

Asymptote Reference Card

Program structure/functions

```
import "filename"  
import "filename" as name  
include "filename"  
type f(type,...);  
type name;  
type f(type arg,...) {  
    statements  
    return value;  
}
```

Data types/declarations

boolean (true or false)
tri-state boolean (true, default, or false)
integer
float (double precision)
ordered pair (complex number)
character string
fixed piecewise cubic Bezier spline
unresolved piecewise cubic Bezier spline
color, line type/width/cap, font, fill rule
label with position, alignment, pen attributes
drawing canvas
affine transform
constant (unchanging) value
allocate in higher scope
no value
inhibit implicit argument casting
structure
create name by data type

bool
bool3
int
real
pair
string
path
guide
pen
Label
picture
transform
const
static
void
explicit
struct
typedef *type name*

3D data types (import three;)

ordered triple
3D path
3D guide
3D affine transform

triple
path3
guide3
transform3

Constants

exponential form
TeX string constant
TeX strings: special characters
C strings: constant
C strings: special characters
C strings: newline, cr, tab, backspace
C strings: octal, hexadecimal bytes

6.02e23
"abc...de"
\\", \'
'abc...de'
\\", \" \' \?
\n \r \t \b
\0-\377 \x0-\xFF

Operators

arithmetic operations
modulus (remainder)
comparisons
not
and or (conditional evaluation of RHS)
and or xor
cast expression to type
increment decrement prefix operators
assignment operators
conditional expression
structure member operator
expression evaluation separator

+ - * /
%
== != > >= < <=
!
&& ||
& | ^
(*type*) *expr*
++ --
+= -= *= /= %=
*expr*_1 ? *expr*_2 : *expr*_3
name.*member*
,

Flow control

statement terminator
block delimiters
comment delimiters
comment to end of line delimiter
exit from while/do/for

;

{ }

/* */

//

break;

continue;

return *expr*;

exit();

abort(string);

Flow constructions (if/while/for/do)

if(*expr*) *statement*
else if(*expr*) *statement*
else *statement*

while(*expr*)
 statement

for(*expr*_1; *expr*_2; *expr*_3)
 statement

for(*type var* : *array*)
 statement

do *statement*
 while(*expr*);

Arrays

array
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 if(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 nth 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
draw visible portion of line through two pairs

type[] name;
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)
drawline(pair,pair)

fill commands

fill path with current pen
fill path with pen
fill path on picture

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

label commands

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

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

clip commands

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

clip(path)
clip(path,pen)
clip(picture,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 heximdecimal string
heximdecimal 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 TeX baseline
pen with font size (pt)
LaTeX pen from encoding,family,series,shape
TeX pen
scaled TeX pen
PostScript font from strings
pen with opacity in [0,1]
construct pen nib from polygonal path
pen mixing operator

gray(g)
rgb(r,g,b)
cmyk(r,g,b)
rgb(string)
hex(hex)
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)
+

path operations

number of segments in path p
number of nodes in path p
is path p cyclic?
is segment i of path p straight?
is path p straight?
coordinates of path p at time t
direction of path p at time t
direction of path p at length(p)
unit(dir(p)+dir(q))
acceleration of path p at time t
radius of curvature of path p at time t
precontrol point of path p at time t
postcontrol point of path p at time t
arclength of path p
time at which arclength(p)=L
point on path p at arclength L
first value t at which dir(p,t)=z
time t at relative fraction l of arclength(p)
point at relative fraction l of arclength(p)
point midway along arclength of p
path running backwards along p
subpath of p between times a and b
times for one intersection of paths p and q
times at which p reaches minimal extents
times at which p reaches maximal extents
intersection times of paths p and q
intersection times of path p with '--a--b--'
intersection times of path p crossing x==x
intersection times of path p crossing y==z.y
intersection point of paths p and q
intersection points of p and q
intersection of extension of P--Q and p--q
lower left point of bounding box of path p
upper right point of bounding box of path p
subpaths of p split by nth cut of knife
winding number of path p about pair z
pair z lies within path p?
pair z lies within or on path p?
path surrounding region bounded by paths
path filled by draw(g,p)
unit square with lower-left vertex at origin
unit circle centered at origin
circle of radius r about c
arc of radius r about c from angle a to b
unit n-sided polygon
unit n-point cyclic cross

pictures

add picture pic to currentpicture
add picture pic about pair z

affine transforms

length(p)
size(p)
cyclic(p)
straight(p,i)
piecewisestraight(p)
point(p,t)
dir(p,t)
dir(p)
dir(p,q)
accel(p,t)
radius(p,t)
precontrol(p,t)
postcontrol(p,t)
arclength(p)
arctime(p,L)
arcpoint(p,L)
dirtime(p,z)
reltime(p,l)
relpoint(p,l)
midpoint(p)
reverse(p)
subpath(p,a,b)
intersect(p,q)
mintimes(p)
maxtimes(p)
intersections(p,q)
intersections(p,a,b)
times(p,x)
times(p,z)
intersectionpoint(p,q)
intersectionpoints(p,q)
extension(P,Q,p,q)
min(p)
max(p)
cut(p,knife,n)
windingnumber(p,z)
interior(p,z)
inside(p,z)
buildcycle(...)
strokepath(g,p)
unitsquare
unitcircle
circle(c,r)
arc(c,r,a,b)
polygon(n)
cross(n)

identity()
shift(real,real)
shift(pair)
xscale(x)
yscale(y)
scale(x)
scale(x,y)
slant(s)
rotate(angle,z=(0,0))
reflect(P,Q)

+
length(string)
find(s,t,pos=0)
rfind(s,t,pos=-1)
insert(s,pos,t)
erase(s,pos,n)
substr(s,pos,n)
reverse(s)
replace(s,before,after)
replace(s,string [] [] table)
format(s,x)
hex(s)
string(x,digits=realDigits)
time(format="%a %b %d %T %Z %Y")
seconds(t,format)
time(seconds,format)
split(s,delimiter="")