```haskell
module Nextchar where
    import Data.Char
    nextchar :: Char -> Char
    nextchar c = chr ((ord c) + 1)

module Fact where
    fact :: Int -> Int
    fact 0 = 1
    fact x = x * fact(x - 1)

--Ejercicio 1:
module NumCbetw2 where
    import Data.Char
    numCbetw2 :: Char -> Char -> Int
    numCbetw2 x y= abs((ord y - ord x) - 1)


--Ejercicio 2:
module AddRange where
    addRange :: Int -> Int-> Int
    addRange x y = if x == y then x else
                   if x < y then x + (addRange(x + 1) y) else
                   addRange y x

--Ejercicio 3:
module Max where
    max' :: Int -> Int -> Int
    max' a b
        |a == b = b
        |a > b = a
        |b > a = b

--Ejercicio 4 - 5:
module Leapyear where
    leapyear :: Int -> Bool
    leapyear x
        |mod x 4 == 0 = if mod x 400 == 0 then True else
                        if mod x 100 == 0 then False else True
        |otherwise = False

    daysAmonth :: Int -> Int -> Int
    daysAmonth a b
        |(a == 2) && (leapyear b)  = 29
        |a == 1 || a == 3 || a == 5 || a == 7 || a == 8 || a == 10 || a == 12
= 31
        |a == 4 || a == 6 || a == 9 || a == 11 = 30
        |otherwise = 28

--Ejercicio 6:
module Remainder where
    remainder :: Int -> Int -> Int
    remainder a b
        |a < b = a
        |otherwise = remainder(a - b) b

--Ejercicio 7:
module SumaFact where

    fact :: Int -> Int
```

```haskell
    fact 0 = 1
    fact x = x * fact(x - 1)

    sumFact :: Int -> Int
    sumFact x
        |x == 0 = 0
        |otherwise = fact x + sumFact (x - 1)
```