CS2104 Programming Language Concepts

Mock Test

You have 60 minutes to complete the quiz. Circle your answer.
After finishing, do an honest self-grading together with the tutor.

Write your matrix card number in the box below.

This paper comprises 24 multiple choice questions with upto 5 possible answers each, and 5 questions with written answers expected. The last option, None of the Above, of MCQ questions should be chosen if the answer cannot be found in the first four choices. Each question carries one mark, and there is no penalty for incorrect answers.

Consider the following code fragment in Haskell.

```
let x =
  let x=5 in x*2
in x+x
```

Question 1: What result will be returned by the above code fragment?

1 [A] 10

1 [B] 20

1 [C] 30

1 [D] 40

1 [E] None of the Above

Consider the following code fragment in Haskell.

```
let x = 5 in
let f y = x+y in
  (f (f (x+1)))
```

Question 2: What result will be returned by the above code fragment?

2 [A] 5

2 [B] 10

2 [C] 15

2 [D] 20

2 [E] None of the Above

Question 3: How many additions will be performed by the same code fragment?

3 [A] 3

3 [B] 4

3 [C] 5

3 [D] 6

3 [E] None of the Above

Consider the following recursive function in Haskell.

```
foo x y =
  if x==0 then y
  else 1+(foo (x-1) (y*2))
```

Question 4: What result will be returned by the call `(foo 4 1)`?

4 [A] 5

4 [B] 10

4 [C] 15

4 [D] 20

4 [E] None of the Above

Question 5: What result will be returned by the following call `(foo 5 0)`?

5 [A] 5

5 [B] 10

5 [C] 15

5 [D] 20

5 [E] None of the Above

Question 6: Which of the following is incorrect about the `foo` method?

6 [A] This method has an accumulating parameter.

6 [B] This method terminates for `n>=0`.

6 [C] This method uses tail-recursion.

6 [D] This method may go into a non-terminating loop.

6 [E] None of the Above

Consider the following recursive function in Haskell.

```
goo x =
  if x<=1 then 1
  else goo (x-1) + (goo (x-3))
```

Question 7: What result will be returned by the following call `(goo 8)`?

7 [A] 18

7 [B] 20

7 [C] 22

7 [D] 24

7 [E] None of the Above

Question 8: How many *distinct* `goo` calls are made by `(goo 8)`? (For the distinct calls, each set of calls with the same arguments are to be counted once.)

8 [A] 6

8 [B] 7

8 [C] 8

8 [D] 9

8 [E] None of the Above

Consider the following recursive function in Haskell.

```
hoo g x = g x (hoo g x)
```

Question 9: What is the most general type that could be inferred for the `hoo` function?

9 [A] `(a -> a) -> a -> a`

9 [B] `(a -> b) -> a -> b`

9 [C] `(a -> a -> a) -> a -> a`

9 [D] `(a -> b -> b) -> a -> b`

9 [E] None of the Above

Question 10: Which of the following statement is **not** true about the `hoo` function?

10 [A] This method may go into a non-termination loop.

10 [B] This method does not have a base-case.

10 [C] This method is a higher-order function.

10 [D] This method will cause stack overflow when it is supplied with its two parameters.

10 [E] None of the Above

Consider the following recursive function in Haskell.

```
koo g x = g x (koo g ((x::Int)-1))
```

Question 11: What is the most general type that could be inferred for the `koo` function?

11 [A] `(a -> a -> a) -> a -> a`

11 [B] `(a -> b -> b) -> a -> b`

11 [C] `(Int -> b -> b) -> Int -> b`

11 [D] `(Int -> Int -> Int) -> Int -> Int`

11 [E] None of the Above

Question 12: What result would be returned by the following code fragment which uses the earlier `koo` function?

```
let fact n = koo (\x y -> if x==0 then 1 else x*y) n
in fact 5
```

12 [A] 5

12 [B] 24

12 [C] 50

12 [D] 120

12 [E] None of the Above

Question 13: What is the best explanation for your answer to Question 13?

13 [A] The `koo` function has a parameter which decreases with each recursive call.

13 [B] The `fact` function has a factorial-like computation.

13 [C] The arguments of calls are evaluated based on strict evaluation.

13 [D] The arguments of calls are evaluated based on lazy evaluation.

13 [E] None of the Above

Consider the following function in Haskell.

```
eoo x = [error ((show ((x::Int)+1)))]
```

Question 14: What is the most general type that could be inferred for the `eoo` function?

14 [A] `(Int -> [Int])`

14 [B] `(Int -> [a])`

14 [C] `(a -> [Exception])`

14 [D] `(Int -> [Exception])`

14 [E] None of the Above

Question 15: Which of the following *best* describes the expected execution of the `eoo` method?

15 [A] The method returns a list of exceptions.

15 [B] The method returns a list of integers

15 [C] The method throws an exception that was subsequently caught within the same method.

15 [D] The method returns an empty list.

15 [E] None of the Above

Consider the following list-based function in Haskell.

```
loo xs w =
  case xs of
    [] -> (w,[])
    y:ys ->
     let (c,cs)=loo ys w in
     if y>0 then (1+c,y:cs)
     else (c,cs)
```

Question 16: What value will be returned by the call `loo [-1,2,3,0] 5`.

16 [A] `(7, [2,3])`

16 [B] `(7, [-1,2,3,0])`

16 [C] `(5, [2,3])`

16 [D] `(7, [3,2])`

16 [E] None of the Above

Question 17: Which of the following higher-order function could best be used to re-implement the `loo` method?

17 [A] `map`

17 [B] `filter`

17 [C] `foldl`

17 [D] `foldr`

17 [E] None of the Above

Consider the following higher-order function in Haskell.

```
gen_ho f n =
 if n==0 then []
 else (f n):(gen_ho f (n-1))
```

Question 18: What value will be returned by the call `gen_ho (\ x -> x-1) 5`.

18 [A] `[5,4,3,2,1]`

18 [B] `[1,2,3,4,5]`

18 [C] `[4,3,2,1,0]`

18 [D] `[4,3,2,1,0,-1]`

18 [E] None of the Above

Question 19: Which of the following is **not** true about the `gen_ho` method in general? You may assume the use of suitable type-safe parameters when considering the scenarios described below.

19 [A] This function can be made to return a decreasing list of integers.

19 [B] This function can be used to return a list of strings.

19 [C] This function can be used to return a list with identical elements.

19 [D] This function can be used to return an empty list.

19 [E] None of the Above

Consider a lazy higher-order method in Haskell:

```
koo g x = g x (koo g (x-1))
```

Question 20: What result would be returned by the expression `sum(take 4 (koo (\ x xs -> x:xs) 3))`?

20 [A] `6`

20 [B] `12`

20 [C] `18`

20 [D] `24`

20 [E] None of the Above

Question 21: What result would be returned by the code following fragment:

```
koo (\ x y -> if x=0 then 1 else x*y) 5
```

21 [A] 5

21 [B] 24

21 [C] 50

21 [D] 120

21 [E] None of the Above

Question 22: Which of the following is *not true* about the evaluation mechanism used in Haskell?

22 [A] Lazy evaluation is the default evaluation mechanism in Haskell.

22 [B] We can force some parameters of Haskell method to be evaluated strictly.

22 [C] Lazy evaluation allows infinite list to be supported.

22 [D] Lazy evaluation is always result in better performance than strict evaluation.

22 [E] None of the Above

Consider the following lazy function in Haskell to return a list of prime numbers.

```
primes = sieve [2..]
sieve (x:xs) = x:(sieve [y | y<-xs, (mod y x)/=0])
```

Question 23: Which of the following is *not true* about this method.

23 [A] The prime method returns an infinite list of primes

23 [B] The expression `take 100 primes` terminates.

23 [C] The expression `head [i | i<-primes, i>1000]` terminates.

23 [D] The expression `length [i | i<-primes, i>1000]` terminates.

23 [E] None of the Above

Question 24: Which of the following is *not true* about input-output mechanism used in Haskell?

24 [A] Like C, the input/output mechanism of Haskell is based on imperative side-effects.

24 [B] Haskell uses monads to capture the effects of IO operations.

24 [C] The monad IO mechanism and lazy evaluation are orthogonal mechanisms.

24 [D] An IO procedure with the type `Int -> IO String` is essentially a function which when applied to an integer may perform some IO operation, followed by the return of a value of `String` type.

24 [E] None of the Above

Question 25: Write down the full type of the bind operator `>>=` for the Monad class.
Answer:

Question 26: The operator `>>` has type `Monad m => m a -> m b -> m b`. Show how this is implemented in terms of `>>=`.
Answer:

Question 27: Consider the following method:

```
perform2 m =
  m >>= \ x1 ->
  m >>= \ x2 -> [x1,x2]
```

Write down the full type of this method.
Answer:

Question 28: Rewrite the `perform2` method with the help of do-notation.
Answer:

Question 29: What is the outcome of `perform2 [1,3]`?
Answer: