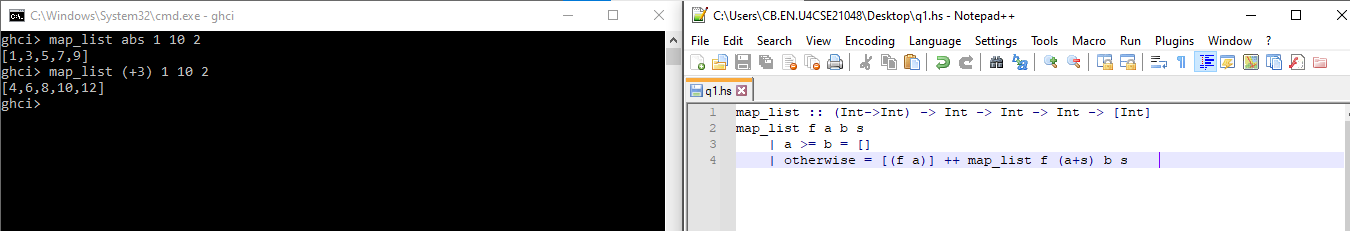
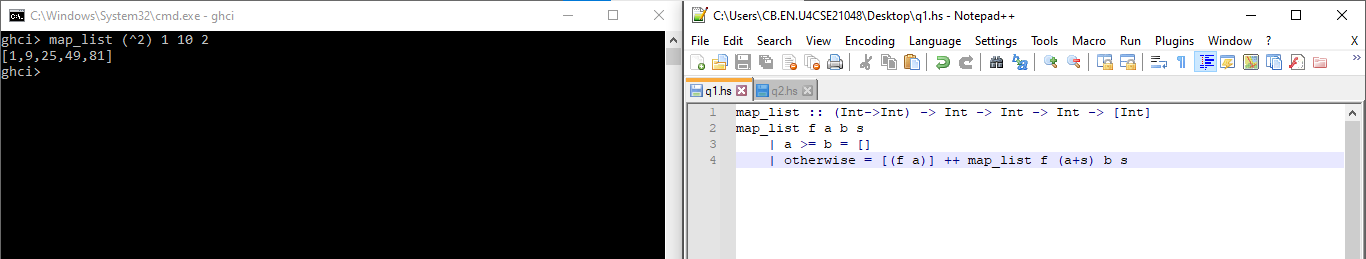
1.Write a haskell higher order function “map\_list” which generates a list of Int values starting from a, with an increment of next until it reaches b. The first argument is a function fn that takes an Int argument and returns a value of some type a. The second and third argument a and b are the lower limit and upper limit of the integer range to generate the list. The fourth argument next is the increment value to use when generating the list.

example usage of the map\_list function:

map\_list square 1 10 2

Output: [1, 9, 25, 49, 81]





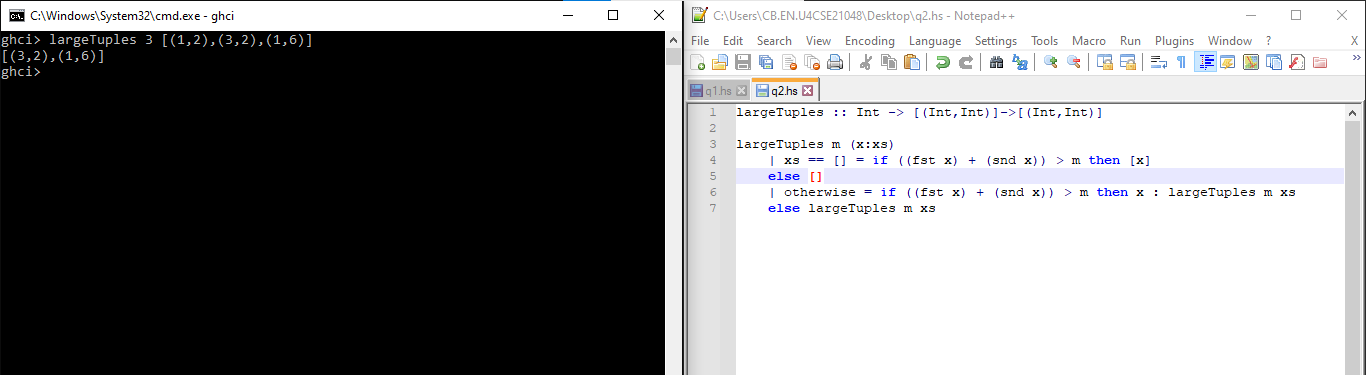
map\_list :: (Int->Int) -> Int -> Int -> Int -> [Int]

map\_list f a b s

| a >= b = []

| otherwise = [(f a)] ++ map\_list f (a+s) b s

2.Write a Haskell program only using recursion without higher-order function and list comprehension with the function largeTuples whose arguments are max and list of   tuples which produces a list of the tuples whose sum is larger than max, i.e., for a tuple (a,b), a + b > max.

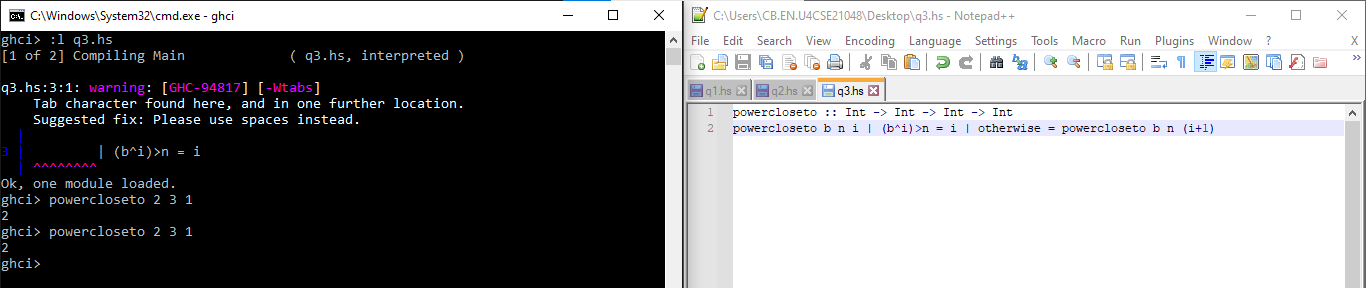


3. The procedure power-close-to which takes three non-zero positive integer b n and i as arguments and returns the smallest power of b, that is greater than n ***by defining the procedures*** ***(expt b i)*** which raises ***b*** to the power ***i*** and ***(inc i)*** which increments ***i*** by 1.

**Example**

**Power-close-to 2 3 1**

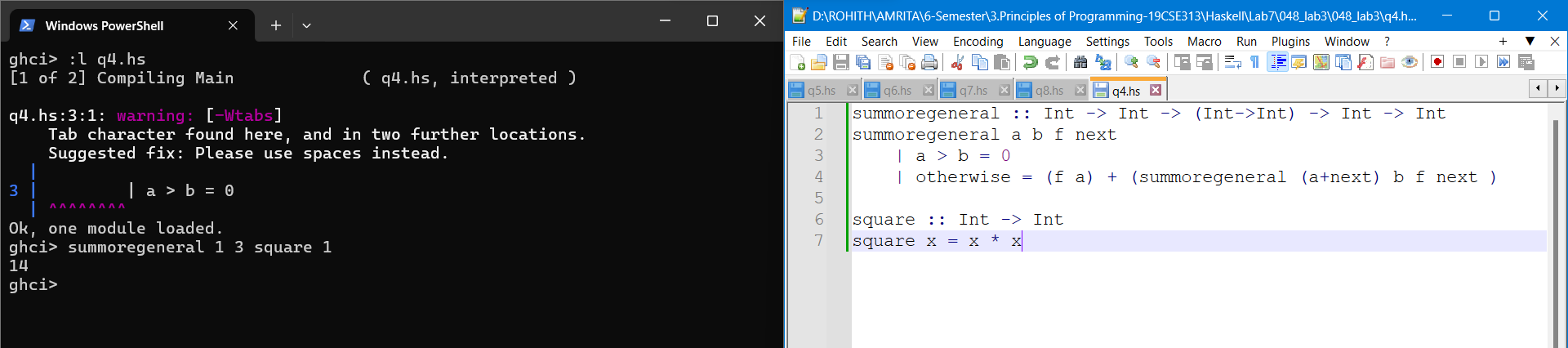
**2**



1. Write a higher-order procedure called **(sum-more-genl a b fn next)** where a and b are the lower and upper limits of integer type, fn is a function and next is the increment value. Some sample executions of **sum-more-genl** are as follows:

**>(sum-more –genl 1 3 square inc) à 14** (where procedures **square** is defined and procedure **inc** increments the argument by 1)

**>(sum-more-genl 1 3 factorial sum2) à 7** (where procedure **factorial** is defined andprocedure **sum2** is defined as the argument plus 2).

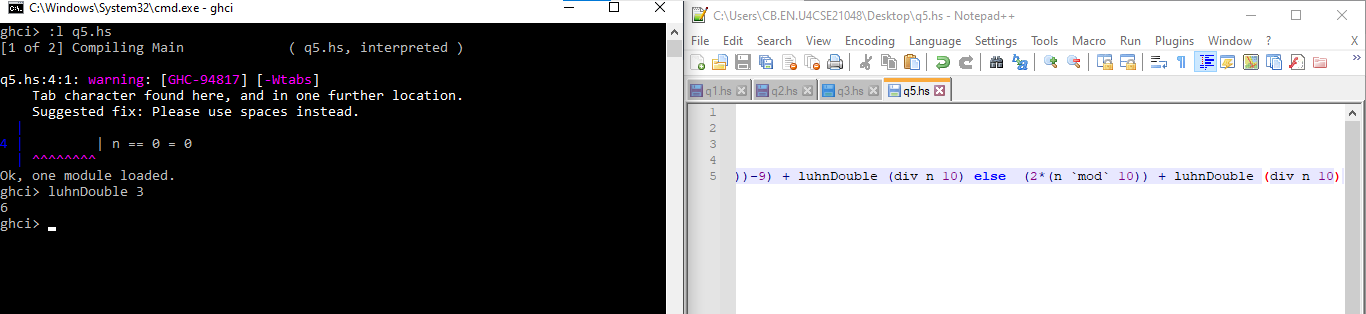


5. The Luhn algorithm is used to check bank card numbers for simple errors such as mistyping a digit, and proceeds as follows: consider each digit as a separate number; moving left, double every other number from the second last; subtract 9 from each number that is now greater than 9; add all the resulting numbers together; if the total is divisible by 10, the card number is valid. In order to accomplish this:

Define a function luhnDouble :: Int -> Int that doubles a digit and subtracts 9 if the result is greater than 9.

For example:

> luhnDouble 3



6. Write a higher-order procedure called **(sum-factorials a b fn next)** where **a** and **b** are the lower and upper limits of integer type, **fn** is a function and **next** is the increment value. **sum-factorials** must calculate factorials for (i) every integer and (ii) every alternate integer a in the limit **a** to **b** (starting from a). Some sample executions of **sum-factorials** are as follows:

**> (sum-factorials 1 3 factorial inc) returns 9 (1! +2! +3!)**

**> (sum-factorials 1 4 factorial inc2) returns 7 (1! +3!)**

