## Algorithmique procédurale

# TD de Révision corrigé

## 1) Tableaux

a) Deux tableaux sont *égaux* s'ils contiennent les mêmes éléments aux mêmes positions. Deux tableaux sont *similaires* s'ils contiennent les mêmes éléments mais pas forcément aux mêmes positions. Par exemple: [9,6,7,6,3,6,1,9] et [1,3,6,6,6,7,9,9] sont similaires alors que [9,6,7,6,3,6,1,9] et [9,9,7,6,3,6,1,9] ne le sont pas. Deux tableaux sont *comparables* si l'ensemble des valeurs qu'ils contiennent est le même. Par exemple, les tableaux [1,3,6,7,9],[6,7,3,6,1,9] et [9,6,7,6,3,6,1,9] sont tous les trois comparables.

- Écrire une méthode qui teste si deux tableaux sont égaux.
- Écrire une méthode qui teste si deux tableaux sont similaires.
- Écrire une méthode qui teste si deux tableaux sont comparables.

```
Fonction égaux (tab1[n] : Tableau de T, tab2[m] : Tableau de T) : Booléen
Variables i : Entier
Début
        Si n != m Alors Retourner Faux FinSi
        Pour i ←1 à n
                Si tab1[i] != tab2[i] Alors Retourner Faux FinSi
        FinPour
        Retourner Vrai
Fin

Fonction nbOcc (tab[n] : Tableau de T, x : T) : Entier
Variables i, c: Entier
Debut
        c ← 0
        Pour i ← 1 à n
                Si tab|i] = x Alors c ← c+1 FinSi
        FinPour
        Retourner c
Fin

Fonction similaires (tab1[n] : Tableau de T, tab2[m] : Tableau de T) : Booléen
Variables i : Entier
Début
        Si n != m Alors Retourner Faux FinSi
        Pour i ← 1 à n
                Si nbOcc(tab1, tab1[i]) != nbOcc(tab2, tab1[i])  Alors Retourner Faux FinSi
        FinPour
        Retourner Vrai
Fin

Fonction comparables (tab1[n] : Tableau de T, tab2[m] : Tableau de T) : Booléen
Variables i : Entier
Début
        Pour i ← 1 à n
                Si nbOcc(tab2, tab1[i]) = 0  Alors Retourner Faux FinSi
        FinPour
        Pour i ← 1 à m
                Si nbOcc(tab1, tab2[i]) = 0  Alors Retourner Faux FinSi
        FinPour
        Retourner Vrai
Fin
```

b) Un tableau carré à deux dimensions contient des lettres à raison d'un caractère par case du tableau. Ecrire une fonction qui détermine si un mot donné est présent dans le tableau en ligne ou en colonne. Ce mot est stocké dans un tableau à une dimension à raison d'un caractère par case.

```
Fonction testUneCaseHor(matrice[n][m] : Tableau de Caractère, i : Entier, j : Entier mot[k] : Tableau de Caractère) : Booléen
Variables t : Entier
Début
        Pour t ← 1 à k
                Si matrice[i+t-1][j] != mot[t] Alors Retourner Faux FinSi
        FinPour
        Retourner Vrai
Fin

Fonction testUneCaseVer (matrice[n][m] : Tableau de Caractère, i : Entier, j : Entier mot[k] : Tableau de Caractère) : Booléen
Variables t : Entier
Début
        Pour t ← 1 à k
                Si matrice[i][j+t-1] != mot[t] Alors Retourner Faux FinSi
        FinPour
        Retourner Vrai
Fin

Fonction présent(matrice[n][m] : Tableau de Caractère, mot[k] : Tableau de Caractère) : Booléen
Variables i, j : Entier
Début
        Pour i ← 1 à n-k+1
                Pour j ← 1 à m-k+1
                        Si (testUneCaseHor(matrice, i, j, mot) OU testUneCaseVer(matrice, i, j, mot)) Alors Retourner Vrai FinSi
                FinPour
        FinPour
        Retourner Faux
Fin
```

## 2) Tris

a) Quels tris connaissez-vous et quel est leur principe?
Appliquez-les au tableau d'entiers suivant : [4, 8, 2, 10, 1, 9, 7, 6, 3, 5]

*Tri insertion, tri sélection, tri à bulles, tri rapide, tri fusion.*

b) Ecrire un algorithme qui trie un tableau de booléens de sorte que tous les FAUX se trouvent à gauche du tableau, et tous les VRAI à droite du tableau. Attention, votre algorithme doit avoir la complexité O(n), et non pas O(n*log n) voire O(n²). Une solution basée sur le tri casier ne compte pas.

```
Procédure TriVraiFaux(ES T: Tableau de Booléen)
Variables i, j : Entier
Début
    i ← 1
    j ← N
    Tantque i < j
        Si T[i] = Faux Alors
            i ← i + 1
        Sinon // T[i] = Vrai
            échanger(T, i, j)
            j ← j - 1
        FinSi
    FinTantque
Fin
```

c) « Tri drapeau » (difficile) : un tableau contient des éléments rouges, blancs et bleus. Triez ce tableau pour que les éléments rouges soient au début, les blancs au milieu et les bleus à la fin. De nouveau, l'algorithme doit être O(n), c'est-à-dire qu'il faut ne tester qu'une fois la couleur d'un élément. Une solution basée sur le tri casier ne compte pas.

```
Procédure TriDrapeau (ES T: Tableau de Couleurs)
Variables i, j, k : Entier
Debut
        k ⟵ 0
        i ⟵ 1
        j ⟵ N
        Tantque i <= j
                Si T[i]=BLANC Alors
                        i ⟵ i + 1
                SinonSi T[i] = BLEU Alors
                        échanger(T, i, j)
                        j ⟵ j - 1
                Sinon // T[i] = ROUGE
                        k ⟵ k + 1
                        échanger(T, k, i)
                        i ⟵ i +1
                FinSi
        FinTantque
Fin
```

## 3) Pointeurs C

Explicitez ce qu'il se passe à chaque ligne du « main » dans le code suivant:

```
#include <stdio.h>
void foo(int val, int res) {
       res = val;
}
void bar(int val, int *res) {
       *res = val;
       if (val < 0) *res *= -1;
}
int main() {
       int a, b, c, d;
       int * p1, * p2, * p3, * p4;
       a = 7;
       b = -3;
       c = 12;
       d = -5;
       p1 = &c;
       p2 = &a;
       p3 = &b;
       p4 = p1;
       foo(c,*p3);
       *p1 += a++;
       *p3 += ++(*p2);
       bar(d, p1);
       printf("Résultat: %d,%d,%d,%d,%d,%d,%d,%d \n", a, b, c, d, *p1, *p2, *p3, *p4);
       return 0; // ou return EXIT_SUCCESS
}
```

1) *foo ne fait strictement rien, car cette fonction reçoit une copie des entiers en question et non pas des pointeurs*
2) **p1 += a++ : on ajoute a à *p1 (donc à c), puis on incrémente a*
3) **p3 += ++(*p2): on incrémente d'abord *p2 (donc a), puis on ajoute cette valeur à *p3 (donc à b)*
4) *bar fait d'abord que *p1 (donc c) vaut d, puis *p1 est multiplié par -1*
5) *L'affichage final : 9 6 5 -5 5 9 6 9*