Lab 4 : Laziness

Task:
- Create file (username)\-lab04.hs
- Make sure lab no. and your name is in end-of-line comment at start

Submission
1. Hand in signed printout of sources into School Office by 12noon, Fri Oct 29th 2010
2. Attach source into email message and send
   To: Andrew.Butterfield@cs.tcd.ie
   Subject: CS4011 Lab4

Graph Building

- Given a graph, and names for selected nodes, we can write a corresponding \texttt{let}

\begin{center}
\begin{tikzpicture}
\node (1) at (0,0) {1};
\node (2) at (1,-1) {2};
\node (3) at (2,-2) {3};
\draw (1) edge[bend right] (2);
\draw (2) edge[bend right] (3);
\end{tikzpicture}
\end{center}

- $g0 = \texttt{let } a = 1:b$
  
  $b = 2:c$

  $c = 3:a$

  \texttt{in } a$

Lab 4 Q1

Write code to generate the following graph:

\begin{center}
\begin{tikzpicture}
\node (a) at (0,0) {'a'};
\node (b) at (1,-1) {'b'};
\node (c) at (2,-2) {'c'};
\draw (a) edge[bend right] (b);
\draw (b) edge[bend right] (c);
\end{tikzpicture}
\end{center}

$g1 = \ldots$

Can you display finite parts of this? If so, how?
(Answer as comments in your program)
Lab 4 Q2

Write code to generate the following graph:

```
\[
1 \quad 2
\]
\[
\text{a} \quad \text{b} \quad \text{c} \quad \text{d}
\]
```

g2 = ...

Can you finite parts of this? How?
What is its type?
Can you describe it using natural language?
Can you apply `length` to the whole thing?

Lab 4 Q3 (Task)

Re-implement the Queue as a *doubly-linked* list:

```
type C_Queue t = ???
c_mtq :: C_Queue t
  c_mtq = ???
c_enqueue :: t -> C_Queue t -> C_Queue t
c_enqueue x q = ???
c_dequeue :: C_Queue t -> (t,C_Queue t)
c_dequeue q = ???
```

You will need to draw pictures to work this out.
Include those pictures with the printout that you submit
Good luck!

Lab 4 Q3 (Background)

Consider a Queue implemented as a linked list:

```
type A_Queue t = [t]
a_mtq :: A_Queue t
  a_mtq = []
a_enqueue :: t -> A_Queue t -> A_Queue t
da_enqueue x q = x::q
da_dequeue :: A_Queue t -> (t,A_Queue t)
da_dequeue [] = error "cannot dequeue empty queue"
da_dequeue [x] = (x,[])
da_dequeue (x:xs) = (y,x:ys) where (y,ys) = a_dequeue xs
```

Putting stuff in the queue is easy and cheap.
However, taking stuff out is long and complicated
This is because `[t]` gives us a *singly-linked* list.