GNU APL Reference Card

(for GNU APL version 1.8)

Use `C-x gnu-apl` to start GNU APL in Emacs.

Emacs mode

Interaction mode:

beginning of defun	`C-M-a`
end of defun	`C-M-e`
find function at point	`M-.`
apropos symbol	`C-c C-a`
edit function	`C-c C-f`
show help for symbol	`C-c C-h`
finnapl list	`C-c TAB`
show keyboard	`C-c C-k`
plot line	`C-c RET`
edit variable	`C-c C-v`
trace	`C-c C-.`

Edit mode:

go to beginning of defun	`C-M-a`
go to end of defun	`C-M-e`
find function at point	`M-.`
apropos symbol	`C-c C-a`
interactive send current function	`C-c C-c`
help for symbol	`C-c C-h`
finnapl list	`C-c TAB`
show keyboard	`C-c C-k`
interactive send buffer	`C-c C-l`
interactive send region	`C-c C-s`
switch to interactive	`C-c C-z`
trace	`C-c C-.`
indent	`C-M-q`

System

Notation for commands:

`F`	filename	`L`	library	`P`	path
`G`	logging facility	`O`	object	`S`	symbol
`W`	workspace				

APL standard commands

check workspace intergity	`)CHECK`
clear workspace	`)CLEAR`
save workspace as `CONTINUE` and exit	`)CONTINUE`
copies objects from given workspace	`)COPY [L] W [O ...]`
remove W	`)DROP [L] W`
dump W (readable, HTML escaped)	`)DUMP-HTML [[L] W]`
dump W (readable APL)	`)DUMP [[L] W]`
dump W (readable APL, verbose)	`)DUMPV [[L] W]`
erase symbol(s)	`)ERASE S ...`
show functions	`)FNS [from-to]`
help	`)HELP [primitive]`
history	`)HIST [CLEAR]`
runs command on host	`)HOST command ...`
loads workspace (IBM .atf format)	`)IN F [O ...]`
show libraries and paths	`)LIBS [[L] path]`
show saved workspaces	`)LIB [L|P] [from-to]`
load workspace W	`)LOAD [L] W`
show more error info	`)MORE`
lists symbols matching name	`)NMS [from-to]`
quit APL	`)OFF`
show operators	`)OPS [from-to]`
dump workspace (IBM .atf format)	`)OUT name [O ...]`
protects during copying	`)PCOPY [L] W [O ...]`
protects during loading	`)PIN F [O ...]`
quiet load	`)QLOAD [[L] W]`
reset state indicator	`)RESET`
save workspace as W	`)SAVE [[L] W]`
clear suspended functions	`)SIC`
see suspended functions and locals	`)SINL`
see suspended functions	`)SIS`
state indicator	`)SI`
show symbol count	`)SYMBOLS [count]`
show values in use by interpreter	`)VALUES`
show variables	`)VARS [from-to]`
get/set workspace ID	`)WSID [W]`

GNU extension commands (mostly for debugging)

toggles boxing of values when printing	`]BOXING [OFF|num]`
toggle colored output	`]COLOR [ON|OFF]`
dump W in HTML file	`]DOXY [path]`
expected error count in test suite	`]EXPECT error_count`
help	`]HELP [primitive]`
show keyboard layout	`]KEYB`
as)LIB, but shows fil eextensions	`]LIB [L|P] [from-to]`
show/set logging facilities	`]LOG [G [ON|OFF]]`
next testcase file	`]NEXTFILE`
performance statistics	`]PSTAT [CLEAR|SAVE]`
as)SIS, with more details	`]SIS`
as)SI, with more details	`]SI`
shared variables	`]SVARS`
describe internal details of symbol S	`]SYMBOL S`
define user command	`]USERCMD [ ... ]`
toggle output coloring on console	`]XTERM [ON|OFF]`

System variables:

character input/output	`□'`
evaluated input/output	`□`
account information	`□AI`
command line arguments	`□ARG`
atomic vector	`□AV`
comparison tolerance	`□CT`
event message	`□EM`
event type	`□ET`
format control	`□FC`
index origin (indexes start: 1, can be set to 0)	`□IO`
left argument	`□L`
line counters	`□LC`
latent expression (executed when workspace is loaded)	`□LX`
print precision (number of digits)	`□PP`
print style	`□PS`
print width (max characters in each printed line)	`□PW`
right argument	`□R`
random link	`□RL`
shared variable event	`□SVE`
system limits	`□SYL`
terminal control characters	`□TC`
time stamp (current time)	`□TS`
time zone (offset from GMT)	`□TZ`
user load	`□UL`
axis argument	`□X`
workspace available (bytes for workspace)	`□WA`
dfn axis argument	`X`
dfn result	λ
dfn left value arg	α
dfn left function arg	$\underline{\alpha}$
dfn right value arg	ω
dfn right function arg	$\underline{\omega}$

System functions:

atomic function	`□AF`
attributes	`□AT`
char representation	`□CR`
delay	`□DL`
D. Knuth's dancing links	`□DLX`
execute alternate	`□EA`
execute both	`□EB`
execute controlled	`□EC`
environment	`□ENV`
event simulate	`□ES`
expunge	`□EX`
fast Fourier transform	`□FFT`
file I/O	`□FIO`
FiX (FFI/call native functions)	`□FX`
Gtk GUI	`□GTK`
MAP ravel elements	`□MAP`
input from script	`□INP`
name association	`□NA`
name class	`□NC`
name list	`□NL`
plot a graph	`□PLOT`
regular expression, regex □RE string	`□RE`
random APL value	`□RVAL`
state indicator	`□SI`
SQL functions	`□SQL`
shared variable control	`□SVC`
shared variable offer	`□SVO`
shared variable query	`□SVQ`
shared variable retraction	`□SVR`
shared variable state	`□SVS`
STOP vector	`□STOP`
transfer form	`□TF`
TRACE vector	`□TRACE`
unicode character	`□UCS`

Notation

comment	`⍝`
statement separator	`⋄`
assignment	`A← ...`
assignment	`(A B C)← ... ... ...`
function definition	`∇`

zilde (empty vector)	⍬
a	`+ a`
a + b	`a + b`
- a	`- a`
a - b	`a - b`
magnitude of a	`| a`
b mod a	`a | b`
signal (-1, 0, +1)	`× a`
ab	`a × b`
1/a	`÷ a`
a/b	`a ÷ b`
floor of a	`⌊a`
min(a,b)	`a⌊b`
ceiling of a	`⌈a`
max(a,b)	`a⌈b`
e^a	`* a`
a^b	`a * b`
log(a)	`⍟ a`
log_b(a)	`b ⍟ a`
first n non-negative integers	`⍳ n`

a = b	`a = b`
a < b	`a < b`
a > b	`a > b`
a ≤ b	`a ≤ b`
a ≥ b	`a ≥ b`
expression max depth	`≡ a`
match (value and type)	`a ≡ b`
expression min depth	`≢ a`
not match	`a ≢ b`
not a	`~ a`
a or b	`a ∨ b`
a and b	`a ∧ b`
a nor b	`a ⍱ b`
a nand b	`a ⍲ b`
a ∈ b ?	`a ∈ b`
find a in b (binary index)	`a ⍷ b ?`
bitwise a or b	`a ⊤∨ b`
bitwise a and b	`a ⊤∧ b`
bitwise a nor b	`a ⊤⍱ b`
bitwise a nand b	`a ⊤⍲ b`
bitwise a ≠ b	`a ⊤≠ b`
bitwise a = b	`a ⊤= b`

a!	`!a`
$\binom{b}{a}$	`a!b`

aπ	`⍟a`
circle (trig) function	`a ⍟ b`
random integer in [1,a]	`?a`
a distinct random integers in [1,b]	`a?b`

makes a vector out of A	`, A`
append B to A	`A,B`
number of components in each dimension of A	`⍴ A`
array with shape A and data elements B	`A⍴ B`
inverse matrix of A	`⌹ A`
B^{-1}A (solution to $Bx = A$)	`A⌹B`
reverse elements of A (1^{st} index)	`⊖A`
rotate B by A positions	`A⊖B`
reverse elements of A (last index)	`⌽A`
rotate B by A positions (last index)	`A⌽B`
drop first A elements of B	`A↓B`
select first A elements of B	`A↑B`
intersection	`A∩B`
set (remove duplicates)	`∪A`
union	`A∪B`
identity	`⊢A`
take right hand side (B)	`A⊢B`
null	`⊣A`
take left hand side (A)	`A⊣B`
i-th element of A	`A[i]`
elements of A with indices i, j, k, ...	`A[i j k ...]`
element of A w/indices i, j, ... in 1^{st} dimension, k, l, ... in second, ...	`A[i ...; k ...; ...]`

transpose of A	`⍉A`
transpose of B, axes ordered by A	`A⍉B`
maps A: 1 for a∈ B, 0 for a∉ B	`A∈B`
grade up A	`⍋A`
grade up B with elements of A as top priority	`A⍋B`
grade down A	`⍒A`
grade down B with elements of A as low priority	`A⍒B`
transpose of A	`⍉A`
enclose A	`⊂ A`
enclose B with selected elements given the binary vector A	`A⊂ B`
disclose A	`⊃ A`
recursively pick elements of B given the indices in A	`A⊃ B`

Decode single digits of B with respect to base A	`A⊥B`
Encode B with respect to bases given by A	`A⊤B`

line label A	`A:`
branch to line A	`→A`

execute APL expression A	`⍎A`
format A as chars	`⍕A`

user input	`⍞`

system var/function	`⎕`

reduce `op` over array A	`op/A`
compress: select B using A as mask	`A/B`
A/B on last dimension	`A⌿B`
expand: insert zeros in B using A as mask	`A\B`
A\B on last dimension	`A⍀B`
inner product with functions f, g	`Af.gB`
outer product with function f	`A∘.fB`
for each b∈B, apply: Ab	`A¨B`
axis: AfC, over Bth axis	`Af[B]C`
commute `1-x` over array A	$1 - \ddot{\sim} A$

⎕CR, ⎕FIO, ⎕PLOT, ⎕SQL

When called with an empty string as right argument, these will show a table with all their possible uses.

Circle function

A	A∘B	A	A∘B
0	$\sqrt{1-B \times B}$		
−1	arcsin B	1	sin B
−2	arccos B	2	cos B
−3	arctan B	3	tan B
−4	$\sqrt{-1+B \times B}$	4	$\sqrt{1+B \times B}$
−5	arcsinh B	5	sinh B
−6	arccosh B	6	cosh B
−7	arctanh B	7	tanh B
−8	−(8∘B)	8	$\pm\sqrt{-1+B \times B}$
−9	B	9	real part of B
−10	+B	10	|B
−11	0J1×B	11	imag part of B
−12	*0J1×B (e^{iB})	12	arc B (phase of B)

For A= 8, the sign before the square root is opposite of B.

Function Definition

Example: $f(d, v) = \left(v_1^d + \cdots + v_n^d\right)^{1/d}$

Dynamic function definition (dfn):

α is the left argument, ω is the right argument.

```
f ← { ( +/ω*α ) * (÷α) }
```

Traditional function definition (tradfn):

∇: begin/end defun. "∇`R ← A f B ;U ;V`" is "`f` takes left arg `A`, right arg `B`, has local vars `U`, `V`, and returns result in `R`".

```
∇res ← d f v ;sq ;sum
  sq ← v * d
  sum ← +/sq
  res ← sum*(÷d)
∇
```