Pitanja

1. [6] Napisati tip podatka `f` koji je definisan na sledeći način:
```
val f = (x: Int) => (y: Int) => x*y
```

2. [6] Objasniti kako se može dohvatiti vrednost upotrebljena pri inicijalizaciji objekta primenom mehanizma uparivanja obrazaca? Dati primer.

3. [6] Objasniti namenu tipa Option i navesti njegove podtipove.

4. [6] Dat je sledeći niz definicija. Napisati linearini poredak za klasu Jogurt.
```
class Proizvod
trait Ambalaziran extends Proizvod
trait Pasterizovan extends Ambalaziran
trait Mlecni extends Proizvod

class Jogurt extends Proizvod
      with Pasterizovan with Mlecni
```

5. [6] Objekat Nil definisan je na sledeći način: `case object Nil extends List[Nothing]`. Objasniti zašto on može da učestvuje u sastavljanju liste proizvoljnog tipa T.

Zadaci

1. [15] Napisati generičku funkciju koja od zadate kerifikovane funkcije sa dva parametra tipa U i V, respektivno, rezultata tipa T, pravi nekerifikovanu verziju te funkcije, sa dva parametra.

```
def uncurry[U, V, T](f : U => V => T) : (U, V)=>T =
    (u, v) => f(u)(v)
```

2. [15] Napisati generičku funkciju `filter` koja iz zadate liste proizvoljnog tipa podataka izostavlja određene elemente. Izostavljanje se vrši na osnovu predikatske funkcije koja se zadaje kao drugi parametar funkcije `filter`.

```
def filter[T](l : List[T], f: T => Boolean) : List[T] = l match
{
      case Nil => Nil
      case h :: Nil if f(h) => List(h)
      case h :: tail if f(h) => h :: filter(tail, f)
      case _ :: tail => filter(tail, f)
}
```

ili

```
def filter2[T](l : List[T], f: T => Boolean) : List[T] =
{
      for( e <- l
                  if f(e) ) yield(e)
}
```

3. [20] Napisati klasu aktera koji računa i ispiše zbir dostavljenih celobrojnih vrednosti. Broj očekivanih vrednosti *n* se zadaje kao parametar konstruktora aktera. Vrednosti se dostavljaju porukama koje sadrže i celobrojni identifikator porekla vrednosti, u opsegu od 1 do *n*. Identifikator porekla ne treba proveravati. Akter ignoriše višestruka dostavljanja vrednosti pod istim identifikatorom porekla. Akter ispiše zbir vrednosti nakon što primi vrednosti za sve očekivane identifikatore.

```
case class Result(r : Int, id: Int)
class Adder(n: Int) extends Actor {
    private var res = 0
    private val done = new Array[Int](n)
    def receive = {
      case Result(r, id) => {
        if( done(id-1) == 0 ) {
          done(id-1) = 1
          res += r
          if( ! done.contains(0) )
            println("Sum = " + res)
        }
      }
      case _ =>
    }
}
```

4. [20] Napisati program koji inicijalizuje jedan sistem aktera i stvori aktera iz zadatka 3 za računanje zbira 5 dostavljenih vrednosti. Zatim program napravi 5 budućih vrednosti, tako da i-ta buduća vrednost računa zbir vrednosti u opsegu od i·10 do i·20, pri čemu čeka 1 ms nakon svakog koraka računanja zbira, i dostavlja tu vrednost akteru. Program čeka 2000 ms pre završetka.

```
import akka.actor._
import scala.concurrent.{Future}
import scala.concurrent.ExecutionContext.Implicits.global

object Sum extends App {
```

```
  val system = ActorSystem("AdderSystem")
  val adder = system.actorOf(Props(new Adder(5)), name = "adder")

  for(i <- 1 to 5) {
    Future {
        var x = 0
        for(j <- i*10 to i*20) {
          Thread.sleep(1)
          x += j
        }
        adder ! Result(x, i)
    }
  }
  Thread.sleep(2000)
  system.shutdown()
}
```