Bedömningsmall för kontrollskrivning EDAA45 Programmering, grundkurs

2022-10-26, 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`			`Boolean`	`false`
`u2`			`Int`	`1`
`u3`		**X**		
`u4`			`Option[String]`	`None`
`u5`	**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 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. Spelet ”sänka skepp”. 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) 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 -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. 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 -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.

```
package solution

case class Pos(row: Int, col: Int):
  override def toString = s"${(row + 'A').toChar}$col"

object Pos:
  def random(rowSize: Int = Sea.Size, colSize: Int = Sea.Size): Pos = //2p
    import scala.util.Random.nextInt
    Pos(nextInt(rowSize), nextInt(colSize))
```

- Metoden `random`
  - konstruktion med eller utan `new` är värt 1p
  - dragning av slumptal är värt 1p
- Predikatet `isOverlap`
  - beräkning av överlappet t.ex. med `intersect` eller `diff` är värt 2p
  - kontroll av om överlappet ej är tomt t.ex. med `nonEmpty` eller motsvarande är värt 1p
- Metoden `hitBlock`
  - konstruktion av nytt skepp, tex via anrop av `copy`, motsvarar 1p
  - uppdatering av rätt index i `blockIsHit` motsvarar 2p
  - uttrycket för uppdaterat par motsvarar 2p

```
package solution

case class Ship(blockIsHit: Vector[(Pos, Boolean)], isHorizontal: Boolean):
  val isSunk: Boolean = blockIsHit.forall(pair => pair._2)
  val blocks: Vector[Pos] = blockIsHit.map(_._1)

  def isOverlap(other: Ship): Boolean = // 3p
    blocks.intersect(other.blocks).nonEmpty

  def hitBlock(i: Int): Ship = // 5p
    copy(blockIsHit.updated(i, blockIsHit(i)._1 -> true))

```

```
  def isHitAt(p: Pos): Boolean = // 4p
    val ix = blocks.indexOf(p)
    ix >= 0 && blockIsHit(ix)._2

object Ship:
  def create(length: Int, at: Pos, isHorizontal: Boolean): Ship =
    val positions: Vector[Pos] = // 4p
      Vector.tabulate(length)(i =>
        if isHorizontal
        then Pos(at.row, at.col + i)
        else Pos(at.row + i, at.col))
    Ship(positions.map(_ -> false), isHorizontal)

  def random(length: Int): Ship =
    val isHorizontal = scala.util.Random.nextBoolean()
    val at: Pos =  if isHorizontal
      then Pos.random(colSize = Sea.Size - length)
      else Pos.random(rowSize = Sea.Size - length)
    create(length, at, isHorizontal)
```

- Predikatet `isHitAt`
  - linjärsökning efter rätt punkt i `blocks` är värt 1p
  - kontroll av om punkten ej ingår motsvarar 1p
  - kontroll av om block är träffat motsvarar 2p
- Värdet `positions`
  - generering av index i itereringen, t.ex. med `tabulate` eller `for` motsvarar 1p
  - kontroll av `isHorizontal` motsvarar 1p
  - konstruktion av ny position med korrekta värden motsvarar 2p

```
package solution

enum Damage { case Miss, Hit, Sunk }

object Sea:
  val Size = 10
  val ShipLengths = List(5, 4, 3, 3, 2)

  def create(): Sea =
    var ships = Vector(Ship.random(ShipLengths.head))
    for len <- ShipLengths.drop(1) do
      var candidate = Ship.random(len)
      while ships.exists(_.isOverlap(candidate)) do candidate = Ship.random(len)
      ships = ships :+ candidate
    new Sea(ships.toArray)

```

```
17 class Sea(val ships: Array[Ship]):
18   var misses = Vector.empty[Pos]
19
20   def isAllSunk: Boolean = ships.forall(_.isSunk) // 1p
21
22   def firedAt(p: Pos): Damage  = // 12p
23     val si =  ships.indexWhere(_.blocks.contains(p))
24     if si < 0 then
25       misses :+= p
26       Damage.Miss
27     else
28       val bi = ships(si).blocks.indexOf(p)
29       ships(si) = ships(si).hitBlock(bi)
30       if ships(si).isSunk then Damage.Sunk else Damage.Hit
31
32   def show(p: Pos, isEnemy: Boolean): Char =
33     ships.indexWhere(_.blocks.contains(p)) match
34       case i if i < 0 => if misses.contains(p) then 'M' else ' '
35       case i if ships(i).isSunk => 'S'
36       case i if ships(i).isHitAt(p) => 'H'
37       case i if isEnemy => ' '
38       case i if ships(i).blocks.indexOf(p) == 0 =>
39         if ships(i).isHorizontal then '>' else 'V'
40       case i => if ships(i).isHorizontal then '-' else '|'
```

- Predikatet `isAllSunk`
  - uttrycket ska vara helt korrekt för 1p (eftersom motsvarande lösning finns i given kod för `isSunk`)
- Metoden `firedAt`
  - linjärsökning efter skeppets index, t.ex. med `indexWhere` och `contains` motsvarar 2p
  - kontroll av om punkten ingår i något skepp motsvarar 2p
  - tillägg i misses motsvarar 2p
  - linjärsökning efter blockets index, t.ex. med `indexOf` motsvarar 2p
  - uppdatering av skepp motsvarar 2p
  - kontroll av om skeppet är sänkt och motsvarande skaderapport motsvarar 2p
  - varje möjlighet till exekveringsfel för indexering utanför gräns ger -2p i avdrag

```scala
package solution

object SeaBattle:
  val (human, computer) = (Sea.create(), Sea.create())

  def printWorld(): Unit =
    def colHead: String = Seq.tabulate(Sea.Size)(c => s"$c ").mkString
    def row(s: Sea, r: Int, isEnemy: Boolean = false): String =
      (for c <- 0 until Sea.Size yield s"${s.show(Pos(r,c), isEnemy)} ").mkString
    println(s"Human${" " * 19}Computer\n  $colHead    $colHead")
    for r <- 0 until Sea.Size do
      val s = ('A' + r).toChar.toString
      println(s"$s ${row(human, r)}  $s ${row(computer, r, isEnemy = true)}")

  def humanMove(): Unit =
    scala.io.StdIn.readLine("Fire at pos (e.g. A0): ") match
      case "exit"  => println("Goodbye!"); sys.exit() // avbryter exekveringen
      case "cheat" => println(computer.ships.mkString(",")); humanMove()
      case input if input.length == 2 =>
        val p = Pos(input(0).toUpper - 'A', input(1) - '0')
        if p.row >= 0 && p.row < Sea.Size && p.col >= 0 && p.col < Sea.Size then
          println(s"You fired at $p was ${computer.firedAt(p)}")
        else
          println(s"Illegal coordinates: $input")
          humanMove()
      case input =>
        println(s"Illegal input: $input")
        humanMove()

  def computerMove(): Unit = // 9p
    var p = Pos.random()
    while human.misses.contains(p) || human.ships.exists(_.isHitAt(p))
    do p = Pos.random()
    println(s"Computer fired at $p was ${human.firedAt(p)}")

  def play() =
    while !human.isAllSunk && !computer.isAllSunk do
      printWorld(); humanMove(); computerMove()
    if human.isAllSunk then println("You lost!") else println("You won!")
```

- Metoden `computerMove`
  - dragning av initial slumpposition motsvarar 1p
  - kontroll av om position ingår bland missar motsvarar 2p
  - kontroll av om position ingår bland träffar motsvarar 3p
  - dragning av ny slumpposition om villkor är uppfyllt motsvarar 1p
  - anrop av `fireAt` motsvarar 1p
  - korrekt utskrift inkl. rätt text och position mostvarar 1p