Final Exam INF 311

Originally proposed by P. Chassignet, D. Monniaux, F. Nielsen and O. Serre
Translated in english by F. Nielsen

7th July 2008

– Only lecture notes and slides (with personal annotations) are authorized.
– Note that english translation is provided here to help foreign students fully understand the *official* french exam. Please always refer to the latter one for cross-checking the *exact* meaning of questions.
– Time allowed : 2 hours (bonus of 30 minutes for EV2 students).

The following 4 exercises are independent of each other, and can thus be answered in any order. For each question, we provide the expected number of lines of a *typical* answer ; If your answer is comparatively much longer, it is likely that your method is too much complicated.
More difficult questions are indicated using the following mark : **.

We particularly appreciate precise and concise explanations in the remainder.

Exercice 1. Mysterious recursive function

We consider the following program that compiles without any error :

```
public class MysteriousProgram {
  public static void display(int[] tab) {
    for (int i = 0; i < tab.length; i++)
      System.out.print(tab[i] + "␣");
    System.out.println();
  }
  public static void swap2(int a, int b) {
    int tmp = a;
    a = b;
    b = tmp;
  }
  public static void swap3(int[] tab, int i, int j) {
    int tmp = tab[i];
    tab[i] = tab[j];
    tab[j] = tmp;
  }

  public static void mysterious(int[] tab, int k) {
    for (int j = k; j < tab.length; j++) {
      swap3(tab, k, j);
```

```
      display(tab);
      swap3(tab, k, j);
    }
  }

  public static void mysteriousRecursive(int[] tab, int k) {
    if (k == tab.length - 1)
      display(tab);
    for (int j = k; j < tab.length; j++) {
      swap3(tab, k, j);
      mysteriousRecursive(tab, k + 1);
      swap3(tab, k, j);
    }
  }

  public static void init(int[] tab) {
    for (int i = 0; i < tab.length; i++)
      tab[i] = i + 1;
  }

  public static void main(String[] args) {
    int n = Integer.parseInt(args[0]);
    int[] t = new int[n];
    init(t);
    swap2(t[0], t[n - 1]);
    mysterious(t, 0);
    //  mysteriousRecursive(t, 0);
  }

}
```

(1a) Once this code compiled, what is the result displayed in the output console by invoking `java MysteriousProgram 4`? (Answer length : a few lines)
Describe explicitly the program execution steps and the outcome for the general case (for any given `n > 0`) (Answer length : 10 to 20 lines)

(1b) ** In the function `main`, we now replace the function call `mysterious(t, 0);` by `mysteriousRecursive(t, 0);`.
Once the program recompiled, what is the console result obtained by launching `java MysteriousProgram 3`?
(Answer length : a few lines)
Describe precisely the various steps of the program execution, and the obtained result in the general case (that is, for any given `n > 0`) (Answer length : 10 to 20 lines)

Exercice 2. Modeling molecules

In this exercise, we are first concerned with modeling molecules as arrays of atoms, where

each atom is defined as a proper 3D sphere with a center and a radius (the Van der Waals radius). We will then study how to detect whether atoms and molecules collide or not.

(2a) Design a class called `Point3D` where each object is defined as a 3D point with coordinates x, y and z, all of type `double`. Further, provide this class with a constructor `Point3D(double x0, double y0, double z0)` that allows to initialize a `Point3D` object. (Answer length : 5 to 10 lines)

(2b) Write a static function `double distance(Point3D p, Point3D q)` that takes as arguments two points p and q, and returns the Euclidean distance $\|q - p\|$ between them. In order to compute the square function, we will use function `static double sqr(double x) { return x*x; }` that is inserted also inside the body of class `Point3D`. To compute the square root, we'll make use of function `static double sqrt(double x)` of the `Math` class. (Answer length : a few lines)

(2c) Give a static function `Point3D add(Point3D p, Point3D q)` that takes as arguments two points p and q, and return a **new** point equal to $p + q$. This function shall be inserted inside class `Point3D`. (Answer length : a few lines)

(2d) Give a static function `void scale(Point3D p, double k)` that multiplies the coordinates of point p by scalar number k. That is, p becomes $k.p$. This function shall be located inside class `Point3D` as well. (Answer length : a few lines)

(2e) Provide a class `Atom` that allows to define atoms with **two** object fields :
- `center` of type `Point3D` denoting the location (x, y, z) of the center of this atom, and
- `radius` of type `double` that encodes the radius of this atom.

Add a constructor `Atom(double x, double y, double z, double rad)` to this class that initialize properly objects of this type.

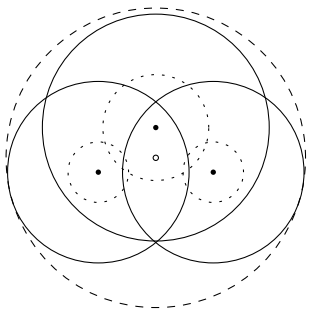
Furthermore, add to this class two constants `H_RADIUS = 1.2` and `O_RADIUS = 1.5` that represent the radii in ångström for the hydrogen and oxygen atoms, respectively.
(Answer length : a dozen lines)

(2f) Write a static function `boolean bump(Atom a, Atom b)` that takes as arguments two atoms a and b, and return `true` if and only if the distance between their centers is strictly less than the sum of their radii (this will represent a collision between two atoms). This function shall be defined inside the class `Atom`. (Answer length : a few lines)

(2g) Write a class `Molecule` that allows to define a 3D molecule as an array of atoms, and provide the class with a constructor that takes as argument a reference to this array. (Answer length : a few lines)

(2h) Using a new class `Test`, write a program that builds a water molecule H_20 with its oxygen atom located at $(0, 0.4, 0)$ and the two hydrogen atoms located at $(0.76, -0.19, 0)$ and $(-0.76, -0.19, 0)$ (units still being ångström). (Answer length : 5 to 10 lines)

For each molecule, we are now going to build an enclosing ball that will allows one to speed-up test for detecting potential collisions. To simplify, assume the center of that enclosing sphere is set as the centroid of atoms (barycenter with uniform weight). That is, we do not take into account respective atom masses. See figure below. We shall use object `Atom` to represent such an enclosing ball.



• denote atom centers. ∘ depicts the centroid (barycenter with uniform weight).
Plain circles denote Van der Walls spheres of respective atoms.

The enclosing sphere of a water molecule.

(2i) Write a static function `middle` that takes as argument an array of atoms (assumed not empty) and that returns the centroid of these atoms (barycenter with uniform weights). This function shall be inserted in the `Atom` class, and will use functions `add` and `scale` of class `Point3D`. This function shall not modify the atom coordinates of the array. (Answer length : 5 to 10 lines)

(2j) Write a static function `double maxDistance(Point3D p, Atom a)` that returns the maximal distance between point p and any point on the sphere of atom a. This function shall be attached inside class `Atom`. (Answer length : a few lines)

(2k) ** Let us describe now how to modify class `Molecule` for building its enclosing sphere and using it for checking for collisions. Note that if we do not intersect the enclosing ball of a molecule, it is not necessary to check for collisions of its atoms.

Write a static function `boolean bump(Atom a, Molecule b)` of class `Molecule` that allows to check whether atom a is colliding with at least one of the atoms of molecule b or not.

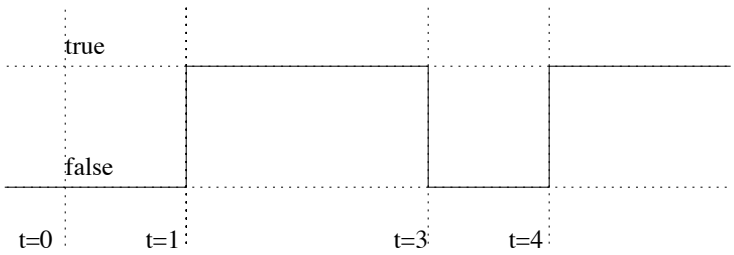
Write a static function `boolean bump(Molecule a, Molecule b)` that extends this test to two molecules. (Answer length : 20 to 30 lines)

Exercice 3. Coding electrical signals

We are interested in modeling binary electrical signals with values ranging in set $\{\mathsf{false}, \mathsf{true}\}$, that only change a finite number of times. These changes occur only at clock ticks. Thus these events can be modelled as positive integers that encode the numbers of clock ticks since the starting time (that is, the time origin).

Such a signal is defined by its initial value v_0, and a sequence of strictly increasing transition states $\tau_1, \ldots, \tau_n \in \mathbb{N}$. In $]-\infty, \tau_1[$, the signal has value v_0. In $[\tau_1, \tau_2[$, the signal takes value $v_1 = \neg v_0$ (where $\neg x$ denotes the logical negate function of x, written as `!x` in Java). In $[\tau_2, \tau_3[$, the signal value is $v_2 = \neg v_1$, and so on (with $v_n = \neg v_{n-1}$ in time range $[\tau_n, +\infty[$). A constant signal shall be represented by its value v_0 and an empty sequence ($n = 0$).

For example, the depicted signal below corresponds to the initial value false and the following sequence of transitions : $\tau_1 = 1, \tau_2 = 3, \tau_3 = 4$.



In the remainder, we shall use the following data structures :

```
class Transition {
  int time;
  Transition next;

  Transition(int time, Transition next) {
    this.time = time;
    this.next = next;
  }
}

class Signal {
  boolean initialValue;
  Transition transitions;

  Signal(boolean value, Transition transitions) {
    this.initialValue = value;
    this.transitions = transitions;
  }
}
```

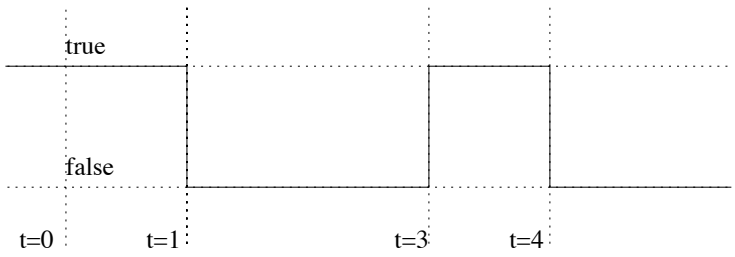
Field `initialValue` denotes the object value of type `Signal` at about $-\infty$. Field `transitions` denotes the beginning of the linked list of transition events for this signal. If that field is set to `null`, this means that the signal is constant. Otherwise, fields `time` of successive elements of the linked list indicate the respective change values $\tau_1, \ldots, \tau_n$, with τ_1 being the head. We insist on the fact that the sequence τ_i is strictly increasing.

For example, we can create the signal depicted above as :

```
Signal signal1 = new Signal(false,
  new Transition(1, new Transition(3, new Transition(4, null))));
```

In the following questions, the functions shall all be located inside the body of a class, whose name can be arbitrarily chosen.

(3a) Given a signal s, we first wish to invert it : that is, to obtain is logical negation. For example, the invert of `signal1` is :



Write a function `static Signal invert(Signal s)` performing this process. Your function shall not modify the source signal. That is, you need to create a **new** `Signal`. The time complexity of this function should not depend on the length of the transition states in `s`. (Answer length : a few lines)

(3b) Given a signal s and time tick t, we want to compute $s(t)$ that is the value (true or false) of signal s at time t. Write an **iterative** function `static boolean valueAt(Signal s, int time)` performing this task. You shall not modify signal `s`, and the time complexity should be linear with the number of transition states in `s`. (Answer length : a dozen lines)

(3c) We would like a function that displays a signal as a sequence of intervals with respective values. For example,

```
-inf -> +inf : false
```

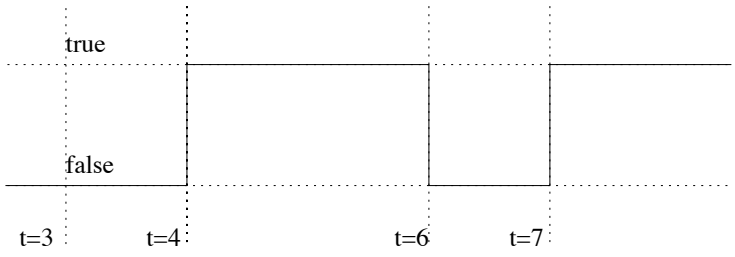
means that the signal takes uniformly value false. For `signal1`, the display function shall produce the following output :

```
-inf -> 1.0 : false
1.0 -> 3.0 : true
3.0 -> 4.0 : false
4.0 -> +inf : true
```

Write such an **iterative** function `static void print(Signal s)`. (Answer length : 10 to 15 lines)

(3d) Given a signal s, we would now like to produce another signal identical to s but shifted in time by a step $\delta > 0$. That is, we want a signal with the same initial value but with transition states (τ'_i) defined by $\tau'_i = \tau_i + \delta$, $\forall i \in \{1, \ldots, n\}$.

To illustrate this operation, the shifted signal of `signal1` with $\delta = 3$ produces the following result :



Write a **recursive** function `static Transition shift(Transition t, int delta)` that shall return a **new** list of transition states, identical to `t` but with values

shifted by `delta`. You shall not modify the original list `t`. Then give a function `static Signal shift(Signal s, int delta)`.

(Answer length : a dozen lines)

(3e) A `Signal` object that has not a strictly increasing sequence of transition states is not correct by definition. In order to detect program errors, it is useful to have a function `static boolean isWellFormed(Signal s)` that returns `true` if and only if `s` is correct. Write a **recursive** function using an auxiliary function. Time complexity shall be linear to the list length of transition events stored in `s`. (Answer length : a dozen lines)

(3f) ** The commutable exclusive or operation (XOR for short) that takes two operands is defined by the following logic table :

XOR	false	true
false	false	true
true	true	false

That is, b_1 XOR b_2 is true if and only if $b_1 = \neg b_2$. In Java, it can be written as `b1 != b2`.

We now want to compute the output signal of the XOR of two input signals. We notice that at a given time step, if only one signal change then the result change, and if both signal change at the same time, then the result does not change.

Write a function `static Transition xorTransitions(Transition t1, Transition t2)` that returns a **new** transition state list that corresponds to the output. This function shall not modify the source lists `t1` and `t2`, but in some cases, it is possible to share a sub-list. The time complexity shall be linear to the length of both lists.

Write now a function `static Signal xorSignals(Signal s1, Signal s2)`. (Answer length : about twenty lines)

Exercice 4. Nim game

Nim game is a game for two players. We consider m bins (indexed from 0 to $m-1$), with bin i containing x_i fruits. A game configuration is therefore represented by an array of integers.
The two players are playing in turn. A player move consists in taking as many as wished (but at least one) fruit(s) in a **same** bin. The winner is the player that takes the last remaining fruit (hence all bins become empty).
It turns out that the outcome of this game is fully predictable : one of the two players (depending on the initial bin configuration and the first player) has a winning strategy whatever the second player does. We shall study here this strategy.

We begin by writing two conversion functions as follows : The first function converts a number written in base 2 (binary) to a number written in base 10 (decimal). The second function is the reciprocal function : That is, conversion from base 10 to 2.

(4a) An integer n encoded in base 2 shall be modeled using an integer array `binaryRepresentation` of length k storing the k bits. We assume that all values in

this array are equal to 0 or 1, and `binaryRepresentation[i]` is the `i`-th bit of the decomposition of n in base 2 :

$$\texttt{n} = \sum_{\texttt{i=0}}^{\texttt{i<k}} \texttt{binaryRepresentation[i]} * 2^{\texttt{i}}$$

Write a function `static int binaryToDecimal(int[] binaryRepresentation)` that returns the integer corresponding to the binary representation of `binaryRepresentation` (Answer length : a dozen lines)

(4b) Here, we assume that `k` is big enough, and we are given the function :

```
public static int[] decimalToBinary(int n, int k) {
  int[] binaryRepresentation = new int[k];
  decimalToBinaryAux(n, 0, binaryRepresentation);
  return binaryRepresentation;
}
```

Complete this code by writing the **recursive** function `decimalToBinaryAux` (Answer length : a few lines)

In the following, given a number y and an integer k, chosen large enough so that y can be written in base 2 using k bits, we denote by $y[i]$ the i-th bit of the binary decomposition of y. That is, we have $y = \sum_{i=0}^{k} y[i] * 2^i$.

We consider the following function called Grundy that takes as argument a number of fruits in each bin and return an integer $Grundy(x_0, x_2, \ldots, x_{m-1}) = a$ denoting the binary representation defined by $a[i] = \left(\sum_{l=0}^{m-1} x_l[i] \right) \mod 2$.

For example, if we want to compute $Grundy(6, 9, 1, 2)$, we first write 6, 9, 1 and 2 using base 2 (using the same number of k bits) and we compute the sum on each column modulo 2. This yields a representation in base 2 of a. We then convert a in base 10.

6	=	0	1	1	0
9	=	1	0	0	1
1	=	0	0	0	1
2	=	0	0	1	0
a	=	1	1	0	0

In this case, we thus obtain $Grundy(6, 9, 1, 2) = a = 12$.

(4c) ** Write function `static int Grundy(int[] decimalTab)` that returns the value of the Grundy function for the current game configuration stored in array `decimalTab`.

It is recommended to write a few auxiliary functions, as follows :
- compute the required number k of bits,
- build an array `int[][]` storing the binary decompositions of values stored in array `decimalTab`,
- compute the Grundy function in binary.

Those functions can be reused in the remainder.

(Answer length : a few dozen lines)

We shall make use of the following result (admitted without proof) :

> Let a Nim game configuration be denoted by x_0, x_1, ..., x_{m-1} fruits in the respective bins. Then the current player has a winning strategy if and only if $Grundy(x_0, \ldots, x_{m-1}) \neq 0$.

In case, we have $Grundy(x_0, \ldots, x_{m-1}) \neq 0$, the winning move is described by the following steps :

- We consider $Grundy(x_0, \ldots, x_{m-1})$ in base 2, and we select the maximum index j of a bit with value 1 in the decomposition. (Such a bit necessarily exists because of $Grundy(x_1, \ldots, x_{m-1}) \neq 0$).
- We search for an index i corresponding to a bin containing x_i fruits, such that index j denoting the bit index of the binary decomposition of x_i has corresponding bit value equal to 1. (Such an index i necessarily exists because of the Grundy function.) Fruits will now be removed from the bin indexed by i.
- We shall remove from bin i a number of fruits such that the remaining number of fruits x'_i shall satisfy for all rank h :

$$x'_i[h] = \begin{cases} 1 - x_i[h] & \text{if } grundy[h] = 1 \\ x_i[h] & \text{otherwise.} \end{cases}$$

In the example given in the former question, the index j is 3 and the bin to select fruits from has index 1 (containing 9 fruits). We remove 4 fruits so that it remains exactly 5 fruits.

(4d) ** Write a function `pick` that takes as argument an integer array denoting a game configuration, and returns an integer array of size 2 : the first integer shall report the index of the bin to select fruits from, and the second integer shall give the number of fruits to remove from the selected bin. In case the Grundy function is zero (we know that we are going to loose), we shall remove a fruit from the first non-empty bin.

(Answer length : a few dozen lines)