Simple programming in cT

Syntax

```
command  tag
```

All commands are lower case only.

A command must be separated from its tag by a TAB.

Commands inside a -loop- / -endloop- or an -if- / -endif- must be indented.

Values in the tag may be numbers, variables, or arithmetic expressions combining variables and numbers.

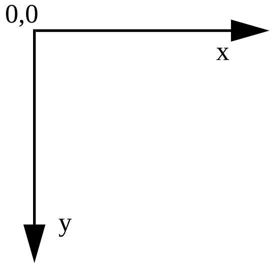
Screen coordinates

0,0 is in upper left hand corner. *x* increases to the right; *y* increases down.

x and *y* are measured in pixels.

Coordinate pairs are separated by semicolons in the tag.

In the following table, command names and key words are bold.



COMMAND	**TAG**	**ACTION**
unit	unitname	creates program segment
define	**float:** var1, var2...	creates floating point variables
at	x,y	move pen to location x,y
write	Some text <|**s,** variable|>	writes text on screen at pen location display value of a variable
circle	radius	draws circle at pen location
disk	radius	draws filled circle at pen location
calc	variable := value	assigns value to variable
mode	**write** **erase**	fills in pixels on screen erases pixels on screen
draw **fill**	x1,y1; x2,y2; x3,y3... x1,y1; x2,y2; x3,y3...	draws line connecting specified points draws filled polygon with given vertices

if
. . . .
endif
	logical expression	executes indented code if expression is true
		example of expressions:
		(a=1) & (b=2) $$ both true
		(c<0) \| (0 < x < 100) $$ inclusive or

loop . . . . **endloop**		executes indented code repeatedly
outloop		exit from loop

COMMAND	TAG	ACTION
color	**zred** **zgreen** **zblue** **zmagenta** **zyellow** **zcyan** **zblack** **zwhite**	sets pen color

Sample program

```
$syntaxlevel 2
unit     Display
         float: x, v, dt
* initialize variables
calc     v := 2
         dt := 0.05
* draw vertical wall
color    zblue
fill     300,65;320,225
color    zred

loop
         mode     erase
         at       x,100
         disk     4
         mode     write
         calc     x := x + v*dt
         at       x,100
         disk     4
         if       x > (300-4)      $$ reached wall
                  outloop          $$ get out of loop
         endif
endloop
at       10,10
write    Done!
```

More commands

COMMAND	TAG	ACTION
do	unitname	execute named unit as a subroutine
vector	x1,y1; x2,y2; headtype	draws arrow if headtype is positive or omitted, filled head if headtype is negative, open head if headtype is integer, size is in pixels if headtype is fraction, size is fraction of vector length
dot	x,y	lights one pixel at specified location
pause	number	pause for specified number of seconds (may be a fraction)
erase		erases whole screen

Graphing commands

COMMAND	TAG	ACTION
gorigin	x,y	sets origin for graph
bounds	xlength,ylength	sets length of +x and +y axes in pixels
axes		draws axes
scalex	xmax	sets scale of x axis
scaley	ymax	sets scale of y axis
labelx	interval, interval	specifies label interval, minor tick marks
labely	interval, interval	specifies label interval, minor tick marks
gat	x,y	locates pen in graphing coordinates
gdraw	x1,y1; x2,y2...	draws line in graphing coordinates
gfill	x1,y1; x2,y2; x3,y3...	draws filled polygon in graphing coordinates
gcircle	radius	draws circle with radius specified in graphing coordinates
gdisk	radius	draws filled circle with radius specified in graphing coordinates
gdot	x,y	lights one pixel at location specified in graphing coordinates
gvector	x1,y1; x2,y2	sane as -vector- but in graphing coordinates see above for headtype documentation

Working with very small numbers

cT uses a “fuzzy” test to decide if a number is equal to zero—this keeps tiny differences due to the accumulation of roundoff error from interfering with ordinary operations. However, if you are working with very small numbers (e.g. 1e-9), you will need to use the following command to turn off this feature, which can interfere with comparisons:

COMMAND	**TAG**	**ACTION**
inhibit	fuzzyeq	treat numbers < 1e-9 as different from zero (put this command in any unit that manipulates small numbers)

Saving your program

Initially your program will be named “Untitled.” To give it a different name (usually your first initial, last name, and problem number), choose Save As from the File menu in cT.

While you are working you should save your program periodically on the hard disk.

At the end of a working session, you need to save your program on a floppy disk, so you can move it to another computer. To do this:

> Choose “Save” from the “File” menu and save your program.
>
> Quit cT Create.
>
> Find the icon for your program on the hard disk, and drag it onto the icon for the floppy disk.

Since floppy disks occasionally go bad, it is a good idea to keep backup copies of all your programs on another floppy disk, or if you own a computer, on the hard disk of your computer.

Online Help

This handout covers only a small subset of cT commands. You can learn more from the online reference manual, which can be accessed by choosing Help from the Window menu. The help contains documentation and small code samples which you can copy and paste into your program to try them.

For additional information about cT, see http://cil.andrew.cmu.edu/ct.html.