Charles Catt a1726075 Wednesday, 24 October 2018

APP assignment 3 - Command Line Graphing App

Overview

- takes user input and prints a representation of a graph to the terminal
    - first asks for a function in the form of a lambda expression, then asks for x and y lower and upper bounds
    - then prompts for further action from the user (e.g. print graph, add function, remove function, etc)
- uses #lang racket
- input can be directed in to the program using bash redirection
    - i.e. $ racket grapher.scm < input.txt
- multiple functions can be graphed at once, with different symbols.

Functions

- functions can be added in the form of a lambda expression or procedure
- examples:
    - + ; y=x line
    - (lambda (x) (* x x)) ; simple parabola
    - (lambda (x) (if (> x 0) (sqrt (- 16 (* x x))) 1e13)) ; top right quarter of a circle
- there are no protections against malicious lambda functions, that would be overly complicated (see if you can break something fun)
- functions can be deleted by choosing a function to delete based on what symbol it uses

Symbols:

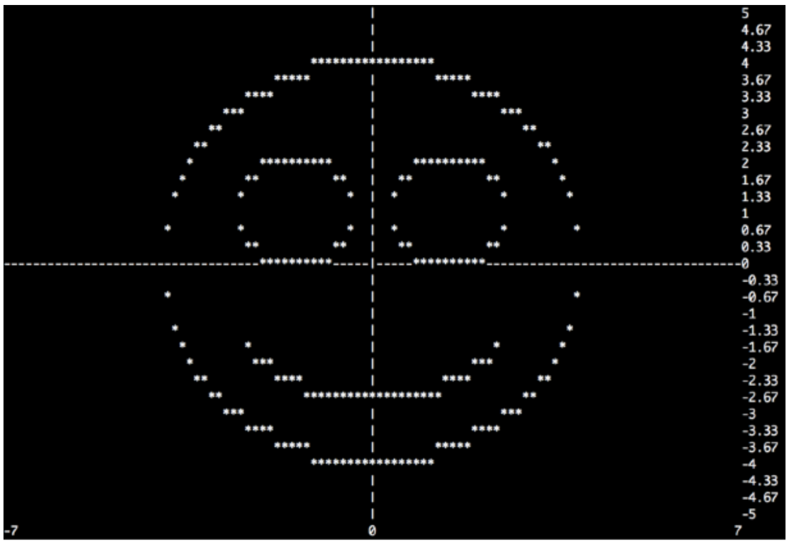
- single character
- a-z, A-Z, =,+,-,$ and other characters may be used. period . and backslash \ may not be used. there is no built-in protection against this.

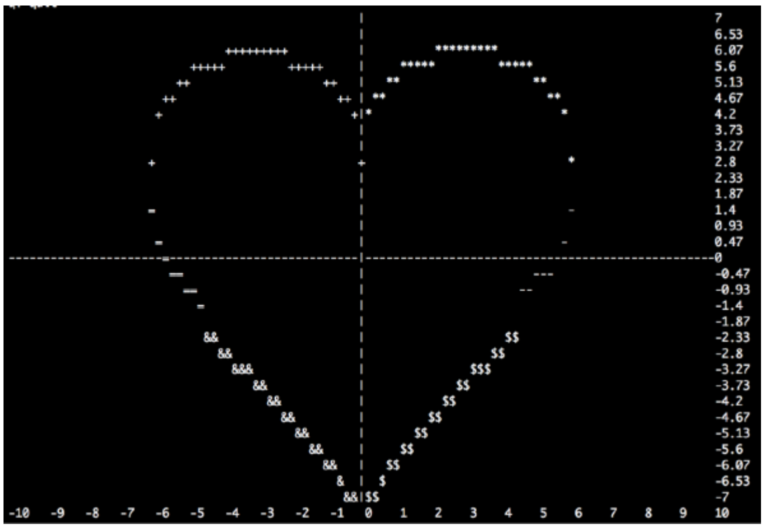
Functional Programming

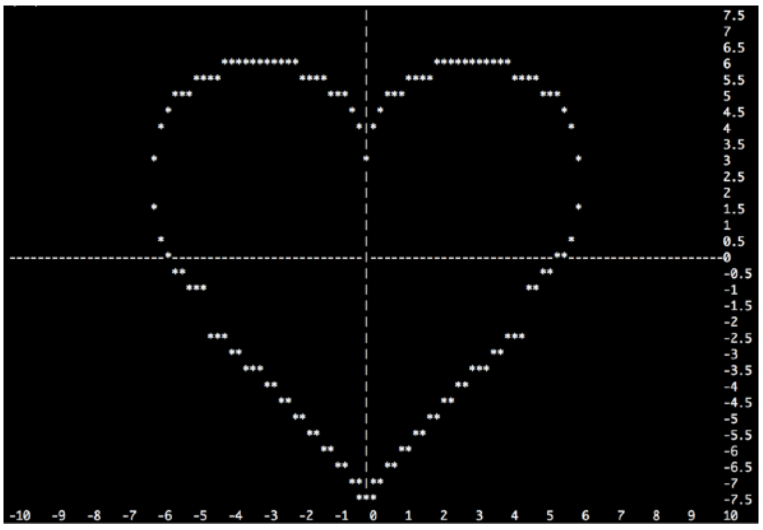
- the program still by most respects conforms to good functional programming practices (not printing as a by-product, no object oriented methods, etc). This is achieved by instead of displaying line-by-line, each line is an appended string of characters, and the lines are appended together in to one very large string. then, that string is printed.
- this method is probably tough on memory but easy on IO and largely conforms to functional programming

Testing

- some test cases are provided. face.txt uses circles to make a face, and heart.txt uses complex curves and conditionals to make a heart-shape (based on the formulas provided by https://shannonsookochoff.wordpress.com/valentines-math/graphing-a-heart-on-your-ti-xx/, 2011)


5
4.67
4.33
4
3.67
3.33
3
2.67
2.33
2
1.67
1.33
1
0.67
0.33
0
-0.33
-0.67
-1
-1.33
-1.67
-2
-2.33
-2.67
-3
-3.33
-3.67
-4
-4.33
-4.67
-5
-7
0
7


7
6.53
6.07
5.6
5.13
4.67
4.2
3.73
3.27
2.8
2.33
1.87
1.4
0.93
0.47
0
-0.47
-0.93
-1.4
-1.87
-2.33
-2.8
-3.27
-3.73
-4.2
-4.67
-5.13
-5.6
-6.07
-6.53
-7
-10 -9 -8 -7 -6 -5 -4 -3 -2 -1 0 1 2 3 4 5 6 7 8 9 10


7.5
7
6.5
6
5.5
5
4.5
4
3.5
3
2.5
2
1.5
1
0.5
0
-0.5
-1
-1.5
-2
-2.5
-3
-3.5
-4
-4.5
-5
-5.5
-6
-6.5
-7
-7.5
-10 -9 -8 -7 -6 -5 -4 -3 -2 -1 0 1 2 3 4 5 6 7 8 9 10