## **Programming Language Support — C**

	Pi	rogramming La	anguage Support — C					
<u>Description</u>	<u>Keystroke</u>	Function	<u>Note</u>					
Editing C Files  CC Mode  File extensions  C  bison-mode  Speedbar	Emacs supports C natively via the built-in c-mode which extends the CC Mode that support the curly-bracket programming languages like C.  PEL activates support for C when one or both of the pel-use-c or pel-use-bison user-option variables is set.  Supported file extensions: code files: .c, .lex, .y, .yacc, header files: .h, .i Add more in pel-auto-mode-alist user option.  If pel-use-bison-mode is set, the .y and .yacc files are associated with bison-mode, the .l and .lex with flex-mode and .jison with jison-mode  When pel-use-speedbar is set all these extensions are recognized by speedbar, otherwise only the main ones are recognized.  Important aspects of C source code syntax controlled by the CC Mode are customizable with PEL user option variables.  PEL customization for C: Simplifies configuration for editing C source code.							
<ul> <li>indentation</li> </ul>	<ul> <li>Emacs customization group: pel-pkg-for-c (access with <f12> <f2>):</f2></f12></li> <li>pel-c-indent-width: Identifies the number of columns used for indentation. Defaults to 3.</li> <li>pel-c-tab-width: The width of a tab used for c-mode files. Defaults to 3.</li> <li>This concept differs from indentation: you can have an indentation of 3 and tab width of 8: M-i will move point to columns that are multiple of 8 <tab> will indent to a column that is a multiple of 3. PEL stores this value inside the tab-width variable for c-mode buffers.</tab></li> <li>For most uses it is best to set both values to the width of your needed indentation level. This way you can use commands that use either to</li> </ul>							
<ul> <li>using tabs</li> <li>bracket style</li> </ul>	pel-c-use-tabs: Wheth     C code style sub-group     pel-c-fill-column:     When pel-c-fill-co     pel-c-backet-style:     define your own 'use     More user options a below.     Emacs customization gro     pel-cc-auto-newline:     The values for those user	<ul> <li>control the indentation level.</li> <li>pel-c-use-tabs: Whether hard tabs are used in indentation or not: t: tabs are used, nil: only spaces are used. Default: nil.</li> <li>C code style sub-group: pel-c-code-style</li> <li>pel-c-fill-column: column where line-wrapping occurs: maximum line length (defaults to 80). You can change the value or set it nil.</li> <li>When pel-c-fill-column user option is nil, c-mode buffers use the Emacs fill-column value like other major modes.</li> <li>pel-c-backet-style: The bracket/indentation style supported by the electric keys. You can select one of the values supported by Emacs or define your own 'user' with some Emacs Lisp code. Default to "linux".</li> <li>More user options are used for controlling C code templates created with PEL tempo skeletons. They are described in tempo skeleton section below.</li> <li>Emacs customization group: pel-pkg-for-cc. Applies to all CC Mode related modes (like c-mode).</li> <li>pel-cc-auto-newline: Whether automatic newline mode is active on all CC Mode (including c-mode).</li> <li>The values for those user option variables can also be stored inside directory local files and even as file local variables. You can also modify them for each buffer and view their current settings using the commands listed in the following set of rows. See File/Directory Variables for more info.</li> </ul>						
Find C header file			The <b>pel-c-file-finder-method</b> determines how <b>pel-open-at-point</b> searches for header files.					
PEL key prefixes:			efixes: <f11> SPC c as well as <f12> and <m-f12> n in keystroke cells (for brevity). The other two prefixes are only available in c-mode buffers.</m-f12></f12></f11>					
Open this PDF file. See also: <u>N Help/Info</u>	<f11> SPC c <f1><f12> <f1></f1></f12></f1></f11>	(pel-help-pdf &optional OPEN-WEB-PAGE)	Open the $\mathfrak{M} - \mathbf{C}$ local PDF. If the prefix argument (like $\mathbf{C} - \mathbf{u}$ or $\mathbf{M} - \mathbf{-}$ ) is used, then it opens the remote GitHub hosted raw PDF instead. If the <b>pel-flip-help-pdf-arg</b> user-option is set it's the other way around.					
<u>▼ Customize</u> 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-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-u</b> ), display in another window.					
CC Mode Style Management • Learn style used in current buffer	Automatic indentation, brace format style and several other C stylistic elements are controlled by the CC Mode and the CC mode variables.  You can impose an indentation style by customization.  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 <i>learns</i> the style and sets the style controlling variables from what it detects in the buffer.  Use this to adapt to source code written by others and want to continue using the same style.  For the following commands all commands that use a key binding that ends with an upper case letter install the style.							
Show/Modify syntactic context	C-c C-o	(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. The optional argument is not used.					
Show syntactic information for current line	C-c C-s	(c-show-syntactic- information ARG)	Show syntactic information for current line.  Display the syntactic information list and highlight the reference position(s) listed as argument to the syntactic list.  Each list starts with a syntactic symbol 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>	(c-guess-buffer-no-install &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>	( <b>c-guess-buffer</b> &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.					
Guess style in the region and install it.	<f12> <f4> g G</f4></f12>	(c-guess &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>	( <b>c-guess-install</b> &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.					
CC Mode support Behaviour control	interactively with prefix argument, the value for WITH-NAME is asked to the user.  The following commands can be used to dynamically change the behaviour of important keys such as the return key, delete key, semi-colon, etc  The CC Mode controls the indentation and bracket style which controls what happens when electric characters are typed (when the electric mode is activated) and provide a better experience when editing C source code.  • CC Mode state displayed in the mode line: \$C{}\$ where:  • \$2\$ is the CC mode programming language name: C, C++, ObjC, etc  • 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  • 'a' for auto-newline mode  • 'h' for hungry mode  • 'w' for subword mode							
			1					

<u>Description</u>	<u>Keystroke</u>	Function	<u>Note</u>				
Toggle Electric state	• C-c C-l • <f12> <f4> e</f4></f12>	(c-toggle-electric-state &optional ARG)	Toggle the electric indentation feature done with the electric character keys.  • Optional numeric ARG, if supplied, turns on electric indentation when positive, turns it off when negative, and just toggles it when zero or left out.				
Set indentation style	• C-c . • <f12> <f4> s</f4></f12>	(c-set-style STYLENAME &optional DONT-OVERRIDE)	Set the <u>bracket/indentation style</u> for the current buffer.  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 for current buffer	<f12> <f4> TAB</f4></f12>	(pel-cc-set-indent-width &optional NEW-WIDTH)	Interactively change the Indentation with for current buffer to NEW-WIDTH.  • Prompt for new value.  • Use 0 to restore value specified by configuration (pel-c-indent-width).  String can be used to change indentation several times in a file.				
Toggle syntactic indentation	<f12> <f4> i</f4></f12>	(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.				
	When it's turned off, the electric M-x c-indent-command adjusted.	in turned on (the default), the indentation functions and electric keys indent according to syntactic context keys, when applicable ectric keys don't reindent, the indentation functions indents every new line to the same level as the previous nonempty line, and the indentation in steps specified by 'c-basic-offset'. The indentation style has no effect in this mode, nor any of the liables, e.g. 'c-special-indent-hook'.					
Toggle Comment Style	• C-c C-k • <f12> <f4> M-;</f4></f12>	(c-toggle-comment-style &optional ARG)	Toggle the comment style between block and line comments.  Optional numeric ARG, if supplied, switches to block comment style when positive, to line comment style when negative, and just toggles it when zero or left out.  The C++ style // comments (also called line comments) are compatible with C since C-99.				
Toggle Hungry Delete mode	<f12> <f4> DEL</f4></f12>	( <b>c-toggle-hungry-state</b> &optional ARG)	Toggle hungry-delete-key feature. Affects < DEL> and C-d keys.  Optional numeric ARG, if supplied, turns on hungry-delete when positive, turns it off when 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 text alignment on pel-newline-and-indent-below See also:  • <u>∑ Align</u> • <u>∑ Indentation</u>	<f11> M-RET</f11>	(pel-toggle-newline-indent-align)	Toggle variable pel-newline-does-align for the local buffer.  This toggles the way function 'pel-newline-and-indent-below' operates.  If pel-newline-does-align is t, it aligns several syntactic element in the current block: the comments, the assignments.  Bildentify modes where pel-newline-does-align is automatically activated (set to t) by adding the major mode to the list in the pel-modes-activating-align-on-return user option.  This affects the behaviour of the following commands:  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 auto-newline insertion mode	• C-c C-a • <f12> <f4> M-RET</f4></f12>	( <b>c-toggle-auto-newline</b> &optional ARG)	Toggle <u>auto-newline</u> feature.  Optional numeric ARG, if supplied, turns on auto-newline when positive, turns it off when negative, and just toggles it when zero or left out.  Turning on auto-newline automatically enables <i>electric indentation</i> .  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, semi-colon, and colon.				
Change RET key behaviour: select return mode.	<f12> <f4> RET</f4></f12>	(pel-cc-change-newline-mode)	Change the way the RET key behaves in the CC modes and display the new mode in the echo area. Changes from one mode to the next and then rotate to the first one. The modes are:  1. context-newline: the default: uses (c-context-line-break) with the extra ability to repeat its execution with an argument.  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> ?</f4></f12>	(pel-cc-mode-info)	Display information about current <b>CC mode</b> derivative for the current c-mode buffer.				
settings	CC mode style currently act The Emacs the c-default-st used to set the style for c-m Return key behaviour: RET (return key) mode. C Whether return performs State of electric C characte whether it is active or not if auto-newline on some of The fill column: the column Tab width and whether hard In c-mode buffer use <f (c-c="" a="" and="" and<="" be="" can="" cc="" changed="" column.="" command="" continuation="" create="" currently="" delete="" dyn="" exam="" file="" for="" having="" hungry="" identification="" ind="" indentation="" is="" lines="" methods="" mode="" new="" or="" otherwise="" own.="" remember="" search="" stars,="" stolement="" style="" style.="" styles="" suppo="" tab="" td="" that="" the="" then="" this="" to="" two="" updates="" us="" used="" usused="" w="" whether="" your=""><td colspan="6">ode. Change with pel-cc-change-newline-mode (<f12> <f4> RET). forms alignment. Change that with pel-toggle-indent-align (<f11> M-RET). haracters (toggle it on/off with c-toggle-electric-state (C-c C-l or <f12> <f4> e): or not, and when active what character(s) exhibit electric behaviour. Some characters (';' and some other based on style) is active. Toggle it with C-c C-a or <f12> <f4> M-RET.  column where force line wrap is done when the auto-fill-mode is active. Toggle auto fill mode with <f11> RET.  er hard tabs are used. These are set by the user options pel-c-tab-width and pel-c-use-tabs.  use <f12> <f2> to open the appropriate customization buffer to change them.  It tab width does not identify the indentation. It controls the spacing used in some commands moving point to the next tab stop on is controlled separately. See next line.  Introlled by c-basic-offset normally set by pel-c-indent-width in PEL and whether syntactic indentation mode is active. Shows how it was override by executing the pel-cc-set-indent-width command for this buffer (use <f12> <f4> TAB) for that command.  ed for indentation and bracket positioning (they should have the same value). Emacs identifies several built-in styles but you can example below shows "bsd" with is another name for the Allman style. You can dynamically change for the current buffer with c-set- or <f12> <f4> s).  s identify everything, including the number of indentation columns. PEL configures the style from the requested pel-c-bracket-style the indentation and other settings from the PEL user option requested. This allows you to slightly modify an existing style without new style name for it.  Supports C-style (/* */) and C++-style (//) comments since both are now accepted in C since C99.  ed dynamically for the current buffer with the c-toggle-comment-style command (C-c C-k or <f12> <f4> M-;). C comment 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 here is used by DEL and</f4></f12></f4></f12></f4></f12></f2></f12></f11></f4></f12></f4></f12></f11></f4></f12></td></f>	ode. Change with pel-cc-change-newline-mode ( <f12> <f4> RET). forms alignment. Change that with pel-toggle-indent-align (<f11> M-RET). haracters (toggle it on/off with c-toggle-electric-state (C-c C-l or <f12> <f4> e): or not, and when active what character(s) exhibit electric behaviour. Some characters (';' and some other based on style) is active. Toggle it with C-c C-a or <f12> <f4> M-RET.  column where force line wrap is done when the auto-fill-mode is active. Toggle auto fill mode with <f11> RET.  er hard tabs are used. These are set by the user options pel-c-tab-width and pel-c-use-tabs.  use <f12> <f2> to open the appropriate customization buffer to change them.  It tab width does not identify the indentation. It controls the spacing used in some commands moving point to the next tab stop on is controlled separately. See next line.  Introlled by c-basic-offset normally set by pel-c-indent-width in PEL and whether syntactic indentation mode is active. Shows how it was override by executing the pel-cc-set-indent-width command for this buffer (use <f12> <f4> TAB) for that command.  ed for indentation and bracket positioning (they should have the same value). Emacs identifies several built-in styles but you can example below shows "bsd" with is another name for the Allman style. You can dynamically change for the current buffer with c-set- or <f12> <f4> s).  s identify everything, including the number of indentation columns. PEL configures the style from the requested pel-c-bracket-style the indentation and other settings from the PEL user option requested. This allows you to slightly modify an existing style without new style name for it.  Supports C-style (/* */) and C++-style (//) comments since both are now accepted in C since C99.  ed dynamically for the current buffer with the c-toggle-comment-style command (C-c C-k or <f12> <f4> M-;). C comment 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 here is used by DEL and</f4></f12></f4></f12></f4></f12></f2></f12></f11></f4></f12></f4></f12></f11></f4></f12>					
Notice the name of	-UU-:F1 c_file.c c-mode state:		WK Anzu Fly <sup>2</sup> ElDoc Abv) 10:35am 1.97				
the PEL user-options that set the significant feature controlling Emacs variables in the message More info is shown	- active style : bs - RET mode : cc - Electric characters : ac - Auto newline : or	n ), auto-filling: off. Set via	<pre>: pel-c-tab-width(8) ==&gt; tab-width(8) when c-mode buffer is opened. : pel-c-use-tabs(nil) ==&gt; indent-tabs-mode(nil) when c-mode buffer is opened.</pre>				
in that buffer as buttons that provide access to more help and ability to customize the values.	- Indent width : 4 - Syntactic indent : or - c-indentation-style : bs - PEL Bracket style : bs - Comment style : Bl	4 Set via: pel-c-indent-width(4) ==> c-basic-offset(4) when c-mode buf on bsd bsd Block comments: /* */ , continued line start with *					
		gry delete seeds: off, but the F11-⊠ and F11-⊠ keys are available.					

<u>Description</u>	<u>Keystroke</u>	oke Function Note					
C Code Help	There are several Emacs exter	ere are several Emacs extension packages that can help writing C code.					
Get man help about C code See: <u>N Help/Info</u>	• <f11> ? m • <m-f8> • ₩-M</m-f8></f11>	(man MAN-ARGS)	Open a Man page inside an Emacs window. See ∑ Help/Info for more info about man.  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 c-eldoc mode	<f12> ? e <f11> SPC c ? e</f11></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. It currently appears when typing a new function with its arguments inside the code.				
	• Ithis package could be i		pel-use-c-eldoc is set to t.  The extra processing required may slow Emacs.  Description only on demand but a LSP-based system might be more performant anyway. I will reformances and the feature set. c-eldoc uses the cpp command to preprocess the buffer content.				
Electric Keys	0		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)  Indicate the description of	Insert a "#".  e variable 'c-electric-pound-behavior', which can only be nil or 'alignleft'. If a numeric ARG is				
0	If 'c-syntactic-indentation' a a literal.  Whitespace between a func	(c-electric-paren ARG) and 'c-electric-flag' are both no	Insert a parenthesis.  Insert a parenthesis.  Insert a parenthesis is inserted inside 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'.  Inil, some newline cleanups are done if appropriate; see the variable 'c-cleanup-list'.				
{}	• { }	(c-electric-brace ARG)	Insert a brace.				
	a) If the auto-newline fea     in 'c-hanging-braces-a     b) Any auto-newlines are	ture is turned on (indicated by alist'. 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 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 fea 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.  o colons will be "cleaned up" leaving a scope operator, if this action is set in 'c-cleanup-list'.				
:,	<ul> <li>tr'c-electric-flag' is non-nil, point isn't inside a literal and a numeric ARG hasn't been supplied, the command performs several electric actions:         <ul> <li>When the auto-newline feature is turned on (indicated by "/la" on the mode line) a newline might be inserted. See the variable 'c-hanging-semi&amp;comma-criteria' for how newline insertion is determined.</li> <li>Any auto-newlines are indented. The original line is also reindented unless 'c-syntactic-indentation' is nil.</li> <li>If auto-newline is turned on, a comma following a brace list or a semicolon following a defun might be cleaned up, depending on the settings of 'c-</li> </ul> </li> </ul>						
Electric pairs	Type the first of a pair to ins	ert this one and its matching ch	vactivating the <b>electric-pair-mode</b> in the buffer.  naracter for (), [], $\{\}$ , "" and ".  In elighter set by the pel-electric-pair-lighter is shown. This defaults to $E(1)$				
Toggle electric-pairmode in current buffer $\dagger$ Lighter:= $E(1)$	<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.				
Insert New Line(s)	active the point also moves to  With PEL the default behavi command (bound to <f12>  The pel-cc-newline comman</f12>	the proper indentation accord iour can be selected by custom <b>M-RET</b> ) see the CC-Mode bel nd also aligns comments and a	Mode electric mode is active or not. When it is not active it simply inserts a new line. When it is ing to the syntactic context. The following commands can also be used.  nization and modified dynamically for the current buffer with the pel-cc-change-newline-mode haviour control section above.  assignment in the code block if the pel-modes-activating-align-on-return user option list to buffer can 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>	• electric-indent-just-newline • If 'pel-newline-does-align' is t, then perform the text alignment done by the function 'align'.  Use: (c-context-line-break): Do a line break suitable to the context.  • When point is outside a comment or macro, insert a newline and indent according to the syntactic context, unless 'c-syntactic-indentation' is nil, in which case the new line is indented as the previous non-empty line instead.  • When point is inside the content of a preprocessor directive, a line continuation backslash is inserted before the line break and aligned appropriately. The end of the cpp directive doesn't count as inside it.  • When point is inside a comment, continue it with the appropriate comment prefix (see the 'c-comment-prefix-regexp' and 'c-block-comment-prefix' variables for details). The end of a C++-style line comment doesn't count as inside it.  • When point is inside a string, only insert a backslash when it is also inside a preprocessor directive.						
See also:  •	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.  • If option 'use-hard-newlines' is non-nil, the newline is marked with the text-property 'hard'.  • If 'electric-indent-mode' is enabled, this indents the final new line that it adds, and reindents the preceding line.  • To just insert a newline, use M-x electric-indent-just-newline.  Calls 'auto-fill-function' if the current column number is greater than the value of 'fill-column' and ARG is nil.  Use: (electric-indent-just-newline ARG): Insert just a newline, without any auto-indentation.  • With ARG, insert that many newlines.						
Insert an indented line below unbroken current line See also:  Indentation	• M-RET • <f11> <tab> RET</tab></f11>	(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.  Bellentify 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>				

Insert a newline	C-j	(electric nowline and			
		(electric-newline-and- maybe-indent)	Insert a newline.  • If 'electric-indent-mode' is enabled, that's that, but if it is "disabled" then:		
	In programming langu	age modes, this is the same as			
Open New Line in			mmand indents to the column specified by the function 'current-left-margin'.		
Context	C-o	(c-context-open-line)	Insert a line break suitable to the context and leave point before it.  b', which is normally bound to <b>C-o</b> . See 'c-context-line-break' for the details.		
See also:  • Whitespace	Normally C-o is bound to	ppen-line. PEL rebinds it to c-context-open-line for the CC modes. e line without indenting the next use open-line via <f12> C-o</f12>			
Open new line	• <f12> C-o • <m-f12> C-o</m-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.		
<u>C Comments</u>	C supports 2 types of comments     '*': Block Comments	: /* comment	*/		
	• '/' : Line Comments	(c-electric-slash ARG)	to end of line (available since C-99)  Insert a slash character.		
Comment character wrapping long comment lines	-	(C-electric-stasti Ali G)	<ul> <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: <u>▼ Filling/Justificatio</u>	vn *	( <b>c-electric-star</b> 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		
O constant to a			** 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, the line is wrapped and the comment continuation string used is automatically inserted. (toggle it with <fil> RET)</fil>		
Comment/un- comment  ★★ See also: ▼ Comment	M-;	(pel-c-comment-dwim ARG)	Comment line or region with // or /* */ style comments depending on the comment style currently 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-region BEG END &optional ARG)	Comment or uncomment each line in the region.  • With just <b>C-u</b> prefix arg, uncomment each line in region BEG END.		
See also: <u>Commer</u>	The strings used as comme end' and 'comment-paddir on each line (even for synta style'.  If you try this when no regions.	<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 termina 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</li> </ul>			
Fill current paragra See also:  Filling/Justification	• <f12> F</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</m-f12></f12></f11>	( <b>subword-mode</b> & optional ARG)	<ul> <li>Toggle subword-mode: a minor mode that treats sections of <u>camelCase</u> and <u>PascalCase</u> as distinct words.</li> <li>With a prefix argument ARG, enable Subword mode if ARG is positive, and disable it otherwise.</li> </ul>		
Hide/Show commers See also:  • <u>Somments</u>	<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.  • This requires the <a href="hide-comnt.el">hide-comnt.el</a> package (see <a href="December Comments">S Comments</a> ). <a href="Light PEL">Light PEL activates it when the pel-use-hide-comnt user option is t.</a>		

<u>Description</u>	<u>Keystroke</u>	Function	<u>Note</u>			
Hungry Deletion of Whitespace	PEL provides the conver     In modes compatible with the of the simple < DEL> and C.     When the Hungry Delete     The Hungry Mode also as mode is activated (and the line modes derived from Content of the modes. PEL provided to the modes.  Toggle hurry deletion mode	le provides two commands that can perform "hungry whitespace deletion" that can also be used in every mode.  rovides the convenient keys with the <f11> prefix keys for those 2 commands, available in all 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 ple <del> and C-d, to perform hungry deletions. That's not currently supported in other modes.  the Hungry Delete Mode is on, the mode-line displays a 'h' to the right of the '//l' indication of electric mode.  Ingry 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 is activated (and that can only be done in modes that are CC Mode compatible).  The standard delete commands delete hungrily, but that does not work for loodes. PEL provides the <f12> M-DEL key for those modes (like C).  The provides the C-d key for the current buffer with c-toggle-hungry-state (<f12> M-DEL).</f12></f12></del></f11>				
Delete preceding char or all preceding whitespace.  See also: <u>See 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-\Overline{\Overli			
Delete next char or all following whitespace.  See also: <u>See Cut &amp; Paste</u>	• C-c C-d • C-c 🗵 • C-c C-🗵 • C-c <c-delete> • <f11> 🗵</f11></c-delete>	(c-hungry-delete-forward)	with 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-⊠ 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></c-delete>			
Indentation	behaviour control section abov You can also explicitly request • The first set of commands p	ve. indentation using the comman perform syntactic indentations s				
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:  •	<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 you can use:</li> <li>pel-indent-rigidly, bound to C-x <tab> and to <f11> <tab><tab><tab><tab><tab><tab><tab><tab></tab></tab></tab></tab></tab></tab></tab></tab></f11></tab></li></ul>					
Indent lines of list after point See also:  Indentation	C-M-q	(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 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 Indentation	A	, it's best to set pel-c-tab-wid	gards to semantics. More information on indentation is available in the <u>Natural Indentation</u> table.  Ith and pel-c-indent-width to the same value: the first 2 commands use the value of pel-c-tab-			
Insert spaces or tabs to next defined tabstop column See also: <u> Indentation</u>	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:  • ∑ Indentation  • ∑ Key-Chords	• C-x <tab> • <f11> <tab> <tab> • <tab>q</tab></tab></tab></f11></tab>	Indent rigidly the marked region or current line N times tab-width columns.  • If a region is marked, it uses 'indent-rigidly' and provides the same prompts to control indentation changes.  • If no region is marked, it operates on current line(s) identified by the numeric argument N (or if not specified N=1):  • N = [-1, 0, 1] : operate on current line  • N > 1 : operate on the current line and N-1 lines below.  • N < -1 : operate on the current line and (abs N) -1 lines above.				
	PEL rebinds this key, but it extends the functionality: pel-indent-rigidly uses the original indent-rigidly.  indent-rigidly Indent all lines starting in the region.  • If called interactively with no prefix argument, activate a transient mode in which the indentation can be adjusted interactively by typing <left>, <right>, <s-left>, or <s-right>.  Both of these commands activate a transient mode where Emacs prompts for extra keys to control how to indent. Indenting and un-indenting is possible. The capabilities are controlled by the variable indent-rigidly-map with by default provides:  • S-<right> indent-rigidly-right-to-tab-stop  • <right> indent-rigidly-left-to-tab-stop  • <right> indent-rigidly-right  • <left> indent-rigidly-left  Typing any other key deactivates the transient mode.  • The S-<right> and S-<left> keys indent/de-indent to the next tab-stop position, which is controlled by the tab-width user option.  • 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.  ⚠ If you use the cua-mode: the cua-mode uses C-x, to invoke this command when cua-mode is active, type it really fast or type C-x C-x <tab> (or use the PEL binding <fi11> <tab> &lt;</tab></tab></tab></tab></tab></tab></tab></tab></tab></tab></tab></tab></tab></tab></tab></tab></tab></tab></tab></tab></tab></tab></tab></tab></tab></tab></tab></tab></tab></tab></tab></tab></tab></tab></tab></tab></tab></tab></tab></tab></tab></tab></tab></tab></tab></tab></tab></tab></tab></tab></tab></tab></tab></tab></tab></tab></tab></tab></tab></tab></tab></tab></tab></tab></tab></tab></tab></tab></tab></tab></tab></tab></tab></tab></tab></tab></tab></tab></tab></tab></tab></tab></tab></tab></tab></tab></tab></tab></tab></tab></tab></tab></tab></tab></tab></tab></tab></tab></tab></tab></tab></tab></tab></tab></tab></tab></tab></tab></tab></tab></tab></tab></tab></tab></tab></tab></tab></tab></tab></tab></tab></tab></tab></tab></tab></tab></tab></tab></tab></tab></fi11></tab></left></right></left></right></right></right></s-right></s-left></right></left>					

<u>Description</u>	<u>Keystroke</u>	Note				
Indent line(s) rigidly See also:	• <f6> <tab> • <f11> <tab> c</tab></f11></tab></f6>	(pel-indent-lines &optional <u>N</u> )	Indent current or marked lines by N indentation levels controlled by <b>pel-c-indent-width</b> .  • Works with point anywhere on the line.			
• <u>Nation</u>	specified, 'pel-unindent-line modifies the region to inclu Handles presence of hard to If indent-tabs-mode is no	es' is used. If a region is marked all characters in all affected abs: on-nil the indentation is created I, any hard tab in the indentatio	gument N can specify more than one indentation level. It defaults to 1. If a negative number is d, the function does not deactivate it to allow repeated execution of the command. It also lines. Use <b>C-g</b> to de-activate the region.  With a mix of hard-tabs and space characters.  In of the marked lines is replaced by the proper number of spaces. Hard tabs after first non-			
Un-indent line(s) rigidly	• <backtab> • <f6> <backtab> • <f11> <tab> C</tab></f11></backtab></f6></backtab>	(pel-unindent-lines &optional N)	Un-indent current line or marked lines by N indentation levels controlled by pel-c-indentwidth. Works with point is anywhere on the line.			
See also:  • Note Indentation	<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 in all affected lines. Use C-g to de-activate the region.</li> <li>Handles presence of hard tabs: <ul> <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 whitespace character on the line are left.</li> </ul> </li> </ul>					
Open file at point	In a c-mode buffer the com		e taken at point (the cursor location). e useful for C programming and has the extra capability of searching files where header files are user-options:			
See also: <b>File mngt</b>	pel-c-file-finder-method     pel-c-file-searched-ext     pel-c-file-finder-ini-tool the pel.ini file, a INI-format cochain can be overridden by the executed). Use the commandation of the pel.ini (see the commandation).	d : identifies one of ra-dir-trees : List of extra dir : The name of a infiguration file. The value map e value of the environment varied pel-cc-set-file-finder-ini-to-	f 4 supported method of identifying the header files. See their descriptions below. ectory trees also searched by the tool identified by <b>pel-ffind-executable</b> user-option. tool chain <i>TTT</i> , to select one of the <i>TTT-c-path</i> tool-chain key inside the [file-finder] section of ped to that key identifies the list of directories to search for that tool-chain. The name of the tool able PEL_CC_FIND_TOOLCHAIN, which is read and used when Emacs starts up (or pel-init is <b>pol-name</b> to change the currently used tool chain name.			
			er-option. With PEL you can control it globally or locally with <b><f11> f M</f11></b>			
Set name of Tool- Chain specific include path • (when the pel-ini-file search method is used)	• <f12> <f4> <m-f6> • <f12> <f4> <f54></f54></f4></f12></m-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.  Only 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-f6> • <f11> f . • <u>6y</u></f11></m-f6>	(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.</f4></f12>			
<u>.</u>	This command works generic	ally but is also specialized for C	The <u>6y</u> key-chord is available if <b>pel-use-key-chord</b> is non-nil. See <u>Ney-Chords</u> .  Comajor mode: it opens the header file identified by the #include statement.			
C Header File finding control	Aside from generic method de and the pel-c-file-searched-tree(s) to search using the sea • generic: the command s project-root-identifiers • pel-ini-file: the comman	escribed below, the command sextra-dir-trees user-options. The pel-fif bearches, in current directory ar user-option. Something like gd searches inside directories id	searches for the header file to open using the method identified by the <b>pel-c-file-finder-method</b> . The first one identifies one of the following search method, the other identifies extra directory			
pel-use-ini F	The file must contain a [f The project-path key. One or several TTT-c The currently used	<b>iile-finder</b> ] section with: The value is a list of directorie path key(s), where TTT is a too tool chain is identified by the form.				
specialized for:  • M reStructuredText  • MI - C++  • MI - Erlang	<ul> <li>The paths identified</li> <li>You can modify this to</li> <li>With several TTT-c-</li> </ul>	d in the two lists may use environt of the two lists may use environt of the two lists may use environt of the two lists are the pel.ini files.	me user-option; which identifies the name of a TTT-c-path key.  In ment variables inside the path strings. Use the \$VARNAME format to identify them.  In editing session by typing <f12> <f4> <m-f6> and specifying another name.  In you can adjust the include path dynamically for various tool chains.  In the include path dynamically for various tool chains.  In the include path dynamically for various tool chains.</m-f6></f4></f12>			
• PI - UNIX Shell	explicit lists: two lists of		options. s the project directory names, the other hold the tool and library directory names. The lists may bles. The \$VARNAME format must be used. Directories are not searched recursively.			
Generic Delimiting characters	The generic mode extraction		and possibly line and column numbers, from text at point and tries to open the file or directory.  nning & end of the file/directory/library/URL name string by delimiter characters, one of: tab,  [] «» · · 〇 () () · 。			
The complete file detection heuristic is described in the File mngt description of the same command.	When finding several file names, the command lists them and prompts using the method selected by <b>pel-prompt-read-method</b> user-option.  The default is a very primitive function implemented by PEL. You can select a more powerful ivv prompting instead.  With ivy selected, PEL will automatically set pel-use-ivy to t and ivy mode will be installed automatically when you restart Emacs.  Note that the command shows all files found by the specified search method, it does not only use the first one found.  Use this to detect potential duplication in header file names in large include paths.					
Select target window <del>►</del>	Select target window:     Without argument:     If file or directory is alr	•	ollowing logic controlled by presence or absence of typed numerical prefix arguments:  we point to that window and to the line column coordinates if specified following the file name at			
N>20 : open the directory ►	<ul> <li>If no window holds the window, if 2: use the c</li> <li>With prefix numeric arguments in N &lt; 0: create a new w</li> <li>(abs N) &gt; 20: then ope</li> <li>N = 0: use the 'other'</li> </ul>	<ul> <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> <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 <u>prefix numeric argument</u> N:</li> <li>N &lt; 0: create a new window and use that.</li> <li>(abs N) &gt; 20: then open the <b>directory</b> 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 doestring for more info.	<ul> <li>N is: 8: up, 2: down, 4</li> <li>N is 9: force opening (eg. macOS Finder, W</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 RL, open it in the OS default web browser. not allowed.			

<u>Description</u>	<u>Keystroke</u>	Function	<u>Note</u>			
Tempo skeletons			ugh the Emacs built-in tempo skeleton mechanism.			
for C	• PEL creates key bindings to invoke the skeletons in the supported major modes, using the same key prefix sequence for each mode: <f12> &lt; the same key bindings for equivalent concepts (such as file header block) as much as possible.</f12>					
See also:			style is controlled by the user options inside the <b>pel-c-code-style</b> group. This group can be edited			
C Code Templates		• from a C mode buffer and inc ler-block-style : allows selecti	elude the following options: ing a user-define module-header comment block.			
• Naserting Text for	pel-c-skel-comment-wi     pel-c-skel-insert-file-tin		nat of C-style continuation comments. utomatically updated timestamp is inserted in the file header block.			
more info and	pel-c-skel-use-separate		iks use horizontal separator lines.			
information about tempo skeleton and	<ul> <li>pel-c-skel-doc-markup</li> <li>pel-c-skel-cfile-section</li> </ul>		cumentation markup supported by the templates. Currently 'none' and 'Doxygen' are available.			
yasnippet template- based text insertion	pel-c-skel-hfile-section	-titles : identifies docum	entation section titles inserted in header files. A section titled "." split sections placed before			
	pel-c-skel-insert-function	lude guard. If not present all sections are placed after the include guard.  nction templates are inserted in the function description comment.				
	• pel-c-skell-function-section-titles : identifies the title of the C function templates sections inserted when pel-c-skel-insert-function-section-section-style : select the C function comment block style. Several styles are provided:					
		- no special com	ment rmat style to describe the function above its code.			
		- a Man-page sty	yle comment block with the sections identified by pel-c-skell-function-section-titles			
	pel-c-skel-function-nan	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	the state of the s	copy right and code license is specified. An option provide ability to insert open source text controlled by 🛂 lice.			
	pel-c-use-include-guard	ds : specify which typ	be of include guard is inserted in header files. The available choices are:			
			clude guard pragma once statement			
			assic #ifdef/#define/#endif block using symbol created from file name #ifdef/#define/#endif block using symbol created from file name and UUID for its uniqueness.			
	Emacs user options by defa		vusing 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 control block at the end of that file. If you want to control the behaviour of the PEL tempo			
	templates for all files inside a	directory tree create a .dir-local	s file and store the values of the relevant options variables inside that file. This allows you to applates precisely and does not affect what you actually type.			
	Once a skeleton was just er	tered (or later by activating the	e pel-tempo-mode) you can move to the next or previous point of interest (so called tempo-marks)			
		•	M-b 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)	Customize PEL C skeleton layout.  • If OTHER-WINDOW is non-nil (use <b>C-u</b> ), display in another window.			
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.			
			<ul> <li>Prompts for the file purpose.</li> <li>For header files, include guard is inserted if requested by customization.</li> </ul>			
			The layout of the entered text is controlled by user options. It is possible to create a user- specified skeleton this command will used instead of the one provided by PEL.			
			<ul> <li>See examples of generated outputs located in <a href="mailto:example/templates/c/files">example/templates/c/files</a> repo directory.</li> <li>Access the customization buffer by typing: <f12> <f2></f2></f12></li> </ul>			
Insert #define	<f12> <f12> d</f12></f12>	(pel-c-define)	Insert a C pre-processor #define statement.			
	1222	(per o dermo)	If there is text between the beginning of the line and point, insert the statement on the next			
			line, otherwise insert it on the current line, even if there is text after point (to allow inserting it before the name of the symbol to define).			
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>If there is text between the beginning of the line and point, insert the statement on the next line, otherwise insert it on the current line.</li> </ul>			
			<ul> <li>If there is text after point, insert a new line to place that text on the next line.</li> <li>The .h extension is written between the angle brackets and point left right before the period.</li> </ul>			
			The next tempo mark is placed at the end of the line (so C-c · move point there).			
Insert #include ".h"	<f12> <f12> I</f12></f12>	(pel-c-include-local)	Insert a C pre-processor <b>#include</b> "" statement to include a local file.  • If there is text between the beginning of the line and point, insert the statement on the next			
			line, otherwise insert it on the current line.  • If there is text after point, insert a new line to place that text on the next line.			
		The .h extension is written between the angle brackets and positive control of the second control of the				
Insert a function	<512> <512> 5	(not a function)	The next tempo mark is placed at the end of the line (so <b>C-c</b> · move point there).			
definition with	<f12> <f12> f</f12></f12>	(pel-c-function)	Insert a C function definition code and comment template.			
comment block	<ul> <li>The command prompts for the function name and its purpose.</li> <li>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.</li> </ul>					
	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 accepted					
	and trimmed and dash characters ('-') are automatically replaced by underscores ('_') for convenience.  If an invalid name is specified it is erased and you are prompted again. Use M-p to bring the old value back.					
	Prompts for function and		tories. Use <b>M-p</b> and <b>M-n</b> to navigate in the histories at the prompt. You can also use the <b><up></up></b>			
	and <b><down></down></b> keys.  • The style of the code inserte	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	• C-c M-f	(tempo-forward-mark)	Jump to the next mark in 'tempo-back-mark-list': the location where code must be updated			
mark	• C-c .		inside the inserted skeleton.  • These key key bindings are only available when pel-tempo-mode is active.			
Jump to previous	• C-c M-b	(tempo-backward-mark)	Jump to the previous mark in 'tempo-back-mark-list': the location where code must be updated			
tempo mark	• C-c ,		inside the inserted skeleton.  • These key binding are only available when pel-tempo-mode is active.			
Tempo Template Tag	• C-c C-, <f12> <f12> <f12></f12></f12></f12>	(tempo-complete-tag	Look for a tag and expand it.			
Insertion	\112/ \112/ \112/	&optional SILENT)	LOUN TO A LAY AND EXPANDIL.			
		•	you can type the template name (shown in the title column like "if", "case", etc) completely or			
	1	•	buffer opens up if the template name is incomplete (or empty in which case the buffer lists <b>all</b> RET. Emacs expands the template.			
	All the tags in the tag lists in	'tempo-local-tags' (this includ	les 'tempo-tags') are searched for a match for the text before the point. The way the string to			
	match at all.		empo-match-finder'. If 'tempo-match-finder' returns nil, then the results are the same as no			
	SILENT is non-nil, the functi		xpanded in place of the matching string. If a partial completion or no match at all is found, and completion is found and 'tempo-show-completion-buffer' is non-nil, a buffer containing possible			
	completions is displayed.					

<u>Description</u>	<u>Keystroke</u>	Function	<u>Note</u>				
Inserting code	Extra text insertion can be dor	ne with the following commands	s. See also above: <f12> M-e activates electric pair: typing ( inserts the matching )</f12>				
Insert Parentheses	M- (	(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.  • A negative ARG encloses the preceding ARG sexps instead.				
	PEL makes parens-require following a function (and with	lent to zero: just insert '()' and leave point between.  quire-spaces buffer local and set it to nil in C mode buffers, allowing the use of this command to insert the argument parentheses and without placing a space between the function name and the opening parenthesis.  t enclosing characters at region boundaries.  This command assumes point is not in a string or comment.					
Marking	Emacs provides the following	des the following command to quickly mark the whole content of the current function. More mark commands exists, see the <u>Narking</u> table.					
Mark the complete	C-M-h	(c-mark-function)	Mark complete function.				
function body  See also: Marking	If point is not inside any them     A mark is left where the con	rent top-level declaration or macro, point at beginning. en the closest following one is chosen. Each successive call of this command extends the marked region by one function. ommand started, unless the region is already active (in Transient Mark mode). d C-M-e, this function does not require the declaration to contain a brace block.					
Getting Syntactic Information	Use the following commands t	o extract syntactic information	from the source code.				
Display name of current function	• C-c C-z • <f12> f • <m-f12> f</m-f12></f12>	( <b>c-display-defun-name</b> &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	The state of the s		is often used. Using superword-mode helps searching.				
Toggle superword-	PEL activates the superword r  • <f11> t m p</f11>	(superword-mode	change this use the <f11> t <f2> to access the customize buffer.  Toggle superword-mode: a minor mode that treats <a href="mailto:snake_case">snake_case</a> as one word. In C '_' are treated</f2></f11>				
mode See also:  • <u>∑ Text Modes</u> • <u>∑ Search/Replace</u>	• <f12> M-p</f12>	&optional ARG)	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 his	ghlights the parens that match	seful modes to highlight blocks of (), {}, and []. es the one before or after point. re highlighted with the same colour.				
Toggle show-paren mode on/off	• <f12> M-9 • <m-f12> M-9 • <f11> h (</f11></m-f12></f12>	( <b>show-paren-mode</b> &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 highlighted in the parenthesis is a single parenthesis.				
See also: <u>National Highlight</u> Enable/Disable	• <f12> M-r</f12>	(rainbow-delimiters-mode	highlighted in 'show-paren-style' after 'show-paren-delay' seconds of Emacs idle time.  Highlight nested parentheses, brackets, and braces with colours according to their depth.				
coloured highlight of nested blocks (),{},[]	• <m-f12> M-r</m-f12>	&optional ARG)	<ul> <li>Customize the depth and colours with M-x customize-group rainbow-delimiters</li> <li>Requires: rainbow-delimiters.el</li> </ul>				
See also: <u>Neighlight</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 describe the specialized commands only. See the others inside <u>Navigation</u>						
By definitions  Find definition of		(xref-find-definitions	for more information to activate the various engines that support cross referencing for C code.  Grab symbol at point and move cursor to its definition.				
identifier at point  See also: Xref	M	IDENTIFIER)	<ul> <li>If there are more than one match, prompt in the *xref* buffer.</li> <li>To search for a symbol entered manually, type C-u M</li> <li>With dumb-jump this performs a search using ag, ripgrep or git grep if available.</li> </ul>				
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	ckage 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 C pre-processor	Move across <b>C preprocessor</b>	conditional inclusion stateme	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- <right></right>				
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-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.				
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 <u>occur</u> 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 occur search.				
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)	<ul> <li>Show C pre-processor conditional statements of current project buffers inside an occur buffer.</li> <li>Each line is shown with NLINES before and after, or -NLINES before if NLINES is negative.</li> <li>NLINES defaults to list-matching-lines-default-context-lines user-option value.</li> <li>See occur search.</li> <li>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></li> </ul>				

<u>Description</u>	<u>Keystroke</u>	Function	<u>Note</u>
By functions     By structures			ucture definition blocks. Jump over comments.  er closing brace and <b>show-paren-mode</b> is on, the matching parentheses are highlighted.
Forward to start of next top level function or struct	<f12> <down></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.  It moves forward but not to the end of the function definition (like end-of-defun) but to the</f12></f6></f6></f6>
Forward to end of current top-level function or struct.	С-м-е	( <b>c-end-of-defun</b> &optional ARG)	beginning of the function definition, which is often what users of other editors expect.  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 procedure and
iunction or struct.	C-M- <end></end>	(end-of-defun &optional	preceding end.  Move forward to the end of next function or type definition.
	<f12> <right></right></f12>	ARG)	With argument, do it that many times. Negative argument -N means move back to Nth preceding end of defun.  ➡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 <b>top-level</b> function. It skips nested functions.</f12></f6>
Backward to beginning of current top-level function or struct	С-М-а	(c-beginning-of-defun &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.
<u>sauc.</u>	C-M- <home> <f12> <up></up></f12></home>	(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>
Backward to end of previous top level function or struct	<f12> <left></left></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>
By blocks	Move across C statements a	and C scope blocks, or any gro	oup of (), [], {} or <> blocks.
By List element	Move to the end or the be		
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.
Move block backward  See also:  ■ Navigation	• C-M-b • C-M- <left> • C-[ C-b • Esc C-b • Esc C-<left></left></left>	(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- <left> : ► Shift marking works with this command.  • ⚠ With PEL: if you want to use Esc C-<left> binding you must ensure that pel-windmove-on-esc-cursor user option is set to nil, otherwise it does something else.  • C-M-<left> does not work on Windows, but H-<left> works.  ■ Several Linux distros map C-M-<left> 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.</left></left></left></left></left>
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 forward  See also:  • ∑ Navigation	• C-M-f • C-M- <right> • C-[ C-f • Esc C-f • Esc C-<right></right></right>	( <b>forward-sexp</b> &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 pelwindmove-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>
in/out of blocks		blocks, or any group of (), [], {}	
Backward Up/outside sexp hierarchy See also:  Navigation	• C-M-u • C-M- <up> • C-[ C-u • Esc C-u • Esc C-<up></up></up>	(backward-up-list &optional ARG ESCAPE- STRINGS NO-SYNTAX- CROSSING)	Move backward out of one level of parentheses or nested blocks.  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.  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.  C-M-u: Shift marking is available in graphics mode, not in terminal mode.  C-M-<up> ir Shift marking works with this command.  C-M-<up> does not work on Windows, but H-<up> does.</up></up></up></up>
Forward Up/outside sexp hierarchy See also: Navigation	С-м-]	(up-list &optional ARG ESCAPE-STRINGS NO- SYNTAX-CROSSING)	Move forward out of one level of parentheses or nested blocks.  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 backward but still to a less deep spot.

<u>Description</u>	<u>Keystroke</u>	Function	<u>Note</u>
Down/inside sexp/ block  See also:  Navigation	• C-M-d • C-M- <down> • C-[ C-d • Esc C-d</down>	(down-list &optional ARG)	Move forward down one level of parentheses.  • 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 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>
By statements	Move to beginning /end of sta	tement of comment sentence.	
Go to beginning of statement (backward)	м-а	(c-beginning-of-statement &optional COUNT LIM SENTENCE-FLAG)	<ul> <li>Go to the beginning of the innermost statement.</li> <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.
<u>C Preprocessor</u>	would not be executed with the external packages are listed by PEL provides a key <a "#else"="" "#if",="" #elif"="" at="" backward,="" but="" by="" followed="" forward.<="" function="" going="" href="https://www.hydra.com/hyd&lt;/th&gt;&lt;th&gt;e Hide-ifdef mode. There are a elow. They can be used for ed elep navigate trough preprocess art with &lt;f12&gt; &lt;f7&gt; open the&lt;/th&gt;&lt;th&gt;nal statements, allow expansion of preprocessor macros, hiding pre-processor statements that also external packages that provide extra support. All commands provided by Emacs and iting C and C++ source code.  sor directives and to hide/show code areas based on preprocessor logic and defined variables. The pel-∑c-preproc Hydra allowing further hydra keys to be typed without any prefix. The se when the pel-use-hydra user option is set to t.&lt;/th&gt;&lt;/tr&gt;&lt;tr&gt;&lt;th&gt;Open the C preprocessor hydra with &lt;f12&gt; &lt;f7&gt; followed by on of the hydra keys:&lt;/th&gt;&lt;th&gt;n: next   #: p: prev   W: C-p: begin   R: C-n: end   H: C-u: up   S:   h:&lt;/th&gt;&lt;th&gt;toggle mode e: toggle shadow d: toggle RO u: hide U: show D:&lt;/th&gt;&lt;th&gt;(C/*la Ifdef WK Fly &lt;sup&gt;2&lt;/sup&gt; Anzu Abbrev)  Vars&lt;/th&gt;&lt;/tr&gt;&lt;tr&gt;&lt;th&gt;Navigate across pre-&lt;br&gt;processor&lt;br&gt;conditionals&lt;/th&gt;&lt;th&gt;&lt;/th&gt;&lt;th&gt;&lt;/th&gt;&lt;th&gt;&lt;b&gt;#elif&lt;/b&gt; and &lt;b&gt;#endif&lt;/b&gt; C pre-processor conditional statements.  bre-processor command used by Emacs. The default depends on the operating system.&lt;/th&gt;&lt;/tr&gt;&lt;tr&gt;&lt;th&gt;Move to previous preprocessor directive&lt;/th&gt;&lt;th&gt;• &lt;f12&gt; # p&lt;br&gt;* &lt;f12&gt; &lt;f7&gt; p&lt;/th&gt;&lt;th&gt;(pel-pp-prev-directive)&lt;/th&gt;&lt;th&gt;Move point to previous preprocessor directive.&lt;/th&gt;&lt;/tr&gt;&lt;tr&gt;&lt;th&gt;Move to next preprocessor directive&lt;/th&gt;&lt;th&gt;• &lt;f12&gt; # n&lt;br&gt;* &lt;f12&gt; &lt;f7&gt; n&lt;/th&gt;&lt;th&gt;(pel-pp-next-directive)&lt;/th&gt;&lt;th&gt;Move point to next preprocessor directive.&lt;/th&gt;&lt;/tr&gt;&lt;tr&gt;&lt;th&gt;Move up in the pre-&lt;br&gt;processor conditional&lt;br&gt;block&lt;/th&gt;&lt;th&gt;• C-c C-u&lt;br&gt;* &lt;f12&gt; &lt;f7&gt; C-u&lt;/th&gt;&lt;th&gt;(c-up-conditional COUNT)&lt;/th&gt;&lt;th&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="" th="" 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	• C-c C-e • <f12> # # • <m-12> # #</m-12></f12>	(c-macro-expand START END SUBST)	Expand C macros in the region, using the C preprocessor.  Normally display output in temp buffer, but prefix arg means replace the region with 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'.
Insert/align or delete end-of-line backslash	C-c C-\	(c-backslash-region FROM TO DELETE-FLAG &optional LINE-MODE)	Insert, align, or delete end-of-line backslashes on the lines in the region.  With no argument, inserts backslashes and aligns existing backslashes.  With an argument, deletes the backslashes.
	backslash (if any) at the end You can put the region arou	of the previous line is deleted.  nd an entire macro definition a	e region. If the region ends at the start of a line and the macro doesn't continue below it, the and use this command to conveniently insert and align the necessary backslashes.  It is a conveniently insert and align the necessary backslashes.  It is a conveniently insert and align the necessary backslashes.
Show state preprocessor modes	• <f12> # ? * <f12> <f7> ?</f7></f12></f12>	(pel-pp-show-state)	Show state of C preprocessor control modes on the echo area.  • Also displays the hide-ifdef-env and the hide-ifdef-define-alist variables by the Hide-ifdef mode (see next page)  • If too long, see the information in the *Messages* buffer.

Description	<u>Keystroke</u>	Function	<u>Note</u>					
Hide-ifdef Mode     hide/show code controlled by C-preprocessor	It supports complete C/C++     It scans for new #define syr     It hides blocks of code the Hide-ifdef environment: t	expression and precedence. inbols and macros. in at would not be include in the	expanded file according to the state of pre-processor symbols that are maintained inside the list Emacs variable (use <f1> v to see the content of Emacs variables). See <u>Nelp/Info</u>. ()</f1>					
	You can make your     Access it hide-if	tious when editing near ellipses, since the hidden text is still in the buffer, and you can move the point into it and modify text unawares. can make your buffer read-only while hide-ifdef-hiding by setting hide-ifdef-read-only user-option to a non-nil value. ccess it hide-ifdef customization group with <f12> # <f3> can toggle this variable with hide-ifdef-toggle-read-only (with C-c @ C-q) or with <f12> # r or <f12> <f7> R.</f7></f12></f12></f3></f12>						
	* The key sequences that si	eart with <f12> <f7> open th</f7></f12>	refix keys can also be reached via the <m-f12> and the <f11> SPC c prefix keys.  ne pel-∑c-preproc Hydra allowing further hydra keys to be typed without any prefix.</f11></m-f12>					
Use <f12> # ?</f12>	Several customize user op to change, execute: M 'hide-ifdef-env'	tion variables affect how the hi -x customize-group hi	tes when the <b>pel-use-hydra</b> user option is set to <b>t</b> .  ding is done:  de-ifdef or type <f12> # <f3>  project. The list holds the following forms:</f3></f12>					
to show the value of hide-ifdef-env and hide-ifdef-define-alist	(SYMBOL) is used     (SYMBOL . VALUE)     'hide-ifdef-define-alist'     An association list of p	when the SYMBOL is defined (I when the symbol is defined with	but without explicit value) th an explicit value.  hide-ifdef-set-define-alist' to save the current 'hide-ifdef-env' and 'hide-ifdef-use-define-alist' to					
	Set to non-nil to not s  'hide-ifdef-initially' Indicates whether 'hid 'hide-ifdef-read-only' Set to non-nil if you w	now #if, #ifdef, #ifndef, #else, a e-ifdefs' should be called wher ant to make buffers read only w ad-only status is restored to pre	h Hide-Ifdef mode is activated.  while hiding.					
Toggle the Hide-Ifdef mode : • hide/show code suppressed by C preprocessor	• <f12> M-# • <m-f12> M-# * <f12> <f7> # • <f11> SPC c M-#</f11></f7></f12></m-f12></f12>	( <b>hide-ifdef-mode</b> &optional ARG)	Toggle features to hide/show #ifdef blocks (Hide-Ifdef mode).  • With a prefix argument, enable Hide-Ifdef mode if ARG is positive, and disable it otherwise.  • 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 hidden from view.					
Toggle read-only mode when text is hidden	• C-c @ C-q • <f12> # r * <f12> <f7> R</f7></f12></f12>	(hide-ifdef-toggle-read- only)	Toggle read-only: toggle 'hide-ifdef-read-only'.  Note that you can make the file read only by default when hide-ifdef is hiding text, by setting the 'hide-ifdef-read-only' user option to t.					
Toggle shadowing of hidden text.	• C-c @ C-w • <f12> # w * <f12> <f7> W</f7></f12></f12>	(hide-ifdef-toggle- shadowing)	Toggle shadowing.  • When shadowing is on, text that would be hidden is "shadowed" instead: it is displayed with the shadow face (normally something dim, all depending of the theme used).					
Hide code suppressed by C preprocessor	• C-c @ h • <f12> # H • <m-f12> # H * <f12> <f7> H • <f11> SPC c # H</f11></f7></f12></m-f12></f12>	(hide-ifdefs &optional NOMSG)	Hide the contents of some #ifdefs.  • Assume that defined symbols have been added to 'hide-ifdef-env'.  • The text hidden is the text that would not be included by the C preprocessor if it were given the file with those symbols defined.  • With prefix command presents it will also hide the #ifdefs themselves.  Turn off hiding by calling 'show-ifdefs'.					
Restore all hidden into view	• C-c @ s • <f12> # S * <f12> <f7> S</f7></f12></f12>	(show-ifdefs)	Cancel the effects of 'hide-ifdef': show the contents of all #ifdefs.					
Hide part of current block that would not be included	• C-c @ C-d • <f12> # h * <f12> <f7> h</f7></f12></f12>	(hide-ifdef-block &optional ARG START END)	Hide the ifdef block (true or false part) enclosing or before the cursor.  • With optional prefix argument ARG, also hide the #ifdefs themselves.					
Show all parts of the current #ifdef block	• C-c @ C-s • <f12> # s * <f12> <f7> s</f7></f12></f12>	(show-ifdef-block &optional START END)	Show the ifdef block (true or false part) enclosing or before the cursor.					
Set a variable to a specific value	• C-c @ d • <f12> # d * <f12> <f7> d</f7></f12></f12>	(hide-ifdef-define VAR &optional VAL)	Define a VAR to VAL (default 1) in 'hide-ifdef-env'.  This allows hiding the block inside <b>#ifndef VAR</b> (or the equivalent) by executing the command hide-ifdefs.					
Undefine a variable	• C-c @ u • <f12> # u * <f12> <f7> u</f7></f12></f12>	(hide-ifdef-undef START END)	Undefine a VAR  • This allows hiding the blocks inside <b>#ifdef VAR</b> (or the equivalent) by executing the command hide-ifdefs.					
Save the symbol environment list into a named list	• C-c @ D • <f12> # D * <f12> <f7> D</f7></f12></f12>	(hide-ifdef-set-define-alist NAME)	Save the state of the current <b>hide-ifdev-env</b> to a list with the specified NAME for later re-use. The value is saved inside the <b>hide-ifdef-define-alist</b> variable.  The list is not saved to disk. You may want to pre-create the value for a given project and store it inside your local directory variables for example.					
Use a named symbol environment list	• C-c @ U • <f12> # U * <f12> <f7> U</f7></f12></f12>	(hide-ifdef-use-define-alist NAME)	Set 'hide-ifdef-env' to the already saved symbol list with the specified NAME.  Takes the value from the hide-ifdef-define-alist.					
Clear the complete list of #define'd symbols inside 'hide- ifdef-env'	• C-c @ C • <f12> # C * <f12> <f7> C</f7></f12></f12>	(hif-clear-all-ifdef-defined)	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 pre- processor macro	• C-c @ e • <f12> # e * <f12> <f7> e</f7></f12></f12>	(hif-evaluate-macro RSTART REND)	Evaluate the macro expansion result for the active region.  • If no region active, find the current #ifdefs and evaluate the result.  A Currently it supports only math calculations; strings or argumented macros can not be expanded.					

<u>Description</u>	<u>Keystroke</u>	F	unction	<u>Note</u>
Rendering markup embedded in	The following commands are unthese markup languages to determine the second sec			ecific markup code embedded inside C source code comments. This can be useful when using state machines for example.
comments	You can also use Graphviz, see	e <u>M</u> Graphv	iz Dot	
Preview UML diagram	<f12> u</f12>	(pel-rende	r-commented-	Render the PlantUML markup embedded in current mode comment.
from plantUML source in current plantUML region of commented source code  See also: M PlantUML		plantuml PREFIX &optional POS)		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.  Use this in source code to describe your code architecture with PlantUML markup, then generate the UML rendering by moving point inside the PlantUML block and issuing this command.  Requires the plantuml-mode external package, activated by pel-use-plantuml user option being non-nil.
C Specific search and replace		Comparing	against these sym	lace functions used to detect and fix code that explicitly compare a pointer to NULL and a bols is poor C or C++ code and should be replaced. The following commands help locating such
Problematic code				alue against TRUE, true, FALSE, and false.
Search for poor code	<f12> s n</f12>	,	<u> </u>	Move point to the next expression like if (ptr == NULL) or if (NULL == ptr)
using comparison against NULL	<f12> S N</f12>	(pel-c-sear	rch-not-	Move point to the next expression like if (ptr != NULL) or if (NULL != ptr)
Search for poor code using comparison	<f12> s f</f12>		rch-equal_false)	Move point to the next expression like if (boolean == false) or if (false == boolean).  Also search for FALSE.
against false or FALSE	<f12> s F</f12>	(pel-c-sea		Move point to the next expression like if (boolean != false) or if (false != boolean).
Search for poor code	<f12> s t</f12>	equal_false (pel-c-sear	e) rch-equal_true)	Also search for FALSE.  Move point to the next expression like if (boolean == true) or if (true != boolean).
using comparison against true or TRUE	<f12> s T</f12>	(pel-c-sea		Also search for TRUE  Move point to the next expression like if (boolean != true) or if (true != boolean).
Search for any of the	<f12> s *</f12>	equal_true	<u>,                                      </u>	Also search for TRUE  Move point to the next instance of any of the expressions searched by the 6 commands above.
poor code listed in the previous 6 commands		(pel-c-search-any- comparison-problem		, , , , , , , , , , , , , , , , , , , ,
remove explicit comparisons against NULL, TRUE, FALSE, true and false		(pel-c-fix-comparison-problems)		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 == false) by if (!value) • if (value == false) by if (!value) • if (pointer != NULL) by if (pointer) • if (value != TRUE) by if (!value) • if (value != TRUE) by if (!value) • if (value != FALSE) by if (value) • if (value !=
Problematic code	<ul> <li>Instead of: #if VAR</li> <li>Instead of: #if VAR == (</li> <li>Instead of: #if VAR == 1</li> </ul>	write #  write #  write #	if ((defined) if (!defined) if (defined)	
Search for poor pre- processor conditional #if VAR	<f12> s #</f12>	(pel-c-sea	rch-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-sea set)	rch-preproc-if-	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  Document</f12>	(pel-c-fix-preproc-if-problems)  Emacs &		Inside current buffer, replace all instances of problematic C pre-processor conditional code listed below with the corresponding safer code.  Instead of: #if VAR it writes #if ((defined(VAR) && (VAR != 0)))  Hest References == 0 it writes #if (!defined(VAR)   (VAR == 0))  Instead of: #if VAR == 1 it writes #if (defined(VAR) && (VAR == 1))  Notes
GNU emacs - CC Mode	Manual			
GNU Emacs Manual - S				
	<u> </u>			
	nacs BSD/Allman Style with 4 Space Tabs?  nacs: Linux Kernel Style but with Allman/BSD Style Braces?			
Emacs Wiki - Indenting				
	_		Does not fully add	dress the way I want to have multi-indentations for pre-processor
•	t preprocessor directives as C code in emacs		,	• •
elisp code - ppindent.e	<u> </u>		implements pre-p	processor indentation with the # always in the first column. Not yet exactly what I want.
company-mode ; Modu Emacs	llar in-buffer completion frame	work for		