TD3 – éléments de corrigé (algorithmes)

Algorithme 1 : AfficherTableauEntier1D
Principe : Parcourir les cases du tableau de 0 à taille − 1 et afficher la case à l'indice courant.

```
Entrée :        - tab : tableau 1D de taille entiers
                - taille : entier
Local :         - i : entier
Sortie :        void
Début
|       Pour i de 0 à taille − 1 avec un pas de +1 Faire
|       |       AfficherEntier(tab[i])
|       Fin Pour
Fin
```

Algorithme 2 : CalculerMoyenneTableauEntier1D
Principe : Initialiser une somme à 0. Parcourir toutes les cases du tableau de l'indice 0 à taille − 1 et ajouter la valeur de la case à la somme. Afficher la somme divisée par la taille du tableau.
Précondition : taille > 0

```
Entrée :        - tab : tableau 1D de taille entiers
                - taille : entier
Local :         - i : entier
                - moyenne : réel
Sortie :        void
Début
|       Soit moyenne = 0
|       Pour i de 0 à taille − 1 avec un pas de +1 Faire
|       |       moyenne = moyenne + tab[i]
|       Fin Pour
|       moyenne = moyenne / taille
|       AfficherRéel(moyenne)
Fin
```

Algorithme 3 : CompterNombreApparitionsEntierDansTableauEntier1D
Principe : Parcourir les cases du tableau de l'indice 0 à taille − 1 et si la case contient l'entier recherché compter 1 (dans une variable initialement à 0 avant le parcours). Afficher le compteur.

```
Entrée :        - tab : tableau 1D de taille entiers
                - taille : entier
Local :         - nCible : entier
                - nApparitions : entier
Sortie :        void
Début
|       Soit nApparitions = 0
|       Soit nCible = SaisirNombreEntier()
|       Pour i de 0 à taille − 1 avec un pas de +1 Faire
|       |       Si tab[i] == nCible Alors
|       |       |       nApparitions = nApparitions + 1
|       |       Fin Si
|       Fin Pour
|       AfficherEntier(nApparitions)
Fin
```

Algorithme 4 : RechercheMaximumDansTableauEntier1D
Principe : Initialiser le maximum à la valeur de la première case (indice 0). Parcourir le tableau de l'indice 1 à taille − 1. Si la case contient une valeur strictement supérieure au maximum, remplacer le maximum par celle-ci. Afficher le maximum.
Précondition : taille > 0
Entrée : - tab : tableau 1D de taille entiers
- taille : entier
Local : - i : entier
- maximum : entier
Sortie : void
Début
| **Soit** maximum = tab[0]
| **Pour** i **de** 1 **à** taille − 1 **avec un pas de** +1 **Faire**
| | **Si** tab[i] > maximum **Alors**
| | | maximum = tab[i]
| | **Fin Si**
| **Fin Pour**
| AfficherEntier(maximum)
Fin

Algorithme 5.1 : AffichageNotes
Principe : idem à l'algorithme 1
Entrée : - lesNotes : tableau 1D de taille entiers
- taille : entier
Local : - i : entier
Sortie : void
Début
| **Pour** i **de** 0 **à** taille − 1 **avec un pas de** +1 **Faire**
| | AfficherEntier(lesNotes[i])
| **Fin Pour**
Fin

Algorithme 5.2 : AffichagePlusPetiteNote
Principe : idem à la recherche du maximum en inversant le signe (algorithme 4)
Précondition : taille > 0
Entrée : - lesNotes : tableau 1D de taille entiers
- taille : entier
Local : - i : entier
- minimum : entier
Sortie : void
Début
| **Soit** minimum = lesNotes[0]
| **Pour** i **de** 1 **à** taille − 1 **avec un pas de** +1 **Faire**
| | **Si** lesNotes[i] < minimum **Alors**
| | | minimum = lesNotes[i]
| | **Fin Si**
| **Fin Pour**
| AfficherEntier(minimum)
Fin

Algorithme 5.3 : AffichageNotesInverse
Principe : idem à l'algorithme 1 avec un parcours de la fin (taille − 1) au début (0)
Entrée : - lesNotes : tableau 1D de taille entiers
- taille : entier
Local : - i : entier
Sortie : void
Début
| **Pour** i **de** taille − 1 **à** 0 **avec un pas de** -1 **Faire**
| | AfficherEntier(lesNotes[i])
| **Fin Pour**
Fin

Algorithme 5.4 : AffichageNotesSuperieuresA12
Principe : Parcourir les notes de l'indice 0 à taille − 1. Si la valeur de la case est supérieure à 12 afficher.

```
Entrée :    - lesNotes : tableau 1D de taille entiers
            - taille : entier
Local :     - i : entier
Sortie :    void
Début
|   Pour i de 0 à taille − 1 avec un pas de +1 Faire
|   |   Si lesNotes[i] ≥ 12 Alors
|   |   |   AfficherEntier(lesNotes[i])
|   |   Fin Si
|   Fin Pour
Fin
```

Algorithme 5.5 : AffichageMoyenneNotesSuperieursA10
Principe : Initialiser une somme à 0 et un compteur à 0. Parcourir les cases du tableau de l'indice 0 à taille − 1. Si la valeur de la case est supérieure à 10, ajouter 1 au compteur et la valeur de la case à la somme. Calculer la somme / compteur si le compteur est > 0 et afficher.

```
Entrée :    - lesNotes : tableau 1D de taille entiers
            - taille : entier
Local :     - i : entier
            - moyenne : réel
            - compteur : entier
Sortie :    void
Début
|   Soit moyenne = 0.0
|   Soit compteur = 0
|   Pour i de 0 à taille − 1 avec un pas de +1 Faire
|   |   Si lesNotes[i] ≥ 10 Alors
|   |   |   moyenne = moyenne + lesNotes[i]
|   |   |   compteur = compteur + 1
|   |   Fin Si
|   Fin Pour
|   Si compteur > 0 Alors
|   |   moyenne = moyenne / compteur
|   |   AfficherRéel(moyenne)
|   Sinon
|   |   AfficherChaîne("Aucune note supérieure à 10 !")
|   Fin Si
Fin
```

Algorithme 5.6 : AffichageIndicePremier6
Principe : Parcours du tableau jusqu'à ce que la valeur 6 soit rencontrée ou la fin du tableau. Si un 6 est rencontré, afficher l'indice courant et quitter la boucle.

```
Entrée :    - lesNotes : tableau 1D de taille entiers
            - taille : entier
Local :     - i : entier
            - indicePremier6 : entier
            - trouve6 : booléen
Sortie :    void
Début
|   Soit indicePremier6 = -1
|   Soit trouve = faux
|   Soit i = 0
|   Tant Que i < taille OU trouve6 == faux Faire
|   |   Si lesNotes[i] == 6 Alors
|   |   |   indicePremier6 = i
|   |   |   trouve6 = vrai
|   |   Fin Si
|   |   i = i + 1
|   Fin Tant Que
|   AfficherEntier(indicePremier6)
Fin
```

Algorithme 5.7 : AffichageIndiceDernier10
Principe : idem à l'algorithme 5.6 mais avec une recherche à l'envers (comme l'algorithme 5.3)

```
Entrée :        - lesNotes : tableau 1D de taille entiers
                - taille : entier
Local :         - i : entier
                - indiceDernier10 : entier
                - trouve10 : booléen
Sortie :        void
Début
|       Soit trouve10 = faux
|       Soit indiceDernier10 = -1
|       Soit i = taille - 1
|       Tant Que i ≥ 0 OU trouve10 == faux Faire
|       |       Si lesNotes[i] == 10 Alors
|       |       |       indiceDernier10 = i
|       |       |       trouve10 = vrai
|       |       Fin Si
|       |       i = i – 1
|       Fin Tant Que
|       AfficherEntier(indiceDernier10)
Fin
```

Algorithme 5.8 : AffichagePresenceValeur20
Principe : Parcours du tableau de l'indice 0 à taille – 1 jusqu'à soit atteindre la fin (20 non présent) soit trouver la valeur dans la case courante. S'arrêter quand 20 est rencontré et afficher « présence » sinon « absence ».

```
Entrée :        - lesNotes : tableau 1D de taille entiers
                - taille : entier
Local :         - i : entier
                - valeur20Existe : booléen
Sortie :        void
Début
|       Soit valeur20Existe = faux
|       Soit i = 0
|       Tant Que i < taille OU valeur20Existe == faux Faire
|       |       Si lesNotes[i] == 20 Alors
|       |       |       valeur20Existe = vrai
|       |       Fin Si
|       Fin Pour
|       Si valeur20Existe == vrai Alors
|       |       AfficherChaîne("La valeur 20 est présente dans le tableau.")
|       Sinon
|       |       AfficherChaîne("La valeur 20 n'est pas présente dans le tableau.")
|       Fin Si
Fin
```

Algorithme 6 : ConversionDecimal8Bits
Principe : Parcourir les puissances de 2 de 2^7 à 2^0 et vérifier si la division du nombre est supérieure ou égale à 1. Si oui mettre 1 dans la case correspondant à l'indice de la puissance et retirer la puissance au nombre, sinon mettre 0. Afficher le tableau à l'envers.

```
Entrée :        - nombre : entier
Local :         - reste : entier
                - i : entier
                - puissances2 : tableau 1D de 8 entiers
Sortie :        void
Début
|       Si nombre < 0 OU nombre > 255 Alors
|       |       AfficherChaîne("Erreur, nombre invalide !")
|       Sinon
|       |       Soit puissances2 = tableau 1D de 8 entiers initialisés à 0
|       |       Soit reste = nombre
|       |       Pour i de 7 à 1 avec un pas de -1 Faire
|       |       |       Si reste / 2^i != 0 Alors // division entière
|       |       |       |       puissances2[i] = 1
|       |       |       |       reste = reste - 2^i
|       |       |       Fin Si
|       |       Fin Pour
|       |       Si reste == 1 Alors
|       |       |       puissances2[0] = 1
|       |       Fin Si
|       |       Pour i de 7 à 0 avec un pas de -1 Faire
|       |       |       AfficherEntier(puissances2[i])
|       |       Fin Pour
|       Fin Si
Fin
```

Algorithme 7 : RechercheIndiceInsertionTableauEntierPartiellementInitialiseEtTrieCroissant
Principe : Parcourir les indices de 0 à nVal − 1. Si la valeur de la case courante est strictement supérieure à la valeur à insérer, sortir de la boucle et retourner cet indice. Sinon retourner nVal.

```
Entrée :        - tab : tableau 1D de taille entiers
                - taille : entier
                - nVal : entier
                - nombre : entier
Local :         - indice : entier
                - indiceInsertion : entier
                - quitter : booléen
Sortie :        void
Début
|       Soit indice = 0
|       Soit indiceInsertion = 0
|       Soit quitter = faux
|       Tant Que indice < nVal OU quitter == faux Faire
|       |       Si tab[indice] > nombre Alors
|       |       |       indiceInsertion = indice
|       |       |       quitter = vrai
|       |       Fin Si
|       |       indice = indice + 1
|       Fin Tant Que
|       AfficherEntier(indiceInsertion)
Fin
```

Algorithme 8 : DecalerEtInsererValeurTableauEntierPartiellementInitialiseEtTrieCroissant
Principe : Décaler les valeurs d'une case vers la droite en allant de la droite (nVal) vers la gauche (indiceInsertion).
Entrée : - tab : tableau 1D de taille entiers
- taille : entier
- nVal : entier
- indiceInsertion : entier
- nombre : entier
Local : indice : entier
Sortie : void
Début

```
|       Pour indice de nVal à indiceInsertion − 1 avec un pas de -1 Faire
|       |       tab[indice] = tab[indice − 1]
|       Fin Pour
|       tab[indiceInsertion] = nombre
|       nVal = nVal + 1
```

Fin

Algorithme 9 : RemplirTableauEntierPartiellementInitialiseEtTrieCroissant
Principe : Demander des nombres et les insérer en utilisant les algorithmes 7 et 8 jusqu'à ce que nVal soit égal à la taille du tableau en partant d'un tableau vide (nVal = 0).
Entrée : /
Local : - tab : tableau 1D de taille entiers
- taille : entier
- nVal : entier
- indiceInsertion : entier
- nombre : entier
- compteur : entier
Sortie : void
Début

```
|       Soit tab = tableau 1D de taille entiers initialisés à 0
|       Soit nVal = 0
|       Pour compteur de 1 à 10 avec un pas de +1 Faire
|       |       nombre = SaisirNombreEntier()
|       |       // recherche de l'indice (algorithme 7)
|       |       Soit indice = 0
|       |       Soit indiceInsertion = 0
|       |       Soit quitter = faux
|       |       Tant Que indice < taille OU quitter == faux Faire
|       |       |       Si tab[indice] > nombre Alors
|       |       |               indiceInsertion = indice
|       |       |               quitter = vrai
|       |       |       Fin Si
|       |       |       indice = indice + 1
|       |       Fin Tant Que
|       |       // décalage + insertion (algorithme 8)
|       |       Pour indice de nVal à indiceInsertion − 1 avec un pas de -1 Faire
|       |       |       tab[indice] = tab[indice − 1]
|       |       Fin Pour
|       |       tab[indiceInsertion] = nombre
|       |       nVal = nVal + 1
|       |       // affichage du tableau
|       |       Pour i de 0 à taille − 1 avec un pas de +1 Faire
|       |       |       AfficherEntier(tab[i])
|       |       Fin Pour
|       Fin Pour
```

Fin

Algorithme 10 : RechercheDichotomiqueTableauEntier1D
Principe : Maintenir trois variables à jour : position de début, de fin et de milieu du tableau. Tant que fin − début ≥ 0, vérifier la valeur de la case du milieu (= (debut + fin) / 2). Soit elle contient le nombre recherché alors on s'arrête, soit la valeur est plus petite alors on continue la recherche en mettant à jour la borne de début, soit elle est plus grande alors on continue en mettant à jour la borne de fin. Si on sort de la boucle sans trouver alors la valeur n'est pas présente.

```
Entrée :        - tab : tableau 1D de taille entiers
                - taille : entier
                - valeur : entier
Local :         - debut : entier
                - milieu : entier
                - fin : entier
                - trouve : booléen
                - indice : entier
                - i : entier
Sortie :        void
Début
|       Soit debut = 0
|       Soit fin = taille − 1
|       Soit milieu = (debut + fin) / 2
|       Soit indice = -1
|       Soit trouve = faux
|       Tant Que fin − debut ≥ 0 OU trouve == faux Faire
|       |       milieu = (debut + fin) / 2
|       |       Si tab[milieu] < valeur Alors
|       |       |       debut = milieu + 1
|       |       Sinon Si tab[milieu] > valeur Alors
|       |       |       fin = milieu − 1
|       |       Sinon // équivaut à tester si tab[milieu] == valeur
|       |       |       indice = milieu
|       |       |       trouve = vrai
|       |       Fin Si
|       Fin Tant Que
|       Si trouve == vrai Alors
|       |       AfficherChaîne("La valeur {valeur} est présente dans le tableau à l'indice {indice}.")
|       Sinon
|       |       AfficherChaîne("La valeur {valeur} n'est pas présente dans le tableau.")
|       Fin Si
Fin
```