Bedömningsmall för tentamen EDAA45 Programmering, grundkurs

2018-01-05, 8:00-13:00

Hjälpmedel: Snabbreferenser för Scala och Java.

Generella bedömningsriktlinjer

Del A: Uppgift 1. Varje rad kan ge max 2p.

- Typerna och värdena ska skrivas som det står i lösningstabellen för full poäng. Dock får svaret skilja sig vad gäller i betydelselös användning av blanktecken och radbrytningar. Citationstecken runt strängar är ok men behövs inte. Om värdet är rätt, så när som på mindre detaljer i `toString`-representationen, till exempel `42.0` i stället för `42`, ges inget avdrag.
- Om rätt svar är ett kryss i kolumn 3 eller 4, men man kryssat för kompileringsfel när det ska vara exekveringsfel, eller tvärt om, eller om man kryssat i båda, ges -1 i avdrag.
- Om man svarat med fel typ men rätt värde ges -1 i avdrag.
- Ofullständig generisk typ, t.ex. `Option` istf. `Option[String]` ger -1 i avdrag.
- Om man svarat med rätt typ men fel värde ges -1 i avdrag.
- Om både värde och typ är fel ges -2 i avdrag.
- Om man svarat med typ/värde *och* kryss i kolumn 3 och/eller 4 på samma rad ges -2 i avdrag.

Del B:

- Totala poängen för en uppgift kan inte bli negativ; om alla avdrag för en uppgift blir mer än maxpoängen ges 0 poäng.
- Enkla misstag som är lätta att åtgärda ger -1 i avdrag eller -2 i avdrag, beroende på hur allvarligt felet är.
- Allvarliga misstag eller stora ofullständigheter ger upp till –*maxpoäng* i avdrag, beroende på hur mycket av koden som måste skrivas om för att det ska fungera och vara begripligt.

Del A. Uppgift 1. Totalt max 20 p

Efter init. av:	Vid kompile-ringsfel sätt kryss.	Vid exekve-ringsfel sätt kryss.	Ange statisk **typ** som kompilatorn härleder om ej kompilerings- eller körtidsfel.	Ange det **värde** som tilldelas vid exe-kvering, med samma format som vid utskrift av värdets `toString`, om ej kompilerings- eller körtidsfel.
`u1`	**X**			
`u2`			`Boolean`	`false`
`u3`			`Int`	`1005`
`u4`			`Unit`	`()`
`u5`	**X**			
`u6`			`Int`	`9`
`u7`			`Int`	`-3`
`u8`			`Char`	`!`
`u9`	**X**			
`u10`		**X**		

Rätta en rad i taget enligt nedan och sätt minuspoäng vid felaktigt svar i marginalen, t.ex. -1 eller -2. Räkna ut totala poängen och resultatet inringat längst ner på sidan direkt i skrivningshäftet. Varje rad kan ge max 2p.

- Typerna och värdena ska skrivas som det står i tabellen ovan för full poäng. Dock får svaret skilja sig vad gäller i betydelselös användning av blanktecken och radbrytningar. Om värdet är *nästan helt rätt*, så när som på obetydliga detaljer i `toString`-representationen, till exempel `10` i stället för `10.0` för ett värde av `Double`-typ, ges inget avdrag.
- Om rätt svar är ett kryss i någon av kolumnerna för kompilerings- eller exekveringsfel, men man kryssat för inkorrekt typ av fel, ges -1 i avdrag.
- Om man helgarderat med två kryss på samma rad ges -2 i avdrag.
- Om man svarat med fel typ men rätt värde ges -1 i avdrag.
- Om man svarat med rätt typ men fel värde ges -1 i avdrag.
- Om både värde och typ är fel ges -2 i avdrag.
- Om man svarat med typ/värde *och* kryss i någon av kolumnerna för kompilerings- eller exekveringsfel på samma rad ges -2 i avdrag.

Del B. Totalt max 80 p

Generella bedömningsriktlinjer för del B:

- Totala poängen för en uppgift kan inte bli negativ; om alla avdrag för en uppgift blir mer än maxpoängen ges 0 poäng.
- Olika formateringsvarianter som skiljer sig från mönsterlösning som ej påverkar funktionen och ej allvarligt försvårar läsningen, t.ex. extra (klammer)parentespar eller radbrytningar, ger inga avdrag.
- Felaktigt matchade (klammer)parentespar, felaktig indentering, eller glömda nödvändiga (klammer)parenteser ger -1 i avdrag.
- Felaktigt avskriven metodsignatur från specifikationen ger -1 i avdrag.
- Om nödvändigt semikolon saknas (t.ex. mellan flera satser på samma rad) ges -1 i avdrag.
- Felaktig användning av tom parameterlista (om den inte ska vara där eller om den saknas) ger -1 i avdrag.
- Mindre fel i likhet med föregående två punkter där det framgår tydligt vad som egentligen avses ger -1 i avdrag.
- Deklaration av `val` när det måste vara `var` för att koden ska fungera ger -2 i avdrag.
- Deklaration av `var` när det skulle kunna vara en `val` och fungera lika bra ger -1 i avdrag.
- Onödigt krångliga booleska uttryck, t.ex. `quit = if (uttryck) true else false` i stället för bara `quit = uttryck`, ger 1p avdrag.
- Onödiga typannoteringar ger inget avdrag.
- Tillägg av element i samling på fel sätt, t.ex. `+:` i stället för `:+` eller liknande, ger -1 i avdrag.
- Användning av `return` ger -2 i avdrag.
- Användning av metoden `length` på samling som ej är en sekvens, t.ex `Set` el. `Map`, ger -1 i avdrag.
- Enkla misstag som är lätta att åtgärda ger -1 eller -2 i avdrag, beroende på hur allvarligt felet är.
- Allvarliga misstag eller stora ofullständigheter ger från -3 upp till *–maxpoäng* i avdrag, beroende på hur mycket av koden som måste skrivas om för att det ska fungera.
- Om precis samma typ av fel förekommer inom *samma* deluppgift (implementerad medlem) mer än en gång kan du välja att bara göra avdrag en gång vid första förekomsten, *om* detta ger en rimligare totalpoäng för deluppgiften. Om du bedömer att detta är rimligt, skriv då *– Samma fel* i kanten för att indikera att avdraget redan är gjort. I vissa speciella fall kan även avdrag för samma fel göras över flera implementerade medlemmar eller till och med över en hel uppgift eller för hela skrivningen, under förutsättning att felet är trivialt, t.ex. för systematiska mindre syntaxfel så som onödiga semikolon vid radslut, konsekvent glömda `} ) then do` eller upprepad användning av `return`.
- Dokumentationskommentarer, paketdeklarationer och importsatser i given kod behöver ej upprepas i lösningen. Om ytterligare importsatser behövs men helt saknas ska -1 i avdrag ges, men smärre felaktigheter i själva sökvägen ger inga avdrag. Om en fullständig sökväg anges direkt på plats (i stället för import), ges inget avdrag om sökvägen har smärre felaktigheter. Dock är det viktigt att distinktionen mellan `collection.mutable` och `collection.immutable` blir rätt och sådana felaktigheter ger -2 i avdrag.

Endast delarna markerade med `// Xp` ingår i bedömningen, där `X` är maximala poängen för respektive del.

Uppgift 2. `FromFile`, 8p.

```
1  object FromFile:
2    import allocation.*
3
4    def getLines(file: String): Vector[String] =
5      scala.io.Source.fromFile(file).getLines.toVector
6
7    def loadPapers(file: String): Vector[Paper] = // 8p
8      for line <- getLines(file) yield                    //2p
9        val xs = line.split(';')                          //1p
10       val authors = xs(0).split(',').toVector           //3p
11       val title = xs(1)                                 //1p
12       Paper(title, authors)                             //1p
13
14   def loadConflicts(file: String): Map[Reviewer, Set[Int]] =
15     getLines(file).map { line =>
16       val xs = line.split(';').toVector
17       val reviewer = xs(0)
18       val conflicts =
19         if xs.length > 1 then xs(1).split(',').map(_.toInt).toSet
20         else Set[Int]()
21       reviewer -> conflicts
22     }.toMap
```

- Avsaknad av kontroll av längden på rad 20 ger inget avdrag då ”inga indatafiler är tomma”.

Uppgift 3. `Multimap`, 15p.

```
case class Multimap(toMap: Map[String, Set[Int]] = Map()):
  def get(key: String): Set[Int] = toMap.getOrElse(key, Set()) // 4p

  def add(kv: (String, Int)): Multimap = // 6p
    copy(toMap.updated(kv._1, get(kv._1) + kv._2))

  def count(value: Int): Int = toMap.count(_._2.contains(value)) // 5p
```

Uppgift 4. `allocation`, 38p.

```
object allocation:
  case class Paper(title: String, authors: Vector[String])

  type Reviewer = String
  type PaperIndex = Int

  class Allocator(val papers: Vector[Paper], val conflicts: Multimap):
    private var allocations = Multimap()

    def currentAllocation: Map[Reviewer, Set[PaperIndex]] = allocations.toMap

    def reviewers: Set[Reviewer] = conflicts.toMap.keySet

    def papersOfReviewer(r: Reviewer): Set[PaperIndex] = allocations.get(r)

    def nbrOfReviewers(pi: PaperIndex): Int = allocations.count(pi) // 3p
                                      // eller: reviewersOfPaper(pi).size

    def reviewersOfPaper(pi: PaperIndex): Set[Reviewer] = // 7p
      (for (r,pis) <- currentAllocation if pis(pi) yield r).toSet
      //el. tex:
      //currentAllocation.collect{ case (r, pis) if pis(pi) => r }.toSet

    def isBadAllocation(reviewer: Reviewer, pi: PaperIndex): Boolean = // 8p
      papers(pi).authors.contains(reviewer) ||      // 3 delpoäng
        papersOfReviewer(reviewer).contains(pi) ||  // 2 delpoäng
          conflicts.get(reviewer).contains(pi)      // 3 delpoäng

    def allocReviewerToPaper(r: Reviewer, pi: PaperIndex): Unit =
      allocations = allocations.add(r, pi)

    def indexOfNextPaper(n: Int): PaperIndex = // 5p
      papers.indices.find(pi => nbrOfReviewers(pi) < n).getOrElse(-1)

    def findNextReviewer(pi: PaperIndex): Option[Reviewer] = // 15p
      if papers.indices.contains(pi) then
        val sorted = reviewers.toVector.sortBy(r => papersOfReviewer(r).size)
        val dropped = sorted.dropWhile(r => isBadAllocation(r, pi))
        dropped.headOption
      else None

    def allocate(n: Int): Unit =
      val pi: PaperIndex = indexOfNextPaper(n)
      val rOpt: Option[Reviewer] = findNextReviewer(pi)
      if pi >= 0 && rOpt.nonEmpty then
        allocReviewerToPaper(rOpt.get, pi)
        allocate(n)
```

Uppgift 5. `ReviewMatrix`, 14p.

```
enum Grade:
  case StrongReject, Reject, Accept, StrongAccept

  def isNegative: Boolean = Set(StrongReject, Reject).contains(this)

  def score: Int = Map(0 -> -2, 1 -> -1, 2 -> 1, 3 -> 2).apply(ordinal)

class ReviewMatrix(val nbrOfPapers: Int, val nbrOfReviewers: Int):
  private val matrix: Array[Array[Option[Grade]]] =
    Array.fill(nbrOfPapers, nbrOfReviewers)(None)

  def apply(paperIndex: Int, reviewIndex: Int): Option[Grade] =
    matrix(paperIndex)(reviewIndex)

  def update(paperIndex: Int, reviewIndex: Int, grade: Grade): Unit =
    matrix(paperIndex)(reviewIndex) = Some(grade)

  def averageGrade(paperIndex: Int): Option[Double] = // 6p
    val scores: Seq[Int] = matrix(paperIndex).collect{ case Some(g) => g.score }
    if scores.nonEmpty then Some(scores.sum.toDouble / scores.length) else None
    /*
    //eller motsvarande imperativa lösning:
    var sum = 0.0
    var n = 0
    for gradeOpt <- matrix(paperIndex) do
      if gradeOpt.nonEmpty then
        sum += gradeOpt.get.score
        n += 1
    if n > 0 then Some(sum / n) else None
     */

  def isClearAccept(paperIndex: Int): Boolean = // 6p
    val gs: Seq[Grade] = matrix(paperIndex).collect{ case Some(g) => g }
    gs.length > 0 && !gs.exists(_.isNegative)
    /*
    //eller motsvarande imperativa lösning:
    var foundNegativeGrade = false
    var i = 0
    while !foundNegativeGrade && i < nbrOfReviewers do
      foundNegativeGrade = apply(paperIndex, i).map(_.isNegative).getOrElse(false)
      i += 1
    i > 0 && !foundNegativeGrade
    */
```

Uppgift 6. `Select`, 3p.

```
@main def Select =
    val reviewerIndex: Map[String, Int] =
      FromFile.loadConflicts("conflicts.txt").keySet.toVector.zipWithIndex.toMap

    val papers = FromFile.loadPapers("papers.txt")

    val rm = ReviewMatrix(papers.length, reviewerIndex.size)

    for line <- FromFile.getLines("reviews.txt") do
      val xs = line.split(';')
      if xs(0).nonEmpty && xs(1).nonEmpty then
        val ri = reviewerIndex(xs(0))
        val reviews = xs(1).split(',').map(_.split('='))
        for Array(pi, g) <- reviews do rm(pi.toInt, ri) = Grade.valueOf(g)

    def selectPaperIndices(p: Int => Boolean): Seq[Int] = // 2p
      papers.indices.filter(p)

    val clearAccepts: Seq[Int] = selectPaperIndices(pi => rm.isClearAccept(pi))

    println("Clear accept: " + clearAccepts.mkString(","))

    val borderline: Seq[Int] = // 5p
      selectPaperIndices(pi =>                                  //1p
        !rm.isClearAccept(pi) &&                                //1p
          rm.averageGrade(pi).map(_ > -0.4).getOrElse(false)  //3p
      )
      /* eller t.ex.
      selectPaperIndicies(pi =>
        !rm.isClearAccept(pi) &&
          rm.averageGrade(pi).getOrElse(Double.MinValue) > -0,4
      )
      */

    println("Borderline:   " + borderline.mkString(","))

    val clearReject: Seq[Int] = papers.indices diff clearAccepts diff borderline

    println("Clear reject: " + clearReject.mkString(","))
```