Emacs support for D

Description	Keystroke	Function	Note
Support for D programming language	The commands and key bindings listed below are specific to editing D source code. • As for other programming languages and some markup languages PEL provides a special, mode-specific key-map assigned to the **<f12> k**ey that is activated when editing a file in d-mode. They are listed in this table. • Other keys are also available, several mapped to main keys such as comma and parentheses. Their meaning is also described.		
	#	(**c-electric-pound** ARG)	Insert a "#". • If 'c-electric-flag' is set, handle it specially according to the variable 'c-electric-pound-behavior'. If a numeric ARG is supplied, or if point is inside a literal or a macro, nothing special happens.
	• **(** • **)**	(**c-electric-paren** ARG)	Insert a parenthesis. • 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'.
	• **{** • **}**		
	,	(**c-electric-semi&comma** ARG)	Insert a comma or semicolon. 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: a) 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. b) Any auto-newlines are indented. The original line is also reindented unless 'c-syntactic-indentation' is nil. c) 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'.
	:		
	;		
Toggle auto-newline insertion mode	• **C-c C-a** • **<f12> a** • **<f11> SPC D a**	(**c-toggle-auto-newline** &optional ARG)	Toggle **auto-newline** feature. • Optional numeric ARG, if supplied, turns on auto-newline when positive, turns it off when negative, and just toggles it when zero or left out. • Turning on auto-newline automatically enables ***electric indentation***. • When the auto-newline feature is enabled (indicated by "/la" on the mode line after the mode name) newlines are automatically inserted after special characters such as brace, comma, semi-colon, and colon.
D comments	D supports 3 types of comments: • Block Comments: **/*** comment ***/** • Line Comments: **//** comment to end of line • Nesting Block Comments: **/+** nesting /+comments +/ can span multiple lines and surround // style comment **+/** • Documentation Comments: Use several prefixes: **///** , **/**** , and **/++**		
	/	(**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.

	*****	(**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 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.

	M-;	(**comment-dwim** ARG)	Comment line or region with // • When no marked region and no comment: • On empty line: insert **//** comment at the proper indentation level. Typed again: move it toward end of line. • On line with code: insert **//** after the code for an end-of-line comment • With marked un-commented region: • Comment region with **//** • With marked commented region: • removes the comment. Call the comment command you want (Do What I Mean). 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.

Description	Keystroke	Function	Note

Fill current paragraph (See also: ∑ Filling/ Justification)	• **M-q**
	• **<f12> f**
	• **<f11> SPC D f**
		(**c-fill-paragraph** &optional ARG)	Like **<f11> t f p** but handles C and C++ 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.

Toggle subword-mode (See also: ∑ Text Modes)	• **<f11> t m b** • **<f12> M-b** • **<f11> SPC D M-b**	(**subword-mode** &optional ARG)	Toggle subword-mode: a minor mode that treats sections of camelCase and PascalCase as distinct words. • With a prefix argument ARG, enable Subword mode if ARG is positive, and disable it otherwise. ☝ Since D naming convention promotes the use of camelCase for functions, enums, constants and variables and PascalCase for types, using the subword-mode allows you to move into, delete, transpose the section of the words with the corresponding word commands.
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 **<DEL>** 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 '//l' 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 the D Mode.		
Delete preceding char or all preceding whitespace. (See also: ∑ Cut & Paste)	• **C-c DEL** • **C-c** ⌫ • **C-c C-**⌫ • **C-c <C-backspace>** • **C-c C-DEL** • **<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.
Delete next char or all following whitespace. (See also: ∑ Cut & Paste)	• **C-c C-d** • **C-c** ⌦ • **C-c C-**⌦ • **C-c <C-delete>** • **<f11>** ⌦	(**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.
Toggle Hungry Delete mode	• **<f12> M-DEL** • **<f11> SPC D M-DEL**	(**c-toggle-hungry-state** &optional ARG)	Toggle hungry-delete-key feature. Affect **<DEL>** and **C-d** keys. • Optional numeric ARG, if supplied, turns on hungry-delete when positive, turns it off when negative, and just toggles it when zero or left out. • When the hungry-delete-key feature is enabled (indicated by "/h" on the mode line after the mode name) the delete key gobbles all preceding whitespace in one fell swoop.

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:
	• **Indentation**:
	  • 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**:
	  • 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**:
	  • 1 space after **for**, **foreach**, **if**, **while** and **version** keyword and the opening parenthesis: `if (x) { ... }`
	  • 1 space between binary operators, assignments, casts, lambdas.
	  • No space between unary operators, after assert, function calls, function definition name.
	• **Naming Conventions**:
	  • 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.

Document	Notes

Emacs Support for D	Support for D for Emacs is based on:
	• Emacs D Mode
	• Code completion support that uses:
	  • A completion front end, either:
	  • 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. (require 'ac-dcd) (add-to-list 'ac-modes 'd-mode) (add-hook 'd-mode-hook #'ac-dcd-setup)
Company-DCD - Company D Code Completion via DCD backend	Available on MELPA as company-dcd. • DCD is the D Completion Daemon (DCD @ Github).
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 instructions 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 -uninttest 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: • Per Nordlöw snippets for D
Emacs CC Mode	The d-mode is based on the CC Mode. The CC Mode, a collection of libraries, provides support for several C-like 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	