Asymptote-Dokumentation: http://asymptote.sourceforge.net/ (Seitenanzahlen beziehen sich auf die aktuelle PDF-Version)

1 Datentypen, Deklarationen und Initialisierungen

Typ	autom. Init.
`bool`	`false`
`int`	0
`real`	0
`pair`	(0,0)
`triple`	(0,0,0)
`string`	
`path`	`nullpath`
`guide`	`nullpath`
`pen`	default pen
`transform`	identity

Beispiele:

```
int x;
real y=3;
```

2 Asymptote als Taschenrechner I

- Einfache Rechenoperationen `+`, `-`, `*`, `/`, `%`, `^`; möglich: `3x` für `3*x`.
- erlaubt: `i +=2` etc., `++i`, `--i`; Achtung: keine Postfix-Operatoren!
- Mathematische Funktion wie z.B. `sqrt`, `exp`, `sin`, etc. `Sin` für Argument in Grad. Siehe S. 66.
- Operationen für Paare:
  - Zugriff auf Real- und Imaginärteil mit `z.x` oder `xpart`, analog mit `y`
  - `conj`, `length`, `angle`, `realmult(z1,z2)` (Hadamard-Produkt), `dot(z1,z2)`, ... (s.S. 25f)
- analoge Funktionen existieren für `triple`, insbesondere auch `cross(t1,t2)` (Kreuzprodukt) (s.S. 26f)
- Stringfunktionen zum Suchen, Ersetzen, etc. von Teilstrings, ... (s.S. 27)

3 Arrays

Siehe S. 68ff.

3.1 Deklaration und Initialisierung

- Deklarationen erfolgen mittels `type[] name`, `type[][] name` etc.
- Arrays werden mit dem empty array initialisiert
- `type[] name = null` gibt das null-array (ungleich empty array!)
- `type[] name = new type[k]` erzeugt uninitialisiertes array der Länge k
- Beispiel: `int[] v = {0,1,2}`, `int[][] A = {{0,1},{1,2}}`

3.2 Arrays vergrößern, verkleinern

- `length` gibt die Länge des arrays
- `cyclic` auf `true` setzen: zu große Indices werden modulo der Länge reduziert
- `push(int x)` und `append(int[] x)` vergrößern das array ums Argument
- `insert(int i ... T[] x)` fügt array `T` an Stelle i ein
- `delete(i,j)` löscht Elemente an Stellen i bis j.

3.3 Zugriff auf Elemente

`x[2]`, `x[2:4]`, `x[:4]`, `x[3:]`

3.4 weitere Operationen auf Arrays

`copy` (echte Kopie!), `concat`, `sort`

4 Asymptote als Taschenrechner II

- Anlegen spezieller Vektoren mittels `sequence`, `map`, `uniform`, `array(int n, T value)`
- Anlegen spezieller Matrizen mittels `identity`, `diagonal`
- `dot(real[] a, real[] b)`, `transpose(T[][])`
- Lösen von Gleichungssystemen: `tridiogonal`, `solve`
- Inverse Matrix: `inverse`
- weitere Matrixoperationen im Paket `math` enthalten

Lösen von Gleichungen zweiten bzw. dritten Grades mittels `quadraticroots(real a, real b, real c)`, `cubicroots` liefert ein array als Ergebnis

5 Zeichnen

5.1 Datenstrukturen fürs Zeichnen

`path`, `guide`, `pen`, `Label`, `transform`

5.2 Zeichnen von einfachen Linien

5.2.1 gerade Linien

`draw((0,0)--(100,100))`: gerade Linie mit den angegebenen Endpunktkoordinaten
`draw((0,0)--(100,100)--(100,50)--(50,0))`: Streckenzüge
`draw((0,0)--(100,100)--(100,0)--cycle`: schließt den Streckenzug
`draw(unitsquare)`: zeichnet Einheitsquadrat: (0,0)–(1,0)–(1,1)–(1,0)–cycle

5.2.2 gekrümmte Linien

`draw((0,0)..(50,50)..(100,0))`: zeichnet einen Halbkreis

`draw((0,0){right}..{up}(50,50))`: Viertelkreis; `right` und `up` geben die Ausgangs- und Eingangsrichtung der Linie an.

`draw(unitcircle)`, `draw(circle(pair M, real r))`, `draw(ellipse(pair M, real a, real b))`

`draw(arc(pair c, real r, real angle1, real angle 2))`

`dot(pair d)`: Punkt in d

siehe auch gebLinien.asy

5.3 Optionen zum Linienzeichnen

für Pfeile, Linenart, Linenstärke, Linienfarbe siehe Linienenden.asy, LinienFarbenArten.asy

für Definition neuer Stiftspitzen siehe makepen.asy

5.4 Beschriftungen

siehe label.asy

5.5 Flächenfüllen

Grundfunktionen `fill` und `filldraw`. Siehe flaechfuell.asy

5.6 Transformationen

Grundfunktionen `shift`, `scale`, `rotate`. Siehe transforms.asy

6 Asymptote als Taschenrechner III

Operationen mit Pfaden: s.S. 31ff

```
pair point(path p, int t)
pair point(path p, real t)
pair dir(path p, int t, int sign=0, bool normalize=true)
pair dir(path p, real t, bool normalize=true)
real arclength(path p)
pair relpoint(path p, real l)
pair midpoint(path p)
pair intersectionpoint(path p, path q, real fuzz=-1)
pair[] intersectionpoints(path p, path q, real fuzz=-1)
pair extension(pair P, pair Q, pair p, pair q)
```

siehe auch TRIII.asy

7 Bilder in LaTeX einbinden

siehe Beispieldatei asybsp.tex

8 Zusatzpakete

Einbinden mittels `import package`. Dann kann auf alle Funktionen zugegriffen werden. Alternativen:

```
access package;
package.function(..)
```

oder `from package access function`. Dann kann auf die eine function direkt zugegriffen werden (s.S. 76f).
Eine Auswahl an Zusatzpaketen:

math enthält Matrixoperationen, Schnittpunkt von Ebene und Gerade, `drawline` (zeichnet der Teil der (unendl.) Geraden durch zwei Punkte, der im aktuellen Bild sichtbar ist)

pattern für Füllmuster (s.o.)

binarytree, drawtree zum Zeichnen von (Binär)bäumen (siehe treetest.asy)

stats für statistische Funktionen

trembling siehe floatingdisk.asy

markers siehe markers1.asy, markers2.asy

flowchart für Flußdiagramme (siehe controlsystem.asy)

three Erweiterung von path etc. auf 3D

solids siehe z.B. cylinder.asy, cones.asy, hyperboloid.asy und torus.as

8.1 Zeichnen von Funktionen

mittels Paket `graph`; siehe Funktion1.asy, Funktion2.asy, Funktion3.asy
`palette` zum Färben von Höhenschichten (siehe palettenbsp.asy)
`contour`: Zum Erzeugen von Höhenlinien `graph3`: Erweiterung von graph auf 3D zum Zeichnen von Kurven und Flächen

8.2 geometry

Dokumentation und Beispiele finden sich unter http://asymptote.sourceforge.net/links.html

8.2.1 Datentypen

`coordsys` für Koordinatensystem

`point` und `vector`: Im normalen Koordinatensystem (nur solche werden wir betrachten) mit `pair` gleichzusetzen

`line` und `segment` für Geraden, Halbgeraden, Strecken

`conic` für Kegelschnitte; wir beschränken uns auf die abgeleiteten Klassen `circle`, `ellipse`, `parabola`, `hyperbola`

`arc` für Ellipsenbögen

`triangle` für Dreiecke

8.2.2 Geraden und Strecken

erzeugen mit `line(point A, bool extendA=true, point B, bool extendB=true)` oder `line(real a, point A)`, `Ox` erzeugt die x-Achse

`draw(line l)` zeichnet die Linie im derzeitigen Fenster, `show(line l)` zeichnet die definierenden Punkte der Linie, sowie Richtung und Normalvektor.

`intersectionpoint(line l1, line l2)`, `intersectionpoints(line l, path g)`: zur Berechnung von Schnittpunkten

Funktionen zu parallelen und orthogonalen Geraden sind z.B. `parallel(point M, line l)`, `bool parallel(line l1, line l2)`, `perpendicular(point M, line l)`

`sector(int n=2, int p=1, line l1, line l2` teilt den Winkel zwsichen l1 und l2 in n gleiche Teile und gibt die p-te Teilungslinie

z.B. `angle(line l1, line l2)`, `distance(point M, line l)`

siehe GeomLine.asy

Operatoren siehe GeomLineTrafo.asy

8.2.3 Transformationen

`scale(real k, point M)`, `projection(point A,point B)`, `projection(A,B,C,D)`

`reflect(line l)`, `reflect(line l1, line l2)`, `projection(line l)`

8.2.4 Kreise

Erzeugen mittels

```
circle(point C, real r)
circle(point A, point B)
circle(point A, point B, point C)
incircle(A, B, C)
```

vergrößern mit `x*circle` (Mittelpunkt fest, Radius*x)

`point@circle`: Abfrage, ob Punkt auf Kreis

`tangents(circle c, point M)` berechnet Tangenten

`angpoint(circle c, real x)` gibt den Punkt auf c im Winkel x

`intersectionspoints` liefert Schnittpunkte

siehe GeomKreis.asy, GeomKreis.asy

8.2.5 Ellipsen, Parabeln, Hyperbeln

analog zu `circle`, siehe GeomEPH.asy

8.2.6 Ellipsenbögen

erzeugen mit `arc(ellipse el, real angle1, real angle2, polarconicroutine)`, mit polarconicroutine= fromCenter oder fromFocus

8.2.7 triangle

Datenstrukur für Dreiecke; viele Funktionen; siehe GeomTriangle.asy

9 Weiterführendes Programmieren

9.1 Schleifen

```
if(x == 1.0) {
   write("x equals 1.0");
} else {
   write{"x is not equal to 1.0");
}

for(int i=0; i < 10; ++i) {
   write(i);
}

int[] a={1,1,2,3,5};
for(int k : a ) {
   write(k);
}
```

logische Operatoren: `==`, `!=`, `<`, `<=`, `>=`, `>`, `&`, |, `!`

9.2 Definition von Funktionen

Beispiel:

```
real Kreisumfang(real radius) {
   return 2*pi*r;
}
```

Defaultwerte von Variablen werden wie folgt gesetzt:
`real Kreisumfang(real radius=1) {`

9.3 Definition von neuen Strukturen

Das Beispiel legt eine neue Struktur Person mit den Variablen `firstname` und `lastname` an. Danach folgt ein Konstruktor für die neue Struktur.

```
struct Person {
   string firstname;
   string lastname;

   void operator init(string firstname, string lastname) {
      this.firstname=firstname;
      this.lastname=lastname;
   }
}
```

Eine neue Person kann jetzt wie folgt angelegt werden:
`Person joe=Person("Joe", "Jones");`

9.4 Casts

automatisch: `int` to `real`, `int` to `pair`, `real` to `pair`, `pair` to `path`, `pair` to `guide`, `path` to `guide` und vice versa.
Explizites casting z.B. `int i = (int) 2.5;`