+0/1/60+

Final Exam (Part 2) in Program Design and Data Structures (1DL201)

Teachers: Johannes Borgström and Dave Clarke

2018-03-15 / 14:00–19:00

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.** Cheating will not be tolerated.
- This is a multiple-choice exam. There are **twenty** questions. Each question has exactly **one** correct answer.
- Read the instructions on the answer sheet before you start.
- You may keep these question sheets. **Only hand in the answer sheet.**
- Johannes or Dave will come to the exam hall around 15:15 to answer questions.

Good luck!

Questions

Please choose a single answer for each question. Read the questions carefully, and watch out for negations (**not**, **except**, etc.).

√ **Question 1:** The (worst-case) complexity of insertion in a red-black tree with n nodes is

[A] $O(n)$ [B] $O(\log n)$ [C] $O(2^n)$ [D] $O(n^2)$ [E] $O(1)$

~~**Question 2:**~~ Consider the following parts of a data structure invariant for a node in a binary tree:

1. all elements in the left and right subtrees are smaller than the element in the current node.
2. the biggest element in the left subtree is less than the smallest element in the right subtree.
3. the smallest element in the left subtree is less than the smallest element in the right subtree.

Which combination of these corresponds to the invariant for a binary ~~min~~ max-heap?

[A] 1 and 2 hold at the root.
[B] 1 and 3 hold at every node.
[C] 1 and 2 hold at every node.
[D] 1 and 3 hold at the root.
[E] 1 holds at every node.

√ **Question 3:** How many elements are there in a binomial tree of rank 5?

[A] 25 [C] 32 [E] None of A-D.
[B] 31 [D] 5!

Abstract Data Type

√ **Question 4:** Which data structure would you use to implement an ADT where items can be removed and inserted, and it is always the most recently inserted item that is removed? a stack

[A] A binary search tree [C] A red-black tree [E] None of A-D.
[B] A list [D] A binomial heap

information hiding

Question 5: Modules are commonly used to encapsulate implementations of abstract datatypes (ADTs) in Haskell. Which of the following is **not** a property of modules that is useful for this purpose?

A They can inherit from other modules, making extensions of ADTs easy.

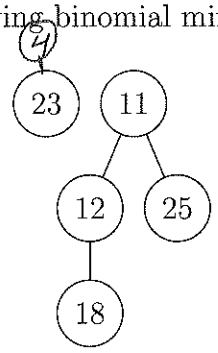
B They can hide the concrete types that the implementation uses.

C They can hide helper functions that are not part of the ADT interface.

D They avoid name clashes by means of separate namespaces.

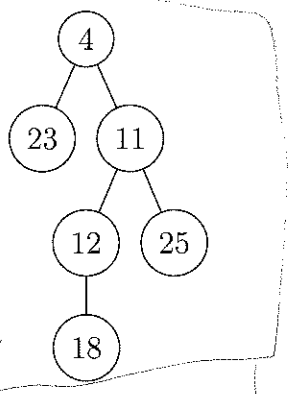
E They allow the implementation of the ADT to change independently of the client, e.g., to use a different data structure.

Question 6: Consider the following binomial min-heap.

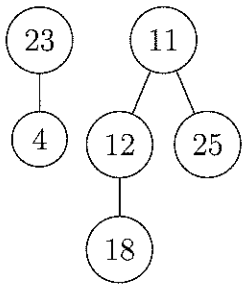


After inserting the element 4, what is a possible resulting binomial heap?

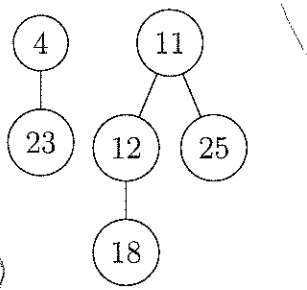
A



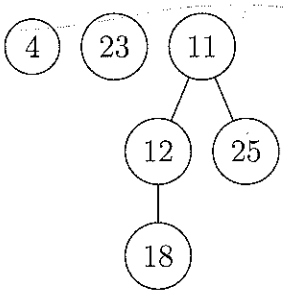
B



C



D



E None of A-D.

+0/4/57+

Kan heta vad som helst!

Question 7: Consider the following declaration. Which statement below is **false**?

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

Eq

True [A] `iSort` is a polymorphic function.

True [B] The implementation of `iSort` can use overloaded definitions of the functions `==` and `<`.

True [C] The type `a` must belong to the type class `Ord`.

? [D] The first argument to `iSort` describes how to compare values of type `a`.

[E] The call `iSort []` can only return the empty list `[]`.

Question 8: Assume that a queue contains the entries

(front) | 1 | 2 | 3 | 4 | 5 | 6 | (back). 3 2 1

What are the contents of the queue if we perform the following operations?

dequeue, dequeue, enqueue 3, dequeue, enqueue 2, enqueue 1,

[A] | 1 | 2 | 3 | 4 | 5 | 6 |

[B] | 1 | 2 | 3 | 1 | 2 | 3 |

[C] | 1 | 2 | 3 | 4 | 2 | 1 |

[D] | 6 | 3 | 2 | 1 |

[E] | 4 | 5 | 6 | 3 | 2 | 1 |

Question 9: Consider the following code which should select the maximum of three values:

```
max3 :: Int -> Int -> Int -> Int
max3 a b c = if a > b then a
    3 2 7    else if b > c then b
             else c
```

→ 3

--7 is the correct answer

Unfortunately, the code is buggy. Which test case reveals the bug?

[A] `TestCase (assertEqual "max3 3 1 2" 3 (max3 3 1 2))`

[B] `TestCase (assertEqual "max3 3 2 1" 3 (max3 3 2 1))`

[C] `TestCase (assertEqual "max3 2 1 3" 3 (max3 2 1 3))`

[D] `TestCase (assertEqual "max3 1 2 3" 2 (max3 1 2 3))`

[E] `TestCase (assertEqual "max3 1 2 2" 2 (max3 1 2 2))`

Question 10: Assume that you have a hash table with 4 slots which uses chaining to resolve collisions. Which of the hashing functions below can produce the following table?

0	1	2	3
"TJARK"	"JOHANNES"	$\perp$	"DAVID"

The vowels are A, E, I, O and U.

- [A] $h(s) = v \mod 4$, where v is the number of vowels in s
- [B] $h(s) = c \mod 4$, where c is the number of consonants in s OK!
- [C] $h(s) = n \mod 4$, where n is the number of letters listed after F in the alphabet Not Johannes Nor David
- [D] $h(s) =$ `length` $s \mod 4$ Nope!
- [E] $h(s) = 0$ → Cannot explain the unused slot 2.

Question 11: Which of the following is the correct way to add the results of two computations, `a` and `b`, of type `IO Int`?

- [A] `do {a + b}`
- [B] `unsafePerformIO a + unsafePerformIO b`
- [C] `unsafePerformIO (a + b)`
- [D] `return $ unsafePerformIO a + unsafePerformIO b`
- [E] `do {x <- a; y <- b; return $ a + b}`

Question 12:
Consider the following hash table of 11 cells, where $\perp$ denotes that a cell was never used.

i=1 i=2 i=3 i=4 i=0

0	1	2	3	4	5	6	7	8	9	10
33	23	57	34	70	$\perp$	6	$\perp$	19	$\perp$	54

Assume that the hash function is $h(k) = k \mod 11$, that open addressing with linear probing function $f(i) = i$ is used as the conflict resolution method, and that duplicates are allowed.
Firstly, 34 is deleted from the hash table. In which cell of the resulting table will 76 be placed? $= 6 \cdot 11 + 10$

[A] Nowhere [B] 3 [C] 5 [D] 9 [E] 10

Question 13: Consider the following code:

```
produce :: String -> IO ()
produce z = do
  putStr "A"
  x <- putStr "B"
  let y = putStr z
  return x
  putStr "D"
  y
  putStrLn ""
  return ()
```

ABD z
↑
C

What is the output (not the result) when evaluating `produce "C"`?

[A] AD
[C] ABDC
[E] ACBD
[B] ABCD
[D] ABCBDC

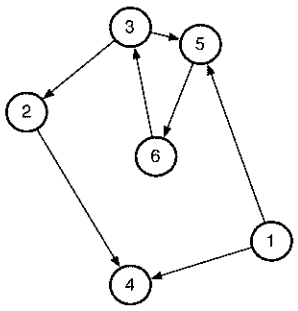
Question 14: What is the purpose of *cheating* in the design methodology?

[A] To reduce system complexity to implementable chunks.
[B] To develop a flexible set of building blocks.
[C] To reuse or steal code wherever possible.
[D] To gain insight into how to solve problems.
[E] All of these.

Question 15: What is the purpose of *dodging* in the design methodology?

[A] To solve a simpler problem in order to gain insight into the problem you are solving.
[B] To get code working quickly and make progress with some other part of a system.
[C] To manage complexity.
[D] To provide a general problem solving strategy.
[E] All of the above.

Question 16: Consider the following graph:



Which of the following is a valid adjacency matrix representation of the graph?

A

	1	2	3	4	5	6
1				1	1	
2			1	1		
3		1			1	1
4	1	1				
5	1		1			1
6			1		1	

B

	1	2	3	4	5	6
1				1	1	
2				1		
3		1			1	
4						
5						1
6			1			

C

1	4, 5
2	3, 4
3	2, 5, 6
4	1, 2
5	1, 3, 6
6	3, 5

adjacency list

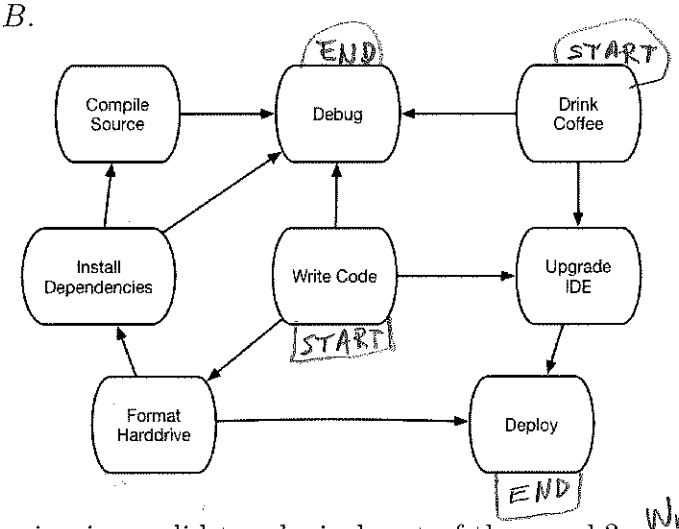
D (1,4), (1,5), (2,4), (3,2), (3,5), (5,6), (6,3) edge list

E

1	4, 5
2	4
3	2, 5
4	
5	6
6	3

adjacency list

Question 17: Consider the following graph of dependencies between tasks. Each box is a task, and an arrow from task A to task B means that task A should be done before task B.



Which of the following is a valid topological sort of the graph?

A Drink Coffee – Format Harddrive – Update IDE – Install Dependencies – Write Code – Compile Source – Debug – Deploy

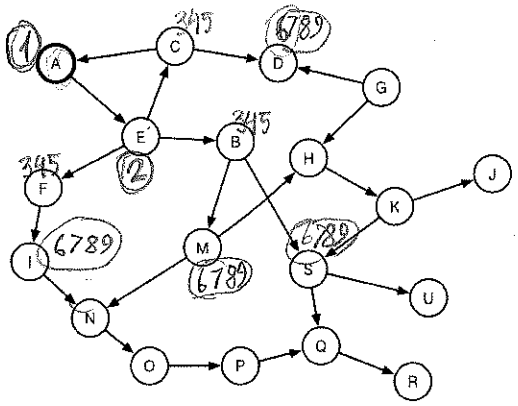
B Format Harddrive – Write Code – Install Dependencies – Drink Covfefe – Deploy – Debug – Compile Source – Upgrade IDE

C Write Code – Format Harddrive – Drink Coffee – Deploy – Update IDE – Install Dependencies – Compile Source – Debug

D Drink Coffee — Write Code — Update IDE – Format Harddrive – Deploy – Install Dependencies – Compile Source – Debug

E Write Code – Format Harddrive – Install Dependencies – Compile Source – Debug – Drink Coffee – Update IDE – Deploy

Question 18: Consider the following graph.



Which of the following nodes **cannot** be finished (painted black) **seventh** when doing a breadth-first search starting from node A?

A D B I C M D N E S

Question 19: Which of the following steps combine to form an algorithm for computing the strongly-connected components of a graph, G?

A Enumerate the nodes of G in DFS visiting order, starting from any node.

B Enumerate the nodes of G in DFS finish order, starting from any node.

C Compute the transpose G^T (that is, reverse all edges)

D Make a DFS in G, considering nodes in finish order from original DFS

E Make a DFS in G, considering nodes in visiting order from original DFS.

F Make a DFS in G, considering nodes in reverse visiting order from original DFS.

G Make a DFS in G^T, considering nodes in reverse finish order from original DFS

H Make a DFS in G^T, considering nodes in visiting order from original DFS.

I Make a DFS in G^T, considering nodes in finish order from original DFS.

J Strongly-connected components are the trees in the resulting depth-first forest.

- [ ] **A–C–H–J**
- [ ] **A–E–J**
- [ ] **A–F–J**
- [x] **B–C–G–J**
- [ ] **B–C–I–J**

√**Question 20:** Consider the following code that uses the exception model covered in class (using Monads). Here `return :: a -> Exceptional a` and `throw :: Exception -> Exceptional a` are used to report good and bad values, respectively.

```
data Exception = DivideByZeroException | BadWordException
               | ConquerByZorroException deriving Show

type Exceptional a = Either Exception a

(///) :: Int -> Int -> Exceptional Int
_ /// 0 = throw DivideByZeroException
a /// b = return $ a `div` b

cleanSentence :: String -> Bool
cleanSentence s = and $ (map clean) (words s)
  where clean s = not (s `elem` ["flip", "jeez", "crumbs"])

censor :: String -> Exceptional String
censor s = if cleanSentence s then return s else throw BadWordException

duplicate :: Int -> String -> Exceptional String
duplicate n s | n < 0 = throw ConquerByZorroException
duplicate n s | otherwise = return $ concat $ replicate n s

prog :: Int -> Int -> String -> Exceptional String
prog a b s = do
  e <- a /// b
  f <- censor s
  duplicate e f
```

What is the result of running `prog 1 0 "flip"`? Left DivideByZeroException

[A] `Right ""`

[B] `Left DivideByZeroException`

[C] `Left BadWordException`

[D] `*** Exception: DivideByZeroException`

[E] `*** Exception: BadWordException`