This entrance exam booklet is being re-used for mid-sem exams.
Please ignore the printed material on the sheets.

cmi | CHENNAI MATHEMATICAL INSTITUTE
https://www.cmi.ac.in

Mid Semester Examination, Aug-Nov 2023

Name: Anubra Maguirdar	Roll Number: MCS202304
Date:	Subject: Haskell
Course & Year: MSC 1 2023	Total No. of Pages:

Part B

No.	Marks	Remarks
11		
12		
13		
14		
15		
16		

Part B (ctd.)

No.	Marks	Remarks
17*		
18*		
19*		
20*		

Part A	35
Part B	10+2+20+20 = 52
Total	87.

Further remarks:

Please indicate on the front page the questions in Part B to be marked.

Solution to Question (II)

1) reverse [(-10) .. 10] ++ [(-10) .. 10])

= [10, 9 .. (-10)] ++ [(-10) (-9) .. 10]

= [10 9 .. -10 . -10 -9 .. 10]

⇓

∴ filla (>5) ( )

= [10 9 8 7 6 5 5 6 7 8 9 10] ✓

2) (reverse [0 .. 10] ++ [(-10) .. 10])

= [10, 9, 8 .. 0, . -10 . , -9 .. 10]

⇓

takewhile (>=-6) (. .)

= [10, 9, 8 .. . 0]

⇓

length ( )

= 11 ✓

Please indicate on the front page the questions in Part B to be marked.

Solution to Question (12)

3 foldr (++) [] [[0..1], [2..5], [6..13]]

= .... ([6..13] ++ [])

[6, 7, 8 ... 13] ++ []

: 6 : 7 : ... : 13 ((++) []) ← 1st time will be called

8 times 2nd time is called

Similarly, [2..5] ++ [6 ... 13]

will call 2nd time length [2..5] times

= 4

1st will call = 2 times as [length [0..1] = 2]

So, 8 + 4 + 2 = (14) times

Please indicate on the front-page the questions in Part B to be marked.

Solution to Question (13)

(4) foldl (++) [] [[0,1], [2..5], [6..13]]

= (([] ++ [0,1]) ++ [2..5]) ++ [6..13])

⇓

0 calls to 2nd Lines 0

Now [0,1] ++ [2..5]

2 calls to 2nd line 2

[0,1,2,3,4,5] ++ [6..13]

6 calls to 2nd line. + 6

= 8

So, total 8 calls

Please indicate on the front-page the questions in Part B to be marked.

Solution to Question (14)

(5). [(1,0), (2,0), (3,0), (4,0), (5,0), (6,0), (7,0), (8,0), (9,0),
(2,1), (3,1), (4,1)] ...

0 1 2 3 4 5 6 7 8 9 10 11

So answer should be (11) ✓

6. foldr (&&) [Val [Val] Val]

(Bool → Bool → Bool) Bool [Bool] Bool

∴ type for expression foldr (&&) will be

Bool → [Bool] → Bool. ✓

~~g f should have an input type~~

~~as (\x -> f (x++z)) takes two~~ lists

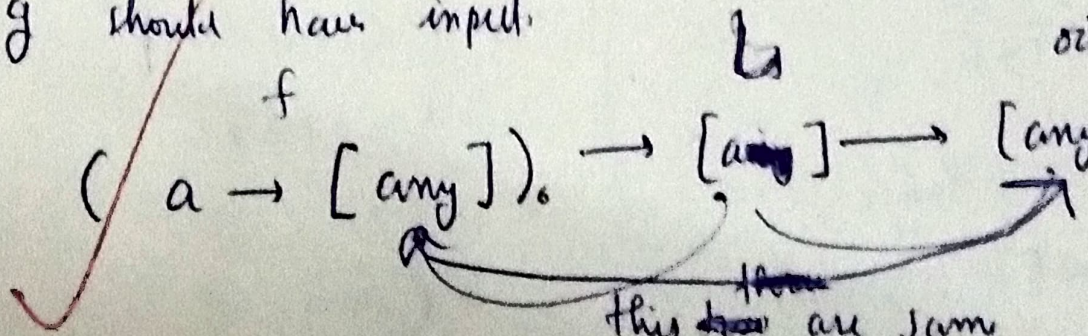
~~and~~

g.f should have an input type say a.

Such that $f :: a \rightarrow [any]$ throughout any will be constant once fixed

and so $g.f :: [a] \rightarrow [any]$

$\therefore$ g should have input



Please indicate on the front page the questions in Part B that you are attempting.

Part B

Solution to Question (12*)

(1) we will replace $\begin{pmatrix} fib(1) & \text{with} & 1 \\ fib(0) & , & 0 \end{pmatrix}$ whenever we see it as the fib base case.

$fib\,5 = fib(3) + fib(4)$

$= (fib(1) + fib(2)) + fib\,4$

$= (1 + fib(2)) + fib\,4$

$= (1 + (fib(0) + fib\,1)) + fib\,4$

$= (1 + 0 + fib(1)) + fib(4)$

$= (1 + 0 + 1) + fib(4)$

$= 2 + fib(4)$

$= 2 + (fib(2) + fib(3))$

$= 2 + ((fib(0) + fib(1)) + fib(3))$

$= 2 + ((0 + fib(1)) + fib(3))$

$= 2 + ((0+1) + fib(3))$

$= 2 + (1 + fib(3))$

$= 2 + (1 + (fib(1) + fib(2)))$

$= 2 + (1 + (1 + fib(2)))$

$= 2 + (1 + (1 + (fib(0) + fib(1))))$

$= 2 + (1 + (1 + (0 + fib(1))))$

$= 2 + (1 + (1 + (0 + 1)))$

$= 2 + (1 + (1 + 1))$

$= 2 + (1 + 2)$

$= 2 + 3$

$= 5$ (ans)

(10)

Head lookup and prepending takes constant

So, reversing is a good option.

3

Intuition : we will use a helper function that takes the reverse of the list and two empty list as its argument

helper : [Int] → [Int] → [[Int]] → [[Int]]

- [Int] ⇓ Input reversed list
- [Int] ⇓ buffer: if we find a lower number wrt its head, we put this into the next list of [Int] and make it null
- [[Int]] ⇓ stores the current answer
- [[Int]] ⇘ returns the current answer (will proceed like tail recursion)

Reversing the list will make the logic/algo easier as everything will go as expected [as head lookup takes constant time]

please indicate on the front page the questions in Part B to be marked.

Solution to Question (20*)

```
upRuns :: [Int] -> [[Int]]
upRuns x = helper (reverse x) [] []

helper : [Int] -> [Int] -> [[Int]] -> [[Int]]
helper [] a b = (a:b)
helper (x:xs) [] b = helper xs [x] b
helper (x:xs) (a:as) b
  | x > a = helper xs [x] ((a:as):b)
  | otherwise = helper xs (x:a:as) (b)
```

Briefly what helper function does on one of the inputs.

upRuns [3,2,1] = helper [1,2,3] [] []

helper [1,2,3] [] []
= helper [2,3] [1] []
= helper [3] [2] [[1]]
= helper [] [3] [[2],[1]]
= ~~Helper~~ [[3],[2],[1]]

(20)

Simulate: upRuns [1,2,3] = helper [3,2,1] [] [] = helper [2,1] [3] []
= helper [1] [2,3] []
= helper [] [1,2,3] []
= [[1,2,3]]

Rough work only. Do not write your final answer here.

```
getMeSum :: IO()
getMeSum = recurse 10 0

recurse :: Int -> Int -> IO()

recurse 0 sum = putStrLn (show sum)

recurse n sum = do
    cur <- getLine
    recurse (n-1) (sum + (read(cur)) :: Int)
```

20

Alternatively ~~we can use integral~~

Rough work only. Do not write your final answer here.

2. Let's say in 'l' we have 3 elements.

So, (foldl f id l)

⇓

= foldl f id a:b:c:[]

= ((id `f` a) `f` b) `f` c

= (((id `f` a) `f` b) `f` c)

2

returns l = (((id `f` a) `f` b) `f` c) []

---

output should be [c, b, a]

---

f(f(f(id,a),b),c) []

at the last depth it will call recursively (id [])

f (id []) a = a : []

f (f(id,a)) b = b : f(id,a)

= b : [a]

= [b,a]

∴ f (f(f(id,a),b),c) = [c,b,a]

So, $f : [a] \longrightarrow a \longrightarrow [a]$

$f\ [\,]\ x = [x]$

~~$f\ (x:xs)$~~

$f\ (y:ys)\ x = (x : y : ys)$

$f\ g\ x\ l =$

$x : (g\ l)$

is the correct answer.

rough work only. Do not write your final answer here.

Part (a) 7

(.) function composition

not is a function that has type

Bool → Bool

the resulting function should look like.

not(.) f_1(.) f_2(.) .. (f_n)

Say f_n has arity k_n

f_{n-1} " " k_{n-1}

f_n will take k_n inputs

and the return value of f_n will be used in f_{n-1}

So, it will have arity k_{n-1}

and we need to provide it with k_{n-1} values

∴ ~~the input be~~ and other one will be of the return type of f_n

the type expression will be

~~$a_{k1} \to a_{k2} \to \cdots \to a_{k-1}$~~

$[\, a_{n1} \to a_{n2} \to \cdots a_{nk_{n-1}} \to a_{(n-1)1} \to \cdots a_{(n-1)(k_n}$

14

$\to \cdots \to$ Bool