This coursework is about operations on points in the 2 dimensional plane. We represent a point by a pair \((x,y)\) where \(x\) and \(y\) are the cartesian coordinates (measured in cm, say):

type Pt = (Float,Float)

We let the \(x\)-axis of our coordinate system point to the right and the \(y\)-axis point upwards.

**Question 1.** Define the following functions:

- `dist` computes the distance of two points
- `close` tests if the distance of two points is less then 1
- `reflectX` reflects a point at the \(x\)-axis
- `reflectY` reflects a point at the \(y\)-axis
- `reflect` reflects a point at the origin
- `rotate` rotates a point by a given angle (given in radian) around the origin
- `reflect1` like `reflect`, but defined in terms of `reflectX` and `reflectY`
- `reflect2` like `reflect`, but defined in terms of `rotate`

\[40 \text{ marks}\]

**Question 2.** Consider the following functions:

```
project :: Pt -> Pt
project (x,y) = (x,0)
```

```
project1 :: Pt -> Pt
project1 p = (fst p,0)
```

Both functions project a point onto the \(x\)-axis. Nevertheless they are not equivalent. This can be seen by evaluating the expressions `snd (project crazy)` and `snd (project1 crazy)` where

```
crazy :: Pt
crazy = crazy
```

Explain the different behaviours by analysing the respective reductions. \[20 \text{ marks}\]
**Question 3.** Define a function `quadrant` that returns for every point \( p \) a string indicating in which quadrant of our coordinate system \( p \) lies (for example, "Point in 2. quadrant" could be returned). The quadrants are counted counterclockwise beginning with the upper right-hand one. If \( p \) lies on the \( x \)- or \( y \)-axis, then the string "Point on axis" should be returned. [20 marks]

**Question 4.** We represent pixels by integers and centimetres by floating point numbers, and we assume that 1 cm corresponds to 25 pixels.

Define functions `pixelToCm` and `cmToPixel` that translate from pixel to cm and vice versa. [20 marks]

**Question 5.** The Haskell Graphics Library uses pixel coordinates

```haskell
type Point = (Int,Int)
```

where the point \((0,0)\) marks the left-hand upper corner of the window.

If we define some global names for our window size

```haskell
xWin, yWin :: Int
xWin  = 600
yWin  = 500
```

then the coordinate of the lower right-hand corner is \((xWin,yWin)\).

Define functions `toPoint :: Pt -> Point` and `toPt :: Point -> Pt` that translate between the window’s and our coordinate system assuming that the origin of our coordinate system lies in the centre of the window. [20 marks]

**Due date:** 18 October 2004

**Notes:**
1. Functions must be defined in Haskell, must have a signature and must be tested.
2. Hand in a printout of your solutions in room 206.
3. Use the template available at [http://www.cs.swan.ac.uk/~csulrich/cs221html](http://www.cs.swan.ac.uk/~csulrich/cs221html).
4. Each solution must contain your printed name, code of the course (cs221), number of coursework, printing date and must have a signed submission form attached.
5. Late submissions will be penalised by taking off marks.
6. Lab classes start on Monday, 4th of October, in three sessions: 2-3pm, 3-4pm, 4-5pm.