Imperative Programming (Parts 1&2): Sheet 1

Joe Pitt-Francis

Most questions by Mike Spivey, adapted by Gavin Lowe

Suitable for around Week 2, Hilary term, 2019

- Any question marked with † is a self-study question and an answer is provided at the end. Unless your tutor says otherwise, you should mark your attempts at these questions, and hand them in with the rest of your answers.
- Questions marked [**Programming**] are specifically designed to be implemented. Unless your tutor says otherwise, provide your tutor with evidence that you have a working implementation.
- In this sheet, loops should be written as `while`-loops, not `for`-loops.
- You should give an invariant for each non-trivial loop you write, together with appropriate justification.

Question 1

[**Programming**] Write short Scala functions which use just standard integer arithmetic (`* + - /`) and, on integer input:

(a) square the input;

(b) compute the remainder on dividing the input by 3; and

(c) find the largest perfect square no more than the input.

How would you convince your tutor that you have run your functions and that they give the correct answers?

This question is taken from Functional programming last term where you may have seen a "Sheet 0" which had this question in GHCi.

Question 2 †

The milk bill program developed in the lecture works by counting up from `i=0` to `i=n` (where `n` is the size of the array). Find a suitable invariant and design a program that instead counts down from `i=n` to `i=0`. What is the variant for this program?

Question 3

Write a program that operates on the same data as the milk bill program, but instead finds the maximum number of pints consumed on any day. Give the invariant *and a variant* of any loop you write.

Question 4

List as many bad points as you can think of in the milk bill program in Figure 1, and where possible suggest ways of improving on them.

```scala
object Milk{
  def findSum(a : Array[Int]) : 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)
    for(i <- 0 until n) a(i) = args(i).toInt
    println(findSum(a))
  }
}
```

Figure 1: The milk bill example

Question 5

[Programming] The Fibonacci numbers are defined by

$$\begin{aligned} fib(0) &= 0 \\ fib(1) &= 1 \\ fib(k) &= fib(k-2) + fib(k-1) \qquad \text{for } k \geq 2 \end{aligned}$$

Write a *recursive* function

```
def fib(n : Int) : Int = ...
```

that calculates the `n`th Fibonacci number $fib(n)$.

Now modify it to produce as a side-effect output which displays the tree of procedure calls like this:-

```
fib(3)
| fib(2)
| | fib(1)
| | = 1
| | fib(0)
| | = 0
| = 1
| fib(1)
| = 1
= 2
```

You might have a solution that passes a depth parameter to `fib`. Can you do it without adding any extra parameters to the calls?

Question 6

Write a *non-recursive* function

```
def fib(n : Int) : Int = ...
```

that calculates the `n`th Fibonacci number $fib(n)$. You should give an invariant for your code.

Question 7

Write a program fragment that, given `x` ≥ 0 and `y > 0`, computes `q` and `r` such that `x = q * y + r` and $0 \leq$ `r` $<$ `y`. Solve the problem using repeated subtraction. Thus the program should be equivalent to

```
  q = x/y; r = x%y
```

but without using the `/` (div) and `%` (mod) operators (or similar) operators. Give invariants for every loop in your program.

Question 8

(a) Write a program that, given two positive integers `m` and `n`, computes their greatest common divisor `a`. Do not use the `%` (mod) operator.

(b) Now adapt your program to also calculate integers `x` and `y` such that `a = mx + ny`.
Give an invariant for each loop in the programs.

Question 9

A *hit* in an array `a[0..N)` is an element that is larger than all elements to the left of it: thus a hit occurs at an index j if whenever $0 \leq i < j$ we have `a`$(i) <$ `a`(j). Assuming `N` ≥ 1, write a program (giving invariants for each loop) that counts the number of hits `h` in the array `a`. For full marks, your code should run in time $\Theta($`N`$)$.

Answer to question 2 †

(For comparison see the Figure 1 code `Milk.scala` from Week 1 lectures.) A suitable invariant is

$$0 \le \mathtt{i} \le \mathtt{n} \quad \text{and} \quad \mathtt{total} = \sum \mathtt{a}[\mathtt{i} \mathinner{..} \mathtt{n}).$$

A suitable variant is simply `i`. The resulting procedure is

```scala
// Calculate sum of a
def findSum(a : Array[Int]) : Int = {
  val n = a.size
  var total = 0; var i = n
  // Invariant I: total = sum(a[i..n)) && 0<=i<=n
  // Variant i
  while(i > 0){
    // I && i>0
    i -= 1
    total += a(i)
  }
  // I && i=0
  // total = sum(a[0..n))
  total
}
```