Word Count – part #1

- Ask for path
- Read file
- Count words
- Print length

Demo

```
import zio._
import zio.console._
import scala.io.Source
object wordCountv1 extends zio.App {
   def run(args: List[String]) =
    myAppLogic.exitCode

    val myAppLogic =
    for {
      _ <- putStrLn(
        "Hello! What path do you want to word count? please enter full path")
      fullPath <- getStrLn
    } yield fullPath

  def countWords(str: String):Int = ???
  def readFileAsString(path: String): String = ???
}
```

How to model this properly?

- UIO
- Task
- ZIO

How to read lines from a file ?

```
  import java.io.BufferedReader
  import java.io.FileInputStream
  import java.io.InputStreamReader

def readFileAsString(path: String): String = {
  var string = ""

  val fstream = new FileInputStream(path)
  val br = new BufferedReader(new InputStreamReader(fstream))
  var strLine: String = null
  while ({
    ({ strLine = br.readLine; strLine }) != null
  }) {
    string += strLine
  }

  string
}
```

DEMO

Where's the bug?

introducing bracket

```
import java.io.BufferedReader
import java.io.FileInputStream
import java.io.InputStreamReader
import zio._
import zio.console._
def readFileAsString(path: String): Task[String] = {
  var string = ""
  var strLine: String = null

  Task(new BufferedReader(new InputStreamReader(new FileInputStream(path))))
    .bracket(inputStream =>UIO(inputStream.close)) {
      br =>
        {
          while ({
            ({ strLine = br.readLine; strLine }) != null
          }) {
            string += strLine
          }
          Task(string)
        }
    }
}
```

Demo

» How can we know this worked ?

Zip Right

```
def readFileAsString(path: String): Task[String] = {
  var string = ""
  var strLine: String = null

  Task(new BufferedReader(new InputStreamReader(new FileInputStream(path))))
    .bracket(inputStream => UIO(println("closing")) *> UIO(inputStream.close)) {
      br =>
        {
          while ({
            ({ strLine = br.readLine; strLine }) != null
          }) {
            string += strLine
          }
          Task(string)
        }
    }
}
```

Demo

Full code

```
object wordCount extends zio.App {

def run(args: List[String]) =
  myAppLogic.exitCode

   def readFileAsString(path: String): Task[String] = {
  var string = ""
  var strLine: String = null

  Task(new BufferedReader(new InputStreamReader(new FileInputStream(path))))
    .bracket(inputStream => UIO(println("closing")) *> UIO(inputStream.close)) {
      br =>
        {
          while ({
            ({ strLine = br.readLine; strLine }) != null
          }) {
            string += strLine
          }
          Task(string)
        }
    }
}

  def countWords(str: String): UIO[Int] = UIO(str.split(" ").length)

  val myAppLogic =
    for {
      _ <- putStrLn(
        "Hello! What path do you want to word count? please enter full path")
      fullPath <- getStrLn
      contents <- readFileAsString(fullPath)
      count <- countWords(contents)
      _ <- putStrLn(count.toString)
    } yield count

}
```

Next step

» Word count on a full directory

```
def getFolderFiles(path: String): Task[List[String]] =
  ???


def countPerFile(path:String):Task[Int] = ???
```

```scala
def countPerFile(path: String) =
  for {
    contents <- readFileAsString(path)
    count <- countWords(contents)
    _ <- putStrLn(s"found ${count} words")
  } yield count

def countWords(str: String): UIO[Int] = UIO(str.split(" ").length)

def readFileAsString(path: String): Task[String] = {
  var string = ""
  var strLine: String = null

  Task(new BufferedReader(new InputStreamReader(new FileInputStream(path))))
    .bracket(inputStream => UIO(println("closing")) *> UIO(inputStream.close)) {
      br =>
        {
          while ({
            ({ strLine = br.readLine; strLine }) != null
          }) { // Print the content on the console
            string += strLine
          }
          Task(string)
        }
    }
}
```

And then

```
val myAppLogic =
  for {
    _ <- putStrLn(
      "Hello! What folder path do you want to word count? please enter full folder path")
    fullPath <- getStrLn
    files <- getFolderFiles(fullPath)
    count <- ZIO.collectAll(files.map(countPerFile(_)))
    _ <- putStrLn(s"found ${count.sum} in all files")
  } yield count.sum
```

Par combinators

Recap

- ZIO modeling using UIO, Task
- Bracket
- Zip Right
- collectAll, CollectAllParN combinators
- ZIO.effect

Testability via zio environments.

```
import zio.{Has, ZLayer}

type FileRepo = Has[FileRepo.Service]

object FileRepo {
  trait Service {
    def readFileAsString(path: String): Task[String]
  }
}
```

Count words again

```
def countWords(str: String): ZIO[FileRepo, Throwable, Int] =
  for {
    content <- ZIO.accessM[FileRepo](_.get.readFileAsString(str))
    count <- countWords(content)
  } yield count
```

Main app

```
def run(args: List[String]) =
    myAppLogic.provideSomeLayer(FileRepo.live ++ zio.console.Console.live).exitCode
```

FileRepo Live

```
val live: Layer[Nothing, FileRepo] = ZLayer.succeed(
  new Service {
    def readFileAsString(path: String): Task[String] = {
      var string = ""
      var strLine: String = null

      Task(
        new BufferedReader(
          new InputStreamReader(new FileInputStream(path))))
        .bracket(inputStream =>
          UIO(println("closing")) *> UIO(inputStream.close)) { br =>
          {
            while ({
              ({ strLine = br.readLine; strLine }) != null
            }) { // Print the content on the console
              string += strLine
            }
            Task(string)
          }
        }
    }
  }
)
```

What was the trouble for?
ZIO test

```
import zio._
import zio.test.Assertion._
import zio.test._

object WordCountSpec extends DefaultRunnableSpec {
  override def spec =
    suite("WordCountSpec")(
      testM("count words properly") {
        for {
          count <- wordCount.wordCountEnv.countWords("/tmp/twoWords")
        } yield assert(count)(equalTo(2))
      }
    ).provideSomeLayer(wordCount.wordCountEnv.FileRepo.live)
}
```

Test env

```
val test: Layer[Nothing, FileRepo] = ZLayer.succeed(new Service {
def readFileAsString(path: String): Task[String] = {
  Task(path match {
    case "/tmp/twoWords"   => "hello world"
    case "/tmp/threeWords" => "hello dear world"
    case _                 => "unknown"
  })
  }
 })

provideSomeLayer(wordCount.wordCountEnv.FileRepo.test)
```