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

Teachers: Johannes Borgström, Dave Clarke

2018-01-11 / 08:00–13:00

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.
- This is a multiple-choice exam. Each question has exactly **one** correct answer.
- You may keep these question sheets. **Only hand in the answer sheet.** Also read the instructions on the answer sheet before you start.
- Johannes or Dave will come to the exam hall around 10:00 to answer questions.

Good luck!

Common Material

Some of the exam questions refer to the following function:

```
{- func a ls
   PRE: ?PRE?
   RETURNS: ?RETURNS?
 -}
func :: ?TYPE?
func = funci []

-- funci a b ls
-- VARIANT: ?VARIANT?
funci r _ [] = r
funci r a ((x,y):ls)
  | x == a    = funci (y:r) a ls
  | otherwise = funci r a ls
```

Questions

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

Question 1: What is the value of `func 3 [(1,'a'),(2,'b'),(3,'c')]` ?

- A `['a','b','c']`
- B `"c"`
- C `["c"]`
- D `'c'`
- E `"abc"`

Question 2: What is a type (`?TYPE?`) of `func`?

- A `Int -> [(Int,b)] -> [b]`
- B `Int -> [(a,b)] -> [b]`
- C `Int -> ([a],[b]) -> [b]`
- D `Int -> [(Int,b)] -> [Int]`
- E `Int -> [(a,b)] -> [a]`

Question 3: What is the most appropriate precondition (`?PRE?`) for `func a ls`?

- A `ls` is a list
- B `ls` contains at most one element matching `(a,x)`
- C `ls` contains no duplicate elements
- D `ls` contains a pair `(a,c)`
- E `True`

Question 4: What is the most appropriate description of the return value (`?RETURNS?`) for `func a ls`?

- A a list of all pairs `(x,y)` in `ls` where `x=a`
- B `[b]`
- C a list of all `y` such that `(a,y)` in `ls`
- D `y:r`
- E a list of all `y` such that `(y,a)` in `ls`

Question 5: Which of the following is a variant (`?VARIANT?`) for the function `funci r a ls`?

- [A] `length ls + length r`
- [B] `length ls - 1`
- [C] `length r`
- [D] `2 * length ls`
- [E] `length ls - length r`

Question 6: Which of the following modifications to the definition of `func` (or `funci`) will reverse the order of the list returned from `func`?

- [A] replace `funci []` by `reverse.(funci [])`.
- [B] replace `funci r _ [] = r` by `funci r _ [] = reverse r`.
- [C] replace `((x,y):ls)` by `(ls++[(x,y)])`.
- [D] replace `y:r` by `r ++ y`.
- [E] replace `x==a` by `a==y`.

Question 7: Which of the following expressions **does not** evaluate to 15?

- [A] `36 'mod' 3 * 7`
- [B] `9 / 3 * 5`
- [C] `7 - 3 + 11`
- [D] `[3,5] !!  1 * 3`
- [E] `11 'div' 3 * 5`

Question 8: Consider the expression

```
let f x y = x + 3 >= y + 3.1 in f 1 1
```

Evaluating this expression will result in ...

- [A] `True`
- [B] a syntax error.
- [C] a type error.
- [D] a run-time error.
- [E] none of these.

Question 9: Consider the declaration

```
f x = let f x = x+1 in f (f x)
```

Which of the following is equivalent to the declaration above?

- [A] `f z = let g y = z+1 in g (g z)`
- [B] `f x = let f y = y+1 in f (f y)`
- [C] `f x = let g z = z+1 in g (g x)`
- [D] `f y = let g y = y+1 in f (f y)`
- [E] None of these.

Question 10: Consider the function

```
zip (x:xs) (y:ys) = (x,y) : zip xs ys
zip _ _ = []
```

Which of the following expressions is **not** a variant for `zip xs ys`?

- A `length ys`
- B `length xs`
- C `length xs * length ys`
- D `length xs + length ys`
- E `abs (length xs - length ys)`

Question 11: Which of the following evaluates to `11`?

- A `foldr (*) 0 [1,7,3]`
- B `foldr (+) 0 [1,7,3]`
- C `foldr (*) 1 [1,7,3]`
- D `foldr (+) 1 [1,5,3]`
- E `foldr (:)  [] [1,5,3]`

Question 12: What is the type of `bar`, which is defined as follows:

```
bar f g a = f (g a) a
```

- A `(a -> b -> c) -> (a -> b) -> a -> c`
- B `(a -> b -> c) -> (a -> b) -> a -> c`
- C `((a -> b -> c), (b -> a), b) -> c`
- D `(a -> b -> c) -> b -> (a -> b) -> c`
- E `(a -> b -> c) -> (b -> a) -> a -> c`

Question 13: Which of the following functions is different from the others — that is, which function gives different results when applied to the same arguments?

- A
```
f [] = []
f (x:xs) | x > 2 = x + 10 : f xs
         | otherwise = f xs
```
- B `f = map (+10) . filter (>2)`
- C `f xs = [ x + 10 | x <- xs, x > 2]`
- D `f = map (\x -> x + 10) . filter (\x -> 2 > x)`
- E `f = filter (>12) . map (+10)`

Question 14: Type τ is an *instance* of type ρ if τ can be obtained from ρ by instantiating ρ's type variables with other types (which may also be variables). Two types τ and ρ are *related by instantiation* if τ is an instance of ρ or ρ is an instance of τ.
Which of the following types is **not** related to any of the others by instantiation?

- A `a -> a`
- B `(a, b) -> (a, b)`
- C `(Int, a) -> (Int, a)`
- D `(a, a) -> (a, b)`
- E `(a -> a) -> (a -> a)`

Question 15: Recall that $\Theta(n^3)$ is the set of functions bounded both above and below by n^3, modulo a constant factor, etc. Which of the following is equal to $\Theta(n^3)$?

- A $O(n^3) \setminus O(n^2)$
- B $O(n^3) \cap \Omega(n^3)$
- C $\Omega(n^2) \setminus \Omega(n^3)$
- D $(O(n^3) \setminus O(n^2)) \cap (\Omega(n^2) \setminus \Omega(n^3))$
- E All of the above

Question 16: Consider the following function

```
foo :: ([a] -> b) -> [a] -> [b]
foo f [] = [f []]
foo f l@(_:as) = f l : foo f as
```

Assuming that the run-time cost of some function g is given by $T_g(n)$, where n is the *size* of its input, which of the following recurrences describes the run-time cost of `foo g`?

A $T(n) = \begin{cases} T_g(0) & \text{if } n = 0 \\ \Theta(1) + T(n-1) + \Theta(n) & \text{if } n \geq 1 \end{cases}$

B $T(n) = \begin{cases} T_g(0) & \text{if } n = 0 \\ T_g(n) + T(n-1) + \Theta(n) & \text{if } n > 1 \end{cases}$

C $T(n) = \begin{cases} T_g(0) & \text{if } n = 0 \\ T(n) + T(n-1) + \Theta(1) & \text{if } n \geq 1 \end{cases}$

D $T(n) = \begin{cases} T_g(0) & \text{if } n = 1 \\ T_g(n) + T(n-1) + \Theta(n) & \text{if } n > 1 \end{cases}$

E $T(n) = \begin{cases} T_g(0) & \text{if } n = 0 \\ T_g(n) + T(n-1) + \Theta(1) & \text{if } n \geq 1 \end{cases}$

Question 17: Which of the following recurrences has the closed form $T(n) = 5n+7$?

A $T(n) = \begin{cases} 7 & \text{if } n = 0 \\ T(n-1) + 5 & \text{if } n \geq 1 \end{cases}$

B $T(n) = \begin{cases} 5 & \text{if } n = 0 \\ 2T(n-1) + 7 & \text{if } n \geq 1 \end{cases}$

C $T(n) = \begin{cases} 5 & \text{if } n = 0 \\ 7T(n) + 1 & \text{if } n \geq 1 \end{cases}$

D $T(n) = \begin{cases} 7 & \text{if } n = 0 \\ 5T(n-1) & \text{if } n \geq 1 \end{cases}$

E $T(n) = \begin{cases} 3 & \text{if } n = 0 \\ 5T(n-1) + 4 & \text{if } n \geq 1 \end{cases}$

Question 18: Consider the following datatype declarations.

```
data BTreeA a = Leaf a
              | Node (BTreeA a) a (BTreeA a)
data BTreeB a = Leaf a
              | Node (BTreeB a) (BTreeB a)
data BTreeC a = Leaf
              | Node (BTreeC a) a (BTreeC a)
data BTreeD a = Leaf a
              | Node a (BTreeD a) (BTreeD a)
data BTreeE a = Leaf a
              | Node (BTreeE a) (BTreeE a) a
```

Which of them can represent trees containing an arbitrary non-negative number of data items?

☐ A `BTreeA` ☐ B `BTreeB` ☐ C `BTreeC` ☐ D `BTreeD` ☐ E `BTreeE`

Question 19: Which of the binary tree datatypes defined above admits insertion of a single new data item in constant time (i.e., $O(1)$)?
With insertion is meant an operation `insert :: a -> BTreeX a -> BTreeX a`, where `insert x t` returns a tree containing `x` and the data items in `t` (similar to `(:) :: a -> [a] -> [a]`).

☐ A `BTreeB` and `BTreeC`
☐ B `BTreeA`, `BTreeD`, and `BTreeE`
☐ C `BTreeC` only
☐ D `BTreeB` only
☐ E All except `BTreeB`

Question 20:
Which of the datatype declarations above **cannot** be used with the standard binary search tree invariant and search algorithm as seen in class?

☐ A `BTreeA` ☐ B `BTreeB` ☐ C `BTreeC` ☐ D `BTreeD` ☐ E `BTreeE`