IMPERATIVE PROGRAMMING HT2018

SHEET 1

GABRIEL MOISE

QUESTION 1

```
object Question1
{
  /* Calculating the square of the input */
  def square (n : Int) : Int = n*n
  /* Calculating the remainder of n, when divided by 3 */
  def divide (n : Int) : Int = n-3*(n/3)
  /* Calculating the largest perfect square no more than n */
  def largest (n : Int) : Int =
    {
      var i = 0
      /* Invariant I : i <= floor(sqrt(n))+1, where sqrt(n) is the square root of n */
      /* Variant : floor(n)+1-i */
      while (i*i<=n)
      {
        // I
        i = i+1
      }
      // When we get out of the while-loop, i*i>n => i=floor(sqrn(n))+1, so we decrement it
      i = i-1
      // Now, i is the largest number whose square is the greatest perfect square less than n
      i * i
      // We then print its square
    }
}
```

QUESTION 2

```
object Question2
{
 def findSum (a : Array [Int]) : Int =
  {
   val n = a.size
   var i = n; var s = 0
   // Invariant I : s = sum (a[i..n)) && 0<=n-i<=n
   // Variant : i
   while (i>0)
   {
    // I && 0<=n-i<n
    i = i-1
    s = s+a(i)
    // I && 0<=n-i<=n
   }
   // I && i=0, so s = sum(a[0..n))
   s
  }
}
```

QUESTION 3

```
object Question3
{
 /** Calculate the biggest element of a
   * Post: returns the maximum of the list a */
 def findMax(a : Array[Int]) : Int =
 {
   val n = a.size
   var max = 0; var i = 0
   // Invariant I: max = max(a[0..i)) && 0<=i<=n
   // Variant n-i
   while(i < n){
     // I && i<n
     if (max<a(i)) max = a(i)
     // max = max(a[0..i+1)) && i<n
     i = i+1
     // I && i<=n
   }
   // I && i=n => max = max(a[0..n))
   max
 }
}
```

QUESTION 4

```
object Milk
{
 def findSum(a : Array[Int]) : Int = //We can use BigInt here instead of Int
 {
  val n = a.size
  var total = 0; var i = 0
  while (i < n)
  {
   total += a(i)
   i += 1
  }
  total
 }
 def main(args : Array[String]) =
 {
  val n = args.size
  val a = new Array[Int](n)
  // a is a bad name for the array, we can name it with something more representating to avoid confusion, such as "pints"
  // We can write here val a = args.map(x => x.toInt) instead of using a for
  /* We can check here if all the elements from the input array are positive numbers:
   i=0
   while (i<n)
          { require a(i)>=0
          i = i +1}  */
  println(findSum(a))
 }
}
```

QUESTION 5

```
object Question5

{

  var depth = 0

  def fib (n : Int) : Int = {

      var result = 0

      var i = 1

      // Here, we print the number of | corresponding to depth

      while (i <= depth) {i = i+1 ; print("| ")}

      println("fib("+n+")")

      if (n==0)

      {

        i=1

        while (i <= depth) {i = i+1; print("| ")}

        println("= 0")

        result = 0

      }

      else if (n==1)

      {

        i=1

        while (i <= depth) {i = i+1; print("| ")}

        println("= 1")

        result = 1

      }

           else

           {

             depth = depth + 1

             result = fib(n-1) + fib(n-2)

             depth = depth - 1 ; i=1

             while(i <= depth) {i = i+1; print("| ")}

             println("= "+result)

           }

      result

    } }
```

QUESTION 6

```
object Question6
{
 def fib (n : Int) : Int =
  {
   var f0 = 0 ; var f1 = 1 ; var fn = 0
   if (n == 0) fn = f0
   else if (n == 1) fn = f1
        else
   {
     var i = 2 ; var fi = 1
     // Invariant I : fi = the ith Fibonacci number &&
     // f1 = the (i-1)th Fibonacci number &&
     // f0 = the (i-2)th Fibonacci number && i<=n
     // Variant n-i
     while (i<n)  {
      i = i + 1
      // fi = fib(i-1) && f1 = fib(i-2) && f0 = fib(i-3)
      f0 = f1
      // fi = fib(i-1) && f1 = fib(i-2) && f0 = fib(i-2)
      f1 = fi
      // fi = fib(i-1) && f1 = fib(i-1) && f0 = fib(i-2)
      fi = f1 + f0
      //using the definition that fib(i) = fib(i-1) + fib(i-2)
      // I
     }
     // i = n
     // I
     // fi = fib(n)
     fn = fi
   }
   fn
  }
}
```

QUESTION 7

```
object Question7
{
 def divMod (x : Int, y : Int) : (Int,Int) =
  {
   var a = x ; var b = y
   var q = 0 ; var r = 0
   // Invariant I : a = b*(x/y-q)+(x%y)
   // variant x/y-q
   while (a>=b)
   {
    // I
    a = a - b
    // The LHS is smaller with b
    q = q + 1
    // The RHS is smaller with b, thus we have I
   }
   // Now, a<b, therefore x/y-q=0 (As the variant becomes 0 eventually, so q = x/y)
   // As x/y=q we have a = x%y now, so we set r to be a and we return (q,r)
   r = a
   (q,r)
  }
}
```

QUESTION 8

```
object Question8
{
 def gcd (m : Int , n : Int) : Int = {
   var a = m ; var b = n
   // Invariant I : gcd(a,b)=gcd(m,n)
   // variant b
   while (b!=0)   {
    if (a>b) a = a-b // gcd(a,b) = gcd(a-b,b)
    else b = b-a // gcd(a,b) = gcd(a,b-a)
   }
   // b==0 when the loop terminates
   // I : gcd (m,n) = gcd(a,0), so gcd(m,n) = a, so we return a
   a
  }
 def extended (m : Int ,n : Int) : (Int,Int) = {
   var a = m ; var b = n
   var x1 = 1 ; var x2 = 0 ; var y1 = 0 ; var y2 = 1
   var q = 0 ; var r = 0
   // Invariant I : gcd (m,n) = gcd(a,b) && a = (x1*m + x2*n) && b = (y1*m+y2*n) && (b>=0)
   // Variant b
   while (b!=0) {
    // I && (b>0)
    q = a / b ; r = a - q*b
    var aux1 = x1 - q * y1 // The new value for y1
    var aux2 = x2 - q * y2 // The new value for y2
    a = b ; b = r
    x1 = y1 ; y1 = aux1
    x2 = y2 ; y2 = aux2
    // I
   }
   // I && b==0, therefore gcd(m,n) = gcd(a,0) = a, and a = (x1*m+x2*n), so we return (x1,x2)
   (x1,x2)
  } }
```

QUESTION 9

```
object Question9

{

  def hits (a : Array[Int]) : Int =

    {

      val n = a.size

      var h = 0 ; var i = 0 ; var max = 0

      // Invariant I : h = hits (a[0..i)) && (0<=i<n)

      // variant (n-i)

      while (i<n)

      {

        //I

        if (max<a(i)) {h = h+1; max = a(i)}

        // Here, we use the fact that a hit is the biggest element we found so far in the list so it has to be greater than the current
maximum of the list up to that position

        //I

        i = i+1

        // I && (0<=i<=n)

      }

      // i==n so, by knowing I is true, we know that h = hits ( a[0..n) )

      h

    }

    // It runs in O(n) as we only use a while loop from i=0 to n and do 3 operations each time.

}
```