



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
B.A. (Mod.) Computer Science & Business
M.S.I.S.S.
Year 3 Multiple Choice Dry-run

Sample Paper 2018

Introduction to Functional Programming

Lá, Uimhir, Mhí

Áit

hh:mm–hh+2:mm

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 (pp8–9).

Materials required for this examination:

An A-E Multiple Choice Answer Form is required.

Section A

Three questions similar to those in past papers, with an adjustment per question from 33 to 35 marks each. This adjustment does not materially affect the content or level of difficulty of those questions.

Section B

1. What is the value of the following expression?

$6 + 12 \text{ 'div' } 5 - 4$

- a. 4
- b. -1
- c. 0
- d. 1
- e. 18

2. What is the type of the following expression?

`chr (head $ tail $ map (+1) $ filter even [1..10])`

- a. [Int]
- b. Int
- c. [Char]
- d. Num a => [a]
- e. Char

3. Given the following parentheses-free expression, which of the following parenthesised expressions is the same?

$a + b/c - d * f \ e \ g + h \ j$

- a. $(a + b)/(c - d) * ((f \ e) \ g) + (h \ j)$
- b. $(a + b)/((c - d) * f) \ (e \ g) + (h \ j)$
- c. $a + (b/c) - (d * (f \ (e \ g))) + (h \ j))$
- d. $a + (b/c) - (d * ((f \ e) \ g)) + (h \ j)$
- e. $a + (b/c) - (d * (f \ e \ (g + h) \ j))$

4. Which of the following expressions has a type error?
- a. `head [1,2,3] + 4`
 - b. `tail [1,2,3] ++ [4]`
 - c. `init [1,2,3] + 4`
 - d. `last [1,2,3] + 4`
 - e. all of the above
5. Which of the following expressions does not have a type error?
- a. `head [1,2,3] ++ [4]`
 - b. `tail [1,2,3] ++ 4`
 - c. `init [1,2,3] + 4`
 - d. `last [1,2,3] ++ 4`
 - e. none of the above
6. Which of the following expressions has the type `[String]`
- a. `[tail $ head ["Hello"]]`
 - b. `tail $ head [[], "Hello"]`
 - c. `head $ tail "Hello"`
 - d. `head $ tail ["Hello"]`
 - e. `tail [head $ tail "Hello"]`
7. Which clause of this pattern match succeeds for `leap 2016`?

```

leap y | y `mod` 400 == 0 = True    -- clause 1
      | y `mod` 100 == 0 = False   -- clause 2
      | y `mod`   4 == 0 = True    -- clause 3
      | otherwise        = False   -- clause 4

```

- a. clause 1
- b. clause 2
- c. clause 3
- d. clause 4
- e. none of the above

8. Which clause of this pattern match succeeds for `sw 42 [9]`?

```
sw _ [] = False -- clause 1
sw c (x:xs) | c < x = False -- clause 2
              | c == x = True -- clause 3
              | otherwise = False -- clause 4
              | c > x = False -- clause 5
```

- a. clause 1
- b. clause 2
- c. clause 3
- d. clause 4
- e. clause 5

9. Which of the following expressions result in a runtime error?

- a. `head (tail [1..1000000])`
- b. `tail (head [], [1])`
- c. `last (init [1..1000])`
- d. `init (last [], [1])`
- e. `tail (head [[1], []])`

10. What is the full Haskell type for the `lkp` function below?

```
lkp _ [] = Nothing
lkp x ((y,z):ys) | x == y = Just z
                  | otherwise = lkp x ys
```

- a. `Eq a => a -> [(a, a)] -> Maybe a`
- b. `Ord a => a -> [(a, b)] -> Maybe b`
- c. `(Eq a, Ord a) => a -> [(a, b)] -> Maybe b`
- d. `Eq a => a -> [(a, b)] -> Maybe b`
- e. `a -> [(a, b)] -> Maybe b`

11. In order to make `Exp` an proper instance of `Num`, what extra variants need to be added to the datatype?

```
data Exp = Nmb Int      -- number
        | Var String   -- variable
        | Add Exp Exp  -- add two Exp
        | Sub Exp Exp  -- subtract second Exp from first
        | Sgn Exp      -- signum of Exp
```

- a. `Mul Exp Exp | Dvd Exp Exp`
 - b. `Neg Exp | Mul Exp Exp | Abs Exp`
 - c. `Abs Exp | Neg Exp`
 - d. `Neg Exp | Dvd Exp Exp | Def String Exp Exp`
 - e. none of the above
12. What is the full Haskell type for the `mlkp` function below?

```
mlkp _ []                = Nothing
mlkp x ((y,z):ys) | x == y = Just z
                    | otherwise = mlkp x ys
```

- a. `(Monad m, Eq t) => t -> [(s,t)] -> m t`
- b. `(Monad t, Eq s) => t -> [(s,t)] -> m t`
- c. `(Monad m, Eq s) => s -> [(s,t)] -> m t`
- d. `Monad m => t -> [(s,t)] -> m t`
- e. `(Monad t, Ord s) => t -> [(s,t)] -> m t`

13. Which reduction step in the sequence below is not in lazy reduction order?

```

take 3 (from 42)
  =1= take 3 (42:from (42+1))
  =2= 42 : take (3-1) (from (42+1))
  =3= 42 : take 2 (from (42+1))
  =4= 42 : take 2 (from 43)
  =5= 42 : take 2 (43:from (43+1))
  =6= 42 : 43 : take (2-1) (from (43+1))
  =7= 42 : 43 : take 1 (from (43+1))
  =8= 42 : 43 : take 1 ((43 + 1) : from (43+1))

```

- a. =1=
- b. =3=
- c. =4=
- d. =7=
- e. =8=

14. Under which forms of evaluation will the following expression produce a concrete list?

```
take 4 threes where threes = 3:threes
```

- a. strict only
- b. lazy only
- c. neither lazy nor strict
- d. both lazy and strict
- e. none of the above

15. Under which forms of evaluation will the following expression return some form of value, and what will that value look like?

```
drop 4 threes where threes = 3:threes
```

- a. lazy, result is threes
- b. lazy only, result is [3,3,3,...,3]
- c. neither lazy nor strict, result is undefined
- d. strict only, result is [3,3,3,...,3]
- e. both lazy and strict, result is threes

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
```

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 ()
```

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
```