The University of Nottingham Ningbo China

SCHOOL OF COMPUTER SCIENCE

A LEVEL 1 MODULE, SPRING SEMESTER 2022-23

PROGRAMMING PARADIGMS

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)

Consider this Java code and answer the following questions.

```
public class SumList{
      public SumList prev;
      public SumList next;
      public int value;

      public SumList(int value){
            this.value = value;
      }

      public SumList next(){
            if(next == null){
                  int sum = value;
                  if(prev != null)
                        sum += prev.value;
                  next = new SumList(sum);
                  next.prev = this;
            }
            return next;
      }

      public String toString(int n){
            String s1 = value + "";
            if(n > 1){
                  s1 = s1 + "; " + next().toString(n-1);
            }
            return s1;
      }
}
```

(a) What is the output of the following Java statements: **[6 marks]**

(i)
```
SumList l1 = new SumList(2);
System.out.println(l1.toString(5));
```

(ii)
```
SumList l2 = new SumList(3);
SumList l3 = l2.next().next();
l3.value = 7;
System.out.println(l3.prev.toString(6));
```

(iii)
```
System.out.println(l1.next().value * l3.next().next().value);
```

(b) The method `String toString(int n)` in the given Java code is defined recursively. Write an iterative method `toString` which also takes an int variable as input and generates the same output as before. **[4 marks]**

(c) You are asked to write a `void setValue(int n)` method to update the value of the current object and all its successors (i.e., its next object, the next object of its next object and so on) as follows:

- The value of the current object is set to n.
- If the current object has a previous object o_i (i.e., `this.prev`), then the connection between o_i and the current object should be removed.
- If the current object has a next object o_j, then the value of o_j and all its successors are also updated based on the computational logic defined in the method `next()`.
- You are not allowed to create any new object in the `setValue` method.

Suppose we have the following `SumList` objects, where o_{n+1} is the next object of o_n.

o_1	o_2	o_3	o_4	o_5
2	2	4	6	10

If we call o_3`.setValue(5)`, then the list of `SumList` objects will be updated as follows. o_3 is no longer the next object of o_2. However, all other relationships are maintained as before, e.g., o_2 is still the next object of o_1, and o_4 is still the next object of o_3. However, o_2 is no longer the previous object of o_3 and o_3 is also not the next object of o_2, as follows. **[10 marks]**

o_3	o_4	o_5
5	5	10

o_1	o_2
2	2

(d) A programmer decides that the code for `toString` can be written without the need for the parameter int n and write the following code as an additional method for the class `SumList`;

```
public String toString(){
      String s1 = value + "";
      if(next() != null){
            s1 = s1 + "; " + next.toString();
      }
      return s1;
}
```

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

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

```
System.out.println(l1.toString());
```

[3 marks]

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

In computer science, a tree is a widely used abstract data structure for representing hierarchical data. Each node in the tree can have several children but is constrained to have exactly one parent except the root node.

(a) Binary trees are a commonly used tree type, in which each node can have at most two children. Write a data declaration for binary tree in Haskell. **[2 marks]**

(b) Given the abstract class `AbstractTreeNode` for integer tree nodes, write a Java class `MyBTreeNode` for binary tree nodes. `MyBTreeNode` should be a concrete Java class that extends `AbstractTreeNode`. Make sure that your `MyBTreeNode` can have at most 2 children. **[8 marks]**

```
public abstract class AbstractTreeNode{
      int value;
      AbstractTreeNode parent;

      public AbstractTreeNode(int value){
            this.value = value;
      }

      public int getValue(){
            return this.value;
      }

      public AbstractTreeNode getParent(){
            return this.parent;
      }
      // return the list of child nodes
      public abstract AbstractTreeNode[] getChildren();
}
```

(c) A developer has added the following method to the abstract class `AbstractTreeNode`:

```
public boolean addChild(AbstractTreeNode node);
```

This abstract method specifies that all the subclass of `AbstractTreeNode` should have a method `addChild` which adds a given `AbstractTreeNode` as the child of the current node. The method fails and returns false, if the given node type is not compatible with the current node or the node cannot be added. Otherwise, the method returns true and the input node should be added successfully. Please revise your code of `MyBTreeNode` to ensure it works properly. Note, you only need to provide the implementation of the new methods. **[5 marks]**

(d) You are asked to write a Java method and a Haskell function to check if a given value exists in an integer binary tree. Assuming your Java method is defined in myBTreeNode and your Haskell code is based on your definition in 2(a). Your code should follow the following definition and traverse the tree in a breadth first manner. **[10 marks]**

```
public boolean hasValue(int n);

hasValue :: Tree Int -> Int -> Bool
```

3. Contrasting Java and Haskell / Java classes / Haskell data declarations (25 marks)

For this question, only use Haskell and in-build functions from *Haskell Standard Prelude* to write Haskell code.

(a) Consider this Java code that outputs a sequence of numbers.

```
public static void main(String args[])
{
  int x = 0;
  for (int i = 1; i <= 10; i++) {
     x+=i;
     if(i>1) System.out.print(",");
     System.out.print(x);
  }
  System.out.println("");
}
```

Write Haskell code using list comprehension to perform the same task and output the sequence of numbers as a list. Do not use recursion. **[6 marks]**

*Note: You do **not** need to use* IO *to output to the terminal, just return a list.*

(b) Explain the difference between an imperative and a declarative style of programming. Give a simple example. **[4 marks]**

(c) What is the difference between variable assignment in Haskell and Java. **[2 marks]**

(d) Consider the Java and Haskell code at the end of this exam paper about different Animals. The two pieces of code are intended to implement the same functionality.

(i) In the Haskell code, what type of symbol is `Dog`? Is it a lambda name, a constructor, or a function? **[1 mark]**

(ii) Compare and contrast these two pieces of code in terms of programming style. Write your answer in no more than 6 bullet points.
*Do **not** give contrast in terms of syntax, only programming style.* **[6 marks]**

(e) Consider the Haskell function:

```
noises a1 = concat (map says a1)
```

(i) What is the type of `noises`? **[2 marks]**

(ii) Write a method in Java that will do the same as `noises`. Use a Java array instead of a list. **[4 marks]**

4. Functional Programming / Haskell (25 Marks)

For this question, only use Haskell and in-build functions from *Haskell Standard Prelude* to write Haskell code.

(a) What is the type of each of these expressions? **[4 marks]**

(i) `"123abc" ++ "xyz"`
(ii) `head [ (True, "Grit"), (False, "Dawn") ]`
(iii) `fst (True, 23)`
(iv) `map (:[]) ['a', 'b', 'c']`

(b) Evaluate each of these expressions. **[4 marks]**

(i) `'X':"banana" ++ "apple"`
(ii) `12 `div` head [3,4]`
(iii) `filter (<3) [1,4,0,3,2,1]`
(iv) `foldr (-) 0 [3,2,8,5]`

(c) Evaluate each of these lambda expressions. Show each step of the reduction. **[6 marks]**

(i) `(\x -> x+x) 3`
(ii) `(\x -> (\y -> x++" "++y++" "++reverse(x) )) "doc" "eats"`
(iii) `(\x -> (\f -> f (f x) -12)) 23 (\x -> x*2)`

(d) What is meant by a higher-order function? **[1 mark]**

(e) Describe clearly what the following code does. **[5 marks]**

```
checkWords :: IO()
checkWords =
  do putStr "Enter a word: "
     x <- getLine
     if  x == "quit" then
       return ()
     else
       do if x == reverse(x) then
            putStrLn (x ++ " is a palindrome!")
          else
            putStrLn (x ++ " is not a palindrome.")
          checkWords
```

(f) Consider the type

```
type PersonValue = (String, Int)
```

which represents that a person has a certain value of money, in CNY. So, for example, `("Sue",34)` means Sue has 34CNY.

Use recursion to write a function called `sumName` that takes a list of `PersonValue` and a name, and returns the sum of values that the named person has in that list.
For example, given the list of `PersonValue`,

```
db1 = [("Mary", 100), ("John", 50), ("Mary", 25),
       ("Yifan", 75), ("John", 10), ("Mary", 12)]
```

then `sumName db1 "Mary"` will return `137`.
Remember to include the type of the function. **[5 marks]**

This page is intentionally blank.

This Java and Haskell code on this page and next is required for Question 3. You may detach it from the exam paper.

```
public abstract class Animal
{
  private String colour;

  public Animal(String c) {
       colour = c;
  }

  public abstract String says();
  public String colour() { return colour; }
  public abstract boolean chases(Animal a2);
}

class Mouse extends Animal
{
  public Mouse(String c) { super(c);}
  public String says() { return "Squeak"; }
  public boolean chases(Animal a2) { return false; }
}

class Dog extends Animal
{
  public Dog(String c) { super(c);}
  public String says() { return "Woof"; }
  public boolean chases(Animal a2) {
    return a2 instanceof Cat;
  }
}

class Cat extends Animal
{
  public Cat(String c) { super(c);}
  public String says() { return "Meow"; }
  public boolean chases(Animal a2) {
    return a2 instanceof Mouse;
  }
}
```

```
data Colour = Brown | White | Black
data Animal = Dog Colour | Cat Colour | Mouse Colour

says :: Animal -> String
says (Dog _) = "Woof"
says (Cat _) = "Meow"
says (Mouse _) = "Squeak"

colour :: Animal -> Colour
colour (Dog c) = c
colour (Cat c) = c
colour (Mouse c) = c

chases :: Animal -> Animal -> Bool
chases (Dog _) (Cat _) = True
chases (Cat _) (Mouse _) = True
chases _ _ = False
```

[END OF EXAM PAPER]