Week 3, Lab Work 2: Functions on Top of Functions

Task

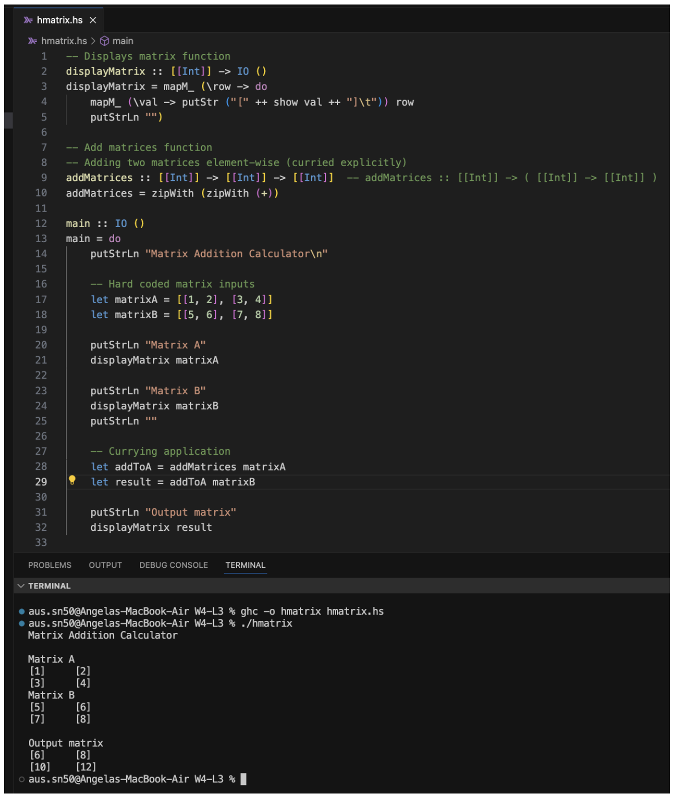
1. Select one operation *p* that is done on a pair of matrices A and B.
2. Using a **procedural** programming language, create a program that accepts from the user the values of A and B. (You may put in this program a requirement that the sizes(i.e. dimensions) of A and B are also user inputs. Alternatively, you can also have this as a predefined set of values). Then, let the program output the result of applying the operation *p* on A and B. Note: You can reuse/adopt any existing code available in literature or create your own. If the former applies, please cite the source in your submission of Lab Work 3.
3. From the program in (2) above, **convert** this program into one that shall be developed using a **functional** programming language. The latter program must use **currying** when performing the operation *p* on inputs A and B.
4. Make a set of screenshots that captures the **inputting** and **outputting** of your program in (3) for the following cases:
    a. Sunny day scenario: a successful application of *p* using A and B that produces a correct output. (at least two examples)
    b. Rainy day scenario: an unsuccessful application of *p* using A and B, e.g. dimensions of A and B are incompatible with respect to *p*, thus, no output is produced. (at least two examples)

Let *p*

Let *p* = matrix addition

Set of Screenshots (Haskell)

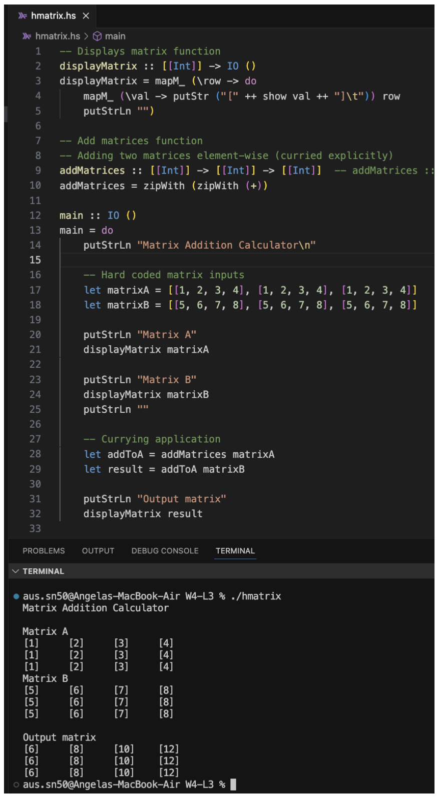
Sunny Day Scenarios



```haskell
-- Displays matrix function
displayMatrix :: [[Int]] -> IO ()
displayMatrix = mapM_ (\row -> do
    mapM_ (\val -> putStr ("[" ++ show val ++ "]\t")) row
    putStrLn "")

-- Add matrices function
-- Adding two matrices element-wise (curried explicitly)
addMatrices :: [[Int]] -> [[Int]] -> [[Int]]  -- addMatrices :: [[Int]] -> ( [[Int]] -> [[Int]] )
addMatrices = zipWith (zipWith (+))

main :: IO ()
main = do
    putStrLn "Matrix Addition Calculator\n"

    -- Hard coded matrix inputs
    let matrixA = [[1, 2], [3, 4]]
    let matrixB = [[5, 6], [7, 8]]

    putStrLn "Matrix A"
    displayMatrix matrixA

    putStrLn "Matrix B"
    displayMatrix matrixB
    putStrLn ""

    -- Currying application
    let addToA = addMatrices matrixA
    let result = addToA matrixB

    putStrLn "Output matrix"
    displayMatrix result

```

```
TERMINAL

aus.sn50@Angelas-MacBook-Air W4-L3 % ghc -o hmatrix hmatrix.hs
aus.sn50@Angelas-MacBook-Air W4-L3 % ./hmatrix
Matrix Addition Calculator

Matrix A
[1]     [2]
[3]     [4]
Matrix B
[5]     [6]
[7]     [8]

Output matrix
[6]     [8]
[10]    [12]
aus.sn50@Angelas-MacBook-Air W4-L3 %
```

#1: Dealing with small matrices
Adding a 2x2 matrix to a 2x2 matrix

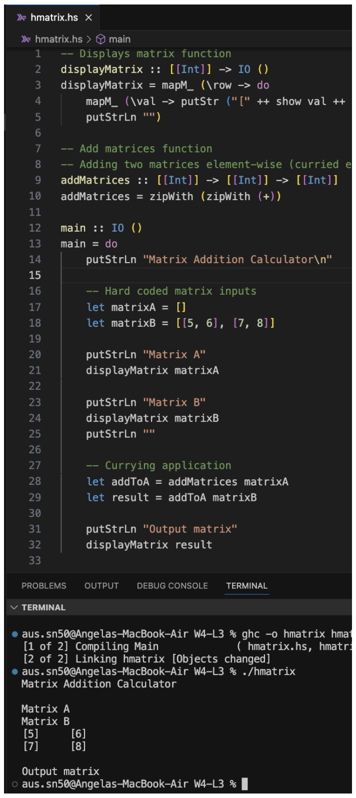


```haskell
-- Displays matrix function
displayMatrix :: [[Int]] -> IO ()
displayMatrix = mapM_ (\row -> do
    mapM_ (\val -> putStr ("[" ++ show val ++ "]\t")) row
    putStrLn "")

-- Add matrices function
-- Adding two matrices element-wise (curried explicitly)
addMatrices :: [[Int]] -> [[Int]] -> [[Int]]  -- addMatrices ::
addMatrices = zipWith (zipWith (+))

main :: IO ()
main = do
    putStrLn "Matrix Addition Calculator\n"

    -- Hard coded matrix inputs
    let matrixA = [[1, 2, 3, 4], [1, 2, 3, 4], [1, 2, 3, 4]]
    let matrixB = [[5, 6, 7, 8], [5, 6, 7, 8], [5, 6, 7, 8]]

    putStrLn "Matrix A"
    displayMatrix matrixA

    putStrLn "Matrix B"
    displayMatrix matrixB
    putStrLn ""

    -- Currying application
    let addToA = addMatrices matrixA
    let result = addToA matrixB

    putStrLn "Output matrix"
    displayMatrix result

```

```
TERMINAL

aus.sn50@Angelas-MacBook-Air W4-L3 % ./hmatrix
Matrix Addition Calculator

Matrix A
[1]     [2]     [3]     [4]
[1]     [2]     [3]     [4]
[1]     [2]     [3]     [4]
Matrix B
[5]     [6]     [7]     [8]
[5]     [6]     [7]     [8]
[5]     [6]     [7]     [8]

Output matrix
[6]     [8]     [10]    [12]
[6]     [8]     [10]    [12]
[6]     [8]     [10]    [12]
aus.sn50@Angelas-MacBook-Air W4-L3 %
```

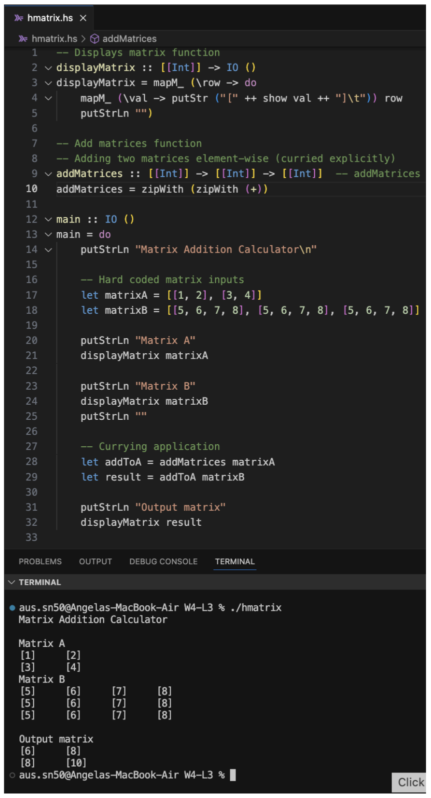
#2: Dealing with larger matrices
Adding a 4x4 matrix to a 4x4 matrix

Rainy Day Scenarios



```haskell
-- Displays matrix function
displayMatrix :: [[Int]] -> IO ()
displayMatrix = mapM_ (\row -> do
    mapM_ (\val -> putStr ("[" ++ show val ++
    putStrLn "")

-- Add matrices function
-- Adding two matrices element-wise (curried e
addMatrices :: [[Int]] -> [[Int]] -> [[Int]]
addMatrices = zipWith (zipWith (+))

main :: IO ()
main = do
    putStrLn "Matrix Addition Calculator\n"

    -- Hard coded matrix inputs
    let matrixA = []
    let matrixB = [[5, 6], [7, 8]]

    putStrLn "Matrix A"
    displayMatrix matrixA

    putStrLn "Matrix B"
    displayMatrix matrixB
    putStrLn ""

    -- Currying application
    let addToA = addMatrices matrixA
    let result = addToA matrixB

    putStrLn "Output matrix"
    displayMatrix result

```

```
TERMINAL

aus.sn50@Angelas-MacBook-Air W4-L3 % ghc -o hmatrix hmat
[1 of 2] Compiling Main             ( hmatrix.hs, hmatri
[2 of 2] Linking hmatrix [Objects changed]
aus.sn50@Angelas-MacBook-Air W4-L3 % ./hmatrix
Matrix Addition Calculator

Matrix A
Matrix B
[5]     [6]
[7]     [8]

Output matrix
aus.sn50@Angelas-MacBook-Air W4-L3 %
```



```haskell
-- Displays matrix function
displayMatrix :: [[Int]] -> IO ()
displayMatrix = mapM_ (\row -> do
    mapM_ (\val -> putStr ("[" ++ show val ++ "]\t")) row
    putStrLn "")

-- Add matrices function
-- Adding two matrices element-wise (curried explicitly)
addMatrices :: [[Int]] -> [[Int]] -> [[Int]]  -- addMatrices
addMatrices = zipWith (zipWith (+))

main :: IO ()
main = do
    putStrLn "Matrix Addition Calculator\n"

    -- Hard coded matrix inputs
    let matrixA = [[1, 2], [3, 4]]
    let matrixB = [[5, 6, 7, 8], [5, 6, 7, 8], [5, 6, 7, 8]]

    putStrLn "Matrix A"
    displayMatrix matrixA

    putStrLn "Matrix B"
    displayMatrix matrixB
    putStrLn ""

    -- Currying application
    let addToA = addMatrices matrixA
    let result = addToA matrixB

    putStrLn "Output matrix"
    displayMatrix result

```

```
TERMINAL

aus.sn50@Angelas-MacBook-Air W4-L3 % ./hmatrix
Matrix Addition Calculator

Matrix A
[1]     [2]
[3]     [4]
Matrix B
[5]     [6]     [7]     [8]
[5]     [6]     [7]     [8]
[5]     [6]     [7]     [8]

Output matrix
[6]     [8]
[8]     [10]
aus.sn50@Angelas-MacBook-Air W4-L3 %
```

#1: Dealing with an empty matrix	**#2: Dealing with matrix dimension mismatch**
Adding a 2x2 matrix to an empty matrix ● Empty output matrix	Adding a 2x2 matrix to a 4x4 matrix ● Truncated output matrix ● Invalid matrix addition