The University of Nottingham Ningbo China

SCHOOL OF COMPUTER SCIENCE

A LEVEL 1 MODULE, SPRING SEMESTER 2021-22

PROGRAMMING PARADIGMS
SOLUTIONS

Time allowed: **TWO Hours THIRTY Minutes**

Candidates may complete the front cover of their answer book and sign their desk card but must NOT write anything else until the start of the examination period is announced

Answer ALL questions.
(Each question is worth equal marks)

Only silent, self-contained calculators with a Single-Line Display are permitted in this examination.

Dictionaries are not allowed with one exception. Those whose first language is not English may use a standard translation dictionary to translate between that language and English provided that neither language is the subject of this examination. Subject specific translation dictionaries are not permitted.

No electronic devices capable of storing and retrieving text, including electronic dictionaries, may be used.

DO NOT turn your examination paper over until instructed to do so

ADDITIONAL MATERIAL: Haskell standard prelude.

INFORMATION FOR INVIGILATORS: Exam papers must be collected at the end of the exam.

1. Object-Oriented Programming / Java / Programming paradigms (25 marks)

(a) Which of these statements are true and which are false? **[4 marks]**

(i) An instance can be created from an abstract class.
(ii) Multiple inheritance of interfaces is possible.
(iii) A protected method in a class A can be accessed by a subclass of A.
(iv) All classes and interfaces in Java are subclasses of the `Object` class.

(b) Describe the four key principles of Object Oriented Programming. **[8 marks]**

(c) Describe these two approaches to polymorphism in Java: Overloading and Overriding. Give an example of each. **[8 marks]**

(d) Describe five key contrasts between these two pieces of code written in Java and Haskell. **[5 marks]**

Java	`int[] x = new int[10];`
	`x[0] = 1;`
	`for (int i=0; i<x.length-1; i++)`
	`  x[i+1] = x[i]*2;`
Haskell	`map (2^) [0..9]`

Note that `^` is an operator such that `a^b` returns `a` to the power `b` in Haskell.

2. Object-Oriented Programming/Java (25 marks)

Consider the Java code at the end of this exam paper that implements a class `IntList` for lists of integers.

(a) What is the output from executing the following sequence of code:-

```
(i)  IntList x1 = new IntList(4);
     System.out.println(String.valueOf(x1.getValue() *
                                     x1.next().getValue()));

(ii) System.out.println(x1.toString(10));

(iii) IntList x2 = new IntList(0);
      IntList x3 = x2.next().next().next();
      x3.setValue(5);
      x3.next().next().setValue(8);
      System.out.println(x2.toString(8));

(iv) System.out.println(x3.previous().toString(2));
```

[8 marks]

(b) Provide the code for a subclass `IntListInc` of `IntList` using the `extends` keyword that changes the behaviour of the list so that each next element has an initial value that increments from the previous element by a given increment value. You should include a constructor

```
IntListInc(int value, int inc)
```

that creates a new list starting with `value` and increments on each node by `inc`. So, for example, the code

```
IntList x4 = new IntListInc(0,3);
System.out.println(x4.toString(5));
```

will output

```
0, 3, 6, 9, 12
```

Hint: You may store the increment value at each element of the list. **[8 marks]**

(c) A programmer decides that the code for `toString` can be written without the need for the parameter `int n` and writes the following code that they include as an additional method for the class `IntList`:

```
public String toString()
{
     String sv = String.valueOf(value);
     if  (next()!=null)
     {
          sv = sv + ", " + next().toString();
     }
     return sv;
}
```

(i) This new method has the same name as the existing `toString` method in `IntList`. Why does this *not* produce a syntax error? **[2 marks]**

(ii) What will happen if the following code is run:-

```
System.out.println(x1.toString());
```

[2 marks]

(d) Write code using a `for` loop that takes an array `int[] ns` and will construct a new `IntList x5` object whose initial elements are populated with the values in `ns`, and then followed by zeros.
So, for example, if `ns = {5,2,6,7}`, after running your code,

```
System.out.println(x5.toString(6) );
```

will output

```
5, 2, 6, 7, 0, 0
```

[5 marks]

3. Functional Programming / Haskell (25 Marks)

(a) Consider each of these function definitions. In each case identify the one syntax error and how it can be fixed. **[10 marks]**

(i)
```
F x y = (x+4) * x * (y-2)
```

(ii)
```
f x -> (\y -> x*(y+2) )
```

(iii)
```
g x y | x>y = x
      | otherwise y
```

(iv)
```
h x (x:xs) = xs
h _ xs     = xs
```

(v)
```
j = foldr + 2
```

(b) Consider the following function definition:

```
indexList :: [a] -> [(Int,a)]
indexList = zip [0..]
```

(i) Explain how currying is being used in this function definition? **[2 marks]**

(ii) Explain why this function is polymorphic, and illustrate with examples. **[2 marks]**

(iii) What is the output of `indexList "cat"` ? **[2 marks]**

(iv) How would you modify the definition so that `indexList` only operates on lists of numeric types. **[2 marks]**

(c) Write a function `areUnique` **using recursion** that will check that all elements in a list are unique, returning `True` if they are and `False` otherwise. For example,

`areUnique [1,3,2]`	returns	`True`
`areUnique [1,3,2,3]`	returns	`False`
`areUnique [5,2,2,5]`	returns	`False`
`areUnique [1,3,5,2]`	returns	`True`
`areUnique "dog"`	returns	`True`
`areUnique "xxx"`	returns	`False`
`areUnique []`	returns	`True`

Remember to include its type definition. **[7 marks]**

Hint: You may use functions `not` and `elem` from Standard Prelude to answer this question.

4. Functional Programming / Haskell (25 Marks)

Consider writing a Domain-specific language for matrix manipulation. We use the following data type to represent any matrix of floating-point numbers:

```
data Matrix = Row [Float] Matrix | Null
```

That is, a matrix is represented as a sequence of rows given as lists of floating-point numbers. So, for example, the matrix

$$\begin{pmatrix} 2.3 & 4.5 & 1.2 \\ -0.4 & 3.2 & 3.4 \end{pmatrix}$$

is represented in the following way:-

```
m1 :: Matrix
m1 = Row [2.3, 4.5, 1.2] (Row [-0.4, 3.2, 3.4] Null)
```

Notice that the `Null` constructor just marks the end of the rows in a matrix.

In this question you may use standard Prelude and IO type, but no other Haskell libraries. Ensure that type definitions are given for all functions you write.

(a) Represent the following matrix using the data type `Matrix` and assign to variable `m2`. **[2 marks]**

$$\begin{pmatrix} 1.2 & -1.0 \\ 4.5 & -0.9 \\ 2.3 & 1.8 \end{pmatrix}$$

(b) We will say that a matrix is legitimate if and only if all rows have exactly the same length.
Write a function `legitMatrix :: Matrix -> Bool` to check if a matrix is legitimate, returning `True` if it is, and `False` otherwise. **[3 marks]**

(c) Write a function `sizeMatrix` to calculate the size of a matrix assuming it is legitimate, returning a pair (x,y) where x is number of rows and y is number of columns. **[2 marks]**

(d) Write a function `insertColumn` to insert a list of floating-point numbers as a column to the left of an existing matrix.

For example, `insertColumn m1 [8.7, -2.5]` will return the matrix

$$\begin{pmatrix} 8.7 & 2.3 & 4.5 & 1.2 \\ -2.5 & -0.4 & 3.2 & 3.4 \end{pmatrix}$$

[2 marks]

(e) Consider this function:

```
        emptyMatrix :: Int -> Matrix
        emptyMatrix 0 = Null
        emptyMatrix n = Row [] (emptyMatrix (n-1))
```

What does `emptyMatrix 3` return? **[1 mark]**

(f) Write a function `transposeMatrix` to implement the transpose of a matrix. **[3 marks]**

Hint: Use `insertColumn` and `emptyMatrix` functions for this question part.
Remember: the transpose of a matrix swaps rows for columns. For example, the transpose of `m1` will be

$$\begin{pmatrix} 2.3 & -0.4 \\ 4.5 & 3.2 \\ 1.2 & 3.4 \end{pmatrix}$$

(g) Write a function `addRows` to add together consecutive elements of two rows to return a new row, using list comprehension and the `zip` function, assuming they are the same length.

For example, `addRows [3.4, 6.7, 1.2] [-0.5, 1.0, -1.2]` will return `[2.9, 7.7, 0.0]`. **[3 marks]**

(h) Write an operator `+#+` to add two matrices together, using normal matrix addition.
You may use the `addRows` function to do this. **[2 marks]**

(i) We know that matrix addition is mathematically only correct if the two matrices being added are the same size. Write a safe operator `+##+` to add two matrices together, but using the `Maybe` data type, so that it returns `Nothing` in the case that either of the two matrices not being legitimate (see part (b) above) or they are not the same size. **[3 marks]**

Hint: You may use the `legitMatrix` and `sizeMatrix` functions and `+#+` operator already defined.

(j) Use the IO type to write a function `showMatrix` to display a matrix in the following way, with tabs between columns. For example,

```
showMatrix m1
```

writes to output:

```
[2.3    4.5     1.2]
[-0.4   3.2     3.4]
```

```
showMatrix m2
```

writes to output:

```
[1.2    -1.0]
[4.5    -0.9]
[2.3    1.8]
```

[4 marks]

Hint 1: You may use one or more auxiliary functions.
Hint 2: `show n` converts a number `n` to `String`, and `'\t'` is a tab character.

[THIS PAGE IS INTENTIONALLY LEFT BLANK]

This Java code is required for Question 2. You may detach it from the exam paper.

```
public class IntList
{
     protected IntList next;
     protected IntList prev;
     protected int value;

     public IntList(int value)
     {
          prev = null;
          this.value = value;
     }

     public int getValue()
     {
          return value;
     }

     public void setValue(int value)
     {
          this.value = value;
     }

     public IntList next()
     {
          if  ( next == null )
          {
               next = new IntList(value);
               next.prev = this;
          }
          return next;
     }

     public IntList previous()
     {
          return prev;
     }

     public String toString(int n)
     {
          String sv = String.valueOf(value);
          if  (n>1)
          {
               sv = sv + ", " + next().toString(n-1);
          }
          return sv;
     }
}
```

[END OF EXAM PAPER]