

# Parcours imbriqué [tb07] – Exercice résolu

Karine Zampieri, Stéphane Rivière

Unisciel  algoprogram  Version 19 mai 2018

## Table des matières

<b>1</b>	<b>Parcours imbriqué / pgimbrique</b>	<b>2</b>
1.1	Procédure calculerNSuperieurs (nombre de supérieurs) . . . . .	2
1.2	Procédure afficherTab (affichage d'un tableau) . . . . .	3
1.3	Programme de test . . . . .	4
1.4	Fonction ninferieursTab (nombre d'inférieurs) . . . . .	5
<b>2</b>	<b>Références générales</b>	<b>6</b>

## C++ - Parcours imbriqué (Solution)



**Mots-Clés** Tableau unidimensionnel ■

**Utilise** Définitions et notations, Tableaux et paramètres, Parcours de tableaux ■

**Difficulté** ●○○ (30 min) ■

# 1 Parcours imbriqué / pgimbrique



## Objectif

Cet exercice détermine le nombre de successeurs supérieurs pour chaque case d'un tableau d'entiers.



## Définitions C++

```
const int TMAX = ...;
using ITableau = int[TMAX];
```

### 1.1 Procédure calculerNSuperieurs (nombre de supérieurs)

Ce problème calcule, pour chaque case d'un `ITableau`, le nombre de cases suivantes qui contiennent un élément strictement supérieur. Exemple :

```
[45, 54, 1, -56, 22, 134, 49, 12, 90, -27]
==> [ 4 2 5 6 3 0 1 1 0 0 ]
```



Écrivez une fonction `nsuperieursTab(t,k,n,valeur)` qui calcule et renvoie le nombre d'éléments de `t[k..n]` supérieur à `valeur` (entier), où `t` est un `ITableau`.



Déduisez une procédure `calculerNSuperieurs(t,n,trs)` qui, dans un `ITableau trs`, calcule le nombre de valeurs successeurs supérieures à `t[ix]` pour chaque case `ix` des `n` éléments d'un `ITableau t`.



Validez votre fonction et votre procédure avec la solution.

## Solution C++

@[pgimbrique.cpp]

```
/**
 * Nombre de valeurs d'un ITableau supérieurs à une valeur
 * @param[in] t - un ITableau
 * @param[in] k - indice de début de t
 * @param[in] n - nombre de valeurs de t
 * @param[in] valeur - valeur
 * @return le nombre de valeurs de t[k..n] supérieurs à valeur
 */
int nsuperieursTab(const ITableau& t, int k, int n, int valeur)
{
    int rs = 0;
    for (int j = k; j < n; ++j)
    {
```

```

    if (t[j] > valeur)
    {
        ++rs;
    }
}
return rs;
}

/**
 * Nombre de successeurs supérieurs d'un ITableau
 * @param[in] t - un ITableau
 * @param[in] n - nombre de valeurs
 * @param[out] trs - un ITableau résultat
 */
void calculerNSuperieurs(const ITableau& t, int n, ITableau& trs)
{
    for (int j = 0; j < n; ++j)
    {
        trs[j] = nsuperieursTab(t, j + 1, n, t[j]);
    }
}

```

## 1.2 Procédure afficherTab (affichage d'un tableau)



Écrivez le **profil** d'une procédure `afficherTab(t,n)` qui affiche les `n` premières valeurs d'un `ITableau t`.



### Affichage des valeurs

L'affichage de `t` est un parcours complet des `n` valeurs.  
Il est donc piloté par une boucle **Pour**.



Écrivez le corps de la procédure. Affichez les valeurs à la queue-leu-leu séparés par un espace, le tout entre crochet. Exemple :

```
[45 54... -27]
```



Validez votre procédure avec la solution.

### Solution C++ @[UtilsTB.cpp]

```

/**
 * Affichage d'un ITableau
 * @param[in] t - un ITableau
 * @param[in] n - nombre de valeurs dans [0..TMAX]
 */
void afficherTab(const ITableau& t, int n)
{
    cout<<"[";

```

```

for (int j = 0; j < n; ++j)
{
    cout<<t[j]<<" ";
}
cout<<" ]"<<endl;
}

```

### 1.3 Programme de test



**Téléchargez** le fichier suivant et mettez-le dans votre dossier.

C++ @[UtilsTB.cpp]



**Copiez/collez** ensuite les lignes suivantes :

C++ **Au début** de votre programme :

```

#include "UtilsTB.cpp"
using namespace UtilsTB;

```



**Soit** la fonction `saisirTab(t)` qui effectue la saisie contrôlée du nombre de valeurs (entier compris entre 1 et `TMAX`), saisit les valeurs entières dans un `ITableau t` puis renvoie l'entier du nombre de valeurs saisies.

C++ @[saisirTab] (dans UtilsTB)



Écrivez un programme qui calcule, pour chaque case d'un `ITableau` de `nelems` valeurs, le nombre de cases suivantes qui contiennent un élément strictement supérieur. Les résultats seront placés dans un autre `ITableau` puis ce dernier sera affiché.



Testez. Exemple d'exécution :

```

Nombre d'éléments dans [1..50]? 10
t[0]? 45
t[1]? 54
t[2]? 1
t[3]? -56
t[4]? 22
1t[5]? 134
t[6]? 49
t[7]? 12
t[8]? 90
t[9]? -27
[ 4 2 5 6 3 0 1 1 0 0 ]

```



Validez votre programme avec la solution.

**Solution C++** @[pgimbrique.cpp]

```
int main()
{
    ITableau tab;
    int nelems = UtilsTB::saisirTab(tab);
    ITableau trs;
    calculerNSuperieurs(tab, nelems, trs);
    UtilsTB::afficherTab(tab, nelems);
}
```

## 1.4 Fonction `ninferieursTab` (nombre d'inférieurs)

En partant de la fonction et procédure écrites ci-dessus,



Écrivez une fonction `ninferieursTab(t,k,n,valeur)` qui calcule et renvoie le nombre de valeurs inférieures à une valeur `valeur` (entier) dans les `t[k..n]` valeurs d'un `ITableau t`.



Déduisez une procédure `calculerNInferieurs(t,n,trs)` qui calcule dans un `ITableau trs` le nombre de valeurs successeurs inférieures à `t[ix]` pour chaque case `ix` des `n` éléments d'un `ITableau t` d'entiers.



Validez votre fonction et votre procédure avec la solution.

**Solution C++** @[pgimbrique.cpp]

```
/**
 * Nombre de valeurs d'un ITableau inférieurs à une valeur
 * @param[in] t - un ITableau
 * @param[in] k - indice de début de t
 * @param[in] n - nombre de valeurs de t
 * @param[in] valeur - valeur
 * @return le nombre de valeurs de t[k..n] inférieurs à valeur
 */
int ninferieursTab(const ITableau& t, int k, int n, int valeur)
{
    int rs = 0;
    for (int j = k; j < n; ++j)
    {
        if (t[j] < valeur)
        {
            ++rs;
        }
    }
    return rs;
}
```

```
/**
 * Nombre de successeurs inférieurs d'un ITableau
 * @param[in] t - un ITableau
```

```
@param[in] n - nombre de valeurs de t
@param[out] trs - un ITableau résultat
*/

void calculerNInferieurs(const ITableau& t, int n, ITableau& trs)
{
    for (int j = 0; j < n; ++j)
    {
        trs[j] = ninferieursTab(t, j + 1, n, t[j]);
    }
}
```

## 2 Références générales

Comprend ■