A Brief cT Tutorial

Start the cT programming environment by double-clicking on the “cT Create” icon. You should see two windows side-by-side on your monitor. For now, leave them where they are. (Later on you may find it convenient to resize or reposition them.)

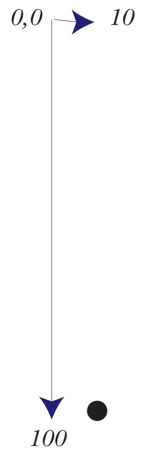
The window entitled “cT” is the “execution” window. The display your program generates will appear here. The window entitled “Untitled” is the “edit” window. You will write and edit your cT code here. There is already one line in this window:

```
$syntaxlevel 2
```

Leave this line alone (it tells the cT compiler what version of cT you are using). Click below this line and type the following lines. Be sure to put a TAB between each command and its tag:

```
$syntaxlevel 2

unit        Display
at          10,100
disk        4
```



A “unit” is the basic building block of a cT program. The -at- command positions the virtual pen on a Cartesian coordinate system in which 0,0 is in the upper left-hand corner of the screen and the +y axis goes down. The units of distance are pixels. The -disk- command plots a filled circle whose radius is 4 pixels. To run this program, choose “Run from beginning” from the “Option” menu. You should see a black disk appear in the execution window.

To animate the disk, we will plot it and erase it repeatedly at different locations. Amend your current program to include the following code, remembering to type a TAB after each command:

```
loop
            calc        x := x + 1
            at          x,100
            disk        4
endloop
```

Note that the code inside the loop must be indented by one extra TAB.

Now run the program by choosing “Run from beginning” from the “Option” menu. You will see a new window, the “message” window, in which you see the message:

```
“Unrecognized variable name: x”
```

To define a floating-point variable *x*, insert a line directly after the “unit” command, like this:

```
unit        Display
            float: x
```

Now run the program from the beginning. You should see a horizontal string of disks. You have plotted the disk repeatedly, but you have not erased it from its old position. Let’s do that, by changing the code to read:

```
loop
            mode        erase
            at          x,100
```

```
                disk        4
                mode        write
                calc        x := x + 1
                at          x,100
                disk        4
        endloop
```

On a fast computer, this may be too fast to see. Let's slow it down, by adding a smaller increment to *x* each time through the loop. Change the calculation to read:

```
                calc        x := x + 0.1
```

Even after the disk disappears off the edge of the screen the program is still running, because we did not tell it to quit. To stop the program, choose "Quit running" from the "Option" menu.

Let's make this a little more real by defining a variable *v* to represent speed, and a time step *dt* to represent the time interval between our plotting of the dots. We'll set the initial value of *v* to 2, and the initial value of *dt* to 0.05 seconds. (Note that one "second" in our virtual time will not take one second in real time.)

```
        unit        Display
                    float: x, v, dt
        calc        v := 2
                    dt := 0.05
```

We'll make the position of the disk change according to the speed:

```
        calc        x := x + v*dt
```

For variety, let's make the disk a different color, by adding:

```
        color       zred
```

Now we have:

```
        unit        Display
                    float: x, v, dt
        calc        v  := 2
                    dt := 0.05
        color       zred
        loop
                    mode        erase
                    at          x,100
                    disk        4
                    mode        write
                    calc        x := x + v*dt
                    at          x,100
                    disk        4
        endloop
```

Run this program and see what happens.

Usually we won't want our program to continue running forever. Let's make a wall for the "ball" to run into, by adding the following statement before the "color zred":

```
        color       zblue
        fill        340,65;370,225
```

Now if we run the program, the ball goes straight through the wall! Not quite what we intended.

Let’s monitor the position of the ball, and get out of the loop if it hits the wall. We’ll put the following code inside the loop, just before the endloop:

```
if          x > (300-4)
            outloop
endif
```

Because the ball has a radius of 4 pixels, we need to stop it before its center reaches the edge of the wall, so we check for $x < (300-4)$.

Now our program looks like this:

```
unit        Display
            float: x, v, dt
calc        v := 2
            dt := 0.05
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)
                        outloop
            endif
endloop
```

You’ve now encountered basic commands that are sufficient for getting started with cT. For details on additional commands and cT options, see the extensive on-line help that is accessible from the “Window” menu.

Debugging

Frequently a program does not work correctly on the first try. For example, type in the following program and run it:

```
unit        First
            f: x, y, r, value

calc        value := 20
            x := sqrt[exp(value)]
            y := 134*sin(value/2)

at          x,y
disk        r
```

Nothing appears on the screen. Why not?

A simple but extremely useful debugging technique is to examine the values of the relevant variables, by printing them in an unused region of the display. Add this code before the -at- statement:

```
at          10,10
write       x  is <|s,x|>
            y  is <|s,y|>
            r  is <|s,r|>
pause
```

The -pause- command waits for a keypress before continuing execution. The symbols <| and |> denote a -show- command that has been embedded in an alphanumeric output statement. Note that quotes are not required for the -write- command.

If we get this output on the screen:

```
x = 5746
y = -324
r = 3E-27
```

we can see that we are trying to plot an infinitesimal disk in a location far off the screen.

Comments

There are two ways to include comments in your program. First, any line beginning with an asterisk is a comment, which is ignored by the compiler:

```
* This line is a comment
```

Second, a comment may be included at the end of a line of code if it is preceded by “$$”

```
calc        x := x+ v*dt                 $$ use velocity to update position
```

It is often useful to comment out a whole set of lines while debugging your program. Use the mouse to select the lines, then choose “Comment Lines” from the Edit menu. An asterisk will appear at the beginning of each of the selected lines. You can use “Un-comment lines” to remove asterisks from a selected set of lines.