Exam – Introduction to Functional Programming

TDA555/DIT440, HT-20
Chalmers och Göteborgs Universitet, CSE

Day: 2020-10-28, *Time:* 14:00-18.00, *Place:* Canvas/Zoom

Course responsible
Alex Gerdes, available for questions via Zoom (see instructions below).

Allowed aids
Course literature, other books, the internet. You are not allowed to discuss the problems or share your answers with anyone!

Submitting
Submit your answers in *one single PDF or one or more Haskell source files*. Number every answer and start every answer on a new page!

You may answer the questions using pen and paper, which is fine! In that case, take a photo of your answer and paste into your answer document. Make sure the photo is readable!

Grading
There are four programming problems on the exam. You will need to solve at least *three* problems to pass the exam. You also need to write a short explanatory text in English or Swedish for each solution, describing your solution in your own words. Each solution is graded on three levels: Fail, Pass and Excellent.

Chalmers grading For a score of x/y where x is the number of Excellent and y is the number of Pass, the following minimum scores apply:

- For grade 3: 0/3 (At least three Passing solutions).
- For grade 4: 2/3 (At least two Excellent solutions, and at least one additional Pass solution).
- For grade 5: 3/4 (At least three Excellent solutions, and one additional Pass solution).

GU grading For a score of x/y where x is the number of Excellent and y is the number of Pass, the following minimum scores apply:

- For grade G: 0/3 (At least three Pass solutions).

- For grade VG: 2/4 (At least two excellent solutions, and two additional Pass solutions).

Expected quality A passing solution needs to be mostly correct, but code does not need to be executable. In this sense grading is the same as for a hall exam on paper, small mistakes will not impact your score if the intention is clear. The explanatory text needs to be sufficient to convince the grader that you understand the code you have written.

An excellent solution needs to be mostly correct, it needs to complete an additional task for the problem (stated in the problem description) and the explanatory text needs to correctly use the concepts taught in the course.

Inspection
Contact Alex about exam inspection.

Note

- Mark your document with your name and personnummer.
- You may write your answers in Swedish and English.
- Excessively complicated answers might be rejected.
- *Write legibly!* Solutions that are difficult to read are marked as Fail!

Online exam instructions

During the exam (14:00–18:00)

ID checking:

- At the beginning of the exam you will be invited to a "breakout room" in Zoom, where your ID will be checked.
- If you are kicked out of the Zoom meeting, then enter again as quickly as possible by using the same link as earlier. You will be ID checked again before being let in.

Do's and don'ts:

- You are allowed to use books and the internet.
- Don't communicate with anyone else during the exam, neither orally nor in any other format. This includes posting questions in chat forums, etc.
- Don't wear earbuds or earphones.
- Don't leave your seat.

Contacting the guard or the examiner:

- Contact the proctor using the Zoom chat channel to "Everyone" (the only possible choice).

- If you need to go to the toilet, inform the exam guard via the Zoom chat.
  - Write "Bathroom" before you go to the toilet.
  - Write "Bathroom return" when you're back.
- Write "Question for the examiner" if you have a question. The exam guard will tag your name with ##EXAMINER##. You will be invited to a breakout Zoom room, but it might take some time.
- If you need to contact the exam guard, write "Contact" in the Zoom chat. The proctor will tag your name with ##CONTACT##. You will be moved to another room to explain further. You will be invited to a breakout Zoom room, but it might take some time.

If you finish early:

- You have to stay logged in to Zoom until you are permitted to leave.
- If you finish earlier, write "Scanning solutions" in the chat before you scan your solutions and merge them into one single document. You only need to do this if you submit a PDF file.
- When your solution is submitted, write "Submitted in Canvas".
- When the exam guard has ticked you off and added ##DONE## before your Zoom name, you can leave the Zoom meeting. But not before that!

When the exam finishes (18:00–18:20)

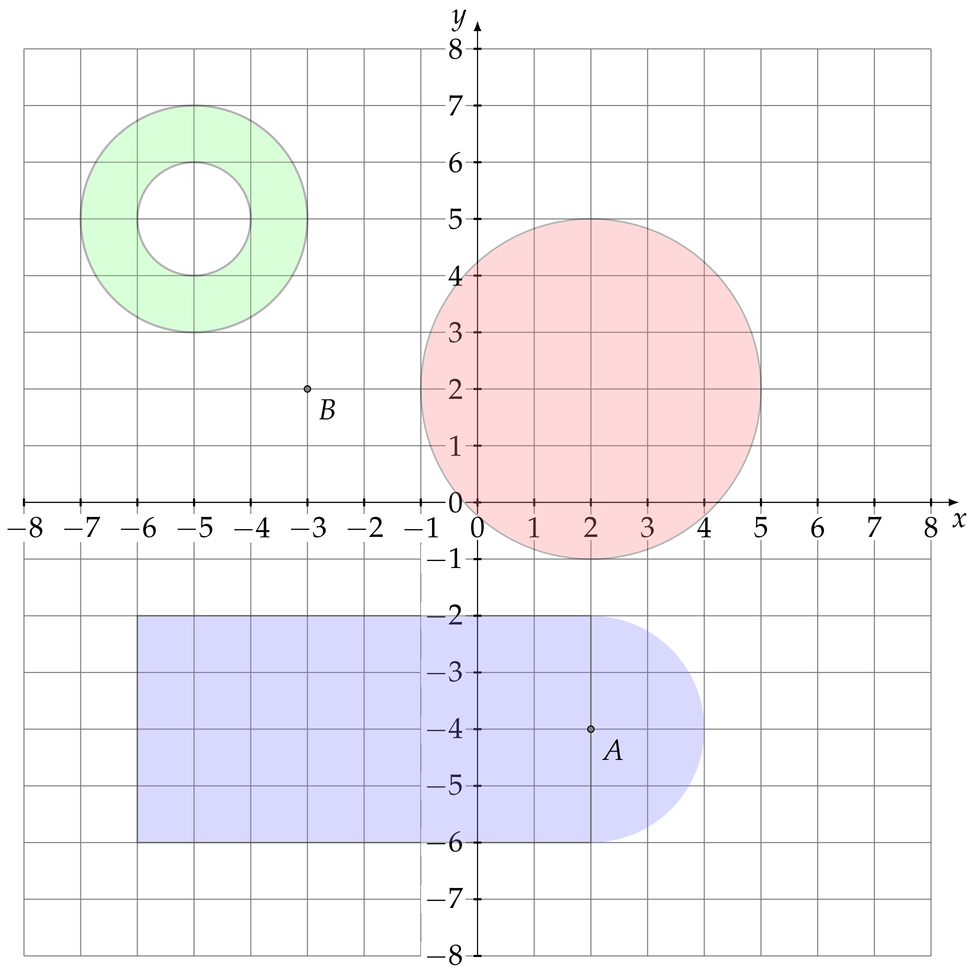
- The exam finishes at 18:00 sharp. After that you are not allowed to continue working.
- You have 20 more minutes to scan your answers and submit them to Canvas.

  You have to submit before 18:20, otherwise you automatically fail the exam.
- Submit your answers in Canvas as a single file upload to the examination assignment. Make sure that you:
  - Submit one single document in PDF, or one or more Haskell source files. Scanned answers should be inserted as images in a PDF document.
  - Name the document "exam-YYYYMMDD-NNNN.pdf", with YMDN replaced by your personnummer.
  - Start every answer on a new page in this document.
- After submitting, write "Submitted in Canvas" in the chat.
- If you have problems submitting to Canvas or creating a single document in the correct format, you may send your solutions in an email to Alex (alexg@chalmers.se). Use the subject "Exam TDA555-DIT440 solutions", and write your name and personnummer in the email.

  You must still do this before 18:20!

1 Regions

This problem evolves around *regions*. A region describes an area in a two-dimensional plane and will be used to determine if a point is contained within a particular region. A region can be constructed with circles, rectangles, and by combining regions, such as taking the conjunction of two regions or the taking the negation of a region. For example, the next figure displays some example regions:



The point A in the figure above is inside the blue region, whereas point B is not contained in any region.

Your task is to define several functions for creating regions and a number of related functions. Note that the tasks do not involve drawing any graphics, you will only treat regions as abstract representation of geometrical shapes and do some computations. We are going to represent a region as a *function*:

```
type Point  = (Double, Double)
type Region = Point -> Bool
```

That is, a `Region` is a function that takes a point of type `Point` as argument and returns

True if the point is contained within that region. For example, we can create a rectangular region centered around the origin $(0,0)$ as follows:

```
rectangle :: Double -> Double -> Region
rectangle w h = \(x, y) -> and [x >= -w', x <= w', y >= -h', y <= h']
 where
  w' = w / 2
  h' = h / 2
```

To check if a point is within a region we can use the following function:

```
inside :: Point -> Region -> Bool
p `inside` r = r p
```

that takes a point and a region, which is a function that takes a point and returns a boolean, and just applies the region (function) to the point.

You need to complete the following tasks:

a) Define a function that calculates the euclidian distance between two points:

```
dist :: Point -> Point -> Double
```

b) Implement the following function:

```
type Radius = Double

circle :: Radius -> Region
```

that creates a circular region with specified radius centered at the origin $(0,0)$.

c) Define a function that creates the negation of a given region

```
outside :: Region -> Region
```

that is, a point is considered to be in the resulting region if it is *not* inside the given region.

d) We want to combine regions into more complex regions. The blue region in the figure above is an example of a region that is result of the combination of a rectangle and a circle. Implement the following function that combines two regions:

```
combine :: Region -> Region -> Region
```

e) A related concept is proximity: how close are two points to each other. Your taks is to complete the following data and function definition:

```
data Proximity = ...

proximity :: Point -> Point -> Proximity
```

The `Proximity` data type expresses that a something is *close*, *near*, or *far*. The function `proximity` takes two points as argument and returns their proximity. If the points are more than 2 units apart from each other than they are considered to be far apart, if the points are between 2 and 1 unit apart then they are *near*, otherwise they are *close*. Note, that this function does not involve regions.

Explanatory text Write a brief explaining the choices you have made in your solution. You may write this as one or a few comments in a `.hs` file or in the `.pdf` file if you prefer.

1.1 For excellent

You only need to answer these questions if you aim for a higher grade.

f) Implement the function `move` that moves the center of a region to the given point:

```
move :: Point -> Region -> Region
```

g) Define a function that creates an *annulus* region:

```
annulus :: Radius -> Radius -> Region
```

The green region in the figure above is an example of an annulus. The function takes the radius of the outer and inner circle as input, in that order, and needs to check if the outer radius is larger than the inner one.

h) Give the declarations for the blue, green, and red regions present in the figure above.

```
redCircle, greenAnnulus, blueThing :: Region
redCircle    = ...
greenAnnulus = ...
blueThing    = ...
```

2 Keith numbers

To determine if a n-digit number N is a *Keith number*, you create a Fibonacci-like number sequence starting with the n digits in the number N, with the most significant digit coming first. The sequence then continues with terms, where each term is the *sum* of the n preceding terms. The number N is a Keith number if it is a member of the constructed number sequence, and has at least two digits. Consider for example the three digit number $N = 197$. The number sequence is as follows:

$$1, 9, 7, 17, 33, 57, 107, 197, 361, \ldots$$

The number 197 is member of the sequence and therefore a Keith number. If we take the the number $N = 123$ the sequence becomes:

$$1, 2, 3, 6, 11, 20, 37, 68, 125, \ldots$$

and we can conclude that the number 123 is *not* a Keith number, because it is not member of the sequence.

Your task is:

a) Write a function that takes a number as input and determines if it is a Keith number:

```haskell
isKeith :: Int -> Bool
```

Hint: write a help function that splits a number into its digits. The `Prelude` functions `mod` and `div` maybe useful here.

Explanatory text Explain how you solved this problem. If you use any help functions or variables (e.g. in a where-clause) explain what they are.

2.1 For excellent

For a higher grade you must do these additional tasks:

b) Write a function that abstracts the base number:

```haskell
isKeithBase :: Int -> Int -> Bool
```

such that we can use the function for number with a different base.

c) Define a function that generates an infinite list of Keith numbers:

```haskell
keithNumbers :: [Int]
```

3 Battle

Your task is to implement a part of an imaginary game that resembles the game Battleship. The game is about hitting pieces, such as a ship or a tank, within a limited number of tries. The computer will place a number of pieces randomly on a two-dimensional plane, and will ask the player to fire a missile in a given number of rounds. A player wins if she has succeeded to hit all pieces, otherwise the computer wins.

Note you are required to reuse some functions from Problem 1 in a number of tasks. You may assume that all functions work as stated, and you can do all tasks without having completed the tasks posed in Problem 1.

We need a data type to represent a piece on the two-dimensional plane. Your tasks are:

a) Define a data type to represent a piece:

```
data Piece = ...
```

A piece must contain a description of type `String` and a region of type `Region`, which we used in the Problem 1. Write a `Show` instance for the `Piece` data type as well.

b) Next you should complete the following definition:

```
allPieces :: [Piece]
allPieces = [heli, tank, ship]
 where
  tank = ...  -- A tank is a rectangle of size 2 by 3
  ship = ...  -- A ship is a rectangle of size 3 by 6
  heli = ...  -- A helicopter is a circle with a radius of 2
```

by replacing the dots (`...`) with an expression as described in the corresponding comment.

c) Now you need to define a function that moves (the region within) a `Piece` to the given point:

```
movePiece :: Point -> Piece -> Piece
```

d) Write a function that asks the user for a point, a x- and y-coordinate, and returns an `IO` instruction that returns a `Point`:

```
fire :: IO Point
```

Explanatory text Write a brief description of the `Piece` data type you wrote and give a short explanation of the functions you have implemented. Explain especially the help functions you have used, if any.

3.1 For excellent

To earn a higher grade you need to do the following tasks as well:

e) Write a QuickCheck generator:

```
genPiece :: Gen Piece
```

that randomly generates a `Piece`, which can be a tank, ship, or helicopter. The piece needs to be placed at a random position, where the x- and y-coordinate both lie between -10 and 10.

f) Define the function:

```
play :: Int -> [Piece] -> IO [Piece]
```

that plays a number of rounds. The maximum number of rounds is given as the first argument to the function. The second argument are the pieces that are present in the cartesian plane and need to be hit to win the game. The `play` function asks the user to fire at a particular point, using the `fire` function you defined earlier, and reports which pieces are hit, if any. After the maximum number of tries or if all pieces are hit, the function will return. The function `play` returns the pieces that have not been hit.

4 Encode

A URL is the address of, for example, a web page or a web service. If you want to transmit a URL over the Internet you sometimes you need to *encode* the URL, because it may contain characters that are not safe to transmit. To make a URL safe for transmission you need to 'escape' all unsafe ASCII characters, that is replace the unsafe characters with a different encoding. For example:

```
"http://alex.nl/age?input=42"  =>  "http%3A%2F%2Falex.nl%2Fage%3Finput%3D42"
```

Your tasks are:

a) Define the following function:

```haskell
escape :: Char -> String
```

that given a character returns a string according to the following table:

':'	⇒	"%3A"
'/'	⇒	"%2F"
'?'	⇒	"%3F"
'='	⇒	"%3D"

if the character is not present in the table a string containing just the input character is returned.

b) Implement the following function:

```haskell
type URL = String

encodeURL :: URL -> URL
```

that encodes a URL such that the unsafe characters are escaped. Use the function `escape` you just defined.

Explanatory text Explain how you solved this problem. If you use any help functions or variables (e.g. in a where-clause) explain what they are.

4.1 For excellent

Do the following additional tasks for a higher grade:

c) Define the `encodeURL` function in terms of a higher-order function, if you haven't already done so.

d) Write a QuickCheck property that tests the `encodeURL` function.