20CYS312 - PPL - Lab Exercise 3

Name: R.Subramanian

Roll.No: CH.EN.U4CYS22043

Date: 13/12/24

---

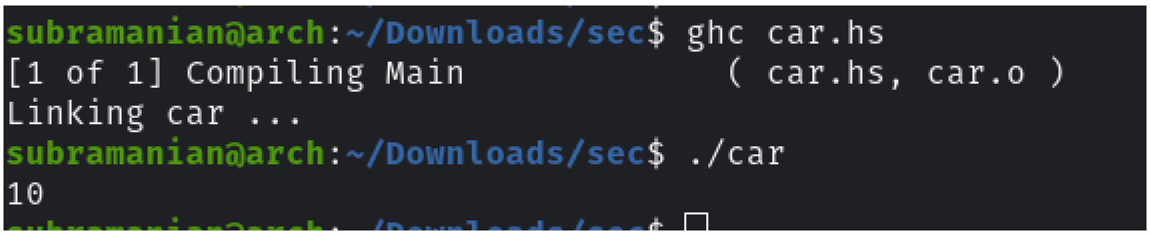
1. Basic Data Types

a. Sum of two integers

```haskell
sumIntegers :: Int -> Int -> Int
sumIntegers x y = x + y

main :: IO ()
main = print (sumIntegers 3 7)  -- Output: 10
```

Output Screenshot:



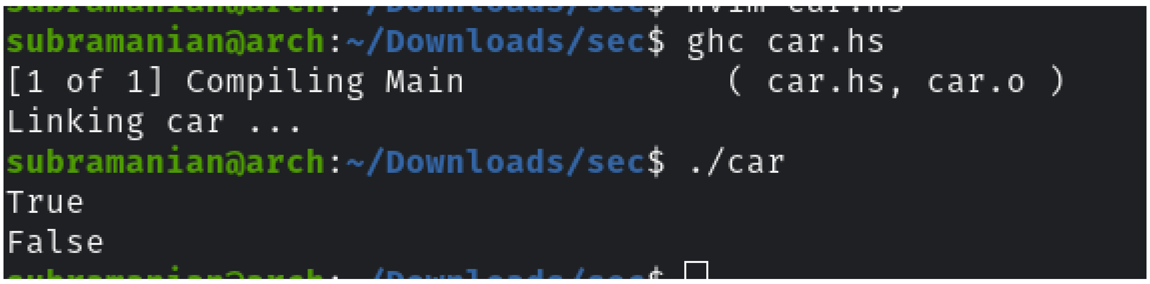
```
subramanian@arch:~/Downloads/sec$ ghc car.hs
[1 of 1] Compiling Main             ( car.hs, car.o )
Linking car ...
subramanian@arch:~/Downloads/sec$ ./car
10
```

b. Check if a number is even or odd

```haskell
isEven :: Int -> Bool
isEven x = x `mod` 2 == 0

main :: IO ()
main = do
    print (isEven 4)  -- Output: True
    print (isEven 5)  -- Output: False
```

Output Screenshot:



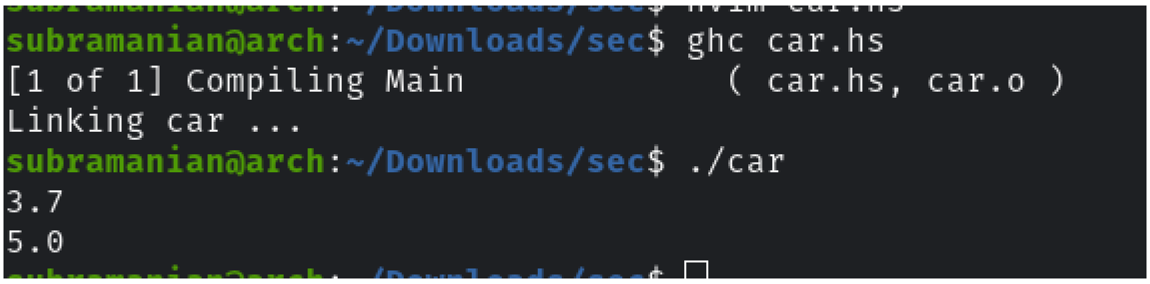
```
subramanian@arch:~/Downloads/sec$ ghc car.hs
[1 of 1] Compiling Main             ( car.hs, car.o )
Linking car ...
subramanian@arch:~/Downloads/sec$ ./car
True
False
```

c. Absolute value

```haskell
absolute :: Float -> Float
absolute x = if x < 0 then -x else x

main :: IO ()
main = do
    print (absolute (-3.7))  -- Output: 3.7
    print (absolute 5.0)     -- Output: 5.0
```

Output Screenshot:



```
subramanian@arch:~/Downloads/sec$ ghc car.hs
[1 of 1] Compiling Main             ( car.hs, car.o )
Linking car ...
subramanian@arch:~/Downloads/sec$ ./car
3.7
5.0
```

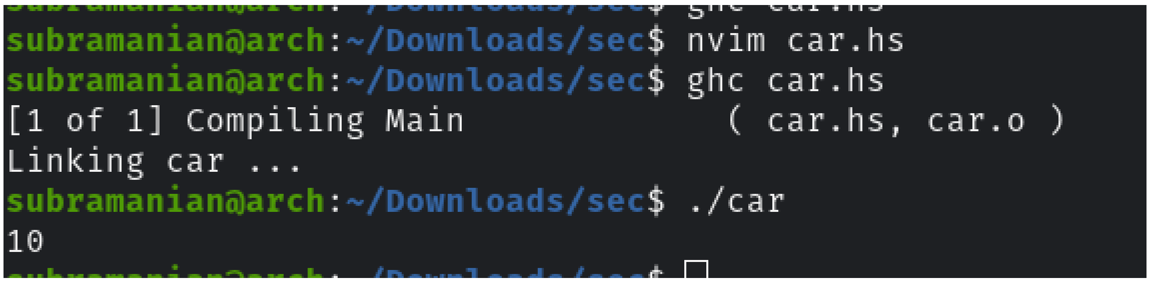
2. List Operations

a. Sum of all elements

```haskell
sumList :: [Int] -> Int
sumList xs = sum xs

main :: IO ()
main = print (sumList [1, 2, 3, 4])  -- Output: 10
```

Output Screenshot:



```
subramanian@arch:~/Downloads/sec$ nvim car.hs
subramanian@arch:~/Downloads/sec$ ghc car.hs
[1 of 1] Compiling Main             ( car.hs, car.o )
Linking car ...
subramanian@arch:~/Downloads/sec$ ./car
10
```

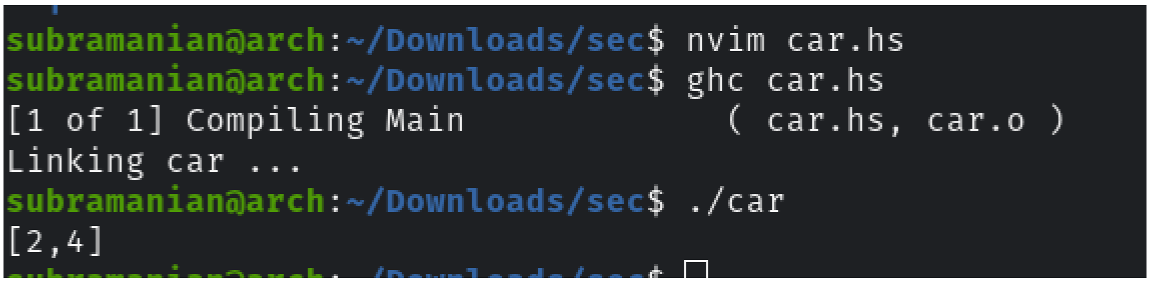
b. Filter even numbers

```haskell
-- Define the isEven function
isEven :: Int -> Bool
isEven x = x `mod` 2 == 0

-- Define the filterEven function
filterEven :: [Int] -> [Int]
filterEven xs = filter isEven xs

-- Main function to print the filtered result
main :: IO ()
main = print (filterEven [1, 2, 3, 4, 5])  -- Output: [2, 4]
```

Output Screenshot:



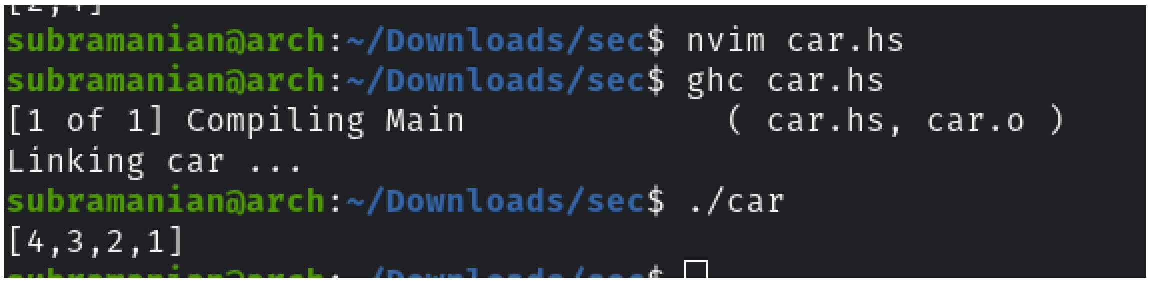
```
subramanian@arch:~/Downloads/sec$ nvim car.hs
subramanian@arch:~/Downloads/sec$ ghc car.hs
[1 of 1] Compiling Main             ( car.hs, car.o )
Linking car ...
subramanian@arch:~/Downloads/sec$ ./car
[2,4]
```

c. Reverse a list

```haskell
reverseList :: [a] -> [a]
reverseList xs = reverse xs

main :: IO ()
main = print (reverseList [1, 2, 3, 4])  -- Output: [4, 3, 2, 1]
```

Output Screenshot:



```
subramanian@arch:~/Downloads/sec$ nvim car.hs
subramanian@arch:~/Downloads/sec$ ghc car.hs
[1 of 1] Compiling Main             ( car.hs, car.o )
Linking car ...
subramanian@arch:~/Downloads/sec$ ./car
[4,3,2,1]
```

---

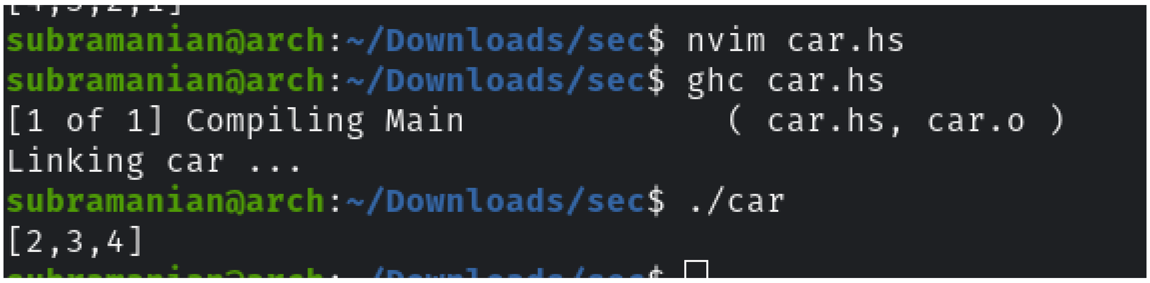
3. Basic Functions

a. Increment each element

```haskell
incrementEach :: [Int] -> [Int]
incrementEach xs = map (+1) xs

main :: IO ()
main = print (incrementEach [1, 2, 3])  -- Output: [2, 3, 4]
```

Output Screenshot:



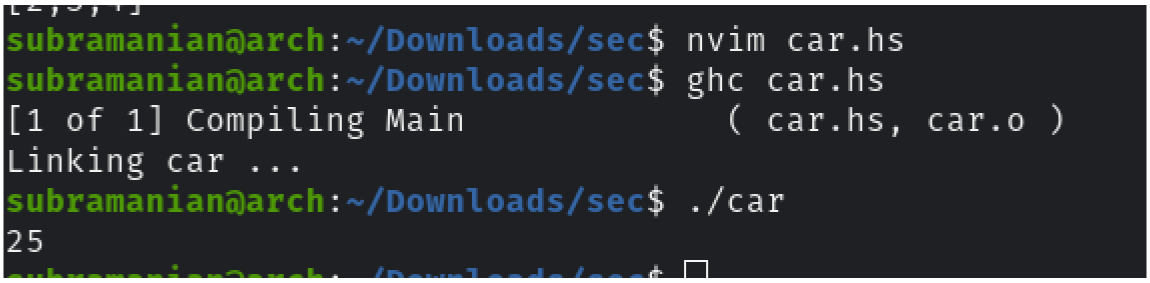
```
subramanian@arch:~/Downloads/sec$ nvim car.hs
subramanian@arch:~/Downloads/sec$ ghc car.hs
[1 of 1] Compiling Main             ( car.hs, car.o )
Linking car ...
subramanian@arch:~/Downloads/sec$ ./car
[2,3,4]
```

b. Square a number

```haskell
square :: Int -> Int
square x = x * x

main :: IO ()
main = print (square 5)  -- Output: 25
```

Output Screenshot:



```
subramanian@arch:~/Downloads/sec$ nvim car.hs
subramanian@arch:~/Downloads/sec$ ghc car.hs
[1 of 1] Compiling Main             ( car.hs, car.o )
Linking car ...
subramanian@arch:~/Downloads/sec$ ./car
25
```

---

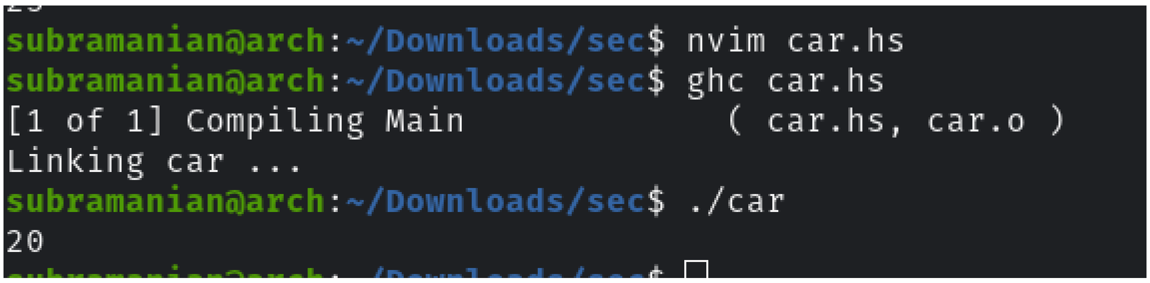
4. Function Composition

a. Compose functions to add and multiply

```haskell
addThenMultiply :: Int -> Int -> Int -> Int
addThenMultiply x y z = (x + y) * z

main :: IO ()
main = print (addThenMultiply 2 3 4)  -- Output: 20
```

Output Screenshot:



```
subramanian@arch:~/Downloads/sec$ nvim car.hs
subramanian@arch:~/Downloads/sec$ ghc car.hs
[1 of 1] Compiling Main             ( car.hs, car.o )
Linking car ...
subramanian@arch:~/Downloads/sec$ ./car
20
```

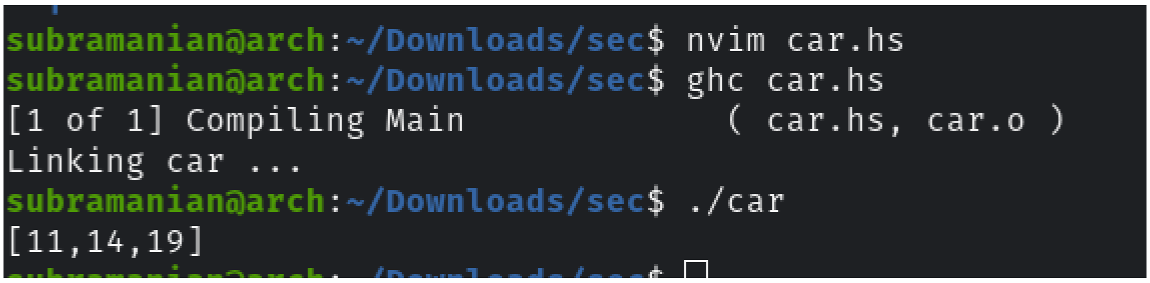
b. Apply multiple transformations

```haskell
square :: Int -> Int
square x = x * x

transformList :: [Int] -> [Int]
transformList = map ((+10) . square)

main :: IO ()
main = print (transformList [1, 2, 3])  -- Output: [11, 14, 19]
```

Output Screenshot:



```
subramanian@arch:~/Downloads/sec$ nvim car.hs
subramanian@arch:~/Downloads/sec$ ghc car.hs
[1 of 1] Compiling Main             ( car.hs, car.o )
Linking car ...
subramanian@arch:~/Downloads/sec$ ./car
[11,14,19]
```