

Asymptote Reference Card

Program structure/functions

| | |
|----------------------------------------------------------------------------------------------------------------|---------------------------------|
| <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> <code>statements</code> <code>return value;</code> <code>}</code> | function definition |

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> |
| TeX string constant | <code>"abc...de"</code> |
| TeX 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> |

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

Flow control

statement terminator
block delimiters
comment delimiters
comment to end of line delimiter
exit from `while/do/for`
next iteration of `while/do/for`
return value from function
terminate execution
abort execution with error message

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> |

`+ - * /`
`%`
`== != > >= < <=`
`!`
`&& ||`
`& | ^`
`(type) expr`
`++ --`
`+= -= *= /= %=`
`expr1 ? expr2 : expr3`
`name.member`
`,`

`;`
`{ }`
`/* */`
`//`
`break;`
`continue;`
`return expr;`
`exit();`
`abort(string);`

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 {*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
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

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 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 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)  
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)  
+
```

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**

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)

add(pic)
add(pic,z)

affine transforms

identity transform
shift by values
shift by pair
scale by **x** in the x direction
scale by **y** in the y direction
scale by **x** in both directions
scale by real values **x** and **y**
map $(x,y) \rightarrow (x+sy,y)$
rotate by real **angle** in degrees about pair **z**
reflect about line from **P--Q**

string operations

concatenate operator
string length
position \geq **pos** of first occurrence of **t** in **s**
position \leq **pos** of last occurrence of **t** in **s**
string with **t** inserted in **s** at **pos**
string **s** with **n** characters at **pos** erased
substring of string **s** of length **n** at **pos**
string **s** reversed
string **s** with **before** changed to **after**
string **s** translated via $\{\{\text{before}, \text{after}\}, \dots\}$
format **x** using C-style format string **s**
casts hexadecimal string to an integer
casts **x** to string using precision **digits**
current time formatted by **format**
time in seconds of string **t** using **format**
string corresponding to **seconds** using **format**
split **s** into strings separated by **delimiter**

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="")

May 2014 v1.1. Copyright © 2014 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.