SE 2S03 — Assignment 4

Ned Nedialkov

15 November 2019

Due date: 27 November

- **Avenue will be open until December 2nd, 12:20pm.**
  No penalty if submitted between 27th and 2nd 12:20pm.
- **No hardcopy will be accepted after December 2nd.**
  NO EXCEPTIONS
  The above captures the 5-day extension for MSAFs, so no need to submit such.

Problem 1 (20 points) A Fibonacci word is defined through a concatenation of strings:

$$
\begin{aligned}
f_1 &= 1 \\
f_2 &= 0 \\
f_3 &= f_2 f_1 = 01 \\
f_4 &= f_3 f_2 = 010 \\
f_5 &= f_4 f_3 = 01001 \\
&\vdots \\
f_n &= f_{n-1} f_{n-2}
\end{aligned}
$$

(see also `https://en.wikipedia.org/wiki/Fibonacci_word_fractal`).

A Fibonacci fractal is defined on the plane by the following algorithm ($|f_n|$ is the number of digits in f_n)

Input: n
Compute:
    nth Fibonacci word
    for each digit at position $k = 1, 2, \ldots |f_n|$
        draw a segment forward
        if digit is 0
            if k is even then turn left by 90^o
            else turn right by 90^o

You need to implement this algorithm in the following function, which must produce an image corresponding to the nth Fibonacci word.

```
int fib(int n, int x, int y, int step, RGB bc, RGB fc, int w, int h,
    RGB* image)
/*Input
   n>=3
   x,y  starting position
   step>=2 how many pixels to move when drawing a segment forward
   bc background color
   fc foreground, drawing color
   w width of image in pixels
   h height of image in pixels
   The input x,y must be in [0,w-1] and [0,h-1], respectively.
  Output
   image is a pointer to a w*h array of RGB values
  Returns
    0 if this function fails, e.g. if a step results in coordinates
       outside the image window or if unable to allocate memory.
   >0 the number of steps the algorithm takes, which is also the
      length of the nth Fibonacci word.
   */
```

From the initial position, draw the first segment vertically up.

The `RGB` data type is defined in `bmp.h`. Do not change this file.

```
#ifndef BMP_H_
#define BMP_H_
typedef struct {
    /* red, green, and blue values */
    unsigned char r, g, b;
} RGB;
void saveBMP(char* file, int width, int height, RGB* image);
#endif
```

The function `saveBPM` stores an image in a bmp file (see e.g. `https://en.wikipedia.org/wiki/BMP_file_format`). This function is provided as file `bmp.c`.

You can use the provided main program, `main_fib.c`.

Store your implementation in a file with name `fib.c`. Create a makefile similar to this one

```
CFLAGS=-Wall -O2 -ansi
fib: main_fib.o fib.o bmp.o timing.o
        $(CXX) -o fib $?
runall:
        ./fib 7 10 10 10 100 100 fib7.bmp
        #./fib 9  ADD YOUR PARAMETERS HERE fib9.bmp
        #./fib 25 ADD YOUR PARAMETERS HERE fib25.bmp
        #./fib 26 ADD YOUR PARAMETERS HERE fib26.bmp
clean:
        @rm -rf fib *.o *bmp
```

so when `make fib; make runall` is typed, the four bmp files will be created. You may want to convert them to jpg format. On mills, you can do this by

```
convert fib7.bmp fig7.jpg
```

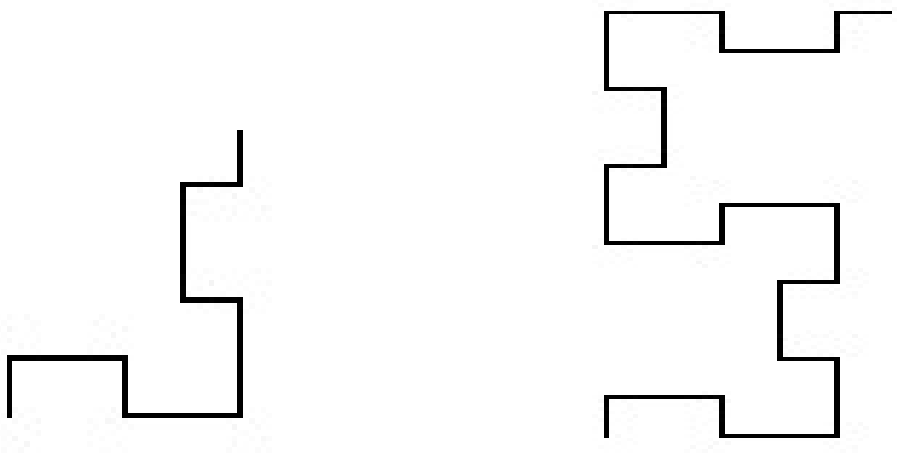
(`convert` is from ImageMagic, `https://www.imagemagick.org`).

Submit

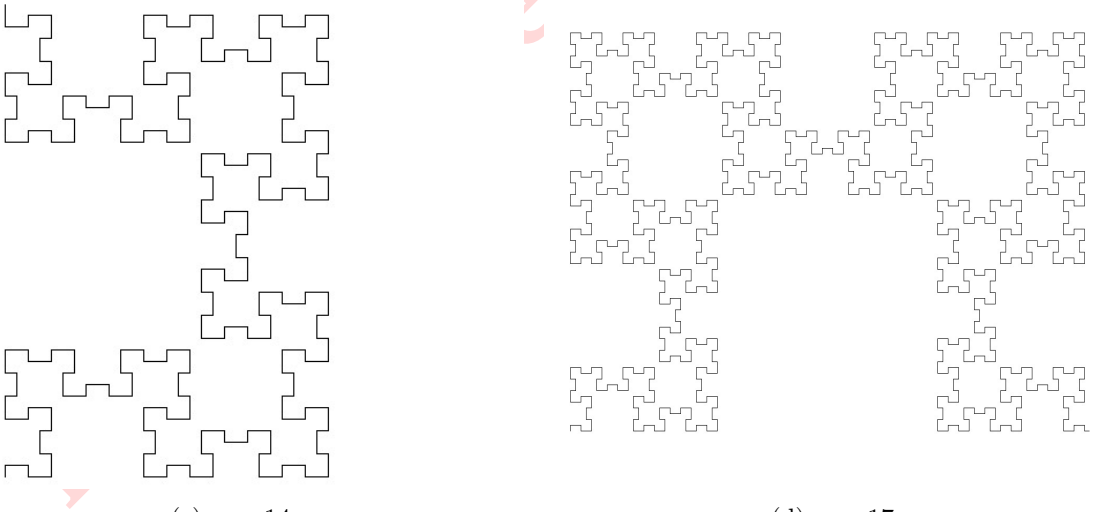
- Avenue: `fib.c` and your makefile. Do not submit image files.
- Hardcopy: `fib.c`, makefile, and your images printed

Correct images receive:

n	points
7	3
9	4
25	6
26	7



(a) $n = 7$ (b) $n = 9$



(c) $n = 14$ (d) $n = 17$

Figure 1: Fibonacci fractals for $n = 7, 9, 14, 17$ with a step of 10 pixels

If we execute `make fib` and then `make runall` and nothing is produced, this problem receives 0 marks.

In Figure 1 you can see some of my output. Note your drawing may not start exactly as mine, but you should get the overall plots.

Problem 2 (5 points) Profile the execution of

```
./fib 30 10 10 5 29000 12000 fib30.bmp
```

If you cannot reach $n = 30$, profile with the largest n for which your program works.

Submit a hard copy of your profiling data. Which line(s) of your `fib` function take most of the time? Explain why.

We will discuss in class and tutorials how to obtain profiling data. You can also read e.g. `https://www.thegeekstuff.com/2012/08/gprof-tutorial/` and the more detailed `https://ftp.gnu.org/old-gnu/Manuals/gprof-2.9.1/html_chapter/gprof_2.html`.

Bonus (3 points) If you can produce an image with $n = 34$ and step $= 2$. Submit a hardcopy of it and also the values of `x, y, step, w, h` you have used.

For fun Optimize your implementation for speed as much as you can.

Compile on mills with an optimization option `-O2` and then execute on mills

```
./fib 30 10 10 5 29000 12000 fib30.bmp
```

Record the steps per second output of the main program at
`https://docs.google.com/spreadsheets/d/1GOr-NMUkISSUZOYLBuqOxAhub9P7JgOLWwhJ19MDHJI/edit#gid=0`.

I have entered the number from my unoptimized version.

Problem 3 (10 points) In image filtering, an image is convoluted with an $n \times n$ kernel. You can read about convolution of images at `https://en.wikipedia.org/wiki/Kernel_(image_processing)#Origin`. Also, image convolutions are a basic building block in convolutional neural networks.

The Portable Pixmap Format (PPM) uses ASCII encoding of pixels for image files; for details, see `https://en.wikipedia.org/wiki/Netpbm_format#PPM_example`.

You can generate a PPM file from a JPG file using

```
convert  -compress none file.jpg  file.ppm
```

On Linux, you can use `gimp` to view a PPM file. On Mac OS X, simply use `open`.

Implement a program, name it `filter`, such that when executed as

```
./filter input.ppm kernel output.ppm
```

it reads a PPM image stored in file `input.ppm`, reads a kernel stored in file `kernel`, applies the kernel on the image, and writes the resulting image into file `output.ppm`.

The format of `kernel` is

```
n
scale
n rows of n numbers each
```

For example the 3×3 mean filter

$$\frac{1}{9}\begin{bmatrix}1 & 1 & 1\\ 1 & 1 & 1\\ 1 & 1 & 1\end{bmatrix}$$

is specified in a text file as

```
3
9
1 1 1
1 1 1
1 1 1
```

You can use zero padding. That is, when part of the filter is outside the image, the values that are outside are convoluted with zeros.

For reading and writing PPM files, you can find source code on the Internet.
Submit

- Avenue: your source code and a makefile. When make is typed, the executable `filter` must be created.
- Hardcopy: your source code.

Problem 4 (10 points) You are given the CSV file `cities.csv`. Write a C program that reads this file and outputs a CSV file with name `sorted.csv` with columns

city population country

where the rows are sorted in decreasing order by population. For example, the first three rows are

```
Tokyo,22006299,Japan
Mumbai,15834918,India
Mexico City,14919501,Mexico
```

For this problem, you must use the `qsort` function from the standard C library to sort. See `man qsort`.

Store your implementation in a file with name `cities.c`. Submit it to avenue and also a hardcopy of it.