Scala Program

1. W.A.P. that reads words from file. Use mutable map to count the words.

→
```
import scala.io.Source.
object wordCount {
def main(args: Array[String]){
if (args.length != 1){
System.err.println("error")
System.exit(1)
}
var filename = args(0)
val wordcount = scala.collection.mutable.Map[String,Int]()
for (line <- Source.fromFile(filename).getLines)
for (word <- line.split(" "))
~~for((k,v) <- wordcount)~~
~~printf("word %s occurs %d times\n", k,v)~~
~~}}~~
wordcount(word) = if (wordcount.contains(word)) wordcount(word)+1
println(wordcount)                                        else 1
for ((k,v) <- wordcount)
printf("word %s occurs %s times\n", k,v)
}}
```

OUTPUT

```
Map(RCB → 2, will → 1, the → 1, match → 1, today → 1)
word RCB occurs 2 times
word will occurs 1 times
word the occurs 1 times
word match occurs 1 times
word today occurs 1 times
```

10/4/23

paperism.
Date: 17/04/2023

2. write a function minmax (values: Array[Int]) that return a pair containing the smallest & largest val in the array.

```
import scala.io.StdIn
import Scala.collection.mutable.ArrayBuffer.
object MinMax {
    def main (args: Array[String]): Unit = {
        var numArray = new ArrayBuffer [Int]()
        println(" Enter no. of elements:")
        val n = scala.io.StdIn.readInt()
        println("Enter elements")
        for (i <- 1 to n)
                numArray+ = Scala.io.StdIn.readInt().
        println (numArray)
        val t= minmax (numArray)
        println("Max is ", t._1)
        println( "Min is ", t._2)
    }
    def minmax (numArray: ArrayBuffer [Int]): (Int, Int)={
        var min: Int = 9993
        var max: Int = (-999)3
        for (value <- numArray) {
            if (value> max)
                max = value.
            else if (value < min)
                min = value.
        }
        (max, min)
    }
}.
```

paperism
Date: / /

OUTPUT

```
Enter no. of Elements
5
Enter elements
2
4
6
3
1
Max is 6
Min is 1
```

3. [illegible]

paperism

Date: 24/04/2023

3. Write the menu driven prog. to implement quick sort algorithm using imperative & functional style.

→

```
object scalasort {
  def sort(a: List[Int]): List[Int] = {
    if (a.length < 2) a
    else {
      val pivot = a(a.length/2)
      sort(a.filter(_ < pivot)) :::
      ~~sort~~ (a.filter(_ == pivot)) :::
      sort(a.filter(_ > pivot))
    }
} }
  def sort1(a: Array[Int]) {
    def swap(l: Int, r: Int) {
      val t = a(l); a(l) = a(r); a(r) = t.
    }
    def sort2(l: Int, r: Int) {
      val pivot = a((l+r)/2)
      var i = l
      var j = r.
      while (i <= j) {
        while (a(i) < pivot) i += 1
        while (a(j) > pivot) j -= 1
        if (i <= j) {
          swap(i, j)
          i += 1
          j -= 1
        }
      }
      if (i < j) sort2(i, j)
      if (j < r) sort2(i, r)
    }
    if (a.length > 0)
      sort2(0, a.length - 1)
}
```

paperism

Date: / /

```
def main(args: Array[String]){
    val xs = List(6,2,8,5,1)
    println(xs)
    println(."Sorted list using functn style")
    println(.sort(xs))
    val xs1 = List(6,21,4,66,23)
    println(xs1)
    println("Sorted list using imperative style")
    println(sort1(xs))
  }
}
```

3.

OUTPUT

```
List(6,2,8,5,1)
sorted list using functn style
List(1,2,5,6,8)
List(6,21,4,66,23)
Sorted list using imperative style
List(4,6,21,23,66)
```

[illegible]

Experiment No. .... Date : 08/05/2023

Name of the Experiment : .... Page No : ....

Spark Programming

1. Word Count : Here the goal is to count how many times each word appears in a file & write out a list of words where word count is strictly greater than 4. Use the file log.txt accompanying file assignment to count the words. Save the wordcounts in text form in the "wordcountsDir" using the saveAsTextFile RDD method. Examine the contents of the above directory. & the contents of the files of the dictionary.

```
import org.apache.spark.SparkContext
import org.apache.spark.SparkConf
import org.apache.spark.rdd.RDD
object wordCount {
        def main(args: Array[String]) {
              val pathToFile = "log.txt"
              val conf = new SparkConf().setAppName("wordCount").
                                                  setMaster("local[*]")
              val sc = new SparkContext(conf)
              val wordsRdd = sc.textFile(pathToFile).flatMap(_.split("
              val wordCountInitRdd = wordsRdd.map(word => (word, 1)).
              val wordCountRdd = wordCountInitRdd.ReduceByKey
                                                  ((v1, v2) => v1 + v2).
              val highFreqWords = wordCountRdd.filter(x =>
                                                  x._2 > 4)
              highFreqWords.saveAsTextFile("wordCountsDir")
        }
```

[illegible] 15/05

NMIT

OUTPUT

```
(^~~~, 10)
(*type, 10)
(htable → table[i].element, 10)
(NULL;, 10)
(, 546)
(warning;; 10)
(from, 10)
('int', 10)
```

4. W.A.P. to illustrate the use of pattern matching scala for the foll. matching on case classes. Def. 2 case classes as follows:
Abstract class Notification
case class Email(sender: String, title: String, body: String) extends Notification

case class SMS(caller: String message: String) extends Notifica
Define a functn showNotification which takes as a para the abstract type Notification & Matches on the type of Notification i.e. it figures out whether its an Email or SMS. In case its an SMS return the String :- s"You got an SMS from $number; Message: $message".

```
abstract class Notification
case class SMS (mobile: String, msg: String) extends Notificatio
case class Email(emailAddr: String, subject: String, body: Str
                                  extends Notification
object temp {
   def showNotification (note : Notification): String = {note
     match {
          case Email(emailAddr, subject, _ ) => s"You got
            an email from $emailAddr, with subject: $su
          case SMS(number, message) => s"You got an SMS
            $number! message: $message".
     }
   }

   def main(args: Array [String]): Unit = {
      val someSMS = SMS("9946317834", "Did you Submit
                                          assignment")
      val some Email = Email ("shobha@gmail.com,
              "BDT Lab", "Intro to Big Data, Spark & R")
      println(showNotification(someSMS))
      println(showNotification(someEmail))
   }
}
```

paperism

Date: / /

OUTPUT

```
You got an SMS from 9946317834! message: Did you
Submit assignment
You got an Email from shobha@gmail.com, with Subject:
BDT LAB
```

[illegible]

paperism

Date: 15/05/2023

2. Tweet Mining: A dataset with the 8198 reduced tweets, reduced-tweets.json will be provided. The data contains reduced tweets as in the sample below:
{"id":"572692378957430785", "user":"Srkian nishu😊",
"text":"@always nidhi @Youtube no idnt understand bti loved of this movie is rocking",
"place":"Orissa", "country":"India"}
Write a functn to parse the tweets into an RDD & print the top 10 tweeters.

```
import org.apache.spark.{SparkContext, SparkConf}
import org.apache.spark.rdd._
object tweetmining {
    val conf = new SparkConf().setAppName("User Mining").
                                   setMaster("local[*]")
    val sc = new SparkContext(conf)
    var pathToFile = ""
    def main(args: Array[String]){
        if (args.length != 1 ){
            System.exit(1)
        }
        pathToFile = args(0)
        val tweets = sc.textFile(pathToFile).mapPartitions
           (TweetUtils.parseFromJson(_))
        val tweetsByUser = tweets.map(x => (x.user, x)).
                                        groupByKey()
        val numTweetsByUser = tweetByUser.map
                                   (x => (x._1, x._2.size)).
        val sortedUserNumByTweets = numTweetsByUser.
                           sortBy (_._2 ascending=false)
        sortedUserByNumTweets.take(10).foreach(println)
        val selectedTweets = sortedUserByNumTweets.take(10)
    }
}
```

paperism.

Date: / /

```
import com.google.gson._
object TweetUtils {
        case class Tweets (
                id : String, user String, userName : String,
                text: String, place: String, country: String, lang: Str
def parseFromJson (lines: Iterator [String]) : Iterator [Twee
        val gson = new Gson
        lines.map( line => gson.fromJson(line, classOf[Twe
}
}
}
```

OUTPUT

```
(#QuissyUpSoon, 958)
(Inco Mendes Askip [illegible], 185)
(#4Retime, 100)
(NIV, 58)
(williampriceking, 46)
([illegible]Follow Me MAEJOR. [illegible] 44)
(Ph Illhy McNasty, 43)
(KONORTS, 41)
(#[illegible] KINGTAECRAZY, 41)
(Sufia ao Ke, 36)
```

[illegible]
13/05

——×——×—×——×—