a)

```
def transf(xs: List[Int]): List[Int] =
{
  xs match {
    case Nil => Nil
    case x :: Nil => List(x)
    case x :: y :: ys => y :: x :: transf(ys)
  }
}
```

b)

```
def product(lst: List[Int]): Int =
{
  lst match {
    case Nil => 1
    case h :: t => h * product(t)
  }
}
```

c)

```
def end(lst:List[Int], e: Int): List[Int] =
{
  lst match{
    case Nil => List(e)
    case h :: Nil => h :: List(e)
    case x => x ++ List(e)
  }
}
```

OR

```
def end(lst: List[Int], e: Int): List[Int] =
{
  lst match {
    case Nil => List(e)
    case _ => lst ++ List(e)
  }
}
```

d)

```
def concat1[E](lst1:List[E], lst2:List[E]): List[E] =
{
  lst1 match {
    case List() => lst2
    case h::t => h::concat1(t,lst2)
  }
}
```

OR

```
def concat[E](lst1:List[E], lst2:List[E]): List[E] = { lst1 ++ lst2}
```

e)

```
def aux1(xs: List[(Int, Int)], count: Int, acc: Int): Int =
{
  xs  match {
    case Nil => acc
    case h :: t => {
      if(count == 2 || count == 4)
        aux1(t, count+1, acc + h._1 + h._2)
      else
    aux1(t, count+1, acc)
    }
  }
}

def sumEven(xs: List[(Int, Int)]): Int = { aux1(xs, 0, 0)}
```

f.a)

```
def lengSum(l : List[Double]) : (Int,Double) =
{
  l match {
    case Nil => (0,0)
    case h :: t => {
      val v = lengSum(t)
      (1+v._1, h + v._2)
    }
  }
}
```

f.b)

```
def average( l : List[Double]) : Double =
{
  val v = lengSum(l)
  v._2 / v._1
}
```

g)

```
def divByValue(l : List[Double], e: Double) : (List[Double],
List[Double]) =
{
  l match {
    case Nil => (Nil, Nil)
    case h :: t => {
      if(h < e)
        (h :: divByValue(t, e)._1, divByValue(t, e)._2)
      else
    (divByValue(t, e)._1, h :: divByValue(t, e)._2)
    }
  }
}
```

h)

```
def divByAverage(l : List[Double]) : List[Double] =
{
  val av = average(l)
  val res = divByValue(l, av)
  res._2
}
```
EXTRA)

```
def aux1[E]( l : List[E], len: Int) : (List[E], List[E]) =
{
  l match {
    case Nil =>  (Nil, Nil)
    case h :: t => {
      if(len != 0)
      {
        val v = aux1(t.init, len-1)
         (List(h) ++ v._1, v._2 ++ List(t.last))
      }
      else {
        if (!t.isEmpty)
          (List(h), List(t.last))
        else
     (List(h), List())
      }
    }
  }
}

def divide[E]( l : List[E]) : (List[E], List[E]) =
{
  val len = l.length / 2
 if(l.length % 2 == 0)
    aux1(l,len-1)
  else
  aux1(l,len)
}
```
OR

```
def divide[E](lst: List[E]) : (List[E], List[E]) =
{
  lst match {
    case Nil =>  (Nil, Nil)
    case List(x) => (x::Nil, Nil)
    case x::xs => { val aux = divide(xs.init)
       (x::aux._1, aux._2++List(xs.last))
    }
  }
}
```
i)

```
def emailPrefix2(lst : LTelef) : List[String] =
{
  lst match {
    case Nil =>  Nil
  case (_ , phone , email) :: tail => {
      if(phone(0) == '2')
```

```
        email :: (emailPrefix2(tail))
      else
    emailPrefix2(tail)
     }
   }
}
```

j)

```
def pairPhoneEmail(lst : LTelef, name: String) : (String, String) =
{
  lst match {
    case Nil => ("","")
    case (n ,phone ,email) :: tail => {
      if(name.equals(n))
        (phone, email)
      else
    pairPhoneEmail(tail, name)
     }
   }
}
```