

X



(<https://swayam.gov.in>)



(https://swayam.gov.in/nc_details/NPTEL)

vivekdubey74rr@gmail.com ▾

NPTEL (<https://swayam.gov.in/explorer?ncCode=NPTEL>) » Introduction To Haskell Programming (course)



If already registered,
click to check your
payment status

Week 5: Programming Assignment

Due on 2023-08-31, 23:59 IST

Course outline

How does an NPTEL
online course work? ()

Week 1: Introduction ()

Week 2: Lists, Strings,



Tuples ()

Week 3: Rewriting, Polymorphism, Higher Order Functions on Lists ()

Week 4: Efficiency, Sorting, Infinite lists, Conditional polymorphism, Using ghci ()

Week 5: User-defined datatypes, abstract datatypes, modules ()

- ☐ User-defined datatypes (unit? unit=45&lesson=46)
- ☐ Abstract datatypes (unit? unit=45&lesson=47)
- ☐ Modules (unit? unit=45&lesson=48)
- ☐ Week 5 Feedback Form: Introduction To Haskell Programming (unit? unit=45&lesson=49)
- ☒ **Week 5: Programming Assignment (/noc23_cs94/progassignment?name=99)**



Week 6: recursive data types, search trees ()

Week 7: arrays, IO ()

DOWNLOAD VIDEOS ()

Text Transcripts ()

Books ()

**Problem Solving
Session - July 2023 ()**

1. Define a function `subSeq :: String -> String -> Bool` which checks whether the first argument is a subsequence of the second. A subsequence is obtained by deleting some letters in a string and retaining the other characters in the same order as in the original string.

Test cases:

```
subSeq "ab" "abc" = True
subSeq "ab" "acb" = True
subSeq "ab" "bca" = False
subSeq ""  "bea" = True
subSeq "ba" "ba" = True
```

2. Define a function `subWord :: String -> String -> Bool` which checks whether the first argument is a subword of the second. A subword is obtained by deleting some number (possibly 0) of letters at the left end and right end in a string and retaining the other characters in the same order.

Test cases:

```
subWord "ab" "abc" = True
subWord "ab" "acb" = False
subWord "ca" "bca" = True
subWord ""  "bea" = True
subWord "ba" "ba" = True
```

3. A two-dimensional matrix can be represented as a list of rows, each row itself being a list of elements. So in general it is of type `[[a]]`. Not every list of lists is a matrix, though. For instance, `[[1,2,3], [], [2,4]]` is a list of three lists, each of a different size.

(a) Define a function `isMatrix :: [[a]] -> Bool` that checks if a list of lists is a valid matrix (nonzero number of rows, each of the same nonzero length).

Test cases:

```
isMatrix [] = False
```



isMatrix $[[[],[],[]]] = \text{False}$
isMatrix $[[2,3], [4,5], [6,7]] = \text{True}$
isMatrix $[[2,3,4,5,6,7]] = \text{True}$

(b) A square matrix is one where the number of rows is equal to the number of columns. Define a function `isSquareMatrix :: [[a]] -> Bool` that checks if a list of lists is a square matrix.

Test cases:

isSquareMatrix $[] = \text{False}$
isSquareMatrix $[[[]]] = \text{False}$
isSquareMatrix $[[1]] = \text{True}$
isSquareMatrix $[[1,2,3],[4,5,6],[7,8,9]] = \text{True}$
isSquareMatrix $[[1,2,3,4],[5,6,7,8],[9,10,11,12]] = \text{False}$

(c) Two matrices are addable if they have the same number of rows and same number of columns. Define a function `addable :: [[a]] -> [[a]] -> Bool` that checks if two matrices are addable.

Test cases:

addable $[[1,2],[3,4]] [[1,2],[3,4]] = \text{True}$
addable $[[1,2],[3,4]] [[5,6,7],[8,9,10]] = \text{False}$
addable $[[1,2],[3,4]] [[1,2],[3,4],[3,4]] = \text{False}$

(d) Define a function `addMatrices :: [[Int]] -> [[Int]] -> [[Int]]` that computes the sum of the input matrices.

Test cases:

addMatrices $[[1,2]] [[3,4]] = [[4,6]]$
addMatrices $[[1,2],[3,4]] [[1,2],[3,4]] = [[2,4],[6,8]]$

(e) Matrix m_1 is multiplyable with matrix m_2 if the number of columns in m_1 is the same as the number of rows in m_2 . Define a function `multiplyable :: [[a]] -> [[a]] -> Bool` that checks if matrix m_1 is



multiplyable with m2.

Test cases:

multiplyable [[1,2,3],[4,5,6]] [[1,2],[3,4]] = False

multiplyable [[1,2,3],[4,5,6],[1,2,3],[4,5,6]] [[1,2],[3,4],[5,6]] = True

(f) Define a function multiplyMatrices :: [[Int]] -> [[Int]] -> [[Int]] that computes the product of the input matrices.

Test cases:

multiplyMatrices [[1,2],[3,4]] [[1,2,3],[4,5,6]] = [[9,12,15],[19,26,33]]

multiplyMatrices [[1,2,3],[4,5,6]] [[1,2],[3,4],[5,6]] = [[22,28],[49,64]]

Private Test cases used
for evaluation

	Input	Expected Output	Actual Output	Status
Test Case 1	subSeq "ab" "abc"	True	True\n	Passed
Test Case 2	subSeq "ab" "acb"	True	True\n	Passed
Test Case 3	subSeq "ab" "bca"	False	False\n	Passed
Test Case 4	subSeq "" "bea"	True	True\n	Passed
Test Case 5	subSeq "ba" "ba"	True	True\n	Passed
Test Case 6	subWord "ab" "abc"	True	True\n	Passed
Test Case 7	subWord "ab" "acb"	False	False\n	Passed
Test Case 8	subWord "ca" "bca"	True	True\n	Passed



Test Case 9

subWord "" "bea"

True

True\n

Passed

Test Case 10

subWord "ba" "ba"

True

True\n

Passed

Test Case 11

isMatrix []

False

False\n

Passed

Test Case 12

isMatrix [[],[],[[]]

False

False\n

Passed

Test Case 13

isMatrix [[2,3], [4,5], [6,7]]

True

True\n

Passed

Test Case 14

isMatrix [[2,3,4,5,6,7]]

True

True\n

Passed

Test Case 15

isSquareMatrix []

False

False\n

Passed

Test Case 16

isSquareMatrix [[]]

False

False\n

Passed

Test Case 17

isSquareMatrix [[1]]

True

True\n

Passed

Test Case 18

isSquareMatrix [[1,2,3],[4,5,6],[7,8,9]]

True

True\n

Passed

Test Case 19

isSquareMatrix [[1,2,3,4],[5,6,7,8],
[9,10,11,12]]

False

False\n

Passed

Test Case 20

addable [[1,2],[3,4]] [[1,2],[3,4]]

True

True\n

Passed

Test Case 21

addable [[1,2],[3,4]] [[5,6,7],[8,9,10]]

False

False\n

Passed

Test Case 22

addable [[1,2],[3,4]] [[1,2],[3,4],
[3,4]]

False

False\n

Passed

Test Case 23

addMatrices [[1,2]] [[3,4]]

[[4,6]]

[[4,6]]\n

Passed



Test Case 24	addMatrices [[1,2],[3,4]] [[1,2],[3,4]]	[[2,4],[6,8]]	[[2,4],[6,8]]\n	Passed
Test Case 25	multiplyable [[1,2,3],[4,5,6]] [[1,2],[3,4]]	False	False\n	Passed
Test Case 26	multiplyable [[1,2,3],[4,5,6],[1,2,3],[4,5,6]] [[1,2],[3,4],[5,6]]	True	True\n	Passed
Test Case 27	multiplyMatrices [[1,2],[3,4]] [[1,2,3],[4,5,6]]	[[9,12,15],[19,26,33]]	[[9,12,15],[19,26,33]]\n	Passed
Test Case 28	multiplyMatrices [[1,2,3],[4,5,6]] [[1,2],[3,4],[5,6]]	[[22,28],[49,64]]	[[22,28],[49,64]]\n	Passed

The due date for submitting this assignment has passed.

28 out of 28 tests passed.

You scored 100.0/100.

Assignment submitted on 2023-08-25, 00:30 IST

Your last recorded submission was :

```

1 extractArgs :: String -> (String, String)
2 extractArgs s = (str1, str2)
3   where
4     (_, _ : s1) = break (== '\\') s
5     (str1, _ : s2) = break (== '\\') s1
6     (_, _ : s3) = break (== '\\') s2
7     (str2, _) = break (== '\\') s3
8
9 main = do
10   line <- getLine;
11   let (func, rest) = break (== ' ') line in
12     case func of
13       "subSeq" -> let (str1, str2) = extractArgs (tail rest) in
14                   putStrLn . show $ subSeq str1 str2
15       "subWord" -> let (str1, str2) = extractArgs (tail rest) in
16                    putStrLn . show $ subWord str1 str2
17       "isMatrix" -> let args = read rest :: [[Int]] in
18                    putStrLn . show $ isMatrix args

```



```

19 "isSquareMatrix" -> let args = read rest :: [[Int]] in
20   putStrLn . show $ isSquareMatrix args
21 "addable" -> let (rest1, rest2) = break (== ' ') $ tail rest
22   arg1 = read rest1 :: [[Int]]
23   arg2 = read rest2 :: [[Int]] in
24   putStrLn . show $ addable arg1 arg2
25 "addMatrices" -> let (rest1, rest2) = break (== ' ') $ tail rest
26   arg1 = read rest1 :: [[Int]]
27   arg2 = read rest2 :: [[Int]] in
28   putStrLn . show $ addMatrices arg1 arg2
29 "multiplyable" -> let (rest1, rest2) = break (== ' ') $ tail rest
30   arg1 = read rest1 :: [[Int]]
31   arg2 = read rest2 :: [[Int]] in
32   putStrLn . show $ multiplyable arg1 arg2
33 "multiplyMatrices" -> let (rest1, rest2) = break (== ' ') $ tail rest
34   arg1 = read rest1 :: [[Int]]
35   arg2 = read rest2 :: [[Int]] in
36   putStrLn . show $ multiplyMatrices arg1 arg2
37 {-
38
39   1. Define a function subSeq :: String -> String -> Bool which checks whether
40 the first argument is a subsequence of the second. A subsequence is obtained by
41 deleting some letters in a string and retaining the other characters in the same
42 order as in the original string.
43
44 Test cases:
45 subSeq "ab" "abc" = True
46 subSeq "ab" "acb" = True
47 subSeq "ab" "bca" = False
48 subSeq "" "bea" = True
49 subSeq "ba" "ba" = True
50
51 2. Define a function subWord :: String -> String -> Bool which checks whether
52 the first argument is a subword of the second. A subword is obtained by deleting
53 some number (possibly 0) of letters at the left end and right end in a string
54 and retaining the other characters in the same order.
55
56 Test cases:
57 subWord "ab" "abc" = True
58 subWord "ab" "acb" = False
59 subWord "ca" "bca" = True
60 subWord "" "bea" = True
61 subWord "ba" "ba" = True
62
63 3. A two-dimensional matrix can be represented as a list of rows, each row
64 itself being a list of elements. So in general it is of type [[a]]. Not every
65 list of lists is a matrix, though. For instance, [[1,2,3], [], [2,4]] is a list
66 of three lists, each of a different size.
67
68 (a) Define a function isMatrix :: [[a]] -> Bool that checks if a list of lists
69 is a valid matrix (nonzero number of rows, each of the same nonzero length).

```




```

70
71 Test cases:
72 isMatrix [] = False
73 isMatrix [[],[],[ ]] = False
74 isMatrix [[2,3],[4,5],[6,7]] = True
75 isMatrix [[2,3,4,5,6,7]] = True
76
77 (b) A square matrix is one where the number of rows is equal to the number of
78 columns. Define a function isSquareMatrix :: [[a]] -> Bool that checks if a
79 list of lists is a square matrix.
80
81 Test cases:
82 isSquareMatrix [] = False
83 isSquareMatrix [[]] = False
84 isSquareMatrix [[1]] = True
85 isSquareMatrix [[1,2,3],[4,5,6],[7,8,9]] = True
86 isSquareMatrix [[1,2,3,4],[5,6,7,8],[9,10,11,12]] = False
87
88 (c) Two matrices are addable if they have the same number of rows and same
89 number of columns. Define a function addable :: [[a]] -> [[a]] -> Bool that
90 checks if two matrices are addable.
91
92 Test cases:
93 addable [[1,2],[3,4]] [[1,2],[3,4]] = True
94 addable [[1,2],[3,4]] [[5,6,7],[8,9,10]] = False
95 addable [[1,2],[3,4]] [[1,2],[3,4],[3,4]] = False
96
97 (d) Define a function addMatrices :: [[Int]] -> [[Int]] -> [[Int]] that computes
98 the sum of the input matrices.
99
100 Test cases:
101 addMatrices [[1,2]] [[3,4]] = [[4,6]]
102 addMatrices [[1,2],[3,4]] [[1,2],[3,4]] = [[2,4],[6,8]]
103
104 (e) Matrix m1 is multiplyable with matrix m2 if the number of columns in m1 is
105 the same as the number of rows in m2. Define a function
106 multiplyable :: [[a]] -> [[a]] -> Bool that checks if matrix m1 is
107 multiplyable with m2.
108
109 Test cases:
110 multiplyable [[1,2,3],[4,5,6]] [[1,2],[3,4]] = False
111 multiplyable [[1,2,3],[4,5,6],[1,2,3],[4,5,6]] [[1,2],[3,4],[5,6]] = True
112
113 (f) Define a function multiplyMatrices :: [[Int]] -> [[Int]] -> [[Int]] that
114 computes the product of the input matrices.
115
116 Test cases:
117 multiplyMatrices [[1,2],[3,4]] [[1,2,3],[4,5,6]] = [[9,12,15],[19,26,33]]
118 multiplyMatrices [[1,2,3],[4,5,6]] [[1,2],[3,4],[5,6]] = [[22,28],[49,64]]
119
120

```



```

121 -}
122
123
124 subSeq :: String -> String -> Bool
125 subSeq "" xs = True
126 subSeq s "" = False
127 subSeq s xs
128     | head s == head xs = subSeq (tail s) (tail xs)
129     | otherwise = subSeq s (tail xs)
130
131 subWord :: String -> String -> Bool
132
133 subWord "" xs = True
134 subWord s "" = False
135 subWord s xs
136     | head s == head xs = exactmatch s xs
137     | otherwise = subWord s (tail xs)
138     where exactmatch s xs
139           | s == take (length s) xs = True
140           | otherwise = subWord s (tail xs)
141
142
143 isMatrix :: [[a]] -> Bool
144 isMatrix [] = False
145 isMatrix [y]
146     | not (null y) = True
147     | otherwise = False
148 isMatrix x
149     | length (head x) == length (x!!1) = isMatrix $ tail x
150     | otherwise = False
151
152
153 isSquareMatrix :: [[a]] -> Bool
154 isSquareMatrix [] = False
155 isSquareMatrix x = foldl k True x
156                   where k acc ele = acc && length ele == length x && not (null ele)
157
158
159 addable :: [[a]] -> [[a]] -> Bool
160 addable x y = isMatrix x && isMatrix y && length x == length y && length (head x) == length (head y)
161
162
163 addMatrices :: [[Int]] -> [[Int]] -> [[Int]]
164 addMatrices xs ys = [ [ xs!!i!!j + ys!!i!!j | j<- [ 0 .. length (xs!!i) - 1] ] | i <- [0 .. length xs -1]]
165
166 multiplyable :: [[a]] -> [[a]] -> Bool
167 multiplyable xs ys = isMatrix xs && isMatrix ys && length ys == length (head xs)
168
169 multiplyMatrices :: [[Int]] -> [[Int]] -> [[Int]]
170 multiplyMatrices xs ys = [ [ k i j | j <- [0 .. length (head ys)-1] ] | i <- [0 .. length xs - 1]]
171     where k i j = sum [ xs!!i!!l * (ys!!l!!j) | l <- [0.. length (xs!!0)-1 ] ]

```



