Lösning till tentamen
EDAA45 Programmering, grundkurs

2021-01-11, 14:00-19:00

Generella bedömningsriktlinjer

- Totala poängen för en uppgift kan inte bli negativ; om alla avdrag för en uppgift blir mer än maxpoängen ges 0 poäng.
- Om samma typ av fel förekommer i samma (del)uppgift mer än en gång görs bara avdrag en gång vid första förekomsten. Det kan vara en svår bedömning om uteblivet avdrag pga ”samma fel” ska tillämpas.
- 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. Helhetsintrycket ska vara med i avvägningen.

Poängmarkering

- Skriv en kommentar bredvid poängmarkering som förklarar poängen om det inte är uppenbart varför poängen dras/ges.
- **+ –**. Välj för en viss uppgift (t.ex. B1) en av dessa sätt att markera poäng: , men om lösningen saknar stora delar och/eller långt ifrån att fungera är det lättast att ange pluspoäng.
  - **Minuspoäng**. Detta är det vanligaste sättet att markera och det lättaste om lösningen är någorlunda komplett och rimlig.
    - Poängen markeras genom att ange *avdrag* med t.ex. `-1`
    - Om ett avdrag görs en gång för samma fel som förekommer på andra ställen i uppgiften ange en punkt efter poängen, t.ex. `-1.`
    - På det ställe där samma fel återkommer ange ett långt minustecken — utan poäng efter.
  - **Pluspoäng**. Om lösningen saknar stora delar och/eller långt ifrån att fungera är det lättast att ange pluspoäng t.ex. `+3` som motsvarar helhetsbedömning av vad denna deluppgift är värd.
- Om det finns brister eller problem som inte renderar avdrag, men som bör påpekas i upplysande syfte, ange ett långt minustecken — utan poäng efter och skriv en kommentar om vad det gäller.
- Obetydliga brister som inte ger avdrag markeras också med ett långt minustecken — utan poäng efter. Om många obetydliga brister förekommer kan dessa adderas till ett engångsavdrag om det är lämpligt när helhetsbedömning görs och uppgiftens poängsumma räknas samman.

Generella avdrag och typiska fel

- Felaktigt matchade (klammer)parentespar, eller glömda (klammer)parenteser ger -1 i avdrag en gång per uppgift.
- Om nödvändigt semikolon saknas (t.ex. mellan flera satser på samma rad) ges -1 i avdrag.
- Mindre fel i likhet med föregående punkter där det framgår tydligt vad som egentligen avses ger -1 i avdrag. Obetydliga fel behöver ej ge avdrag om de är enstaka.
- Mindre fel vid användning av samlingsmetod, t.ex. `+:` i stället för `:+` eller liknande, ger -1 i avdrag.
- Deklaration av **`var`** när det skulle kunna vara en **`val`** och fungera lika bra ger -1 i avdrag
- Deklaration av **`val`** när det måste vara **`var`** för att koden ska fungera ger -2 i avdrag.
- Onödig användning av **`this`** ger -1 i avdrag.
- `=` istf. `==` ger -1 i avdrag.
- Saknad **`else`**-gren så att värde saknas ger -2 i avdrag.

Lösning. 80 p

Uppgift 1. Matris. 22p

```scala
sealed trait Direction
case object Horizontal extends Direction
case object Vertical   extends Direction

class Crossword(val rows: Int, val cols: Int){
  private val xss = Array.fill(rows, cols)(' ')

  def charAt(row: Int, col: Int): Char =
    if (row >=0 && col >= 0 && row < rows && col < cols) xss(row)(col) else '*'

  def get(row: Int, col: Int, length: Int, dir: Direction): String = { // 10p
    val charSeq = dir match {
      case Horizontal => (col until col + length).map(charAt(row, _))
      case Vertical   => (row until row + length).map(charAt(_, col))
    }
    charSeq.mkString
  }

  def canFit(row: Int, col: Int, s: String, dir: Direction): Boolean = {
      val current = get(row, col, s.length, dir)
      current.indices.forall(i => current(i) == ' ' || current(i) == s(i))
  }

  def update(row: Int, col: Int, s: String, dir: Direction): Boolean =  // 12p
    if (canFit(row, col, s, dir)) {
      dir match {
        case Horizontal => s.indices.foreach(i => xss(row)(col + i) = s(i))
        case Vertical   => s.indices.foreach(i => xss(row + i)(col) = s(i))
      }
      true
    } else false

  override def toString = {
    def rowString(r: Int) = (-1 to cols).map(charAt(r, _)).mkString + "\n"
    (-1 to rows).map(rowString).mkString
  }
}
```

- **`get`** 10p fördelas ungefär så här: 2p för kontroll av riktning, 3p för respektive loop, 2p för retur

  –1 för missat mkString eller fel returtyp t.ex. Seq[Char]

- **`update`** 12p fördelas ungefär så här: 2p för kontroll med `canFit`, 2p för kontroll av riktning, 3p för respektive loop, 2p för retur

Uppgift 2. Registrering. 34p

```scala
object labreg {

  type LabId = String
  type StudentId = String

  sealed trait Status
  case object Approved  extends Status //godkänd
  case object Residual  extends Status //kompletteras
  case object Postponed extends Status //uppskov tex pga sjukdom
  case object Missing   extends Status //saknad eller okänd status

  object Status {
    val values = Vector[Status](Approved, Residual, Postponed, Missing)

    def fromString(s: String): Status = s match { //4p
      case "godkänd"      => Approved
      case "kompletteras" => Residual
      case "uppskov"      => Postponed
      case _              => Missing
    }
  }

  case class Student(id: StudentId, labStatus: Map[LabId, Status]){
    def updated(lab: LabId, status: Status): Student =  //4p
      copy(labStatus = labStatus + (lab -> status))

    def statusOf(lab: LabId): Status = labStatus.getOrElse(lab, Missing) //3p
  }

  class LabReg {
    import scala.collection.mutable
    private val reg = mutable.Map[StudentId, Student]()
    private val labs = mutable.Set[LabId]()

    def apply(sid: StudentId): Student = reg(sid)

    def studentIds: Vector[StudentId] = reg.keys.toVector.sorted //3p

    def labIds: Vector[LabId] = labs.toVector.sorted  //2p

    def add(sid: StudentId, lab: LabId, status: Status): Unit = { //12p
      val student = reg.getOrElse(sid, Student(sid, Map.empty))
      reg.update(sid, student.updated(lab, status))
      labs.add(lab)
    }

    def labStatusCount(lab: LabId, status: Status): Int =  //6p
      reg.values.count(_.statusOf(lab) == status)
  }
}
```

Poängfördelning 2

Alla poängfördelningar nedan är *ungefärlig* uppdelning av max poäng på olika delar i en deluppgift; anpassa bedömningen till den specifika lösningen och gör en helhetsbedömning.

- `fromString` 4p fördelning: 2p för koll av strängar, 2p för hantering av saknat värde
- `updated` 4p fördelning: 2p för skapa ny student, 2p för skapa ny Map med uppdaterad status
- `statusOf` 3p fördelning: 1p för hämta ur `labStatus`, 1p för att det är `lab` som hämtas, 1p för hantering av saknat värde med `Missing`
- `studentIds` 3p fördelning: 1p för hämta nycklarna ur `reg`, 1p för att göra om till sekvens `toVector`, 1p sortering
- `labIds` 2p fördelning: 1p för att göra om till sekvens `toVector`, 1p sortering
- `add` 12p fördelning: 2p för koll om saknad studentid, 3p för skapa ny student med rätt Map vid saknat värde, 2p för uppdatering av studentens labStatus, 3p för uppdatering av reg, 2p för uppdatering av labs
- `labStatusCount` 6p fördelning: 3p för att ta fram alla studenter, 3p för räkna rätt status.

Uppgift 3. Java. 24p

```java
import java.util.ArrayList;

public class HighScoreList {
    private ArrayList<GameResult> results; //2p

    public HighScoreList(){
        results = new ArrayList<GameResult>();  //2p
    }

    public static boolean isBefore(GameResult gr1, GameResult gr2){  //4p
      if (gr1.getScore() == gr2.getScore()) {
        return gr1.getName().compareTo(gr2.getName()) < 0;
      } else {
        return gr1.getScore() > gr2.getScore();
      }
    }

    public void insert(GameResult gr){ //8p
        int pos = 0;
        while (pos < results.size() && isBefore(results.get(pos), gr)){
            pos++;
        }
        results.add(pos, gr);
    }

    public ArrayList<GameResult> topList(int n){ //8p
        ArrayList<GameResult> topResults = new ArrayList<GameResult>();
        for (int i = 0; i < n && i < results.size(); i++){
            topResults.add(results.get(i));
        }
        return topResults;
    }
}
```

- Ok om konstruktur är tom eller saknas om attributet initialiseras korrekt vid deklaration.
- Saknat **`new`** av ArrayList vid initialisering -1 i avdrag
- `isBefore`: saknat `compareTo` -2 i avdrag
- `topList`
  - glömt skapa ny lista och/eller returnerar intern struktur istf kopiering -2 i avdrag
  - hårdkodat antalet till 10 eller andra väsentliga förenklingar/avsteg -3 i avdrag
- Typiska **engångsavdrag** med -1 i avdrag:
  - saknat semikolon
  - saknat return men rätt värde i slutet av blocket
  - `Int` istf `int`
  - saknade tomma parenteser efter metod `()`
  - fel indexering med `xs(i)` istf `xs[i]` eller `xs.get(i)`
  - felaktig uppdatering av ArrayList, tilldelning istället för `add`
  - Onödigt **`this`** på statisk medlem