Université Paris 7 – Denis Diderot
Introduction à l'informatique et à la programmation (IF1)
Corrigé de l'examen du 6 janvier 2009

Exercice 1.

1.1. Pour répondre à la question, nous avons regroupé dans une table ce qui est effectivement calculé par le programme selon les valeurs des variables booléennes `a`, `b` et `c` (ce qui n'est pas évalué par le programme est matérialisé par une séquence `---`).

a	b	c	!b	a||!b	c	!a	!a&&c	c	a||b	!c&&(a||b)
0	0	0	1	1	---	1	0 : Affichage de C	---	---	---
0	0	1	1	1	1 : Affichage de A	---	---	---	---	---
0	1	0	0	0	---	---	---	1	1	1 : Affichage de D
0	1	1	0	0	---	---	---	0	1	0 : Affichage de E
1	0	0	1	1	---	0	0 : Affichage de C	---	---	---
1	0	1	1	1	1 : Affichage de A	---	---	---	---	---
1	1	0	1	1	---	0	0 : Affichage de C	---	---	---
1	1	1	0	1	1 : Affichage de A	---	---	---	---	---

Le programme affiche donc :

- `C` si `(!a && !b && !c) || (a && !b && !c) || (a && b && !c)`
- `A` si `(!a && !b && c) || (a && !b && c) || (a && b && c)`
- `D` si `(!a && b && !c)`
- `E` si `(!a && b && c)`

1.2. On observe que : La première condition correspondant à l'affichage de `C` se réduit à :
`(!c && (!b || (a && b)))`
La fonction peut donc se définir par (on a repoussé le retour de la valmeur `A` au dernier `else` pour éviter l'écriture explicite d'une condition compliquée).

```
if (!c && (!b || (a && b)))
    return('C');
else if (!a && b && !c)
    return('D');
else if(!a && b && c)
    return('E');
else
    return('A');
```

Exercice 2.

2.1.

```
   public static int combien(int[] t){
       boolean[] dejaVu = new boolean[21];
       int cpt = 0;  // le compteur de nombres differents
       for(int i = 0; i < t.length; i++) {
              // pas dans l'intervalle ou deja vu : nombre ignore
              if(t[i] < 0 || t[i] > 20 || dejaVu[t[i]])
                   continue;
              // un nouveau nombre a ete rencontre
              cpt++;   // on le comptabilise
              dejaVu[t[i]] = true; // on note qu'on l'a vu
       }
       return cpt;
   }
}
```

2.2. *Nombre de nombres différents sans connaissance d'intervalles*

La solution utilisée précédemment n'est pas brutalement utilisable puisqu'on ne connaît pas l'intervalle de définition des nombres contenus dans le tableau.

On va pour chaque élément `t[i]` le comparer avec tous ses précédents dans le tableau et on le comptabilisera que s'il n'est égal à aucun d'eux.

```
   public static int combien(int[] t){
          boolean vu;
          int n = 1;
        for(int i = 1; i < t.length; i++) {
            vu = false;
            for(int j = 0 ; j < i; j++)
                 if(t[i] == t[j]){ vu = true; break; }
            if (!vu)  n++;
        }
        return n;
   }
```

Exercice 3. *Jeu du pendu/Mastermind*

3.1. *Tableau contenant les nombres de pions bien et mal placés*

```
   public static int[] trouve(char[] tab1, char[] tab2){
       int[] cumul = new int[2];
       boolean[] dejaUtilise = new boolean[tab1.length];
       for (int i = 0; i < tab1.length; i++){
           if(tab1[i] == tab2[i]) { cumul[0] ++; continue; }
           for(int j = 0; j < tab1.length; j++){
               if(tab1[j] == tab2[j] || dejaUtilise[j]) continue;
               if(tab1[j] == tab2[i]){
                   cumul[1]++;
                   dejaUtilise[j]= true;
                   break;
               }
           }
       }
       return cumul;
   }
```

3.2. *Construction de la réponse*

```
   public static char[] reponse (int[] cumul){
       char[] rep = new char[cumul[0] + cumul[1]];
       for(int i = 0; i < cumul[0]; i++) rep[i] ='+';
       for(int i = 0; i < cumul[1]; i++) rep[cumul[0] + i] ='-';
       return rep;
   }
```

3.3. *Une partie contre la machine*

```
import fr.jussieu.script.*;
class MasterMind {
   public static int[] trouve(char[] tab1, char[] tab2){
       /* code deja ecrit */
            ......
   }
   public static char[] reponse (int[] cumul){
       /* code deja ecrit */
            ......
   }
   public static void ecrireReponse(char[] t){
       for(int i = 0; i < t.length; i++)
           Deug.print(t[i]);
       Deug.println();
   }
   public static void main(String[] args){
       char[] solution = couleur();
       for (int = 0; i < 4) solution[i] = couleur();
       char[] proposition = new char[4];
       int [] nombre;
       char[] reponse;
       char c;
       for (int rep = 0; rep < 12; rep ++) {
           Deug.print("Proposition " + (rep + 1) + " : ");
           for(int i = 0; i < solution.length; i++)
                proposition[i] = Deug.readChar();
           c = Deug.readChar();
           nombre = trouve(proposition, solution);
           if(nombre[0] == 4){
                 Deug.println("Bravo. Solution trouvee en " + (rep + 1) + " coups");
                 Deug.exit();}
           reponse =  reponse(nombre);
           ecrireReponse(reponse);
           Deug.println("====================");
       }
       Deug.print("Vous avez perdu ! La reponse etait : ");
       ecrireReponse(solution);
   }
}
```

Exercice 4.

4.1. *Construction du tableau des points de rupture*

```
public static int[] rupture(int[] t){
    int[] tt = new int[t.length];
    int n = 1;  tt[0] = 0;
    for (int i = 1; i < t.length; i++){
       if (t[i] > t[i-1]) continue;
       tt[n] = i; n++;
       }
    int[] res = new int[n];
    for(int i = 0; i < n; i++) res[i] = tt[i];
    return res;
}
```

4.2. *Décomposition d'un tableau en suite de tableaux strictement croissants*

```
public static int[][] factorisation(int[] t){
    int[] r = rupture(t);
    int[][] res = new int[r.length][];
    for (int i = 0; i < r.length - 1; i++){
       res[i] = new int[r[i+1] - r[i]];
       for(int j = 0; j < res[i].length; j++)
          res[i][j] = t[r[i] + j];
    }
    res[r.length - 1] = new int[t.length - r[r.length-1]];
    for(int j = 0; j < res[r.length - 1].length; j++)
          res[r.length - 1][j] = t[r[r.length-1] + j];
    return res;
}
```

Exercice 5. *Othello/Reversi*

5.1. *Initialisation du jeu*

```
public static void init(int[][] t){
    // initialisation de toutes les cases a 2 (libre)
   for(int i = 0; i < t.length; i++)
       for(int j = 0; j < t[i].length; j++)
            t[i][j] = 2;
   t[3][3] = t[4][4] = 0;  // pions blancs
   t[3][4] = t[4][3] = 1;  // pions noirs
}
```

5.2. *Fonction retournant le nombre de voisins d'une case*

```
public static int nVoisins(int lig, int col){
    if(lig == 0 || lig == 7)
       if(col == 0 || col == 7) return 3;
       else return 5;
    if(col == 0 || col == 7) return 5;
    return 8;
}
```

5.3.

5.5. *Positions valides sur la table*

```
public static int[][] valides(int[][] table, int couleur){
  int a, b, c;
  int pos[][] = possibles(table, couleur);
  if (pos == null)
     return null;  // le joueur est bloque
  int[][] valides = new int[pos.length][];
  int n = 0; // le nombre de cases envisageables
  // pour chaque position potentielle, il faut rechercher s'il existe
  // un pion notre couleur dans la direction du voisin de couleur differente
  for (int i = 0; i < pos.length; i++) {
     a = pos[i][0] + 2 * pos[i][2];
     b = pos[i][1] + 2 * pos[i][3];
     c = 1;
     while(a >= 0 && a <= table.length - 1 && b >= 0 && b <= table.length - 1) {
        if(table[a][b] == 2)
            break;
        if(table[a][b] == couleur){
            valides[n] = new int[3];
            valides[n][0] = pos[i][0];
            valides[n][1] = pos[i][1];
            valides[n][2] = c; n++;
            break;
        }
        c++;
        a += pos[i][2];
        b += pos[i][3];
     }
   }
   if (n == 0)
       return null;
   int[][] res = new int[n][];
   for(int i = 0; i < n; i++)
       res[i] = valides[i];
   return res;
}
```