b1. Value of following expression:
6 + 12 ‘div‘ 5 - 4
a. 4
b. -1
c. 0
d. 1
e. 18

2. Type of following expression:
chr (head $ tail $ map (+1) $ filter even [1..10])
a. [Int]
b. Int
c. [Char]
d. Num a => [a]
e. Char

3. Which parenthesised expression 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 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 doesn’t 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 has 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 the pattern matches 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 the pattern matches 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 one results in a run time 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 a proper instance of Num, what needs to be added to the data type?

```
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