C Final Test
Jurassic Park dragon curve

Tuesday 7th June 2016, 14:00

THREE HOURS

(including 15 minutes planning time)

- Please make your swipe card visible on your desk.
- After the planning time, log in using your username as **both** your username and password.

The maximum mark is 40.

Credit will be awarded throughout for clarity, conciseness and *useful* commenting.

Important note: THREE MARKS will be deducted from solutions that do not compile. Comment out any code that does not compile before you leave. Commented-out code will not be marked.

1 Jurassic Park dragon curve

In this test you will draw a fractal curve (the Jurassic Park dragon) using L-systems. The Jurassic Park[1] curve is also known as the Heighway dragon curve or the Harter-Heighway dragon curve and is a member of a family of self-similar fractal curves, which can be drawn using recursive methods such L-systems. As you learnt from L-systems in Haskell, the *Lindenmayer Systems* and their variants are used extensively in computer graphics to build structures dynamically, on request.

L-systems consist of a collection of *rules* that provide a way to expand a starting string (axiom) into a sequence of turtle commands by successive expansion of the string using defined rules. In this test, we are going to focus on a deterministic two-dimensional structure called Jurassic Park curve. Figure 1 illustrates the curve.

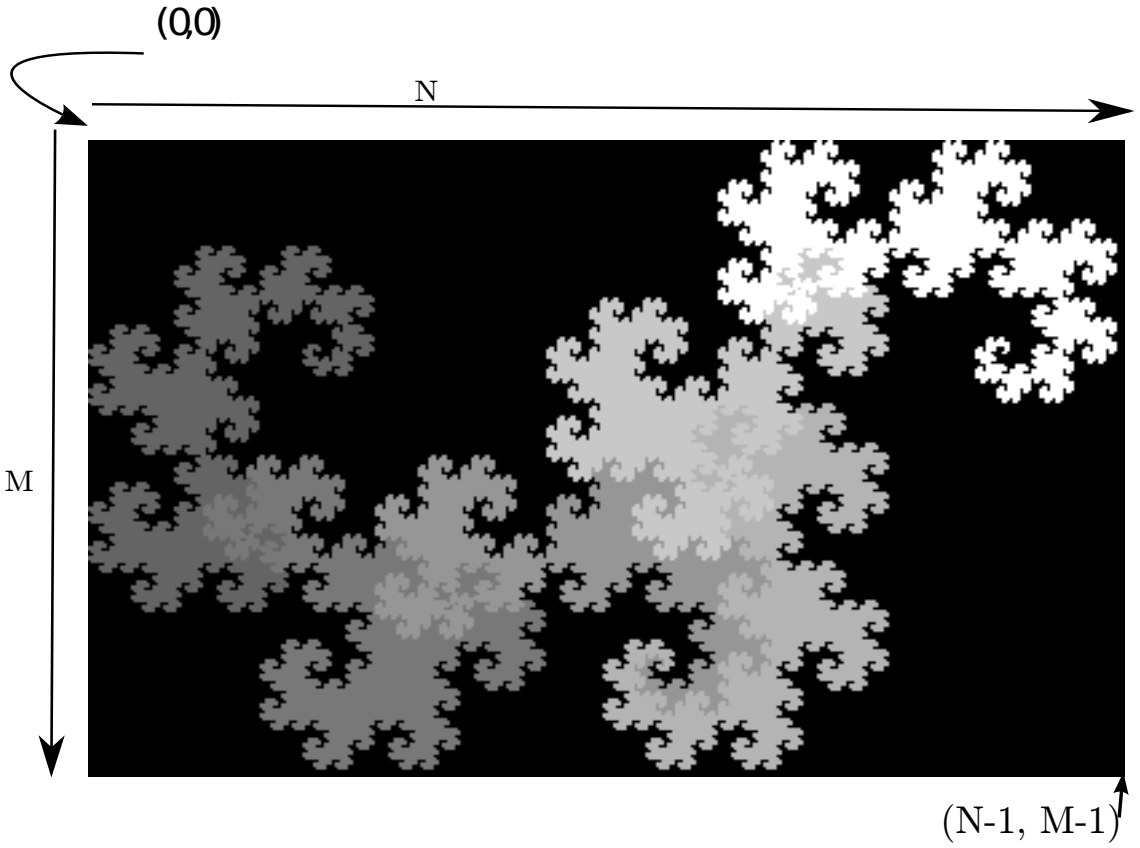

Figure 1: The Jurassic Park or Heighway dragon curve rendered in greyscale.

[1]It appeared on the section title pages of the Michael Crichton novel *Jurassic Park*

2 Building the Jurassic Park dragon curves

The L-system for drawing the Jurassic Park dragon curve is shown below:

Jurassic Park dragon

- variables: X Y
- constants: F (draw forward) - (turn 90 degrees clockwise) + (turn 90 degrees anticlockwise)
- axiom: FX
- rules: (X $\longmapsto$ X+YF+) and (Y $\longmapsto$ -FX-Y)
- angle: 90 degrees

By changing the axiom of the L-system described above, we can construct two dragon curves back-to-back called the *Twin Dragon*:

Twin dragon

- variables: X Y
- constants: F (draw forward) - (turn 90 degrees clockwise) + (turn 90 degrees anticlockwise)
- axiom: FX+FX+
- rules: (X $\longmapsto$ X+YF) and (Y $\longmapsto$ FX-Y)
- angle: 90 degrees

The dragon curves are drawn using turtle graphics. The turtle commands are the constants ('F', '-' ,'+'), where (F) means draw forward, (-) rotate clockwise and (+) rotate anti-clockwise. The rotation angle is 90 degrees.

For example with the Jurassic Park dragon, the sequence of commands is built from the starting axiom string (e.g. "FX") and expanding the component characters according to the given rules. In the Haskell L-Systems exercise you may recall that you built an intermediate string representing the entire list of commands and then processed the individual commands separately. For example, with four levels of expansion, the command string would evolve as follows:

```
"FX"
-> "FX+YF+"
-> "FX+YF++-FX-YF+"
-> "FX+YF++-FX-YF++-FX+YF+--FX-YF+"
```

and so on. We will refer to each expansion step as an *iteration*. In this exercise you're going to avoid building the string explicitly by using recursion to expand the variables (e.g. 'X' and 'Y') whenever you encounter them. Similarly, the basic turtle commands (i.e. 'F', '+' and '-') will be executed as soon as you encounter them. This means that you only need to traverse the strings representing the individual right-hand sides of the expansion rules, not the entire command string. For example, starting with the axiom "FX", the Jurassic Park dragon would be drawn by:

1. Processing the 'F' and moving forwards in the current direction
2. Processing the 'X' by recursively traversing the string "X+YF+"

3. As there are no further characters in the (axiom) string the process terminates

Similarly, to process the string "X+YF+" we first process the 'X' by recursively traversing the string "X+YF+". We then process the command '+' and then the 'Y', by recursively traversing the string "-FX-Y', followed by the 'F' and '+' commands. We are then done. How does the evaluation terminate? You will carry round an *iteration count* in an iteration processing function which you will use to stop the expansion process after the required number of iterations.

Figure 2 illustrates the implementation of the Jurassic Park dragon curve after a varying number of iterations.

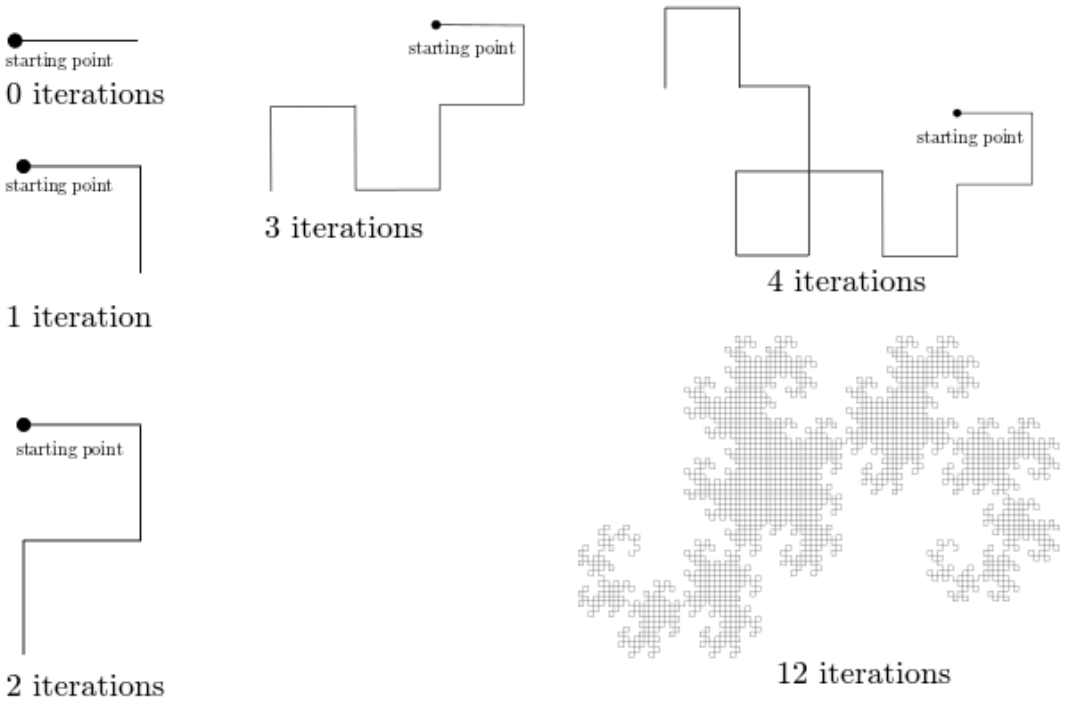

Figure 2: Different iterations of Heighway dragon or Jurassic Park dragon curve.

When drawing a dragon curve it is important to initialise the turtle so that it is pointing in the right direction, otherwise part of the curve may end up outside of the boundaries of the image, i.e. it may be clipped so that it cannot be seen in full. The initialisation rules will be explained later on.

3 What to do

The test consists of two parts and you are provided with the files for each part. In part I (see the `PartI` directory), you **only** need to use the following files provided: a header file `dragon.h` and a program file `dragon.c` . Therefore, the `dragon.c` and `dragon.h` files is where you will write your code for Part I. Part II (see the `PartII` directory) comprises a completely separate exercise which requires you to analyse a buggy program (`program.c`) and implement a solution that fixes the bug.

In the first part, you will draw the Jurassic Park and Twin dragon curves using L-systems on a greyscale image like Figure 1. You may use any of the utility functions provided in `image.c` and `image.h` (specifically `set_ pixel`, `init_image`, and `image_write`) for Part I, but these files should not be modified.

At present the functions in `dragon.c` are nothing more than stubs; you should complete each as you write and test your functions. The file `dragon.h` contains prototypes of all the functions you are expected to write. Do *not* modify the definitions in `dragon.h` as this will impede our ability to evaluate and test your code.

You may also define and use appropriate auxiliary functions in `dragon.c` and `dragon.h`. These should be appropriately scoped and not be visible outside the module in which they are defined. You may compile your code using the provided `Makefile`. By default your code will be compiled with debugging symbols (via `gcc`'s `-g` option) so you will be able to debug your code with a capable debugger: `gdb`, `cgdb`, `ddd`, or `eclipse` for example.

The `Makefile` instructs `make` to build an executable program called `dragon` which provides an image of the Jurassic Park dragon curve or a Twin dragon curve image. An example of a Jurassic Park dragon curve and a Twin dragon curve in PGM (Portable Graymap Format) image files are provided in the `images` directory for reference. These can be loaded by double-clicking them in a file manager (`nautilus`) window. You might prefer to use `KolourPaint` because it shows the location of the pixel under the mouse.

Part I – Draw Jurassic Dragon curve using L-systems

The image in Figure 1 is represented as a two-dimensional array of $M \times N$ pixels where $M, N > 0$ and each element is an integer representing the pixel's shade of grey from 0 (black) to 255 (white). In this part, you will create and save a PGM greyscale image like Figure 1. You will save the image as `jurassicdragon.pgm` in the `output` directory. After that, you will create and save another PGM greyscale image to draw a Twin Dragon. You will save the image called `twindragon.pgm` in the `output` directory. Both curves should be drawn using the L-system rules provided in the previous section. In this part, you need to take into account that the total number of iterations affects both the size and orientation of the final image. To avoid clipping when rendering, both the dimensions of the buffer and the turtle's direction must be correctly initialised. The initial direction depends on the required number of iterations as explained below.

Five functions should be implemented in `dragon.c`:

1. Write a function:

   ```
   vector_t starting_direction(int total_iterations);
   ```

   which provides the orientation of the initial segment given the total number of iterations being executed. As explained before, the initial direction will define the dragon curve orientation. The turtle's initial direction is computed by rotating `total_iterations` times the

direction (`direction.dx` and `direction.dy`) 45 degrees anti-clockwise as shown in Figure 3. `direction.dx` and `direction.dy` are provided as static global variables. The orientation of the initial segment (before `starting_direction` is called) is represented as (`direction.dx` = 1 and `direction.dy` = 0). HINT: there are 8 different initial directions of movements as illustrated in Figure 3.

[5 marks]

2. Write a function:

```
void draw_greyscale(image_t *dst, long x, long y);
```

which draws a grey value in the pixel `(x,y)` of the `dst` image. At this point, you should inspect `image.c` and `image.h` to understand the `image_t` type. Assume that there are only 6 levels of grey scale as illustrated in Figure 1. Each level is computed as the number of pixels drawn (`drawn_pixels`) multiplied by the number of greyscale levels `LEVEL` (in this case 6) and divided by the square of `dst` image height:

$$level = LEVEL * drawn\_pixels/(height * height)$$

`drawn_pixels` is also defined as a global static variable in `dragon.c` and it stores the number of pixels drawn.

Assign the following grey values to each level: for level 0 use the grey value of 100, for level 1 use the grey value of 120, for level 2 use the grey value of 150, for level 3 use the grey value of 180, for level 4 use the grey value of 200. The rest, will be consireded white pixels and they will have the value of 255.

[4 marks]

3. Write a function:

```
void string_iteration(image_t *dst, const char *str, int iterations);
```

which recursively traverses the string `str` representing the right-hand side of one of the expansion rules, executing turtle commands as it goes. Notice that, when '-' is encountered, it means that the direction of movement (`direction.dx` and `direction.dy`) should be 90 degrees clockwise. The '+' symbol means that the direction of movement should be 90 degrees anticlockwise. When 'F' is encountered, you should increment the size of the path (i.e. `drawn_pixels`), draw the current pixel (using `draw_greyscale`) and then move forward (i.e. add to the current pixel `x,y` the direction of movement (`direction.dx` and `direction.dy`). Divide always the position of the current pixel by the `scale` value[2] before drawing to differentiate between dragon curve (`scale` value 1) and twindragon curve (`scale` value 2). HINT: notice that the `iterations` will change each time that `string_iteration` is called.

[10 marks]

[2] `scale` value is used to avoid clipping when rendering both curves

4. Write a function:

```
void dragon(long size, int total_iterations);
```

which creates an image. The image `height` is equal to `size` and the image `width` is equal to one and a half of the `size`. The image will have one channel as it is a grey image and the depth value should be set to 255. After creating the image, it sets the initial pixel coordinates to (`x = size/3; y = size/3;` and `scale = 1`) and calls `starting_direction` to get the initial direction of movement to start, giving the number of iterations. After that, it calls the `string_iteration` to draw the dragon curve on the image. Once it finishes drawing the curve on the image, it saves the image called `jurassicdragon.pgm` into the `output` directory. HINT: to create and save an image, you should use the code provided in `image.c` and `image.h`.

[5 marks]

5. Write a function:

```
int main(int argc, char **argv);
```

which takes only the number of `iterations` as a command-line argument. If no argument is given, this function should set the number of `iterations` to a default value of 9. The `size` parameter required by the `dragon` function determines the height of the image to be created and is defined to be 2^n where n is the required number of `iterations`. The `main` function should call the `dragon` function passing as arguments the computed `size` and the total number of iterations. To keep the correct resolution of the curves at redering time the total number of iterations, `total_iterations`, should be twice the number of iterations. HINT: you may wish to use the `atoi` function

[3 marks]

6. Change the axiom and the rules to draw a Twindragon image. To draw the Twindragon use the following initial coordinates `x = size;` and `y = size;` and the `scale` should have 2 as a value. Save the image as `twindragon.pgm` in the `output` directory.
[5 marks]

7. Comment on the design of the provided `dragon.c` skeleton file and explain how it could be improved. Write your explanation in the file `PartI.txt` in the `output` directory
[3 marks]

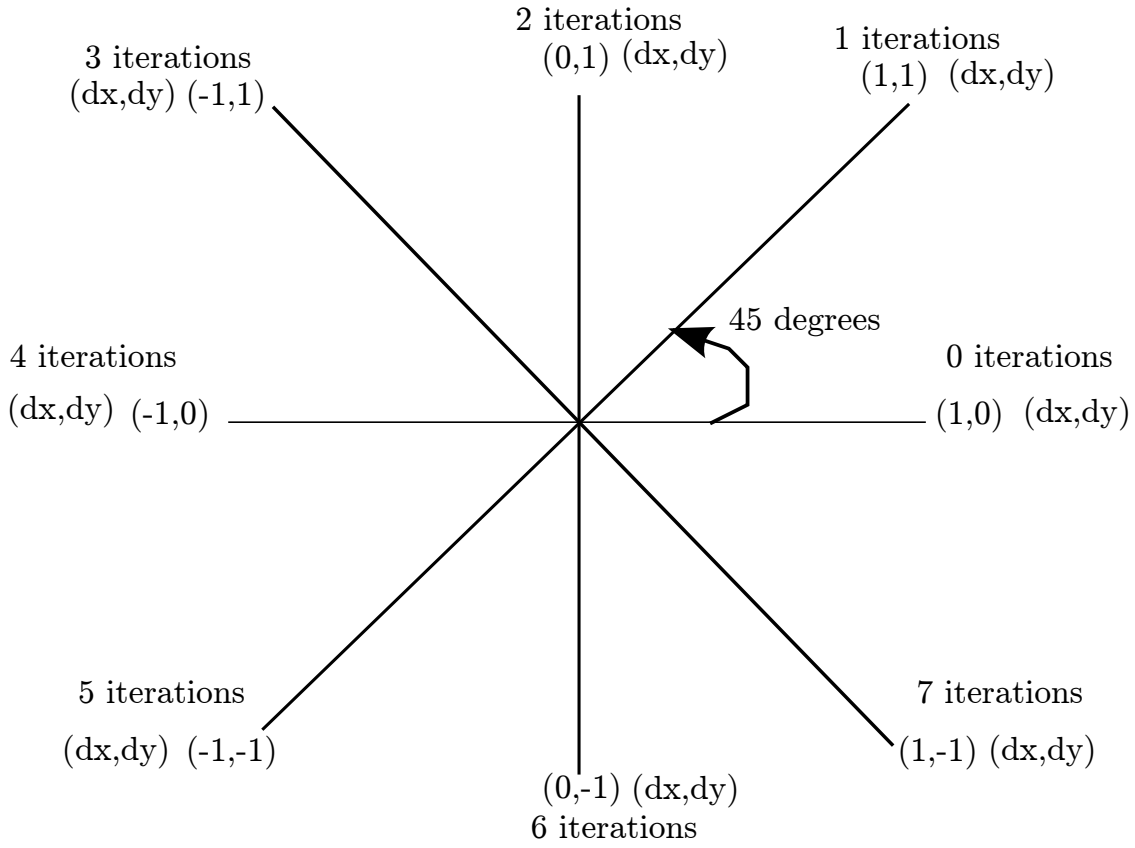

Figure 3: The orientation of the initial segment defines the orientation of the dragon curve.

Part II – Inspecting and debugging a program

1. Determine why `program.c` in the `PartII` directory has a segmentation fault. Indicate where the segmentation fault is and provide a solution. Write your explanation in the file `PartII.txt` in the directory `output` and modify `program.c` to provide a solution. HINT: use Valgrind.

   [5 marks]

You may use this page for planning