

Department of Applied Mathematics and Computational Sciences

PSG College of Technology

Programme: MSc SS VIII semester

Course: Functional Programming Lab

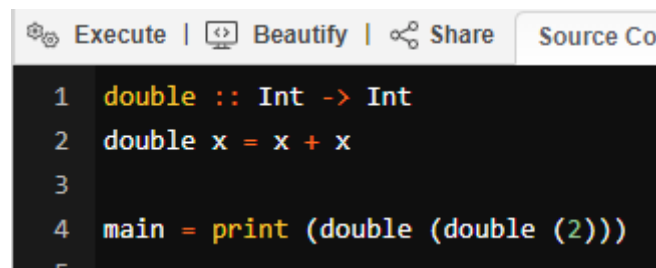
20PW16

PROBLEM SHEET 1

(exercises based on Programming in Haskell, Graham Hutton)

INTRODUCTION

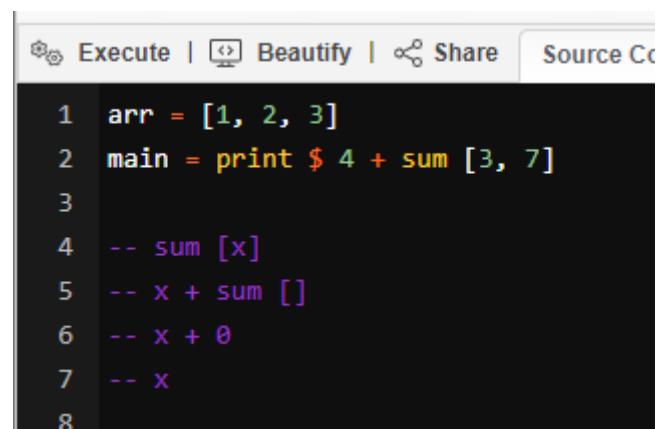
1. Give another possible calculation for the result of `double (double 2)`.



```
Execute | Beautify | Share | Source Co
1 double :: Int -> Int
2 double x = x + x
3
4 main = print (double (double (2)))
5
```

`double(2+2)`
`(2+2)+(2+2)`
`4+4`
`8`

2. Show that `sum[x] = x` for any number `x`.



```
Execute | Beautify | Share | Source Co
1 arr = [1, 2, 3]
2 main = print $ 4 + sum [3, 7]
3
4 -- sum [x]
5 -- x + sum []
6 -- x + 0
7 -- x
8
```

3. Define a function `product` that produces the product of a list of numbers, and show using your definition that `product [2, 3, 4] = 24`.sum

```

Execute | Beautify | Share | Source Code
1  product' :: [Int] -> Int
2
3  product' [] = 1
4  product' (x : arr) = x * product' arr
5
6  main = do
7      print $ product' [2, 3, 4]
8

```

4. How should the definition of the function *qsort* be modified so that it produces a *reverse* sorted version of a list?

```

1  qsort [] = []
2
3  qsort (x:arr) = qsort left ++ [x] ++ qsort right
4      where
5          left = [i | i <- arr, i >= x]
6          right = [i | i <- arr, i < x]
7
8  main = print (qsort([8, 2, 2, 3, 1, 1]))
9

```

5. What would be the effect of replacing \leq by $<$ in the definition of *qsort*? Hint: consider the example *qsort* [2, 2, 3, 1, 1].

```

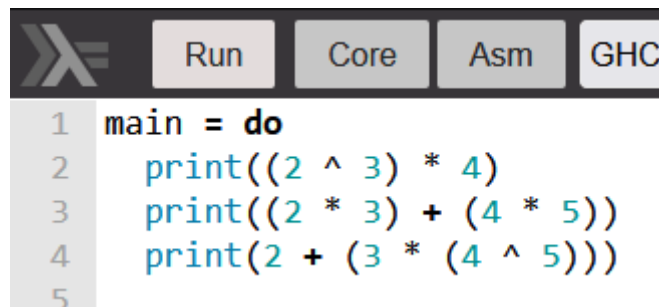
Run | Core | Asm | GHC 9.4.8 | -O1 | Save &
1  qsort [] = []
2
3  qsort (x:arr) = qsort left ++ [x] ++ qsort right
4      where
5          left = [i | i <- arr, i < x]
6          right = [i | i <- arr, i > x]
7
8  main = print (qsort([2, 2, 3, 1, 1]))
9
10  --output [1, 2, 3]
11

```

FIRST STEPS

6. Parenthesise the following arithmetic expressions:

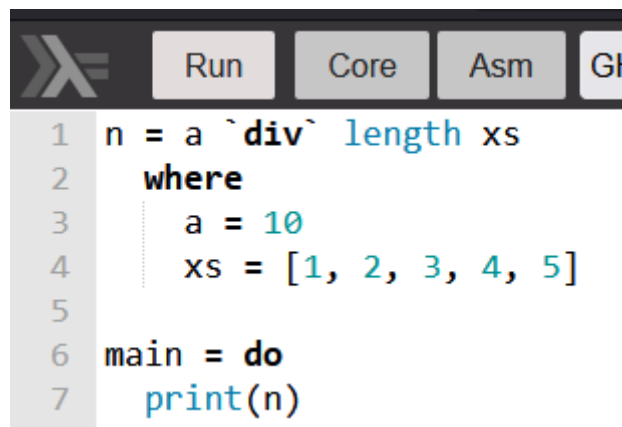
$2 \uparrow 3 \times 4$
 $2 \times 3 + 4 \times 5$
 $2 + 3 \times 4 \uparrow 5$



```
1 main = do
2   print((2 ^ 3) * 4)
3   print((2 * 3) + (4 * 5))
4   print(2 + (3 * (4 ^ 5)))
5
```

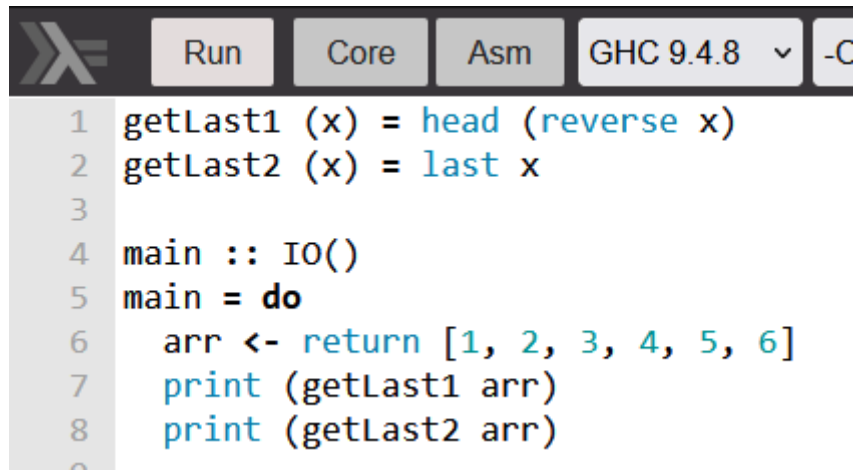
7. Work through the examples from this chapter using Hugs.
8. The script below contains three syntactic errors. Correct these errors and then check that your script works properly using Hugs.

$N = a \text{ 'div' } \text{length } xs$
where
 $a = 10$
 $xs = [1, 2, 3, 4, 5]$



```
1 n = a `div` length xs
2   where
3     a = 10
4     xs = [1, 2, 3, 4, 5]
5
6 main = do
7   print(n)
```

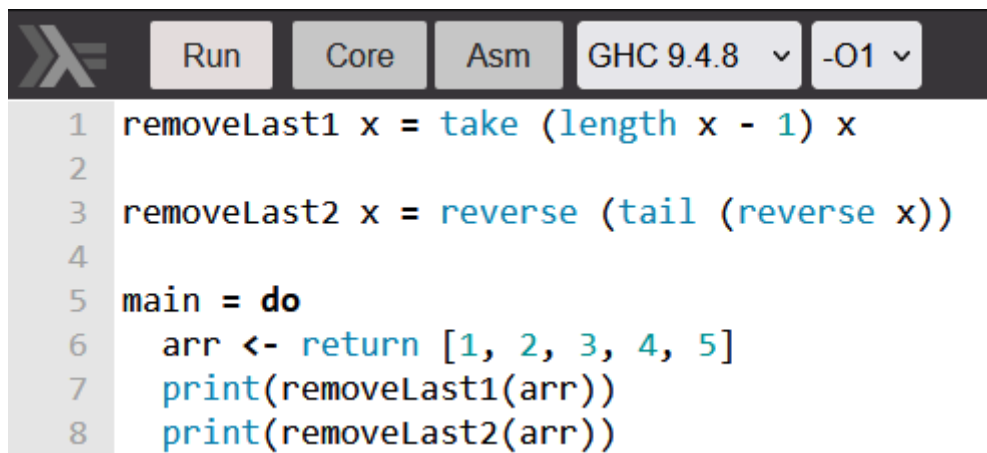
9. Show how the library function `last` that selects the last element of a nonempty list could be defined in terms of the library functions introduced in this chapter. Can you think of another possible definition?



The image shows a screenshot of the GHCi (Glasgow Haskell Compiler interactive) interface. At the top, there is a toolbar with buttons for 'Run', 'Core', 'Asm', and a dropdown menu showing 'GHC 9.4.8'. To the left of these buttons is a large 'X' icon. Below the toolbar, a list of Haskell code snippets is displayed, numbered 1 through 8. The code defines two functions, 'getLast1' and 'getLast2', and a 'main' function that uses them.

```
1 getLast1 (x) = head (reverse x)
2 getLast2 (x) = last x
3
4 main :: IO()
5 main = do
6   arr <- return [1, 2, 3, 4, 5, 6]
7   print (getLast1 arr)
8   print (getLast2 arr)
```

10. Show how the library function `init` that removes the last element from a non-empty list could similarly be defined in two different ways.



The image shows a screenshot of the GHCi (Glasgow Haskell Compiler interactive) interface. At the top, there is a toolbar with buttons for 'Run', 'Core', 'Asm', a dropdown menu showing 'GHC 9.4.8', and another dropdown menu showing '-O1'. To the left of these buttons is a large 'X' icon. Below the toolbar, a list of Haskell code snippets is displayed, numbered 1 through 8. The code defines two functions, 'removeLast1' and 'removeLast2', and a 'main' function that uses them.

```
1 removeLast1 x = take (length x - 1) x
2
3 removeLast2 x = reverse (tail (reverse x))
4
5 main = do
6   arr <- return [1, 2, 3, 4, 5]
7   print(removeLast1(arr))
8   print(removeLast2(arr))
```