REVISION PAPER

Honour School of Computer Science

Honour School of Mathematics and Computer Science

Honour School of Computer Science and Philosophy

Imperative Programming Parts 1, 2 & 3

Trinity Term 2019

Time: 3 hours

This paper contains two questions on Imperative Programming Part 1, three questions on Imperative Programming Part 2 and three questions on Imperative Programming Part 3.

Answer a total of five questions with no more than two from each of the three parts.

You should try to keep time strictly and test yourself in "mock-exam" conditions, either in your own room or in the library. Please complete the rest of the questions in your own time before revision tutorial.

Do not turn this page until you are told that you may do so

Imperative Programming Part 1

Question 1

Given an integer $N > 1$, the decimal expansion of $1/N$ can be represented by two integers $0 \le j < k$ and an array of digits $d[0\,..\,k)$, with k as small as possible, such that the $1/N$ is equal to

$$0.d_0d_1d_2\ldots d_{j-1}\overline{d_jd_{j+1}\ldots d_{k-1}},$$

where $d_i = d[i]$ satisfies $0 \le d_i < 10$, and the over-bar denotes a recurring decimal. Thus:

- If $N = 2$, then $j = 1$, $k = 2$ and $d[0\,..\,2) = [5, 0]$ because $1/2 = 0.5 = 0.5000\ldots$.
- If $N = 3$, then $j = 0$, $k = 1$ and $d[0\,..\,1) = [3]$ because $1/3 = 0.333\ldots$.
- If $d = 12$, then $j = 2$, $k = 3$ and $d[0\,..\,3] = [0, 8, 3]$ because $1/12 = 0.08333\ldots$.
- If $d = 14$, then $j = 1$, $k = 7$ and $d[0\,..\,7) = [0, 7, 1, 4, 2, 8, 5]$ because $1/14 = 0.0714285714285\ldots$.

(a) Write a program fragment so that, given `d`, `j`, and `k` have been calculated as above, the reciprocal $1/N$ will be output over two lines, with `-` symbols representing the over-bar:

```
   -
0.50

  -
0.3

    -
0.083

   ------
0.0714285
```

(2 marks)

(b) Write a program fragment that computes the decimal expansion of $1/N$ in this way. You may use whatever auxiliary variables, including arrays, that you wish, but you may not use floating-point arithmetic. Your program should run in a time proportional to N. (9 marks)

(c) Explain why your program fragment computes the right result, being careful to show that the sequence of decimal digits is correct, that your program correctly identifies the recurring segment, and that it terminates whatever the given value of N. (9 marks)

Question 2

(a) In terms of a function `Partition` that you should specify precisely, write a function

```
def Sort(a: Array[Int], n:Int) : Unit
```

that sorts an array of integers $a[0\,..\,n)$ into ascending order using the Quicksort algorithm. Show informally that your implementation correctly sorts the input array; if your argument depends on induction, make it clear what induction principle you are using. (8 marks)

(b) One implementation of `Partition` chooses a pivot value x, and separates the elements y in a segment of the array into those that satisfy $y < x$ and those that satisfy $y \geq x$. Explain why this can make Quicksort inefficient for certain input arrays, stating precisely what the behaviour of the procedure is for these arrays. (4 marks)

(c) A proposal to avoid this behaviour is to choose a pivot element x, then scan inwards from both ends of a given segment of the array, gathering elements y with $y \leq x$ at one end, and elements with $y \geq x$ at the other. When an element $\geq x$ has been found at the left, and an element $\leq x$ at the right, the two elements are swapped before scanning continues. In particular, elements equal to x that are found at the left end are moved to the right end like those that are greater than x; similarly, elements equal to x found at the right end are moved to the left end.

Implement this scheme by writing a version of `Partition`, and if necessary adjust the implementation of `Sort` that you gave in part (a). Discuss whether this idea does indeed avoid the problem of part (b). (8 marks)

Imperative Programming Part 2

Question 3

(a) Write a formal specification for the *abstract datatype* set of integers, as a Scala `trait IntSet`, describing the abstract state space and the pre- and post-conditions of each operation. The following are the required operations of the `trait`:

- `add(elem:Int)`
- `isIn(elem:Int)`
- `remove(elem:Int)`
- `size`

(5 marks)

(b) You are now informed that the `IntSet` trait is required only to store small integers in the range $[0 \ldots N)$ where N is a known fixed value. You are further informed that the `add` and `delete` operations should return a `Boolean` which indicates whether the operation has made a change to the set. Rewrite all parts of your answer to part (a) which should be changed because of this new information. (4 marks)

(c) One way to implement such a set is to use an array `a` of booleans such that `a(x)` is true precisely if `x` is in the set. This data structure is often called a *bit map*, since each element of the array could be a single bit.

Implement a class `BitMapSet` following this idea. Make all operations as efficient as possible. Write down a suitable abstraction function and datatype invariant.

(7 marks)

(d) Use `BitMapSet` to write a function

```
def sort(xs: Array[Int]) : Array[Int] = {
```

which sorts an array `xs` in linear time. Preconditions for this function are that all the elements of `xs` are distinct and in the range $[0 \ldots N)$. Your function should check these preconditions using the functionality of your `BitMapSet` class.

(4 marks)

Question 4

(a) Define a suitable class `Tree` for representing reference-linked binary trees with nodes labelled by integers. (2 marks)

(b) The *inorder traversal* of a binary tree visits all the nodes in the left subtree, followed by the root, followed by all the nodes in the right subtree. Write a recursive procedure that performs an inorder traversal on a binary tree, printing the label as it visits each node. (2 marks)

(c) A binary tree is *nearly balanced* if at each node, the size of the left subtree and the size of the right subtree differ by at most 1. Write a recursive function

```
def MakeTree(a: Array[Int], a:Int, b:Int) : Tree
```

that constructs a nearly balanced binary tree whose inorder traversal is given by $u[a\,..\,b)$ where $a \leq b$, and does so in a time proportional to $b - a$.

(4 marks)

(d) Write a procedure that prints an inorder traversal of a binary tree – as in part (b) – without using recursion. Your solution may destroy the tree as it prints it, but should use only a constant amount of additional storage in addition to the space taken up by the input tree. This precludes you from using Scala API collections such as Stacks or Queues.

[Hint: If the root of the tree is not the first element of the inorder traversal, then massage the tree until it is.] (8 marks)

(e) Show that your solution to part (d) runs in a time proportional to the size of the tree.

[Hint: Define the left-size of a non-empty tree as the size of its left subtree plus the left-size of its right subtree, and use this idea to formulate a bound function for each loop in your program.] (4 marks)

Question 5

This question concerns a program for counting the frequency of different words in a text. The words, together with their frequency counts, are to be held in a chained hash table with N buckets. This table is contained in a `HashBag` class.

```
class HashBag{
  private val N = 100   // # buckets in the hash table
  private val table = new Array[Node](N) // the hash table
  //...
}
```

(a) Declare a `Node` class suitable for representing linked lists of words, each associated with a frequency count. (2 marks)

(b) Assuming that `HashBag` has a suitable hash operation $hash(w)$ acting on words, give an appropriate variable declaration for the hash table, and show how to implement a `HashBag` operation $add(w)$ that inserts the word w into the hash table if necessary and increases its count by 1. (3 marks)

(c) After all the words have been added, the program outputs all the words with their frequency counts, in decreasing order of frequency. As a step towards this end, write an operation which removes all the entries from a given hash table bucket (`table(i)`) and returns a linked list of those nodes in the desired order. This operation should use insertion sort to arrange the new list of nodes. [*Hint: consider using a dummy list header.*] (4 marks)

(d) Write a function that takes two sorted lists of nodes and merges them into one sorted list. (4 marks)

(e) Using the operations you have written in earlier parts, write an operation that efficiently rearranges all the records from the hash table into a single sorted list by using insertion sort on each bucket, then merging the contents of different buckets together. (4 marks)

(f) If the number S of distinct words input to the program is allowed to become large, but with the ratio N/S remaining constant, how does the average time taken for the sorting grow as a function of S? How does the time grow for the worst case, and how does the worst case arise? (3 marks)

Imperative Programming Part 3

Question 6

In this question you will be asked to design and implement a small part of a proposed new software application called *SongMate* that arranges a collection of songs in a playlist.

The application should provide commands to insert a specified song at a given position in the playlist, to delete a song given its position in the playlist, and to move a song from one place to another in the playlist.

The application should also provide commands to undo one at a time any changes that have been made to the playlist, and later to redo them one at a time.

It has been decided to represent the playlist using an implementation of the following `Sequence` trait. The code for this implementation will be provided by another programmer.

```
trait Sequence[A] {
    def length: Int
    def elementAt(i: Int): A
    def insert(i: Int, x: A): Sequence[A]
    def delete(i: Int): Sequence[A]
}
```

(a) Define a class `SongSuite` that holds a sequence of `Song` objects and provides methods corresponding to the insert, delete and move commands in the *SongMate* application. You may assume without checking that the arguments to these methods are valid.

Each of these methods should return information about the change that has been made, so that this change can later be undone and redone via a standard interface. You should define whatever traits or classes are needed to represent this information. (8 marks)

(b) What are the main components of the *Command* design pattern? What role could your `SongSuite` class play in this design pattern? (4 marks)

(c) Instead of storing information about each specific change that is made, a colleague suggests that the undo mechanism should simply store `Sequence` objects corresponding to the state of the `SongSuite` before and after each command.

(i) Explain briefly how your `SongSuite` class could be modified to implement the colleague's suggestion.

(ii) Does this change have any significant advantages or disadvantages?

(8 marks)

Question 7

(a) What is *polymorphism*, and how does it arise in object-oriented programming?

(6 marks)

(b) Complete the following code skeleton to implement a generic Scala class that represents a partial function between arbitrary types:

```
class PartialFn[T,U](val data : List[(T,U)]) {
  // Represents a partial function f
  // mapping x to y for every pair (x,y) in data
  // (the pairs supplied in data are assumed to define a valid function)

  // If f(x) is defined and equal to y, then get(x) returns Some(y);
  // otherwise, it returns None.
  def get(x : T) : Option[U] =  ...

  // remove(x) returns a partial function which is equal to f
  // except that x is removed from the domain (if present)
  def remove(x:T) : PartialFn[T,U] = ...

  // add(x,y) returns a partial function which is equal to f
  // except that the value assigned to x is now defined to be y
  def add(x:T,y:U) : PartialFn[T,U] = ...
}
```

(5 marks)

(c) Now write a suitable `compose` method for your `PartialFn` class that returns the partial function obtained by composing this partial function with another partial function, "`that`", of any suitable type (the function `that` should be applied second). (4 marks)

(d) What does it mean to say that a type parameter of a generic class is *covariant*?

(2 marks)

(e) What changes would need to be made to the code you wrote in parts (b) and (c) to ensure that the second type parameter, `U`, of your `PartialFn` class was covariant? (3 marks)

Question 8

(a) Describe the *Observer* software design pattern. In what circumstances is it used, and what potential design flaw does it help to avoid? (5 marks)

(b) The `scala.swing` package provides the following traits:

```
trait Reactor {
  def listenTo(ps: Publisher*)
  def deafTo(ps: Publisher*)
  val reactions: Reactions
}

trait Publisher {
  def publish(e: Event)
}
```

Explain how these traits can be used to implement the Observer design pattern in `scala.swing` applications. (4 marks)

(c) Extend the following code so that when a valid number is typed into any of the editable fields the other fields are updated to show appropriate values.

```
import scala.swing._
import scala.swing.event.EditDone

object CurrencyConverter extends SimpleSwingApplication {
    def top = new MainFrame {

        val pounds  = new TextField(5); pounds.text = "%.2f".format(10.0)
        val euros   = new TextField(6); euros.text = "%.2f".format(11.5)
        val rate    = new TextField(6); rate.text = "%.2f".format(1.15)

        contents = new FlowPanel {
            contents += (pounds, new Label(" Pounds    = "), euros)
            contents += new Label(" Euros at ")
            contents += (rate, new Label(" Euros to the Pound"))
        }
    }
}
```

(5 marks)

(d) Describe the *model-view-controller* software architecture. What are the main components of a software system with this architecture and what are their responsibilities? How do they interact with each other and with the user? How does the model-view-controller architecture support good design principles? How does it relate to the Observer design pattern? (6 marks)

LAST PAGE