Prof. Dr. Visvanathan Ramesh
Alperen Kantarcı
Gamze Akyol

Goethe-Universität Frankfurt am Main
Institut für Informatik

Programmierparadigmen und Compilerbau

Sommersemester 2023

Exercise Nr. 4

Deadline Wednesday 22.06.2023, 10:00 in Moodle!

Question 0 (0 Points)

Read the following tips!

- Group work is **not allowed!** You can talk and discuss with your friends about the exercises, but you should solve the exercises **individually**. Plagiarism is not tolerated.
- You can use ChatGPT or other web sources but you cannot ask the whole question. You should be able to explain every step of your code.
- Upload all the files you want to submit into **a single .zip file on Moodle**. Please ensure that you **only submit .pdf and .hs files.**
- Please write your **matriculation number**, **name** and **group number** to each file that you are submitting.
- Your submission file name should be in the following order:

  Name_matriculationNumber_Group_GroupNumber_Exercise_ExerciseNumber.extension.

  For example **AlperenKantarci_1111111_Group_2_Exercise_4.zip**

Question 1 - Interpreter (= 10 Points)

In this exercise, you will write an interpreter for a basic Turtle Graphic language. Turtle Graphic was part of the original Logo programming language developed by Wally Feurzeig, Seymour Papert and Cynthia Solomon in 1967.

The 'turtle' is an on-screen cursor, which can be commanded to move around the screen, drawing as it goes. It is called a 'turtle' because the on-screen cursor can be represented in a simple form as a triangle or a turtle icon in more complex forms. Our turtle (or pen) has four directions to go: North, South, East, and West. It has four different colors: Red, Blue, Green, and Yellow.

The following data types are given to you in main.hs file. This file also contains the main function to you.

```haskell
data Direction = North | South | East | West
    deriving (Show, Eq)

data Color = Red | Blue | Green | Yellow
    deriving (Show, Eq)

data Command
    = Move Int      -- Move forward by a certain number of steps
    | Turn Direction    -- Turn to face a given direction
    | PenUp             -- Lift the pen (stop drawing)
```

```
  | PenDown           -- Lower the pen (start drawing)
  | SetColor Color    -- Set the color of the pen
  deriving (Show, Eq)
```

For the interpreter you will use a State which contains all necessary information for the given State.

```
data State = State
    { position :: (Int, Int)
    , direction :: Direction
    , pen :: Bool
    , color :: Color
    }
    deriving (Show, Eq)
```

You will implement the following functions to interpret this turtle language:

1. The 'move' function updates the position based on given direction. x is the coordinate of horizontal axis and value of x increases as you go east. y is the coordinate of vertical axis and value of y increases as you go south.
2. The 'run' function takes a list of commands and an initial state, and returns a list of line segments representing the trail left by the turtle
3. The interpret function takes a list of commands and an initial state, and returns the tuple of (final state, list of lines). If no line is drawn with the command, then it should have an empty list.

Here are some example inputs and outputs for the program:

Example 1

Input

```
cmds = [PenDown, Move 10, Turn East, Move 10, PenUp, SetColor Red, PenDown, Move 10, Move 10]
initialState = State (0, 0) North False Blue
```

The trail:

```
[[],[((0,0),(0,-10),Blue)],[],[((0,-10),(10,-10),Blue)],[],[],[],
↪  [((10,-10),(20,-10),Red)],[((20,-10),(30,-10),Red)]]
```

Final State:

```
State {position = (30,-10), direction = East, pen = True, color = Red}
```

Example 2

Input

```
cmds = [PenDown, Turn East, Move 50, PenUp, SetColor Blue, PenDown, Move 40, PenUp, SetColor
↪  Red, PenDown, Move 10, Move 20]

initialState = State (20, 20) South False Yellow
```

The trail:

```
[[],[],[((20,20),(70,20),Yellow)],[],[],[],[((70,20),(110,20),Blue)],[],[],[],
↪  [((110,20),(120,20),Red)],[((120,20),(140,20),Red)]]
```

Final State:

```
State {position = (140,20), direction = East, pen = True, color = Red}
```

1 Question 2 - Combinators (= 10 Points)

1.1 Drawing Shapes

```haskell
-- Code to build upon for the exercise
data Point = Point Int Int
data Line = Line Point Point
data Curve = Curve [Point]
data Polygon = Polygon [Point]
data Circle = Circle Point Int  -- Center and radius
data Ellipse = Ellipse Point Point  -- Two foci

data Shape = SPoint Point
           | SLine Line
           | SCurve Curve
           | SPolygon Polygon
           | SCircle Circle
           | SEllipse Ellipse

-- Move a point by dx in the x direction and dy in the y direction
movePoint :: Point -> Int -> Int -> Point
movePoint (Point x y) dx dy = Point (x + dx) (y + dy)

-- Move a shape by dx in the x direction and dy in the y direction
moveShape :: Shape -> Int -> Int -> Shape
moveShape (SPoint p) dx dy = SPoint (movePoint p dx dy)
moveShape (SLine (Line p1 p2)) dx dy = SLine (Line (movePoint p1 dx dy) (movePoint p2 dx
↪  dy))
-- ... continue this pattern for Curve, Polygon, Circle, Ellipse
```

(a) Complete the moveShape function to support Curve, Polygon, Circle, and Ellipse.

(b) Implement the scaleShape function that scales a shape by a given factor. For Point, this operation has no effect.

(c) Implement the rotatePoint function that rotates a point around the origin by a given angle (in degrees). Then, implement the rotateShape function that rotates a shape around the origin.