Bibliothèque des Tableaux

Environnement de la Bibliothèque

Constantes :

max = 1000

maxl = 1000

maxc = 1000

Type :

tab1D = tableau [1..max] d'entier

tab2D = tableau [1..maxl,1..maxc] d'entier

Groupes de Fonctions et Procédures

- Lecture d'un tableau
- Affichage d'un tableau
- Procédures élémentaires
- Recherche de l'indice des extrémums
- Recherche de la valeur des extrémums
- Somme d'un tableau
- Comptage des éléments
- Conversions des tables
- Insertion et suppression dans les tableaux
- Tri des tableaux d'une dimension
- Tri dans les tableaux à deux dimensions
- Recherche d'indice dans les tableaux d'une dimension triées

Lecture D'un Tableau

- Lect1D
- Lect1Dalea
- Lect2D
- Lect2Dalea
- LectNtab

Menu Principal

Affichage d'un Tableau

- Ecri1D
- Ecri2D

Menu Principal

Procédures élémentaires

- Permute

Menu Principal

Recherche de l'indice des extrémums

- Ind_petit
- Ind_grand
- Lig_mincol
- Lig_maxcol
- Col_minlig
- Col_maxlig

Menu Principal

Recherche de la valeur des extrémums

- Mintab1D
- Maxtab1D
- Mintab2D
- Maxtab2D

Menu Principal

Somme d'un tableau

- Sommetab1D
- Sommecol
- Sommelig

Menu Principal

Comptage des éléments

- Comptinf
- FreqV1D
- FreqV2D
- FreqVlig
- FreqVcol

Menu Principal

Conversions des tables

- Convert2to1
- Convert1to2
- Invert1D
- Transpose

Menu Principal

Insertion et suppressions dans les tableaux

- Insertcase
- Delcase
- Dellig
- Delcol

Menu Principal

Tri des Tableaux d'une dimension

- Tri_select
- Tri_transp
- Tri_bulle
- Tri_compt3
- Tri_compt2
- Tri_compt1

Menu Principal

Tri dans un Tableau de deux dimensions

- Tri2dim
- Trilig
- Tricol

Menu Principal

Recherche d'indice dans les tableaux d'une dimension triées

- Rechdich

Menu Principal

Procédure

Rôle : Permet de Remplir un tableau **T** d'une dimension avec **n** éléments (*n = taille effective*).

Modules Utilisées :

- Aucun Modules

Algorithme & Syntaxe de la déclaration

Procédure **Lect1D** (var **T** : tab1D; var **n** : entier)

Variable **i** : entier

Debut

Ecrire(‘ n = ‘)

Lire(**n**)

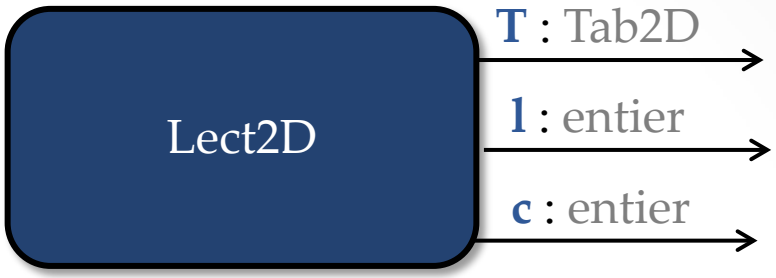
Pour **i** allant de 1 à **n** faire lire(**T[i]**)

Fin

Rôle

Groupe Modules

Procédure



Rôle : Permet de Remplir un tableau **T** de deux dimensions de **l** lignes, **c** colonnes.

Modules Utilisées :

- Aucun Modules

Algorithme & Syntaxe de la déclaration

Procédure **Lect2D** (var **T** : tab2D; var **l , c** : entier)

Variable **i, j** : entier

Debut

Ecrire(‘ l = ‘)

Lire(**l**)

Ecrire(‘C = ‘)

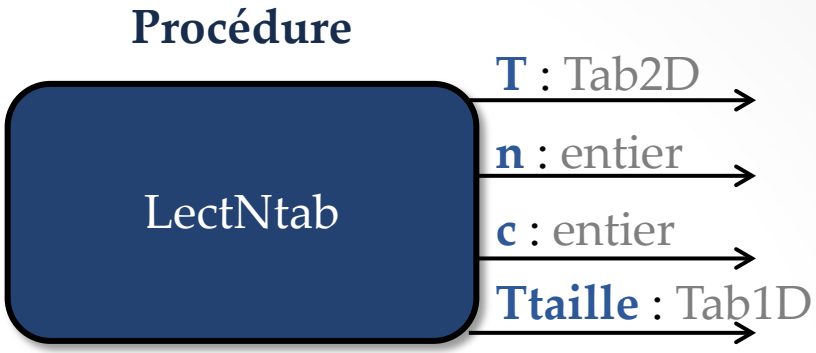
Lire(**C**)

Pour **i** allant de 1 à **l** faire pour **j** allant de 1 à **c** faire lire(**T[i, j]**)

Fin

Rôle

Groupe Modules



Rôle : Permet de Remplir **n** tableaux dont chacun à sa propre taille effective sauvegardé dans le tableau **Ttaille** (*Ttaille [i] = taille effective du tableau i ; C = Maximum des tailles*)

Modules Utilisées :

- Aucun Modules

Algorithme & Syntaxe de la déclaration

Procédure **LectNtab** (var **T** : tab2D; var **n, c** : entier; var **Ttaille** : tab1D)

Variable **i, j** : entier

Debut

Ecrire(‘ entrez le nombre de tableaux, n = ‘)

Lire(**n**)

C ← 1

Pour **i** allant de 1 à **n** faire

Debut

lire(**Ttaille[i]**)

si C < **Ttaille[i]** alors **C** ← **Ttaille[i]**

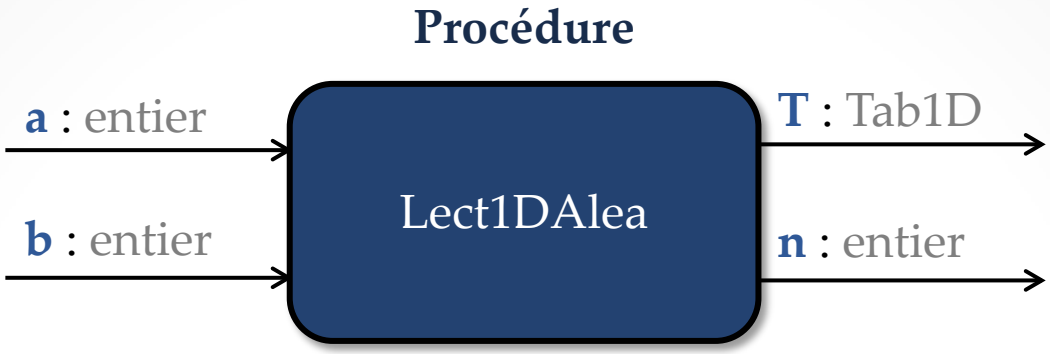
Pour **j** allant de 1 à **Ttaille[i]** faire lire(**T[i , j]**)

Fin

Fin

Rôle

Groupe Modules



Rôle : Permet de Remplir aléatoirement un tableau **T** d'une dimension avec **n** éléments compris entre **a** et **b** .

Modules Utilisées :

- Aucun Modules

Algorithme & Syntaxe de la déclaration

Procédure **Lect1Dalea** (var **T** : tab1D; var **n** : entier ; **A, B** : entier)

Variable **i** : entier

Debut

Ecrire('n = ')

lire(n)

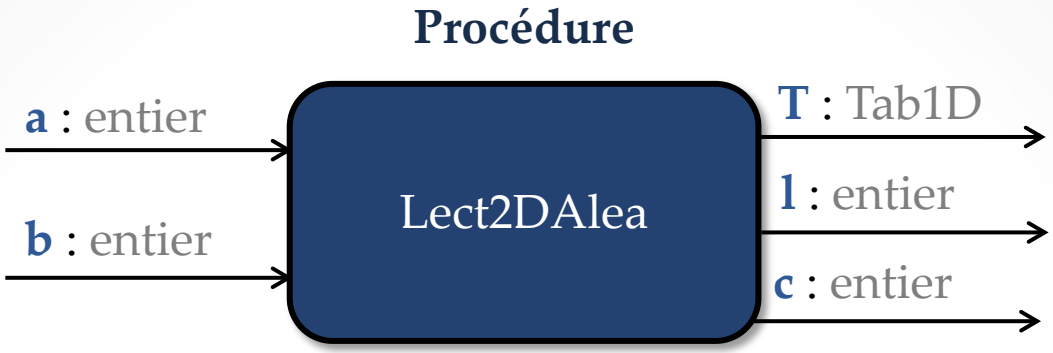
randomize

pour **i**:=1 to **n** do **T[i]** := random(**B**-**A**+1) + **A**

Fin

Rôle

Groupe Modules



Rôle : Permet de Remplir aléatoirement un tableau **T** de deux dimensions de **l** lignes, **c** colonnes avec des éléments compris entre **a** et **b**

Modules Utilisées :

- Aucun Modules

Algorithme & Syntaxe de la déclaration

Procédure **Lect2Dalea** (var **T** : tab2D; var **l, c** : entier ; **A, B** : entier)

Variable **i , j** : entier

Debut

Ecrire('l = ')

Lire (**l**)

Ecrire('C = ')

Lire(**C**)

randomize;

pour **i** allant de 1 à **l** faire pour **j** allant de 1 à **C** faire

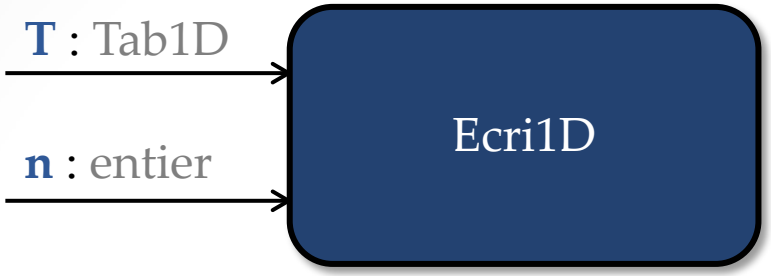
T[i, j] ← random(**B**-**A**+1) + **A**

Fin

Rôle

Groupe Modules

Procédure



Rôle : Permet d'afficher un tableau **T** de **n** éléments

Modules Utilisées :

- Aucun Modules

Algorithme & Syntaxe de la déclaration

Procédure **Ecri1D** (**T** : tab1D; **n** : entier)

Variable **i** : entier

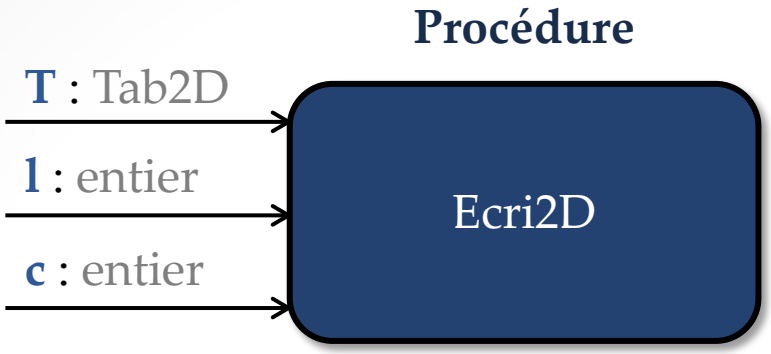
Debut

Pour **i** allant de 1 à **n** faire Ecrire('T [',i,'] = ', **T[i]**)

Fin

Rôle

Groupe Modules



Rôle : Permet d'afficher un tableau T de l lignes, C colonnes.

Modules Utilisées :

- Aucun Modules

Algorithme & Syntaxe de la déclaration

Procédure **Ecri2D** (**T** : tab2D; **l , c** : entier)

Variable **i, j** : entier

Debut

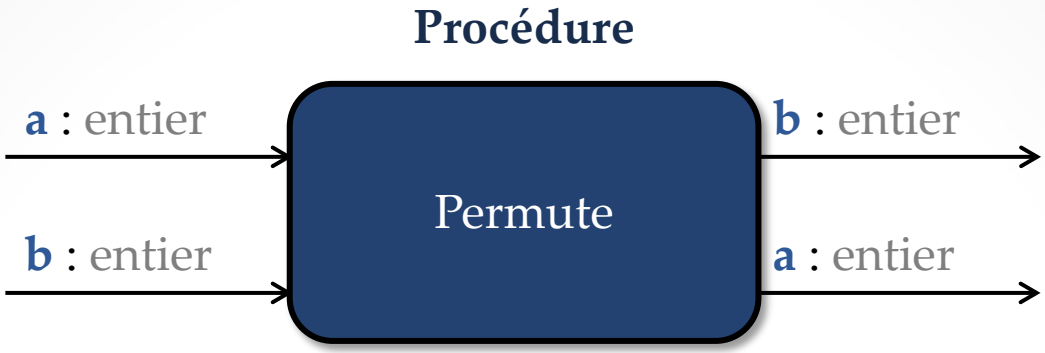
Pour **i** allant de 1 à **l** faire Pour **j** allant de 1 à **c** faire

Ecrire('T [',i ,',' ,j,'] = ', **T[i , j]**)

Fin

Rôle

Groupe Modules



Rôle : Permute entre **a** et **b**

Modules Utilisées :

- Aucun Modules

Procédure **Permute** (var **a , b** : entier)

Variable **c** : entier

Debut

c ← **a**

a ← **b**

b ← **c**

Fin

Fonction

Rôle : Renvoi l'indice du plus petit élément situé entre l'indice **a** et l'indice **b** d'un tableau **T** d'une dimension,

Modules Utilisées :

- Aucun Modules

Algorithme & Syntaxe de la déclaration

Fonction **Ind_petit** (**T** : tab1D; **a**, **b** : entier) : entier

Variable **i, inter** : entier

Debut

inter ← **a**

Pour **i** allant de **a**+1 à b faire si **T[i]** < **T[inter]** alors **inter** ←**i**

Ind_petit ← **inter**

Fin

Rôle

Groupe Modules

Fonction

Rôle : Renvoi l'indice du plus grand élément situé entre l'indice **a** et l'indice **b** d'un tableau **T** d'une dimension,

Modules Utilisées :

- Aucun Modules

Algorithme & Syntaxe de la déclaration

Fonction **Ind_grand** (**T :** tab1D**; a, b** : entier) : entier

Variable **i, inter** : entier

Debut

inter ← **a**

Pour **i** allant de **a**+1 à b faire si **T[i]** > **T[inter]** alors **inter** ←**i**

Ind_grand ← **inter**

Fin

Rôle

Groupe Modules

Fonction

Rôle : Renvoi la ligne du plus petit élément de la colonne **Col** d'un tableau **T** de deux dimensions de **l** lignes.

Modules Utilisées :

- Aucun Modules

Algorithme & Syntaxe de la déclaration

Fonction **lig_mincol** (**T** : tab2D; **l**, **col** : entier) : entier

Variable **i** : entier

Debut

lig_mincol ← 1

Pour **i** allant de 2 à **l** faire si **T[i,Col]** < **T[lig_mincol,col]** alors **lig_mincol** ← **i**

Fin

Rôle

Groupe Modules

Fonction

Rôle : Renvoi la ligne du plus grand élément de la colonne **Col** d'un tableau **T** de deux dimensions de **l** lignes.

Modules Utilisées :

- Aucun Modules

Algorithme & Syntaxe de la déclaration

Fonction **lig_maxcol** (**T** : tab2D; **l**, **col** : entier) : entier

Variable **i** : entier

Debut

lig_maxcol ← 1

Pour **i** allant de 2 à **l** faire si **T[i,Col]** > **T[lig_maxcol,col]** alors **lig_maxcol** ← **i**

Fin

Rôle

Groupe Modules

Fonction

Rôle : Renvoi la colonne du plus petit élément de la ligne **lig** d'un tableau **T** de deux dimensions de **C** colonnes.

Modules Utilisées :

- Aucun Modules

Algorithme & Syntaxe de la déclaration

Fonction **col_minlig** (**T** : tab2D; **lig** ,**c** : entier) : entier

Variable **j** : entier

Debut

col_minlig ← 1

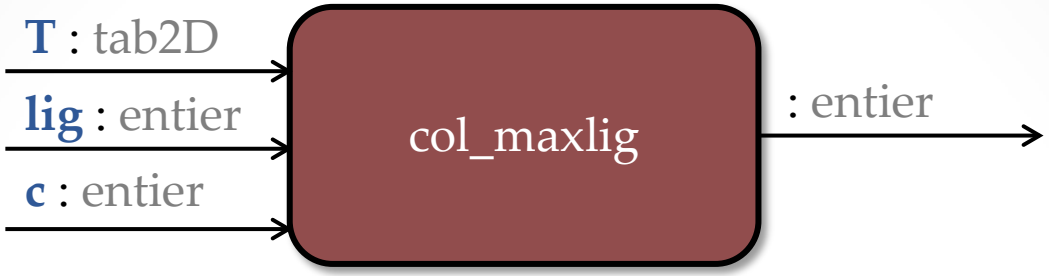
Pour **j** allant de 2 à **c** faire si **T[lig,j]** < **T[lig,col_minlig]** alors **col_minlig** ← **j**

Fin

Rôle

Groupe Modules

Fonction



Rôle : Renvoi la colonne du plus grand élément de la ligne **lig** d'un tableau **T** de deux dimensions de **C** colonnes.

Modules Utilisées :

- Aucun Modules

Algorithme & Syntaxe de la déclaration

Fonction **col_maxlig** (**T** : tab2D; **lig** ,**c** : entier) : entier

Variable **j** : entier

Debut

col_maxlig ← 1

Pour **j** allant de 2 à **c** faire si **T[lig,j]** > **T[lig,col_maxlig]** alors **col_maxlig** ← **j**

Fin

Rôle

Groupe Modules

Fonction

Rôle : Renvoi la valeur du plus petit élément d'un tableau **T** de **n** éléments.

Modules Utilisées :

- Aucun Modules

Algorithme & Syntaxe de la déclaration

Fonction **Mintab1D** (**T** : tab1D; **n** : entier) : entier

Variable **i**, **inter** : entier

Debut

inter ← **T[1]**

pour **i** allant de 1 à **n** faire si **inter** > **T[i]** alors **inter** := **T[i]**

mintab1D ← **inter**

Fin

Rôle

Groupe Modules

Fonction

Rôle : Renvoi la valeur du plus grand élément d'un tableau **T** de **n** éléments.

Modules Utilisées :

- Aucun Modules

Algorithme & Syntaxe de la déclaration

Fonction **Maxtab1D** (**T** : tab1D; **n** : entier) : entier

Variable **i**, **inter** : entier

Debut

inter ← **T[1]**

pour **i** allant de 1 à **n** faire si **inter** < **T[i]** alors **inter** := **T[i]**

maxtab1D ← **inter**

Fin

Rôle

Groupe Modules

Rôle : Renvoi la valeur du plus petit élément d'un tableau **T** de **l** lignes, **c** colonnes.

Modules Utilisées :

- Aucun Modules

Algorithme & Syntaxe de la déclaration

Fonction **Mintab2D** (**T** : tab2D; **l, c** : entier) : entier

Variable **i**, **j**, **inter** : entier

Debut

inter ← **T[1,1]**

pour **i** allant de 1 à **l** faire pour **j** allant de 1 à **c** faire

si **inter** > **T[i,j]** alors **inter** := **T[i,j]**

mintab2D ← **inter**

Fin

Rôle

Groupe Modules

Rôle : Renvoi la valeur du plus grand élément d'un tableau **T** de **l** lignes, **c** colonnes.

Modules Utilisées :

- Aucun Modules

Algorithme & Syntaxe de la déclaration

Fonction **Maxtab2D** (**T** : tab2D; **l, c** : entier) : entier

Variable **i**, **j**, **inter** : entier

Debut

inter ← **T[1,1]**

pour **i** allant de 1 à **l** faire pour **j** allant de 1 à **c** faire

si **inter** < **T[i,j]** alors **inter** := **T[i,j]**

maxtab2D ← **inter**

Fin

Rôle

Groupe Modules

Rôle : Renvoi la somme de **n** éléments d'un tableau **T** d'une dimension.

Modules Utilisées :

- Aucun Modules

Algorithme & Syntaxe de la déclaration

Fonction **Sommetab1D** (**T** : tab1D; **n** : entier) : entier

Variable **i**, **inter** : entier

Debut

inter ← 0

pour **i** allant de 1 à **n** faire **inter** ← **inter** + **T[i]**

Sommetab1D ← **inter**

Fin

Rôle

Groupe Modules

Rôle : Renvoi la somme des éléments de la colonne **col** d'un tableau **T** de deux dimensions de **l** lignes.

Modules Utilisées :

- Aucun Modules

Algorithme & Syntaxe de la déclaration

Fonction **Sommecol** (**T** : tab2D; **l, col** : entier) : entier

Variable **i**, **inter** : entier

Debut

inter ← 0

pour **i** allant de 1 à **l** faire **inter** ← **inter** + **T[i,col]**

Sommecol ← **inter**

Fin

Rôle

Groupe Modules

Rôle : Renvoi la somme des éléments de la ligne **lig** d'un tableau **T** de deux dimensions de **C** colonnes.

Modules Utilisées :

- Aucun Modules

Algorithme & Syntaxe de la déclaration

Fonction **Sommelig** (**T** : tab2D; **lig, c** : entier) : entier

Variable **j**, **inter** : entier

Debut

inter ← 0

pour **j** allant de 1 à **c** faire **inter** ← **inter** + **T[lig,j]**

Sommelig ← **inter**

Fin

Rôle

Groupe Modules

Fonction

Rôle : Renvoi le nombre d'éléments strictement inferieur de **V** d'un tableau **T** de **n** éléments.

Modules Utilisées :

- Aucun Modules

Algorithme & Syntaxe de la déclaration

Fonction **Comptinf** (**T** : tab1D; **n, V** : entier) : entier

Variable **i**, **inter** : entier

Debut

inter ← 0

pour **i** allant de 1 à **n** faire si **T[i]** < **V** alors **inter** ← **inter** + **1**

Comptinf ← **inter**

Fin

Rôle

Groupe Modules

Fonction

Rôle : Renvoi la fréquence d'apparition des éléments identique à **V** d'un tableau **T** de **n** éléments.

Modules Utilisées :

- Aucun Modules

Algorithme & Syntaxe de la déclaration

Fonction **FreqV1D** (**T** : tab1D; **n, V** : entier) : entier

Variable **i**, **inter** : entier

Debut

inter ← 0

pour **i** allant de 1 à **n** faire si **T[i]** = **V** alors **inter** ← **inter** + **1**

FreqV1D ← **inter**

Fin

Rôle

Groupe Modules

Rôle : Renvoi la fréquence d'apparition des éléments identique à **V** d'un tableau **T** de deux dimensions de **l** lignes, **c** colonnes.

Modules Utilisées :

- Aucun Modules

Algorithme & Syntaxe de la déclaration

Fonction **FreqV2D** (**T** : tab2D; **l, c, V** : entier) : entier

Variable **i**, **j**, **inter** : entier

Debut

inter ← 0

pour **i** allant de 1 à **l** faire pour **j** allant de 1 à **c** faire

si **T[i, j]** = **V** alors **inter** ← **inter** + **1**

FreqV2D ← **inter**

Fin

Rôle

Groupe Modules

Rôle : Renvoi la fréquence d'apparition des éléments identique à **V** dans la ligne **lig** dans un tableau **T** de deux dimensions de **c** colonnes.

Modules Utilisées :

- Aucun Modules

Algorithme & Syntaxe de la déclaration

Fonction **FreqVlig** (**T** : tab2D; **lig, c, V** : entier) : entier

Variable **j**, **inter** : entier

Debut

inter ← 0

pour **j** allant de 1 à **c** faire si **T[lig, j]** = **V** alors **inter** ← **inter** + **1**

FreqVlig ← **inter**

Fin

Rôle

Groupe Modules

Rôle : Renvoi la fréquence d'apparition des éléments identique à **V** dans la colonne **col** dans un tableau **T** de deux dimensions de **l** lignes.

Modules Utilisées :

- Aucun Modules

Algorithme & Syntaxe de la déclaration

Fonction **FreqVcol** (**T** : tab2D; **l, col, V** : entier) : entier

Variable **i**, **inter** : entier

Debut

inter ← 0

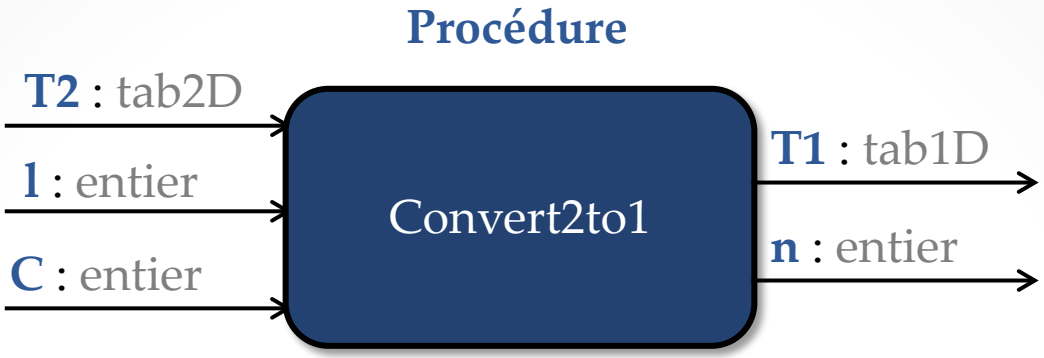
pour **i** allant de 1 à **l** faire si **T[i, col]** = **V** alors **inter** ← **inter** + **1**

FreqVcol ← **inter**

Fin

Rôle

Groupe Modules



Rôle : Transforme un tableau **T2** de **l** lignes, **c** colonnes en un tableau **T1** d'une dimension de **n** éléments.

Modules Utilisées :

- Aucun Modules

Algorithme & Syntaxe de la déclaration

Procédure **Convert2to1** (**T2 :** tab2D**;** **l, c** : entier; var **T1 :** tab1D**;** var **n** : entier)

Variable **i**, **j**: entier

Debut

inter ← 0

pour **i** allant de 1 à **l** faire pour **j** allant de 1 à **c** faire

T1[j + (i - 1)*C] ← **T2[i ,j]**

n ← **l** * **c**

Fin

Procédure

Rôle : Transforme un tableau **T1** de **n** éléments, en un tableau **T2** de deux dimensions de **l** lignes (*l = entrée indiquant le découpage du tableau*), **c** colonnes (*C = se calcule automatiquement en fonction du découpage*)

Modules Utilisées :

- Aucun Modules

Algorithme & Syntaxe de la déclaration

Procédure **Convert1to2** (**T1** **:** tab1D**;** **n** : entier; var **T2** **:** tab2D; **l** : entier**,** var **c** : entier)

Variable **i** : entier

Debut

si (**n** mod **l**) = 0 then **c** ← **n** div **l** si non **c** ← **n** div **l** + 1

pour **i** allant de 1 à **l*****c** faire **T2[**(**i**-1) div **c** + 1,(**i**-1) mod **c** + 1**]** ← 0

pour **i** allant de 1 à **n** faire **T2[**(**i**-1) div **c** + 1,(**i**-1) mod **c** + 1] ← **T1[i]**

Fin

Rôle

Groupe Modules

Procédure

Rôle : Inverser un tableau **T** de **n** éléments

Modules Utilisées :

- Permute

Algorithme & Syntaxe de la déclaration

Procédure **Invert1D** (var **T** : tab1D; **n** : entier)

Variable **i** : entier

Debut

pour **i** allant de 1 à **n** div **2** faire

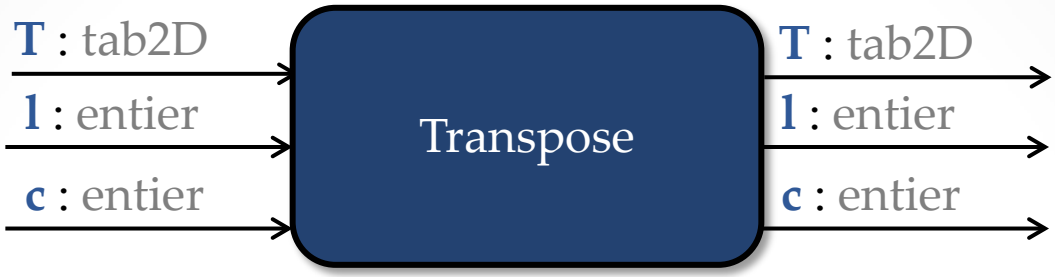
Permute (**T[n-i+1) , T[i]**)

Fin

Rôle

Groupe Modules

Procédure



Rôle : Transpose un tableau **T** de **l** lignes, **c** colonnes

Modules Utilisées :

- Permute

Algorithme & Syntaxe de la déclaration

Procédure **Transpose** (var **T** : tab2D;var **l, C** : entier)

Variable **i, j, LC** : entier

Debut

si **l** > **C** alors **LC** ← **l** si non **LC** ← **C**

Pour **i** allant de 1 à **LC** faire pour **j** allant de 1 à **LC** faire

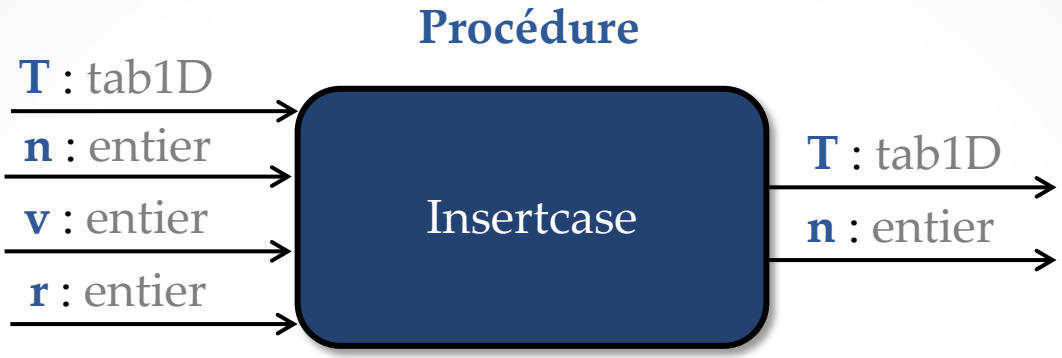
si **j** > **i** alors permute(**T[i , j]** ,**T[j , i]**)

permute(**l**, **c**)

Fin

Rôle

Groupe Modules



Rôle : Insérer un élément **v** au rang **r** dans un tableau **T** d'une dimension de **n** éléments (*La taille effective du tableau devient n+1*)

Modules Utilisées :

- Permute

Algorithme & Syntaxe de la déclaration

Procédure **Insertcase** (var **T** : tab1D; var **n** : entier; **V, R** : entier)

Variable **i** : entier

Debut

si **R** <= **n** alors Pour **i** allant de **R** à **n** + 1 faire permute(**T[i]** ,**V**)

Si non **T[n+1]** ← **V**

n ← **n** + 1

Fin

Rôle

Groupe Modules

Procédure

Rôle : Retire l'élément du rang **r** dans un tableau **T** d'une dimension de **n** éléments (*La taille effective du tableau devient n-1*)

Modules Utilisées :

- Aucun modules

Algorithme & Syntaxe de la déclaration

Procédure **delcase** (var **T** : tab1D; var **n** : entier; **r** : entier)

Variable **i** : entier

Debut

si **R** <= **n** alors

Debut

Pour **i** allant de **R** à **n** - 1 faire **T[i]** ← **T[i+1]**

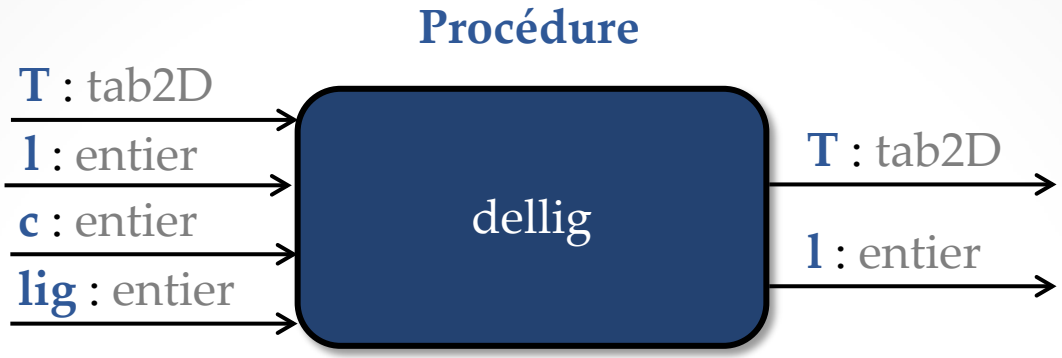
n ← **n** – 1

Fin

Fin

Rôle

Groupe Modules



Rôle : retire la ligne **lig** dans un tableau **T** de **l** lignes, **c** colonnes (*Le nombre effectif des lignes du tableau devient l-1*)

Modules Utilisées :

- Aucun modules

Algorithme & Syntaxe de la déclaration

Procédure **dellig** (var **T** : tab2D; var **l** : entier; **c** : entier; **lig** : entier)

Variable **i**, **j** : entier

Debut

si **lig** <= **l** alors

Debut

Pour **i** allant de **lig** à **l** - 1 faire pour **j** allant de 1 à **c** faire

T[i, j] ← **T[i+1, j]**

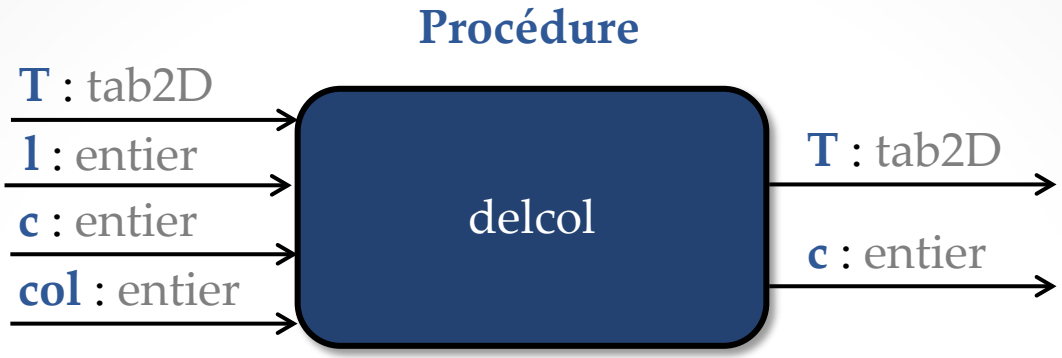
l ← **l** – 1

Fin

Fin

Rôle

Groupe Modules



Rôle : Retire la colonne **col** dans un tableau **T** de **l** lignes, **c** colonnes

(Le nombre effectif des colonnes du tableau devient C-1)

Modules Utilisées :

- Aucun modules

Algorithme & Syntaxe de la déclaration

Procédure **delcol** (var **T** : tab2D; **l** : entier; var **c** : entier; **col** : entier)

Variable **i**, **j** : entier

Debut

si **col** <= **c** alors

Debut

Pour **j** allant de **col** à **c** - 1 faire pour **i** allant de 1 à **l** faire

T[i, j] ← **T[i, j+1]**

c ← **c** – 1

Fin

Fin

Rôle

Groupe Modules

Procédure

Rôle : Tri le Tableau **T** de **n** éléments en utilisant la méthode du

Tri par Sélection

Modules Utilisées :

- Permute
- Ind_petit

Algorithme & Syntaxe de la déclaration

Procédure **Tri_select** (var **T** : tab1D; **n** : entier)

Variable **i** : entier

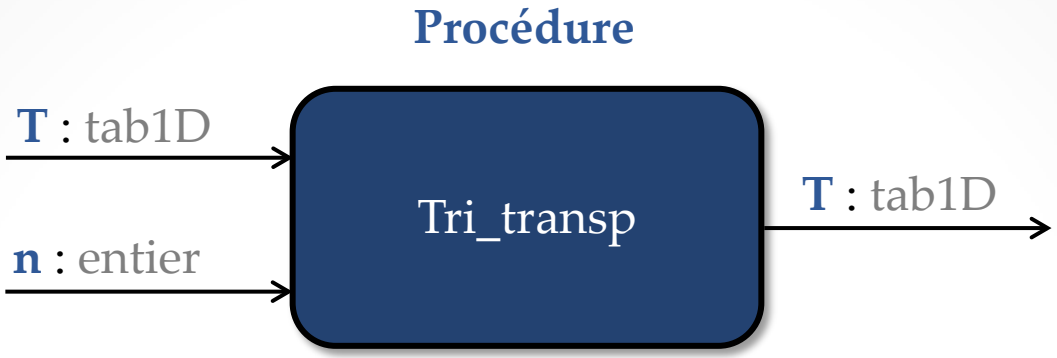
Debut

Pour **i** allant de 1 à **n** - 1 faire Permute (**T[i]** , **T[ind_petit (T , i, n)]**)

Fin

Rôle

Groupe Modules



Rôle : Tri le Tableau **T** de **n** éléments en utilisant la méthode du

Tri par Transposition

Modules Utilisées :

- Permute

Algorithme & Syntaxe de la déclaration

```
Procédure Tri_transp (var T : tab1D;  n : entier )
Variable i, j : entier
Debut
Pour i allant de 1 à n - 1 faire
   Debut
   j ← i
   tant que (T[j]  > T[j+1])  et (j > 0) faire
        Debut
        Permute (T[j]  , T[j+1] )
        j ← j-1
        fin
   fin
Fin
```

Rôle

Groupe Modules

Procédure

Rôle : Tri le Tableau **T** de **n** éléments en utilisant la méthode du

Tri à bulle

Modules Utilisées :

- Permute

Algorithme & Syntaxe de la déclaration

```
Procédure Tri_Bulle (var T : tab1D;  n : entier )
Variable i, m : entier ;  modif : booléen
Debut
modif ← Vrai
m ← n
Tant que Modif ou (m > 2) faire
    Debut
    modif ← Faux
    Pour i allant de 1 à m - 1 faire si T[i] > T[i+1] alors
        debut
        permute(T[i] , T[i+1])
        modif ← Vrai
        fin
    m ← m-1
    fin
Fin
```

Rôle

Groupe Modules

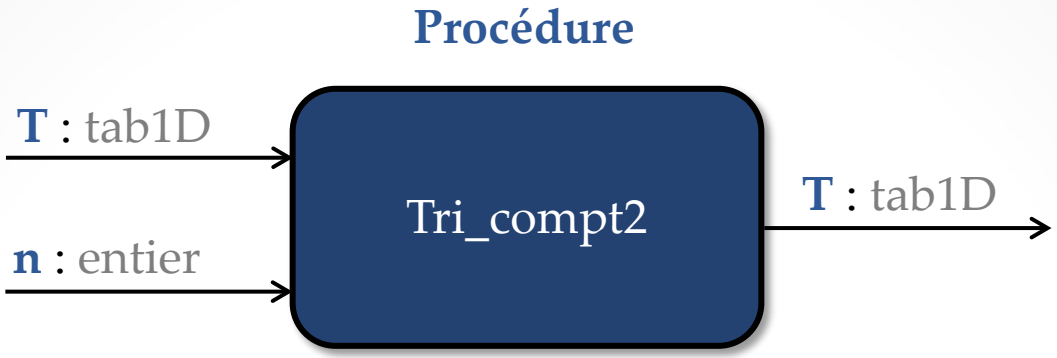
Rôle : Tri le Tableau **T1** de **n** éléments dans **T2** en utilisant la méthode du **Tri par comptage 3 tables**

Modules Utilisées :

- Permute
- Comptinf

Algorithme & Syntaxe de la déclaration

```
Procédure Tri_compt3 (T1 : tab1D;  n : entier ; var T2 : tab1D )
Variable i, j, k : entier ;  Tabcompt : tab1D
Debut
Pour i allant de 1 à n faire tabcompt[i] := comptinf (T1,n,T1[i]);
Pour i allant de 1 à n – 1 faire
   Debut
   k ← 1
   Pour j allant de i + 1 à n faire si tabcompt[i] = tabcompt[j] alors
        Debut
        tabcompt[j] ← tabcompt[j] + k
        k := k+1
        fin
  fin
Pour i allant de 1 à n faire alors T2[tabcompt[i]+1] ← T1[i]
Fin
```



Rôle : Tri le Tableau **T** de **n** éléments en utilisant la méthode du

Tri par Comptage 2 tables

Modules Utilisées :

- Permute
- Comptinf

Algorithme & Syntaxe de la déclaration

```
Procédure Tri_compt2 (var T : tab1D;  n : entier)
Variable i, j, k : entier ;  Tabcompt : tab1D
Debut
Pour i allant de 1 à n faire tabcompt[i] := comptinf (T1,n,T1[i]);
Pour i allant de 1 à n – 1 faire
   Debut
   k ← 1
   Pour j allant de i + 1 à n faire si tabcompt[i] = tabcompt[j] alors
        Debut
        tabcompt[j] ← tabcompt[j] + k
        k := k+1
        fin
  fin
i ← 1
Tant que i<n  faire
    Debut
    tant que i <> tabcompt [i]+1 faire
        Debut
        Permute(T[i],T[tabcompt [i]+1])
        Permute(tabcompt [i],tabcompt [tabcompt [i]+1])
        fin
    i ← i+1
    fin
Fin
```

Rôle

Groupe Modules

Procédure

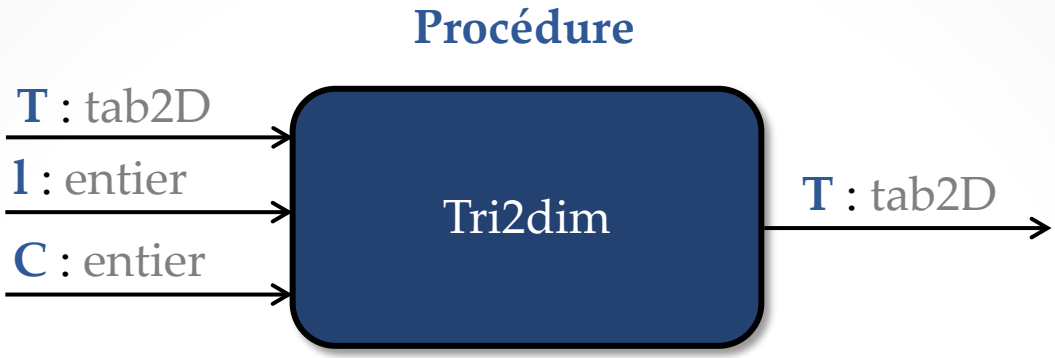
Rôle : Tri le Tableau **T** de **n** éléments en utilisant la méthode du

Tri par Comptage 1 table

Modules Utilisées :

- Permute
- Comptinf

```
Procédure Tri_compt1 (var T : tab1D;  n : entier)
Variable i, cpt, k : entier
Debut
i ← 1
Tant que i < n faire
    Debut
    cpt ← comptinf(T,n,T[i])
    si (cpt+1) <= i alors i ← i + 1 si non
        Debut
        k ← 1
        tant que T[i] = T[cpt+k] do k ← k + 1
        permute (T[i],T[cpt+k])
        fin
    fin
Fin
```



Rôle : Tri le Tableau **T** de **l** lignes, **c** colonnes en utilisant la méthode du **Tri par Comptage 1 table** (*Le tri se fait comme si les lignes étaient placés l'une après l'autre*)

Modules Utilisées :

- Permute
- Comptinf
- Tri_compt1
- Convert2to1
- Convert1to2

Algorithme & Syntaxe de la déclaration

Procédure **Tri2dim** (var **T** : tab1D; **l, c** : entier)

Variable **inter** : tab1D ; **n** : entier

Debut

Convert2to1(**T**, **l**, **C**, **inter**, **n**)

Tri_compt1(**T**, **n**)

Convert1to2(**inter** ,**n** , **T** , **l** , **C**)

Fin

Rôle

Groupe Modules

Rôle : Tri la ligne **lig** a partir d'un tableau **T** de **c** colonnes, en utilisant la méthode du **Tri par Comptage 1 table**

Modules Utilisées :

- Permute
- Comptinf
- Tri_compt1

Algorithme & Syntaxe de la déclaration

Procédure **Trilig** (var **T** : tab2D; **lig, c** : entier)

Variable **j** : entier ; **inter** : tab1D

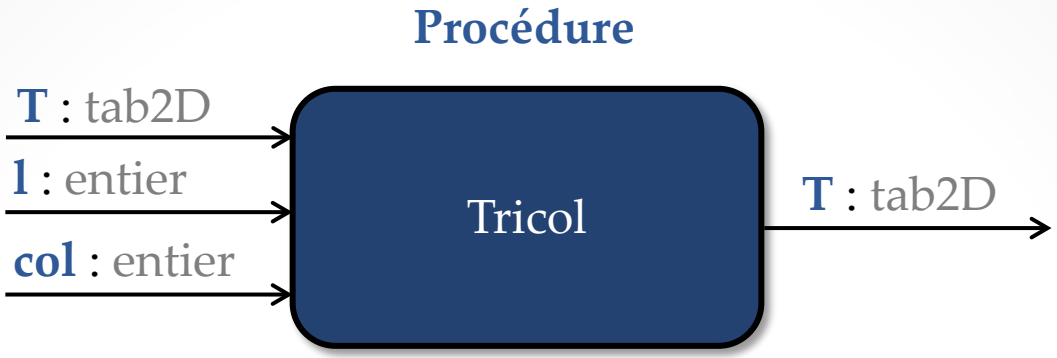
Debut

Pour **j** allant de 1 à **c** faire **inter[j]** ←**T[lig , j]**

Tri_compt1(**inter**, **c**)

Pour **j** allant de 1 à **c** faire **T[lig , j]** ← **inter[j]**

Fin



Rôle : Tri la colonne **col** a partir d'un tableau **T** de **l** lignes, en utilisant la méthode du **Tri par Comptage 1 table**

Modules Utilisées :

- Permute
- Comptinf
- Tri_compt1

Algorithme & Syntaxe de la déclaration

Procédure **Tricol** (var **T** : tab2D; **l, col** : entier)

Variable **i** : entier ; **inter** : tab1D

Debut

Pour **i** allant de 1 à **l** faire **inter[i]** ←**T[i , col]**

Tri_compt1(**inter**, **l**)

Pour **i** allant de 1 à **l** faire **T[i , col]** ← **inter[i]**

Fin

Rôle

Groupe Modules

Fonction

Rôle : Renvoi l'indice de l'élément recherché **V** dans un tableau **T** de **n** éléments s'il existe, sinon la fonction renvoi la valeur -1(*La recherche se fait par méthode dichotomique*)

Modules Utilisées :

- Aucun modules

Algorithme & Syntaxe de la déclaration

```
Fonction Rechdich (var T : tab1D;  n, V : entier) : entier
Variable Binf , Bsup , médiane : entier
Debut
Binf ← 1
Bsup ← n
mediane ← (Binf + Bsup) div 2 ;
while (Bsup >= Binf) et (T[mediane] <> V) do
    Debut
    Si V < T[mediane] alors Bsup ← mediane - 1 si non Binf ← mediane + 1
    mediane ← (Bsup + Binf) div 2
    end
si T[mediane] = V alors rechdich ← mediane si non rechdich ← -1
Fin
```

Rôle

Groupe Modules