Imperative Programming (Parts 1&2): Sheet 2

Joe Pitt-Francis

Most questions by Mike Spivey, adapted by Gavin Lowe

Suitable for around Week 3, 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 †

(a) What is the effect of the following procedure?

```
def swap(x: Int, y: Int) = { val t = x; x = y; y = t }
```

(b) What is the effect of the following procedure?

```
def swapEntries(a: Array[Int], i: Int, j: Int) = {
  val t = a(i); a(i) = a(j); a(j) = t
}
```

Question 2 †

Consider the program

```scala
object SideEffects{
  var x = 3; var y = 5

  def nasty(x: Int) : Int = { y = 1; 2 * x }

  def main(args: Array[String]) = println(nasty(x) + y)
}
```

What integer does the program print? If the expression `nasty(x) + y` were replaced by `y + nasty(x)`, would that affect the result?

Question 3

[**Programming**] Scala has a fairly extensive library of utility functions associated with its sequence-like datatypes. For example, we can easily sort the members of an (`Int`) array into ascending order.

```
scala> val digits=Array(3,1,4,1,5,9,2,6,5)
digits: Array[Int] = Array(3, 1, 4, 1, 5, 9, 2, 6, 5)

scala> digits.sorted
res0: Array[Int] = Array(1, 1, 2, 3, 4, 5, 5, 6, 9)
```

Write a small test suite, with ScalaTest if possible, which operates on an array of `Strings` (a `val` of type `Array[String]`) containing at least 6 elements. Then test that sorting with the default `String` ordering behaves as expected. Also test the behaviour when sorting the array by something other than dictionary order and test the behaviour when applying any other utility of your choice.[1]. Note that your test suite is black-box in nature—it's not necessary to know which algorithms Scala is using but only what the functions are meant to achieve.

Question 4

The following loop represents the calculation of discrete time steps for the numerical solution of a problem such as may be found later in the *Continuous Mathematics* course.

```scala
val timeEnd:Double = 1.0
val numSteps:Int = ...
val timeStep:Double = timeEnd/numSteps
// timeEnd=numSteps*timeStep and numSteps∈ ℕ
var time = 0.0
while (time < timeEnd)
{
  // Inv: 0 <= time <= timeEnd and time=k*timeStep for some k∈ ℕ
  time += timeStep
}
// Inv => time == timeEnd
```

[1]See `http://www.scala-lang.org/api/2.12.1/scala/Array.html` for ideas

Recall that floating point calculations are subject to rounding on the scale of *machine epsilon* (ε), where ε is the smallest number such that $1 + \varepsilon$ can be represented as a floating point number which can be shown to differ from 1.

(a) How many times might the body of the loop actually be executed?

(b) The postcondition for the loop would be correct if arithmetic were exact, but is actually incorrect with floating point arithmetic. How inaccurate might `time` be after the loop?

(c) Suggest improvements to the loop such that the postcondition is correct—as 'correct' as possible. How inaccurate might `time` be after the loop now?

Question 5

```
 1  /** Does pat appear as a substring of line? */
 2  def search(pat: Array[Char], line: Array[Char]) : Boolean = {
 3    val K = pat.size; val N = line.size
 4    // Invariant: I: found = (line[i..i+K) = pat[0..K) for
 5    //                       some i in [0..j)) and 0 <= j <= N-K
 6    var j = 0; var found = false
 7    while(j <= N-K && !found){
 8      // set found if line[j..j+K) = pat[0..K)
 9      // Invariant: line[j..j+k) = pat[0..k)
10      var k = 0
11      while(k<K && line(j+k)==pat(k)) k = k+1
12      found = (k==K)
13      j = j+1
14    }
15    // I && (j=N-K+1 || found)
16    // found = ( line[i..i+K) = pat[0..K) for some i in [0..N-K+1) )
17    found
18  }
```

Figure 1: Substring searching code

Figure 1 shows our code for finding whether the pattern `pat` appears as a substring of `line`. What follows is a list of mistakes that might have been made in writing this program. For each mistake, suggest test data that would reveal the resulting bug, if any. Write a test suite to define your tests. Use ScalaTest if possible.

(a) On line 6, replace `false` by `true`.

(b) On line 7, replace `<=` by `<`.

(c) On line 7, replace `N-K` by `N-K+1`.

(d) On line 10, replace `0` by `1`.

(e) On line 11, replace `<` by `<=`.

(f) On line 12, replace `==` by `>=`.

Question 6

(Some help for Lab 1.) Say that an array of characters `s[0..N)` *recurs* with period `n` if $1 \le$ `n` $\le$ `N` and `s[0..N-n) = s[n..N)`; in other words, s starts to repeat after the first n characters. It is clear that if `N > 0` then `s` recurs with period `N` if with no smaller period. Write Scala code that finds the smallest value of `n` such that `s` recurs with period `n`. Do not use string "slicing" for comparing string subsequences. Rather use a loop and compare characters as in the *String searching* lecture.

Question 7

`Int => Boolean` represents the type of functions from `Int` to `Boolean`, and one can define an example function like so:

```
// This function returns true if a (positive) integer has 3 digits in decimal
val threedigits:(Int=>Boolean) = i => {i >= 100 && i<1000}
```

Write a loop-based definition for a function

```
def exists(p : Int => Boolean, N : Int): Boolean = ...
```

that tests whether $\mathtt{p}(i)$ holds for some $i \in [0 \mathinner{..} \mathtt{N})$, i.e. it returns the value

$$\exists i \in [0 \mathinner{..} \mathtt{N}) \bullet \mathtt{p}(i).$$

Give an invariant for your loop, together with suitable justification of the correctness of your program.

Question 8

Some fractions can be expressed as a *sum of distinct reciprocals*; for example, $\frac{5}{6} = \frac{1}{2} + \frac{1}{3}$, and $\frac{2}{35} = \frac{1}{30} + \frac{1}{42}$. Assuming $0 < \mathtt{p} < \mathtt{q}$, we could try to express the fraction $\mathtt{p}/\mathtt{q}$ in this form by first choosing the largest possible reciprocal $1/\mathtt{m} \le \mathtt{p}/\mathtt{q}$, and if this is not equal to $\mathtt{p}/\mathtt{q}$, repeating the procedure with the fraction $\mathtt{p}'/\mathtt{q}' = \mathtt{p}/\mathtt{q} - 1/\mathtt{m}$.

(a) Express the integer `m` mathematically in terms of `p` and `q`, and write a Scala expression that computes it.

(b) Write a loop that implements the idea suggested above, computing an array `d[0..n)` such that

$$\frac{\mathtt{p}}{\mathtt{q}} = \frac{1}{\mathtt{d}(0)} + \frac{1}{\mathtt{d}(1)} + \cdots + \frac{1}{\mathtt{d}(n-1)}.$$

Your code may change the values of `p` and `q`. You may assume that the array `d` is initialised to be large enough to hold all the terms you compute. In each iteration, there's no need to reduce the new fraction $\mathtt{p}'/\mathtt{q}'$ to its lowest terms.

(c) Prove that the value of `p` is reduced by the loop body, and deduce that (ignoring the possibility of overflow) the loop always terminates.

(d) Show in addition that the array d is strictly increasing, so that the reciprocals computed by the program are distinct.

Question 9

Given $\mathtt{x} \geq 1$, write a program to set $\mathtt{y} = \lfloor \log_3 \mathtt{x} \rfloor$, using a linear search. Your program should not use logs or exponentiation directly.

Question 10

We can represent a polynomial by storing its coefficients in an array. Hence an array `a[0..n)` represents the polynomial

$$p(x) \quad = \quad \sum_{i=0}^{n-1} \mathtt{a}(i) \times x^i$$

Write a function

```
def eval(a: Array[Double], x: Double) : Double = ...
```

that evaluates the polynomial represented by `a` at `x`.

Harder: do so in a way that uses only `n` multiplications.

Answer to question 1 †

(a) This produces a compiler error "`reassignment to val`". Parameters of procedures are value parameters: in other words, they are treated the same as variables introduced by `val`. Hence reassigning to them is an error. (Note: the same goes for `for` loops—we cannot reassign `i` inside `for(i<-1 until 10){...}`).

Some languages allow variables to be passed by address (sometimes called `var` parameters); this introduces an interesting class of programming error, where one variable stands as an alias for another. Also, some languages allow the values of parameters to be changed, but those changes do not propogate outside the procedure, which just seems confusing to many. Scala allows neither of these.

(b) This swaps the entries of `a` in positions `i` and `j` (assuming those are within the indices of `a`; otherwise it gives an array bounds error). The point is that `a` itself is a value parameter, but that doesn't prevent us changing the individual entries within `a`.

Answer to question 2 †

Scala evaluates the arguments of `+` (and, indeed, of any operator) from left to right (ref. *Programming in Scala*, Section 5.8). Hence in `nasty(x) + y`, `nasty(x)` is evaluated before `y` and it changes the contents of `var y`, so the result is $2 \times 3 + 1 = 7$. Swapping the arguments, as in `y + nasty(x)`, gives a result of $5 + 2 \times 3 = 11$.