Bedömningsmall för kontrollskrivning EDAA45 Programmering, grundkurs

2023-10-25, 14:00-19:00

Hjälpmedel: Snabbreferenser för Scala och Java.

Instruktioner för kamraträttning

- Läs igenom hela denna bedömningsmall innan bedömningen påbörjas. Sätt dig in i mönsterlösningens angreppssätt och funktion, samt hur poängsättningen ska ske.
- För varje del: läs noga lösningen du ska bedöma och sätt dig in i det specifika angreppssättet.
- Ge maxpoäng om koden är korrekt, läsbar och lika bra som mönsterlösningen även om den är annorlunda. Det finns många olika sätt att lösa varje del, varav många är lika bra som mönsterlösningen. En lösning kan vara lika bra som mönsterlösningen, även om den är längre eller kortare eller har ett annat angreppssätt.
- Ge noll poäng om lösningen saknas, är oläsbar, obegriplig eller helt fel.
- Markering av poängavdrag (negativa) och uppgiftspoäng (inringad, positiv):
  - Markeringar ska göras med en penna med **avvikande färg**, gärna *röd*.
  - *Stryk under* de delar av en rad som är felaktiga eller bara delvis korrekta.
  - Om något saknas, *markera med en pil* var det som saknas borde finnas och skriv en kommentar om vad som saknas.
  - Markera poängavdrag i *högerkanten* med *minuspoäng* enligt bedömningsriktlinjerna i detta häfte.
  - När du markerat relevanta avdrag för en hel uppgift, beräkna och skriv total uppgiftspoäng *inringad* i en cirkel, så att det går lätt att skilja uppgiftspoäng från poängavdrag. Uppgiftspoängen ska inte vara negativ utan ligga i intervallet [*0, Maxpoäng*].
  - Om det efter studier av de markerade poängavdrag du gjort i skrivningen och dessa riktlinjer, inte är uppenbart varför du ej givit maxpoäng skriv då en kort förklaring i kanten.
  - Om en lösning är oläslig eller helt obegriplig, skriv *"oläsligt"* eller *"obegripligt"* i kanten.
- Avsluta med att göra en *kvalitativ helhetsbedömning* av hela skrivningen och justera poäng för vissa delar om du tycker att poängsumman för hela uppgiften inte överensstämmer med din helhetsbedömning. Om du ändrar i din poängsättning, stryk över de gamla poängen med ett kryss och skriv nya poäng bredvid.
- Summera poängen och fyll i summan på omslagets framsida.
- Ange era identitetsnummer i fälten för rättare på omslagets framsida.
- Kamraträttningar med invändningar kommer att bedömas igen i efterhand av lärare. Även de kamratbedömningar som godkänns utan invändningar kommer att stickprovsbedömas i efterhand av lärare.

Del A. Tolka uttryck. Totalt max 10 p.

Efter init. av:	Vid kompileringsfel sätt kryss.	Vid exekveringsfel sätt kryss.	Ange statisk **typ** som kompilatorn härleder om ej kompilerings- eller körtidsfel.	Ange det **värde** som tilldelas vid exekvering, med samma format som vid utskrift av värdets `toString`, om ej kompilerings- eller körtidsfel.
`u1`		**X**		
`u2`	**X**			
`u3`			`Boolean`	`true`
`u4`			`Vector[Int]`	`Vector(4)`
`u5`			`Vector[Int]`	`Vector(0,1)`

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 ges inget avdrag, till exempel `10` i stället för `10.0` för ett värde av `Double`-typ, eller saknade citationsteckenpar `"..."` runt en sträng.
- Om rätt svar är ett kryss i någon av kolumnerna för kompilerings- eller exekveringsfel, men man kryssat för inkorrekt typ av fel, ges -1p i avdrag.
- Om man helgarderat med två kryss på samma rad ges -2p i avdrag.
- Om man svarat med fel typ men rätt värde ges -1p i avdrag.
- Om man svarat med rätt typ men fel värde ges -1p i avdrag.
- Om både värde och typ är fel ges -2p i avdrag.
- Om man svarat med typ/värde *och* kryss i någon av kolumnerna för kompilerings- eller exekveringsfel på samma rad ges -2p i avdrag.

Del B. Skriva kod. Totalt max 40 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 −1p i avdrag.
- Felaktigt avskriven metodsignatur från specifikationen ger −1p i avdrag.
- Om nödvändigt semikolon saknas (t.ex. mellan flera satser på samma rad) ges −1p i avdrag.
- Felaktig användning av tom parameterlista (om den inte ska vara där eller om den saknas) ger −1p i avdrag.
- Mindre fel i likhet med föregående två punkter där det framgår tydligt vad som egentligen avses ger −1p i avdrag.
- Deklaration av `val` när det måste vara `var` för att koden ska fungera ger −2p i avdrag.
- Deklaration av `var` när det skulle kunna vara en `val` och fungera lika bra ger −1p i avdrag.
- Onödigt krångliga booleska uttryck, t.ex. `quit = if uttryck then true else false` i stället för bara `quit = uttryck`, ger −1p i avdrag.
- Onödiga typannoteringar ger inget avdrag.
- Tillägg av element i samling på fel sätt, t.ex. `+:` i stället för `:+` eller liknande, ger −1p i avdrag.
- Användning av `return` ger −2p i avdrag.
- Användning av metoden `length` på samling som ej är en sekvens, t.ex `Set` el. `Map`, ger −1p i avdrag.
- Enkla misstag som är lätta att åtgärda ger −1 eller −2p 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.
  Ett enda avdrag för *många* förekomster av samma fel kan göras för en mängd medlemmar, en hel uppgift eller till och med för hela skrivningen, **om felet är trivialt**, till exempel vid systematiska mindre syntaxfel så som flera onödiga semikolon vid radslut, många glömda slutmarkeringar: `}` `)` eller `then` eller `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 −1p 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 −2p i avdrag.

Lösningar och specifika bedömningsriktlinjer del B

- Mönsterlösningen nedan är bara ett av flera möjliga sätt att lösa uppgiften som är lika bra. Du ska vid bedömningen avgöra om den inlämnade lösningen är likvärdig mönsterlösningen eller om avdrag ska göras.
- Endast de poängmarkerade delarna t.ex. `//2p` ingår i bedömningen.
- Maxpoängen färdelas jämt över rader och (del)uttryck som ingår i nedan lösning, om inget annat anges i efterföljande punkter.
- Om motsvarande funktionalitet **helt saknas** eller det finns **stort fel** i motsvarande funktionalitet så dras motsvarande delpoäng, annars om det är ett **litet fel** så dras lämplig andel av delpoängen.
- Raden `package solution` ska ej ingå i studentens lösning.

Uppgift B1. Moduloräknare. Totalt max 15 p.

```
package solution

//Totalt 15p
class ModuloCounter(val modulo: Int): // 2p
  require(modulo > 0, s"modulo=$modulo, must be > 0") // 2p
  private var counter: Int = 0 // 3p

  // metoden next: totalt 8p
  def next(): Int = // 2p
    val result = counter // 1p
    counter = (counter + 1) % modulo // 4p
    result // 1p
```

- Klasshuvud
  - Glömd `val` ger −1p i avdrag
  - Glömd typ `Int` ger −1p i avdrag
  - Om `case` framför klass ges −1p i avdrag
- `require`
  - Glömd eller helt felaktig `require` ger −2p i avdrag.
  - Fel i utskriftssträng ger −1p i avdrag.
  - Fel i villkor ger −1p i avdrag.
- `counter`
  - Glömd `private` framför `var` ger −2p i avdrag.
  - Utelämnad typannotering ger inget avdrag.
  - Glömd initialisering ger −1p i avdrag.
- `next`
  - Glömd `()` ger −1p i avdrag
  - Glömd explicit returtyp ger −1p i avdrag (eftersom detta var ett krav)
  - Glömd spara undan och returnera värdet av `counter` före uppdatering max −2p i avdrag
  - Felaktig uppdatering av `counter` ger max −4p i avdrag beroende på hur allvarligt felet är. Notera: Det finns andra lösningar, t.ex. med if-uttryck som kan fungera lika bra som användning av `%`-operatorn; exempelvis ger denna uppdatering inget avdrag:
    `counter = if counter < modulo - 1 then counter + 1 else 0`

Uppgift B2. Registrering och gruppindelning. Totalt max 25 p.

```
package solution

extension (d: Double) def toRoundedPercent: Int = (d * 100).round.toInt //tot 3p

enum Experience { case None, Low, Medium, High }

type Lines = Vector[String]

object Lines:
  def loadLines(path: String): Lines =
    val source = scala.io.Source.fromFile(path)
    try source.getLines().toVector finally source.close()

  extension (lines: Lines) def saveToFile(path: String): Unit =
      val pw = java.io.PrintWriter(java.io.File(path), "UTF-8")
      try pw.write(lines.mkString("","\n","\n")) finally pw.close()

case class Student(name: (String, String), id: String, pre: Experience)

object Student:
  def fromString(s: String, delim: Char = ','): Student =
    val parts = s.split(delim)
    require(parts.length == 4, s"must be exactly 4 elems separated by $delim\n$s")
    Student(parts(0) -> parts(1), parts(2), Experience.fromOrdinal(parts(3).toInt))

type Students = Vector[Student]

object Students:
  def fromLines(lines: Lines, dropHeading: Boolean = true): Students = //tot 6p
    val bodyLines = lines.drop(if dropHeading then 1 else 0)  //3p
    for line <- bodyLines yield Student.fromString(line)  //3p
    // eller t.ex.: bodyLines.map(line => Student.fromString(line))

  def registerExperience(students: Students): Vector[(Experience, Int)] = //tot 7p
    val xs = Experience.values.toVector  //2p
    for level <- xs yield level -> students.count(_.pre == level)  //5p
    // eller t.ex.:
    // Experience.values.toVector.map(x => x -> students.count(_.pre == x))

  def allocate(nbrGroups: Int, students: Students): Vector[(Int, Student)] =//tot 9p
    val mc = ModuloCounter(nbrGroups)  //1p
    val sorted = students.sortBy(_.pre.ordinal)  //2p
    val grouped =  for s <- sorted yield (mc.next() + 1) -> s  //4p
                // sorted.map(s => (mc.next() + 1) -> s)
    grouped.sortBy(_._1)  //2p

@main def createGroups(inPath: String, outPath: String, groupSize: Int) =
```

```
48  import Lines.*, Students.*
49  val students = fromLines(loadLines(inPath))
50  println(s"Number of students in $inPath: ${students.length}")
51  println("Experience:")
52  for (x, n) <- registerExperience(students) do
53    val percent = (n.toDouble / students.length).toRoundedPercent
54    println(s"$x:${" " * (8 -x.toString.length)}$n, $percent%")
55  val output: Vector[String] =
56    "grupp,förnamn,efternamn,id,förkunskaper" +: (
57      for (g, s) <- allocate(groupSize, students)
58      yield s"$g,${s.name._1},${s.name._2},${s.id},${s.pre.ordinal}"
59    )
60  output.saveToFile(outPath)
61  println(s"Groups saved to $outPath")
```

- Metoden `toRoundedPercent`
  - glömt `* 100` ger −1p i avdrag
  - glömt `.round` ger −1p i avdrag
  - glömt `.toInt` ger −1p i avdrag
  - avkortning med heltalsdivision istf. avrundning ger −2p i avdrag
- Metoden `fromLines`
  - Saknad eller felaktig funktionalitet att skippa rubrikrad beroende på parameter ger upp till −3p i avdrag.
  - Saknad eller felaktig omvandling från strängvektor till studentvektor ger upp till −3p i avdrag
- Metoden `registerExperience`
  - Saknad eller felaktig användning av det givna tipset om `Experience.values.toVector` ger up till −2p i avdrag
  - Saknad eller felaktigt skapande av vektor med par av förkunskapsnivå och antal kan ge upp till −5p i avdrag beroende på hur mycket som är fel/saknas.
- Metoden `allocate`
  - Glömt eller felaktigt skapande av moduloräknare ger −1p i avdrag.
  - Glömd eller felaktig sortering av studenter ger −2p i avdrag.
  - Felaktig algoritm för skapande av grupper ur studenter genom parande med utdata från `next()` ger upp till −4p i avdrag, beroende på hur stort felet är.
  - Avsaknad av resultat sorterat på gruppnummer ger −1p i avdrag.