UPPSALA
UNIVERSITET

Department of Information Technology

INSTRUCTIONS

Please check that you have the correct exam!

This sheet should always be submitted, even if you haven't solved any of the exam problems.

Each solution should be written on a separate paper.

Write your exam code on each paper.

Please use only *one side* of the papers and do not use a pencil with red colour.

Sort the solutions in question order, with question 1 first, before you submit them.

FRONT SHEET FOR EXAMS	**DATE:** 2022-06-10
Course name 1DL201 Programkonstruktion och Datastrukturer	
Your exam code	
Semester when you were first registered to the course:	Programme:
Time for submitting the exam:	Table number:

No.	Solved problems (mark with X)	Points earned
1		
2		
3		
4		
5		
6		
7		
8		
9		
10		
11		
12		
13		
14		
15		
16		
17		
18		
19		
20		
21		
22		
23		
24		
25		
26		
27		
28		
29		
30		

Comments from the teacher

Σ

☐ Exam with bonus points: Grade is not shown.[1]

5 ≥ ☐ 4 ≥ ☐ 3 ≥ ☐

Exam grade:

1 The final result (points including bonus points and grade) can be found in Ladok after certification.

Final Exam (Part 2)

Program Design and Data Structures (1DL201)

Teachers: Johannes Borgström and Eva Darulova

2022-06-10 / 14:00 - 18:00

Instructions

Read and follow these instructions carefully to increase your chance of getting good marks.

- This is a closed-book exam. You may use a standard English dictionary. Otherwise, **no notes, calculators, mobile phones, or other electronic devices are allowed**.
- Write your **exam code** at the top of **each page**.
- There are **16 questions**, in two parts.
  1. There are 9 questions in part 1. Each question can award two points: one point for the correct answer, and one point for an explanation/justification of your answer. Your justification should relate the information given in the question, the course content, and your answer.
  2. There are 7 questions in part 2. Each question can award the stated number of points (2 or 4). Your solution should make suitable references to the course content and the information given in the question, and have a clear structure. Code that you write should conform to the coding convention.

  You can obtain a maximum of **40 points**.
- On the ordinary exam, the grade limits were 17p for grade 3, 24p for grade 4 and 32p for grade 5. This reexam should have similar limits.
- Write your answers directly on the question sheet.
- Read the questions carefully, and watch out for negations (**not, except**, etc.).
- Johannes/Eva will come to the exam hall around 15:00 to answer questions.

Part 1

Question 1: What kind of type declaration is most suitable for giving an informative name `T` to integers modulo 17, if you want that the arithmetic operators (+, -, *, /) on `T` perform modular arithmetic (mod 17): **`type`** or **`data`**?
Answer (1 point):

Your Justification (1 point):

Question 2: What is the worst-case time complexity of insertion into a binomial heap with n elements?
Answer (1 point):

Your Justification (1 point):

Question 3: Recall the `RBTree` inductive data type:

```
{-  RBTree represents red-black binary search trees with integer keys.
    Void represents an empty red-black tree.
    Node c x l r represents a non-empty red-black tree with root node color c,
    root label x, left subtree l and right subtree r.
   INVARIANT:
     Every red-black tree is a binary search tree, and
     MISSING, and
     every path from the root to an empty subtree contains the same number of black nodes.
 -}
data RBTree = Void | Node Color Int RBTree RBTree
```

What is the MISSING part of the invariant?

Answer (1 point):

Your Justification (1 point):

Question 4: Recall the Stack data structure with the following interface:

```
module Stack (Stack(), empty, isEmpty, push, top, pop) where

empty :: Stack a
isEmpty :: Stack a -> Bool
push :: a -> Stack a -> Stack a
top :: Stack a -> a
pop :: Stack a -> (a,Stack a)
```

and assume the following import statement (e.g. in ghci or a separate file): `import Stack`
For the following expressions, state the result of the expression, or explain why the expression results in an error:

Answer (4 points):

```
top $ snd $ pop $ push 3 $ snd $ pop $ push 7 empty
```

```
isEmpty $ snd $ pop $ push 2 $ push 1 empty
```

Question 5: Consider the following function:

```
1  testFnc :: Int -> Int -> Int
2  testFnc 0 acc = acc
3  testFnc n acc = testFnc (n - 1) (acc + 1)
```

and suppose that a breakpoint was set in the terminal: `:break 3`. The numbers on the left of the code show the line numbers in the file. What value does variable `n` have when running `testFnc 3 0` and when the program execution stops?

Answer (1 point):

Your Justification (1 point):

Question 6: Which of the following expressions have side effects?

(a) `length "hello"`
(b) `putStrLn "hello"`
(c) `openFile "hello.txt" ReadMode`

Answer (1 point):

Your Justification (1 point):

Question 7: What are the possible results of the following function `twoCoins`:

```
import System.Random

twoCoins :: StdGen -> (Bool, Bool)
twoCoins gen =
  let (firstCoin, g') = random gen
      (secondCoin, _) = random gen
  in (firstCoin, secondCoin)
```

Answer (1 point):

Your Justification (1 point):

Question 8: What gets printed to the screen when running the following `main` function?

```
import Data.Array.IO
type Array a = IOArray Int a

main = do
  arr <- newArray (1,5) 42 :: IO (Array Int)
  writeArray arr 2 0
  a <- readArray arr 1
  b <- readArray arr 2
  print (a,b)
```

Answer (1 point):

Your Justification (1 point):

Question 9: What is the worst-case time complexity of Kosaraju's algorithm for computing the strongly connected components of a graph, for graphs with n nodes and k ~~vertices~~ edges? Assume that the graph is represented as adjacency lists.

Answer (1 point):

Your Justification (1 point):

Turn page for Part 2.

Part 2

Question 10: Recall the `BinoTree` data type for binomial trees:

```
{- BinoTree is a binominal tree: a min-heap of a particular structure.

   in the binomial tree Node r v ts,
   r is the rank of the tree;
   v is the key at its root;
   ts is the list of children.

   INVARIANT: ...
   -}
data BinoTree = Node Int Int [BinoTree]
```

Write the missing cases of the `link` operation. Make sure to preserve the min-heap property.

Answer (2 points):

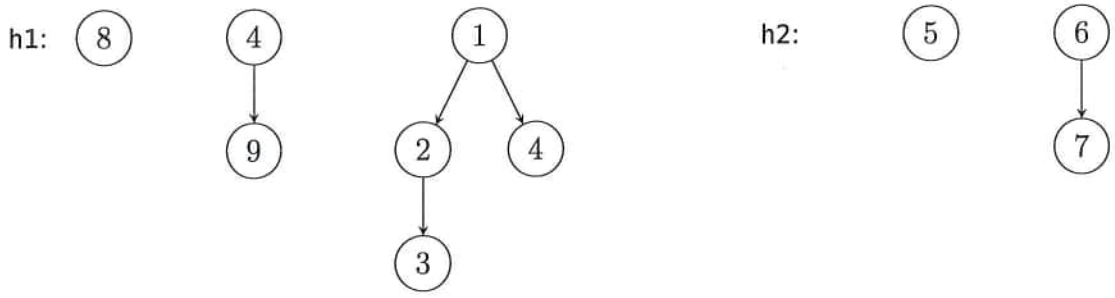
```
{- link t1 t2
   PRE:  rank t1 == rank t2
   RETURNS: a binomial tree containing the union of the elements of t1 and t2
 -}
link :: BinoTree -> BinoTree -> BinoTree

link t1@(Node r1 k1 ts1) t2@(Node r2 k2 ts2)

    |

    |
```

Question 11: Give the result of `merge h1 h2` where h1 and h2 are the following binomial heaps:



Answer (4 points):

Question 12: Recall the `Table` abstract data type:

```
-- in file Table.hs
module Table(Table(), empty, insert, exists, lookup, delete, iterate) where
import Prelude hiding (lookup, iterate)

empty :: Table k v
insert :: Eq k => Table k v -> k -> v -> Table k v
exists :: Eq k => Table k v -> k -> Bool
lookup :: Eq k => Table k v -> k -> Maybe v
delete :: Eq k => Table k v -> k -> Table k v
iterate :: Table k v -> (b -> (k, v) -> b) -> b -> b

newtype Table k v = T [(k, v)] deriving Eq
```

and assume the following import statements (in a separate file):

```
-- in file Example.hs
import Table
import Prelude hiding (lookup, iterate)
```

Write a function `copyValue` that takes as input a table, a reference key and a new key and that extends the table with the new key mapping it to the same value as the reference key. Assume that the reference key is already in the table.

Answer (4 points):

```
{- copyValue tbl refKey newKey
  PRE: exists tbl refKey == True
  RETURNS: Extends the table tbl such that newKey maps to the same value as refKey.
  EXAMPLES: copyValue (insert empty "Apple" 3) "Apple" "Orange" ==
              insert (insert empty "Apple" 3) "Orange" 3
-}
copyValue :: Table String Integer -> String -> String -> Table String Integer
copyValue tbl refKey newKey =
```

(Hint: Recall the definition of the *Maybe* type:
`data Maybe a = Just a | Nothing deriving (Eq, Ord)`)

Question 13: Write a function `main` that asks a user ("Guess a word:") for a word on the Terminal. If the word typed by the user matches "foo", the function prints "Correct!" and exists, and otherwise asks for a new word. This is an example run of `main`:

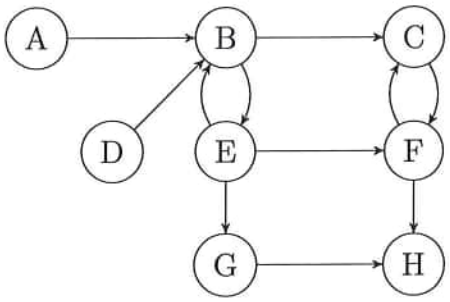
```
*Main> main
Guess a word:
bar
Guess a word:
foo
Correct!
*Main>
```

The functions `putStrLn` and `getLine` may be useful.

Answer (4 points):

```
main =
```

Question 14: What is the adjacency matrix representation of the following graph?



Answer (2 points):

Question 15:

Perform a DFS in the graph of Question 14, with (re)starts in the order CBADGEHF.

Answer (2 points):

Finish order =

DFS trees =

Question 16: We can implement DFS without recursion, by storing gray nodes in a particular data structure. Which one, and what happens if we change it?

Answer (2 points):

Which data structure is suitable for DFS (stack or queue)?

What happens if you swap it for the other one (queue or stack, respectively)?