



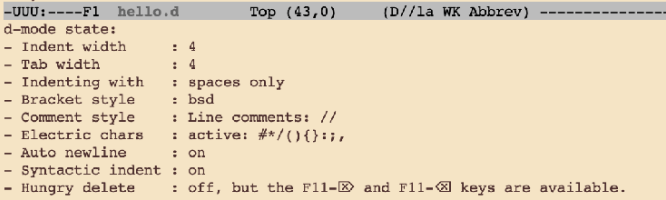





























Emacs Support for D

Description	Keystroke	Function	Note
Support for the D programming language	<div> Emacs supports the D programming language via a the d-mode external package. This package extends the Emacs CC Mode built-in package which supports the curly-bracket programming languages like D. Other external packages provide other features for working with D, they are listed in the reference table below.</div> <div> PEL activates D support via the customize user option variable pel-use-d. It must be set to t to activate support for D.</div> <div> Important aspects of D source code syntax controlled by the CC Mode are customizable with PEL user option variables. PEL customization for D: Simplifies configuration for editing D source code (To change, execute: M-x customize-group pel-pkg-for-d): Emacs customization group: pel-pkg-for-d<ul style="list-style-type: none">pel-d-indentation: Identifies the number of columns used for indentation. Defaults to 4.pel-d-tab-width: The width of a tab. Defaults to 4. This concept differs from indentation: you can have an indentation of 4 and tab width of 8: M-i will move point to columns that are multiple of 8 <tab> will indent to a column that is a multiple of 4.pel-d-use-tabs: Whether hard tabs are used in indentation or not: t: tabs are used, nil: only spaces are used. Default: nil.pel-d-backet-style: The bracket/indentation style supported by the electric keys. One of the values supported by Emacs (also possible to define your own with Elisp code). Default to "bsd".Emacs customization group: pel-pkg-for-cc. Applies to all CC Mode related modes (like d-mode).<ul style="list-style-type: none">pel-cc-auto-newline: Whether automatic newline mode is active on all CC Mode (including d-mode).<p>The values for those user option variables can also be stored inside directory local files and even as file local variables. You can also modify them for each buffer and view their current settings using the commands listed in the following set of rows. None of the commands below change PEL default; they change the value for the current buffer only.</p><ul style="list-style-type: none">PEL provides the following set of mode-specific key prefixes: <f11> SPC D, <f12> and <M-f12> The first one is always available. The other two prefixes are only available when the current buffer is in d-mode. The <M-f12> prefix helps the typing flow when the next key is a Meta key. For simplification, the <f11> SPC D prefix is normally omitted in the table.</div>		
Customize PEL D Support (See also:  Customize)	<ul style="list-style-type: none"><f11> <f1> SPC D<f12> <f1>	(pel-cfg-pkg-d &optional OTHER-WINDOW)	Customize PEL D support. <ul style="list-style-type: none">If OTHER-WINDOW is non-nil (use C-u), display in another window and open the d-mode customize group as well.The <f12> <f1> binding is available when point is in a buffer visiting a D file.
CC Mode Style Management	Automatic indentation is done by the CC Mode according to its syntactic interpretation of the current line and the indentation mode in use. You can impose an indentation style by customization. But you may use source code written by others and want to continue using the same style. In those cases you can use CC Mode's ability to analyze the style and report it or start using it (installing it) with the following commands. Not all commands are documented here, see the CC Mode manual for more info.		
Guess the style used in the current buffer, do not install it	M-x c-guess-buffer-no-install	(c-guess-buffer-no-install &optional ACCUMULATE)	Guess the style on the whole current buffer; don't install it. <ul style="list-style-type: none">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	M-x c-guess-buffer	(c-guess-buffer &optional ACCUMULATE)	Guess the style on the whole current buffer, and install it. <ul style="list-style-type: none">The style is given a name based on the file's absolute file name.If given a prefix argument (or if the optional argument ACCUMULATE is non-nil) then the previous guess is extended, otherwise a new guess is made from scratch.
Guess style in the region	M-x c-guess	(c-guess &optional ACCUMULATE)	Guess the style in the region up to 'c-guess-region-max', and install it. <ul style="list-style-type: none">The style is given a name based on the file's absolute file name.If given a prefix argument (or if the optional argument ACCUMULATE is non-nil) then the previous guess is extended, otherwise a new guess is made from scratch.
Guess the style of a region	M-x c-guess-region	(c-guess-region START END &optional ACCUMULATE)	Guess the style on the region and install it. <ul style="list-style-type: none">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.
View Guessed style	M-x c-guess-view	(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. <ul style="list-style-type: none">"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.
Determine syntactic context of current line.	M-x c-guess-basic-syntax	(c-guess-basic-syntax)	Determine the syntactic context of the current line.
Show/Modify syntactic context	C-c C-o	(c-set-offset SYMBOL OFFSET &optional IGNORED)	Change the value of a syntactic element symbol in 'c-offsets-alist'. <ul style="list-style-type: none">SYMBOL is the syntactic element symbol to change and OFFSET is the new offset for that syntactic element. The optional argument is not used and exists only for compatibility reasons.
Show syntactic information for current line	C-c C-s	(c-show-syntactic-information ARG)	Show syntactic information for current line. <ul style="list-style-type: none">With universal argument, inserts the analysis as a comment on that line.
CC Mode support	The following commands are CC Mode specific, available for each of the programming languages similar that have a mode derived from CC Mode like D. The CC Mode controls the indentation and bracket style which controls what happens when electric characters are typed (when the electric mode is activated) and provide a better experience when editing D source code.		
Display current Mode settings	<ul style="list-style-type: none"><f12> M-?<M-f12> M-?<f11> SPC D M-?	(pel-cc-mode-info)	Display information about current CC mode derivative for the current buffer. <ul style="list-style-type: none">Example of the information displayed (which reflects PEL's defaults):<div></div>
Toggle Electric state	<ul style="list-style-type: none">C-c C-1<f12> M-e<M-f12> M-e	(c-toggle-electric-state &optional ARG)	Toggle the electric indentation feature done with the electric character keys. <ul style="list-style-type: none">Optional numeric ARG, if supplied, turns on electric indentation when positive, turns it off when negative, and just toggles it when zero or left out.
Toggle auto-newline insertion mode	<ul style="list-style-type: none">C-c C-a<f12> M-RET<M-f12> M-RET	(c-toggle-auto-newline &optional ARG)	Toggle auto-newline feature. <ul style="list-style-type: none">Optional numeric ARG, if supplied, turns on auto-newline when positive, turns it off when negative, and just toggles it when zero or left out.Turning on auto-newline automatically enables electric indentation.When the auto-newline feature is enabled (indicated by "/la" on the mode line after the mode name) newlines are automatically inserted after special characters such as brace, comma, semi-colon, and colon. <div> Emacs allows customizing the style and how automatic newlines are used. See the CC Mode Manual section: Customizing Auto-newlines.</div>
Set indentation style	<ul style="list-style-type: none">C-c .<f12> M-s<M-f12> M-s	(c-set-style STYLENAME &optional DONT-OVERRIDE)	Set the bracket/indentation style for the current buffer. <ul style="list-style-type: none">Prompts for the name.Supports tab completion (so use tab to see the list). Can be one of the values supported by Emacs but you can also add your customized mode with some Emacs Lisp code.

Description	Keystroke	Function	Note
Toggle syntactic indentation	<ul style="list-style-type: none"> <f12> M-i <M-f12> M-i 	(c-toggle-syntactic-indentation &optional ARG)	Toggle syntactic indentation. <ul style="list-style-type: none"> Optional numeric ARG, if supplied, turns on syntactic indentation when positive, turns it off when negative, and just toggles it when zero or left out. When syntactic indentation is turned on (the default), the indentation functions and the electric keys indent according to the 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'.
Electric Keys and Keywords	The following characters have special meaning when the electrical state is active in a buffer using d-mode.		
	#	(c-electric-pound ARG)	Insert a "#". <ul style="list-style-type: none"> If 'c-electric-flag' is set, handle it specially according to the variable 'c-electric-pound-behavior', which can only be nil or 'alignleft'. If a numeric ARG is supplied, or if point is inside a literal or a macro, nothing special happens. D does not use the pound character much. It only uses it for #line statements.
	<ul style="list-style-type: none"> () 	(c-electric-paren ARG)	Insert a parenthesis. <ul style="list-style-type: none"> If 'c-syntactic-indentation' and 'c-electric-flag' are both non-nil, the line is reindented unless a numeric ARG is supplied, or the parenthesis is inserted inside a literal. Whitespace between a function name and the parenthesis may get added or removed; see the variable 'c-cleanup-list'. 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'.
	<ul style="list-style-type: none"> { } 	(c-electric-brace ARG)	Insert a brace. <ul style="list-style-type: none"> If 'c-electric-flag' is non-nil, the brace is not inside a literal and a numeric ARG hasn't been supplied, the command performs several electric actions: <ol style="list-style-type: none"> If the auto-newline feature is turned on (indicated by "/la" on the mode line) newlines are inserted before and after the brace as directed by the settings in 'c-hanging-braces-alist'. Any auto-newlines are indented. The original line is also reindented unless 'c-syntactic-indentation' is nil. If auto-newline is turned on, various newline cleanups based on the settings of 'c-cleanup-list' are done.
	:	(c-electric-colon ARG)	Insert a colon. <ul style="list-style-type: none"> If 'c-electric-flag' is non-nil, the colon is not inside a literal and a numeric ARG hasn't been supplied, the command performs several electric actions: <ol style="list-style-type: none"> If the auto-newline feature is turned on (indicated by "/la" on the mode line) newlines are inserted before and after the colon based on the settings in 'c-hanging-colons-alist'. Any auto-newlines are indented. The original line is also reindented unless 'c-syntactic-indentation' is nil. If auto-newline is turned on, whitespace between two colons will be "cleaned up" leaving a scope operator, if this action is set in 'c-cleanup-list'.
	<ul style="list-style-type: none"> ; , 	(c-electric-semi&comma ARG)	Insert a comma or semicolon. <ul style="list-style-type: none"> If 'c-electric-flag' is non-nil, point isn't inside a literal and a numeric ARG hasn't been supplied, the command performs several electric actions: <ol style="list-style-type: none"> When the auto-newline feature is turned on (indicated by "/la" on the mode line) a newline might be inserted. See the variable 'c-hanging-semi&comma-criteria' for how newline insertion is determined. Any auto-newlines are indented. The original line is also reindented unless 'c-syntactic-indentation' is nil. If auto-newline is turned on, a comma following a brace list or a semicolon following a defun might be cleaned up, depending on the settings of 'c-cleanup-list'.
D comments	2 more characters have electric behaviour: / and * to help support comments in D. D supports the following types of comments (only the first 2 are explicitly supported by Emacs): <ul style="list-style-type: none"> Block Comments: <code>/* comment */</code> Line Comments: <code>// comment to end of line</code> Nesting Block Comments: <code>/* nesting */+comments +/- can span multiple lines and surround // style comment +/</code> Documentation Comments: Use several prefixes: <code>///</code> , <code>/** multi-line documentation */</code> , and <code>/** multi-line documentation */</code> 		
	/	(c-electric-slash ARG)	Insert a slash character. <ul style="list-style-type: none"> 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: <ol style="list-style-type: none"> The slash is second of a "/" line oriented comment introducing token and we are on a comment-only-line, or 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.
	*	(c-electric-star ARG)	Insert a star character. <ul style="list-style-type: none"> 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 it becomes easy to type the following two styles of multi-line block comment: <pre> /* Two star ** continuation ** prefix for ** multi-line ** C comment. */ /* Single star * prefix for * multi-line * C comment. */ </pre> 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.
Toggle Comment Style	<ul style="list-style-type: none"> C-c C-k <f12> M-; <M-f12> M-; 	(c-toggle-comment-style &optional ARG)	Toggle the comment style between block and line comments. <ul style="list-style-type: none"> Optional numeric ARG, if supplied, switches to block comment style when positive, to line comment style when negative, and just toggles it when zero or left out. ⚠️ Only the <code>//</code> and <code>/* */</code> styles are supported. The <code>/*+*/</code> comments are not supported. 🙌 This is part of CC Mode. Use <code><f12> M-?</code> to display the current state.

Description	Keystroke	Function	Note
Comment/un-comment	M-;	(comment-dwim ARG)	<p>Comment line or region with <code>//</code> or <code>/* */</code> style comments depending on the comment style currently used in the buffer.</p> <ul style="list-style-type: none"> When no marked region and no comment: <ul style="list-style-type: none"> 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: <ul style="list-style-type: none"> Comment region (each line is commented) With marked commented region: <ul style="list-style-type: none"> removes the comment. <ul style="list-style-type: none"> Call the comment command you want (Do What I Mean). <ul style="list-style-type: none"> If the region is active and 'transient-mark-mode' is on, call 'comment-region' (unless it only consists of comments, in which case it calls 'uncomment-region'). Else, if the current line is empty, call 'comment-insert-comment-function' if it is defined, otherwise insert a comment and indent it. Else if a prefix ARG is specified, call 'comment-kill'. Else, call 'comment-indent'. You can configure 'comment-style' to change the way regions are commented: see <F12> M-; to toggle the comment style.
	C-c C-c	(comment-region BEG END &optional ARG)	<p>Comment or uncomment each line in the region.</p> <ul style="list-style-type: none"> 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. <p>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'.</p> <p>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'.</p> <p>👉 If you try this when no region is marked and the <code>/* */</code> style comments is active, the comment ends on the next space, which is probably not what you want. The command comment-dwim works better.</p>
Fill current paragraph (See also: Σ Filling/Justification)	<ul style="list-style-type: none"> M-q <f12> F <M-f12> F <f11> SPC D F 	(c-fill-paragraph &optional ARG)	<p>Like <f11> t f p but handles <code>//</code> and <code>/* */</code> style comments.</p> <ul style="list-style-type: none"> 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: <pre>char description[] = "\ A very long description of something that you want to fill to make nicely formatted output.";</pre> <ul style="list-style-type: none"> If point is in any other situation, i.e. in normal code, do nothing. Optional prefix ARG means justify paragraph as well.
Toggle subword-mode (See also: Σ Text Modes)	<ul style="list-style-type: none"> <f11> t m b <f12> M-b <M-f12> M-b <f11> SPC D M-b 	(subword-mode &optional ARG)	<p>Toggle subword-mode: a minor mode that treats sections of <code>camelCase</code> and <code>PascalCase</code> as distinct words.</p> <ul style="list-style-type: none"> With a prefix argument ARG, enable Subword mode if ARG is positive, and disable it otherwise. <p>👉 Since D naming convention promotes the use of <code>camelCase</code> for functions, enums, constants and variables and <code>PascalCase</code> for types, using the subword-mode allows you to move into, delete, transpose the section of the words with the corresponding word commands.</p>
Hungry Deletion of Whitespace	<p>The CC mode provides two commands that can perform “hungry whitespace deletion” that can also be used in every mode.</p> <ul style="list-style-type: none"> 👉 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. <ul style="list-style-type: none"> When the Hungry Delete Mode is on, the mode-line displays a 'h' to the right of the <code>'/I'</code> 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 D). 		
Delete preceding char or all preceding whitespace. (See also: Σ Cut & Paste)	<ul style="list-style-type: none"> C-c DEL C-c  C-c C- C-c <C-backspace> C-c C-DEL <f11>  	(c-hungry-delete-backwards)	<p>Delete the preceding character or all preceding whitespace back to the previous non-whitespace character.</p> <p>🖥️ 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.</p> <p>👉 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.</p>
Delete next char or all following whitespace. (See also: Σ Cut & Paste)	<ul style="list-style-type: none"> C-c C-d C-c  C-c C- C-c <C-delete> <f11>  	(c-hungry-delete-forward)	<p>Delete the following character or all following whitespace up to the next non-whitespace character.</p> <p>🖥️ 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.</p> <p>👉 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.</p>
Toggle Hungry Delete mode	<ul style="list-style-type: none"> <f12> M-DEL <M-f12> M-DEL 	(c-toggle-hungry-state &optional ARG)	<p>Toggle hungry-delete-key feature. Affect and C-d keys.</p> <ul style="list-style-type: none"> Optional numeric ARG, if supplied, turns on hungry-delete when positive, turns it off when negative, and just toggles it when zero or left out. When the hungry-delete-key feature is enabled (indicated by <code>"/h"</code> on the mode line after the mode name) the delete key gobbles all preceding whitespace in one fell swoop. <p>👉 This is part of CC Mode. Use <f12> M-? to display the current state.</p>
Indentation	<p>All syntactic indentation control for D is controlled by the CC-Mode logic and provided commands listed below.</p> <ul style="list-style-type: none"> Rigid indentation commands are also available and listed at the end of this list. They are also listed in the Σ Indentation table. 		

Description	Keystroke	Function	Note
Insert an indented line below current line (See also: ↗Indentation)	<ul style="list-style-type: none"> M-RET <f11> <tab> RET 	(pel-newline-and-indent-below)	Insert an indented line just below current line regardless of the position of point. So 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.
Marking	Emacs provides the following command to quickly mark the whole content of the current function. More mark commands exists, see the ↗Marking table.		
Mark the complete function body (See also: ↗Marking)	C-M-h	(c-mark-function)	Mark complete function. <ul style="list-style-type: none"> Put mark at end of the current top-level declaration or macro, point at beginning. If point is not inside any then the closest following one is chosen. Each successive call of this command extends the marked region by one function. A mark is left where the command started, unless the region is already active (in Transient Mark mode). As opposed to C-M-a and C-M-e, this function does not require the declaration to contain a brace block.
Highlighting blocks	The following commands can be used to activate or toggle useful modes to highlight blocks of <code>()</code> , <code>{}</code> , and <code>[]</code> . <ul style="list-style-type: none"> show-paren-mode, which highlights the parens that matches the one before or after point. rainbow delimiters mode, where matching nested parens are highlighted with the same colour. 		
Toggle show-paren mode on/off (see also: ↗Highlight)	<ul style="list-style-type: none"> <f12> M-9 <f11> SPC D M-9 <f11> b h ((show-paren-mode &optional ARG)	Toggle visualization of matching parens (Show Paren mode). <ul style="list-style-type: none"> With a prefix argument ARG, enable Show Paren mode if ARG is positive, and disable it otherwise. Show Paren mode is a global minor mode. When enabled, any matching parenthesis is highlighted in 'show-paren-style' after 'show-paren-delay' seconds of Emacs idle time.
Enable/Disable coloured highlight of nested blocks <code>()</code>,<code>{}</code>,<code>[]</code> (see also: ↗Highlight)	<ul style="list-style-type: none"> <f12> M-r <f11> SPC D M-r <f11> b h R 	(rainbow-delimiters-mode &optional ARG)	Highlight nested parentheses, brackets, and braces with different colours according to their depth. <ul style="list-style-type: none"> Customize the depth and colours with M-x customize-group rainbow-delimiters <div>  Requires: rainbow-delimiters.el </div> <div>  PEL activates this when the pel-use-rainbow-delimiters customize variable is set to t. </div>
Navigation in D (See also: ↗Navigation)	Emacs provides commands to navigate across source of curly bracket programming languages like D. Most commands are specialization of the normal navigation commands which are described in the table ↗Navigation , along with the other commands that are also available. The list below describe the specialized commands only. See the others inside ↗Navigation , like the navigation by blocks, very useful in D.		
Go to beginning of statement	M-a	(c-beginning-of-statement &optional COUNT LIM SENTENCE-FLAG)	Go to the beginning of the innermost statement. <ul style="list-style-type: none"> 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	M-e	(c-end-of-statement &optional COUNT LIM SENTENCE-FLAG)	Go to the end of the innermost statement. <ul style="list-style-type: none"> 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.
Backward to beginning of current top-level function or struct	C-M-a	(c-beginning-of-defun &optional ARG)	Move backward to the beginning of a defun. <ul style="list-style-type: none"> Every top level declaration that contains a brace paren block is considered to be a defun. With a positive argument, move backward that many defuns. A negative argument -N means move forward to the Nth following beginning.
	<ul style="list-style-type: none"> C-M-<home> <f6> p <f6> <up> 	(beginning-of-defun &optional ARG)	Move backward to the beginning of a defun. <ul style="list-style-type: none"> With ARG, do it that many times. Negative ARG means move forward to the ARGth following beginning of defun. <div>  Shift marking is available in graphics mode, not in terminal mode (for C-M-a and C-M-<home>). However <f6> p handles Shift-marking fine in terminal mode. </div> <div>  This command moves to the beginning of the next function or of the same nesting level of the current location. It skips the functions and methods that are more deeply nested. </div>
Forward to end of current top-level function or struct.	C-M-e	(c-end-of-defun &optional ARG)	Move forward to the end of a top level declaration. <ul style="list-style-type: none"> With argument, do it that many times. Negative argument -N means move back to Nth preceding end.
	<ul style="list-style-type: none"> C-M-<end> <f6> <right> 	(end-of-defun &optional ARG)	Move forward to next end of defun. <div>With argument, do it that many times. Negative argument -N means move back to Nth preceding end of defun.</div> <div>  Shift marking is available in graphics mode, not in terminal mode (both keys). </div> <div>  This command moves to the end of the next top-level function or class. It skips the nested functions and methods. </div>
Forward to start of next top level function or struct	<ul style="list-style-type: none"> <f6> n <f6> <down> 	(pel-beginning-of-next-defun &optional SILENT DONT-PUSH_MARK)	Move forward to the beginning of the next function definition. <ul style="list-style-type: none"> 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. <ul style="list-style-type: none"> Move back to previous position with M-`. <div>  Shift marking is available. </div> <div>  This command complements what end-of-defun does. </div> <ul style="list-style-type: none"> 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. It handles nested functions or class methods in languages like Python and others.
Backward to end of previous top level function or struct	<f6> <left>	(pel-end-of-previous-defun &optional SILENT DONT-PUSH_MARK)	Move backwards to the end of the previous function definition. <ul style="list-style-type: none"> 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. <ul style="list-style-type: none"> Move back to previous position with M-`. <div>  Shift marking is available. </div> <div>  This command complements this set of 4 commands. </div> <div>  In some cases it fails to detect the end of the previous block and fails.  </div>
Rendering markup embedded in comments	The following commands are used to create images from specific markup code embedded inside D source code comments. This can be useful when using these markup languages to describe UML diagrams or finite-state machines for example.		
Preview UML diagram from plantUML source in current plantUML region of commented source code (See also: ↗PlantUML)	<f12> u	(pel-render-commented-plantuml PREFIX &optional POS)	Render the PlantUML markup embedded in current mode comment. <ul style="list-style-type: none"> Use region if identified otherwise use PlantUML block at point. Uses prefix (as PREFIX) to choose where to display it: <ul style="list-style-type: none"> 4 (when prefixing the command with C-u) -> new window 16 (when prefixing the command with C-u C-u) -> new frame. else -> new buffer This can be used inside buffer using any major mode, when PlantUML markup is embedded inside source code comment. <div>  Use this in source code to describe your code architecture with PlantUML markup, then generate the UML rendering by moving point inside the PlantUML block and issuing this command. </div> <div>  Requires the plantuml-mode external package,  activated by pel-use-plantuml user option being non-nil. </div>

Description	Keystroke	Function	Note
Preview diagram created from Graphviz DOT markup embedded in comments (See also: <code>\JGraphviz Dot</code>)	<code><f12> G</code>	<code>(pel-render-commented-graphviz-dot &optional POS)</code>	Render the Graphviz-Dot markup embedded in current mode comment. Search at POS if specified, otherwise search around point. Use region if identified otherwise use Graphviz-Dot block. 🖱️ The graphviz DOT code must be located within a block delimited by the following special keywords (that are also in comments): <ul style="list-style-type: none"> <code>@start-gdot</code> <code>@end-gdot</code> ⚠️ The current implementation leaves the created image file in a temporary directory. You will probably want to move that file or delete it, otherwise the size of this directory will increase with each of these created files. The file names use the pel-gdot- prefix. 📦 Requires the graphviz-dot-mode package external package,  activated by pel-use-graphviz-dot user option set to <code>t</code> .

Emacs & D— References

Document	Notes
The D Programming Language	
D (programming language) - Wikipedia	Overview of D
D Home Page	
D Home Page - Documentation	Links to the Language Reference , Library Reference , Command-line Reference , Feature Overview and Articles .
DUB - The D Package Registry	Browsable/searchable list of packages
The D Style - D Code Guideline	This document provides a set of style conventions promoted by the D community. Several items in this guideline identify stylistic aspects that can be configured in Emacs. Some of them are listed here: <ul style="list-style-type: none"> Indentation: <ul style="list-style-type: none"> spaces instead of tabs ➤ indentation level: 4 columns ➤ c-basic-offset = 4 Line Length : soft limit of 80, hard limit of 120. They can exceed 80 columns but never 120. Brackets style: <ul style="list-style-type: none"> Use the Allman style (also called BSD style) where each brace is on their own line. ➤ add: (d-mode . “bsd”) to c-default-style Whitespace in statements: <ul style="list-style-type: none"> 1 space after for, foreach, if, while and the version keyword and the opening parenthesis: <code>if (x) { ... }</code> 1 space between binary operators, assignments, casts, lambdas. No space between unary operators, after assert, function calls, function definition name. Naming Conventions: <ul style="list-style-type: none"> Constant, enums, variable and function names should be camelCased. User defined type names should be PascalCased.
The Next Big Programming Language You've Never Heard Of WIRED - 2014	D is a very nice language, unfortunately it never got the attention could have got if it had some big corporate backup. Interview with Andrei Alexandrescu discussing his encounter with Walter Bright and the D language.
Emacs Support for D	Support for D for Emacs is based on: <ul style="list-style-type: none"> Emacs D Mode Code completion support that uses: <ul style="list-style-type: none"> A completion front end, either: <ul style="list-style-type: none"> Auto-Complete based using ac-dcd. Company based using company-dcd. Both of these depend on flycheck-dmd-dub, which uses DCD, the D Completion Daemon, written in D. Both require/use flycheck D Unit test support: flycheck-d-unittest
Emacs D Mode	The main support for D. Available on MELPA as d-mode . The d-mode is based on cc-mode.
ac-dcd : Auto Complete D Code Completion via DCD backend	Available on MELPA as ac-dcd . • This project also recommend using yasnippet and popwin .
Company-DCD - Company D Code Completion via DCD backend	Available on MELPA as company-dcd . • DCD is the D Completion Daemon (DCD @ Github). • For Emacs customization of company-dcd, see the company-dcd Emacs customization group (use <code><f11> <f1> G company-dcd</code>)
flycheck-dmd-dub	Available from melba as flycheck-dmd-dub . • Flycheck support for D: reads D library dependency information from DUB (the D Package Registry). • To use it you must install DCD separately: see next row.
DCD: D Completion Daemon	<ul style="list-style-type: none"> DCD instruction installation on the DCD Github page. See also the DUB DCD page which has the same info as GitHub but also has internal documentation of the D code interfaces down to the source code. On macOS, the dcd-client and dcd-server commands can be installed with Homebrew.
D Unit Test support: flycheck-d-unittest	Available on MELPA as flycheck-d-unittest . • Runs D unit test with “dmd -unittest and -main options”. • Takes advantage that D has built-in syntax and dmd support for unit test builds and runs. • The project has a wiki page, “ Start D with Emacs ”, describing how to install Emacs support for D (but only describes d-mode and flycheck-d-unittest)
yasnippets for D	I have found the following: <ul style="list-style-type: none"> Per Nordlöw snippets for D
Emacs Support for Curly Bracket Programming Languages	The d-mode is based on the CC Mode. The CC Mode, a collection of libraries, provides support for several curly-bracket programming languages like C, C++, Java, Objective-C, Pike, AWK and it also applies to D. Several features of the CC Mode are used for the D support, so it's useful to be aware of them.
GNU Emacs CC Mode Manual	D is a curly-bracket programming language and therefore supported by Emacs CC Mode. It controls: <ul style="list-style-type: none"> whether hard tabs or spaces are used: indent-tabs-mode the number of columns per tab: tab-width the indentation style (see indentation style meanings): c-default-style a-list with an entry for D PEL provides user options to activate the use of D in Emacs (pel-use-D) and user options for the tab, style to use and what CC modes are activated by default.
GNU Emacs Manual - C and Related Modes	The main Emacs manual also provides information on the support for C and similar programming languages, and these apply to the D programming language as well. The sections include: <ul style="list-style-type: none"> Motion in C Electric C Hungry Delete Other C Commands