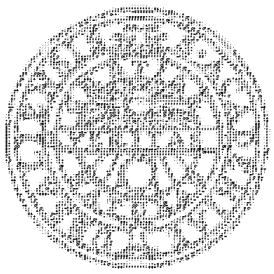


UPPSALA
UNIVERSITET

Program Design and Data Structures 20c

Project ASCIIlles

Authors:
Elli Virtanen
David Pikas
Wietze Schelhaas

Supervisors:
Carlos Penichet
Dave Clarke
Tjark Weber

Bachelor Programme in Computer Science
Uppsala Universitet
March 3, 2016

Contents

1 Project introduction **1**
1.1 Program description 1

2 User manual **1**
2.1 Files included in the program 1
2.2 How to run 1

3 Program documentation **2**

4 Algoritm description **2**
4.1 ImageTo2dList 2
4.2 Asciilate 2

5 The solutions limitations **3**

6 Problems and reflections **3**

7 Test cases **3**

8 Examples **4**

1 Project introduction

Project ASCIIlles is a short project in the course Program Design and Data Structures teached at Uppsala University during the fall semester 2015 and the spring semester 2016. The subject of this project is to create a program called ASCIIfy which converts an image to ASCII-art. The side effects are that the team gains experience in working with projects and using modules made by others.

1.1 Program description

ACSIIfy scans an image and splits it into smaller segments. The average greyscale value of every segment is calculated and represented as an ASCII-character in the exported into the terminal or as a HTML-file. This program relies on the use of JuicyPixels.

2 User manual

The main.hs file is, as the name suggests the file that will actually run the program. The program takes one to three command-line arguments. The first is the file path to the image that is to be turned into ascii. This argument is mandatory. The other two are the output file and the scale. If no output file is provided, the program will print the result in the terminal. If the output file has the .html extension, some html and css will be added to make it look nicer (namely, line breaks, smaller font-size, monospaced font and better horizontal letter spacing). Scale should be given as a natural number, the bigger the number the smaller the returned ASCII will be. If no scale is given a standard scale will be used.

To run this program you need the Haskell module JuicyPixels, which can be installed with cabal or downloaded from `http://hackage.haskell.org/package/JuicyPixels`.

2.1 Files included in the program

main.hs — Contains the main function and the functions responsible for loading and writing files.

imageTo2dlist.hs — Contains the functions to convert an image into a list that represents the pixels of the image.

asciilate.hs — Contains the functions to convert a list of pixels into ascii-art.

Utils.hs — — Contains tests and

2.2 How to run

To run, run the following line in the terminal

```
runhaskell main.hs filepath [destination] [scale]
```

Example: To run the file dave.jpeg and export it as a HTML-file with the greatest possible resolution one should run the following in the terminal:

```
runhaskell main.hs dave.jpeg dave.html 1
```

3 Program documentation

When the program starts, the firsts thing it does is to read the image or gif from the specified filepath. It does this by calling the functions readImage or readGifImages, which are funtions provided by the Haskell library JuicyPixels.

Then the program converts colored image/images to grayscale. Giving every pixel has a value between 0 and 255, where 0 is black and 255 is white and the numbers between are different shades of grey.

The program determines what symbol is to replace the pixels with by calculating the average greyscale value of a chunk of pixels.The chunk size is determined by the scale argument, for instance if you specified a scale of 2 the program would take a chunk containing 2x2 pixels. Chunks with lower values are darker so it should be replaced with a symbol containing alot of black.

This function is recursively applied to rows of pixels from the image and returns a string for evey row. These rows are combined together and written into the specified file, if the user hasen't specified a location to write the file then the program will print the ASCII-art straight into the terminal.

4 Algoritm description

The two most important algorithms that ASCIIfy uses are ImageTo2dList and Asciilate. These algorithms will be described below.

4.1 ImageTo2dList

Takes an image as an argument and first rewrites the image to greyscale with 8-bit encoding. Thereafter it maps over all pixels in the image and inserts the greyscale values into a list of lists, where each list in the list represents a row of pixels in the image.

4.2 Asciilate

Asciilate takes the list of lists as an argument and determines which ASCII-characters should replace each chunk of the image. The size of each chunk is determined by Asciilates second argument, scale, which determines the length of a side of the squareshaped chunk.

First Asciilate splits the list of lists into groups of scale amount of lists in each groups. There after it turns each of these groups into a string of ASCII characters This is done taking scale amount of elements from each list in the group and calculating the mean greyscale value among them, the greyscale value then decides which character should substitute the chunk of pixels. These strings are combined into a list which can then be printed in the terminal, saved in a file and so forth.

5 The solutions limitations

ASCIIfy is designed to support all types of images that the Haskell library JuicyPixels (used for loading and storing images) supports, which are PNG-files, JPEG-files and GIF-files. The program is not equipped to handle files in the CMYK colorspace.

The support of GIF-files is somewhat limited, the framerate for animated GIFs is constant och can thus appear awkward.

ASCIIfy is also a bit slow, so high resolution images are slow to run.

6 Problems and reflections

The greatest challenge durning this project was not the programming itself, but to understand and to use JuicyPixels.

7 Test cases

Tests:

ImageTo2DList:

- fillTransparency y a 255 should result in a value between 0 and 255 if y and a are also between 0 and 255
- Given a 2DList, calling list2DtoImage and then ImageTo2DList should result in the original list (if the original list isn't empty)
- fillTransparency 0 127 255 should equal 128
- fillTransparency 0 0 255 should equal 255

Asciilate:

- Turning a 2DList of symbols into a 2DList of grey values and then calling asciilate on that should result in the original 2DList
- The scale paramater should result in the result decreasing in size relative to its input in the way described in its specification

8 Examples

Here is the Haskell logotype, which we are going to use to demonstrate our different commands. It was created by Darrin Thompson och Jeff Wheeler, so let's gice them some credit for that.

To print out the ASCII-logo in the terminal we write this command:

```
runhaskell main.hs Haskell-Logo.png
```

```
oWWWWWWWW‘  o$$$$$$$i
.@WWWWWWW%  ‘$$$$$$$:
 !WWWWWWWW;  :$$$$$$$‘
  oWWWWWWWW‘  o$$$$$$$i
  .@WWWWWWW%  ‘$$$$$$$:
   !WWWWWWWW;  :$$$$$$$‘
    oWWWWWWWW‘  o$$$$$$$i
    .@WWWWWWW%  ‘$$$$$$$:
     !WWWWWWWW;  :$$$$$$$‘
      oWWWWWWWW‘  o$$$$$$$i
      .@WWWWWWW%  ‘$$$$$$$:  ’iiiiiiiiiiiiiiiii
       !WWWWWWWW;  :$$$$$$$‘  $WWWWWWWWWWWWWWWWW
        oWWWWWWWW‘  o$$$$$$$i  -WWWWWWWWWWWWWWWWW
        .@WWWWWWW%  ‘$$$$$$$:  :WWWWWWWWWWWWWWWWW
         !WWWWWWWW;  :$$$$$$$‘  $WWWWWWWWWWWWWWWWW
          oWWWWWWWW‘  o$$$$$$$i  -WWWWWWWWWWWWWWWWW
          .@WWWWWWW%  ‘$$$$$$$:  -‘‘‘‘‘‘‘‘‘‘‘‘‘‘‘‘
           +WWWWWWWW+  ;$$$$$$$‘
           $WWWWWWW@. .i$$$$$$$$i
          :WWWWWWWW+  *$$$$$$$$$:  !OOOOOOOOOOOO
         -WWWWWWWWi  ^$$$$$$$$$$‘  $WWWWWWWWWWW
         $WWWWWWW@. .i$$$$$$$$$$i  -WWWWWWWWWWW
        :WWWWWWWW+  *$$$$$$$$$$:  :WWWWWWWWWWW
       -WWWWWWWWi  ^$$$$$$$$$$‘  $WWWWWWWWWW
       $WWWWWWW@. .i$$$$$$$co$$$$$$i  -WWWWWWWWW
      :WWWWWWWW+  *$$$$$$$-‘$$$$$$$:
     -WWWWWWWWi  ^$$$$$$$+  :$$$$$$$‘
     $WWWWWWW@. .i$$$$$$c    o$$$$$$i
    :WWWWWWWW+  *$$$$$$$-    ‘$$$$$$$:
   -WWWWWWWWi  ^$$$$$$$+      :$$$$$$$‘
   $WWWWWWW@. .i$$$$$$c        o$$$$$$i
  :WWWWWWWW+  *$$$$$$$-        ‘$$$$$$$:
 -WWWWWWWWi  ^$$$$$$$+          :$$$$$$$‘
 $WWWWWWW@. .i$$$$$$c            o$$$$$$i
:WWWWWWWW+  *$$$$$$$-            ‘$$$$$$$:
```

If the users wants to specify the resolution one can add an integer to the command.

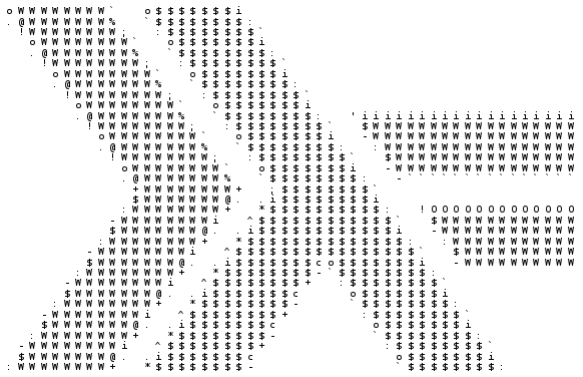
```
runhaskell main.hs Haskell-Logo.png 5
```

```
oWWWWW' c$$$$$.
.@WWWW% -$$$$$*
 !WWWWW* +$$$$$^
  oWWWWW' c$$$$$.
  .@WWWW% -$$$$$*
   !WWWWW* +$$$$$^
    oWWWWW' c$$$$i. .------------
    .@WWWW% -$$$$$* ^WWWWWWWWWWWW
     !WWWWW* +$$$$$^ cWWWWWWWWWWW
      oWWWWW' c$$$$i..OWWWWWWWWWW
      .@WWWW% -$$$$$* ^$$$$$$$$$$
       !WWWWW; +$$$$$^
       cWWWWW' c$$$$$i. `,,,,,,,,,
      ^WWWWW* !$$$$$$$* ^WWWWWWWW
     .OWWWWO -$$$$$$$$$^ cWWWWWWW
     cWWWWW' c$$$$$$$$$i..OWWWWWW
    ^WWWWW* !$$$$$:$$$$$* ^oooooo
   .OWWWWO -$$$$$c +$$$$$^
   cWWWWW' c$$$$$-  c$$$$i.
  ^WWWWW* !$$$$$!   -$$$$$*
 .OWWWWO -$$$$$c     +$$$$$^
 cWWWWW' c$$$$$-      c$$$$i.
^WWWWW* !$$$$$!       -$$$$$*
```

If the user wants to save the ASCII as a text file they should also give a textfile name as an argument for ASCIIfy like this.

```
runhaskell main.hs Haskell-Logo.png Haskell-ASCII.txt 5
```

For those users who are annoyed with the obscured proportions of this art there is a solution; by entering .html as the file extension for the destination file ASCIIfy creates a HTML-file which can be opened with your browser for a glorious monospaced view as in the picture below.



References

[1] JuicyPixels is a Haskell library which can be downloaded at `http://hackage.haskell.org/package/JuicyPixels`