ECS713: Functional Programming week 10 Monads

Part Wt Cha Double Part With Chapter Port Property Part William Part W

Aims

Assignment Project Exam Help

- To explain how the do notation can be used with general monads.
- To explain the link between it and bind.

Learning Objectives

• Understand how to use the do notation with a general monad https://tutorcs.com

WeChat: cstutorcs

Understand the systematic relation between it and bind.

do and lO

```
IO action (Input)
do
 text <- readFile "myfile.txt" Assignment Project Exam Help Ure computation
  let wordCount = length $ words text
  print "Word Count is "++(show wordCount)++"\n"
                                                              IO action (output)
```

```
do
  text <- readFile "myfile.txt" Assignment Project Exam Help
  let wordCount = length $ words text/tutorcs.com print "Word Count is "++(show
  print "Word Count is "++(show
                                     WeChat: cstutorcs wordCount)++"\n"
wordCount)++"\n"
```

Start at the bottom: last line is IO action

```
do
  text <- readFile "myfile.txt" Assignment Project Exam Help wordCount = length $ words text
  let wordCount = length $ words text/tutorcs.com
                                                     in
  print "Word Count is "++(show
                                                   print "Word Count is "++(show
                                     WeChat: cstutorcs
wordCount)++"\n"
                                                wordCount)++"\n"
```

Move up: next line is let

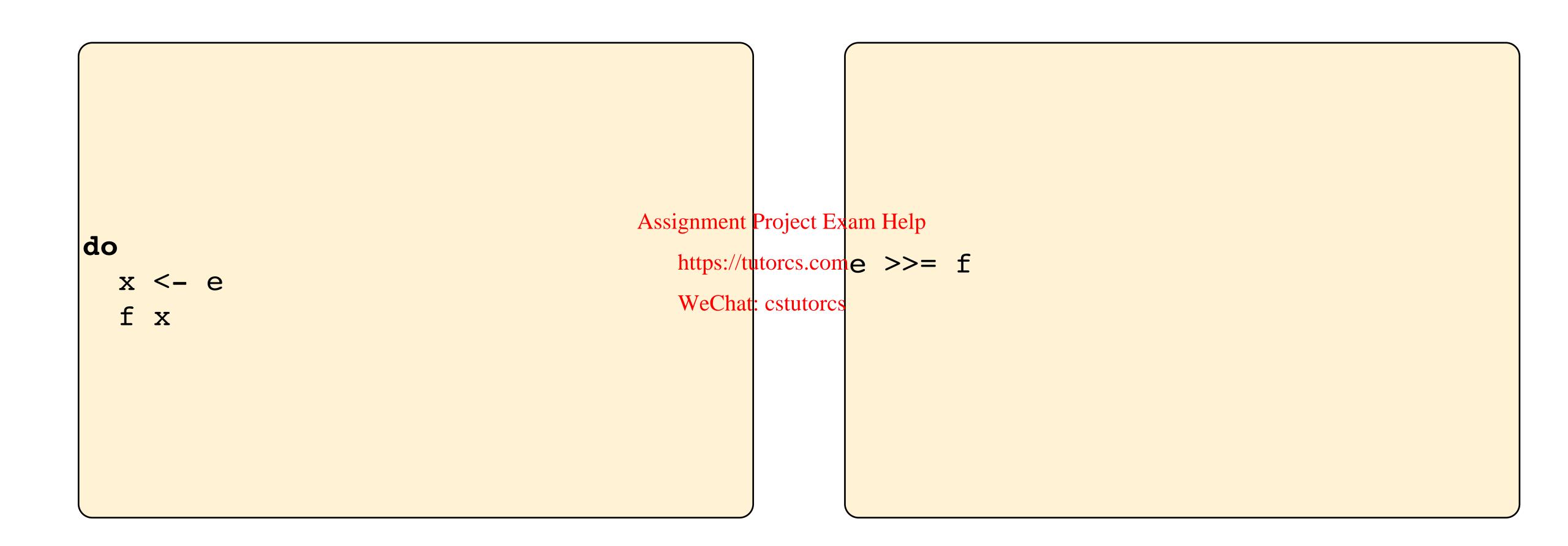
```
readFile "myfile.txt"
do
                                                   >>=
  text <- readFile "myfile.txt" Assignment Project Exam Helptext ->
  let wordCount = length $ words thext/tutorcs.com
                                                  let wordCount = length $ words text
  print "Word Count is "++(show
                                                     in
                                     WeChat: cstutorcs
wordCount)++"\n"
                                                  print "Word Count is "++(show
                                                wordCount)++"\n")
```

Move up: next line is bind

```
readFile "myfile.txt"
do
                                                                                                                                                                                                                                                                                                                                                                                                           >>=
               text <- readFile "myfile.txt" Assignment Project Exam Helptext ->
                                                                                                                                                                                                                                                                                                                                                                                                       let wordCount = length $ words text
                  let wordCount = length $ words the total company to the total company to the comp
                 print "Word Count is "++(show
                                                                                                                                                                                                                                                                                                                                                                                                                          in
                                                                                                                                                                                                                                                                                                    WeChat: cstutorcs
wordCount)++"\n"
                                                                                                                                                                                                                                                                                                                                                                                                          print "Word Count is "++(show
                                                                                                                                                                                                                                                                                                                                                                                         wordCount)++"\n")
```

Done: next line is do

translate to do



Translate right to left

Another example

```
tf <*> ta =
                                       Assignment Project Exame Help*> ta =
 do
                                          https://tutorcs.com tf >>= ( \f ->
 f <- tf
                                         WeChat: cstutorcs ta >>= (\a ->
  a <- ta
                                                         return (f a)))
  return (f a)
```

Sequential application as an example: RHS is as given in video 3

do notation and arbitrary monads

Assignment Project Exam Help

- The formal definition of the meaning of the do notation uses this transformation (that's why it's a flotation)
- It follows we can use it for arbitrary monads.

do notation and arbitrary monads (Either)

```
safeHead [] = Left "exception: head []"
safeHead (x:xs) = Right x
safeLog x | x<=0 = Left "exception: log of negative"
            otherwise = Right $ log x
foo list =
  do
                                  Assignment Project Exam Help
    x <- safeHead list
                                     https://tutorcs.com
    lg <- safeLog x
                                     WeChat: cstutorcs
    return lq
foo []
Left "exception: head []"
foo [2..4]
Right 0.6931471805599453
foo [-1..3]
Left "exception: log of negative"
```

do notation and arbitrary monads (Lists)

```
newMap f xs = do { x < - xs; return $ f x }
newMap (+1) [1..4]
[2,3,4,5]
                                       https://tutorcs.com
                                       WeChat: cstutorcs
do { c1 <- "abc"; c2 <- "xy"; return $ c1:c2:[] }
["ax", "ay", "bx", "by", "cx", "cy"]
```

```
do { c2 <- "xy"; c1 <- "abc"; return $ c1:c2:[] }
["ax","bx","cx","ay","by","cy"]
```

Summary

• The do notation can be expressed using bind instead.

https://tutorcs.com

WeChat: cstutorcs

- But the do notation can be used for arbitrary monads, not just IO
- And it can be much clearer than a functional expression with bind.