

# 1 Exercice 1

Ecrire les algorithmes permettant :

1. Le calcul du nombre d'occurrences d'un élément donné dans un tableau.
2. Le calcul de la moyenne et du minimum des éléments d'un tableau.
3. De tester si un tableau est trié.
4. Le calcul du produit scalaire de deux vecteurs réels  $u$  et  $v$  de dimension  $n$  :  $u.v = \sum_{i=1}^{i=n} u_i * v_i$

Traduire les algorithmes précédents en langage C.

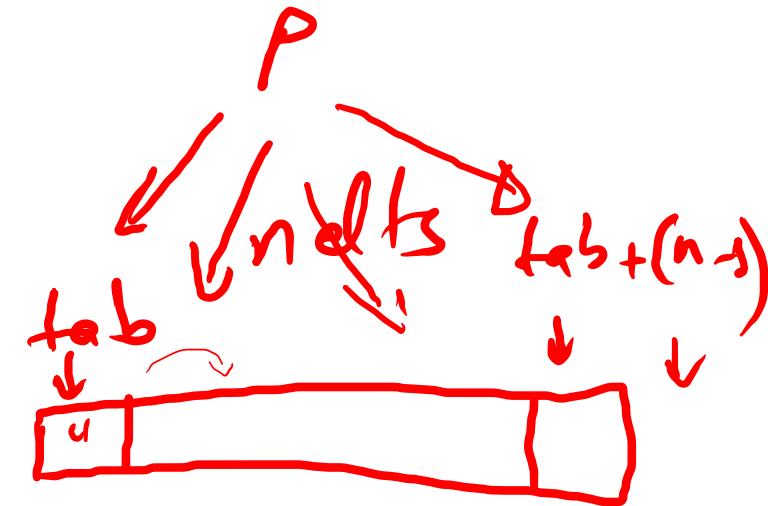
```
int nbOccurrence (int v, int* tab, int n); // signature de la
                                              // fonction .  
void main() {  
    int T[] = {0, 1, 2, 3, 4, 20, 22}; //  
    int taille = 7;  
    int x, nb = 0;  
    printf(" Saisir l'entier recherché : ");  
    scanf("%d", &x);  
    nb = nbOccurrence(x, T, taille);  
    printf(" Le nombre d'occurrence de %d est %d.", x, nb);  
}
```

```

int nbOccurrence(int v, int *ptab, int n) {
    int nb = 0;
    int i;
    for(i=0; i<n; i++){
        if (ptab[i] == v) nb++;
    }
    return nb;
}

```

gadgets  
gadgets



```

// utilise de pointeur
int nbOccurrence(int v, int *tab, int n) {
    int nb = 0;
    int *p;
    for(p=tab; p < tab+n; p++){
        if (*p == v) nb++;
    }
    return nb;
}

```

## Example :

```
void main() {
```

```
    int *P;      0   1   2   3   4  
    int T[] = {1, -1, 1, 3, 4};
```

```
    int i;
```

```
    for (i = 0; i < 5; i++) { // indices
```

```
        printf(" l'elt d'indice %d : %d\n", i, T[i]);
```

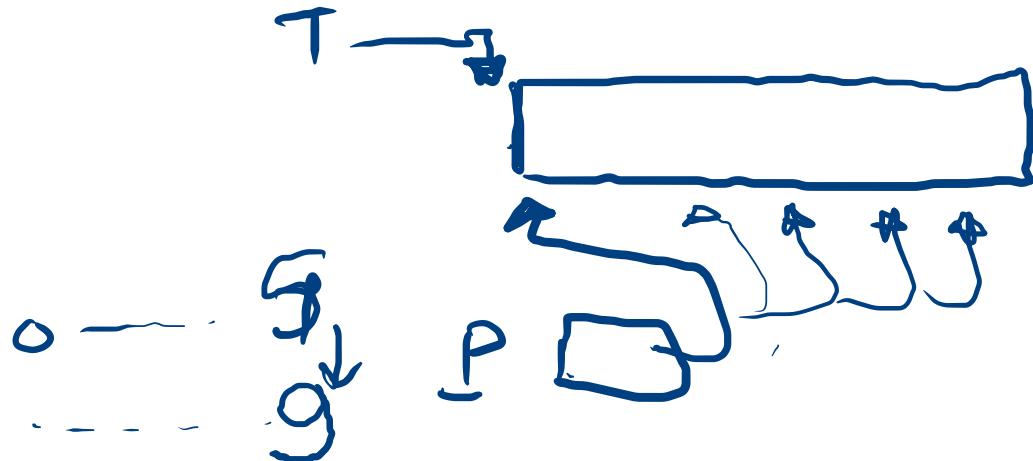
```
    } printf(" son adresse est %p ", (T + i));  
    printf(" & T[%d] <--> ", i);
```

```
    printf(" affichage par pointeur");
```

```
    for (P = T; P < T + 5; P++) { // address
```

```
        printf(" l'elt d'indice %d : %d \n", (P - T), *P);
```

```
    } printf(" son adresse est %p ", P);
```



## Création de procédures et fonctions

### ③ possibilités

① — monfichier.c —

// les fonctions avec leurs corps

f<sub>1</sub>( ) ;

{ f<sub>2</sub>( ) ; }

// prog principale

void main( ) ;

    // appel aux f<sup>n</sup>t et proc.

    { f<sub>1</sub>( ); f<sub>2</sub>( ); }

② " chemin "

" biblioth."  
#include < stdlib.h >

// biblioth contenant les  
les fonctions /procédures avec  
leurs corps

void main( ) {

    // appel aux f<sup>n</sup>t et proc  
    f<sub>1</sub>( ); f<sub>2</sub>( );

③ — monfichier.c —

// Déclaration des f<sup>n</sup>s et proc

type f<sub>1</sub>( - - ) ;

type f<sub>2</sub>( - - ) ;

// prog principal

void main( ) {

    // appel aux f<sup>n</sup>s

} f<sub>1</sub>( ); f<sub>2</sub>( );

} // Définir le corps de f<sub>1</sub>

type f<sub>1</sub>( ) ; ... ;

f<sub>2</sub>( ) ; ... ;

# 1 Exercice 1

void

Ecrire les algorithmes permettant :

1. Le calcul du nombre d'occurrences d'un élément donné dans un tableau.
2. Le calcul de la moyenne et du minimum des éléments d'un tableau.
3. De tester si un tableau est trié.
4. Le calcul du produit scalaire de deux vecteurs réels  $u$  et  $v$  de dimension  $n$  :  $u.v = \sum_{i=1}^{i=n} u_i * v_i$

Traduire les algorithmes précédents en langage C.

```
void minEtMoy(int & tab, int n, int & min, float & moy);  
void main() {  
    int T[] = {0, 1, 2, 3, 4, 20, 22}; //  
    int taille = 7; int min; float moy;  
    int x, nb = 0;  
    printf(" Saisir l'entier recherché : ");  
    scanf("%d", &x);  
    nb = nbOccurrence(x, T, taille);  
    printf(" Le nombre d'occurrence de %d est %d\n", x, nb);  
    minEtMoy(T, taille, &min, &moy);  
    printf(" Le min est %d, la moyenne est %f\n", min, moy);  
}
```

# 1 Exercice 1

Ecrire les algorithmes permettant :

1. Le calcul du nombre d'occurrences d'un élément donné dans un tableau.
2. Le calcul de la moyenne et du minimum des éléments d'un tableau.
3. De tester si un tableau est trié.
4. Le calcul du produit scalaire de deux vecteurs réels  $u$  et  $v$  de dimension  $n$  :  $u.v = \sum_{i=1}^n u_i * v_i$

Traduire les algorithmes précédents en langage C.

taille du tab  $\leftarrow$   $i$   $\rightarrow n - 1$   
 $\text{tab} \rightarrow \text{tab} + 0$

```
void minEtMoy(int *tab, int n, int *min, float *moy) {
    int *p;
    *min = *tab; // le contenu de min est le contenu du 1er élé
    *moy = 0.0;
    for (p = tab; p < tab + n; p++) {
        // somme des élts
        *moy = *moy + *p;
        if (*min > *p)
            *min = *p;
    }
    *moy = *moy / n;
}
```

for ( $i = 0; i < n; i++$ ) {  
     $*moy = *moy + T[i];$  ←  
     $\Leftrightarrow *moy = *moy + *(&T[i])$   
     $\Leftrightarrow *moy = *moy + *(\overset{P}{T} + i)$

# 1 Exercice 1

Ecrire les algorithmes permettant :

1. Le calcul du nombre d'occurrences d'un élément donné dans un tableau.
2. Le calcul de la moyenne et du minimum des éléments d'un tableau.
3. De tester si un tableau est trié. *par ordre croissant*
4. Le calcul du produit scalaire de deux vecteurs réels  $u$  et  $v$  de dimension  $n$ :  $u \cdot v = \sum_{i=1}^{i=n} u_i * v_i$

Traduire les algorithmes précédents en langage C.

Algorithm : TestTrié

$T[0 \dots 100]$ ;  $i$ : entier;  $\text{trié}$ : booléen;  $\text{taille}$ : entier

debut

    taille  $\leftarrow 100$ ;

    pour  $i \leftarrow 0$  à  $\text{taille} - 2$

        si  $T[i] \leq T[i + 1]$

$i \leftarrow i + 1$ ;

        sinon

            vrai = faux

            FinPour  $\leftarrow 11 \leftarrow i + 1$

        fin si

    fin Pour

    suite

$\text{trié} \leftarrow \text{vrai}$

$i = 0$

$i = 1$

$i = 2$

$\text{vrai}$

$\text{vrai}$

$\text{vrai}$

$i = 1$

$i = 2$

$i = 3$

    si ( $\text{trié} = \text{vrai}$ ) alors

        racine (" le tableau est trié") ,

        sinon

        racine (" le tableau n'est pas trié");

    fin si

taille

$i$

$\text{trié} \leftarrow \text{vrai}$   
—  $\leftarrow \text{faux}$

$T_1$  trié

$T_2$  non trié



$T_1$

$T_2$



# 1 Exercice 1

Ecrire les algorithmes permettant :

1. Le calcul du nombre d'occurrences d'un élément donné dans un tableau.
2. Le calcul de la moyenne et du minimum des éléments d'un tableau.
3. De tester si un tableau est trié. *par ordre croissant*
4. Le calcul du produit scalaire de deux vecteurs réels  $u$  et  $v$  de dimension  $n$ :  $u.v = \sum_{i=1}^{i=n} u_i * v_i$

Traduire les algorithmes précédents en langage C.

Algorithme : TestTrié

T [0...100]; i: entier; trié: boolian; table: entier

debut

```
    table ← 100;
    pour i ← 0 à table - 2
        si T[i] > T[i + 1]
            trié ← faux;
        fin si
        i ← i + 1;
    fin pour
```

fin pour

suite

trié ← vrai

i = 0      i = 2      i = 2      i = 3

vrai      vrai      vrai

i = 1      i = 2      i = 3

3

1

```
si (trié = vrai) alors
    jecrive ("Le tableau est trié");
    sinon
        jecrive ("Le tableau n'est pas
                  trié");
    fin si
```

# 1 Exercice 1

Ecrire les algorithmes permettant :

1. Le calcul du nombre d'occurrences d'un élément donné dans un tableau.
2. Le calcul de la moyenne et du minimum des éléments d'un tableau.
3. De tester si un tableau est trié. *par ordre croissant* ←
4. Le calcul du produit scalaire de deux vecteurs réels  $u$  et  $v$  de dimension  $n$  :  $u \cdot v = \sum_{i=1}^{i=n} u_i * v_i$

Traduire les algorithmes précédents en langage C.

```
int estTrie ( int *Tab , int n ) {  
    int *P ;  
    int trie = 1 ;  
    for ( P = Tab ; P <= Tab + (n - 2) ; P++ ) {  
        if ( *P > *(P + 1) ) {  
            trie = 0 ;  
            break ;  
        }  
    }  
    return trie ;  
}
```

taille  
%

```
int estTrie ( int *Tab , int n ) {  
    int P ;  
    for ( P = Tab ; P <= Tab + (n - 2) ; P++ ) {  
        if ( *P > *(P + 1) ) {  
            return 0 ;  
        }  
    }  
    return 1 ;  
}
```

Diagramme d'un tableau de taille  $n$  avec  $n = 3$ . Les éléments sont 2, 0, 1. L'indexation commence à 0. Un pointeur  $P$  est placé sur l'élément 0. Un autre pointeur  $P+1$  est placé sur l'élément 1. Des flèches indiquent que  $P$  et  $P+1$  sont comparés pour vérifier l'ordre croissant.

1

```
for ( P = tab ;  
      P <= tab + (n - 2) && (tries == 1) ;  
      P++ ) {  
    if ( *P > *(P + 1) ) {  
        tries = 0 ;  
    }  
}
```

# 1 Exercice 1

taille

Ecrire les algorithmes permettant :

1. Le calcul du nombre d'occurrences d'un élément donné dans un tableau.
2. Le calcul de la moyenne et du minimum des éléments d'un tableau.
3. De tester si un tableau est trié. *par ordre croissant*
4. Le calcul du produit scalaire de deux vecteurs réels  $u$  et  $v$  de dimension  $n$  :  $u \cdot v = \sum_{i=1}^{i=n} u_i * v_i$

Traduire les algorithmes précédents en langage C.

```
float produit (float *u , float *v , int n){  
    /  
    /  
    /  
}  
}
```

fin

# 1 Exercice 1

Ecrire les algorithmes permettant :

1. Le calcul du nombre d'occurrences d'un élément donné dans un tableau.
2. Le calcul de la moyenne et du minimum des éléments d'un tableau.
3. De tester si un tableau est trié.
4. Le calcul du produit scalaire de deux vecteurs réels  $u$  et  $v$  de dimension  $n$  :  $u \cdot v = \sum_{i=1}^{i=n} u_i * v_i$

Traduire les algorithmes précédents en langage C.

Algorithm : produit

Si  $n$  : entier ;  $U[0..n-1]$  : réel ;  $V[0..n-1]$  : réel ;  $i$  : entier ;

Début  
 $n \leftarrow 5$  ;  $S \leftarrow 0$  ;  
 pour  $i \leftarrow 0$  à  $n-1$  faire  
 $S \leftarrow S + (U[i] * V[i])$  ;  
 $i \leftarrow i + 1$  ;  
 fin pour  
 Ecrire( $S$ ) ;  
 fin

$$\vec{v} = (v_1, v_2, v_3) \sim v[0..2]$$

$$\vec{u} = (u_1, u_2, u_3) \sim u[0..2]$$

$$\vec{u} \cdot \vec{v} = u_1 v_1 + u_2 v_2 + u_3 v_3$$

$$v[0..n-1] \quad \text{dimension } n \\ u[0..n-1]$$

$$\begin{array}{cccc} 0 & 1 & 2 & 3 \\ \downarrow & \downarrow & \downarrow & \downarrow \\ u & -1 & 0 & -2 & 0 \\ v & 1 & 1 & 0 & -1 \end{array}$$

$$\begin{aligned} i=0 & S = -1 \\ i=1 & S' = -1 + 0 = -1 \\ i=2 & S = -1 + -2 + 0 = -3 \\ i=3 & S = -1 + 0 = -1 \\ i=4 & S = -1 \\ \vec{u} \cdot \vec{v} & = \boxed{S = -1} \end{aligned}$$

1

-

|  
r.

in

# 1 Exercice 1

Ecrire les algorithmes permettant :

1. Le calcul du nombre d'occurrences d'un élément donné dans un tableau.
2. Le calcul de la moyenne et du minimum des éléments d'un tableau.
3. De tester si un tableau est trié.
4. Le calcul du produit scalaire de deux vecteurs réels  $u$  et  $v$  de dimension  $n$  :  $u \cdot v = \sum_{i=1}^{i=n} u_i * v_i$

Traduire les algorithmes précédents en langage C.

*Algortithme : produit*

*V[0..n-1] : réel ; i : entier ;  
Si réel ; n : entier ; U[0..n-1] : réel ;*

*Début*

*n ← 5 ; S ← 0 ;  
pour i ← 0 à n - 1 faire  
    S ← S + (U[i] \* V[i]) ;*

*i ← i + 1 ;*

*fin pour  
Ecrit(S) ;*

*fin*

1

-

1r

in

```
void main () {  
    float u[] = {-2, 0, 0, 5};  
    float v[] = {0, 2, 1, 1};  
    int i, n = 4;  
    float s = 0.0;  
    for (i = 0; i < n; i++) {  
        s = s + u[i] * v[i];  
        i++;  
    }  
    printf (" le produit  
    scalaire de u et v est : ");  
    printf ("%f", s);  
}
```

# 1 Exercice 1

Ecrire les algorithmes permettant :

1. Le calcul du nombre d'occurrences d'un élément donné dans un tableau.
2. Le calcul de la moyenne et du minimum des éléments d'un tableau.
3. De tester si un tableau est trié.
4. Le calcul du produit scalaire de deux vecteurs réels  $u$  et  $v$  de dimension  $n$  :  $u.v = \sum_{i=1}^{i=n} u_i * v_i$

Traduire les algorithmes précédents en langage C.

```
#include <stdio.h>
#define NB_MAX 100
void main() { int x; // elt recherche
    // Déclaration du tableau
    int T[100]; // 0 → 99
    // Déclaré un tab initialisé
    int T2[6] = {12, 0, 13, 15, 14, -1};;
    int i; // compteur pour parcourir T
    // Remplir le tableau T
    int taille = 10; // taille ≤ 100 ① ou ②
    int nb = 0; // nombre d'occurrences de x
    // lecture de taille depuis le clavier
```

100 entiers

    -----     // int T[NB\_Max];

    // forcer l'utilisateur à donner une taille ≤ 100 .

    do{

        printf("Entrer la taille du tableau (<100)");

        scanf("%d", &taille);

    } while( taille > 100 || taille < 0 );

    | fin

```

i <= taille -1
for (i = 0 ; i < taille ; i++) {
    printf (" T[%d]", i);
    scanf ("%d", &T[i]);
}
→ i <= taille
// Saisie de l'elt recherché
printf (" Donner l'entier recherché:"); x = 15)
scanf ("%d", &x);
// recherche
for (i = 0 ; i < taille ; i++)
    if (T[i] == x) nb++;
→ i = taille
// affichage du nombre d'occurrence
printf (" le nombre d'occurrence de %d est: %d", x, nb);

```

affichage du tab. T

```

// i = 0 ; printf (" [  

while (i < taille) {
    printf ("%d", T[i]);
    i++;
}
printf ("]");

```

} // fin main

suite

## 2 Exercice 2

Ecrire l'algorithme effectuant le décalage des éléments d'un tableau.

Exemple :

- Tableau initial 

|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
| D | E | C | A | L | A | G | E |
|---|---|---|---|---|---|---|---|

- Tableau modifié (décalage à gauche) 

|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
| E | C | A | L | A | G | E | D |
|---|---|---|---|---|---|---|---|

Traduire l'algorithme précédent en langage C.

Algorithme : Décalage

$T[0..100]$  de réel ;  $x$ : réel ;  $i$ : entier

$N$ : entier ; //  $N < 100$  nombre d'elts.

// lecture du nombre d'elts  $N$

repetez

{ écrire ("Donner le nombre d'elts:");

lire ( $N$ );

→ jusqu'à ce que ( $N < 100$ );

// lecture du tableau

Pour  $i \neq 0$  à  $N-1$  faire :

    écrire ("entrez l'elt d'indice:",  $i$ );

    lire ( $T[i]$ );

$x \leftarrow i+1$ ;

fin Pour

(de  $\rightarrow$  while ( $i < 100$ )  
    de /  
        *un tel* ( $i$ )  
        // affichage avant  
        // décalage

$x \leftarrow T[0]$ ;

Pour  $i \neq 0$  à  $N-2$  faire :

$T[i] \leftarrow T[i+1]$ ;

fin Pour

$T[N-1] \leftarrow x$ ;

$T[0] \leftarrow x$ ;

    // affichage après décalage

fin

## 2 Exercice 2

Ecrire l'algorithme effectuant le décalage des éléments d'un tableau.

Exemple :

- Tableau initial 

|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
| D | E | C | A | L | A | G | E |
|---|---|---|---|---|---|---|---|
- Tableau modifié (décalage à gauche) 

|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
| E | C | A | L | A | G | E | D |
|---|---|---|---|---|---|---|---|

Traduire l'algorithme précédent en langage C.

```

void main () {
    char T[] = {'D', 'E', 'C', 'A', 'L', 'A', 'G', 'E'} x ← T[0];
    // char T[] = "DECALAGE";
    int N = 8;
    char x;
    // afficher avant décalage
    printf("avant: %s", T); // { for(i=0; i<N; i++)
    //                                { printf("%c", T[i]); } finPour
    // Décalage
    x = T[0];
    for(i=0; i < N-2; i++)
        T[i] = T[i+1];
    T[N-1] = x; // T[i+1] = x;
    printf("après décalage: %s", T);
}

```

(de u<wo  
while ( )  
de /  
{  
 // affichage avant  
 // décalage  
 x ← T[0];  
 Pour i=0 à N-2 faire:  
 T[i] ← T[i+1];  
 finPour  
 T[N-1] ← x;  
 // T[i+1] ← x;  
 // affichage après décalage  
 fin