Université Paris 7 - Denis Diderot
IF1 — Corrigé du partiel du 25 Novembre 2006

Exercice 1. Exécution à la main de la séquence Java :

```
byte b1 = 111; byte b2 = 19; byte b3 = -108;
Deug.println(b1 & b2); Deug.println(b1 | b2); Deug.println(b1 ^ b2);
Deug.println(b1 & b3); Deug.println(b1 | b3); Deug.println(b1 ^ b3);
```

- on a : $111 = 2^6 + 2^5 + 2^3 + 2^2 + 2^1 + 2^0$.
  La représentation binaire du nombre $(111)_{10}$ est donc 1101111. En tant que valeur d'une variable de type `byte`, cette représentation est étendue à 8 bits en ajoutant un 0 à gauche, ce qui donne 01101111 ;
- on a : $19 = 2^4 + 2^1 + 2^0$.
  La représentation binaire du nombre $(19)_{10}$ est donc 10011 et en tant que valeur d'une variable de type `byte`, elle est étendue à 8 bits en ajoutant trois 0 à gauche, ce qui donne 00010011 ;
- pour trouver la représentation binaire de $(-108)_{10}$ en tant que valeur de type `byte`, on commence par rechercher celle de $(108)_{10}$.
  On a : $108 = 2^6 + 2^5 + 2^3 + 2^2$, ce qui sur huit bits donne 01101100.
  Le nombre $(-108)_{10}$ est représenté en complément à 2 sur 8 bits. Cette représentation est obtenue en inversant les bits à la gauche du premier bit égal à 1 (à partir de la droite) dans la représentation précédente.
  Cela donne 10010100.

Les calculs des valeurs obtenues par les opérations bit à bit donnent :

```
   b1 : 01101111        b1 : 01101111        b1 : 01101111
   b2 : 00010011        b2 : 00010011        b2 : 00010011
b1&b2 : 00000011     b1|b2 : 01111111     b1^b2 : 01111100

   b1 : 01101111        b1 : 01101111        b1 : 01101111
   b3 : 10010100        b3 : 10010100        b3 : 10010100
b1&b3 : 00000100     b1|b3 : 11111111      b1^b3 : 11111011
```

Les valeurs décimales correspondantes affichées par la méthode `println` sont :

- pour `b1&b2=00000011` : `3` $(2^0 + 2^1)$;
- pour `b1|b2=01111111` : `127` $(2^6 + 2^5 + 2^4 + 2^3 + 2^2 + 2^1 + 2^0)$;
- pour `b1^b2=01111100` : `124` $(2^6 + 2^5 + 2^4 + 2^3 + 2^2)$
- pour `b1&b3=00000100` : `4` (2^2)
- pour `b1|b3=11111111` : `-1`. Ici, le bit de gauche étant égal à 1, le nombre est négatif. On cherche la représentation du nombre positif opposé par complément à 2. Il s'agit du nombre représenté par 00000001, c'est-à-dire 1 ;
- pour `b1^b3=11111011` : `-5` . Comme précedemment, il s'agit d'un nombre négatif dont le nombre positif opposé a comme représentation binaire le complément à 2 de 11111011, c'est-à-dire 00000101 qui est la représentation binaire de $2^2 + 2^1$. D'où le résultat -5 affiché.

Exercice 2. Exécution à la main de la séquence Java :

```
int a = 10, b = 4, c;
float x, y;
c = a + 2 * b; a = a + 5; b = b + 2;
Deug.println(a); Deug.println(b); Deug.println(c);
c = a / b; x = a /b;
Deug.println(c); Deug.println(x);
y = b; x = a / y;
Deug.println(x);
```

La valeur ω d'une variable correspond au fait qu'elle est non initialisée.

instruction	a	b	c	x	y	affichage à l'écran	commentaire
déclarations	10	4	ω	ω	ω		
`c = a + 2 * b`	10	4	18	ω	ω		
`a = a + 5`	15	4	18	ω	ω		
`b = b + 2`	15	6	18	ω	ω		
`Deug.println(a)`	15	6	18	ω	ω	15	
`Deug.println(b)`	15	6	18	ω	ω	6	
`Deug.println(c)`	15	6	18	ω	ω	18	
`c=a/b`	15	6	2	ω	ω		division entière de 15 par 6
`x=a/b`	15	6	2	2.0	ω		div. entière de 15 par 6, affectation à un `float`
`Deug.println(c)`	15	6	2	2.0	ω	2	
`Deug.println(x)`	15	6	2	2.0	ω	2.0	
`y=b`	15	6	2	2.0	6.0		
`x=a/y`	15	6	2	2.5	6.0		division réelle de 15 par 6.0
`Deug.println(x)`	15	6	2	2.5	3.0	2.5	

Exercice 3. *Programme de calcul des sièges obtenus*
Dans la solution donnée ici, en cas d'égalité de voix entre listes, c'est la première dans l'ordre alphabétique qui est considérée comme gagnante.

```
import fr.jussieu.script.*;
public class Exo3 {
  public static void main(String[] args) {
    Deug.print("Nombre de sieges a pourvoir ? ");
    int n = Deug.readInt();
    Deug.print("Nombre de suffrages pour la liste 1 ? ");
    int a = Deug.readInt();
    Deug.print("Nombre de suffrages pour la liste 2 ? ");
    int b = Deug.readInt();
    Deug.print("Nombre de suffrages pour la liste 3 ? ");
    int c = Deug.readInt();
    if( (n < 1) || (a < 0) || (b < 0) || (c < 0)) {
       Deug.println("Donnees incorrectes");
       Deug.exit(); }
    int v = a + b + c ; // Nombre total de suffrages exprimés
    int q = v / n; // coefficient electoral
    Deug.println("Suffrages exprimes : " + v);
    Deug.println("Coefficient electoral : " + q);
    int s1 = a / q; // nombre de sieges automatique de la liste 1
    int s2 = b / q; // nombre de sieges automatique de la liste 2
    int s3 = c / q; // nombre de sieges automatique de la liste 3
    int r = n - (s1 + s2 + s3); // nombre de sieges restants
    if (a >= b && a >= c) s1 = s1 + r ;
    else if (b >= c ) s2 = s2 + r ;
    else s3 = s3 + r ;
    Deug.println ("Methode de la meilleure liste");
    Deug.println("   liste 1 : " + s1 + " siege(s)");
    Deug.println("   liste 2 : " + s2 + " siege(s)");
    Deug.println("   liste 3 : " + s3 + " siege(s)");
  }
}
```

Exercice 4. Calcul à la volée des sommes et produits des nombres positifs et négatifs dans une suite de nombres.

```
import fr.jussieu.script.*;
class Exo4 {
  public static void main(String[] args) {
    int nbPos = 0; // nombre de positifs lus
    int nbNeg = 0; // nombre de negatifs lus
    int sPos = 0; // la variable de calcul de la somme des nombres positifs
    int pPos = 1; // la variable de calcul du produit des nombres positifs
    int sNeg = 0; // la variable de calcul de la somme des nombres positifs
    int pNeg = 1; // la variable de calcul du produit des nombres positifs
    int x; // variable utilisé pour lire un nombre
    x = Deug.readInt();
    while (x != 0) {
      if(x > 0) { // le nombre lu est positif
        nbPos++;
        sPos += x; // sPos = sPos +x; x est ajouté à la somme des (> 0)
        pPos *= x; // pPos = pPos +x; le produit des (> 0) est multiplie par x
        x = Deug.readInt(); // le nombre suivant est lu
        }
      else { // le nombre lu est négatif
        nbNeg++;
        sNeg += x; // sNeg = sNeg +x; x est ajouté à la somme des (< 0)
        pNeg *= x; // pNeg = pNeg +x; le produit des (< 0) est multiplie par x
        x = Deug.readInt(); // le nombre suivant est lu
        }
   }
   // le nombre qui a ete lu est nul. On affiche les valeurs et on termine
   if(nbPos == 0) pPos = 0; // aucun nombre positif lu => pPos est nul
   if(nbNeg == 0) pNeg = 0; // aucun nombre negatif lu => pNeg est nul
   Deug.println("Somme des positifs : " + sPos);
   Deug.println("Produit des positifs : " + pPos);
   Deug.println("Somme des negatifs : " + sNeg);
   Deug.println("Produit des negatifs : " + pNeg);
   }
}
```

Remarque : une solution élégante pour savoir si au moins un nombre positif (resp. négatif) a été lu, est de tester à la sortie de la boucle la valeur de $sPos$ (resp.$sNeg$). Si cette valeur est nulle, aucun nombre positif (resp. négatif) n'a été lu et il faut alors donner à $pPos$ (resp. $pNeg$) la valeur 0.

Exercice 5.

5.1. Méthode Java prenant en argument un tableau t d'entiers et un entier n, et construisant sa représentation occurrentielle relativement à n avec lissage aux extrémités :

```
static int[] repOcc1(int[] tab, int n) {
  // declaration/allocation du tableau résultat
  // (éléments intialisés à 0 automatiquement)
  int[] occ = new int[n+1];
  int i;  // pour parcourir le tableau tab
  // parcours du tableau tab donne
  for(i = 0; i < tab.length; i++) {
     // si tab[i] est nul, on incremente occ[0]
     if(tab[i] <= 0) occ[0]++;
     // si tab[n] est >= n, on incremente occ[n]
     else if(tab[i] >= n) occ[n]++;
     // sinon, on incremente occ[tab[i]]
     else occ[tab[i]]++;
  }
  return occ;
}
```

5.2. Méthode Java qui calcule la représentation occurrentielle optimale d'un tableau.

La représentation occurrentielle optimale est obtenue en prenant pour valeur de n la valeur du plus grand entier positif du tableau. La fonction `valMax` recherche le plus grand entier dans le tableau (si tous les éléments du tableau sont négatifs, la fonction renvoie 0).
Le calcul de la représentation optimale est réalisée en appelant la fonction écrite précédemment avec cette valeur n.

```
static int valMax(int[] tab) {
  int val = 0;
  for(int i = 0; i < tab.length; i++)
      if(tab[i] > val) val = tab[i];
  return val;
}
static int[] repOcc2(int[] tab) {
  int n = valMax(tab);
  int[] occ = repOcc1(tab, n);
  return occ;
}
```

5.3. Méthode Java qui trie un tableau d'entiers en lissant les valeurs négatives sur 0 et en éliminant les répétitions.

```
static int[] tabTriSansRep(int[] tab) {
  // construction de la représentation occurentielle optimale
  int [] occ = repOcc2(tab);
  int taille; // la taille du tableau a construire
  // la taille du tableau est égale au nombre d'éléments non nuls du tableau occ
  for(int i = 0; i < occ.length; i++)
     if(occ[i] != 0) taille ++;
  int[] tab; // le tableau resultat
  tab = new int[taille]; // allocation du tableau resultat
  int pos = 0; // indice prochaine ecriture dans tab
```

```
    // Parcours de la representation occurentielle optimale
    // Pour chaque element non nul, son indice est ecrit dans le tableau resultat
    for(int i = 0; i < occ.length; i++)
       if(occ[i] > 0) {tab[pos] = i; pos++;}
    return tab;
  }
```

5.4. Méthode qui affiche un tableau à raison de 20 valeurs par ligne et séparées par un espace.

```
  static void afficheTab(int[] t) {
    for(int i = 0; i < t.length; i++) {
       Deug.print(t[i] + " ");
       if((i+1)%20 == 0) Deug.println(); }
    Deug.println();
  }
```

5.5. Programme Java utilisant ces méthodes :

```
import fr.jussieu.script.*;
class Exo5 {
  // recopier ici la definition de toutes les methodes precedentes
  public static void main(String[] args) {
    int n; // la taille du tableau
    int[] t; // le tableau
    // lecture de la taille du tableau
    Deug.print("taille du tableau ? ");
    n = Deug.readInt();
    t = new int[n]; // allocation du tableau
    // lecture du tableau
    Deug.println("lecture du tableau  ");
    for(int i = 0; i < n; i++){
       Deug.print("t[" + i + "] ? ");
       t[i] = Deug.readInt();}
    int val; // la valeur choisie pour construire la rep. occurrentielle
    // lecture de sa valeur
    Deug.print("lecture d'un nombre postif ou nul ? ");
    val = Deug.readInt();
    while (val < 0){
        Deug.print("lecture d'un nombre positif ou nul ? ");
        val = Deug.readInt(); }
    // construction de la rep. occurrentielle de t associee a cette valeur
    int[] occ = repOcc1(t, val);
    Deug.println("representation ocurrentielle pour " + val);
    afficheTab(occ);
    // construction de la rep. occurrentielle optimale par appel à repOcc2
    occ = repOcc2(t);
    Deug.println("representation ocurrentielle optimale");
    afficheTab(occ);
    // construction du tableau trie sans repetition
    //  avec valeurs negatives identifiees a 0
    int[] tri = tabTriSansRep(t);
    Deug.println("tableau trie sans repetition");
    afficheTab(tri);
  }
}
```