UNIVERSITY OF MALTA
BOARD OF STUDIES FOR INFORMATION TECHNOLOGY/
FACULTY OF SCIENCE
Department of Computer Science & AI
B.Sc. I.T.(Hons.) / B.Sc.(Hons.) – I Year
May/June 2004 Assessment Session

CSA1080: Declarative Programming xxth June 2004

xx:xx–xx:xx

Answer question 1 and 4, and any other two questions. Each question carries 25 marks. Students are allowed to use course notes, books and calculators.

Intentionally blank
Intentionally blank
Section B: Functional Programming

4. (a) The following allows the extension of normal numbers to allow the manipulation of infinite numbers:

```haskell
data Infinite a = Finite a
  | PositiveInfinity
  | NegativeInfinity
```

Show your understanding of the above datatype by defining two functions `negateInf` and `mulInf` which calculate the negation$^1$ and multiplication respectively, on objects of type `Infinite Integer`.

(b) What does the following function do? What is the function’s type?

```haskell
wonder [] = []
wonder (f:fs) = f 1: wonder fs
```

(c) Give the type of the following three functions:

```haskell
example0 fs xs = [ f x | f <- fs, x <- xs ]
example1 f [] = ['!']
example1 f (x:xs) = example1 f xs ++ [f x]
example2 x [] = [x]
example2 x (y:ys)
  | x==y = example2 x ys
  | otherwise = y:example2 x ys
```

$^1$The negation of $x$ is $-x$. 
5. (a) Consider the following definitions:

\[
data Tree a = \text{Empty} \mid \text{Node } a \ (\text{Tree } a, \text{Tree } a)
\]

\[
mystery f \ \text{Empty} = \text{Empty}
\]
\[
mystery f \ (\text{Node } x \ (\text{left}, \text{right})) =
\]
\[
\text{Node } (f \ x) \ (\text{left}', \text{right}')
\]
\[
\text{where}
\]
\[
\text{left}' = \mystery f \ \text{left}
\]
\[
\text{right}' = \mystery f \ \text{right}
\]

Give the type of the function \(\text{mystery}\) and explain (possibly through the use of examples) what it does.

(b) Give all the types of the functions you define in this part of the question.

Define a function \(\text{fromTo}\), which takes two integers and returns the list of integers starting from the first number and finishing with the second. For example, \(\text{fromTo } 2 \ 5\) would return \([2, 3, 4, 5]\).

Define a function \(\text{dots}\) which takes an integer parameter \(n\) and returns a string consisting of \(n\) dots. For example, \(\text{dots } 4\) would return \("...."\).

Using these two definitions, define a function \(\text{triangle}\) which takes an integer parameter \(n\) and returns a list of strings, with the first element being a single dot, the second two dots, and so on, with the last string in the list consisting of \(n\) dots. For example, \(\text{triangle } 4\) would return \(['.','.','..','....']\).

6. Consider the following definition of a datatype to handle exceptions when calculating a value:

\[
data \text{Exception } a = \text{Result } a \mid \text{Error } \text{String}
\]

For example, if we had a function which returns an \text{Integer} but is not defined on all inputs, we could modify it to return an object of type \text{Exception Integer}. The new function would work just like the old function (returning \text{Result } n\) when the old function returned \(n\), but
in the case of an error, it allows us to return a string explaining the origin error.

(a) Recall that \texttt{div} returns the integer division of two integers. This may cause a runtime error if the second number is zero. Define a function \texttt{safeDiv} of type \texttt{Int \rightarrow Int \rightarrow Exception Int}, which acts just like the Haskell function \texttt{div}, but returns the value \texttt{(Error "Division by zero")} whenever the second parameter is zero.

(b) A computation returns a list of results, all of which are of type \texttt{Exception a}. We would like a function \texttt{compute} which returns (i) the first error in the list, if one exists or (ii) \texttt{Result xs} if no errors occur in the given list, and \texttt{xs} is the list of actual results. For example:

\begin{verbatim}
compute [Result 5, 
        Error "Division by zero", 
        Error "Out of memory", 
        Result 4]
\end{verbatim}

would return \texttt{Error "Division by zero"} (the first error), while:
\begin{verbatim}
compute [Result 5, Result 4, Result 1, Result 4]
\end{verbatim}
would return \texttt{Result [5, 4, 1, 4]}. Give the type of \texttt{compute} and define it.