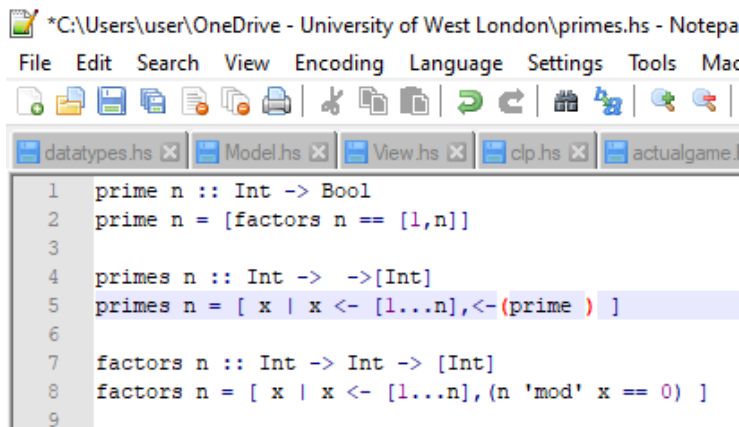


Functional Programming A1 Element 1 Resit

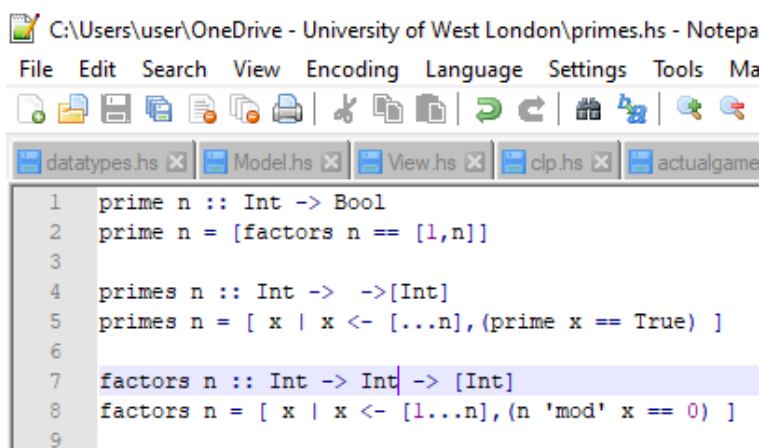
Stefan M Ahmed 21359035

1. An integer is prime if it can be divided by one and itself i.e. one and itself are its factors. Working with list comprehension I am going to write a function to list all the prime numbers up to a given limit. Give the function a definition first:



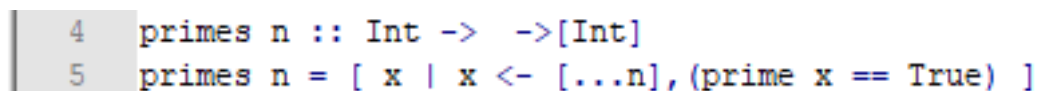
```
*C:\Users\user\OneDrive - University of West London\primes.hs - Notepad++
File Edit Search View Encoding Language Settings Tools Mac
[Icons]
datatypes.hs x Model.hs x View.hs x clp.hs x actualgame.hs
1 prime n :: Int -> Bool
2 prime n = [factors n == [1,n]]
3
4 primes n :: Int -> ->[Int]
5 primes n = [ x | x <- [1..n], <-(prime ) ]
6
7 factors n :: Int -> Int -> [Int]
8 factors n = [ x | x <- [1..n], (n 'mod' x == 0) ]
9
```

I start by writing code to define the functions in order to list the prime numbers up to a given limit.



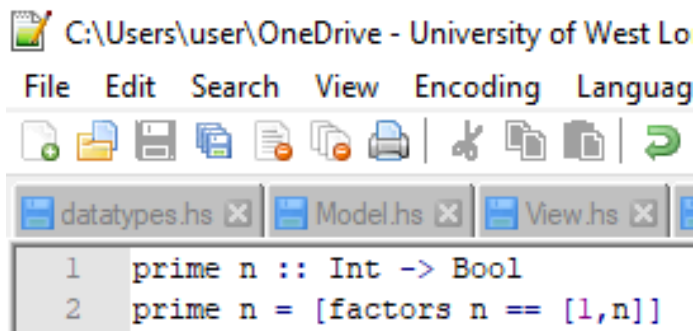
```
C:\Users\user\OneDrive - University of West London\primes.hs - Notepad++
File Edit Search View Encoding Language Settings Tools Ma
[Icons]
datatypes.hs x Model.hs x View.hs x clp.hs x actualgame.hs
1 prime n :: Int -> Bool
2 prime n = [factors n == [1,n]]
3
4 primes n :: Int -> ->[Int]
5 primes n = [ x | x <- [...n], (prime x == True) ]
6
7 factors n :: Int -> Int -> [Int]
8 factors n = [ x | x <- [1..n], (n 'mod' x == 0) ]
9
```

The function prime works with a data of type integer and then a Boolean type of data being returned (Yes or No, Y or N, True or false). It checks if a number n is a prime number by working with the factors function. If the factors of n are one and itself, given by one and n in a list [], the function gives a Boolean value Y, Yes, or True.#



```
4 primes n :: Int -> ->[Int]
5 primes n = [ x | x <- [...n], (prime x == True) ]
```

And then there is the function primes that gives the list of prime numbers up to a limit n. It returns data of type integers in a list. I have left a space in between the Int it returns in a list and the integers it first works with, for another type of data such as Bool (for Boolean) if that is needed. All the functions, including this one return a list [] of integers x, except prime. This is by writing the code in square brackets [...n] to make the list up to a limit n. List [...n] points to x which means x is each integer in the list to be returned up to the limit n. To make each of them be returned we have x at the beginning – [x | x <- [...n]].



```
1 prime n :: Int -> Bool
2 prime n = [factors n == [1,n]]
```

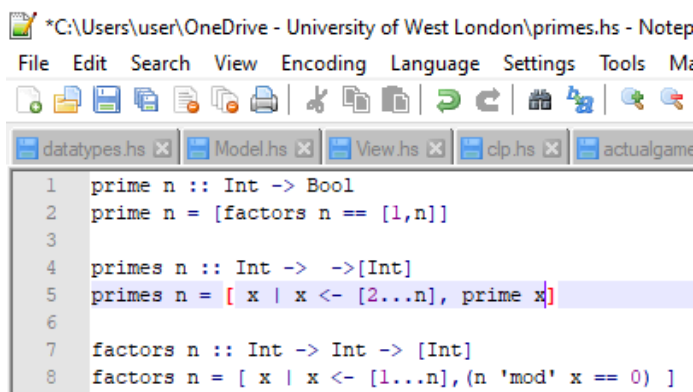
The prime function is needed to make this function work by checking if each integer is a prime number. It checks if each integer is a prime or not before putting it in the list.

```
1 prime n :: Int -> Bool
2 prime n = factors n == [1,n]
3
4 primes n :: Int -> [Int]
5 primes n = [ x | x <- [2..n], factors x == [1,x]]
```

I could also try checking the factors of each integer instead by putting factors of x equalling and itself, in square brackets to confirm they are prime or not. Since this code is in the definition of the prime function we can try it in this functions definition.

```
4 primes n :: Int -> [Int]
5 primes n = [ x | x <- [2..n], prime x ]
```

Since “one is not a prime number” let’s put a two in the brackets of integers up to a limit n. So the list starts from two up to a limit n.



```
1 prime n :: Int -> Bool
2 prime n = [factors n == [1,n]]
3
4 primes n :: Int -> [Int]
5 primes n = [ x | x <- [2..n], prime x]
6
7 factors n :: Int -> [Int]
8 factors n = [ x | x <- [1..n], (n `mod` x == 0) ]
```

I have decided not to have prime x == True if that does not make the function work. We only need the function and the number. No need to put Bool between the Int and [Int].

```
1 prime n :: Int -> Bool
2 prime n = [factors n == [1,n]]
3
4 primes n :: Int -> [Int]
5 primes n = [ x | x <- [2..n], prime x]
6
7 factors n :: Int -> [Int]
8 factors n = [ x | x <- [1..n], (n 'mod' x == 0) ]
```

So the function factors works with numbers that are integers and returns a list of the factors x of the number n. This is by the number n being divided by its factors x with no remainder by the code n 'mod' x == 0. Modulo is the number being divided by its factors with no remainder and is "mod" for short.

```
1 prime n :: Int -> Bool
2 prime n = factors n == [1,n]
3
4 primes n :: Int -> [Int]
5 primes n = [ x | x <- [2..n], prime x]
6
7 factors n :: Int -> [Int]
8 factors n = [ x | x <- [1..n], n 'mod' x == 0]
```

I have put the correct data types including those that are returned, and removed brackets, I don't need, from the code to correct it. I have to check the code is completely correct before I run it.

```
1 prime :: Int -> Bool
2 prime n = factors n == [1,n]
3
4 primes :: Int -> [Int]
5 primes n = [ x | x <- [2..n], prime x]
6
7 factors :: Int -> [Int]
8 factors n = [ x | x <- [1..n], n `mod` x == 0]
```

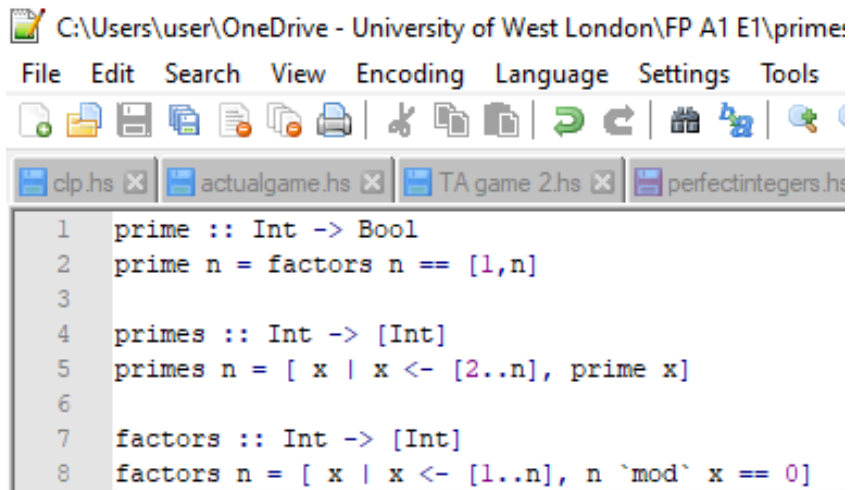
I have to remove the apostrophes around 'mod' and put backquotes around it instead. 'mod' now becomes `mod`.

```

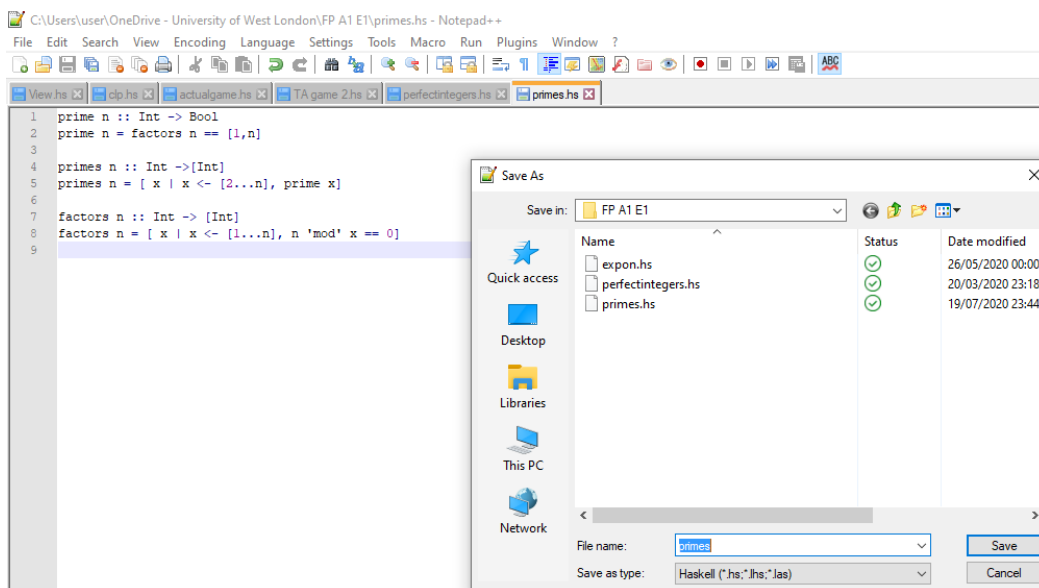
4 primes :: Int -> [Int]
5 primes n = [ x | x <- [2..n], prime x]
6
7 factors :: Int -> [Int]
8 factors n = [ x | x <- [1..n], n `mod` x == 0]

```

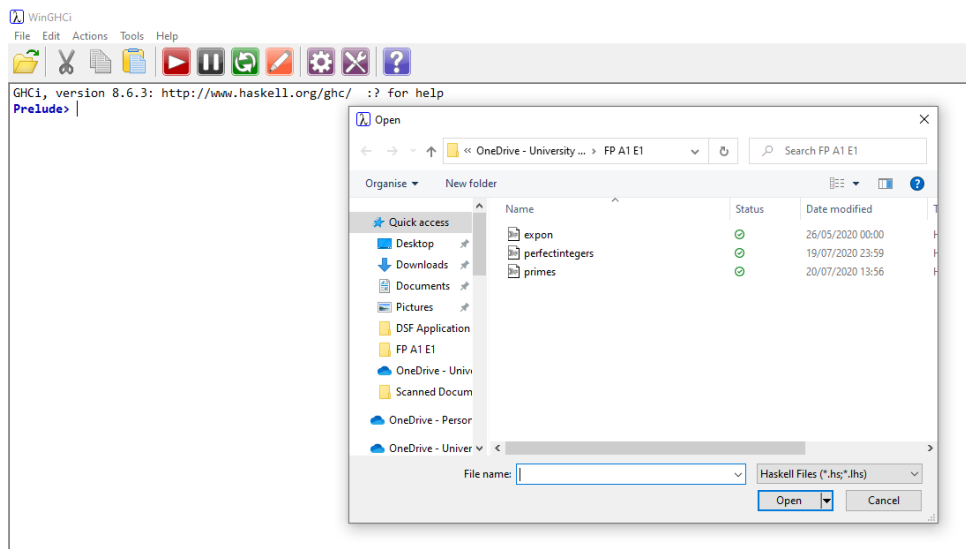
I then have to remove a dot from the square brackets, that make the list of prime numbers and factors up to n. Two dots in the square brackets make the code run instead of three.



The code is now completely correct. It is ready to be run.



I call it primes and save it with the .hs file extension. The .hs extension is for code written in the Haskell programming language – hs for Haskell.



So I open the WinGhci compiler and load the code file I saved as primes.hs.

```

WinGHCi
File Edit Actions Tools Help

GHCi, version 8.6.3: http://www.haskell.org/ghc/  :? for help
Prelude> :cd C:\Users\user\OneDrive - University of West London\FP A1 E1
Prelude> :load "primes.hs"
[1 of 1] Compiling Main                ( primes.hs, interpreted )
Ok, one module loaded.
*Main> prime 13
True
*Main>

```

The file is loaded free off errors. I start checking if 13 is a prime number by typing prime 13. The result returned below it is True. The prime function I wrote code for does work.

```

WinGHCi
File Edit Actions Tools Help

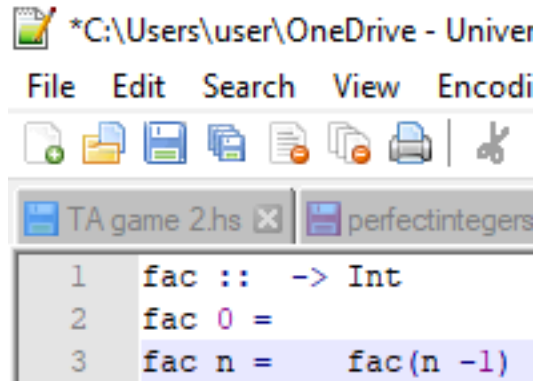
GHCi, version 8.6.3: http://www.haskell.org/ghc/  :? for help
Prelude> :cd C:\Users\user\OneDrive - University of West London\FP A1 E1
Prelude> :load "primes.hs"
[1 of 1] Compiling Main                ( primes.hs, interpreted )
Ok, one module loaded.
*Main> prime 13
True
*Main> primes 64
[2,3,5,7,11,13,17,19,23,29,31,37,41,43,47,53,59,61]
*Main> factors 3
[1,3]

```

I then try the other functions. I see if the primes function works by typing primes 64 to get a list of all the prime numbers up to 64. The result returned below it is a list of all the prime numbers in square brackets all the way from 2 to 61. 61 is by far the last prime number close to 64. I then try the factors function to find the factors of a number. This returns the result 1 and 3 in square brackets which are the factors of 3 when I type factors 3. The two functions work.

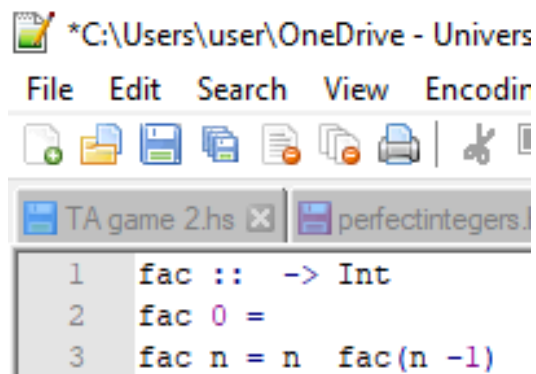
- Recursive functions are functions that work with themselves in their definition. Recursive functions have their function in their definition, for example the function is defined as product (n:ns) = n * product ns with product being in the definition.

I will be trying a function that is recursive by putting the function in its definition to work with itself.



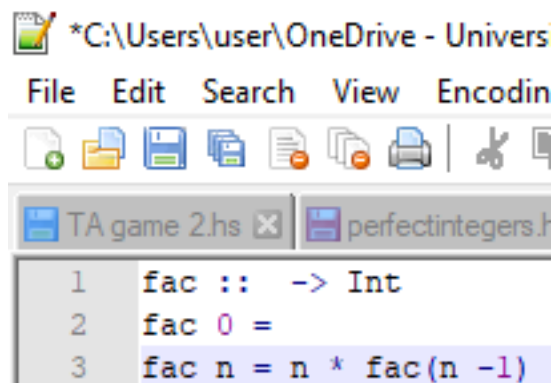
```
*C:\Users\user\OneDrive - Univers
File Edit Search View Encodi
TA game 2.hs x perfectintegers
1 fac :: -> Int
2 fac 0 =
3 fac n = fac (n -1)
```

The factorial of a number is basically every number from 1 to the number itself being multiplied together but this function works it out recursively. I start it with the data types being Int if this function involves numbers and is a sum or a maths problem and the word fac being short for factorial. I think of putting Bool if it is needed here.



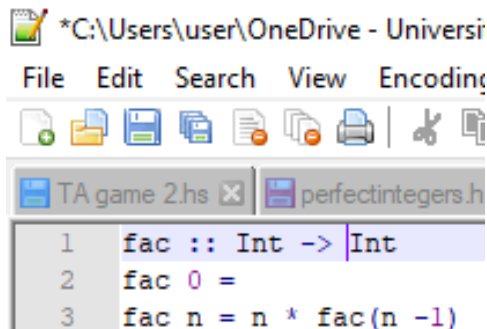
```
*C:\Users\user\OneDrive - Univers
File Edit Search View Encodin
TA game 2.hs x perfectintegers.
1 fac :: -> Int
2 fac 0 =
3 fac n = n fac (n -1)
```

To be recursive the function itself needs to go in the definition. It goes before, in or after the brackets. It has n for any number. The factorial of a number such as 5 is written as factorial 5 or !5.



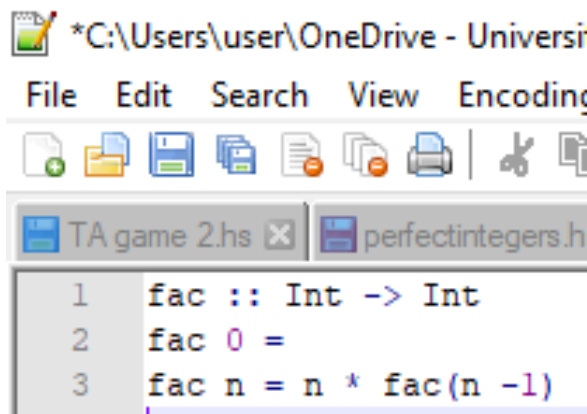
```
*C:\Users\user\OneDrive - Univers
File Edit Search View Encodin
TA game 2.hs x perfectintegers.f
1 fac :: -> Int
2 fac 0 =
3 fac n = n * fac (n -1)
```

Since this function works with numbers a multiplication and subtraction sign is needed in the definition since numbers are being multiplied and taken away.



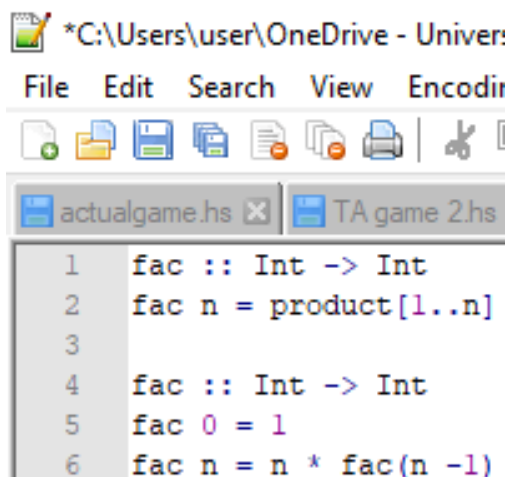
```
*C:\Users\user\OneDrive - Universi
File Edit Search View Encoding
TA game 2.hs x perfectintegers.h
1 fac :: Int -> Int
2 fac 0 =
3 fac n = n * fac(n -1)
```

Since this recursive function works with numbers, I am thinking to put data types as Int above the definition. It works by multiplying and subtracting numbers and then returning a number as a result, so I decide to make it return an Int and not put other data types such as Bool.



```
*C:\Users\user\OneDrive - Universi
File Edit Search View Encoding
TA game 2.hs x perfectintegers.h
1 fac :: Int -> Int
2 fac 0 =
3 fac n = n * fac(n -1)
```

The letter n for a number, to find the factorial of, goes before the equal sign with fac. After the equals I put n multiplied by the factorial of that number subtract one. To work out the factorial of a number that isn't negative, by being recursive, the function has itself in the definition with the number n subtract 1 in brackets.



```
*C:\Users\user\OneDrive - Univer:
File Edit Search View Encodir
actualgame.hs x TA game 2.hs
1 fac :: Int -> Int
2 fac n = product[1..n]
3
4 fac :: Int -> Int
5 fac 0 = 1
6 fac n = n * fac(n -1)
```

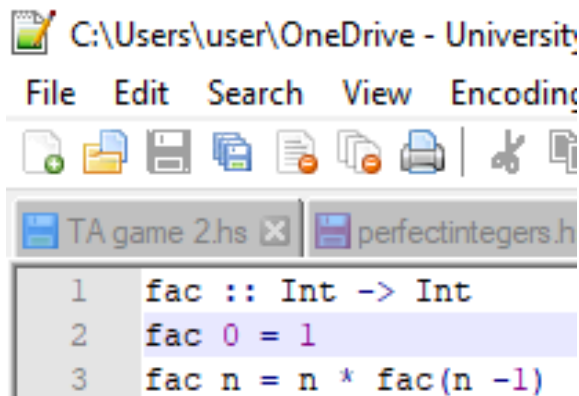
We also have the factorial function that's not recursive. It is non-recursive. Basically it works out the factorial of a number by calling the product function instead of itself. Every number from one to the number itself is multiplied together.

```

1 fac :: Int -> Int
2 fac n = product[1..n]
3
4 fac :: Int -> Int
5 fac n = fac n *(n -1)

```

If the recursive fac needs to call this function in its definition to work I have been thinking of having both of them in this file but have decided not to. Having both functions with the same name makes error messages appear when I run the code. I keep the one in this file.

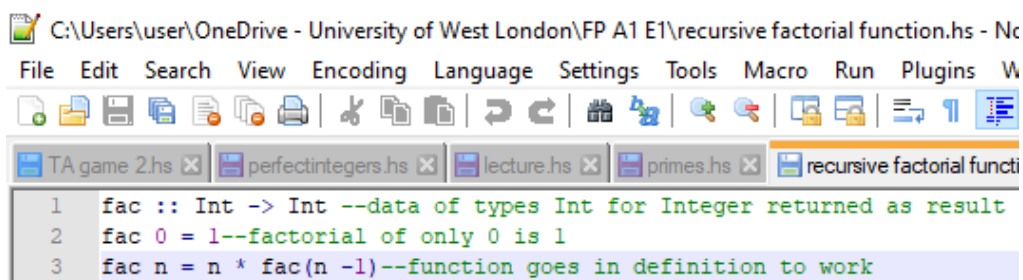


```

1 fac :: Int -> Int
2 fac 0 = 1
3 fac n = n * fac(n -1)

```

Like any number to the power of zero except zero itself, the factorial of zero equals one. I have put that above the definition. This function works only with positive numbers from one to n. Not negative. Zero does not have any numbers up to it to be multiplied together. It is the base case. The function does not work recursively below it such as finding or multiplying zero by the factorial of any negative number. Doing so on WinGHCi throws an overflow error message which basically means you have given a number which is below this limit. 1 is the factorial of 0 and recursion works above 0. Not below it.

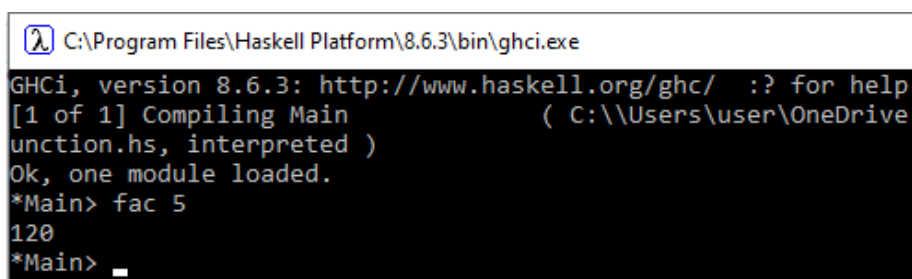


```

1 fac :: Int -> Int --data of types Int for Integer returned as result
2 fac 0 = 1--factorial of only 0 is 1
3 fac n = n * fac(n -1)--function goes in definition to work

```

I have gone through the code and checked it to see it is correct. I have added comments. I have saved it as a Haskell program file (.hs file extension) and now run it on the WinGHCi compiler.



```

C:\Program Files\Haskell Platform\8.6.3\bin\ghci.exe
GHCi, version 8.6.3: http://www.haskell.org/ghc/  :? for help
[1 of 1] Compiling Main             ( C:\Users\user\OneDrive
unction.hs, interpreted )
Ok, one module loaded.
*Main> fac 5
120
*Main>

```


On the WinGHCi I type fac and then 5. The Haskell code is fac n. I replace n with 5. It is now working. The recursive factorial of 5 or !5 is 120.

The factorial of 5 or !5 is worked out like this:

```
fac 5 = 5 * fac (5 - 1)
      = 5 * fac (4)
      = 5 * (4 * fac 3)
      = 5 * (4 * (3 * fac 2))
      = 5 * (4 * (3 * (2 * fac 1)))
      = 5 * (4 * (3 * (2 * (1 * fac 0))))
      = 5 * (4 * 3 * 2 * 1 * 1)
      = 5 * 4 * 3 * 2 * 1
      = 120
```

The function works by multiplying five by the factorial of the number before it four. Five takes away one in brackets so the factorial of four is worked out next. The factorial of the number is done by multiplying it by that of its previous number until the previous number is zero. This is the function working recursively by being it in its definition to find the factorial of a number and each of the numbers before it to zero. After the factorial of zero, which is one anyway, the numbers before the number, to find the factorial, all the way to one are multiplied together.

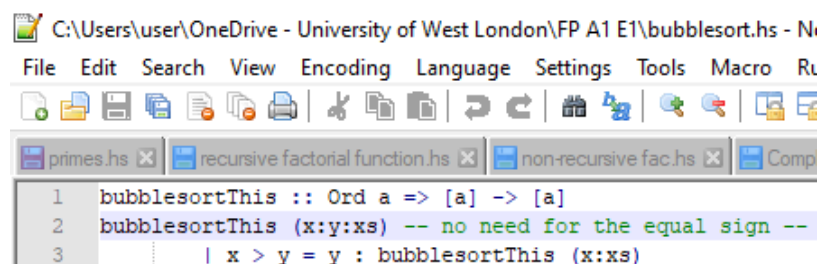
It is not recursive if it works out the factorial of four at the beginning by multiplying the numbers from one to four together and by five.

4. A recursive function bubble sort puts a list of numbers into order. I bubble sorted a list of numbers in my Algorithms and data types assignment so I know bubble sort is putting a list into order by taking pairs of numbers in the list and putting each pair into numerical order. I start defining it with data types including [a] since a list of numbers is returned. It works with itself in the definition. If it needs to be in its definition several times to put the numbers in order I will put it there.

The code starts with the data of type a's in square brackets. Type a is for any type of value. Any type of value a is in square brackets [a]. This is the list that's going to be sorted and the last a is the sorted list being returned. Each [a] is being put into order by this function so we need the "Ord a" at the beginning to do that by being a guard.

```
1 bubblesortThis :: Ord a => [a] -> [a]
```

I need to put two a's, for any type of value, rather than three since the first a is the list before it is sorted and the second is the list that is returned. I thought of putting three or more for the lists while they are being sorted but that's not necessary. I might as well have two because that makes the code run.

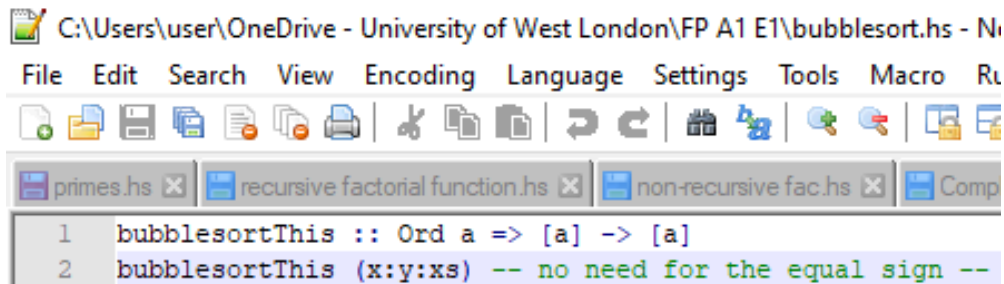


The screenshot shows a Haskell IDE window titled "C:\Users\user\OneDrive - University of West London\FP A1 E1\bubblesort.hs - N". The menu bar includes File, Edit, Search, View, Encoding, Language, Settings, Tools, Macro, and Run. The toolbar contains icons for file operations and compilation. The tab bar shows "primes.hs", "recursive factorial function.hs", "non-recursive fac.hs", and "Comp". The code editor displays the following Haskell code:

```
1 bubblesortThis :: Ord a => [a] -> [a]
2 bubblesortThis (x:y:xs) -- no need for the equal sign --
3   | x > y = y : bubblesortThis (x:xs)
```

I have put x, y and xs. The elements are x and y for the numbers and xs is the list being put into order. The pair of numbers in the list are x, y which are taken and swapped into numerical order. I am not putting square brackets [], that are empty. The bubble sort function does not need empty lists to sort the list of numbers.

Putting the pair of numbers x, y into numerical order is the smallest of the pair going before the biggest. Either x or y is smaller. This is done, for each pair of numbers, along the list until it is in order. If x is smaller than y the next pair of numbers is put in order otherwise y is put before x.

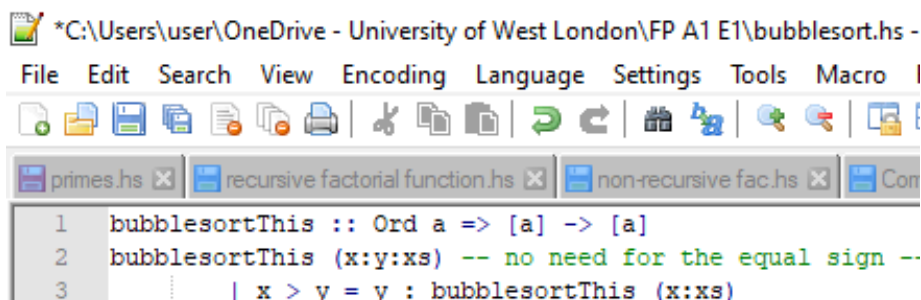


```

1 bubblesortThis :: Ord a => [a] -> [a]
2 bubblesortThis (x:y:xs) -- no need for the equal sign --

```

I start the function with the argument in parentheses having elements x,y and the list xs separated by colons. Each of these is called in the rest of the code to put each pair in the list xs in numerical order. They have to be separated by colons in parentheses to make the function put the list in order.

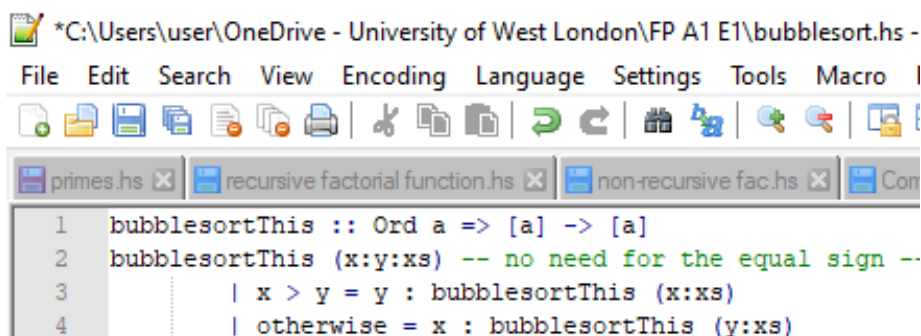


```

1 bubblesortThis :: Ord a => [a] -> [a]
2 bubblesortThis (x:y:xs) -- no need for the equal sign --
3     | x > y = y : bubblesortThis (x:xs)

```

I write the code for x being bigger than y which is y being moved before x in the pair in the list xs.



```

1 bubblesortThis :: Ord a => [a] -> [a]
2 bubblesortThis (x:y:xs) -- no need for the equal sign --
3     | x > y = y : bubblesortThis (x:xs)
4     | otherwise = x : bubblesortThis (y:xs)

```

If x is not bigger than y or the pair of numbers are the same, I need to put otherwise and then equals to put the next pair in order. The next pair to sort starts from y which becomes x and the number after it becomes y. The sorting is done again by the code which is recursive by having the function 'bubblesortThis' in these lines in its definition.

```
1 bubblesortThis :: Ord a => [a] -> [a]
2 bubblesortThis (x:y:xs) -- no need for the equal sign --
3 | x > y = y : bubblesortThis (x:xs)
4 | otherwise = x : bubblesortThis (y:xs)
5 bubblesortThis x = (x)
```

Before I run the code on WinGHCi I need to put the line of code to put x into order. This line is the name of the function with x, with no brackets, equalling x, in brackets, to bubble sort x. This is needed to nearly puts the list of numbers in order.

```
C:\Program Files\Haskell Platform\8.6.3\bin\ghci.exe
GHCi, version 8.6.3: http://www.haskell.org/ghc/  :? for help
[1 of 1] Compiling Main                ( C:\Users\user\OneDrive - University of West London\FP A1 E1\bubblesort.hs, interpreted )

C:\Users\user\OneDrive - University of West London\FP A1 E1\bubblesort.hs:3:44: warning: [-Wtabs]
    Tab character found here, and in two further locations.
    Please use spaces instead.
3 |         | x > y = y : bubblesortThis (x:xs)
   |                                     ^^^^^
Ok, one module loaded.
*Main> bubblesortThis [2,1,3,3,5,4,5,5,7,6,9,8]
[1,2,3,3,4,5,5,5,6,7,8,9]
*Main>
```

I manage to run it on WinGHCi by typing bubblesortThis [3,2,5,5,7,9,8,4,6,1,1,1]. The result is the numbers that have been put into numerical order by bubble sorting them. The code sorts the numbers correctly since I put the line bubblesortThis x = (x). Like the merge sort function, we have numbers x and y where the smallest of the two go first on the list.

```
C:\Program Files\Haskell Platform\8.6.3\bin\ghci.exe
C:\Users\user\OneDrive - University of West London\FP A1 E1\bubblesort.hs:3:44: warning: [-Wtabs]
    Tab character found here, and in two further locations.
    Please use spaces instead.
3 |         | x > y = y : bubblesortThis (x:xs)
   |                                     ^^^^^
Ok, one module loaded.
*Main> bubblesortThis [2,1,3,3,5,4,5,5,7,6,9,8]
[1,2,3,3,4,5,5,5,6,7,8,9]
*Main> bubblesortThis [3,2,5,5,7,9,8,4,6,1,1,1]
[2,3,5,5,7,8,4,6,1,1,1,9]
*Main>
```

However if try to bubble sort the list [3,2,5,5,7,9,8,4,6,1,1,1] I get the result [2,3,5,5,7,8,4,6,1,1,1,9]. The list of numbers is not completely in order. It is in order but not completely. This is because I haven't added code to make the function sort the numbers

recursively. The code I have written for the function runs. It depends which numbers you type to bubble sort. It puts some lists of numbers in order but not all of them like the ones I typed first.