Your exam code: | | | | | | |

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

Teachers: Dave Clarke, Tjark Weber

Bergsbrunnagatan 15, room 1
2015-03-19 / 8:00–13:00

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.
- Read and follow the instructions on the front sheet.
- In the table below, clearly mark **at most one** answer for each question. (If you think that a question is ambiguous or has no correct answer, mark the question number with a $\star$ and explain on a separate sheet of paper what the problem is and what assumptions you have made to answer the question.)
- Dave Clarke will come to the exam hall around 9:30 to answer questions.

Good luck!

Your Answers

Question	Answer					Question	Answer				
1	A	B	C	D	E	11	A	B	C	D	E
2	A	B	C	D	E	12	A	B	C	D	E
3	A	B	C	D	E	13	A	B	C	D	E
4	A	B	C	D	E	14	A	B	C	D	E
5	A	B	C	D	E	15	A	B	C	D	E
6	A	B	C	D	E	16	A	B	C	D	E
7	A	B	C	D	E	17	A	B	C	D	E
8	A	B	C	D	E	18	A	B	C	D	E
9	A	B	C	D	E	19	A	B	C	D	E
10	A	B	C	D	E	20	A	B	C	D	E

Questions

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

1. Consider the following function.

```
lookup :: Eq a => a -> AssocList a b -> Maybe b
lookup _ [] = Nothing
lookup k ((x,y):xs) = if x==k then Just y else lookup k xs
```

   Which of the following type declarations will make the above code type-correct?

   (A) `type AssocList a b = a -> Maybe b`
   (B) `type AssocList = (x,y):xs`
   (C) `type AssocList a b = [(a,Maybe b)]`
   (D) `type AssocList a b = [(a,b)]`
   (E) `type AssocList a b = Eq a => Maybe b`

   Answer: (D)

2. Suppose you want to define a datatype to model *snarps*. You have no idea what a snarp is, but you have been told that there are only three kinds of them:

   (i) the *grob* is a special snarp (that is different from all other snarps),
   (ii) every string is a different snarp, and
   (iii) one can *gister* any snarp and any integer to obtain a new snarp.

   Which datatype definition would you use?

   (A) `data Snarp = Snarp Grob String (Snarp,Integer)`
   (B) `data Snarp = Grob | Str String | Gister Snarp Integer`
   (C) `data Snarp = Grob | String | Gister`
   (D) `data Snarp = Snarp Grob | Snarp String | Snarp Snarp Integer`
   (E) `data Snarp = Grob Snarp | Str String | Gister Snarp Integer`

   Answer: (B)

3. Consider the type of general trees, defined as

```
data Tree a = Node a [Tree a]
```

   What does the following function `f` compute?

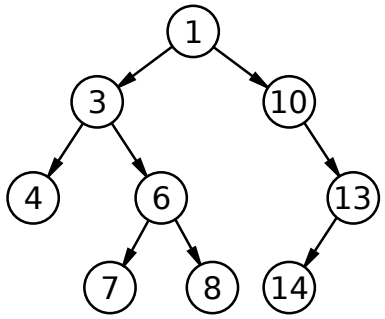
```
f (Node a ts) = a : fs ts

fs [] = []
fs (t:ts) = f t ++ fs ts
```

   (A) The number of nodes in a tree.
   (B) A post-order list of all values in a tree.
   (C) The height of a tree.
   (D) A pre-order list of all values in a tree.
   (E) The sum of all values in a tree.

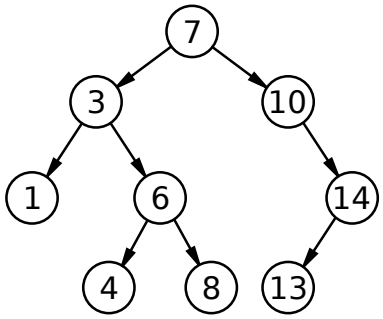
   Answer: (D)

4. Which of the following trees is a binary search tree?

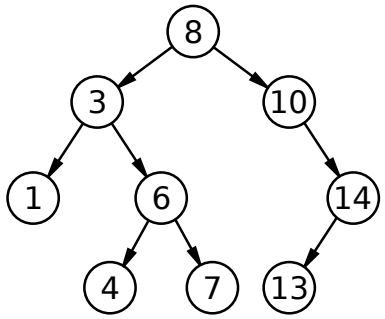
(A)



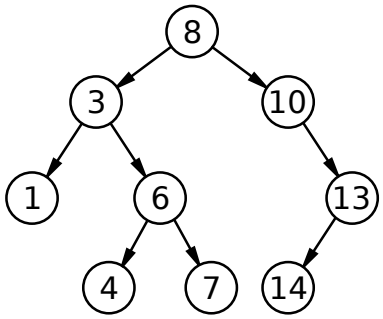
(B)



(C)



(D)



(E) None of these.

Answer: (C)

5. The (worst-case) complexity of searching in a binary search tree with n nodes is

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

Answer: (C)

6. Which of the following statements about Abstract Datatypes (ADTs) is **false**?

(A) ADTs make it easier to preserve representation invariants.
(B) Pattern matching is not available on ADTs.
(C) The client of an ADT is independent of the ADT's specific representation.
(D) ADTs support programming to an interface.
(E) ADTs are more efficient than regular datatypes.

Answer: (E) ADTs often introduce overhead.

7. Assume that the the module `Table` includes the following declaration:

```
module Table(Table, empty, insert, exists, lookup, delete,
  iterate, keys, values) where
```

Client code imports `Table` using the following import statement:

```
import qualified Table(empty,insert) as T hiding(exists)
```

Which additional names does this import statement introduce into the code?

(A) `empty, insert`
(B) `T.empty, T.insert`
(C) `Table, empty, insert, exists, lookup, delete, iterate, keys, values`
(D) `Table, empty, insert, lookup, delete, iterate, keys, values`
(E) `empty, insert, T.empty, T.insert`

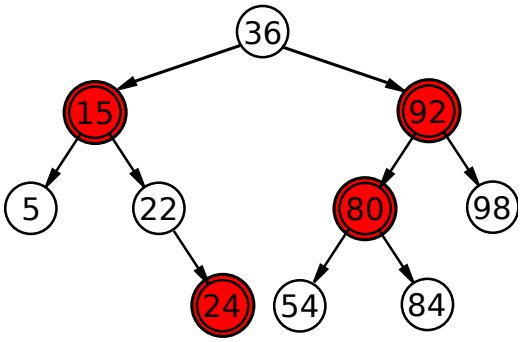
Answer: (B)

8. Suppose the following numbers are inserted, in the given order, into an initially empty red-black tree. (Grey=red.)

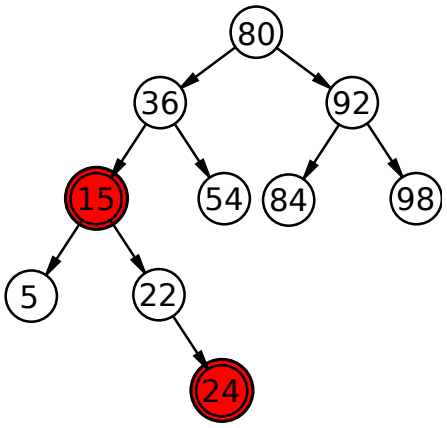
22 36 84 92 24 5 15 80 98 54

What is the resulting red-black tree?

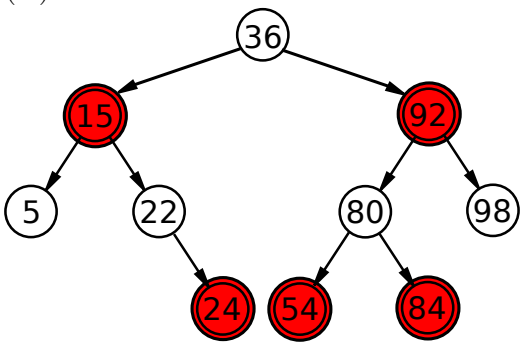
(A)



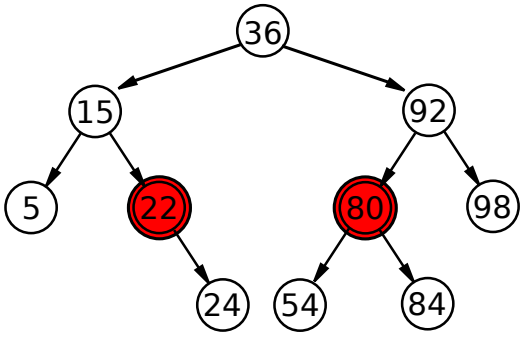
(B)



(C)



(D)



(E) None of these.

Answer: (B)

9. The (worst-case) complexity of inserting an item into a red-black tree with n nodes is

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

   Answer: (D)

10. The function `unsafePerformIO :: IO a -> a` from module `System.IO.Unsafe` is considered unsafe. It can be used to embed imperative code into pure code.

    Why is `unsafePerformIO` considered to be unsafe?

    (A) Using it properly requires advanced programming skills.
    (B) It invalidates Haskell's optimisations.
    (C) It violates purity.
    (D) It can result in I/O operations occurring in unexpected orders.
    (E) All of the above.

    Answer: (E) It's really bad.

11. Given the following datatype

```
data Tree a b = Leaf a | Branch b (Tree a b) (Tree a b)
```

    and the following typeclass

```
class Foo a where
  foo :: a -> Int
```

    with the following instances

```
instance Foo Bool where
  foo True = 1
  foo False = 0

instance Foo Int where
  foo n = n `mod` 5

instance (Foo a, Foo b) => Foo (Tree a b) where
  foo (Leaf a) = foo a
  foo (Branch a b c) = (foo a + foo b + foo c) `mod` 5

instance Foo a => Foo [a] where
  foo l = (sum (map foo l)) `mod` 5
```

    What is the value of

```
foo $ Branch True (Leaf [1::Int,2,3])
                  (Branch False (Leaf [0,3]) (Leaf [2,5]))
```

    (A) 0 (B) 1 (C) 2 (D) 3 (E) 4

    Answer: (C)

12. Consider the following function:

```
saywhat a b = if a < foo b then show a else show b
```

What type will Haskell give to this function, where `foo` is the function belonging to the type class in the previous question?

(A) `Int -> Int -> String`
(B) `(Foo a, Ord a) => Int -> a -> String`
(C) `(Show a) => Int -> a -> String`
(D) `(Foo a, Show a) => Int -> a -> String`
(E) `(Foo a, Ord a, Show a) => Int -> a -> String`

Answer: (D)

13. The purpose of monads in Haskell is:

(A) to make functional code look like imperative code.
(B) to enable I/O.
(C) to introduce the `do`-notation.
(D) to enable the mixing of pure and impure computation.
(E) to take something beautiful and make it ugly.

Answer: (D)

14. Consider the following interface to a stack datatype:

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

with the same semantics as considered in class. What is the value of

```
top $ pop $ push 15 $ push 12 $ push 10 empty
```

(A) A stack containing 15 and 12 (B) A stack containing 12 and 10
(C) 10 (D) 15 (E) 12

Answer: (E)

15. The following code is broken:

```
0.  broken = do
1.    putStrLn $ "Enter "
2.      ++ "your "++ "name: "
3.    let x = getLine
4.    y <- putStrLn "Thank you"
5.    x
```

Running the code in GHCi results in the following:

```
*Main> broken
Enter your name:
Thank you
Dave
"Dave"
```

The first `Dave` is the input entered by the user and the second `"Dave"` is the result of the function.

Why does `Thank you` appear before the user inputs his name!?!

Which line is the **cause** of the bug?

(A) 1 (B) 2 (C) 3 (D) 4 (E) 5

Answer: (C)

16. Consider the following hash table of 11 cells, where $\perp$ denotes that a cell was never used and Δ denotes that the element in a cell has been deleted.

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

Assume that the hash function is $h(k) = k \bmod 11$, that open addressing with linear probing function $f(i) = i$ is used as the conflict resolution method, and that duplicates are allowed.

In which cell will key 109 be placed?

(A) 9 (B) 5 (C) 3 (D) 10 (E) Nowhere.

Answer: (C)

17. Recall that the function `random` has type `random :: (Random a, RandomGen g) => g -> (a, g)`. You write the following code to generate three random coin tosses (where head = `True` and tails = `False`).

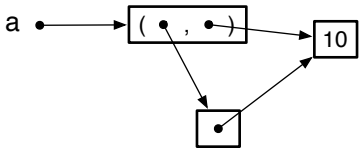
```
import System.Random

threeCoins :: (Bool, Bool, Bool)
threeCoins =
  let gen = (mkStdGen 21)
  in (fst (random gen), fst (random gen), fst (random gen))
```

Which of the following sentences **best** describes `threeCoins`?

(A) Oh dear!
(B) `threeCoins` gives the same value each time it is accessed, and the three coins are equal to each other.
(C) `threeCoins` gives the same value each time it is accessed, but the three coins may be different from each other.
(D) `threeCoins` generates different values each time it is accessed, but the three coins are always equal to each other.
(E) `threeCoins` generates different values each time it is accessed, and the three coins may be different from each other.

Answer: (B)

18. The following depicts the under-the-hood structure of a mystery Haskell expression stored in variable `a`. The arrows represent references.

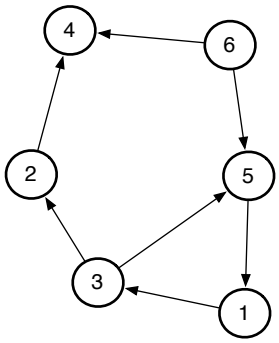


Which of the following is a possible type of `a`?

(A) `(Int, Int)` (B) `(IORef Int, Int)` (C) `IORef (IORef Int, Int)`
(D) `IORef (IORef (IORef Int), IORef Int)` (E) `IORef (IORef Int, IORef Int)`

Answer: (D)

19. Consider the following directed graph:



Which of the following is a valid depth-first search ordering—that is, the order in which nodes are visited when performing a depth-first search?

(A) 123456 (B) 324516 (C) 135264 (D) 654321 (E) 651342

Answer: (B)

20. Which of the following corresponds to the adjacency list representation of the graph from the previous question?

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

(C)

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

(D)

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

(E) (1,3), (3,2), (3,5), (2,4), (6,5), (6,4), (5,1)

Answer: (C)