# Bedömningsmall för kontrollskrivning EDAA45 Programmering, grundkurs

## 2023-01-07, 08:00-13:00

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. Tolka uttryck.** 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` | | | `String` | `SE3` |
| `u2` | | | `String` | `SE1` |
| `u3` | | | `Option[String]` | `Some(SE1)` |
| `u4` | | **X** | | |
| `u5` | | | `Double` | `1.0` |
| `u6` | **X** | | | |
| `u7` | | **X** | | |
| `u8` | | | `Boolean` | `false` |
| `u9` | | | `String` | `SE1` |
| `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. Spelet ”Bubblor”.** 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.

### Lösningar och specifika bedömningsriktlinjer del B

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

- 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.

```scala
package solution

import scala.util.Random

enum Bubble(symbol: String):
    case Empty extends Bubble(" ")
    case Red extends Bubble("r")
    case Blue extends Bubble("b")
    case Green extends Bubble("g")
    case Orange extends Bubble("o")

    override def toString(): String = this.symbol

object Bubble: // Uppgift 2, Totalt: 4p
    def length = values.length
    def random(): Bubble = random(Random)

    // 4p
    def random(rand: Random): Bubble = Bubble.fromOrdinal(rand.nextInt(length-1)+1)
```

- Metoden `random`
  - dragning av slumptal är värt 1p
  - exkludering av `Empty` vid dragning är värt 2p
  - korrekt retur och användandet av `fromOrdinal` är värt 1p
  - om konstant värde för längden används istället för `length` så dras 1p

```scala
package solution

// x = column
// y = row
case class Pos(x: Int, y: Int) extends Ordered[Pos]: // Uppgift 3, Totalt: 4p
    def compare(that: Pos) =
        val dx = x - that.x // 1p
        if dx == 0 then // 2p
            y - that.y // 1p
        else
            dx

    override def toString = s"${(y + 'A').toChar}$x"
```

- Metoden `compare`
  - beräknandet av skillnad i kolumn (x-led) är värt 1p
  - korrekt villkor när det inte finns skillnad i kolumner, x-led och seperationen till två fall eller motsvarande är värt 2p
  - korrekt hantering av returvärde värt 1p

```scala
package solution

case class BubbleIsland(bubbles: Vector[Pos]): // Uppgift 4, Totalt: 1p
    // not empty and sorted
    require(bubbles.nonEmpty
            && bubbles.zip(bubbles.tail).forall((a,b) => a.compare(b) <= 0))

    def score = bubbles.length * (bubbles.length-1)
```

- Metoden `score`
  - helt korrekt uttryck är värt 1p

```scala
package solution

import scala.util.Random

object BubbleBoard: // Uppgift 5, Totalt: 4p
    val Size = 10

    def neighbors(pos: Pos): Vector[Pos] =
        Vector(
            Pos(pos.x-1, pos.y),
            Pos(pos.x+1, pos.y),
            Pos(pos.x, pos.y-1),
            Pos(pos.x, pos.y+1)
        )

    def create(): BubbleBoard = new BubbleBoard(Size, Size)

    def randomBubbles(n: Int, rand: Random): Vector[Bubble] =
        Vector.fill(n)(Bubble.random(rand)) // 4p


class BubbleBoard(val width: Int = Size,
                  val height: Int = Size) extends Cloneable:
    import BubbleBoard._
    import scala.collection.mutable

    private var random = new scala.util.Random()
    private var board = Vector.fill(width,height)(Bubble.random()) // 3p
    private var board_alt = Vector.fill(width)(randomBubbles(height, random))

    def apply(pos: Pos): Bubble = apply(pos.x, pos.y)

    def apply(x: Int, y: Int): Bubble =   // 4p
        if x < 0 || x >= width || y < 0 || y >= height then
            Bubble.Empty
        else
            board(x)(y)

    def set(pos: Pos, bubble: Bubble): Unit = set(pos.x, pos.y, bubble)

    def set(x: Int, y: Int, bubble: Bubble): Unit = // 3p
        board = board.updated(x, board(x).updated(y, bubble))

    def isEmptyRowAtTop: Boolean = // 3p
        (0 until width).forall(x => this(x, 0) == Bubble.Empty)

    def isEmptyRowAtTop_alt: Boolean =
        (0 until width).filter(x => this(x,0) != Bubble.Empty).isEmpty

```

```
    def isEmptyRowAtTop_alt2: Boolean =
        var isEmpty = true
        for x <- 0 until width do
            if this(x,0) != Bubble.Empty then
                isEmpty = false

        isEmpty

    def isEmptyColumnAtLeft: Boolean = // 3p
        (0 until height).forall(y => this(0, y) == Bubble.Empty)

    def isEmptyColumnAtLeft_alt: Boolean =
        (0 until height).filter(y => this(0,y) != Bubble.Empty).isEmpty

    def isEmptyColumnAtLeft_alt2: Boolean =
        var isEmpty = true
        for y <- 0 until height do
            if this(0,y) != Bubble.Empty then
                isEmpty = false

        isEmpty

    def addNewRowToTop(): Unit =  // 3p
        for x <- 0 until width do
            set(x,0, Bubble.random(random))

    def insertNewColumn(): Unit = //5p
        // Slumpmässiga bubblor från 1 till höjden, 2p
        val randomColumn = randomBubbles(random.nextInt(height)+1, random)

        // Padding, 1p
        val fullCol = Vector.fill(height-randomColumn.length)(Bubble.Empty)
            ++ randomColumn

        // For-loop för att sätta, 1p för räckvidd och 1p för sätta korrekt
        (0 until height).foreach(y => set(0, y, fullCol(y)))

        // Används setColumn så motsvarar det padding + loop = 3p

    def insertNewColumn_alt(): Unit =
        // Different way of padding using builtins
        val fullCol = randomBubbles(random.nextInt(height)+1, random)
                        .reverse
                        .padTo(height, Bubble.Empty)
                        .reverse

        (0 until height).foreach(y => set(0, y, fullCol(y)))

    def clearOut(island: BubbleIsland) =
        island.bubbles.foreach(pos => set(pos, Bubble.Empty)) // 3p
```

```scala

    def setColumn(x: Int, col: Vector[Bubble]) = // 4p
        val padded = Vector.fill(height-col.length)(Bubble.Empty) ++ col // 2p

        board = board.updated(x, padded) // 2p

    def setColumn_alt(x: Int, col: Vector[Bubble]) =
        val padded = Vector.fill(height-col.length)(Bubble.Empty) ++ col // 2p

        for y <- 0 until height do // 2p
            set(x, y, padded(y))

    def shiftDown(): Unit = // 6p
        for x <- 0 until width do // 1p
            val filteredBubbles =
                (0 until height).map(y => apply(x, y))          // 1p
                                .filter(b => b != Bubble.Empty) // 2p

            setColumn(x, filteredBubbles.toVector) // 2p

    def shiftDown_alt(): Unit =
        // alternative imperative solution
        for x <- 0 until width do // 1p
            var i = height-1
            var y = height-1

            while y >= 0 do // 3p
                if this(x,y) != Bubble.Empty then
                    set(x, i, this(x,y))
                    i -= 1
                y -= 1

            // Do not forget to clear out ones at the top
            for y <- 0 to i do
                set(x, y, Bubble.Empty)               // 2p

    def compactColumnsRight() = // 10p
        // filter out non empty columns
        val nonEmptyCols: Vector[Vector[Bubble]] =
            board.filter(col => col.exists(c => c != Bubble.Empty)) // 3p

        // calculate offset
        val offset = width - nonEmptyCols.length // 2p

        // Move non-empty to the right
        for x <- 0 until nonEmptyCols.length do
            setColumn(x+offset, nonEmptyCols(x)) // 3p

        // Clear out empty to the left
        for x <- 0 until offset do
```

```
            setColumn(x, Vector.empty[Bubble]) // 2p

    def compactColumnsRight_alt() =
        // Alternative, imperative solution:
        var k = width-1 // 1p
        var x = width-1
        while x >= 0 do
            // if current column is not empty copy
            if (0 until height).exists(y => this(x,y) != Bubble.Empty) then // 3p
                // Not empty column
                for y <- 0 until height do
                    set(k, y, apply(x,y))       // 2p

                k -= 1                          // 1p
            x -= 1                              // 1p

        // Clear out columns
        for x <- 0 to k do                      // 1p
            for y <- 0 until height do          // 1p
                set(x, y, Bubble.Empty)     // 1p

    def findIsland(start: Pos): Option[BubbleIsland] = // 14p
        var returnValue: Option[BubbleIsland] = None

        if this(start) != Bubble.Empty then    // 1p
            var workList   = Vector(start)     // 1p
            var visited    = mutable.Set.empty[Pos]    // 1p
            val matchBubble = this(start)

            while workList.nonEmpty do            // 1p
                val current: Pos = workList.head  // 1p
                visited += current                // 1p
                workList = workList.tail          // 1p
                workList = workList ++ (                    // 1p
                    neighbors(current)                      // 1p
                             .filterNot(                    // 1p
                                visited.contains(_)         // 1p
                             )
                             .filter(pos => this(pos) == matchBubble)) // 1p

            if visited.size > 1 then  // 1p
                returnValue = Some(BubbleIsland(visited.toVector.sorted)) // 1p

        returnValue

    // Hjälpmetod för hasIsland
    def indices: IndexedSeq[Pos] =
        for x <- 0 until width; y <- 0 until height yield Pos(x,y)

    def hasIsland: Boolean = // 6p
```

```
        indices.filter(pos => this(pos) != Bubble.Empty)              // 2p
               // first, filter out all empty squares
               .exists(pos =>
                    // has any neighbor of same color
                    neighbors(pos).exists(
                        neighbor => this(neighbor) == this(pos)
                    ) // 2p
                ) // 2p
```

- Metoden `randomBubbles` i kompanjonsobjektet
  - skapa en vektor av variabel storlek med `fill` eller motsvarande imperativ lösning är värt 2p
  - korrekt antal slumpmässiga bubblor, dvs $n$ är värt 1p
  - användning av `Bubble.random` eller motsvarande är värt 1p
- Metoden `apply`
  - korrekt villkor för x/kolumn, dvs om utanför ge `Bubble.Empty` är värt 1p
  - korrekt villkor för y/rad, dvs om utanför är värt 1p
  - korrekt seperation till 2 fall, är värt 1p
  - korrekt returvärde och indexering för att ruta är värt 1p
- Attributet `board`
  - skapa en vektor med `fill` eller motsvarande imperativ lösning är värt 1p
  - korrekt storlek i x-led, width är värt 1p
  - korrekt storlek i y-led och användandet av `Bubble.random` eller `randomBubbles` är värt 1p
- Metoden `set`
  - ersätta gammal referens till ny efter ändring är värt 1p
  - korrekt uppdatering i enbart x-led eller y-led (en axel) är värt 1p
  - korrekt uppdetering i bägge leder (x och y) är värt 1p utöver ovan
- Predikatet `isEmptyRowAtTop`
  - går igenom hela översta raden (längs x, y = 0) är värt 1p
  - verifierar att alla är tomma eller motsvarande (se alt2 för imperativ lösning) är värt 1p
  - korrekt returvärde, kräver att ovan är korrekt är värt 1p
- Predikatet `isEmptyColumnAtLeft`
  - går igenom hela vänstra kolumnen (längs y, x = 0) är värt 1p
  - verifierar att alla är tomma eller motsvarande (se alt2 för imperativ lösning) är värt 1p
  - korrekt returvärde, kräver att ovan är korrekt är värt 1p
- Metoden `addNewRowToTop`
  - uppdaterar hela översta raden värt 1p
  - användandet av `set` eller motsvarande är värt 1p
  - användandet av `Bubble.random` är värt 1p

- Metoden `insertNewColumn`
  - skapar en ny kolumn från 1 till $n$ med slumpmässiga bubblor, dvs använder `randomBubbles` eller återimplementerar är värt 2p
  - ser till att den slumpade kolumnen sätts in från botten är värt 2p om:
    * bubblorna sätts in direkt (inplace) i kolumnen då vi kan anta att kolumnen är tom
    * eller om det skapas en ny kolumn så måste den vara paddad till rätt storlek
  - sätter in kolumnen på rätt plats är värt 1p
  - `setColumn` som implementeras efteråt är tillåtet att användas vilket då motsvarar 2p (rätt storlek och sätts in från botten)
- Metoden `clearOut`
  - går igenom alla positioner inuti ön är värd 1p
  - användandet av `set` med rätt argument eller motsvarande är värt 1p
  - sätter Bubble.Empty för alla positioner är värt 1p
- Metoden `setColumn`
  - paddar, fyller ut från början med tomma bubblor upp till höjden är värt 2p
  - kopering till brädet för hela kolumnen är värt 2p
- Metoden `shiftDown`
  - går igenom alla kolumner är värt 1p
  - packar kolumn dvs ser till att inga tomma bubblor finns emellan är värt 3p
  - ser till att kolumnen är paddad på toppen med tomma bubblor eller använder `setColumn` är värt 2p
- Metoden `compactColumnsRight`
  - hittar de kolumner med innehåll är värt 3p
  - placerar de icke tomma kolumnerna sist är värt 5p
  - ser till att kolumner som ska vara tomma efter operationen faktiskt är tomma är värt 2p
- Metoden `findIsland`
  - korrekt hantering av returvärdet är värt 4p
    * ger ö tillbaka på korrekt sätt
    * ger ö endast om åtminstone 2 eller fler bubblor finns
    * ger ö om position har en bubbla
    * om ö ges tillbaka så är positionerna sorterade, max 1p i avdrag för specifikt detta moment om det ej uppfylls
  - skapar lämpliga datastruktur för arbetslista och mängd är värt 1p
  - intiterar dessa datastruktuer på korrekt sätt är värt 1p
  - reducerar arbetslistan och markerar reducerat element som besökt är värt 3p
  - utökar arbetslistan med alla icke besökta grannar som har samma färg som start position är värt 5p
- Metoden `hasIsland`

- går igenom alla icke-tomma platser är värt 2p
- inre undersökning om det finns en granne med samma färg utifrån en given position är värt 2p
- reducerar undersökningen av alla icke-tomma platser till ett enda booleskt returvärde är värt 2p
- löser uppgiften men är inte effektiv lösning så som tenta beskriver utan använder `findIsland` dras 2p