Coláiste na Tríonóide, Baile Átha Cliath
Trinity College Dublin
Ollscoil Átha Cliath | The University of Dublin

Faculty of Engineering, Mathematics and Science

School of Computer Science & Statistics

Integrated Computer Science — Trinity Term 2018
B.A. (Mod.) Computer Science & Business
M.S.I.S.S.
Junior Sophister Annual Examination

Introduction to Functional Programming

Tuesday, 1st May — Sports Centre — 14:00–16:00

Dr Andrew Butterfield

Instructions to Candidates:

- Attempt two questions from Section A. Each counts for 35% of this exam.
- Attempt all 15 of the multiple choice questions in Section B. This section counts for 30% of this exam.
  - All questions in Section B carry equal marks (2%).
  - Each correct answer in Section B is awarded 2%.
  - Each incorrect answer in Section B reduces the marks awarded by 0.5%
  - Each blank answer in Section B is awarded 0.0%
  - Answer each question in Section B on an A-E Multiple Choice Answer Form.
- There is a Reference section at the end of the paper (pp10–12).

Materials required for this examination:

An A-E Multiple Choice Answer Form is required.

Section A

1. Give a complete implementation of the Prelude functions described below. By "complete" is meant that any other functions used to help implement those below must also have their implementations given.

   (a) Returns the list with its first element removed, if it is non-empty, with a runtime error otherwise.

   ```
   tail              :: [a] -> [a]
   ```

   [4 marks]

   (b) Concatenate two lists together.

   ```
   (++) :: [a] -> [a] -> [a]
   ```

   [5 marks]

   (c) returns everything but the last element of a list, if it is non-empty, with a runtime error otherwise.

   ```
   init              :: [a] -> [a]
   ```

   [6 marks]

   (d) Reverse its list argument

   ```
   reverse           :: [a] -> [a]
   ```

   [6 marks]

   (e) Uses a predicate to split a list into two, the first list being the longest prefix that *does not satisfy* the predicate, while the second list is what remains

   ```
   break             :: (a -> Bool) -> [a] -> ([a],[a])
   ```

   [7 marks]

   (f) Compute the maximum of a non-empty list

   ```
   maximum           :: Ord a => [a] -> a
   ```

   [7 marks]

2. We have a binary search tree built from number-string pairs, ordered by the number (acting as key),

```
data Tree = Empty
          | Single Int String
          | Many Tree Int String Tree
```

and one function `search` partially defined over it:

```
search :: Int -> Tree -> String

search x (Single i s)
 | x == i  =  s

search x (Many left i s right)
 | x == i  =  s
 | x  > i  =  search x right
 | x  < i  =  search x right
```

(a) Describe the two ways in which function `search` can fail, with Haskell runtime pattern-matching errors. [8 marks]

(b) Add in error handling for function `search` above, using the `Maybe` type, to ensure this function is now total. Note that this will require changing the type of the `search` function. [12 marks]

(c) Add in generic error handling for the `search` function above, using monads, ensuring it is now total, and giving back a useful error message. Note that this will also require changing the type (again) of the `search` function.

[15 marks]

3. (a) Consider the following function definition:

```
length [] = 0
length (x:xs) = 1 + length xs
```

Use the shorthand Abstract Syntax Tree (AST) notation to show how the application `length [3,39]` is evaluated, indicating clearly where copying takes place. You need not draw the full AST (with cons-nodes) for the lists but just show any list instead as a single node, `[]`, `[3]`, etc, as appropriate. [10 marks]

(b) Consider the following function definitions:

```
evenup n = n : evenup (n+2)
take 0 xs = []
take n (x:xs) = x : take (n-1) xs
```

Show the evaluation of `take 2 (evenup 2)` using both *Strict* Evaluation and *Lazy* Evaluation. Show enough evaluation steps to either indicate the final result, or to illustrate why no such result will emerge. [8 marks]

(c) Write a function `toDOS` that takes a string representing a filename (root.ext) and converts to one satisfying the DOS 8+3 format (All uppercase, root and extension with max length of 8 and 3 respectively). If the root or extension needs to be shortened, then they should be truncated. For example: `toDOS "complicatedName.textfile" = "COMPLICA.TEX"`

You may make use of any Prelude functions (See Reference pp10–12). [7 marks]

(d) Write a program that prompts the user for a filename and then uses `toDOS` to convert the filename to DOS 8+3 format, opens that file, reads its contents, maps all its characters to lowercase, and outputs the result to file "LOWER.OUT".

You may make use of any Prelude functions (See Reference pp10–12). [10 marks]

Reference

Prelude List Functions

```
map          :: (a -> b) -> [a] -> [b]
(++)         :: [a] -> [a] -> [a]
filter       :: (a -> Bool) -> [a] -> [a]
concat       :: [[a]] -> [a]
head         :: [a] -> a
tail         :: [a] -> [a]
last         :: [a] -> a
init         :: [a] -> [a]
null         :: [a] -> Bool
length       :: [a] -> Int
(!!)         :: [a] -> Int -> a
foldl        :: (a -> b -> a) -> a -> [b] -> a
foldl1       :: (a -> a -> a) -> [a] -> a
scanl        :: (a -> b -> a) -> a -> [b] -> [a]
scanl1       :: (a -> a -> a) -> [a] -> [a]
foldr        :: (a -> b -> b) -> b -> [a] -> b
foldr1       :: (a -> a -> a) -> [a] -> a
scanr        :: (a -> b -> b) -> b -> [a] -> [b]
scanr1       :: (a -> a -> a) -> [a] -> [a]
iterate      :: (a -> a) -> a -> [a]
repeat       :: a -> [a]
replicate    :: Int -> a -> [a]
cycle        :: [a] -> [a]
take         :: Int -> [a] -> [a]
drop         :: Int -> [a] -> [a]
splitAt      :: Int -> [a] -> ([a],[a])
takeWhile    :: (a -> Bool) -> [a] -> [a]
dropWhile    :: (a -> Bool) -> [a] -> [a]
span, break  :: (a -> Bool) -> [a] -> ([a],[a])
```

Other Common Functions

```
even, odd :: Integral a -> a -> Bool
chr :: Int -> Char
ord :: Char -> Int
lines :: String -> [String]
words :: String -> [String]
unlines :: [String] -> String
unwords :: [String] -> String
```

Prelude Type Constructors

```
data Maybe t = Nothing | Just t
data Either a b = Left a | Right b
```

Some Prelude Classes

```
class Eq a where
  (==), (/=) :: a -> a -> Bool

class Eq a => Ord a where
  compare              :: a -> a -> Ordering
  (<), (>=), (>), (<=) :: a -> a -> Bool
  max, min             :: a -> a -> a

class Num a where
  (+), (*), (-) :: a -> a -> a
  negate        :: a -> a
  abs           :: a -> a
  signum        :: a -> a
  fromInteger   :: Integer -> a
```

Data.Char Functions

```
isControl :: Char -> Bool
isSpace :: Char -> Bool
isLower :: Char -> Bool
isUpper :: Char -> Bool
isAlpha :: Char -> Bool
isAlphaNum :: Char -> Bool
isPrint :: Char -> Bool
isDigit :: Char -> Bool
toUpper :: Char -> Char
toLower :: Char -> Char
toTitle :: Char -> Char
digitToInt :: Char -> Int
intToDigit :: Int -> Char
ord :: Char -> Int
chr :: Int -> Char
```

Prelude IO Functions

```
type  FilePath = String

putChar     :: Char -> IO ()
putStr      :: String -> IO ()
putStrLn    :: String -> IO ()
print       :: Show a => a -> IO ()
getChar     :: IO Char
getLine     :: IO String
getContents :: IO String
readFile    :: FilePath -> IO String
writeFile   :: FilePath -> String -> IO ()
```