# Chapter 7

## Character Graphics

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>7.1</td>
<td>Character Locations in the Execution Window</td>
</tr>
<tr>
<td>7.2</td>
<td>Creating a Graphical Pattern with Characters</td>
</tr>
<tr>
<td>7.3</td>
<td>Drawing in Color</td>
</tr>
<tr>
<td>7.4</td>
<td>Background Color</td>
</tr>
<tr>
<td>7.5</td>
<td>Hiding the Cursor</td>
</tr>
<tr>
<td>7.6</td>
<td>Animation with Graphics</td>
</tr>
<tr>
<td>7.7</td>
<td>Controlling the Speed of Animation</td>
</tr>
<tr>
<td>7.8</td>
<td>Pausing for User Input</td>
</tr>
<tr>
<td>7.9</td>
<td>Exercises</td>
</tr>
<tr>
<td>7.10</td>
<td>Technical Terms</td>
</tr>
</tbody>
</table>
7.1 Character Locations in the Execution Window

The output of Turing programs so far has been limited to a display of lines of characters in the Execution window. These characters have been values of expressions which are of real, integer, or string type. The output on the lines followed in sequence, one line after the next. To produce successful graphics we must be able to output characters anywhere in the window, in any order that we choose. We will be outputting one character at a time at a location in the window which is selected as the location of the cursor before the output instruction is executed.

The Execution window can have characters in any one of 25 lines (or rows) and on each line in any one of 80 columns. To output a character in a particular location in the Execution window we first place the cursor at that location by a locate predefined procedure in the form

\[ \text{locate} \left( \text{row}, \text{column} \right) \]

The row can be any integer between 1 and 25 inclusive, and the column any integer between 1 and 80 inclusive. To place the letter T approximately at the center of the Execution window we would give the instructions

\[ \text{locate} \left( 13, 40 \right) \]
\[ \% 13\text{th row down, 40th column across} \]
\[ \text{put} \left( \text{T} \right) . . \]

Notice that there are two dots in the put instruction. Without the two dots the put instruction assumes that the line is complete and would make the rest of the output line blank. To prevent this happening we use the two dots which indicate that the cursor should be left where it was. If we give the instructions:

\[ \text{locate} \left( 13, 40 \right) \]
the character U would appear after the T on the line in the location (13, 41).

The following program writes Fred in the four corners and in the middle of the Execution window.

```
locate (1, 1)  % Not necessary.
put "Fred" ..
locate (1, 77)
put "Fred" ..
locate (13, 38)
put "Fred" ..
locate (25, 1)
put "Fred" ..
locate (25, 77)
put "Fred" ..
```

At the beginning of the program the cursor is always located at (1, 1) so it is not really necessary to include the first locate. However, if the program segment is used in another program, the cursor might not be located at (1, 1) so it is helpful to include the locate.

The .. after each put statement serves two purposes. First, it stops the rest of the line from being erased and second, it stops the window from scrolling when there is output on the last line. If you removed the .. and ran this program again, the window would scroll, causing the Freds at the top of the window to disappear and the two Freds at the bottom to appear on different lines.

### 7.2 Creating a Graphical Pattern with Characters

Now that you know how to output a character in the Execution window in any position you can use this facility to draw simple pictures. To begin a new graphic we clear the window so that we
are not drawing on top of some other output. To do this we use the procedure cls. The instruction

```
cls
```

will clear the window and then place the cursor at the location (1, 1), which is the top left-hand corner. Here is a program which draws a line of asterisks down the window from top to bottom in the column you specify.

```
% The "DrawVerticalLine" program
% Draws a vertical line of asterisks
var column : int
put "Choose a column for line ",
   "between 1 and 80 inclusive"
get column
cls
for row : 1 .. 25
   locate (row, column)
   put "*" ..
end for
```

Notice that after the prompt appears and the column value has been entered the window is cleared, using cls, so that the prompt disappears and does not interfere with the appearance of the asterisks.

### 7.2.1 Interactive Graphics

Suppose you wanted to have a graphics program with which you were going to interact. For example, you might want to draw a picture a little at a time. You would need to split the Execution window into two parts: part for the graphic and part for the prompting message and the input. We will call each part a **subwindow**. The graphics will be in the top part of the window from row 1 to row 20 and the prompting message and your input will be at the bottom in lines 23 and 24. The two subwindows will be separated by a line of minus signs in row 21.
Here is the program.

```plaintext
% The "DrawVerticalLines" program
% Draw a series of vertical lines, one at a time
cls
var column : int
% Draw line of minus signs between the two subwindows
locate (21, 1)
for count : 1 .. 80
  put "-" ..
end for
% Could instead use put repeat ("-", 80)
loop
  % Move cursor into prompt subwindow
  locate (23, 1)
  put " " % Clear the line
  locate (23, 1)
  put "Choose a column for the line between 1 and 80, end with -1: "..
  get column
  exit when column = -1
for row : 1 .. 20
  locate (row, column)
  put "*" ..
end for
end loop
```

### 7.2.2 Diagonal Lines and Patterns

Here is a program to draw a diagonal line in the Execution window starting at the point (1, 1).

```plaintext
% The "DrawDiagonalLine" program
% Draw a diagonal line in window
% starting in row 1, column 1
var row, column := 1
cls
loop
  % Stop when diagonal touches bottom of window
  exit when row > 25
```
locate (row, column)
put ".* ..
% Move down diagonal
row := row + 1
column := column + 1
end loop

A more interesting program that is interactive can be obtained by letting the user choose the starting point for the diagonal. If we did this we would have to worry about the diagonal hitting the side of the window as well as the bottom and the exit condition would have to be

exit when row > 25 or column > 80

A still more interesting program is obtained if we let the diagonal "bounce" off the edge of the window and be reflected just as a hockey puck bounces off the boards. If it bounces off the bottom we must start decreasing the row number but keep the column number increasing. Here is the complete program for a bouncing (or reflected) motion. The reflection is not exactly accurate because the column change of one character width and row change of one line spacing are not exactly equal for the window.

% The "Bounce" program
% Simulates the action of a bouncing puck
var row, column : int
put "Enter starting row from 2-25 " ..
get row
put "Enter starting column from 2-80 " ..
get column
var rowChange, columnChange := 1
cls
loop
% The next six lines are a Selection construct
% See the Selection chapter if you do not
% understand them
if row = 25 or row = 1 then
    rowChange := – rowChange
end if
  if column = 80 or column = 1 then
    columnChange := – columnChange
  end if
locate (row, column)
put **.
  row := row + rowChange
  column := column + columnChange
end loop

If the drawing is formed too rapidly for you to enjoy the
bouncing you can slow it down by including a time-wasting for
loop right after the put statement that draws the asterisk.

Here is such a time wasting for loop.

  var garbage : int
  for i : 1 .. 10
    % Perform a time wasting calculation
    garbage := 237 * 26
  end for

You can also use the built in procedure delay in order to slow
execution down. It has the form

delay (duration)

where the duration is in milliseconds (thousandths of a second).

7.3 Drawing in Color

Graphics can be made more interesting by displaying the
results in color. To choose a color for a character to be displayed
use the color predefined procedure in the form

  color (chosenColor)

The chosenColor can be one of Turing's predefined colors or a
number. These colors are white, blue, green, cyan, red, magenta,
brown, black, gray, brightblue, brightgreen, brightcyan, brightred,
brightmagenta, yellow, and darkgray. If using a number, the number
can range from 0 to the maximum color number available in Turing, usually 255.

Note: in Turing you may spell **color** as **colour** and **gray** as **grey** if you prefer.

Here is a program that displays a box outlined by randomly colored asterisks that blink.

```turing
% The "Marquee" program.
% Draws a box outlined by randomly colored asterisks.
var depth, width : int
put "Enter the width of the box (less than 60)"
get width
put "Enter the depth of the box (less than 20)"
get depth
cls

% Arrange to center box in window.
class := (25 – depth) div 2
class := (80 – width) div 2
var colorNo : int

% Draw top of box
locate (topRow, leftColumn)
for count : 1 .. width
    % Choose a random color
    randint (colorNo, 0, 15)
    % Set the color.
color (colorNo)
    put "*" ..
end for

% Draw bottom of box
locate (topRow + depth – 1, leftColumn)
for count : 1 .. width
    randint (colorNo, 0, 15)
color (colorNo)
    put "*" ..
end for
```
% Draw left side of box
const sideTop := topRow + 1
const sideBottom := topRow + depth - 2
for row : sideTop .. sideBottom
  randint (colorNo, 0, 15)
  color (colorNo)
  locate (row, leftColumn)
  put "*" ..
end for

% Draw right side of box
const rightColumn := leftColumn + width - 1
for row : sideTop .. sideBottom
  randint (colorNo, 0, 15)
  color (colorNo)
  locate (row, rightColumn)
  put "*" ..
end for

7.4 Background Color

As well as controlling the color of the displayed characters, the background for each character can be set to a variety of colors by the colorback predefined procedure. It has the form

    colorback (colorNumber)

If a blank is output, the background color is all that you see.

Here is a program to color the entire window blue.

% The "DrawSky" program
% Color the whole window blue
colorback (1)
for row : 1 .. 25
  for column : 1 .. 80
    locate (row, column)
    % Output a blank
    put " " ..
Notice that the window is colored a character at a time.

Here is a program that colors the window green in a random fashion.

```
% The "LeafFall" program
% Color parts of the window randomly
cls
% Set to color background green
colorback (green)
var row, column : int
loop
    randint (row, 1, 25)
    randint (column, 1, 80)
    locate (row, column)
    put " ".. 
end loop
```

Run the program to see how long it takes for the leaves to completely cover the window.

### 7.5 Hiding the Cursor

Sometimes the appearance of a graphical pattern can be confused by the display of the cursor which always appears following the last output or echoed input. This is particularly true in animated graphics.

To hide the cursor we would use the predefined procedure `setscreen` in the form `setscreen ("nocursor")`. To have the cursor show again use `setscreen ("cursor")`. 
7.6 Animation with Graphics

The illusion of continuous motion can be created by having a series of still pictures in which an object is shown in slightly different positions from one to the next. We will color the window green then move a magenta asterisk around at random starting it at the center. When it reaches the edge of the window we will start it back at the center again.

```plaintext
% The "BrownianMotion" program
% Moves an asterisk around on
% the window like a smoke particle
% This is known as Brownian motion
% Color window green
colorback (green)
for row : 1 .. 25
  put repeat (" ", 80) ..
end for

% Set character color to magenta
color (magenta)
setscreen ("nocursor") % Hide cursor
loop
  % Start at center
  var row := 13
  var column := 40
  loop
    % Start over when asterisk goes off the window
    exit when column < 1 or column > 80
    or row < 1 or row > 25
    locate (row, column)
    put "*
    delay (100)
    % Erase asterisk in old location by
    % outputting a blank there
    locate (row, column)
    put " " ..
    % Compute new value for row
    randint (row, row - 1, row + 1)
```

% Compute new value for column
randint (column, column – 1, column + 1)
end loop
play (">C<")  % Sound a note
% This is explained in the Music chapter
end loop

7.7 Controlling the Speed of Animation

In the BrownianMotion program the particle moves about very rapidly. It is possible that you might want to introduce a delay between the put instruction which outputs the asterisk in a given place and the put instruction which erases it. You can do this as you have seen by inserting between the two put instructions a call to the delay predefined procedure. The object of the delay procedure is to just waste time. Nothing is actually being accomplished other than that. You can control the length of the delay by setting the duration to a variable like duration where duration is an integer value which is a measure of how much time you want to waste. Here is the time wasting call to delay.

delay (duration)

The statements for controlling the speed can be inserted into the program right after the first 5 lines of comments. These are

\begin{verbatim}
var duration: int
put "Choose delay time for animation: " ..
get duration
\end{verbatim}

The choice of a good value for duration must be found by experimenting with various values on your computer to get the speed of animation you want.
7.8 Pausing for User Input

You can also instruct a program to pause until the user presses a key. This is done using the statements

```turing
var reply : string (1)
getch (reply)
```

The `getch` procedure causes Turing to wait until a key is pressed and assigns the key to `reply`. The `reply` is a one-character string and it is used in the `getch` (get a character) command. The `reply` variable can only be declared once. Here is a program segment that prompts the user to hit a key to resume the execution.

```turing
put "Press any key to continue the program execution. " ..
getch (reply)
```

Once the user presses a key the execution will resume. The `getch` `reply` instruction is useful because the user indicates when they are ready to continue, whereas a delay waits for a period of time set by the programmer.

To stop the character the user presses from being output to the window, use

```turing
setscreen ("noecho")
```

To have the user input displayed on the window, use

```turing
setscreen ("echo")
```

7.9 Exercises

1. Write a program to plot a horizontal line of minus signs in any row of the window that you specify.
2. Make the program of question 1 interactive so that you can plot as many horizontal lines as you want, one after the other.
3. Write a program to fill the Execution window with diagonal lines of asterisks separated by diagonal blank lines. Note that this can be accomplished easily by outputting lines of alternating asterisks and blanks using, in turn, either

    put repeat (" **", 40)

or

    put repeat ("* ", 40)

(Note the space before and after the asterisk.) Try doing this in color with light green asterisks on a background of blue. Next, make the asterisks blink.

4. Write a program to draw a funny face in the window with its center at any point you specify. Make the face of a size 9 characters by 9 characters. Arrange so that the color of the face and the background can be input at the same time as you specify its location in the Execution window. Refuse to plot the face if it goes outside the window. You will need a selection construct for this. See the next chapter for how to draw a face in pixel graphics.

5. Write a program to plot a red box on a blue background. To do this wash the whole window with blue then change the background color to red and plot a box full of blanks.

6. Write a program to output a box which is tilted in the window so that its top and bottom follow diagonal lines like those in question 3. How would you fill the box with a color different from the background.

7. By modifying the BrownianMotion program, arrange that a picture of a funny face is moved randomly around the window.

8. Modify the BrownianMotion program so that you do not erase the image of the particle each time but leave it as a trail when it moves. Change the color of the trail randomly each time the particle starts at the center. Use a black background.

9. Write a program that outputs a cat’s face at the center of the window, erases it when you type the letter e, then plots it again when you type the letter p.
10. Modify the *Bounce* program so that the bouncing asterisk changes color randomly on each bounce.

11. There are a number of characters available in the ASCII code of the computer besides the ones given in the appendix. You can explore these with this program

```
% The "CharValues" program
% See what characters are available
var value: int
loop
    put "Enter a character value, 1 to 255 inclusive " ..
    get value
    put "Character whose value is ", value, " is ", chr (value)
end loop
```

Try to see what different characters you get. One of the characters will produce a beep rather than appearing in the window. What is its value? What happens when you try the value 0?

12. Display the letters of the alphabet in different colors on one line. The letters should appear one at a time and each letter should appear slowly and then disappear when the next letter appears (like the floors lighting up on an elevator).

13. Ask the user for the upper left row and column and the bottom right row and column to represent the top left and bottom right corners of a rectangle. Ask the user which character they would like the rectangle to be created with. Create the outline of the rectangle from the top-left corner to the bottom-right corner given by the user in the character they indicated.

14. Ask the user for the upper left row and column and the bottom right row and column to represent the top left and bottom right corners of a rectangle. Ask the user for the number of the color they wish to use. Ask the user for the character they wish to use. Create a filled in rectangle in the color and character indicated by the user going from the top-left to the bottom-right co-ordinates.
7.10 Technical Terms

- location of character in the Execution window
- locate
- graphics
- cls
- interactive graphics
- window
- color
- background color
- colorback
- animation
- delay