2006 A SEMESTER EXAMINATIONS

DEPARTMENT	Computer Science
PAPER TITLE	Programming Languages
TIME ALLOWED	Three Hours
NUMBER OF QUESTIONS IN PAPER	Seven
NUMBER OF QUESTIONS TO BE ANSWERED	Seven
VALUE OF EACH QUESTION	As indicated
GENERAL INSTRUCTIONS	Nil
SPECIAL INSTRUCTIONS	Nil
CALCULATORS PERMITTED	No

1. Ruby: (a) Summarise in 4-5 sentences what the main differences are between Ruby and a language like Java or C++.

[7 Marks]

(b) For each of the three following expressions, determine their result:

```
(1…10).find_all { |x| x>3}

[“a”,”b”,”c”].inject { |x,y| x+y }

[1,2,3,4,5,6,7].find { |x| x*x > 20 }
```

[6 marks]

(c) Define a Ruby function allTriplets(), that computes all solutions for following problem and returns all solutions in an array: X, Y, and Z are all digits from 0 to 9 (inclusive), X, Y, and Z are different from each other, and the following integer equation holds: (10*X + Y) / (10*Y + Z) = X / Z.

[7 Marks]

2. GC algorithms: Select four of the GC algorithms listed below and a) briefly describe how they work, as well as b) discuss advantages and disadvantages of each of the four algorithms you have selected:

reference counting

mark-and-sweep

mark-and-compact

copying GC

generational GC

[20 Marks]

3. Java: (a) The following code example exercises overloaded methods. What output does this program produce?

[8 Marks]

```
class A {
  public void f(Object o) { System.out.println("object"); }
  public void f(A a) { System.out.println("an a");}
}
class B extends A {
  public void f(B b) { System.out.println("a b"); }
}
class Test {
  public static void main(String[] args) {
    Object o = new A();
    A a = (A) o;
    Object o1 = new B();
    B b = (B) o1;
    a.f(a);
    a.f(b);
    a.f(o);
    a.f(o1);
    b.f(a);
    b.f(b);
    b.f(o);
    b.f(o1);
  }
}
```

(b) List and explain four constraints a user-defined equals-method must obey. Additionally, if a class implements its own specialised equals-method, does it also have to supply an appropriate a) a specialised hashCode-method and/or b) a specialised compareTo-method?

[12 Marks]

4.

(a) Say what types, in Haskell, the following expressions have:

(i) `(+)`
(ii) `(1+)`
(iii) `(1+2)`
(iv) `map (1+) [1,2,3]`
(v) `map (1+)`
(vi) `map (\x -> ('a',x)) [1,2,3]`

[6 marks]

(b) Say what values, in Haskell, the following expressions have:

(i) `map (1+) [1,2,3]`
(ii) `map (\x -> ('a',x)) [1,2,3]`
(iii)
```
f (Rectangle 6.0 7.0)  1.0  2.0
      where f s dx dy =
            case s of
              RtTriangle s1 s2 = RtTriangle s1*dx  s2*dy
              Rectangle s1 s2  = Rectangle s1*dx  s2*dy
              Ellipse r1 r2    = Ellipse r1*dx  r2*dy
```

given the data declaration

(Question 4 continued next page)

```
data Shape = Rectangle Float Float
           | Ellipse Float Float
           | RtTriangle Float Float
```

[4 marks]

5. In Haskell, the type constructor `Tree` defined by

```
data Tree a = Leaf a
            | Node a (Tree a) (Tree a)
```

can be used to represent binary trees with data at internal nodes and leaves.

(a) Define a function

```
sumLeaves :: Tree Integer -> Integer
```

which adds-up the data values **just** at the leaves

[4 marks]

(b) Define a function

```
sumNodes :: Tree Integer -> Integer
```

which adds-up the data values **just** at the internal (non-leaf) nodes

[4 marks]

(c) Define a function

```
sumTree :: Tree Integer -> Integer
```

which adds-up the data values at all nodes in a tree, and **do not** use recursion in your definition.

[4 marks]

6. Given the Haskell definition

```
x = x + 1
```

(a) Show three steps in the evaluation of the expression

```
x
```

[1 mark]

(b) What value does `x` have?

[1 mark]

(c) If a function f is strict, what value does

```
f ⊥
```

have?

[1 mark]

7. For Haskell,

(a) Prove that, for any list `xs`,

```
xs ++ [] = [] ++ xs
```

[4 marks]

(b) Prove that, for any lists `xs` and `ys`,

```
length (xs ++ ys) = length xs + length ys
```

[4 marks]

(c) Prove that

```
sumList . map (2*) = (2*) . sumList
```

where

```
sumList :: [Integer] -> Integer
sumList []     = 0
sumList (x:xs) = x + sumList xs
```

[7 marks]

Definitions of Haskell prelude-defined functions needed in the paper:

```
map           :: (a -> b) -> [a] -> [b]
map f []      =  []
map f (x:xs)  =  f x : map f xs

(++)          :: [a] -> [a] -> [a]
[] ++ ys      =  ys
(x:xs) ++ ys  =  x : (xs ++ ys)

length         :: [a] -> Integer
length []      =  0
Length (x:xs)  =  1 + length xs
```

(Question 4 continued next page)