Programming Language Support — C

Description	<u>Keystroke</u>	Function	Note
Editing C Files	Emacs supports C natively via	the built-in <u>c-mode</u> which exte	ends the <u>CC Mode</u> that support the <u>curly-bracket programming languages</u> like <u>C</u> .
CC Mode File extensions			use-c or pel-use-bison user-option variables is set.
• C	I		yacc, header files: .h, .i Add more in pel-auto-mode-alist user option.
 bison-mode ∑ Speedbar 			les are associated with bison-mode , the .l and .lex with flex-mode and .jison with jison-mode ecognized by speedbar, otherwise only the main ones are recognized.
See: Info on C			the CC Mode are customizable with PEL user option variables.
		Simplifies configuration for ed	
		up: pel-pkg-for-c (access wit	h <f12> <f2>): s used for indentation. Defaults to 3.</f2></f12>
indentation	• pel-c-tab-width: The	width of a tab used for c-mode	e files. Defaults to 3.
	<tab> will indent to</tab>	a column that is a multiple of	e an indentation of 3 and tab width of 8: M-i will move point to columns that are multiple of 8 3. PEL stores this value inside the tab-width variable for c-mode buffers. e width of your needed indentation level. This way you can use commands that use either to
using tabs	control the indentati	on level.	
bracket style	C code style sub-group		tation or not: t: tabs are used, nil: only spaces are used. Default: nil.
			ccurs: maximum line length (defaults to 80). You can change the value or set it nil. e buffers use the Emacs fill-column value like other major modes.
	pel-c-backet-style	The bracket/indentation styl	e supported by the electric keys. You can select one of the values supported by Emacs or
	1	er' with some Emacs Lisp code re used for controlling C code	e. Default to "linux". templates created with PEL tempo skeletons. Described in tempo skeleton section below.
			o all CC Mode related modes (like c-mode). ode is active on all CC Mode (including c-mode).
	The values for those user	option variables can also be s	tored inside directory local files and even as file local variables. You can also modify them for
	None of the commands b	elow change PEL default; they	nmands listed in the following set of rows. See <u>File/Directory Variables</u> for more info. change the value for the current buffer only. er user-option is set to t, PEL downloads, installs and provides tree-sitter support via:
• <u>Tree-Sitter</u>			nese are installed automatically by PEL when pel-use-tree-sitter is on.
(@GitHub)	Other Requirements:		
		nodule loading, and built with t :arted with Tree-Sitter	tree-sitter support. tree-sitter library must be installed separately.
	 Emacs must find the tr 	ee-sitter language dynamic libr	rary files that have a name similar to 'libtree-sitter-python.so' (for Linux) or .dylib (for macOS).
Find C header file	Average .		nd-path. See the docstring of that user-option for further instructions. The pel-c-file-finder-method determines how pel-open-at-point searches for header files.
Find C neader life			efixes: <f11> SPC c as well as <f12> and M-<f12></f12></f12></f11>
PEL key prefixes:	'		n in keystroke cells (for brevity). The other two prefixes are only available in c-mode buffers.
Open this PDF file.	<f11> SPC c <f1></f1></f11>	(pel-help-pdf &optional	Open the $\mathfrak{P}I - \mathbf{C}$ local PDF. If the prefix argument (like $\mathbf{C} - \mathbf{u}$ or \mathbf{M}) is used, then it opens the
See also: <u>∑ Help/Info</u>	<f12> <f1></f1></f12>	ÖPEN-WEB-PAGE)	remote GitHub hosted raw PDF instead. If the pel-flip-help-pdf-arg user-option is set it's the other way around.
∑ Customize PEL C	• <f12> <f2></f2></f12>	(pel-customize-pel	Customize PEL C support.
support S Customize Emacs C	• <f21> # <f2> <f12> <f3></f3></f12></f2></f21>	&optional OTHER-WINDOW) (pel-customize-library	If OTHER-WINDOW is non-nil (use C-u), display in another window. Customize Emacs C support: c, c-macro, bison-mode, electricity
support Customize Emacs C	<f12> # <f3></f3></f12>	&optional OTHER-WINDOW) (pel-customize-library	
pre-processor support		&optional OTHER-WINDOW)	
CC Mode Style Management	 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 		
• Learn/Modify			t buffer: Emacs provides the following commands to parse the source code and identify the style liables from what it detects in the buffer.
style used in	• duse this to adapt to sour	ce code written by others and	want to continue using the same style, or to modify the style.
current buffer	For the following comma	nds all commands that use a ke	ey binding that ends with an upper case letter install the style.
Show/Modify	C-c C-o	(c-set-offset SYMBOL	Change the value of a syntactic element symbol in 'c-offsets-alist'.
syntactic context Set style indentation		OFFSET &optional IGNORED)	SYMBOL is the syntactic element symbol to change and OFFSET is the new offset for that syntactic element.
Show syntactic information for	C-c C-s	(c-show-syntactic- information ARG)	Show syntactic information for each syntactic element present on the current line. • Display the syntactic information list: style and position highlight the reference position(s)
current line		miorination ANG)	listed as argument to the syntactic list.
			 Each list starts with a <u>syntactic symbol</u> with zero or several reference positions. With universal argument, inserts the analysis as a comment on that line.
Guess the style used	<f12> <f4> g g</f4></f12>	(c-guess-buffer-no-install	Guess the style on the whole current buffer; don't install it.
in the current buffer, do not install it		&optional ACCUMULATE)	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	<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.
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 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	<f12> <f4> g R</f4></f12>	(c-guess-region START	Guess the style on the region and install it.
region and install it.		END &optional ACCUMULATE)	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
		ACCOMOLATE)	 If given a prefix argument (or if the optional argument ACCOMOLATE is non-nil) then the previous guess is extended, otherwise a new guess is made from scratch.
Set buffer style to	<f12> <f4> g I</f4></f12>	(c-guess-install &optional	Install the latest guessed style into the current buffer.
guessed style and install it.		STYLE-NAME)	 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-
V			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	<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.
Lisp statements		,	• "STYLE NAME HERE" is used as the name for the style in the emitted code. If WITH-NAME is
			given, it is used instead. WITH-NAME is expected as a string but if this function called interactively with prefix argument, the value for WITH-NAME is asked to the user.
Toggle preprocessor	<f12> <f4> #</f4></f12>	(c-toggle-cpp-indent-to-	Toggle the C preprocessor indent-to-body feature. When enabled, preprocessor directives which
line indentation		body &optional ARG)	are words in 'c-cpp-indent-to-body-directives' are indented as if they were statements.

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

• CC Mode state displayed in the mode line: \$C(...) where: Behaviour control • 9 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, 'h' for hungry mode, 'w' for subword mode to display the current (c-toggle-electric-state Toggle the electric indentation feature done with the electric character keys. Toggle Electric state • C-c C-1 &optional ARG) · Optional numeric ARG, if supplied, turns on electric indentation when positive, turns it off <f12> <f4> e † when negative, and just toggles it when zero or left out. • C-c . • <f12> <f4> s (c-set-style STYLENAME Set indentation style Set the bracket/indentation style 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</u> &optional DONT-OVERRIDE) † Emacs but you can also add your customized mode with some Emacs Lisp code. (pel-cc-set-indent-width <f12> <f4> TAB Interactively change the Indentation with for current buffer to NEW-WIDTH. Change indentation width for current &optional NEW-WIDTH) Prompt for new value. Use 0 to restore value specified by configuration (pel-c-indenthuffer width). This can be used to change indentation several times in a file. Toggle syntactic <f12> <f4> i (c-toggle-syntactic-Toggle syntactic indentation. Toggle if no ARG or if ARG is 0. indentation indentation & optional ARG) · With positive ARG turn on syntactic indentation, turns it off when negative. · When syntactic indentation turned on (the default), the indentation functions and electric keys indent according to syntactic context keys, when applicable. When it's turned off, the electric keys don't reindent, the indentation functions indents every new line to the same level as the previous nonempty line, and **M-x c-indent-command** adjusts the indentation in steps specified by 'c-basic-offset'. The indentation style has no effect in this mode, nor any of the indentation associated variables, e.g. 'c-special-indent-hook'. Toggle the <u>C comment style</u> between block/C-style (/* */) and line/C++-style (//) comments.

• With optional numeric ARG, switch to block comment style when positive, to line comment (**c-toggle-comment-style** & optional ARG) **Toggle Comment** • C-c C-k <f12> <f4> M-; t Style style when negative, and just toggles it when zero or left out. deline: **C/*** or **C/**/ The C++ style // comments (also called line comments) is compatible with C since C-99. for. C/block or line **Toggle Hungry Delete** <f12> <f4> DEL (c-toggle-hungry-state 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. &optional ARG) + mode 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 (pel-toggle-newline-Toggle variable pel-newline-does-align for the local buffer: toggles how 'pel-newline-and-<f11> M-RET on pel-newline-and-indent-below indent-below' operates: If *pel-newline-does-align* is t, it aligns several syntactic element in the current block: the comments, the assignments. indent-align) See also: Identify modes where pel-newline-does-alian is automatically activated (set to t) by ∑ Align
 ∑ Indentation 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: Ť • pel-cc-newline (assigned to **RET** in CC modes like c-mode, c++-mode and d-mode). pel-newline-and-indent-below (assigned the M-RET) (c-toggle-auto-newline Toggle auto-newline feature. Toggle auto-newline • C-c C-a &optional ARG) 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. insertion mode <f12> <f4> M-RET t 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 <f12> <f4> RET (pel-cc-change-newline-Change the way the RET key behaves in the CC modes and display the new mode in the echo area. 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 behaviour: select .. mode) return mode. its execution with an argument. newline-and-indent: uses (**newline** ARG t) to insert newline and indent. + 3. just-newline-no-indent: uses (electric-indent-just-newline ARG) ► Emacs default is to use newline. PEL sets the default to c-context-line-break which provides more functionality for CC modes. A mode change is local to the current buffer and does not affect RET key behaviour in the other buffers using the same mode. PEL user option pel-initial-c-newline-mode can be set to change the default for c-mode. **Display current Mode** <f12> <f4> ? (pel-cc-mode-info) Display information about current **CC mode** derivative for the current c-mode buffer. settings The information displayed in specialized help buffer includes the following: • CC mode style currently active, along with a list of styles associated with current mode. Change it for the current buffer with C-c . or <f12> <f4> s. The 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 C Language Revisions used to set the style for c-mode. Use <f12> <f2> from a c-mode buffer to access the customization buffer to change it. C coding styles: Return key behaviour: **GNU Coding** • RET (return key) mode. Change with pel-cc-change-newline-mode (<f12> <f4> RET). Standard / C 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): coding style 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. Barr Group C 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. Coding Standard 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. CS50 Style defined that tab width does not identify the indentation. It controls the spacing used in some commands moving point to the next tab stop Carnegie 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 it Mellon majerle C99/ is set and whether it was override by executing the **pel-cc-set-indent-width** command for this buffer (use <f12> <f4> TAB) for that command. The style currently used for indentation and bracket positioning (they should have the same value). Emacs identifies several built-in styles but you can later clang-format create your own. The example below shows "bsd" with is another name for the Allman style. You can dynamically change for the current buffer with c-setstyle command (C-c . or <f12> <f4> s). 🤞 CC Mode styles identify everything, including the number of indentation columns. PEL configures the style from the requested pel-c-bracket-style and then updates the indentation and other settings from the PEL user option requested. This allows you to slightly modify an existing style without having to create a new style name for it. The comment style. Supports C-style (/* */) and C++-style (//) comments since both are now accepted in C since C99. • This can be changed dynamically for the current buffer with the c-toggle-comment-style command (C-c C-k or <f12> <f4> M-;). C comment continuation lines can use 1 or 2 star characters: if a second one is used on a comment continuation line the remainder of the comment continuation lines used two stars, otherwise only one is used. Whether hungry delete is used by **DEL** and **C-d**. Toggle this for the current buffer with **c-toggle-hungry-state** (<f12> <f4> DEL). The file search methods and parameters used by pel-open-at-point (see sections below). Example: Notice the c-mode state: : bsd. c-default-style: (bsd)
: context-newline, and aligns (comments, assignments, etc...) active style name of the PEL user-RET mode : context-newline, and Electric characters : active on: #*/(){}:;, options that set the Auto newline fill column Tab width significant feature on
80, auto-filling: off.
4 Set via: pel-c-tab-width(4) ==> tab-width(4) when c-mode buffer is opened.
hard-tabs & spaces Set via: pel-c-use-tabs(t) ==> indent-tabs-mode(t) when c-mode buffer is opened.

Set via: pel-c-indent-width(4) ==> c-basic-offset(4) when c-mode buffer is opened. controlling Emacs variables in the Indentation chars message Indent width More info is shown Syntactic indent c-indentation-style linux-kernel in that buffer as PEL Bracket style Comment style Hungry delete Block comments: /* */, continued line start with * off, but the F11-© and F11-© keys are available.

None found, searching for files identified in pel-project-root-identifiers: (.git .hg .projectile .pel-project) generic buttons that provide access to more help and ability to customize Project rootFile finder method

pel-ffind-executable: fd

<u>Description</u>	<u>Keystroke</u>	Function	Note Note
C Code Help	There are several Emacs exter	nsion packages that can help w	rriting C code.
Get man help about C code See: <u>∑ Help/Info</u>	• <f11> ? m • M-<f8> • %-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.
	1 1	mproved into providing the info	pel-use-c-eldoc is set to t.
Electric Keys	9		then 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 sp	e variable 'c-electric-pound-behavior', which can only be nil or 'alignleft'. If a numeric ARG is pecial happens.
()	a literal.		Insert a parenthesis. un-nil, the line is reindented unless a numeric ARG is supplied, or the parenthesis is inserted inside may get added or removed; see the variable 'c-cleanup-list'.
0	Also, if 'c-electric-flag' and '	c-auto-newline' are both non-	nil, 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 featin 'c-hanging-braces-a b) Any auto-newlines are	ture is turned on (indicated by lilist'. 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.
:	c) if auto-newline is turne	(c-electric-colon ARG)	s based on the settings of 'c-cleanup-list' are done. Insert a colon.
	If 'c-electric-flag' is non-nil, a) If the auto-newline feath hanging-colons-alist'.	ture is turned on (indicated by	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- llso reindented unless 'c-syntactic-indentation' is nil.
		ed on, whitespace between two	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) When the auto-newline semi&comma-criteria' b) Any auto-newlines are	e feature is turned on (indicated for how newline insertion is de indented. The original line is 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-termined. llso reindented unless 'c-syntactic-indentation' is nil. icce list or a semicolon following a defun might be cleaned up, depending on the settings of 'c-
Electric pairs	Type the first of a pair to inse	ert this one and its matching ch	activating the electric-pair-mode in the buffer. aracter for (), [], {}, "" and ''. line lighter set by the pel-electric-pair-lighter is shown. This defaults to $\epsilon(1)$
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		aoptonai <u>Atto</u>)	With a prefix argument ARG, enable Electric Pair mode if ARG is positive, and disable it otherwise.
$\textbf{Lighter:=} \ \epsilon(1)$			 Electric Pair mode is a global minor mode. When enabled, typing an open parenthesis automatically inserts the corresponding closing parenthesis, and vice versa. (Likewise for brackets, etc.). If the region is active, the parentheses (brackets, etc.) are inserted around the region instead.
Use Linux Kernel code style	<f12> M-k</f12>	(pel-linux-kernel-code- style)	Manually activate Linux Kernel Coding Style regardless of file content.
Insert New Line(s)	active the point also moves to With PEL the default behavi command (bound to <f12> The pel-cc-newline comman</f12>	the proper indentation accordi our can be selected by custom M-RET) see the CC-Mode belnd also aligns comments and a	olode 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. Dization and modified dynamically for the current buffer with the pel-cc-change-newline-mode naviour control section above. Is signment in the code block if the pel-modes-activating-align-on-return user option list 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.	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)
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.	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: • E Filling/ Justification	Use: (newline & optional ARG INTERACTIVE): Insert a newline, and move to left margin of the new line if it's blank. • With ARG, insert that many newlines. • 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.		marked with the text-property 'hard'. inal new line that it adds, and reindents the preceding line. st-newline. is greater than the value of 'fill-column' and ARG is nil.
	Use: (electric-indent-just • With ARG, insert that r		ewline, without any auto-indentation.
Insert an indented line below unbroken current line See also: Indentation	• M-RET • <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. But lightly 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>

<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:
	In programming langua	age modes, this is the same as	
a N 1: :	·		mmand indents to the column specified by the function 'current-left-margin'.
Open New Line in Context	C-o	(c-context-open-line)	Insert a line break suitable to the context and leave point before it.
See also: • <u>National Whitespace</u>	Normally C-o is bound to c	pen-line. PEL rebinds it to c-c	', which is normally bound to C-o . See 'c-context-line-break' for the details. ontext-open-line for the CC modes. t use open-line via <f12> C-o</f12>
Open new line	• <f12> C-o • M-<f12> C-o</f12></f12>	(open-line N)	Insert a newline and leave point before it. With arg N, insert N newlines. • If there is a fill prefix and/or a 'left-margin', insert them on the new line if the line would have been blank.
<u>C Comments</u>	2 more characters have electri • 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 also: Filling/Justification	*	(c-electric-star ARG)	Insert a star character. If 'c-electric-flag' and 'c-syntactic-indentation' are both non-nil, and the star is the second character of a C style comment starter on a comment-only-line, indent the line as a comment. If a numeric ARG is supplied, point is inside a literal, or 'c-syntactic-indentation' is nil, this indentation is inhibited. With this key being electric it becomes easy to type the following two styles of multi-line block comment: /* Two star ** continuation ** prefix for
			** multi-line ** C comment. /* Single star * prefix for * multi-line * C comment. */ • When typing the '*' at the beginning of the line, it indents automatically. If another '*' is typed, indentation is set to allow a two-star continuation, otherwise it is placed for a single star continuation. • When auto-fill-mode is active, when you type a comment that would be longer than the line, 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>▼ Comments</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: • ***Comment region with style selected by pel-c-multiline-comments user-option: • default (like comment-dwim): each line is commented with a /* */ • 1: single start multi-line comment (see example in box above) • 2: double star multi-line comment (see example in the box above) • With marked commented region: • removes the comment. • When a prefix ARG is specified, call 'comment-kill'. Else, call 'comment-indent'. • You can toggle between C-style /* */ and C++ style // comments (compatible with C since C-99) <f12> M-;</f12>
Comment/un- comment	C-c C-c		Comment or uncomment each line in the region. • With just C-n prefix are uncomment each line in region REG. END
See also: <u>▼ Comments</u>	 &optional ARG) With just C-u prefix arg, uncomment each line in region BEG END. 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. 		If ARG is negative, delete that many comment characters instead. ent-start' and 'comment-padding'; the strings used as comment ends are built from 'comment-start' markers are inserted at the current indentation of the region, and comments are terminated end the comment and blank lines do not get comments). This can be changed with 'comment-yle comments is active, the comment ends on the next space, which is probably not what you
Fill current paragraph See also: E Filling/Justification	• M-q • <f12> F • M-<f12> F</f12></f12>	(c-fill-paragraph &optional ARG)	Like <f11> t f p but handles // and /* */ style comments. • If any of the current line is a comment or within a comment, fill the comment or the paragraph of it that point is in, preserving the comment indentation or line-starting decorations (see the 'c-comment-prefix-regexp' and 'c-block-comment-prefix' variables for details). • If point is inside multiline string literal, fill it. This currently does not respect escaped newlines, except for the special case when it is the very first thing in the string. The intended use for this rule is in situations like the following: char description[] = "\ A very long description of something that you want to fill to make nicely formatted output."; • If point is in any other situation, i.e. in normal code, do nothing. • Optional prefix ARG means justify paragraph as well.</f11>
Toggle subword-mode See also: Text Modes	• <f11> t m b • <f12> M-b • M-<f12> M-b</f12></f12></f11>	(subword-mode &optional ARG)	Toggle subword-mode: a minor mode that treats sections of <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>S Comments</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 <u>hide-comnt.el</u> package (see <u>▶ Comments</u>). • PEL activates it when the pel-use-hide-comnt user option is t.

<u>Description</u>	Keystroke	Function	<u>Note</u>
Hungry Deletion of Whitespace	The CC mode provides two commands that can perform "hungry whitespace deletion" that can also be used in every mode. PEL provides the convenient keys with the <f11> prefix keys for those 2 commands, available in all modes. In modes compatible with the CC Mode (e.g. for C, C++, D, Java, Pike, etc) it is also possible to activate the Hungry Delete Mode to modify the behaviour of the simple and C-d, to perform hungry deletions. That's not currently supported in other modes. When the Hungry Delete Mode is on, the mode-line displays a 'h' to the right of the '//!' indication of electric mode. The Hungry Mode also activates the key prefixes below that start with C-c. They are listed but remember they are only available once the Hungry state mode is activated (and that can only be done in modes that are CC Mode compatible). In modes derived from CC Mode you can also activate the hungry state to make standard delete commands delete hungrily, but that does not work for other modes. PEL provides the <f12> M-DEL key for those modes (like C). Toggle hurry deletion mode of the DEL and C-d key for the current buffer with c-toggle-hungry-state (<f12> M-DEL).</f12></f12></f11>		
Delete preceding char or all preceding whitespace. See also:	• C-c DEL • C-c @ • C-c C-@ • C-c C-Sbackspace> • C-c C-DEL • <f11> @ @ • <f11> DEL DEL</f11></f11>	(c-hungry-delete- backwards)	Delete the preceding character or all preceding whitespace back to the previous non-whitespace character. In terminal mode, even though C-@, C- <backspace> and C-DEL are not available, they are mapped to the non-control key so attempting to type them end up invoking the command anyway because the first key bindings are recognized. With PEL, the <f11> @ @ 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></backspace>
Delete next char or all following whitespace. See also: <u>See Cut & Paste</u>	• C-c C-d • C-c ® • C-c C-® • C-c C- <delete> • <f11> ®</f11></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-® and C- <delete> are not available, they are mapped to the non-control key so attempting to type them end up invoking the command anyway because the first key bindings are recognized. With PEL, the <f11> ® binding is always available, in all modes. The other keys are only available in modes derived from the CC Mode. This prevents conflicts with other modes that may use the popular C-c bindings.</f11></delete>
<u>Indentation</u>	behaviour control section above You can also explicitly request • The first set of commands p	ve. indentation using the comman erform syntactic indentations s	
Indent current line or region See also: Indentation	 (c-indent-line-or-region & optional ARG REGION) 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	d to M-i to insert spaces to the (indent-pp-sexp & optional ARG)	 Indent each line of the list starting just after point, or pretty-print it. A prefix argument (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	For most editing scenarios, width while the other 2 use pe	, it's best to set pel-c-tab-wid tel-c-indent-width.	pards to semantics. More information on indentation is available in the <u>Nation</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: Indentation	M-i	(tab-to-tab-stop)	Insert spaces or tabs to next defined tab-stop column. The exact location of the next tab stop is identified by the value of the tab-stop-list and tab-width for the current buffer. With PEL, the tab-stop interval is controlled by the value of pel-c-tab-width. PEL sets tab-width to the value of pel-c-tab-width for each c-mode buffer.
Indent/Unindent rigidly See also: • ∑ Indentation • ∑ Key-Chords	• C-x <tab> • <f11> <tab> <tab> • <tab>q</tab></tab></tab></f11></tab>	(pel-indent-rigidly &optional N)	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.
	indent-rigidly Indent all lines s If called interactively w <right>, S-<left> Both of these commands activ capabilities are controlled by the second of th</left></right>	starting in the region. ith no prefix argument, activate >, or S- <right>. rate a transient mode where En ne variable indent-rigidly-map v rigidly-right-to-tab-stop rigidly-left-to-tab-stop rigidly-right rigidly-left es the transient mode. Left> keys indent/de-indent terval is controlled by the value the value of pel-c-tab-width fon ne cua-mode uses C-x, to invo</right>	to the next tab-stop position, which is controlled by the tab-width user option.

<u>Description</u>	<u>Keystroke</u>	Function	<u>Note</u>	
Indent line(s) rigidly	• <f6> <tab> • <f11> <tab> c</tab></f11></tab></f6>	(pel-indent-lines &optional N)	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>	 All lines touched by the region are indented. A special argument N can specify more than one indentation level. It defaults to 1. If a negative number is specified, 'pel-unindent-lines' is used. If a region is marked, the function does not deactivate it to allow repeated execution of the command. It also modifies the region to include all characters in all affected lines. Use C-g to de-activate the region. Handles presence of hard tabs: If indent-tabs-mode is non-nil the indentation is created with a mix of hard-tabs and space characters. If indent-tabs-mode is nil, any hard tab in the indentation of the marked lines is replaced by the proper number of spaces. Hard tabs after first non-whitespace character on the line are left. 			
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-indent-width. Works with point is anywhere on the line. 	
See also: • Indentation	 All lines touched by the region are un-indented. If region was marked, the function does not deactivate it to allow repeated execution of the command. 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 characters in all affected lines. Use C-g to de-activate the region. Handles presence of hard tabs: If indent-tabs-mode is: non-nil the indentation is created with a mix of hard-tabs and space characters. nil, any hard tab in the indentation of the marked lines is replaced by the proper number of spaces. Keeps hard tabs after first non-whitespace character. 			
Open file at point See also: File mngt After changing user- options you currently must restart PLL by executing pel-init or restarting Emacs!	In a c-mode buffer the computation of the search method pel-c-file-finder-method pel-c-file-searched-ext pel-c-file-finder-ini-tool the pel.ini file, a INI-format conchain can be overridden by the executed). Use the command	The following command allow opening files from the file name taken at point (the cursor location). In a c-mode buffer the command is specialized to be more useful for C programming and has the extra capability of searching files where header files are stored. The search method is controlled by the following user-options: pel-c-file-finder-method : identifies one of 4 supported method of identifying the header files. See their descriptions below. pel-c-file-searched-extra-dir-trees : identifies one of 4 supported method of identifying the header files. See their descriptions below. pel-c-file-finder-ini-tool-name : List of extra directory trees also searched by the tool identified by pel-ffind-executable user-option. The name of a tool chain TTT, to select one of the TTT-c-path tool-chain key inside the [file-finder] section of the pel.ini file, a INI-format configuration file. The value mapped to that key identifies the list of directories to search for that tool-chain. The name of the tool chain can be overridden by the value of the environment variable PEL_CC_FIND_TOOLCHAIN, which is read and used when Emacs starts up (or pel-init is executed). Use the command pel-cc-set-file-finder-ini-tool-name to change the currently used tool chain name. Note that when using the Ido completion mode, it is possible to instruct Ido to use a file name at point as the basis for the file name to open.		
Show active file finder setup for current buffer	• <f12> <f4> f • <m-f12> <f4></f4></m-f12></f4></f12>	(pel-cc-find-show-status &optional APPEND)	Print C specific PEL file finding control user-options and variables info inside a *pel-cc-ffind-status* help-mode buffer. Prints current state and values of relevant user-options and variables as buttons you can use to get more info and change the values of the user options. Clear previous buffer content by default. Use prefix arg (like C-u) to append instead.	
Set name of Tool-Chain specific include path. • (with pel-ini-file search method)	• <f12> <f4> M-<f6> • <f12> <f4> <f54></f54></f4></f12></f6></f4></f12>	(pel-cc-set-file-finder-ini- tool-name &optional TOOL- NAME)	Change activate value of tool-chain name key identified by value of pel-c-file-finder-ini-tool-name user-option. The change is not persistent. Requires pel-use-ini set to t. Used when the pel-c-file-finder-method is set to pel-ini-file. In that case it effectively select a new set of tool-chain specific directories to search by pel-open-at-point. The directories are identified by the corresponding TTT-c-path key in the [file-finder] section of the pel.ini file.	
Open file or web-page whose name is at point ★★	• M- <f6> • <f11> f . • 6y</f11></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. The 6y key-chord is available if pel-use-key-chord is non-nil. See Key-Chords.</f4></f12>	
C Header File finding control	This command works generically but is also specialized for C major mode: it opens the header file identified by the #include statement. Aside from generic method described below, the command searches for the header file to open using the method identified by the pel-c-file-rmethod and the pel-c-file-searched-extra-dir-trees user-options. The first one identifies one of the following search method, the other identifies extra directory tree(s) to search using the search tool identified by the pel-ffind-executable user-option: • generic: the command searches, in current directory and its parents, for a file identifying the parent root directory; a file with a name identified in the pel-project-root-identifiers user-option. Something like .git, .h.gproject or .pel-project by default. Then searches for files inside that directory tree. • pel-ini-file: the command searches inside directories identified by lists defined in the pel.ini file which PEL identifies for the project like it does for project			
pel-use-ini F	 marker. The <i>pel.ini</i> file is a INI file format. When found, it is opened and information inside the file identifies where to search. The file must contain a [file-finder] section with: The project-path key. The value is a list of directories to search recursively. One or several TTT-c-path key(s), where TTT is a tool-chain name. The value is a list of directories to search recursively for that tool-chain. The currently used tool chain is identified by the following values in order (first one takes priority on startup): The centent of the DEL COLUMN TOOL CHAIN equipment variable, if it exists. 			
specialized for: • MrestructuredText • MrestructuredText • MrestructuredText • MrestructuredText • MrestructuredText • MrestructuredText	 The content of the PEL_CC_FIND_TOOLCHAIN environment variable, if it exists. The content of the pel-c-file-finder-ini-tool-name user-option; which identifies the name of a TTT-c-path key. The paths identified in the two 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> M-<f6> and specifying another name.</f6></f4></f12> 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. Directories are not searched recursively for the last 2 options. 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 			
Generic Delimiting characters	In general the command extra The generic mode extraction	cts the file or directory name, a	bles. The \$VARNAME format must be used. Directories are not searched recursively. Ind possibly line and column numbers, from text at point and tries to open the file or directory. Inning & end of the file/directory/library/URL name string by delimiter characters, one of: tab, I	
The complete file detection heuristic is described in the <u>File</u> mngt description of the same command.	The default is a very prim With ivy selected, PEL Note that the command s	itive function implemented by I will automatically set 🛂 pel-ushows all files found by the spe	n and prompts using the method selected by pel-prompt-read-method user-option. PEL. You can select a more powerful <u>ivy</u> prompting instead. Ise-ivy to t and <u>Ivy mode</u> will be installed automatically when you restart Emacs. Icified search method, it does not only use the first one found. If the names in large include paths.	
Select target window F	 Select target window: Without argument: 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. 			
N>20 : open the directory • See function docstring for more info.	 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. With prefix numeric argument N: N < 0: create a new window and use that. (abs N) > 20: then open the directory instead of the file. Interpret the window position from the N value adjusted: N-20 (or N+20 if N is negative) N = 0: use the 'other' (the next) window. N = 1, 3, 7 or above (excluding 8, 9 and 10): select the target window based on the number of editable windows in frame: if 1 window: split that window and use the new window, if 2 windows: use the other window, if 3 or more windows: use the current window. N is: 8: up, 2: down, 4:left, 5:current, 6:right. N is 9: force opening the file in the OS associated application (with N=29 or N=-29, open the file's directory with the OS associated application) 			
Open file with alternate extension		ndows Explorer). If this is a UF istent or dedicated window is r (pel-open-file-alternate)	Open a file with same name but an alternate extension. The new extension depends on the current file extension.	
Supports: • <u>∑ File-mngt</u> • <u>¾í - C++</u>			 The list of alternate extensions is currently very limited and restricted to C and C++. If the alternate file is not found, save the file basename in the kill ring and prompt for the file name to open. 	

Tempo skeletons for C See also: C Code Templates	PEL creates key bindings to	· · · · · · · · · · · · · · · · · · ·	ough the Emacs built-in tempo skeleton mechanism.		
See also:		sino onolotorio ili tire su	upported major modes, using the same key prefix sequence for each mode: <f12> <f12>, with</f12></f12>		
			le header block) as much as possible.		
 C Code Templates 		PEL Emacs Lisp Source Code S From a C mode buffer and inc	Style is controlled by the user options inside the pel-c-code-style group. This group can be edited		
20	pel-c-skel-module-hea	der-block-style: allows select	ting a user-define module-header comment block.		
• <u>∑ Inserting Text</u> for	 pel-c-skel-comment-w pel-c-skel-insert-file-til 		mat of C-style continuation comments. automatically updated timestamp is inserted in the file header block.		
more info and information about	pel-c-skel-use-separat	0.0	cks use horizontal separator lines.		
tempo skeleton and	 pel-c-skel-doc-markup pel-c-skel-cfile-section 		cumentation markup supported by the templates. Currently 'none' and 'Doxygen' are available. nentation section titles inserted in code files.		
yasnippet template- based text insertion	pel-c-skel-hfile-section	-titles : identifies docum	nentation section titles inserted in header files. A section titled "." split sections placed before		
		on-sections : set whether C fu	clude guard. If not present all sections are placed after the include guard. Support of the function description comment.		
	 pel-c-skell-function-se pel-c-skel-function-det 		e of the C function templates sections inserted when pel-c-skel-insert-function-sections is t. ction comment block style. Several styles are provided:		
	·	- no special com			
		- a Man-page st	tyle comment block with the sections identified by pel-c-skell-function-section-titles		
	pel-c-skel-function-nai		tempo skeleton loaded from a user specified file name. See the <u>source code example.</u> s 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		
	pel-c-use-include-guar		e text controlled by 🛂 lice. pe of include guard is inserted in header files. The available choices are:		
			clude guard tpragma once statement		
		- use c	classic #ifdef/#define/#endif block using symbol created from file name		
	Emacs user options by def		u #ifdef/#define/#endif block using symbol created from file name and UUID for its uniqueness. y using file and directory variables (see Eile/Directory Variables) they can also be used to take		
	effect on a single file or all file	s inside a directory tree. So by	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 ils file and store the values of the relevant options variables inside that file. This allows you to		
			mplates precisely and does not affect what you actually type. e pel-tempo-mode) you can move to the next or previous point of interest (so called tempo-marks)		
			\mathbf{M} - \mathbf{b} or some other keys like \mathbf{C} - \mathbf{c} . and \mathbf{C} - \mathbf{c} .		
© Customize PEL C	<f12> <f12> <f2></f2></f12></f12>	(pel-customize-pel &optional OTHER-WINDOW)	Customize PEL C skeleton layout. • If OTHER-WINDOW is non-nil (use C-u), display in another window.		
Skeletons layout	<f12> <f12> h</f12></f12>	(pel-elisp-file-header)	Insert a file description block. Distinguish between code files and header files.		
insert a me neader	<112> <112> II	(per-elisp-file-fileader)	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	<f12> <f12> f</f12></f12>	(pel-c-function)	Insert a C function definition code and comment template.		
comment block	 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></up> 				
	and <down></down> keys.		-		
	 The style of the code insert Use C-g to cancel at any 		tions inside the pel-c-code-style group and the various C style element controls of the CC-mode. ne examples in the PEL manual.		
Toggle pel-tempo-	<f12> <f12> SPC</f12></f12>	(pel-tempo-mode &optional			
mode		ARG)	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.		
lump to provious	• C-c C	(tempo-backward mark)	Jump to the previous mark in 'tempo-back-mark-list': the location where code must be updated		
Jump to previous tempo mark	• C-c M-b • C-c ,	(tempo-backward-mark)	inside the inserted skeleton.		
	• C-c C-,		These key binding are only available when pel-tempo-mode is active.		
Tempo Template Tag Insertion	<f12> <f12> <f12></f12></f12></f12>	(tempo-complete-tag &optional SILENT)	Look for a tag and expand it.		
	Instead of using the < £12	· · · · · · · · · · · · · · · · · · ·			
		· · · · · · · · · · · · · · · · · · ·	r, you can type the template name (shown in the title column like "ir", "case", etc) completely or a buffer opens up if the template name is incomplete (or empty in which case the buffer lists all		
			t RET. Emacs expands the template. des 'tempo-tags') are searched for a match for the text before the point. The way the string to		
	match for is determined ca		rempo-match-finder'. If 'tempo-match-finder' returns nil, then the results are the same as no		
			expanded in place of the matching string. If a partial completion or no match at all is found, and		
			al completion is found and 'tempo-show-completion-buffer' is non-nil, a buffer containing possible		
	completions is displayed.	9 9 р			

<u>Description</u>	<u>Keystroke</u>	Function	<u>Note</u>
Inserting code	Extra text insertion can be dor	ne with the following command	s. See also above: <f12> M-e activates electric pair: typing (inserts the matching)</f12>
Insert Parentheses	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>R Marking</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 the A mark is left where the con	nmand started, unless the region	cro, point at beginning. hosen. Each successive call of this command extends the marked region by one function. on is already active (in Transient Mark mode). 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</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 snake_case 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 []. ses the one before or after point. ser highlighted with the same colour.
Toggle show-paren mode on/off	• <f12> M-9 • M-<f12> M-9</f12></f12>	(show-paren-mode &optional ARG)	Toggle visualization of matching parens (Show Paren mode). • With prefix argument ARG, enable Show Paren mode if ARG is positive, disable it otherwise. • Show Paren mode is a global minor mode. When enabled, any matching parenthesis is
See also: <u>E Highlight</u>	• <f11> h (</f11>		highlighted in 'show-paren-style' after 'show-paren-delay' seconds of Emacs idle time.
Enable/Disable coloured highlight of nested blocks (),{},[]	• <f12> M-r • M-<f12> M-r</f12></f12>	(rainbow-delimiters-mode &optional ARG)	Highlight nested parentheses, brackets, and braces with colours according to their depth. • Customize the depth and colours with M-x customize-group rainbow-delimiters • Requires: rainbow-delimiters.el
See also:	• <f11> h R</f11>		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 of	nly. See the others inside <u>Navigation</u>
By definitions	Move to the definition of funct		f for more information to activate the various engines that support cross referencing for C code.
Find definition of identifier at point See also: Xref	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 With dumb-jump this performs a search using ag, ripgrep or git grep if available.
Go back to where M was last issued	М-,	(xref-pop-marker-stack)	 Pop back to where M was last invoked. Marker depth is controlled by the xref-marker-ring-length user option.
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 C preprocessor	conditional inclusion statem	ents #if #ifdef #ifndef #else #elif #endif / ! Does not yet support C++23 #elifdef and #elifndef
Move point forward to matching #endif or matching #else #elif	<f6> <right></right></f6>	(pel-c-preproc-forward- conditional &optional TO- ELSE)	Move point forward to matching #endif If point on a #if #ifdef #ifndef statement moves to the matching endif With C-u or numerical arg: move forward to matching #else #elif On success, push the original position on the mark ring and return the new position. On error, issue user error on mismatch. Shift marking is available with C-M- <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 list-matching-lines-default-context-lines 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)	Show C pre-processor conditional statements of current project buffers inside an occur buffer. • Each line is shown with NLINES before and after, or -NLINES before if NLINES is negative. • NLINES defaults to list-matching-lines-default-context-lines user-option value. See occur search. • This command uses Projectile. You must have pel-use-projectile user-option set and projectile active (use <f11> <f8> <f8> to activate it.)</f8></f8></f11>

<u>Description</u>	<u>Keystroke</u>	Function	<u>Note</u>
By blocks functions structures	Move across C++ statemen When point is located be	efore opening brace or right aft	ucture definition blocks. ny group of (), [], {} or < > blocks. Jump over comments. er closing brace and show-paren-mode is on, the matching parentheses are highlighted. or several bindings to ease typing some sequences. The one easier to type is identified in bold .
Move block forward See also: ■ Navigation Use this to move to end of next syntax element or to end of block when already outside the block. Use C-M-u to exit a block (see below).	<pre> <f12> <right> <m-f12> <right> C-M-f C-M-<right> C-[C-f Esc C-f </right></right></m-f12></right></f12></pre>	(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>
Forward block/list See also: Navigation	С-м-п	(forward-list &optional ARG)	Move forward across one balanced group of parentheses. This command will also work on other parentheses-like expressions defined by the current language mode. With ARG, do it that many times. Negative arg -N means move backward across N groups of parentheses. This command assumes point is not in a string or comment. C-M-n : Shift marking is available in graphics mode, not in terminal mode.
Move block backward See also: Navigation	• <f12> <left> • <m-f12> <left> • C-M-b • C-M-<left> • C-[C-b • Esc C-b • Esc C-<left></left></left></left></m-f12></left></f12>	(backward-sexp &optional ARG)	Move backward across one balanced expression (sexp). • With ARG, do it that many times. Negative arg -N means move forward across N balanced expressions. This command assumes point is not in a string or comment. • C-M-b : ▼ Shift marking is available in graphics mode, not in terminal mode. • C-M- <left> : ▼ Shift marking works with this command. • ⚠ With PEL: if you want to use Esc C-<left> binding you must ensure that pelwindmove-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>
Backward block/list See also: Navigation	С-М-р	(backward-list &optional ARG)	Move backward across one balanced group of parentheses. • This command will also work on other parentheses-like expressions defined by the current language mode. • With ARG, do it that many times. • Negative arg -N means move forward across N groups of parentheses. • This command assumes point is not in a string or comment. • C-M-p : ► Shift marking is available in graphics mode, not in terminal mode.
Backward to beginning of current top-level function or struct	C-M-a • <f12> <up> • <m-f12> <up> C-M-<home></home></up></m-f12></up></f12>	(c-beginning-of-defun &optional ARG) (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. 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. </f12></f6> 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.
Forward to end of current top-level function or struct.	<pre>C-M-e • <f12> <down> • <m-f12> <down> </down></m-f12></down></f12></pre>	(c-end-of-defun &optional ARG) (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. 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
	C-M- <end></end>		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 top-level function. It skips nested functions.</f12></f6>
Backward to end of previous top level function or struct	• <f12> <m-up> • <m-f12> <m-up></m-up></m-f12></m-up></f12>	(pel-end-of-previous-defun &optional SILENT DONT- PUSH_MARK)	Move backwards to the end of the previous function or type definition. • Beeps if does not find end of previous function unless SILENT is non-nil. • If the end of previous function is found, push the start location to the mark ring unless DONT-PUSH_MARK is non-nil. • Move back to previous position with M−` or <f6><f6>. ■ Shift marking is available. With <f6> and <f12> hit Shift after function key, before cursor key. ⚠ In some cases it fails to detect the end of the previous block and fails.</f12></f6></f6></f6>
Forward to start of next top level function or struct Use this to move from the top of the file to the first block.	• <f12> <m-down> • <m-f12> <m-down></m-down></m-f12></m-down></f12>	(pel-beginning-of-next- defun &optional SILENT DONT-PUSH_MARK)	Move forward to the beginning of the next function or type definition. • Move point before the function type or the struct or typedef keyword. • Beeps if does not find beginning of next function unless SILENT is non-nil. • If the beginning of next function is found, push the start location to the mark ring unless DONT-PUSH_MARK is non-nil. • Move back to previous position with M− or <f6><f6>. ■ Shift marking is available. With <f6> and <f12> hit Shift after function key, before cursor key. ■ This command complements what end-of-defun does. • It moves forward but not to the end of the function definition (like end-of-defun) but to the beginning of the function definition, which is often what users of other editors expect.</f12></f6></f6></f6>
in/out of blocks Packward Un/outside		locks, 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. Mith PEL: if you want to use Esc C- <up> binding you must ensure that pelwindmove-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> : 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	C-M-]	(up-list &optional ARG ESCAPE-STRINGS NO- SYNTAX-CROSSING)	Move forward out of one level of parentheses or nested blocks. • Also work on other parentheses-like expressions defined by the current language mode. • With ARG, do it that many times. Negative arg means move backward but to a less deep spot.
Down/inside sexp/block See also: Navigation	• C-M-d • C-M- <down> • C-[C-d • Esc C-d • Esc C-<down></down></down>	(down-list &optional ARG)	Move forward down one level of parentheses. Also work on other parentheses-like expressions defined by the current language mode. With ARG, do it that many times. Negative arg mans move backward but still go down a level. This command assumes point is not in a string or comment. With PEL: if you want to use Esc C- <down> binding you must ensure that pel-windmove-on-esc-cursor user option is set to nil. C-M-d :▼ Shift marking is available in graphics mode, not in terminal mode. C-M-<down> :▼ Shift marking works with this command. C-M-<down> does not work on Windows, but H-<down> does.</down></down></down></down>

<u>Description</u>	<u>Keystroke</u>	Function	<u>Note</u>
By statements	Move to beginning /end of sta	tement or comment.	
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.
Go to start of current switch statement	<f6> t w s</f6>	(pel-cc-to-switch-begin)	Move point to the start { of current switch statement, if any. If point is inside switch statement, mark position before moving point. Move it back with M-`. If point is not inside a switch statement, issue a user error.
Go to end of current switch statement	<f6> t w e</f6>	(pel-cc-to-switch-end)	Move point just past the end } of current switch statement, if any • If point is inside switch statement, mark position before moving point. Move it back with M-`. • If point is not inside a switch statement, issue a user error.
Go to start of enum definition block	<f6> t e s</f6>	(pel-cc-to-enum-begin)	Move point to the start { of current enum definition block, if any. If point is inside such a block, mark position before moving point. Move it back with M-`. If point is not inside an appropriate block statement, issue a user error.
Go to end of enum definition block	<f6> t e e</f6>	(pel-cc-to-enum-end)	Move point just past the end } of current enum definition block, if any. If point is inside such a block, mark position before moving point. Move it back with M-`. If point is not inside an appropriate block statement, issue a user error.
Go to start of struct definition block	<f6> t s s</f6>	(pel-cc-to-struct-begin)	Move point to the start { of current struct definition block, if any. If point is inside such a block, mark position before moving point. Move it back with M-`. If point is not inside an appropriate block statement, issue a user error.
Go to end of struct definition block	<f6> t s e</f6>	(pel-cc-to-struct-end)	Move point just past the end } of current structdefinition block, if any. • If point is inside such a block, mark position before moving point. Move it back with M-`. • If point is not inside an appropriate block statement, issue a user error.
Go to start of union definition block	<f6> t u s</f6>	(pel-cc-to-union-begin)	Move point to the start { of current union definition block, if any. If point is inside such a block, mark position before moving point. Move it back with M-`. If point is not inside an appropriate block statement, issue a user error.
Go to end of union definition block	<f6> t u e</f6>	(pel-cc-to-union-end)	Move point just past the end } of current union definition block, if any. If point is inside such a block, mark position before moving point. Move it back with M-\[^\circ\]. If point is not inside an appropriate block statement, issue a user error.
Open the C preprocessor hydra with <f12> <f7> followed by on of the hydra keys:</f7></f12>	* Requires the hydra ex -UUU:**F1 a c C preprocessor: Move to	ternal package PEL activate file.c All (4,0) de	The pel-Zc-preproc Hydra allowing further hydra keys to be typed without any prefix. The swhen the pel-use-hydra user option is set to t. The control of t
Navigate across pre- processor conditionals	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.		
Move to previous preprocessor directive	• <f12> # p * <f12> <f7> p</f7></f12></f12>	(pel-pp-prev-directive)	Move point to previous preprocessor directive.
Move to next preprocessor directive	• <f12> # n * <f12> <f7> n</f7></f12></f12>	(pel-pp-next-directive)	Move point to next preprocessor directive.
Move up in the pre- processor conditional block	• C-c C-u * <f12> <f7> C-u</f7></f12>	(c-up-conditional COUNT)	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. "#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 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-<f12> # #</f12></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 ar	region. If the region ends at the start of a line and the macro doesn't continue below it, the nd use this command to conveniently insert and align the necessary backslashes. ling to: 'c-backslash-column', 'c-backslash-max-column' and 'c-auto-align-backslashes'.
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.
			10

<u>Description</u>	<u>Keystroke</u>	Function	<u>Note</u>		
Hide-ifdef Mode			of code that the C preprocessor wouldn't pass through.		
 hide/show code controlled by C- 	 It supports complete C/C++ It scans for new #define syn 	nbols and macros.			
preprocessor	Hide-ifdef environment: t	he hide-ifdef-env association	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>Elelp/Info</u>.</f1>		
		en code is marked by ellipses (ting near ellipses, since the hid	(). den 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-ifde	ef-hiding by setting hide-ifdef-read-only user-option to a non-nil value.		
		-ifdef customization group with <f12> # <f3> is 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>			
		■ With PEL, the commands are reachable via the <f12></f12> prefix keys can also be reached via the M-<f12></f12> and the <f11></f11> SPC c prefix keys. * The key sequences that start with <f12></f12> <f7></f7> open the pel-∑c-preproc <u>Hydra</u> allowing further hydra keys to be typed without any prefix.			
			ivates when the pel-use-hydra user option is set to t .		
	Several customize user op	tion variables affect how the hi	ding is done:		
Use <f12> # ?</f12>	• to change, execute: M-x customize-group hide-ifdef or type <f12> # <f3></f3></f12>				
to show the value of hide-ifdef-env and			project. The list holds the following forms:		
hide-ifdef-define-alist	(SYMBOL . VALUE)	when the SYMBOL is defined (when the symbol is defined wi			
			nide-ifdef-set-define-alist' to save the current 'hide-ifdef-env' and 'hide-ifdef-use-define-alist' to		
	 'hide-ifdef-lines' 	def-env' from one of the lists in			
	 'hide-ifdef-initially' 	how #if, #ifdef, #ifndef, #else, a			
	'hide-ifdef-read-only'	e-ifdefs' should be called wher			
		ant to make buffers read only vad-only status is restored to pre			
Toggle the Hide-Ifdef mode :	• <f12> M-# • M-<f12> M-#</f12></f12>	(hide-ifdef-mode &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/show code suppressed by C	* <f12> M-# * <f12> <f7> #</f7></f12></f12>	,	Hide-Ifdef mode is a buffer-local minor mode for use with C and C-like major modes. When enabled, code within #ifdef constructs that the C preprocessor would eliminate may be		
preprocessor	• <f11> SPC c M-#</f11>		hidden from view.		
Toggle read-only mode when text is	• C-c @ C-q • <f12> # r</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		
hidden	* <f12> <f7> R</f7></f12>		the 'hide-ifdef-read-only' user option to t.		
Toggle shadowing of hidden text.	• C-c @ C-w • <f12> # w</f12>	(hide-ifdef-toggle- shadowing)	Toggle shadowing. • When shadowing is on, text that would be hidden is "shadowed" instead: it is displayed with		
	* <f12> <f7> W</f7></f12>		the shadow face (normally something dim, all depending of the theme used).		
Hide code suppressed by C	• C-c @ h • <f12> # H</f12>	(hide-ifdefs &optional NOMSG)	Hide the contents of some #ifdefs. • Assume that defined symbols have been added to 'hide-ifdef-env'.		
preprocessor	• M- <f12> # H</f12>	, remes,	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.		
	* <f12> <f7> H - <f11> SPC c # H</f11></f7></f12>		With prefix command presents it will also hide the #ifdefs themselves. Turn off hiding by calling 'show-ifdefs'.		
Restore all hidden	• C-c @ s	(show-ifdefs)	Cancel the effects of 'hide-ifdef': show the contents of all #ifdefs.		
into view	• <f12> # S * <f12> <f7> S</f7></f12></f12>				
Hide part of current	• C-c @ C-d	(hide-ifdef-block &optional	Hide the ifdef block (true or false part) enclosing or before the cursor.		
block that would not be included	• <f12> # h * <f12> <f7> h</f7></f12></f12>	ARG START END)	With optional prefix argument ARG, also hide the #ifdefs themselves.		
Show all parts of the	• C-c @ C-s		Show the ifdef block (true or false part) enclosing or before the cursor.		
current #ifdef block	• <f12> # s * <f12> <f7> s</f7></f12></f12>	START END)			
Set a variable to a	• C-c @ d	(hide-ifdef-define VAR	Define a VAR to VAL (default 1) in 'hide-ifdef-env'.		
specific value	• <f12> # d * <f12> <f7> d</f7></f12></f12>	&optional VAL)	 This allows hiding the block inside #ifndef VAR (or the equivalent) by executing the command hide-ifdefs. 		
Undefine a variable	• C-c @ u	(hide-ifdef-undef START	Undefine a VAR		
	• <f12> # u * <f12> <f7> u</f7></f12></f12>	END)	 This allows hiding the blocks inside #ifdef VAR (or the equivalent) by executing the command hide-ifdefs. 		
Save the symbol	• C-c @ D	(hide-ifdef-set-define-alist	Save the state of the current hide-ifdev-env to a list with the specified NAME for later re-use.		
environment list into a named list	• <f12> # D * <f12> <f7> D</f7></f12></f12>	NAME)	The value is saved inside the hide-ifdef-define-alist variable. 1. 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>	(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.		
	* <f12> <f7> U</f7></f12>				
Clear the complete list of #define'd	• C-c @ C • <f12> # C</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		
symbols inside 'hide- ifdef-env'	* <f12> * C</f12>		clearance.		
Evaluate pre-	• C-c @ e	(hif-evaluate-macro	Evaluate the macro expansion result for the active region.		
processor macro	• <f12> # e * <f12> <f7> e</f7></f12></f12>	RSTART REND)	If no region active, find the current #ifdefs and evaluate the result. Only supports math calculations; strings or argumented macros can not be expanded.		
Rendering markup		used to create images from spe	ecific markup code embedded inside C source code comments. This can be useful when using		
embedded in	these markup languages to describe UML diagrams or finite-state machines for example.				
comments	You can also use Graphviz, se		Developing Distribution of the Control of the Contr		
Preview UML diagram from plantUML	<f12> u</f12>	(pel-render-commented- plantuml PREFIX &optional	Render the PlantUML markup embedded in current mode comment. • Use region if identified otherwise use PlantUML block at point.		
source in current plantUML region of		POS)	 Uses prefix (as PREFIX) to choose where to display it: 4 (when prefixing the command with C-u) -> new window 		
commented source code			 16 (when prefixing the command with C-u C-u) -> new frame. else -> new buffer 		
See also: M PlantUML			This can be used inside buffer using any major mode, when PlantUML markup is embedded inside source code comment.		
			subsethis 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 <u>plantuml-mode</u> external package, activated by <u>pel-use-plantuml</u> user option being non-nil.		
		1	. •		

<u>Description</u>	<u>Keystroke</u>	Function	<u>Note</u>
C Specific search and replace	boolean value to true or false.		blace 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 explicitly uses the keyword.
Problematic code	Problem: C code that compa	re pointer against NULL and va	alue against TRUE, true, FALSE, and false.
Search for poor code	<f12> s n</f12>	(pel-c-search-equal_NULL)	Move point to the next expression like if (ptr == NULL) or if (NULL == ptr)
using comparison against NULL	<f12> s N</f12>	(pel-c-search-not- equal_NULL)	Move point to the next expression like if (ptr != NULL) or if (NULL != ptr)
Search for poor code using comparison	<f12> s f</f12>	(pel-c-search-equal_false)	Move point to the next expression like if (boolean == false) or if (false == boolean). Also search for FALSE.
against false or FALSE	<f12> s F</f12>	(pel-c-search-not- equal_false)	Move point to the next expression like if (boolean != false) or if (false != boolean). Also search for FALSE.
Search for poor code using comparison	<f12> s t</f12>	(pel-c-search-equal_true)	Move point to the next expression like if (boolean == true) or if (true != boolean). Also search for TRUE
against true or TRUE	<f12> s T</f12>	(pel-c-search-not- equal_true)	Move point to the next expression like if (boolean != true) or if (true != boolean). Also search for TRUE
Search for any of the poor code listed in the previous 6 commands	<f12> s *</f12>	(pel-c-search-any- comparison-problem	Move point to the next instance of any of the expressions searched by the 6 commands above.
Improve C/C++ code: remove explicit comparisons against NULL, TRUE, FALSE, true and false	<f12> s C-f</f12>	(pel-c-fix-comparison-problems)	Replace all instances of C/C++ code that explicitly compares a pointer against NULL or a boolean value against true, false, TRUE and FALSE by the logically equivalent expression that does not use the keyword: For example this replaces: if (pointer == NULL) by if (!pointer) if (value == TRUE) by if (value) if (value == FALSE) by if (!value) if (value == false) by if (value) if (value == false) by if (value) if (value == false) by if (!value) if (value == false) by if (!value) if (value == TRUE) by if (pointer) if (value != TRUE) by if (value) if (value != TRUE) by if (value) if (value != false) by if (value) if (value != false
Problematic code	• Instead of: #if VAR • Instead of: #if VAR ==		
Search for poor pre- processor conditional #if VAR	<f12> s #</f12>	(pel-c-search-preproc-if)	Move point to the end of the next #if VAR expression.
Search for poor pre- process conditional #if VAR==0 #if VAR==1	<f12> s 0</f12>	(pel-c-search-preproc-if-set)	Move point to the end of the next #if VAR == 0 expression or #if VAR == 1 expression.
Improve C/C++ code: remove explicit comparisons against NULL, TRUE, FALSE, true and false	<f12> s C-p</f12>	(pel-c-fix-preproc-if- problems)	Inside current buffer, replace all instances of problematic C pre-processor conditional code listed below with the corresponding safer code. • Instead of: #if VAR
Programming Help	PEL has bindings for the follow	wing commands that are useful	when editing source code, markup files or any file that has a mode that supports imenu.
Show what completion mode is currently used.	<f11> M-c ?</f11>	(pel-show-active-completion-mode)	Display the completion mode currently used.
Show function at point	<f11> ? F</f11>	(pel-show-function)	Display the name of the current "function" at point in the mini-buffer.
Toggle which- function-mode to display name of	• <f11> ? f • <f11> M-d f</f11></f11>	(which-function-mode &optional ARG)	Toggle mode line display of current function (Which Function mode). • With a prefix argument ARG, enable Which Function mode if ARG is positive, and disable it otherwise.
current function at point See also: • ∑ Menus • ∑ Mode Line The concept of "function" is major mode specific. For example, in C++ mode, if point is inside a class definition it shows the name of the class.			

Emacs & C - References

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