Programming Language Support — C

	Pr	ogramming La	anguage Support — C
Description	<u>Keystroke</u>	Function	<u>Note</u>
Editing C Files CC Mode File extensions C	Emacs supports C natively via the built-in c-mode which extends the CC Mode that support the c-mode which extends the CC Mode that support the c-mode which extends the CC Mode that support the c-mode which extends the c-mode which		
• <u>bison-mode</u>	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		
• <u>∑ Speedbar</u>	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. Emacs customization group: pel-pkg-for-c (access with <f12> <f2>):</f2></f12>		
indentation	 pel-c-indent-width: Identifies the number of columns used for indentation. Defaults to 3. pel-c-tab-width: The width of a tab used for c-mode files. Defaults to 3. 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> 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 		
using tabsbracket style	control the indentation level. • pel-c-use-tabs: Whether hard tabs are used in indentation or not: t: tabs are used, nil: only spaces are used. Default: nil. • C code style sub-group: pel-c-code-style • pel-c-fill-column: column where line-wrapping occurs: maximum line length (defaults to 80). You can change the value or set it nil. • When pel-c-fill-column user option is nil, c-mode buffers use the Emacs fill-column value like other major modes. • 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". • More user options are used for controlling C code templates created with PEL tempo skeletons. They are described in tempo skeleton section below.		
Find C header file	pel-cc-auto-newline: The values for those user each buffer and view thei None of the commands butters.	Whether automatic newline mo option variables can also be st r current settings using the con elow change PEL default; they	to all CC Mode related modes (like c-mode). The pel-c-file-finder-method determines how pel-open-at-point searches for header files.
PEL key prefixes:	PEL provides the following	g set of mode-specific key pre	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 $\mathfrak{B}\mathfrak{l}$ - \mathfrak{C} local PDF. If the prefix argument (like \mathfrak{C} - \mathfrak{u} or \mathfrak{M}) is used, then it opens the remote GitHub hosted raw PDF instead. If the \mathfrak{pel} -flip-help-pdf-arg user-option is set it's the other way around.
∑ Customize PEL C support	• <f12> <f2> • <f21> # <f2></f2></f21></f2></f12>	(pel-customize-pel &optional OTHER-WINDOW)	Customize PEL C support. • If OTHER-WINDOW is non-nil (use C-u), display in another window.
<u>∑ Customize</u> Emacs C support	<f12> <f3></f3></f12>	(pel-customize-library &optional OTHER-WINDOW)	,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,
<u>> Customize</u> Emacs C pre-processor support	<f12> #f3></f12>	(pel-customize-library &optional OTHER-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 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
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>	(c-guess-buffer &optional ACCUMULATE)	Guess the style on the whole current buffer, and install it. The style is given a name based on the file's absolute file name. If given a prefix argument (or if the optional argument ACCUMULATE is non-nil) then the previous guess is extended, otherwise a new guess is made from scratch.
<u>Guess style</u> in the region and install it.	<f12> <f4> g G</f4></f12>	(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>	(c-guess-install &optional STYLE-NAME)	Install the latest guessed style into the current buffer. This guessed style is a combination of 'c-guess-guessed-basic-offset', 'c-guess-guessed-offsets-alist' and 'c-offsets-alist'. The style is entered into CC Mode's style system by 'c-add-style'. Its name is either STYLE-NAME, or a name based on the absolute file name of the file if STYLE-NAME is nil.
<u>View Guessed style</u> as a set of Emacs Lisp statements	<f12> <f4> g ?</f4></f12>	(c-guess-view &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	The CC Mode controls the indactivated) and provide a better • CC Mode state displayed if • 12 is the CC mode program.	entation and bracket style which experience when editing C soon the mode line: \$C{} when gramming language name: C, Cyle: '*' for block command (ric flags: mode	re:

Use <f12> M-? to display the current state.

• 'w' for subword mode

Description	<u>Keystroke</u>	Function	Note
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). Solution This 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.	ctric keys don't reindent, the in	entation functions and electric keys indent according to syntactic context keys, when applicable. dentation functions indents every new line to the same level as the previous nonempty line, and becified by 'c-basic-offset'. The indentation style has no effect in this mode, nor any of the ok'.
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>	(c-toggle-hungry-state &optional ARG)	 Toggle hungry-delete-key feature. Affects 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: • ∑ Align • ∑ Indentation	<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. Identify 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>	(c-toggle-auto-newline &optional ARG)	Toggle auto-newline 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 electric indentation. When the auto-newline feature is enabled (indicated by "/la" on the mode line after the mode name) newlines are automatically inserted after special characters such as brace, comma, 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 CC mode derivative for the current c-mode buffer.
settings	The information includes the following: CC mode style currently active, along with a list of styles associated with current mode. Change it for the current buffer with C-c · or <f12> <f4> is the Emacs the c-default-style user option defines associations between major modes and the style to use. PEL provides the pel-c-backet-style that is used to set the style for c-mode. Use <f12> <f2> ff0> from a c-mode buffer to access the customization buffer to change it. Return key behaviour: RET (return key) mode. Change with pel-cc-change-newline-mode (<f12> <f4> RET). Whether return performs alignment. Change that with pel-toggle-indent-align (<f11> M-RET). State of electric C characters (toggle it on/off with c-toggle-electric-state (C-c C-1 or <f12> <f4> e): whether it is active or not, and when active what character(s) exhibit electric behaviour. if auto-newline on some characters (";' and some other based on style) is active. Toggle it with C-c C-a or <f12> <f4> M-RET. The fill column: the column where force line wrap is done when the auto-fill-mode is active. Toggle auto fill mode with <f11> RET. Tab width and whether hard tabs are used. These are set by the user options pel-c-tab-width and pel-c-use-tabs. In c-mode buffer use <f12> <f2> to open the appropriate customization buffer to change them. Remember that tab width does not identify the indentation. It controls the spacing used in some commands moving point to the next tab stop column. Indentation is controlled separately. See next line. Indentation width controlled by c-basic-offset normally set by pel-c-indent-width in PEL and whether syntactic indentation mode is active. Shows how is set and whether it was override by executing the pel-cc-set-indent-width command for this buffer (use <f12> <f4> TAB) for that command. The style currently used for indentation and backet positioning (they should have the same value). Emacs identifies several built-in styles but you can create your own. The example below shows "bsd" with is another name for the Al</f4></f12></f2></f12></f11></f4></f12></f4></f12></f11></f4></f12></f2></f12></f4></f12>		
	-UU-:F1 c_file.c	· · · · · ·	n-at-point (see sections below). WK Anzu Fly ² ElDoc Abv) 10:35am 1.97
Notice the name of the PEL user-options that set the significant feature controlling Emacs variables in the message	- RET mode : co - Electric characters : ac - Auto newline : on - fill column : 80 - Tab width : 8	, auto-filling: off. Set via:	: pel-c-tab-width(8) ==> tab-width(8) when c-mode buffer is opened.
coodgo	- Indentation chars : sp - Indent width : 4 - Syntactic indent : on - c-indentation-style : bs - PEL Bracket style : bs	Set via: d	: pel-c-use-tabs(nil) ==> indent-tabs-mode(nil) when c-mode buffer is opened. : pel-c-indent-width(4) ==> c-basic-offset(4) when c-mode buffer is opened.

Description	<u>Keystroke</u>	Function	<u>Note</u>		
C Code Help	There are several Emacs exter	nsion packages that can help v	vriting C code.		
Get man help about C	<f12> ? ?</f12>	(man MAN-ARGS)	Open a Man page inside an Emacs window. See <u>Yelp/Info</u> for more info about man.		
code See: <u>Nelp/Info</u>	• <f11> ? m • %-M</f11>		 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</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		
	<f11> SPC c ? e</f11>		point. It currently appears when typing a new function with its arguments inside the code.		
	Requires <u>c-eldoc</u> external package. Activated when pel-use-c-eldoc is set to t. The extra processing required may slow Emacs. This package could be improved into providing the information only on demand but a LSP-based system might be more performant anyway. I am currently looking at this to see if I can improve the performances and the feature set. c-eldoc uses the cpp command to preprocess the buffer content.				
Electric Keys			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)	Insert a "#".		
		dle it specially according to the a literal or a macro, nothing s	e variable 'c-electric-pound-behavior', which can only be nil or 'alignleft'. If a numeric ARG is pecial happens.		
0	• ()	(c-electric-paren ARG)	Insert a parenthesis.		
	If 'c-syntactic-indentation' a inside a literal.	and 'c-electric-flag' are both no	on-nil, the line is reindented unless a numeric ARG is supplied, or the parenthesis is inserted		
			may get added or removed; see the variable 'c-cleanup-list'. nil, some newline cleanups are done if appropriate; see the variable 'c-cleanup-list'.		
{}	• { }	(c-electric-brace ARG)	Insert a brace.		
	If 'c_electric_flag' is non-nil	the brace is not inside a literal	and a numeric ARG hasn't been supplied, the command performs several electric actions:		
	a) If the auto-newline fea settings in 'c-hanging-	ture is turned on (indicated by braces-alist'.	"/la" on the mode line) newlines are inserted before and after the brace as directed by the		
			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.		
			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-hanging-colons-alis	t'.	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'.		
:,	• ; ,	(c-electric-semi, ARG)	Insert a comma or semicolon.		
			a numeric ARG hasn't been supplied, the command performs several electric actions: d by "/la" on the mode line) a newline might be inserted. See the variable 'c-hanging-		
	semi&comma-criteria' for how newline insertion is determined. b) Any auto-newlines are indented. The original line is also reindented unless 'c-syntactic-indentation' is nil.				
	c) If auto-newline is turne cleanup-list'.	ed on, a comma following a bra	ace list or a semicolon following a defun might be cleaned up, depending on the settings of 'c-		
Electric pairs	It is also possible to control the insertion of character pairs by activating the electric-pair-mode in the buffer. • Type the first of a pair to insert this one and its matching character for (), [], {}, "" and ". • When the electric-pair-mode is active in a buffer the mode-line lighter set by the pel-electric-pair-lighter is shown. This defaults to £(1)				
Toggle electric-pair- mode in current	<f11> M-e</f11>	(electric-pair-local-mode & optional ARG)	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.		
buffer			 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 		
Lighter:= $E(I)$			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		
Insert New Line(s)	The behaviour of the RET key	depends on whether the CC N	region instead. Tode electric mode is active or not. When it is not active it simply inserts a new line. When it is		
(0)	active the point also moves to the proper indentation according to the syntactic context. The following commands can also be used. • With PEL the default behaviour can be selected by customization and modified dynamically for the current buffer with the pel-cc-change-newline-mode				
	The pel-cc-newline comman	nd also aligns comments and a	haviour control section above. assignment in the code block if the pel-modes-activating-align-on-return user option list		
I			t 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	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): 		
selected return mode.			c-context-line-break (PEL default for RET) newline (Emacs default for RET)		
With PEL, modify behaviour with <f12></f12>			electric-indent-just-newline If 'pel-newline-does-align' is t, then perform the text alignment done by the function 'align'.		
M-RET.		ak) : Do a line break suitable to	o the context.		
	which case the new lir	ne is indented as the previous r			
	The end of the cpp dir	ective doesn't count as inside			
	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 it is also inside a preprocessor directive. wline, and move to left margin of the new line if it's blank.		
See also: •	With ARG, insert that it	many newlines.	s marked with the text-property 'hard'.		
<u>Justification</u>	If 'electric-indent-mod		final new line that it adds, and reindents the preceding line.		
	Calls 'auto-fill-function	n' if the current column number	r is greater than the value of 'fill-column' and ARG is nil.		
	Use: (electric-indent-just • With ARG, insert that i		ewline, without any auto-indentation.		
Insert an indented line below unbroken	• 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.		
current line See also:	Cabe RET	,	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.		
<u>∑ Indentation</u>			If <i>pel-newline-does-align</i> 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.</f11>		
			• Solution like the list in the pel-modes-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.		

<u>Description</u>	<u>Keystroke</u>	Function	<u>Note</u>
Insert a newline	С-ј	(electric-newline-and- maybe-indent)	Insert a newline. • If 'electric-indent-mode' is enabled, that's that, but if it is *disabled* then:
	 when disable: additionally indent according to major mode. Indentation is done using the value of 'indent-line-function': In programming language modes, this is the same as TAB. In some text modes, where TAB inserts a tab, this command indents to the column specified by the function 'current-left-margin'. 		
Open New Line in	• In some text modes, w	(c-context-open-line)	Insert a line break suitable to the context and leave point before it.
Context See also:	• This is the 'c-context-line-break' equivalent to 'open-line', which is normally bound to C-o. See 'c-context-line-break' for the details.		
• <u>▼ Whitespace</u>	Normally C-o is bound to d	open-line. PEL rebinds it to c-o	context-open-line for the CC modes. xt 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>	2 more characters have electr • C supports 2 types of comments • '*': Block Comments • '/': Line Comments (: /* comment	
Comment characters, wrapping long comment lines	/	(c-electric-slash ARG)	Insert a slash character. 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'. Indent the line as a comment, if: 1. The slash is second of a "/" line oriented comment introducing token and we are on a comment-only-line, or 2. The slash is part of a "*/" token that closes a block oriented comment. 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.
See also: **See Tilling/Justification** **Filling/Justification** **Tilling/Justification** **Till	*	(c-electric-star ARG)	Insert a star character. If 'c-electric-flag' and 'c-syntactic-indentation' are both non-nil, and the star is the second character of a C style comment starter on a comment-only-line, indent the line as a comment. If a numeric ARG is supplied, point is inside a literal, or 'c-syntactic-indentation' is nil, this indentation is inhibited. With this key being electric it becomes easy to type the following two styles of multi-line block comment: /* Two star ** continuation ** prefix for ** multi-line ** C comment. */ /* Single star * prefix for * multi-line * C comment. */ */ When typing the '*' at the beginning of the line, it indents automatically. If another '*' is typed, indentation is set to allow a two-star continuation, otherwise it is placed for a single star continuation. When auto-fill-mode is active, when you type a comment that would be longer than the line, the line is wrapped and the comment continuation string used is automatically inserted. (toggle it with <f11> RET)</f11>
Comment/un- comment ★★ See also: <u>Somments</u>	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: • Somment 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 C-u prefix arg, uncomment each line in region BEG END.
See also: Comments	 Numeric prefix ARG means use ARG comment characters. If ARG is negative, delete that many comment characters instead. The strings used as comment starts are built from 'comment-start' and 'comment-padding'; the strings used as comment ends are built from 'comment end' and 'comment-padding'. By default, the 'comment-start' markers are inserted at the current indentation of the region, and comments are terminated on each line (even for syntaxes in which newline does not end the comment and blank lines do not get comments). This can be changed with 'comment-style'. If you try this when no region is marked and the /* */ style comments is active, the comment ends on the next space, which is probably not what you want. The command comment-dwim works better and pel-c-comment-dwim (above) even better. 		
Fill current paragraph	• M-q	(c-fill-paragraph &optional	Like <f11> t f p but handles // and /* */ style comments.</f11>
See also: Filling/Justification	• <f12> F • <m-f12> F</m-f12></f12>	ARG)	 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.
Toggle subword-mode See also: Text Modes	• <f11> t m b • <f12> M-b • <m-f12> M-b</m-f12></f12></f11>	(subword-mode &optional ARG)	Toggle subword-mode: a minor mode that treats sections of <u>camelCase</u> and <u>PascalCase</u> as distinct words. • With a prefix argument ARG, enable Subword mode if ARG is positive, and disable it otherwise.
Hide/Show comments 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 hide-comnt.el package (see Comments). PEL activates it when the pel-use-hide-comnt user option is t.

Description	<u>Keystroke</u>	Function	<u>Note</u>	
Hungry Deletion of Whitespace	PEL provides the converted in modes compatible with the simple DEL and C When the Hungry Delete The Hungry Mode also as mode is activated (and the In modes derived from C other modes. PEL provi	nient keys with the <f11> preme CC Mode (e.g. for C, C++, E-d, to perform hungry deletion Mode is on, the mode-line disperitivates the key prefixes below that can only be done in modes C Mode you can also activate des the <f12> M-DEL key for</f12></f11>	Ingry whitespace deletion" that can also be used in every mode. fix keys for those 2 commands, available in all modes. D. Java, Pike, etc) it is also possible to activate the Hungry Delete Mode to modify the behaviour is. That's not currently supported in other modes. plays a 'h' to the right of the '//' indication of electric mode. If that start with C-c . They are listed but remember they are only available once the Hungry state that are CC Mode compatible). The hungry state to make standard delete commands delete hungrily, but that does not work for or those modes (like C). The current buffer with c-toggle-hungry-state (<f12> M-DEL</f12>).	
Delete preceding char or all preceding whitespace. See also: <u>See Also:</u> <u>Cut & Paste</u>	• C-c DEL • C-c ⊠ • C-c C-⊠ • C-c C-backspace> • 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- (I), (C-backspace) and C-DEL are not available, they are mapped to the non-control key so attempting to type them end up invoking the command anyway because the first key bindings are recognized. With PEL, the <f11> (I) (II) binding is always available, in all modes. The other keys are only available in modes derived from the CC Mode. This prevents conflicts with other modes that may use the popular C-c bindings.</f11>	
Delete next char or all following whitespace. See also:	• C-c C-d • C-c 🖄 • C-c C-\overline{\times} • C-c <c-delete> • <f11> \overline{\times}</f11></c-delete>	(c-hungry-delete-forward)	Delete the following character or all following whitespace up to the next non-whitespace character. In terminal mode, even though C-\overline{\text{S}} 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> \overline{\text{S}} 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>	
<u>Indentation</u>	behaviour control section above You can also explicitly request • The first set of commands p	ve. indentation using the commar perform syntactic indentations		
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: • <u>▼ Indentation</u>	 Behaviour depends on syntactic-indentation mode (enabled by default but can be toggled on/off with the <f12> M-i key):</f12> With syntactic-indentation on (the default): In Transient Mark mode, when the region is active, reindent the region. Otherwise, with a prefix argument, rigidly reindent the expression starting on the current line. Otherwise reindent just the current line. 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> With syntactic-indentation off: <tab> always indent current line by one level</tab> C-u - <tab> or M <tab> always un-indent current line by one level.</tab></tab> Indenting marked region is done without syntax knowledge and at the same level as previous line. If you want to indent rigidly you can use: 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>			
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 (C - u) 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>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:	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 tabwidth 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></tab> <f11> <tab></tab></f11> <tab></tab> <tab></tab> <tab></tab> <tab></tab> <tab></tab> <tab></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 indentation changes. If no region is marked, it operates on current line(s) identified by the numeric 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. 			
	indent-rigidly Indent all lines: If called interactively w <right>, <s-left activ="" are="" both="" capabilities="" commands="" controlled="" of="" s-<right="" the="" these=""> indent- <right> indent- <right> indent- * Cleft> indent- * Cleft> indent- * Cleft> indent- * Cleft> indent- * Cleft> indent- * Other indent- * United the standard of the st</right></right></s-left></right>	starting in the region.	to the next tab-stop position, which is controlled by the tab-width user option. e of pel-c-tab-width .	

Description	<u>Keystroke</u>	Function	<u>Note</u>
Indent line(s) rigidly	• <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 pel-c-indent-width . • Works with point anywhere on the line.
See also: • <u>∑ Indentation</u>	If a region is marked, the full in all affected lines. Use C Handles presence of hard to If indent-tabs-mode is not to If indent-tabs-mode is not to If indent-tabs-mode is not If it is	pecify more than one indentation to does not deactivate it to - g to de-activate the region. abs: on-nil the indentation is created I, any hard tab in the indentatio	on level. It defaults to 1. If a negative number is specified, 'pel-unindent-lines' is used. allow repeated execution of the command. It also modifies the region to include all characters 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 See also:	• <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-indent-width. Works with point is anywhere on the line.
• <u>∑ Indentation</u>	If a region was marked, the characters in all affected lin Handles presence of hard to If indent-tabs-mode is no	nction does not deactivate it to function does not deactivate it es. Use C-g to de-activate abs: on-nil the indentation is created and hard tab in the indentation.	o allow repeated execution of the command. to allow repeated execution of the command. It also modifies the region to include all 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-
Open file at point	In a c-mode buffer the comwould be stored. The searc Modify pel-c-file-finder-ini Note that when using the l	mand is specialized to be more the method is controlled by the pa- tool-name for the current edi do completion mode, it is poss	e taken at point (the cursor location). e useful for C programming and has the extra capability of searching files where header files pel-c-file-finder-method and the pel-c-file-finder-ini-tool-name user-options. ting session with the pel-cc-set-file-finder-ini-tool-name command. ible to instruct Ido to use a file name at point as the basis for the file name to open. This Ido er-option. With PEL you can control it globally or locally with <f11> f M</f11>
Set name of Tool-Chain specific include path • (when the pel-ini-file search method is used)	<f12> <f4> C-^</f4></f12>	(pel-cc-set-file-finder-ini- tool-name &optional TOOL- NAME)	Change the value of pel-c-file-finder-ini-tool-name for C file for the current editing session. It is used to add directories to the search path used by pel-open-at-point . The directory list is taken from the key with the name specified by <i>TTT-c-path</i> inside the pel.ini file where <i>TTT</i> is the tool chain name. pel-open-at-point uses this only when the file search method identified by the pel-c-file-finder-method user-option is pel-ini-file. When used it searches pel.ini file in the directory tree identified by the presence of the file identified by pel-project-root-identifiers user-option
Open file or web-page whose name is at point	• C-^ • <f11> f . • <m-f11> M-f M • 6y</m-f11></f11>	(pel-open-at-point &optional N)	Open the file, library or the URL, named at point, with potential line & column #s. The <u>6v</u> key-chord is available if pel-use-key-chord is non-nil. Command prefixes are supported with the key-chord. See <u>Ney-Chords</u> . Type <f12> <f4>? to show the file search method currently used.</f4></f12>
See also: •	Use this command to open the header file identified by the #include statement. The various search methods (see below) support searching inside the file's directory tree or search using the directories identified by an INCLUDE environment variable or from the directories identified explicitly. This command extracts the file name to search from text at point. The file name is either surrounded by white space characters or the delimiters listed below. • If embedded space(s) are allowed in the filename, then point must be located at the first of the 2 delimiter characters. • These delimiter character can be any of the following: "`' ()[]{}<> ''"" 「J () ⟨⟩ ⟨⟨⟩ (] (] (] () ⟨⟨⟩ ⟨⟨⟩ (] (] (] () ⟨⟨⟩ () () () () () ⟨⟨⟩ () () () () () () () () () () () () ()		
	extracted. The file must contain a [file-finder] section. • Inside that section it must contain the project-path key which holds a list of directories to search. • If the PEL_CC_FIND_TOOLCHAIN environment variable exists it defines the name of a toolchain. If the pel-c-file-finder-ini-tool-name user-option overrides the value of the environment variable. These identify the name of a TTT-c-path key (where TTT is the tool chain name) which holds the names of tool-chain specific directories to search in addition to the directories defined by the project-path key. • The paths identified in the tow lists may use environment variables inside the path strings. Use the \$VARNAME format to identify them. • You can modify this tool chain name anytime during an editing session by typing <f12> <f4> C-^ and specifying another name. With several TTT-c-path keys inside the pel.ini file, you can adjust the include path dynamically for various tool chains. • environment variable name: the name of an environment variable (like INCLUDE) that holds a list of directory names to search files in. • explicit lists: two lists of directory names: one list holds the project directory names, the other hold the tool and library directory names. The lists may identify directory names indirectly via environment variables. The \$VARNAME format must be used.</f4></f12>		
• Select prompt method • ∑ Completion/ Input	 When several file names are found, the command lists them and prompts using the method selected by pel-prompt-read-method user-option. The default is a very primitive function implemented by PEL. You can select a more powerful ivy prompting instead. With ivy selected PEL will automatically set pel-use-ivy to made 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. This allows you to detect potential duplication in header file names in large include paths. 		
Select target window where file is opened	 It Prompts for incomplete file names, allowing editing the find file (with completion), search for libraries files (type 1) according to current file type. Select target window: Without argument: If file 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. If no window holds that file, select the target window based on 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. With numeric argument N: 		
N>20 : open the directory	 N = 0: use the 'other' (N = 1, 3, 7or above (e) if 1 window: if 3 or more window N is: 8: up, 2: down, 4 N is 9: open the file open a direct N is 10: open the URL 	n the directory instead of the fithe next) window. koluding 8, 9 and 10): select the split that window and use t s: use the current window. Eleft, 5:current, 6:right. in the system's browser (with	

Description	<u>Keystroke</u>	Function	<u>Note</u>
Tempo skeletons for C	PEL creates key bindings to with the same key bindings	invoke the skeletons in the su for equivalent concepts (such	as file header block) as much as possible.
See also: •	PEL creates key bindings to invoke the skeletons in the supported major modes, using the same key prefix sequence for each mode: <f12> <f12>, with the same key bindings for equivalent concepts (such as file header block) as much as possible. ■ Several aspects of the PEL Emacs Lisp Source Code Style is controlled by the user options inside the pel-c-code-style group. This group can be edited with <f12> <f1>< f1> code buffer and include the following options: ■ pel-c-skel-module-header-block-style: allows selecting a user-define module-header comment block. ■ pel-c-skel-norment-with-2-star: ■ pel-c-skel-insent-file-timestamp: ■ pel-c-skel-insent-file-timestamp: ■ pel-c-skel-doc-markup: ■ identifies the documentation markup supported by the templates. Currently 'none' and 'Doxygen' are available. ■ pel-c-skel-insent-function-section-titles: ■ pel-c-skel-insent-function-sections: ■ pel-c-skel-insent-function-sections: ■ pel-c-skel-insent-function-sections: ■ pel-c-skel-function-define-style: ■ pel-c-skel-function-define-style: ■ pel-c-skel-function-define-style: ■ pel-c-skel-function-name-on-first-column: identifies whether return type is located on the same line as function name or just above. ■ pel-c-skel-function-name-on-first-column: identifies whether return type is located on the same line as function name or just above. ■ pel-c-skel-with-license: ■ pel-c-skel-function-name-on-first-column: identifies whether return type is located on the same line as function name or just above. ■ specify which type of include guard is inserted in header files. The available choices are: ■ pel-c-use-include-guards: ■ pel-c-use-include-guards: ■ pel-c-skel-function-on-first-column: identifies whether return type is located on the same line as function name or just above. ■</f1></f12></f12></f12>		
<u>▼ Customize</u> PEL C	with the standard tempo-me <f12> <f12> <f2></f2></f12></f12>	ode keys C-c M-f and C-c (pel-customize-pel &optional OTHER-WINDOW)	M-b or some other keys like C-c . and C-c ,. Customize PEL C skeleton layout. If OTHER-WINDOW is non-nil (use C-u), 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. • Prompts for the file purpose. • For header files, include guard is inserted if requested by customization. • 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. • See examples of generated outputs located in example/templates/c/files repo directory. • Access the customization buffer by typing: <f12> <f12> <f2></f2></f12></f12>
Insert #define	<f12> <f12> d</f12></f12>	(pel-c-define)	Insert a C pre-processor #define statement. 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. • 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 point left right before the period. 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 #include "" 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 point left right before the period. The next tempo mark is placed at the end of the line (so C-c · move point there).
Insert a function definition with comment block	<f12> <f12> f</f12></f12>	(pel-c-function)	Insert a C function definition code and comment template. • The command prompts for the function name and its purpose. • You can hit return both prompts to specify no text; in that case a tempo skeleton marker is left at the location where the text must be inserted and point is left at the first one. • If you enter a function name, it must be a valid C function name (as far as the syntax is concerned). However leading and trailing whitespace is 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 purpose maintain separate histories. Use M-p and M-n to navigate in the histories at the prompt. You can also use the <up>and <down> keys. • The style of the code inserted is controlled by the user options 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 prompt. See some examples in the PEL manual.</down></up>
Toggle pel-tempo- mode	<f12> <f12> SPC</f12></f12>	(pel-tempo-mode &optional ARG)	Toggle PEL tempo mode on/off. PEL tempo mode activates C-c . and C-c , as well as to C-c C and C-c C-, key bindings to navigate across tempo mark hot-spots. When pel-tempo-mode is active the pel-tempo-mode lighter (‡) is shown on the status bar. The second set are only available when Emacs runs in graphics mode. When a skeleton is inserted via the execution of one of the pel-rst commands, the pel-tempo-mode is automatically activated.
Jump to next tempo mark	• C-c M-f • C-c . • C-c C	(tempo-forward-mark)	Jump to the next mark in 'tempo-back-mark-list': the location where code must be updated inside the inserted skeleton. • These key key bindings are only available when pel-tempo-mode is active.
Jump to previous tempo mark	• C-c M-b • C-c , • C-c C-,	(tempo-backward-mark)	Jump to the previous mark in 'tempo-back-mark-list': the location where code must be updated inside the inserted skeleton. • These key binding are only available when pel-tempo-mode is active.

Description	<u>Keystroke</u>	Function	<u>Note</u>
Tempo Template Tag	<f12> <f12> <f12></f12></f12></f12>	(tempo-complete-tag	Look for a tag and expand it.
Insertion		&optional SILENT)	Instead of using the <f12> <f12> key bindings above, you can type the template name (shown in the title column like "if", "case", etc) completely or partially and then hit <f12> <f12> <f12>. A completion buffer opens up if the template name is incomplete (or empty in</f12></f12></f12></f12></f12>
			which case the buffer lists all available template names). Select the template name and hit RET. Emacs expands the template.
			 All the tags in the tag lists in 'tempo-local-tags' (this includes 'tempo-tags') are searched for a match for the text before the point. The way the string to match for is determined can be altered with the variable 'tempo-match-finder'. If 'tempo-match-finder' returns nil, then the
			 results are the same as no match at all. If a single match is found, the corresponding template is expanded in place of the matching string. If a partial completion or no match at all is found, and SILENT is non-nil, the function
			will give a signal. If a partial completion is found and 'tempo-show-completion-buffer' is non- nil, a buffer containing possible completions is displayed.
			➤ Since only one template is available in emac-lisp-mode, the usefulness of this command is limited here.
Inserting code	Extra text insertion can be don	ne with the following command	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. • No argument is equivalent to zero: just insert '()' and leave point between. • PEL makes parens-require-spaces buffer local and set it to nil in C mode buffers, allowing the use of this command to insert the argument parentheses following a function (and without placing a space between the function name and the opening parenthesis. • If region is active, insert enclosing characters at region boundaries. • This command assumes point is not in a string or comment.
Marking	Emacs provides 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	С-м-h	(c-mark-function)	Mark complete function.
function body See also: Marking			 Put mark at end of the current top-level declaration or macro, point at beginning. If point is not inside any then the closest following one is chosen. Each successive call of this command extends the marked region by one function. A mark is left where the command started, unless the region is already active (in Transient Mark mode).
			Mark mode). As opposed to C-M-a and C-M-e, this function does not require the declaration to contain a brace block.
Getting Syntactic Information	Use the following commands	to extract syntactic information	from the source code.
Display name of current function	• C-c C-z • <f12> f • <m-f12> f</m-f12></f12>	(c-display-defun-name &optional ARG)	Display the name of the current CC mode defun and the position in it. • With a prefix arg, push the name onto the kill ring too.
Search Support			case is often used. Using superword-mode helps searching. change this use the <f11> t <f2> to access the customize buffer.</f2></f11>
Toggle superword- mode See also: ■ ∑ Text Modes ■ ∑ Search/Replace	• <f11> t m p • <f12> M-p</f12></f11>	(superword-mode &optional ARG)	 Toggle superword-mode: a minor mode that treats <u>snake case</u> as one word. In C '_' are treated as part of words. With a prefix argument ARG, enable superword mode if ARG is positive, and disable it otherwise.
Highlighting blocks	show-paren-mode, which h	ighlights the parens that match	seful modes to highlight blocks of (), {}, and []. the one before or after point. the highlighted with the same colour.
Toggle show-paren mode on/off	• <f12> M-9 • <m-f12> M-9</m-f12></f12>	(show-paren-mode &optional ARG)	Toggle visualization of matching parens (Show Paren mode). • With a prefix argument ARG, enable Show Paren mode if ARG is positive, and disable it
See also: <u>Nighlight</u>	• <f11> h (</f11>		 otherwise. Show Paren mode is a global minor mode. When enabled, any matching parenthesis is highlighted in 'show-paren-style' after 'show-paren-delay' seconds of Emacs idle time.
Enable/Disable coloured highlight of	• <f12> M-r • <m-f12> M-r</m-f12></f12>	(rainbow-delimiters-mode &optional ARG)	Highlight nested parentheses, brackets, and braces with different colours according to their depth.
nested blocks (),{},[] See also: <u>∑ Highlight</u>	• <f11> h R</f11>	, ,	• Customize the depth and colours with M-x customize-group rainbow-delimiters Requires: rainbow-delimiters.el
			PEL activates this when the pel-use-rainbow-delimiters user option is set to t .
Navigation in C	This current list below describ	e the specialized commands o	nly. See the others inside <u>Navigation</u>
By definitions			ef for more information to activate the various engines that support cross referencing for C code.
Find definition of identifier at point	M	(xref-find-definitions IDENTIFIER)	Grab symbol at point and move cursor to its definition. If there are more than one match, prompt in the *xref* buffer. To search for a symbol entered manually, type C-u M
See also: <u>∑ Xref</u> Go back to where M		(was non monton stock)	With dumb-jump this performs a search using ag, ripgrep or git grep if available. Den besides where Manual leaf involved.
was last issued	М-,	(xref-pop-marker-stack)	 Pop back to where M was last invoked. Marker depth is controlled by the xref-marker-ring-length user option.
By functionsBy structures	Jump over comments.	function definition blocks or struction definition blocks or struction defined by the structure of the struc	ucture definition blocks. er closing brace and show-paren-mode is on, the matching parentheses are highlighted.
Forward to start of next top level function or struct	<f6> <down></down></f6>	(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.
	<f12> <down></down></f12>		 Move back to previous position with M-` or <f6><f6>.</f6></f6> Shift marking is available. With <f6> and <f12> hit Shift after function key, before cursor key.</f12></f6> This command complements what end-of-defun does. It moves forward but not to the end of the function definition (like end-of-defun) but to the beginning of the function definition, which is often what users of other editors expect.
Forward to end of current top-level function or struct.	С-М-е	(c-end-of-defun &optional ARG)	Move forward to the end of a top level declaration. • With argument, do it that many times. Negative argument -N means move back to Nth preceding end.
	• C-M- <end> • <f6> <right></right></f6></end>	(end-of-defun &optional ARG)	Move forward to the end of next function or type definition. With argument, do it that many times. Negative argument -N means move back to Nth preceding end of defun. Shift marking is available. With <f6> and <f12> hit Shift after function key, before cursor key.</f12></f6>
	<f12> <right></right></f12>		This command moves to the end of the next top-level function. It skips nested functions.

<u>Description</u>	<u>Keystroke</u>	Function	<u>Note</u>
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.
Ulay.	• C-M- <home> • <f6> <up></up></f6></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.
	<f12> <up></up></f12>		→ 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	<f6> <left></left></f6>	(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.
	<f12> <left></left></f12>		 Move back to previous position with M⁻ˆ or <f6><f6>.</f6></f6> Shift marking is available. With <f6> and <f12> hit Shift after function key, before cursor key.</f12></f6> In some cases it fails to detect the end of the previous block and fails.
By blocks	Move across C statements	and C scope blocks, or any gro	oup of (), [], {} or <> blocks.
By List element	Move to the end or the be	eginning of a block	
Backward block/list	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
See also: Navigation		,	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. ⑤ 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->settings->keyboard->shortcuts to prevent it from using that key sequence.</left></left></left></left></left>
Forward block/list	C-M-n	(forward-list &optional	Move forward across one balanced group of parentheses.
See also: Navigation		ARG)	 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>	(forward-sexp &optional ARG)	Move forward across one balanced expression (sexp). • With ARG, do it that many times. Negative arg -N means move backward across N balanced expressions. This command assumes point is not in a string or comment. • C-M-f : Shift marking is available in graphics mode, not in terminal mode. • C-M- <right> : Shift marking works with this command. • With PEL: if you want to use Esc C-<right> binding you must ensure that 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->settings->keyboard->shortcuts to prevent it from using that key sequence.</right></right></right></right></right>
in/out of blocks	Move in or out of C scope b	 	, , , ,
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> im 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	C-M-]	(up-list &optional ARG ESCAPE-STRINGS NO-	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
See also: Navigation		SYNTAX-CROSSING)	 In scorinfiand will also work off 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.
Down/inside sexp/ block	• C-M-d • C-M- <down></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
See also:	• C-[C-d		language mode. • With ARG, do this that many times. A negative argument means move backward but still go
• <u>▼ Navigation</u>	• Esc C-d • Esc C- <down></down>		down a level. This command assumes point is not in a string or comment. Mith PEL: if you want to use Esc C- <down> binding you must ensure that pelwindmove-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)	Go to the beginning of the innermost statement. • With prefix arg, go back N - 1 statements. • 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). • If within or next to a comment or multiline string, move by sentences instead of statements.
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.

Description Keystroke **Function** Emacs supports navigation through C preprocessor conditional statements, allow expansion of preprocessor macros, hiding pre-processor statements that would not be executed with the Hide-ifdef mode. There are also external packages that provide extra support. All commands provided by Emacs and C Preprocessor external packages are listed below. They can be used for editing C and C++ source code. PEL provides a key hydra to help navigate trough preprocessor directives and to hide/show code areas based on preprocessor logic and defined variables.

* The key sequences that start with <f12> <f7> open the pel-\(\subseteq\)c-preproc Hydra allowing further hydra keys to be typed without any prefix. Requires the hydra external package APEL activates when the pel-use-hydra user option is set to t. ж -UUU:**--F1 a c file.c All (4,0) (C/*la Ifdef WK Fly 2 Anzu Abbrev) -----C preprocessor: Move to # Vars Other Open the C preprocessor <u>hydra</u> with <f12> <f7> #: toggle mode followed by on of the d: define hvdra kevs: C-p: begin R: toggle RO C-n: end H: hide C-u: up S: show D: Save list h: hide block C: Clear all s: show block Navigate across pre-The following commands move point across the #if, #else, #elif and #endif C pre-processor conditional statements. 🕵 Customizations: 'c-macro-preprocessor' specifies the pre-processor command used by Emacs. The default depends on the operating system. conditionals • <f12> # p Move to previous (pel-pp-prev-directive) Move point to previous preprocessor directive. # <f12> <f7> p preprocessor directive (pel-pp-next-directive) Move to next • <f12> # n Move point to next preprocessor directive. preprocessor * <f12> <f7> n directive (c-up-conditional COUNT) Move back to the containing preprocessor conditional, leaving mark behind. Move up in the pre-• C-c C-u A prefix argument acts as a repeat count. With a negative argument, move forward to the processor conditional * <f12> <f7> C-u end of the containing preprocessor conditional.
"#elif" is treated like "#else" followed by "#if", so the function stops at them when going backward, but not when going forward. Move to the previous • C-c C-p (c-backward-conditional Move back across a preprocessor conditional, leaving mark behind. COUNT &optional TARGET-A prefix argument acts as a repeat count. * <f12> <f7> C-p pre-processor With a negative argument, move forward across a preprocessor conditional. conditional block DEPTH WITH-ELSE) Move to the next pre--forward-conditional Move forward across a preprocessor conditional, leaving mark behind. * <f12> <f7> C-n COUNT &optional TARGETprocessor conditional · 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. DEPTH WITH-ELSE) • "#elif" is treated like "#else" followed by "#if", except that the nesting level isn't changed when tracking subconditionals. (c-macro-expand START Expand C macros in the region, using the C preprocessor. **Expand Pre-**• C-c C-e FND SUBST) 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 <f12> # # • <M-12> # # preprocessor (e.g. '-DDEBUG -I ./include'), otherwise use 'c-macro-cppflags'. (**c-backslash-region** FROM TO DELETE-FLAG &optional Insert, align, or delete end-of-line backslashes on the lines in the region Insert/align or delete C-c C-\ end-of-line backslash With no argument, inserts backslashes and aligns existing backslashes LINE-MODE) With an argument, deletes the backslashes. · This function does not modify blank lines at the start of the region. If the region ends at the start of a line and the macro doesn't continue below it, the backslash (if any) at the end of the previous line is deleted. · You can put the region around an entire macro definition and use this command to conveniently insert and align the necessary backslashes 📰 Customizations: The backslash alignment is done according to: 'c-backslash-column', 'c-backslash-max-column' and 'c-auto-align-backslashes', Show state • <f12> # ? (pel-pp-show-state) Show state of C preprocessor control modes on the echo area. preprocessor modes * <f12> <f7> ? If too long, see the information in the *Messages* buffer. **Hide-ifdef Mode** Hide-ifdef mode suppresses (hides or shadows) the display of code that the C preprocessor wouldn't pass through. It supports complete C/C++ expression and precedence
 It scans for new #define symbols and macros. hide/show code controlled by C-• It hides blocks of code that would not be include in the expanded file according to the state of pre-processor symbols that are maintained inside the preprocessor Hide-ifdef environment: the hide-ifdef-env association list Emacs variable (use <f1> v to see the content of Emacs variables). See \(\subseteq \text{Help/Info.} \) • When hiding code, the hidden code is marked by ellipses (...). Le cautious 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. · You can make your buffer read-only while hide-ifdef-hiding by setting hide-ifdef-read-only user-option to a non-nil value. Access it hide-ifdef customization group with <f12> # <f3> You can toggle this variable with hide-ifdef-toggle-read-only (with C-c @ C-q) or with <f12> # r or <f12> <f7> R. Mith PEL, the commands are reachable via the <f12> prefix keys can also be reached via the <M-f12> and the <f11> SPC c prefix keys. ★ The key sequences that start with <f12> <f7> open the pel-∑c-preproc Hydra allowing further hydra keys to be typed without any prefix. Requires the <u>hydra</u> external package 2 PEL activates when the **pel-use-hydra** user option is set to **t**. Several customize user option variables affect how the hiding is done: to change, execute: M-x customize-group hide-ifdef or type <f12> # <f3> 'hide-ifdef-env' An association list of defined symbols for the current project. The list holds the following forms: (SYMBOL) is used when the SYMBOL is defined (but without explicit value)
 (SYMBOL . VALUE) when the symbol is defined with an explicit value. · 'hide-ifdef-define-alist' An association list of pre-defined symbol lists. Use 'hide-ifdef-set-define-alist' to save the current 'hide-ifdef-env' and 'hide-ifdef-use-define-alist' to set the current 'hide-ifdef-env' from one of the lists in 'hide-ifdef-define-alist'. Set to non-nil to not show #if. #ifdef, #ifndef, #else, and #endif lines when hiding. · 'hide-ifdef-initially' Indicates whether 'hide-ifdefs' should be called when Hide-Ifdef mode is activated. · 'hide-ifdef-read-only' Set to non-nil if you want to make buffers read only while hiding. After 'show-ifdefs', read-only status is restored to previous value. Toggle the Hide-Ifdef • <f12> M-# (hide-ifdef-mode &optional 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. mode: • <M-f12> M-# ARG) hide/show code Hide-Ifdef mode is a buffer-local minor mode for use with C and C-like major mode * <f12> <f7> # When enabled, code within #ifdef constructs that the C preprocessor would eliminate may be suppressed by C preprocessor • <f11> SPC c M-# hidden from view.

<u>Description</u>	<u>Keystroke</u>	Function	<u>Note</u>
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 #ifndef VAR 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 #ifdef VAR 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 hide-ifdev-env to a list with the specified NAME for later re-use. The value is saved inside the hide-ifdef-define-alist variable. The list is not saved to disk. You may want to pre-create the value for a given project and store 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)	• If no region active, find the current #ifdefs and evaluate the result. • Currently it supports only math calculations; strings or argumented macros can not be expanded.
Rendering markup embedded in comments	The following commands are used to create images from specific markup code embedded inside C source code comments. This can be useful when using these markup languages to describe UML diagrams or finite-state machines for example. You can also use Graphviz, see M. Graphviz Dot		
Preview UML diagram from plantUML source in current plantUML region of commented source code See also: M PlantUML	<f12> u</f12>	(pel-render-commented-plantuml PREFIX &optional POS)	Render the PlantUML markup embedded in current mode comment. • Use region if identified otherwise use PlantUML block at point. • Uses prefix (as PREFIX) to choose where to display it: • 4 (when prefixing the command with C-u) -> new window • 16 (when prefixing the command with C-u C-u) -> new frame. • else -> new buffer • This can be used inside buffer using any major mode, when PlantUML markup is embedded inside source code comment. • 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.

Emacs & C - References

Document	Notes
GNU emacs - CC Mode Manual	
GNU Emacs Manual - Styles	
Emacs BSD/Allman Style with 4 Space Tabs?	
Emacs: Linux Kernel Style but with Allman/BSD Style Braces?	
Emacs Wiki - Indenting C	
Indent preprocessor directives as C code in emacs	Does not fully address the way I want to have multi-indentations for pre-processor
elisp code - ppindent.el	Implements pre-processor indentation with the # always in the first column. Not yet exactly what I want.
company-mode ; Modular in-buffer completion framework for Emacs	