Jean-François Remm, Jean-François Berdjugin, Jean-François

Correction TD Algorithmique

5.2 Tableaux

Utilisation des tableaux

Exercice 36

Exécuter l'algorithme suivant :

```
Algo Exercice1
        var carre : Tableau <entier>
        var i : entier
Début
        carre ← nouveau Tableau<entier>(4)
        carre[0] ← 2
        carre[1] ← 5
        carre[2] ← 3
        carre[3] ← 10
        Pour i de 1 à carre.longueur()-1 Faire
                carre[i] ← carre[i] * carre[i]
        Fin Pour
        Pour i de 0 à carre.longueur()-1 Faire
                Ecrire(carre[i])
        Fin Pour
Fin
```

Variables
carre

2	25	9	100
carre[0]	carre[1]	carre[2]	carre[3]

i 4

Affichage
2
25
9
100

Exercice 37

Exécuter l'algorithme suivant :

Algo Exercice2
    var nb : Tableau <*entier*>
    var i : *entier*
Début
    nb ← **nouveau** Tableau<*entier*>(6)
    nb[0] ← 1
    Pour i **de** 1 **à** nb.longueur()-1 **Faire**
        nb[i] ← nb[i-1] + 2
    Fin Pour
    Pour i **de** 0 **à** nb.longueur()-1 **Faire**
        Ecrire(nb[i])
    Fin Pour
Fin

Variables
nb

1	3	5	7	9	11
nb[0]	nb[1]	nb[2]	nb[3]	nb[4]	nb[5]

i 6

Affichage
1
3
5
7
9
11

Exercice 38

Exécuter l'algorithme suivant :

```
Algo Fibonacci
        var suite : Tableau <entier>
        var i : entier
Début
        suite ← nouveau Tableau<entier>(6)
        suite[0] ← 1
        suite[1] ← 1
        Pour i de 2 à suite.longueur()-1 Faire
                suite[i] ← suite[i-1] + suite[i-2]
        Fin Pour
        Pour i de 0 à suite.longueur()-1 Faire
                Ecrire(suite[i])
        Fin Pour
Fin
```

Variables
suite

1	1	2	3	5	8
suite[0]	suite[1]	suite[2]	suite[3]	suite[4]	suite[5]

i 6

Affichage
1
1
2
3
5
8

Exercice 39

Écrire un algorithme permettant d'obtenir la somme des éléments d'un tableau tab.

```
Algo sommeTab
        var tab : Tableau <entier>
        var i : entier
        var somme : entier
Début
        //declaration + instanciation de tab
        //initialisation de tab
        somme ← 0
         Pour i de 0 à tab.longueur()-1 Faire
                somme ← somme + tab[i]
        Fin Pour
        Ecrire(somme)
Fin
```

Exercice 40

Même exercice, mais sous forme d'une fonction prenant un tableau d'entiers en paramètre et renvoyant un entier.

```
Function sommeTab(tab : Tableau <entier>) : entier
        var i : entier
        var somme : entier

        somme ← 0
         Pour i de 0 à tab.longueur()-1 Faire
                somme ← somme + tab[i]
        Fin Pour
        Renvoie(somme)
Fin
```

Exercice 41 (Maximum)

Écrire une fonction qui prend un tableau d'entiers en paramètre et détermine le plus grand élément de celui-ci.

```
Function maxTab(tab : Tableau <entier>) : entier
        var i : entier
        var max : entier

        max ← tab[0]

        Pour i de 1 à tab.longueur()-1 Faire
                Si (max < tab[i]) Alors
                        Max ←  tab[i]
                Finsi
        Fin Pour
        Renvoie(max)
Fin
```

Exercice 42 (Position du maximum)

Écrire une fonction qui prend un tableau d'entiers en paramètre et détermine la position du plus grand élément de celui-ci.

```
Function posMaxTab(tab : Tableau <entier>) : entier
        var i : entier
        var posMax : entier

        posMax ← 0

        Pour i de 1 à tab.longueur()-1 Faire
                Si (tab[posMax]) < tab[i]) Alors
                        posMax ←  i
                Finsi
        Fin Pour
        Renvoie(posMax)
```

```
Fin
```

Exercice 43 (Position du maximum (2))

Même exercice, mais en indiquant les bornes min et max de recherche du maximum.

```
Function posMaxTab(tab : Tableau <entier>, int min, int max) : entier
        var i : entier
        var posMax : entier

        posMax ← min

        Pour i de min+1 à max Faire
                Si (tab[posMax]) < tab[i]) Alors
                        posMax ←  i
                Finsi
        Fin Pour
        Renvoie(posMax)
Fin
```

Exercice 44 (Inversion)

Écrire une procédure qui permet d'inverser un tableau.

```
Function inversionTab(tab : Tableau <E>) : vide
        var i : entier
        var n : entier
        var temp : E

        n ← tab.longueur()-1

        Pour i de 0 à n/2 Faire
                temp ←  tab[i]
                tab[i] ←  tab[n-i]
                tab[n-i] ←  temp
        Fin Pour
Fin
```

Exercice 45 (Suppression)

Écrire une fonction qui supprime l'élément à la position pos dans un tableau tab de E.

```
Function suppressionTab(tab : Tableau <E>, pos : entier) : vide
        var i : entier
        var n : entier
        Pour i de pos à tab.longueur()-2 Faire
                tab[i] ←  tab[i+1]
        Fin Pour
Fin
```

Exercice 46 (Insertion)

Écrire une fonction qui insère un élément elt de type E dans un tableau tab de E à une position donnée. On suppose que :

- soit il reste une position de libre dans le tableau (à la fin)
- soit on perd le dernier élément

```
Function insertionTab(tab : Tableau <E>,  elt : E, pos : entier) : vide
        var i : entier

        Pour i de tab.longueur()-2 à pos pas -1 Faire
                tab[i+1] ← tab[i]
        Fin Pour
        Tab[pos] ← elt
Fin
```

Exercice 47 (Fusion)

Écrire une fonction qui prend en paramètre deux tableaux et renvoie le tableau résultant de leur concaténation.

```
Function fusionTab(tab1 : Tableau <E>,  tab2 : Tableau <E>) : Tableau <E>
        var i : entier
        var j : entier
        var tabRes : Tableau <E>

        tabRes ← nouveau Tableau <E>) (tab1.longueur() + tab2.longueur())
        j ← 0
        Pour i de 0 à tab1.longueur()-1 Faire
                tabRes[j] ← tab1[i]
                j ← j + 1
        Fin Pour
        Pour i de 0 à tab2.longueur()-1 Faire
                tabRes[j] ← tab2[i]
                j ← j + 1
        Fin Pour
        Renvoie(tabRes)
Fin
```

Exercice 48 (Recherche séquentielle dans un tableau non trié)

Écrire une fonction qui prend en paramètre un tableau et une valeur et qui renvoie la position (première trouvée) de la valeur ou -1 si cette valeur n'a pas été trouvée dans le tableau.

```
Function rechSeqTab(tab : Tableau <entier>,  elt : entier) : entier
        var i : entier
        var position : entier
        var trouve : booleen

        trouve ← faux
        position ← -1
        i ← 0
```

```
        Tantque non trouve et i < tab.longueur() faire
                Si (tab[i]=elt) Alors
                        trouve ← vrai
                        postion ← i
                Sinon
                        i ← i + 1
                Fin Si
        Fin Tant que
        Renvoie(position)
Fin
```

Exercice 49 (Recherche séquentielle dans un tableau trié)

On peut arrêter la recherche quand la valeur recherchée ne peut plus être trouvée. On suppose le tableau trié de manière croissante. le type des éléments du tableau doit être doté d'un ordre total (i.e. ∀ elt1, elt2 : type elt1≥ elt2 ou elt1 < elt2)

```
Function rechSeqTab(tab : Tableau <entier>,  elt : entier) : entier
        var i : entier
        var position : entier
        var arret : booleen

        arret ← faux
        position ← -1
        i ← 0

        Tantque non arret et i < tab.longueur() faire
                Si (tab[i]=elt) Alors
                        arret ← vrai
                        postion ← i
                Sinon
                        Si  (tab[i] > elt) Alors
                                arret ← vrai
                        Sinon
                                i ← i + 1
                        Fin Si
                Fin Si
        Fin Tant que
        Renvoie(position)
Fin
```

Exercice 50 (Recherche dichotomique dans un tableau trié)

Le principe de cette recherche est très astucieux (et économique).
Algo en log(n).

```
Function rechDicoTab(tab : Tableau <entier>,  elt : entier) : entier
        var inf : entier
        var sup : entier
        var milieu : entier
```

```
        var position : entier
        var trouve : booleen

        trouve ← faux
        position ← -1
        inf ← 0
        sup ← tab.longueur()-1

        Tantque non trouve et inf ≤ sup faire
                milieu ← (inf+sup)/2

                Si (tab[milieu]=elt) Alors
                        trouve ← vrai
                        postion ← milieu
                Sinon
                        Si (tab[milieu] > elt) Alors
                                sup ← milieu - 1
                        Sinon
                                Inf ← milieu + 1
                        Fin Si
                Fin Si
        Fin Tant que
        Renvoie(position)
Fin
```

Exercice 51 (Tri par échange)

Écrire une fonction qui trie un tableau (d'entier) donné en paramètre. Le principe de l'algorithme est :

- de rechercher la position du maximum sur les cases de 0 à n (n initialement à longueur-1)
- d'échanger le maximum et la case n
- de diminuer n et recommencer jusqu'à ce que n soit égal à 1.

Tri en n^2.

```
Function triEchangeTab(tab : Tableau <entier>) : vide
        var temp : entier
        var limite : entier
        var posMax : entier

        temp ← 0

        Pour limite de tab.longueur()-1 à 1 pas -1 Faire
                posMax ← 0
                Pour i de 1 à limite Faire
                        Si (tab[posMax] < tab[i]) Alors
                                posMax ← i
                        Fin Si
```

```
                Fin Pour
                temp ← tab[posMax]
                tab[posMax] ←  tab[limite]
                tab[limite] ←  temp
        Fin Pour
Fin
```

Exercice 52 (Tri rapide)

Le principe est de séparer le tableau en deux parties. La séparation est effectuée en choisissant une valeur pivot. Les valeurs sont réparties en deux ensembles suivant qu'elles sont plus grandes ou plus petites que le pivot. Ensuite, les deux ensembles sont triés séparément, suivant la même méthode. L'algorithme est donc récursif.
Algorithme en n log(n) (sauf dans le cas d'un tableau trié ☺ mais pas le dire).

```
Fonction quickSort(tab : tableau<entier>, p : entier, r: entier):vide
      var entier q;
      Si (p<r) Alors
            q = partitionner(tableau, p, r)
            quickSort(tableau, p, q)
            quickSort(tableau, q+1, r)
      Fin Si
Fin Fonction

Fonction partitionner(tab: tableau<entier>, p: entier, r:entier): entier
         var pivot : entier
         var i : entier
         var j : entier
         var temp : entier
         var ok: booleen

         pivot = tab[pivot]
         i = pivot -1
         j = r+1
         ok ←  vrai
      Tant que (ok) Faire
            j ←  j - 1
            Tant que (tab[j] > pivot) Faire
                  I ← i +1
            Tant que (tab[i] < pivot) Faire
            Si (i<j) Alors
                  temp = tab[i]
                  tab[i] = tab[j]
                  tab[j] = temp
             Sinon
                  ok ←  faux
             Fin Si
      Renvoie(j)
 Fin Fonction
```