

<code>import "filename"</code>	import module
<code>import "filename" as name</code>	import filename as module name
<code>include "filename"</code>	include verbatim text from file
<code>type f(type,...);</code>	optional function declaration
<code>type name;</code>	variable declaration
<code>type f(type arg,...) {</code>	function definition
<code>statements</code>	
<code>return value;</code>	
<code>}</code>	

Data types/declarations

boolean (true or false)	<code>bool</code>
tri-state boolean (true, default, or false)	<code>bool3</code>
integer	<code>int</code>
float (double precision)	<code>real</code>
ordered pair (complex number)	<code>pair</code>
character string	<code>string</code>
fixed piecewise cubic Bezier spline	<code>path</code>
unresolved piecewise cubic Bezier spline	<code>guide</code>
color, line type/width/cap, font, fill rule	<code>pen</code>
label with position, alignment, pen attributes	<code>Label</code>
drawing canvas	<code>picture</code>
affine transform	<code>transform</code>
constant (unchanging) value	<code>const</code>
allocate in higher scope	<code>static</code>
no value	<code>void</code>
inhibit implicit argument casting	<code>explicit</code>
structure	<code>struct</code>
create name by data type	<code>typedef type name</code>

3D data types (import three;)

ordered triple	<code>triple</code>
3D path	<code>path3</code>
3D guide	<code>guide3</code>
3D affine transform	<code>transform3</code>

Constants

exponential form	<code>6.02e23</code>
T _E X string constant	<code>"abc...de"</code>
T _E X strings: special characters	<code>\\, \"</code>
C strings: constant	<code>'abc...de'</code>
C strings: special characters	<code>\\, \" \' \?</code>
C strings: newline, cr, tab, backspace	<code>\n \r \t \b</code>
C strings: octal, hexadecimal bytes	<code>\0-\377 \x0-\xFF</code>

modulus (remainder)	<code>%</code>
comparisons	<code>== != > >= < <=</code>
not	<code>!</code>
and or (conditional evaluation of RHS)	<code>&& </code>
and or xor	<code>& ^</code>
cast expression to type	<code>(type) expr</code>
increment decrement prefix operators	<code>++ --</code>
assignment operators	<code>+= -= *= /= %=</code>
conditional expression	<code>expr1 ? expr2 : expr3</code>
structure member operator	<code>name.member</code>
expression evaluation separator	<code>,</code>

Flow control

statement terminator	<code>;</code>
block delimiters	<code>{ }</code>
comment delimiters	<code>/* */</code>
comment to end of line delimiter	<code>//</code>
exit from while/do/for	<code>break;</code>
next iteration of while/do/for	<code>continue;</code>
return value from function	<code>return expr;</code>
terminate execution	<code>exit();</code>
abort execution with error message	<code>abort(string);</code>

Flow constructions (if/while/for/do)

<code>if(expr) statement</code>
<code>else if(expr) statement</code>
<code>else statement</code>
<code>while(expr)</code>
<code>statement</code>
<code>for(expr1; expr2; expr3)</code>
<code>statement</code>
<code>for(type var : array)</code>
<code>statement</code>
<code>do statement</code>
<code>while(expr);</code>

array element *i*
 array indexed by elements of int array *A*
 anonymous array
 array containing *n* deep copies of *x*
 length
 cyclic flag
 pop element *x*
 push element *x*
 append array *a*
 insert rest arguments at index *i*
 delete element at index *i*
 delete elements with indices in *[i,j]*
 delete all elements
 test whether element *n* is initialized
 array of indices of initialized elements
 complement of int array in *{0,...,n-1}*
 deep copy of array *a*
 array *{0,1,...,n-1}*
 array *{n,n+1,...,m}*
 array *{n-1,n-2,...,0}*
 array *{f(0),f(1),...,f(n-1)}*
 array obtained by applying *f* to array *a*
 uniform partition of *[a,b]* into *n* intervals
 concat specified 1D arrays
 return sorted array
 return array sorted using ordering *less*
 search sorted array *a* for key
 index of first true value of bool array *a*
 index of *n*th true value of bool array *a*

Initialization

initialize variable
 initialize array

path connectors

straight segment
 Beziér segment with implicit control points
 Beziér segment with explicit control points
 concatenate
 lift pen
 ..tension atleast 1..
 ..tension atleast infinity..

Labels

implicit cast of string *s* to Label
 Label *s* with relative position and alignment
 Label *s* with absolute position and alignment
 Label *s* with specified pen

draw commands

draw path with current pen
 draw path with pen
 draw labeled path
 draw arrow with pen
 draw path on picture

```

name[i]
name[A]
new type[dim]
array(n,x)
name.length
name.cyclic
name.pop()
name.push(x)
name.append(a)
name.insert(i,...)
name.delete(i)
name.delete(i,j)
name.delete()
name.initialized(n)
name.keys
complement(a,n)
copy(a)
sequence(n)
sequence(n,m)
reverse(n)
sequence(f,n)
map(f,a)
uniform(a,b,n)
concat(a,b,...)
sort(a)
sort(a,less)
search(a,key)
find(a)
find(a,n)

```

```

type name=value;
type[] name={...};

```

```

--
..
..controls c0 and c1.
&
^^
::
---
```

```

s
Label(s,real,pair)
Label(s,pair,pair)
Label(s,pen)

```

```

draw(path)
draw(path,pen)
draw(Label,path)
draw(path,pen,Arrow)
draw(picture,path)

```

fill path with pen
 fill path on picture

label commands

label a pair with optional alignment *z*
 label a path with optional alignment *z*
 add label to picture

clip commands

clip to path
 clip to path with fill rule
 clip picture to path

pens

Grayscale pen from value in *[0,1]*
 RGB pen from values in *[0,1]*
 CMYK pen from values in *[0,1]*
 RGB pen from hexadecimal string
 hexadecimal string from rgb pen
 hsv pen from values in *[0,1]*
 invisible pen
 default pen
 current pen
 solid pen
 dotted pen
 wide dotted current pen
 wide dotted pen
 dashed pen
 long dashed pen
 dash dotted pen
 long dash dotted pen
 PostScript butt line cap
 PostScript round line cap
 PostScript projecting square line cap
 miter join
 round join
 bevel join
 pen with miter limit
 zero-winding fill rule
 even-odd fill rule
 align to character bounding box (default)
 align to T_EX baseline
 pen with font size (pt)
 LaTeX pen from encoding,family,series,shape
 T_EX pen
 scaled T_EX pen
 PostScript font from strings
 pen with opacity in *[0,1]*
 construct pen nib from polygonal path
 pen mixing operator

fill(path,pen)
 fill(picture,path)

```

label(Label,pair,z)
label(Label,path,z)
label(picture,Label)

```

```

clip(path)
clip(path,pen)
clip(picture,path)

```

```

gray(g)
rgb(r,g,b)
cmyk(r,g,b)
rgb(string)
hex(pen)
hsv(h,s,v)
invisible
defaultpen
currentpen
solid
dotted
Dotted
Dotted(pen)
dashed
longdashed
dashdotted
longdashdotted
squarecap
roundcap
extendcap
miterjoin
roundjoin
beveljoin
miterlimit(real)
zerowinding
evenodd
nobasealign
basealign
fontsize(real)
font(strings)
font(string)
font(string,real)
Courier(series,shape)
opacity(real)
makepen(path)
+

```

number of nodes in path p	size1(p)	shift by value s	shift(real,real)
is path p cyclic?	cyclic(p)	shift by pair	shift(pair)
is segment i of path p straight?	straight(p,i)	scale by x in the <i>x</i> direction	xscale(x)
is path p straight?	pieewisestraight(p)	scale by y in the <i>y</i> direction	yscale(y)
coordinates of path p at time t	point(p,t)	scale by x in both directions	scale(x)
direction of path p at time t	dir(p,t)	scale by real values x and y	scale(x,y)
direction of path p at length(p)	dir(p)	map $(x,y) \rightarrow (x+sy,y)$	slant(s)
unit(dir(p)+dir(q))	dir(p,q)	rotate by real angle in degrees about pair z	rotate(angle,z=(0,0))
acceleration of path p at time t	accel(p,t)	reflect about line from P--Q	reflect(P,Q)
radius of curvature of path p at time t	radius(p,t)		
precontrol point of path p at time t	precontrol(p,t)	string operations	
postcontrol point of path p at time t	postcontrol(p,t)	concatenate operator	+
arclength of path p	arclength(p)	string length	length(string)
time at which arclength(p)=L	arctime(p,L)	position \geq pos of first occurence of t in s	find(s,t,pos=0)
point on path p at arclength L	arcpoint(p,L)	position \leq pos of last occurence of t in s	rfind(s,t,pos=-1)
first value t at which dir(p,t)=z	dirtime(p,z)	string with t inserted in s at pos	insert(s,pos,t)
time t at relative fraction l of arclength(p)	reltime(p,l)	string s with n characters at pos erased	erase(s,pos,n)
point at relative fraction l of arclength(p)	relpoint(p,l)	substring of string s of length n at pos	substr(s,pos,n)
point midway along arclength of p	midpoint(p)	string s reversed	reverse(s)
path running backwards along p	reverse(p)	string s with before changed to after	replace(s,before,after)
subpath of p between times a and b	subpath(p,a,b)	string s translated via $\{\{\text{before,after}\},\dots\}$	replace(s,string [] [] table)
times for one intersection of paths p and q	intersect(p,q)	format x using C-style format string s	format(s,x)
times at which p reaches minimal extents	mintimes(p)	casts hexadecimal string to an integer	hex(s)
times at which p reaches maximal extents	maxtimes(p)	casts x to string using precision digits	string(x,digits=realDigits)
intersection times of paths p and q	intersections(p,q)	current time formatted by format	time(format="%a %b %d %T %Z %Y")
intersection times of paths p and a--b	intersections(p,a,b)	time in seconds of string t using format	seconds(t,format)
intersection times of path p crossing $x=x$	times(p,x)	string corresponding to seconds using format	time(seconds,format)
intersection times of path p crossing $y=z.y$	times(p,z)	split s into strings separated by delimiter	split(s,delimiter="")
intersection point of paths p and q	intersectionpoint(p,q)		
intersection points of p and q	intersectionpoints(p,q)		
intersection of extension of P--Q and p--q	extension(P,Q,p,q)		
lower left point of bounding box of path p	min(p)		
upper right point of bounding box of path p	max(p)		
subpaths of p split by n th cut of knife	cut(p,knife,n)		
winding number of path p about pair z	windingnumber(p,z)		
pair z lies within path p ?	interior(p,z)		
pair z lies within or on path p ?	inside(p,z)		
path surrounding region bounded by paths	buildcycle(...)		
path filled by draw(g,p)	strokepath(g,p)		
unit square with lower-left vertex at origin	unitsquare		
unit circle centered at origin	unitcircle		
circle of radius r about c	circle(c,r)		
arc of radius r about c from angle a to b	arc(c,r,a,b)		
unit n -sided polygon	polygon(n)		
unit n -point cyclic cross	cross(n)		

pictures

add picture pic to currentpicture	add(pic)
add picture pic about pair z	add(pic,z)

August 2012 v1.0. Copyright © 2012 John C. Bowman

Permission is granted to make and distribute copies of this card, with or without modifications, provided the copyright notice and this permission notice are preserved on all copies.