested.</f12></f6>
Forward to end of current top-level function or struct.	С-М-е	( <b>c-end-of-defun</b> &optional ARG)	Move forward to the end of a top level declaration.  • With argument, do it that many times. Negative argument -N means move back to Nth preceding end.
	• <f12> <down> • <m-f12> <down> C-M-<end></end></down></m-f12></down></f12>	( <b>end-of-defun</b> &optional ARG)	Move forward to the end of next function or type definition.  With argument, do it that many times. Negative argument -N means move back to Nth preceding end of defun.  F Shift marking is available. With <f6> and <f12> hit Shift after function key, before cursor key.  This command moves to the end of the next top-level function. It skips nested functions.</f12></f6>
Backward to end of previous top level function or struct	• <f12> <m-up> • <m-f12> <m-up></m-up></m-f12></m-up></f12>	(pel-end-of-previous-defun &optional SILENT DONT- PUSH_MARK)	Move backwards to the end of the previous function or type definition.  • Beeps if does not find end of previous function unless SILENT is non-nil.  • If the end of previous function is found, push the start location to the mark ring unless DONT-PUSH_MARK is non-nil.  • Move back to previous position with M−ˆ or <f6><f6>.  ▼ Shift marking is available. With <f6> and <f12> hit Shift after function key, before cursor key.  In some cases it fails to detect the end of the previous block and fails.</f12></f6></f6></f6>
Forward to start of next top level function or struct  Use this to move from the top of the file to the first block.	• <f12> <m-down> • <m-f12> <m-down></m-down></m-f12></m-down></f12>	(pel-beginning-of-next-defun &optional SILENT DONT-PUSH_MARK)	Move forward to the beginning of the next function or type definition.  • Move point before the function type or the struct or typedef keyword.  • Beeps if does not find beginning of next function unless SILENT is non-nil.  • If the beginning of next function is found, push the start location to the mark ring unless DONT-PUSH_MARK is non-nil.  • Move back to previous position with M→ or <f6><f6>.  ▼ Shift marking is available. With <f6> and <f12> hit Shift after function key, before cursor key.  • This command complements what end-of-defun does.  • It moves forward but not to the end of the function definition (like end-of-defun) but to the beginning of the function definition, which is often what users of other editors expect.</f12></f6></f6></f6>
in/out of blocks  Backward <u>Up/outside</u>	Move in or out of C scope b     C-M-u	locks, or any group of (), [], {} (backward-up-list	or < > blocks.  Move backward out of one level of parentheses or nested blocks.
sexp hierarchy See also:  Navigation	• C-M- <up> • C-M-<up> • C-[ C-u • Esc C-u • Esc C-<up></up></up></up>	&optional ARG ESCAPE- STRINGS NO-SYNTAX- CROSSING)	<ul> <li>This command will also work on other parentheses-like expressions defined by the current language mode. With ARG, do this that many times. A negative argument means move forward but still to a less deep spot.</li> <li>⚠ With PEL: if you want to use Esc C-<up> binding you must ensure that pel-windmove-on-esc-cursor user option is set to nil.</up></li> <li>C-M-u : Shift marking is available in graphics mode, not in terminal mode.</li> <li>C-M-<up> : Shift marking works with this command.</up></li> <li>C-M-<up> does not work on Windows, but H-<up> does.</up></up></li> </ul>
Forward Up/outside sexp hierarchy See also:   Navigation	C-M-]	( <b>up-list</b> & optional ARG ESCAPE-STRINGS NO- SYNTAX-CROSSING)	Move forward out of one level of parentheses or nested blocks.  • Also work on other parentheses-like expressions defined by the current language mode.  • With ARG, do it that many times. Negative arg means move backward but to a less deep spot.
Down/inside sexp/block See also: • ∑ Navigation	• C-M-d • C-M- <down> • C-[ C-d • Esc C-d • Esc C-<down></down></down>	(down-list &optional ARG)	Move forward down one level of parentheses.  • Also work on other parentheses-like expressions defined by the current language mode.  • With ARG, do it that many times. Negative arg mans move backward but still go down a level.  • This command assumes point is not in a string or comment.  • With PEL: if you want to use Esc C- <down> binding you must ensure that pel-windmove-on-esc-cursor user option is set to nil.  • C-M-d : ▼ Shift marking is available in graphics mode, not in terminal mode.  • C-M-<down> : ▼ Shift marking works with this command.  • C-M-<down> does not work on Windows, but H-<down> does.</down></down></down></down>

<u>Description</u>	<u>Keystroke</u>	Function	<u>Note</u>
By C pre-processor	Move across C preprocesso	r conditional inclusion statem	ents_#if #ifdef #ifndef   #else #elif   #endif _!_Does not yet support C++23 #elifdef and #elifndef
Move point forward to matching #endif or matching #else   #elif	<f6> <right></right></f6>	(pel-c-preproc-forward- conditional &optional TO- ELSE)	Move point forward to matching #endif  If point on a #if #ifdef #ifndef statement moves to the matching endif  With C-u or numerical arg: move forward to matching #else #elif  On success, push the original position on the mark ring and return the new position.  On error, issue user error on mismatch. Shift marking is available with C-M- <ri> Tight&gt;</ri>
Move point backward to matching #if  #ifdef   #ifndef or matching #else   #elif	<f6> <left></left></f6>	(pel-c-preproc-backward-conditional &optional TO-ELSE)	Move point backward to matching beginning of #if #ifdef #ifndef conditional.  • With C-u or numerical arg: move backward to matching #else #elif  • On success, push the original position on the mark ring and return the new position.  • On error, issue user error on mismatch. Shift marking is available with C-M- <left></left>
Move outward forward to matching #endif	<f6> <down></down></f6>	(pel-c-preproc-outward- forward-conditional &optional NEST-COUNT)	Move point forward, outward to end of current #if #ifdef #ifndef statement.  By default move 1 nest level outward. A larger count can be specified with optional NEST-COUN numeric argument.  On success, push the original position on the mark ring and return the new position.  On error, issue user error on mismatch.
Move outward backward to matching #if   #ifdef   #ifndef	<f6> <up></up></f6>	(pel-c-prepcroc-outward-backward-conditional &optional NEST-COUNT)	Move point backward, outward to beginning of current #if #ifdef #ifndef statement.  By default move 1 nest level outward. A larger count can be specified with optional NEST-COUNT numeric argument.  On success, push the original position on the mark ring and return the new position. On error, issue user error on mismatch.
Show all C pre-processor conditional statements inside an occur buffer	<f6> o</f6>	(pel-c-preproc- conditionals-occur &optional NLINES)	Show C pre-processor conditional statements inside an occur buffer.  • Each line is shown with NLINES before and after, or -NLINES before if NLINES is negative.  • NLINES defaults to <b>list-matching-lines-default-context-lines</b> user-option value.  • If a region is defined the search is restricted to the region.  See <u>occur search</u> .
Show all C pre-processor conditional statements of project buffers inside an occur buffer	<f6> <f8> o</f8></f6>	(pel-c-preproc- conditionals-multi-occur &optional NLINES)	Show C pre-processor conditional statements of current project buffers inside an occur buffer.  • Each line is shown with NLINES before and after, or -NLINES before if NLINES is negative.  • NLINES defaults to list-matching-lines-default-context-lines user-option value.  See occur search.  • This command uses Projectile. You must have pel-use-projectile user-option set and projectile active (use <f11> <f8> <f8> to activate it.)</f8></f8></f11>
C Preprocessor	not be executed with the Hid are listed below. They can be PEL provides a key <a "#else"="" "#if",="" #elif"="" at="" backward="" but="" by="" followed="" forward.<="" function="" going="" href="https://www.nydra.com/hydra.c&lt;/td&gt;&lt;td&gt;e-ifdef mode. There are also ex&lt;br&gt;e used for editing C and C++ so&lt;br&gt;help navigate trough preprocess&lt;br&gt;start with &lt;f12&gt; &lt;f7&gt; open to&lt;/td&gt;&lt;td&gt;nal statements, allow expansion of preprocessor macros, hiding pre-processor statements that would ternal packages that provide extra support. All commands provided by Emacs and external packages&lt;/td&gt;&lt;/tr&gt;&lt;tr&gt;&lt;td&gt;Open the C preprocessor hydra with &lt;f12&gt; &lt;f7&gt; followed by on of the hydra keys:&lt;/td&gt;&lt;td&gt;n: next p: prev C-p: begin C-n: end C-u: up&lt;/td&gt;&lt;td&gt;Hide #  #: toggle mode   e W: toggle shadow   d R: toggle RO   u H: hide   U S: show   D&lt;/td&gt;&lt;td&gt;(C/*la Ifdef WK Fly &lt;sup&gt;2&lt;/sup&gt; Anzu Abbrev)  Vars   Other  &lt;/td&gt;&lt;/tr&gt;&lt;tr&gt;&lt;td&gt;Navigate across pre-&lt;br&gt;processor conditionals&lt;/td&gt;&lt;td&gt;&lt;/td&gt;&lt;td&gt;&lt;/td&gt;&lt;td&gt;telif and #endif C pre-processor conditional statements.  bre-processor command used by Emacs. The default depends on the operating system.&lt;/td&gt;&lt;/tr&gt;&lt;tr&gt;&lt;td&gt;Move to previous preprocessor directive&lt;/td&gt;&lt;td&gt;• &lt;f12&gt; # p&lt;br&gt;* &lt;f12&gt; &lt;f7&gt; p&lt;/td&gt;&lt;td&gt;(pel-pp-prev-directive)&lt;/td&gt;&lt;td&gt;Move point to previous preprocessor directive.&lt;/td&gt;&lt;/tr&gt;&lt;tr&gt;&lt;td&gt;Move to next preprocessor directive&lt;/td&gt;&lt;td&gt;• &lt;f12&gt; # n&lt;br&gt;* &lt;f12&gt; &lt;f7&gt; n&lt;/td&gt;&lt;td&gt;(pel-pp-next-directive)&lt;/td&gt;&lt;td&gt;Move point to next preprocessor directive.&lt;/td&gt;&lt;/tr&gt;&lt;tr&gt;&lt;td&gt;Move up in the pre-&lt;br&gt;processor conditional&lt;br&gt;block&lt;/td&gt;&lt;td&gt;• C-c C-u&lt;br&gt;* &lt;f12&gt; &lt;f7&gt; C-u&lt;/td&gt;&lt;td&gt;(c-up-conditional COUNT)&lt;/td&gt;&lt;td&gt;Move back to the containing preprocessor conditional, leaving mark behind.  A prefix argument acts as a repeat count. With a negative argument, move forward to the end of the containing preprocessor conditional.  " is="" like="" not="" so="" stops="" td="" the="" them="" treated="" when=""></a>		
Move to the previous pre- processor conditional block	• C-c C-p * <f12> <f7> C-p</f7></f12>	(c-backward-conditional COUNT &optional TARGET- DEPTH WITH-ELSE)	Move back across a preprocessor conditional, leaving mark behind.  A prefix argument acts as a repeat count.  With a negative argument, move forward across a preprocessor conditional.
Move to the next pre- processor conditional block	C-c C-n * <f12> <f7> C-n</f7></f12>	(c-forward-conditional COUNT &optional TARGET- DEPTH WITH-ELSE)	Move forward across a preprocessor conditional, leaving mark behind.  A prefix argument acts as a repeat count.  With a negative argument, move backward across a preprocessor conditional.  If there aren't enough conditionals after (or before) point, an error is signaled.  "#elif" is treated like "#else" followed by "#if", except that the nesting level isn't changed when tracking subconditionals.
Expand Pre-Processor macro	• C-c C-e • <f12> # #</f12>	(c-macro-expand START END SUBST)	Expand C macros in the region, using the C preprocessor, in the *Macroexpansion* buffer.  • With prefix arg (like C-u) replace the region with the expansion instead of showing it.  • If the user option 'c-macro-prompt-flag' is non-nil prompt for arguments to the preprocessor (e.g. '-DDEBUG -I ./include'), otherwise use 'c-macro-cppflags'.
	C-c C-\	(c-backslash-region FROM	Insert, align, or delete end-of-line backslashes on the lines in the region.  • With no argument, inserts backslashes and aligns existing backslashes.
Insert/align or delete end-of-line backslash	This function does not mover (if any) at the end of the preserved you can put the region around the r	evious line is deleted. und an entire macro definition a	With an argument, deletes the backslashes.  region. If the region ends at the start of a line and the macro doesn't continue below it, the backslashed use this command to conveniently insert and align the necessary backslashes.  ling to: 'c-backslash-column', 'c-backslash-max-column' and 'c-auto-align-backslashes'.

Hide ifdet PA - J.	<u>Keystroke</u>	Function	Note		
<u>Hide-ifdef Mode</u>			of code that the C preprocessor wouldn't pass through.		
hide/show code	<ul> <li>It supports complete C/C+-</li> <li>It scans for new #define syr</li> </ul>				
controlled by C-			expanded file according to the state of pre-processor symbols that are maintained inside the Hide-		
preprocessor	ifdef environment: the <b>hide-ifdef-env</b> association list Emacs variable (use <b><f1></f1></b> v to see the content of Emacs variables). See ∑ Help/Info.				
	When hiding code, the hidden code is marked by ellipses ().				
			den text is still in the buffer, and you can move the point into it and modify text unawares.  ef-hiding by setting hide-ifdef-read-only user-option to a non-nil value.		
		def customization group with <			
		• •	read-only (with C-c @ C-q) or with <f12> # r or <f12> <f7> R.</f7></f12></f12>		
	With PEL, the commands	are reachable via the <f12> pr</f12>	refix keys can also be reached via the M- <f12> and the <f11> SPC c prefix keys.</f11></f12>		
	* The key sequences that	t start with <f12> <f7> oper</f7></f12>	n the pel-∑c-preproc <u>Hydra</u> allowing further hydra keys to be typed without any prefix.		
	Requires the hydra	a external package 🛂 PEL acti	ivates when the <b>pel-use-hydra</b> user option is set to <b>t</b> .		
	-	tion variables affect how the hi			
Use <f12> # ?</f12>	<ul><li>to change, execute: M</li><li>'hide-ifdef-env'</li></ul>	-x customize-group hi	de-ifdef or type <f12> # <f3></f3></f12>		
to show the value of hide- ifdef-env and hide-ifdef-		defined symbols for the current	project. The list holds the following forms:		
define-alist		when the SYMBOL is defined (			
	'hide-ifdef-define-alist'	when the symbol is defined wi	tri ari explicit value.		
			ide-ifdef-set-define-alist' to save the current 'hide-ifdef-env' and 'hide-ifdef-use-define-alist' to set t		
	current 'hide-ifdef-env  'hide-ifdef-lines'	' from one of the lists in 'hide-i	fdef-define-alist'.		
		how #if, #ifdef, #ifndef, #else, a	nd #endif lines when hiding.		
	<ul> <li>'hide-ifdef-initially'</li> </ul>				
		e-ifdefs' should be called wher	n Hide-Ifdef mode is activated.		
	<ul> <li>'hide-ifdef-read-only'         Set to non-nil if you w     </li> </ul>	ant to make buffers read only v	vhile hiding.		
		ad-only status is restored to pre			
Toggle the Hide-Ifdef	• <f12> M-#</f12>	(hide-ifdef-mode &optional	Toggle features to hide/show #ifdef blocks (Hide-Ifdef mode).		
mode :	• M- <f12> M-#</f12>	ARG)	With a prefix argument, enable Hide-Ifdef mode if ARG is positive, and disable it otherwise.		
<ul> <li>hide/show code suppressed by C</li> </ul>	* <f12> <f7> #</f7></f12>		Hide-Ifdef mode is a buffer-local minor mode for use with C and C-like major modes.  When enabled, code within #ifdef constructs that the C preprocessor would eliminate may be		
preprocessor	• <f11> SPC c M-#</f11>		hidden from view.		
Toggle read-only mode	• C-c @ C-q	(hide-ifdef-toggle-read-	Toggle read-only: toggle 'hide-ifdef-read-only'.		
when text is hidden	• <f12> # r</f12>	only)	Note that you can make the file read only by default when hide-ifdef is hiding text, by setting the		
	* <f12> <f7> R</f7></f12>		'hide-ifdef-read-only' user option to t.		
Toggle shadowing of	• C-c @ C-w	(hide-ifdef-toggle-	Toggle shadowing.		
hidden text.	• <f12> # w</f12>	shadowing)	When shadowing is on, text that would be hidden is "shadowed" instead: it is displayed with the		
	* <f12> <f7> W</f7></f12>		shadow face (normally something dim, all depending of the theme used).		
Hide code suppressed by	• C-c @ h	(hide-ifdefs &optional	Hide the contents of some #ifdefs.		
C preprocessor	• <f12> # H</f12>	NOMSG)	<ul> <li>Assume that defined symbols have been added to 'hide-ifdef-env'.</li> </ul>		
	• M- <f12> # H</f12>		<ul> <li>The text hidden is the text that would not be included by the C preprocessor if it were given th file with those symbols defined.</li> </ul>		
-	* <f12> <f7> H</f7></f12>		With prefix command presents it will also hide the #ifdefs themselves.		
	• <f11> SPC c # H</f11>		Turn off hiding by calling 'show-ifdefs'.		
Restore all hidden into	• C-c @ s	(show-ifdefs)	Cancel the effects of 'hide-ifdef': show the contents of all #ifdefs.		
view	• <f12> # S</f12>	(,			
	* <f12> <f7> S</f7></f12>				
Hide part of current block	• C-c @ C-d	(hide-ifdef-block &optional	Hide the ifdef block (true or false part) enclosing or before the cursor.		
that would not be	• <f12> # h</f12>	ARG START END)	With optional prefix argument ARG, also hide the #ifdefs themselves.		
included	* <f12> <f7> h</f7></f12>				
Show all parts of the	• C-c @ C-s		Show the ifdef block (true or false part) enclosing or before the cursor.		
current #ifdef block	• <f12> # s</f12>	START END)			
	* <f12> <f7> s</f7></f12>				
Set a variable to a	• C-c @ d	(hide-ifdef-define VAR	Define a VAR to VAL (default 1) in 'hide-ifdef-env'.		
specific value	• <f12> # d</f12>	&optional VAL)	This allows hiding the block inside <b>#ifndef VAR</b> (or the equivalent) by executing the command hide ifdefer.		
	* <f12> <f7> d</f7></f12>		hide-ifdefs.		
Undefine a variable	• C-c @ u	(hide-ifdef-undef START	Undefine a VAR		
	• <f12> # u</f12>	END)	This allows hiding the blocks inside <b>#ifdef VAR</b> (or the equivalent) by executing the command hide-ifdefs.		
	* <f12> <f7> u</f7></f12>		hide-ifdefs.		
			0 11 11 11 11 11 11 11 11 11 11 11 11 11		
Save the symbol	• C-c @ D	(hide-ifdef-set-define-alist	Save the state of the current <b>hide-ifdev-env</b> to a list with the specified NAME for later re-use.		
environment list into a	• <f12> # D</f12>	(hide-ifdef-set-define-alist NAME)	The value is saved inside the <b>hide-ifdef-define-alist</b> variable.		
•		,	The value is saved inside the <b>hide-ifdef-define-alist</b> variable.  1 The list is not saved to disk. You may want to pre-create the value for a given project and store		
environment list into a named list	• <f12> # D</f12>	NAME)	The value is saved inside the <b>hide-ifdef-define-alist</b> variable.  1. The list is not saved to disk. You may want to pre-create the value for a given project and store inside your local directory variables for example.		
environment list into a named list  Use a named symbol	• <f12> # D * <f12> <f7> D</f7></f12></f12>	NAME) (hide-ifdef-use-define-alist	The value is saved inside the hide-ifdef-define-alist variable.  The list is not saved to disk. You may want to pre-create the value for a given project and store inside your local directory variables for example.  Set 'hide-ifdef-env' to the already saved symbol list with the specified NAME.		
environment list into a named list	• <f12> # D * <f12> <f7> D  • C-c @ U • <f12> # U</f12></f7></f12></f12>	NAME)	The value is saved inside the <b>hide-ifdef-define-alist</b> variable.  1. The list is not saved to disk. You may want to pre-create the value for a given project and store inside your local directory variables for example.		
environment list into a named list  Use a named symbol environment list	• <f12> # D * <f12> <f7> D • C-c @ U • <f12> # U * <f12> <f7> U</f7></f12></f12></f7></f12></f12>	NAME) (hide-ifdef-use-define-alist NAME)	The value is saved inside the hide-ifdef-define-alist variable.  The list is not saved to disk. You may want to pre-create the value for a given project and store inside your local directory variables for example.  Set 'hide-ifdef-env' to the already saved symbol list with the specified NAME.  Takes the value from the hide-ifdef-define-alist.		
environment list into a named list  Use a named symbol environment list  Clear the complete list of	• <f12> # D * <f12> <f7> D • C-c @ U • <f12> # U * <f12> * U • C-c @ C</f12></f12></f7></f12></f12>	NAME) (hide-ifdef-use-define-alist	The value is saved inside the hide-ifdef-define-alist variable.  The list is not saved to disk. You may want to pre-create the value for a given project and store inside your local directory variables for example.  Set 'hide-ifdef-env' to the already saved symbol list with the specified NAME.  Takes the value from the hide-ifdef-define-alist.  Clears all symbols defined in 'hide-ifdef-env'.		
environment list into a named list  Use a named symbol environment list	• <f12> # D * <f12> <f7> D • C-c @ U • <f12> # U * <f12> = T • C-c @ C • <f12> # C</f12></f12></f12></f7></f12></f12>	NAME) (hide-ifdef-use-define-alist NAME)	The value is saved inside the hide-ifdef-define-alist variable.  The list is not saved to disk. You may want to pre-create the value for a given project and store inside your local directory variables for example.  Set 'hide-ifdef-env' to the already saved symbol list with the specified NAME.  Takes the value from the hide-ifdef-define-alist.		
environment list into a named list  Use a named symbol environment list  Clear the complete list of #define'd symbols inside 'hide-ifdef-env'	• <f12> # D  * <f12> <f7> D  • C-c @ U • <f12> # U * <f12> # T  • C-c @ C • <f12> # C * <f12> # C * <f12> <f7> C</f7></f12></f12></f12></f12></f12></f7></f12></f12>	NAME)  (hide-ifdef-use-define-alist NAME)  (hif-clear-all-ifdef-defined)	The value is saved inside the hide-ifdef-define-alist variable.  The list is not saved to disk. You may want to pre-create the value for a given project and store inside your local directory variables for example.  Set 'hide-ifdef-env' to the already saved symbol list with the specified NAME.  Takes the value from the hide-ifdef-define-alist.  Clears all symbols defined in 'hide-ifdef-env'.  It first backup this variable to 'hide-ifdef-env-backup' before clearing to prevent accidental clearance.		
environment list into a named list  Use a named symbol environment list  Clear the complete list of #define'd symbols inside 'hide-ifdef-env'  Evaluate pre-processor	<pre>     <f12> # D</f12></pre>	NAME)  (hide-ifdef-use-define-alist NAME)  (hif-clear-all-ifdef-defined)	The value is saved inside the hide-ifdef-define-alist variable.  The list is not saved to disk. You may want to pre-create the value for a given project and store inside your local directory variables for example.  Set 'hide-ifdef-env' to the already saved symbol list with the specified NAME.  Takes the value from the hide-ifdef-define-alist.  Clears all symbols defined in 'hide-ifdef-env'.  It first backup this variable to 'hide-ifdef-env-backup' before clearing to prevent accidental clearance.  Evaluate the macro expansion result for the active region.		
environment list into a named list  Use a named symbol environment list  Clear the complete list of #define'd symbols inside 'hide-ifdef-env'	<pre>     <f12> # D</f12></pre>	NAME)  (hide-ifdef-use-define-alist NAME)  (hif-clear-all-ifdef-defined)	The value is saved inside the hide-ifdef-define-alist variable.  The list is not saved to disk. You may want to pre-create the value for a given project and store inside your local directory variables for example.  Set 'hide-ifdef-env' to the already saved symbol list with the specified NAME.  Takes the value from the hide-ifdef-define-alist.  Clears all symbols defined in 'hide-ifdef-env'.  It first backup this variable to 'hide-ifdef-env-backup' before clearing to prevent accidental clearance.  Evaluate the macro expansion result for the active region.  If no region active, find the current #ifdefs and evaluate the result.		
environment list into a named list  Use a named symbol environment list  Clear the complete list of #define'd symbols inside 'hide-ifdef-env'  Evaluate pre-processor macro	• <f12> # D  * <f12>   T  • C-c @ U  • <f12> # U  * <f12>   T  • C-c @ C  • <f12> # C  * <f12>   C  • C-c @ C  • <f12>   C  * <f12>   C  • C-c @ C  • <f12>   C  • C-c @ C</f12></f12></f12></f12></f12></f12></f12></f12></f12></f12></f12></f12></f12></f12></f12></f12></f12></f12></f12></f12></f12></f12></f12>	(hide-ifdef-use-define-alist NAME)  (hif-clear-all-ifdef-defined)  (hif-evaluate-macro RSTART REND)	The value is saved inside the hide-ifdef-define-alist variable.  The list is not saved to disk. You may want to pre-create the value for a given project and store inside your local directory variables for example.  Set 'hide-ifdef-env' to the already saved symbol list with the specified NAME.  Takes the value from the hide-ifdef-define-alist.  Clears all symbols defined in 'hide-ifdef-env'.  It first backup this variable to 'hide-ifdef-env-backup' before clearing to prevent accidental clearance.  Evaluate the macro expansion result for the active region.  If no region active, find the current #ifdefs and evaluate the result.  Only supports math calculations; strings or argumented macros can not be expanded.		
environment list into a named list  Use a named symbol environment list  Clear the complete list of #define'd symbols inside 'hide-ifdef-env'  Evaluate pre-processor macro  Rendering markup	<pre>     <f12> # D          <f12> <f7> D           </f7></f12></f12></pre> <pre></pre>	(hide-ifdef-use-define-alist NAME)  (hif-clear-all-ifdef-defined)  (hif-evaluate-macro RSTART REND)	The value is saved inside the hide-ifdef-define-alist variable.  The list is not saved to disk. You may want to pre-create the value for a given project and store inside your local directory variables for example.  Set 'hide-ifdef-env' to the already saved symbol list with the specified NAME.  Takes the value from the hide-ifdef-define-alist.  Clears all symbols defined in 'hide-ifdef-env'.  It first backup this variable to 'hide-ifdef-env-backup' before clearing to prevent accidental clearance.  Evaluate the macro expansion result for the active region.  If no region active, find the current #ifdefs and evaluate the result.  Only supports math calculations; strings or argumented macros can not be expanded.		
environment list into a named list  Use a named symbol environment list  Clear the complete list of #define'd symbols inside 'hide-ifdef-env'  Evaluate pre-processor macro  Rendering markup embedded in	<pre>     <f12> # D          <f12> <f7> D           </f7></f12></f12></pre> <pre></pre>	(hide-ifdef-use-define-alist NAME)  (hif-clear-all-ifdef-defined)  (hif-evaluate-macro RSTART REND)	The value is saved inside the hide-ifdef-define-alist variable.  The list is not saved to disk. You may want to pre-create the value for a given project and store inside your local directory variables for example.  Set 'hide-ifdef-env' to the already saved symbol list with the specified NAME.  Takes the value from the hide-ifdef-define-alist.  Clears all symbols defined in 'hide-ifdef-env'.  It first backup this variable to 'hide-ifdef-env-backup' before clearing to prevent accidental clearance.  Evaluate the macro expansion result for the active region.  If no region active, find the current #ifdefs and evaluate the result.  Only supports math calculations; strings or argumented macros can not be expanded.		
environment list into a named list  Use a named symbol environment list  Clear the complete list of #define'd symbols inside 'hide-ifdef-env'  Evaluate pre-processor macro  Rendering markup	<pre>     <f12> # D          <f12> <f7> D           </f7></f12></f12></pre> <pre></pre>	(hide-ifdef-use-define-alist NAME)  (hif-clear-all-ifdef-defined)  (hif-evaluate-macro RSTART REND)  used to create images from species UML diagrams or finite-state in the second secon	The value is saved inside the hide-ifdef-define-alist variable.  The list is not saved to disk. You may want to pre-create the value for a given project and store inside your local directory variables for example.  Set 'hide-ifdef-env' to the already saved symbol list with the specified NAME.  Takes the value from the hide-ifdef-define-alist.  Clears all symbols defined in 'hide-ifdef-env'.  It first backup this variable to 'hide-ifdef-env-backup' before clearing to prevent accidental clearance.  Evaluate the macro expansion result for the active region.  If no region active, find the current #ifdefs and evaluate the result.  Only supports math calculations; strings or argumented macros can not be expanded.		
environment list into a named list  Use a named symbol environment list  Clear the complete list of #define'd symbols inside 'hide-ifdef-env'  Evaluate pre-processor macro  Rendering markup embedded in	<pre>     <f12> # D</f12></pre>	(hide-ifdef-use-define-alist NAME)  (hif-clear-all-ifdef-defined)  (hif-evaluate-macro RSTART REND)  used to create images from species UML diagrams or finite-state in the second secon	The value is saved inside the hide-ifdef-define-alist variable.  The list is not saved to disk. You may want to pre-create the value for a given project and store inside your local directory variables for example.  Set 'hide-ifdef-env' to the already saved symbol list with the specified NAME.  Takes the value from the hide-ifdef-define-alist.  Clears all symbols defined in 'hide-ifdef-env'.  It first backup this variable to 'hide-ifdef-env-backup' before clearing to prevent accidental clearance.  Evaluate the macro expansion result for the active region.  If no region active, find the current #ifdefs and evaluate the result.  Only supports math calculations; strings or argumented macros can not be expanded.		
environment list into a named list  Use a named symbol environment list  Clear the complete list of #define'd symbols inside 'hide-ifdef-env'  Evaluate pre-processor macro  Rendering markup embedded in comments  Preview UML diagram from plantUML source in	<ul> <li><f12> # D</f12></li> <li><f12> <f7> D</f7></f12></li> <li>C-c @ U</li> <li><f12> # U</f12></li> <li><f12> <f7> U</f7></f12></li> <li>C-c @ C</li> <li><f12> # C</f12></li> <li><f12> <f7> C</f7></f12></li> <li>C-c @ e</li> <li><f12> # e</f12></li> <li><f12> <f7> e</f7></f12></li> <li>The following commands are markup languages to describe</li> <li>You can also use Graphviz, see</li> </ul>	(hide-ifdef-use-define-alist NAME)  (hif-clear-all-ifdef-defined)  (hif-evaluate-macro RSTART REND)  used to create images from specially because the common of the common	The value is saved inside the hide-ifdef-define-alist variable.  The list is not saved to disk. You may want to pre-create the value for a given project and store inside your local directory variables for example.  Set 'hide-ifdef-env' to the already saved symbol list with the specified NAME.  Takes the value from the hide-ifdef-define-alist.  Clears all symbols defined in 'hide-ifdef-env'.  It first backup this variable to 'hide-ifdef-env-backup' before clearing to prevent accidental clearance.  Evaluate the macro expansion result for the active region.  If no region active, find the current #ifdefs and evaluate the result.  Only supports math calculations; strings or argumented macros can not be expanded.  scific markup code embedded inside C source code comments. This can be useful when using these machines for example.		
environment list into a named list  Use a named symbol environment list  Clear the complete list of #define'd symbols inside 'hide-ifdef-env'  Evaluate pre-processor macro  Rendering markup embedded in comments  Preview UML diagram from plantUML source in current plantUML region	<ul> <li><f12> # D</f12></li> <li><f12> <f7> D</f7></f12></li> </ul> • C-c @ U <ul> <li><f12> # U</f12></li> <li><f12> <f7> U</f7></f12></li> </ul> • C-c @ C <ul> <li><f12> # C</f12></li> <li><f12> <f7> C</f7></f12></li> </ul> • C-c @ e <ul> <li><f12> # e</f12></li> <li><f12> # e</f12></li> <li><f12> <f7> e</f7></f12></li> </ul> The following commands are markup languages to describe You can also use Graphviz, see	(hide-ifdef-use-define-alist NAME)  (hif-clear-all-ifdef-defined)  (hif-evaluate-macro RSTART REND)  used to create images from spectum of the company of th	The value is saved inside the hide-ifdef-define-alist variable.  The list is not saved to disk. You may want to pre-create the value for a given project and store inside your local directory variables for example.  Set 'hide-ifdef-env' to the already saved symbol list with the specified NAME.  Takes the value from the hide-ifdef-define-alist.  Clears all symbols defined in 'hide-ifdef-env'.  It first backup this variable to 'hide-ifdef-env-backup' before clearing to prevent accidental clearance.  Evaluate the macro expansion result for the active region.  If no region active, find the current #ifdefs and evaluate the result.  Only supports math calculations; strings or argumented macros can not be expanded.  Decific markup code embedded inside C source code comments. This can be useful when using these machines for example.  Render the PlantUML markup embedded in current mode comment.  Use region if identified otherwise use PlantUML block at point.  Uses prefix (as PREFIX) to choose where to display it:		
environment list into a named list  Use a named symbol environment list  Clear the complete list of #define'd symbols inside 'hide-ifdef-env'  Evaluate pre-processor macro  Rendering markup embedded in comments  Preview UML diagram from plantUML source in	<ul> <li><f12> # D</f12></li> <li><f12> <f7> D</f7></f12></li> </ul> • C-c @ U <ul> <li><f12> # U</f12></li> <li><f12> <f7> U</f7></f12></li> </ul> • C-c @ C <ul> <li><f12> # C</f12></li> <li><f12> <f7> C</f7></f12></li> </ul> • C-c @ e <ul> <li><f12> # e</f12></li> <li><f12> # e</f12></li> <li><f12> <f7> e</f7></f12></li> </ul> The following commands are markup languages to describe You can also use Graphviz, see	(hide-ifdef-use-define-alist NAME)  (hif-clear-all-ifdef-defined)  (hif-evaluate-macro RSTART REND)  used to create images from specially because the common of the common	The value is saved inside the hide-ifdef-define-alist variable.  The list is not saved to disk. You may want to pre-create the value for a given project and store inside your local directory variables for example.  Set 'hide-ifdef-env' to the already saved symbol list with the specified NAME.  Takes the value from the hide-ifdef-define-alist.  Clears all symbols defined in 'hide-ifdef-env'.  It first backup this variable to 'hide-ifdef-env-backup' before clearing to prevent accidental clearance.  Evaluate the macro expansion result for the active region.  If no region active, find the current #ifdefs and evaluate the result.  Only supports math calculations; strings or argumented macros can not be expanded.  Decific markup code embedded inside C source code comments. This can be useful when using these machines for example.  Render the PlantUML markup embedded in current mode comment.  Use region if identified otherwise use PlantUML block at point.  Uses prefix (as PREFIX) to choose where to display it:  4 (when prefixing the command with C-u) -> new window		
environment list into a named list  Use a named symbol environment list  Clear the complete list of #define'd symbols inside 'hide-ifdef-env'  Evaluate pre-processor macro  Rendering markup embedded in comments  Preview UML diagram from plantUML source in current plantUML region of commented source code	<ul> <li><f12> # D</f12></li> <li><f12> <f7> D</f7></f12></li> </ul> • C-c @ U <ul> <li><f12> # U</f12></li> <li><f12> <f7> U</f7></f12></li> </ul> • C-c @ C <ul> <li><f12> # C</f12></li> <li><f12> <f7> C</f7></f12></li> </ul> • C-c @ e <ul> <li><f12> # e</f12></li> <li><f12> # e</f12></li> <li><f12> <f7> e</f7></f12></li> </ul> The following commands are markup languages to describe You can also use Graphviz, see	(hide-ifdef-use-define-alist NAME)  (hif-clear-all-ifdef-defined)  (hif-evaluate-macro RSTART REND)  used to create images from specially because the common of the common	The value is saved inside the hide-ifdef-define-alist variable.  The list is not saved to disk. You may want to pre-create the value for a given project and store inside your local directory variables for example.  Set 'hide-ifdef-env' to the already saved symbol list with the specified NAME.  Takes the value from the hide-ifdef-define-alist.  Clears all symbols defined in 'hide-ifdef-env'.  It first backup this variable to 'hide-ifdef-env-backup' before clearing to prevent accidental clearance.  Evaluate the macro expansion result for the active region.  If no region active, find the current #ifdefs and evaluate the result.  Only supports math calculations; strings or argumented macros can not be expanded.  Decific markup code embedded inside C source code comments. This can be useful when using these machines for example.  Render the PlantUML markup embedded in current mode comment.  Use region if identified otherwise use PlantUML block at point.  Uses prefix (as PREFIX) to choose where to display it:		
environment list into a named list  Use a named symbol environment list  Clear the complete list of #define'd symbols inside 'hide-ifdef-env'  Evaluate pre-processor macro  Rendering markup embedded in comments  Preview UML diagram from plantUML source in current plantUML region of commented source	<ul> <li><f12> # D</f12></li> <li><f12> <f7> D</f7></f12></li> </ul> • C-c @ U <ul> <li><f12> # U</f12></li> <li><f12> <f7> U</f7></f12></li> </ul> • C-c @ C <ul> <li><f12> # C</f12></li> <li><f12> <f7> C</f7></f12></li> </ul> • C-c @ e <ul> <li><f12> # e</f12></li> <li><f12> # e</f12></li> <li><f12> <f7> e</f7></f12></li> </ul> The following commands are markup languages to describe You can also use Graphviz, see	(hide-ifdef-use-define-alist NAME)  (hif-clear-all-ifdef-defined)  (hif-evaluate-macro RSTART REND)  used to create images from specially because the common of the common	The value is saved inside the hide-ifdef-define-alist variable.  The list is not saved to disk. You may want to pre-create the value for a given project and store inside your local directory variables for example.  Set 'hide-ifdef-env' to the already saved symbol list with the specified NAME.  Takes the value from the hide-ifdef-define-alist.  Clears all symbols defined in 'hide-ifdef-env'.  It first backup this variable to 'hide-ifdef-env-backup' before clearing to prevent accidental clearance.  Evaluate the macro expansion result for the active region.  If no region active, find the current #ifdefs and evaluate the result.  Only supports math calculations; strings or argumented macros can not be expanded.  Edific markup code embedded inside C source code comments. This can be useful when using these machines for example.  Render the PlantUML markup embedded in current mode comment.  Use region if identified otherwise use PlantUML block at point.  Uses prefix (as PREFIX) to choose where to display it:  4 (when prefixing the command with C-u) -> new window  16 (when prefixing the command with C-u C-u) -> new frame.  else -> new buffer  This can be used inside buffer using any major mode, when PlantUML markup is embedded inside		
environment list into a named list  Use a named symbol environment list  Clear the complete list of #define'd symbols inside 'hide-ifdef-env'  Evaluate pre-processor macro  Rendering markup embedded in comments  Preview UML diagram from plantUML source in current plantUML region of commented source code	<ul> <li><f12> # D</f12></li> <li><f12> <f7> D</f7></f12></li> </ul> • C-c @ U <ul> <li><f12> # U</f12></li> <li><f12> <f7> U</f7></f12></li> </ul> • C-c @ C <ul> <li><f12> # C</f12></li> <li><f12> <f7> C</f7></f12></li> </ul> • C-c @ e <ul> <li><f12> # e</f12></li> <li><f12> # e</f12></li> <li><f12> <f7> e</f7></f12></li> </ul> The following commands are markup languages to describe You can also use Graphviz, see	(hide-ifdef-use-define-alist NAME)  (hif-clear-all-ifdef-defined)  (hif-evaluate-macro RSTART REND)  used to create images from specially because the common of the common	The value is saved inside the hide-ifdef-define-alist variable.  The list is not saved to disk. You may want to pre-create the value for a given project and store inside your local directory variables for example.  Set 'hide-ifdef-env' to the already saved symbol list with the specified NAME.  Takes the value from the hide-ifdef-define-alist.  Clears all symbols defined in 'hide-ifdef-env'.  It first backup this variable to 'hide-ifdef-env-backup' before clearing to prevent accidental clearance.  Evaluate the macro expansion result for the active region.  If no region active, find the current #ifdefs and evaluate the result.  Only supports math calculations; strings or argumented macros can not be expanded.  Ecific markup code embedded inside C source code comments. This can be useful when using these machines for example.  Render the PlantUML markup embedded in current mode comment.  Use region if identified otherwise use PlantUML block at point.  Uses prefix (as PREFIX) to choose where to display it:  4 (when prefixing the command with C-u) -> new window  16 (when prefixing the command with C-u) -> new frame.  else -> new buffer  This can be used inside buffer using any major mode, when PlantUML markup is embedded inside source code comment.		
environment list into a named list  Use a named symbol environment list  Clear the complete list of #define'd symbols inside 'hide-ifdef-env'  Evaluate pre-processor macro  Rendering markup embedded in comments  Preview UML diagram from plantUML source in current plantUML region of commented source code	<ul> <li><f12> # D</f12></li> <li><f12> <f7> D</f7></f12></li> </ul> • C-c @ U <ul> <li><f12> # U</f12></li> <li><f12> <f7> U</f7></f12></li> </ul> • C-c @ C <ul> <li><f12> # C</f12></li> <li><f12> <f7> C</f7></f12></li> </ul> • C-c @ e <ul> <li><f12> # e</f12></li> <li><f12> # e</f12></li> <li><f12> <f7> e</f7></f12></li> </ul> The following commands are markup languages to describe You can also use Graphviz, see	(hide-ifdef-use-define-alist NAME)  (hif-clear-all-ifdef-defined)  (hif-evaluate-macro RSTART REND)  used to create images from specially because the common of the common	The value is saved inside the hide-ifdef-define-alist variable.  ⚠ The list is not saved to disk. You may want to pre-create the value for a given project and store inside your local directory variables for example.  Set 'hide-ifdef-env' to the already saved symbol list with the specified NAME.  • Takes the value from the hide-ifdef-define-alist.  Clears all symbols defined in 'hide-ifdef-env'.  • It first backup this variable to 'hide-ifdef-env-backup' before clearing to prevent accidental clearance.  Evaluate the macro expansion result for the active region.  • If no region active, find the current #ifdefs and evaluate the result.  ⚠ Only supports math calculations; strings or argumented macros can not be expanded.  setfic markup code embedded inside C source code comments. This can be useful when using these machines for example.  Render the PlantUML markup embedded in current mode comment.  • Use region if identified otherwise use PlantUML block at point.  • Uses prefix (as PREFIX) to choose where to display it:  • 4 (when prefixing the command with C-u) -> new window  • 16 (when prefixing the command with C-u C-u) -> new frame.  • else -> new buffer  • This can be used inside buffer using any major mode, when PlantUML markup is embedded inside source code comment.		
environment list into a named list  Use a named symbol environment list  Clear the complete list of #define'd symbols inside 'hide-ifdef-env'  Evaluate pre-processor macro  Rendering markup embedded in comments  Preview UML diagram from plantUML source in current plantUML region of commented source code	<ul> <li><f12> # D</f12></li> <li><f12> <f7> D</f7></f12></li> </ul> • C-c @ U <ul> <li><f12> # U</f12></li> <li><f12> <f7> U</f7></f12></li> </ul> • C-c @ C <ul> <li><f12> # C</f12></li> <li><f12> <f7> C</f7></f12></li> </ul> • C-c @ e <ul> <li><f12> # e</f12></li> <li><f12> # e</f12></li> <li><f12> <f7> e</f7></f12></li> </ul> The following commands are markup languages to describe You can also use Graphviz, see	(hide-ifdef-use-define-alist NAME)  (hif-clear-all-ifdef-defined)  (hif-evaluate-macro RSTART REND)  used to create images from specially because the common of the common	The value is saved inside the hide-ifdef-define-alist variable.  The list is not saved to disk. You may want to pre-create the value for a given project and store inside your local directory variables for example.  Set 'hide-ifdef-env' to the already saved symbol list with the specified NAME.  Takes the value from the hide-ifdef-define-alist.  Clears all symbols defined in 'hide-ifdef-env'.  It first backup this variable to 'hide-ifdef-env-backup' before clearing to prevent accidental clearance.  Evaluate the macro expansion result for the active region.  If no region active, find the current #ifdefs and evaluate the result.  Only supports math calculations; strings or argumented macros can not be expanded.  Reific markup code embedded inside C source code comments. This can be useful when using these machines for example.  Render the PlantUML markup embedded in current mode comment.  Use region if identified otherwise use PlantUML block at point.  Uses prefix (as PREFIX) to choose where to display it:  4 (when prefixing the command with C-u) -> new window  16 (when prefixing the command with C-u) -> new frame.  else -> new buffer  This can be used inside buffer using any major mode, when PlantUML markup is embedded inside.		

<u>Description</u>	<u>Keystroke</u>	Function	<u>Note</u>
C Specific search and replace	value to true or false. Compar		place functions used to detect and fix code that explicitly compare a pointer to NULL and a boolean oor C or C++ code and should be replaced. The following commands help locating such code and ses the keyword.
Problematic code	Problem: C code that compa	re pointer against NULL and va	alue against TRUE, true, FALSE, and false.
Search for poor code using comparison	<f12> s n</f12>	(pel-c-search-equal_NULL)	Move point to the next expression like if (ptr == NULL) or if (NULL == ptr)
against NULL	<f12> s N</f12>	(pel-c-search-not- equal_NULL)	Move point to the next expression like if (ptr != NULL) or if (NULL != ptr)
Search for poor code using comparison against false or FALSE	<f12> s f</f12>	(pel-c-search-equal_false)	Move point to the next expression like if (boolean == false) or if (false == boolean). Also search for FALSE.
against laise of FALSE	<f12> s F</f12>	(pel-c-search-not- equal_false)	Move point to the next expression like if (boolean != false) or if (false != boolean).  Also search for FALSE.
Search for poor code using comparison against true or TRUE	<f12> s t</f12>	(pel-c-search-equal_true)	Move point to the next expression like if (boolean == true) or if (true != boolean).  Also search for TRUE
against trac or TrioL	<f12> s T</f12>	(pel-c-search-not- equal_true)	Move point to the next expression like if (boolean != true) or if (true != boolean).  Also search for TRUE
Search for any of the poor code listed in the previous 6 commands	<f12> s *</f12>	(pel-c-search-any- comparison-problem	Move point to the next instance of any of the expressions searched by the 6 commands above.
Improve C/C++ code: remove explicit comparisons against NULL, TRUE, FALSE, true and false	<f12> s C-f</f12>	(pel-c-fix-comparison-problems)	Replace all instances of C/C++ code that explicitly compares a pointer against NULL or a boolean value against true, false, TRUE and FALSE by the logically equivalent expression that does not use the keyword:  For example this replaces:  if (pointer == NULL) by if (!pointer)  if (value == TRUE) by if (value)  if (value == FALSE) by if (!value)  if (value == true) by if (value)  if (value == true) by if (value)  if (pointer != NULL) by if (pointer)  if (value != TRUE) by if (!value)  if (value != TRUE) by if (!value)  if (value != TRUE) by if (!value)  if (value != FALSE) by if (!value)  if (value != true) by if (!value)  if (value != true) by if (value)  if (value != false) by if (value)  if (value != false) by if (value)  if (value != false) by if (value)  The command can detect and reformat a large number of expressions but not all of them.  Therefore it's a good idea to backup the original code and check the difference after executing this command to detect potential errors. If the translation is correct there should be no change in the generated assembler code.  The best way to check if the reformatting is correct is to compare the generated assembler code for the file before and after the code reformatting done by this command.  With GCC toolchain, use the objdump disassemble command on the object file to generate the assembler files. The LLVM toolchain provide the llvm-objdump equivalent.
Problematic code	• Instead of: #if VAR • Instead of: #if VAR ==	write #if ((defined	
Search for poor pre- processor conditional #if VAR	<f12> s #</f12>	(pel-c-search-preproc-if)	Move point to the end of the next <b>#if VAR</b> expression.
Search for poor pre- process conditional #if VAR==0 #if VAR==1	<f12> s 0</f12>	(pel-c-search-preproc-if- set)	Move point to the end of the next #if VAR == 0 expression or #if VAR == 1 expression.
Improve C/C++ code: remove explicit comparisons against NULL, TRUE, FALSE, true and false	<f12> s C-p</f12>	(pel-c-fix-preproc-if- problems)	Inside current buffer, replace all instances of problematic C pre-processor conditional code listed below with the corresponding safer code.  Instead of: #if VAR
Programming Help	PEL has bindings for the follow	ving commands that are useful	when editing source code, markup files or any file that has a mode that supports imenu.
Show what completion mode is currently used.	<f11> M-c ?</f11>	(pel-show-active- completion-mode)	Display the completion mode currently used.
Show function at point	<f11> ? F</f11>	(pel-show-function)	Display the name of the current "function" at point in the mini-buffer.
Toggle which-function- mode to display name of current function at point	• <f11> ? f • <f11> M-d f</f11></f11>	(which-function-mode &optional ARG)	Toggle mode line display of current function (Which Function mode).  • With a prefix argument ARG, enable Which Function mode if ARG is positive, and disable it otherwise.
See also:  • ∑ Menus • ∑ Mode Line  • ∑ Mode Line  • The concept of "function" is major mode specific. For example, in C++ mode, if point is inside a class definition it shows the name of the class.	<ul> <li>The which-function-mode is a global minor mode. When enabled, the current function name is continuously displayed in the mode line.</li> <li>⚠ Detection of functions and variables depend on the imenu functionality. If you modify the content of a buffer, you need to force a menu rescan to get proper results. You can force a rescan with pel-imenu-rescan, bound to <f11> <f10> r.</f10></f11></li> <li>☑ Identify major modes that automatically active the mode with which-function-mode user-option.</li> <li>Use M-x customize-option which-function-mode to open the relevant customization buffer.</li> <li>With PEL you can use:         <f11> <f3> to access the which-func customization group. It will provide access to the customization group even when the feature has not yet been loaded, something that Emacs does not do by default.</f3></f11></li> <li><f11> <f2> o which-function-mode RET to access the user-option directly.</f2></f11></li> </ul>		

## Emacs & C - References

Document	Notes
GNU emacs - CC Mode Manual	
GNU Emacs Manual - Styles	
Emacs BSD/Allman Style with 4 Space Tabs?	
Emacs: Linux Kernel Style but with Allman/BSD Style Braces?	
Emacs Wiki - Indenting C	
Indent preprocessor directives as C code in emacs	Does not fully address the way I want to have multi-indentations for pre-processor
elisp code - ppindent.el	Implements pre-processor indentation with the # always in the first column. Not yet exactly what I want.
company-mode ; Modular in-buffer completion framework for Emacs	
C Programming Language Information	<ul> <li>C @ Wikipedia, C history</li> <li>C/C++ OPerator precedence @ Wikipedia: a very good source of information. Read the notes after the table!</li> <li>C Reference @ cppreference.com</li> <li>The C Book, second edition</li> </ul>