Algorithmique
Cours 4

IUT Informatique de Lens, 1ère Année
Université d'Artois

Frédéric Koriche
koriche@cril.fr
2011 - Semestre 1

Sommaire

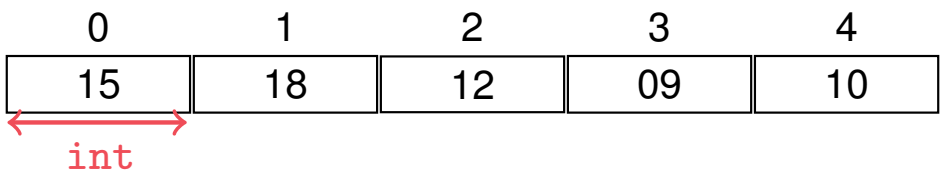
L'objectif de ce cours est d'étudier les **tableaux** en algorithmique et programmation C++.

Sommaire

L'objectif de ce cours est d'étudier les **tableaux** en algorithmique et programmation C++.

Tableaux

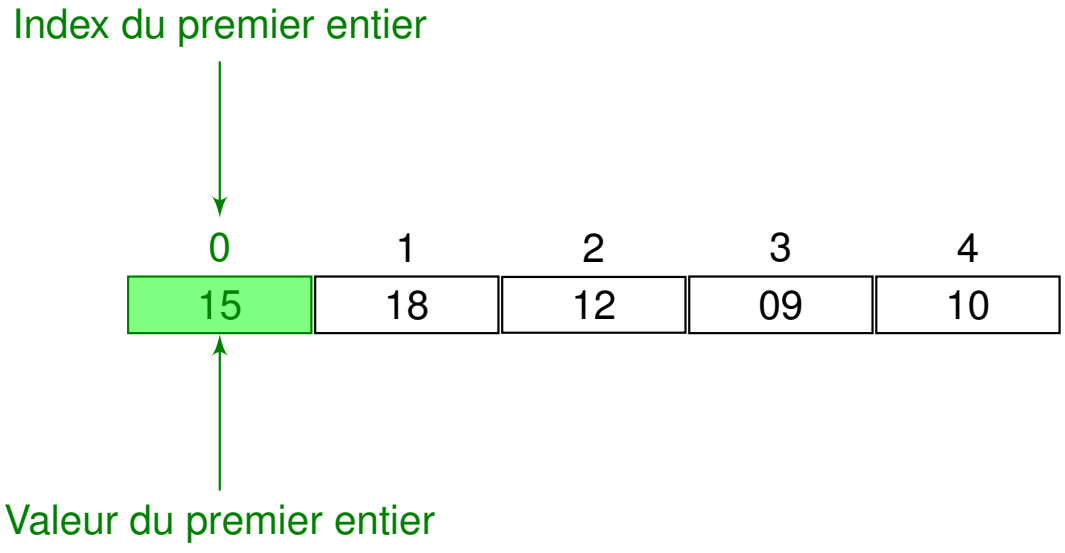
Un **tableau** (unidimensionnel) est une séquence de données du même type accessibles par leur index.



Un tableau de 5 entiers

Tableaux

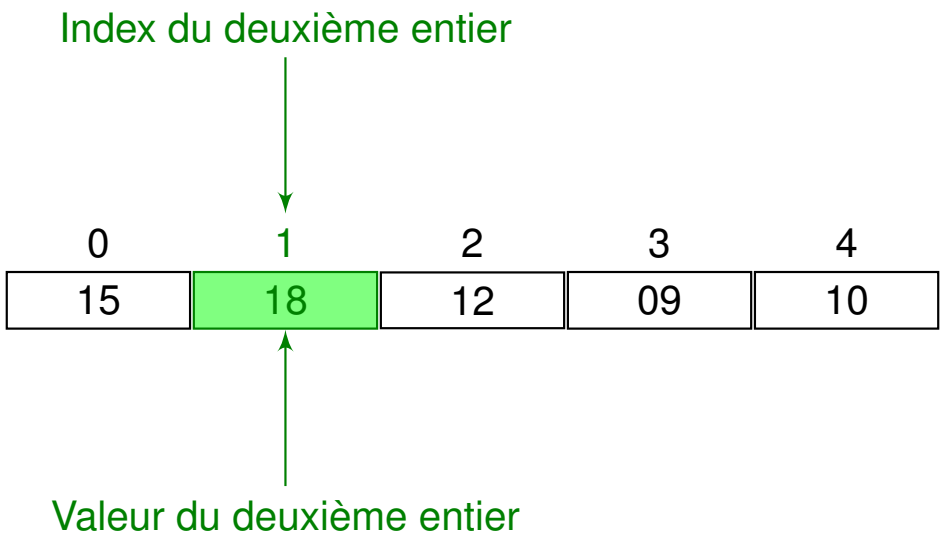
Un **tableau** (unidimensionnel) est une séquence de données du même type accessibles par leur index.



Un tableau de 5 entiers

Tableaux

Un **tableau** (unidimensionnel) est une séquence de données du même type accessibles par leur index.



Un tableau de 5 entiers

Tableaux Statiques

Un tableau est **statique** si sa taille est fixe : elle est connue à l'avance et ne bouge pas pendant toute l'exécution du programme.

Tableaux Statiques

Un tableau est **statique** si sa taille est fixe : elle est connue à l'avance et ne bouge pas pendant toute l'exécution du programme.

Tableaux Dynamiques

Un tableau est **dynamique** si sa taille peut évoluer : elle n'est pas forcément connue à l'avance et peut changer en cours d'exécution du programme.

Tableaux Statiques

Un tableau est **statique** si sa taille est fixe : elle est connue à l'avance et ne bouge pas pendant toute l'exécution du programme.

Tableaux Dynamiques

Un tableau est **dynamique** si sa taille peut évoluer : elle n'est pas forcément connue à l'avance et peut changer en cours d'exécution du programme.

Note

Dans ce cours nous étudierons les tableaux statiques. En C++, les tableaux dynamiques peuvent être construits à partir de pointeurs ou en utilisant la classe `vector`.

Déclaration d'un tableau statique

Un tableau statique est défini par

- son **nom**
- le **type** de ses éléments
- sa **taille** ou le nombre de ses éléments

Déclaration d'un tableau statique

Un tableau statique est défini par

- son **nom**
- le **type** de ses éléments
- sa **taille** ou le nombre de ses éléments

Déclaration en pseudo-code

Un début d'algorithme

```
variables
    entier notes[5]
début
    ...
fin
```

Nombre d'éléments du tableau

Nom du tableau

Type du tableau

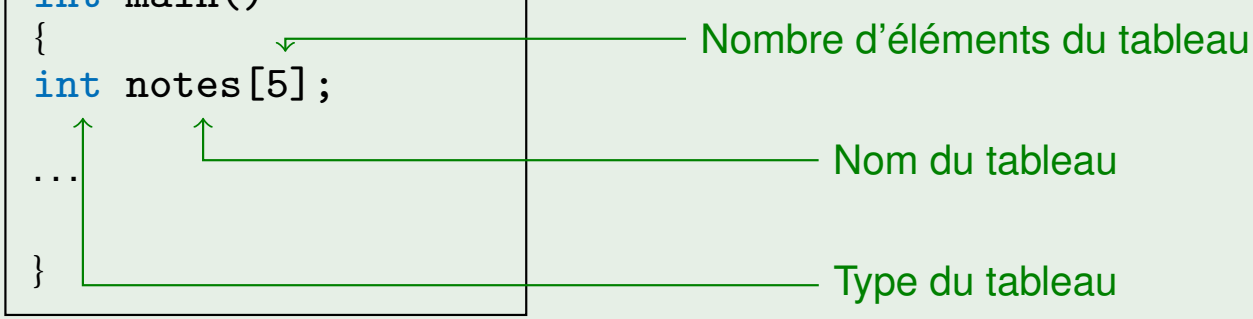
Déclaration d'un tableau statique

Un tableau statique est défini par

- son **nom**
- le **type** de ses éléments
- sa **taille** ou le nombre de ses éléments

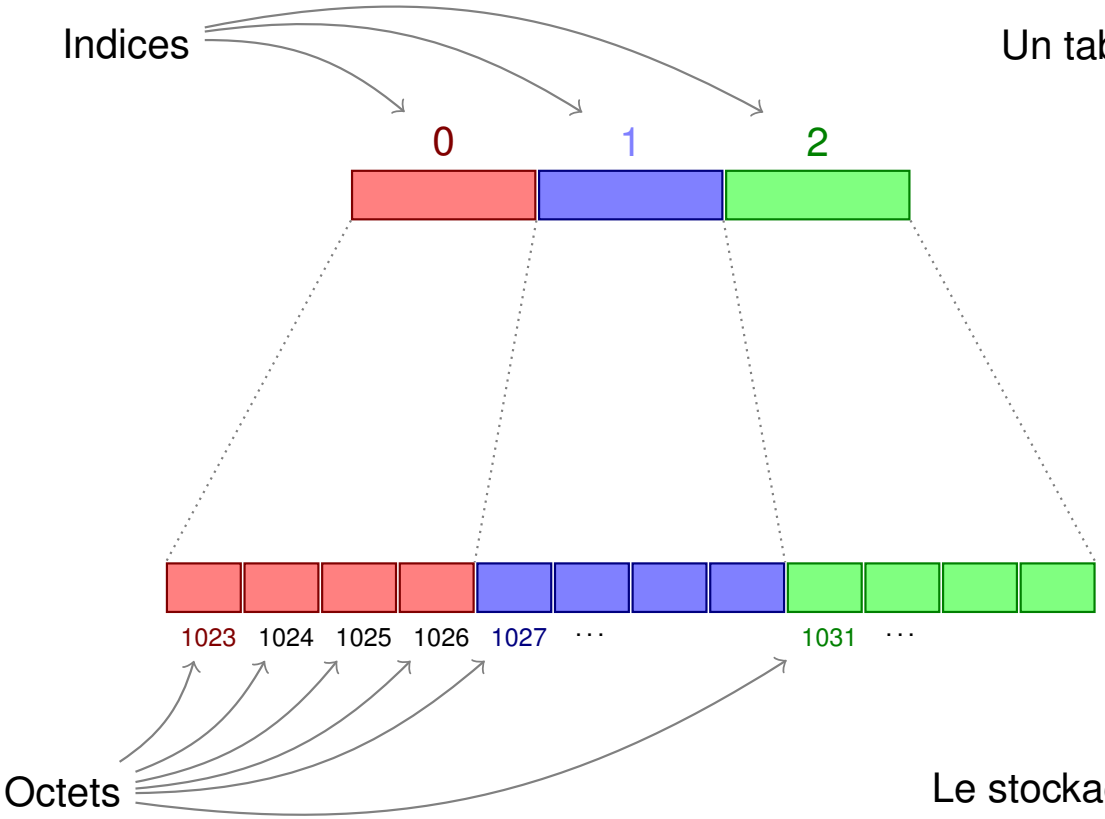
Déclaration en C++

```
#include <iostream>
using namespace std;

int main()
{
int notes[5];

...

}
```



Représentation en mémoire

Les tableaux statiques sont stockés en utilisant des zones contiguës de mémoire.



Un tableau d'entiers en C++

Le stockage du tableau en mémoire

Accès aux éléments d'un tableau

Chaque élément d'un tableau peut être traité comme une **variable**. En utilisant l'index de l'élément du tableau, il est possible en **une seule instruction** d'accéder à la valeur de cet élément.

Accès aux éléments d'un tableau

Chaque élément d'un tableau peut être traité comme une **variable**. En utilisant l'index de l'élément du tableau, il est possible en **une seule instruction** d'accéder à la valeur de cet élément.

Affectation en pseudo-code

Un algorithme

```
variables
    entier x
    entier notes[3]

début
    notes[0] ← 18 ⟵ Le 1er élément prend la valeur 18
    notes[2] ← 12
    x ← notes[0]
fin
```

Accès aux éléments d'un tableau

Chaque élément d'un tableau peut être traité comme une **variable**. En utilisant l'index de l'élément du tableau, il est possible en **une seule instruction** d'accéder à la valeur de cet élément.

Affectation en pseudo-code

Un algorithme

```
variables
    entier x
    entier notes[3]

début
    notes[0] ← 18 ⟵ Le 1er élément prend la valeur 18
    notes[2] ← 12 ⟵ Le 3ème élément prend la valeur 12
    x ← notes[0]
fin
```

Accès aux éléments d'un tableau

Chaque élément d'un tableau peut être traité comme une **variable**. En utilisant l'index de l'élément du tableau, il est possible en **une seule instruction** d'accéder à la valeur de cet élément.

Affectation en pseudo-code

Un algorithme

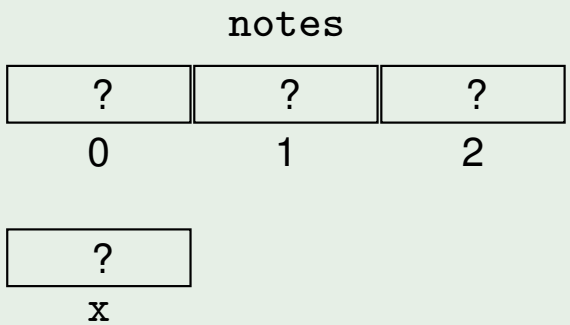
```
variables
    entier x
    entier notes[3]

début
    notes[0] ← 18      ⟵ Le 1er élément prend la valeur 18
    notes[2] ← 12      ⟵ Le 3ème élément prend la valeur 12
    x ← notes[0]       ⟵ La variable x prend la valeur du 1er élément
fin
```

Accès aux éléments d'un tableau

Chaque élément d'un tableau peut être traité comme une **variable**. En utilisant l'index de l'élément du tableau, il est possible en **une seule instruction** d'accéder à la valeur de cet élément.

Accès en C++

```
#include <iostream>
using namespace std;

int main()
{
int notes[3];
int x;


notes[0] = 18;
notes[2] = 12;
x = notes[0];


return 0;
}
```



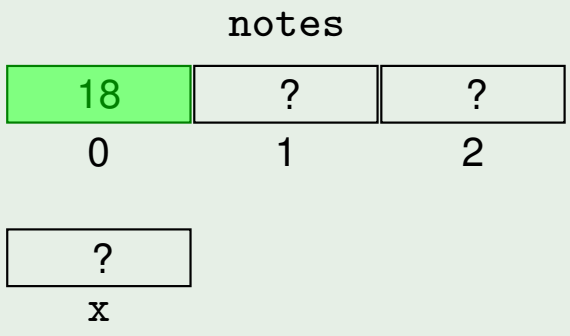
Déclaration des variables ; les valeurs ne sont pas connues

Accès aux éléments d'un tableau

Chaque élément d'un tableau peut être traité comme une **variable**. En utilisant l'index de l'élément du tableau, il est possible en **une seule instruction** d'accéder à la valeur de cet élément.

Accès en C++

```
#include <iostream>
using namespace std;

int main()
{
int notes[3];
int x;


notes[0] = 18;
notes[2] = 12;
x = notes[0];


return 0;
}
```



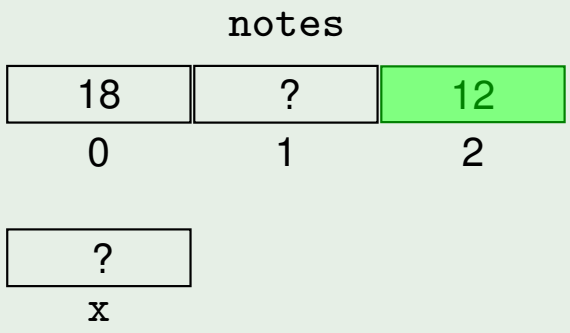
Affectation de la valeur 18 à la case 0

Accès aux éléments d'un tableau

Chaque élément d'un tableau peut être traité comme une **variable**. En utilisant l'index de l'élément du tableau, il est possible en **une seule instruction** d'accéder à la valeur de cet élément.

Accès en C++

```
#include <iostream>
using namespace std;

int main()
{
int notes[3];
int x;


notes[0] = 18;
notes[2] = 12;
x = notes[0];


return 0;
}
```



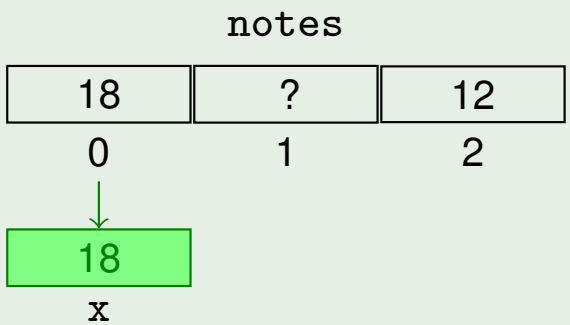
Affectation de la valeur 12 à la case 2

Accès aux éléments d'un tableau

Chaque élément d'un tableau peut être traité comme une **variable**. En utilisant l'index de l'élément du tableau, il est possible en **une seule instruction** d'accéder à la valeur de cet élément.

Accès en C++

```
#include <iostream>
using namespace std;

int main()
{
int notes[3];
int x;


notes[0] = 18;
notes[2] = 12;
x = notes[0];


return 0;
}
```



Affectation de la valeur de la case 0 à la variable `x`

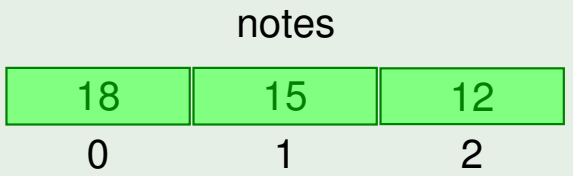
Initialisation d'un tableau

Il est possible d'initialiser les valeurs des éléments d'un tableau au moment de sa déclaration

Initialisation en pseudo-code

Un début d'algorithme

```
variables
    entier notes[3] ← {18, 15, 12}

début
    ...
fin
```



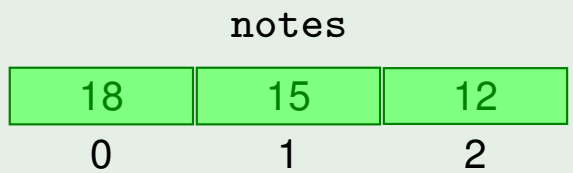
Initialisation d'un tableau

Initialisation d'un tableau

Il est possible d'initialiser les valeurs des éléments d'un tableau au moment de sa déclaration

Initialisation en C++

```
#include <iostream>
using namespace std;

int main()
{
int notes[3] = {18,15,12};

...

}
```



Initialisation d'un tableau

Validité des accès

Pour un tableau de taille n, un indice est valide s'il est dans l'intervalle $\{0, \cdots, n-1\}$.
Lors d'une affectation, le compilateur C++ ne vérifie pas si l'indice est valide : ce travail revient au programmeur.

Erreurs d'exécution

```
#include <iostream>
using namespace std;

int main()
{
int notes[3];
int x;


notes[3] = 10;

x = notes[0];

x = notes[3];


}
```

Validité des accès

Pour un tableau de taille n, un indice est valide s'il est dans l'intervalle $\{0, \cdots, n-1\}$.
Lors d'une affectation, le compilateur C++ ne vérifie pas si l'indice est valide : ce travail revient au programmeur.

Erreurs d'exécution

```
#include <iostream>
using namespace std;

int main()
{
int notes[3];
int x;


notes[3] = 10;

x = notes[0];

x = notes[3];


}
```

L'affectation à `notes[3]` est invalide, puisque l'index 3 est en dehors de l'intervalle autorisé !

Validité des accès

Pour un tableau de taille n, un indice est valide s'il est dans l'intervalle $\{0, \cdots, n-1\}$.
Lors d'une affectation, le compilateur C++ ne vérifie pas si l'indice est valide : ce travail revient au programmeur.

Erreurs d'exécution

```
#include <iostream>
using namespace std;

int main()
{
int notes[3];
int x;


notes[3] = 10;

x = notes[0];

x = notes[3];


}
```

L'affectation à `notes[3]` est invalide, puisque l'index 3 est en dehors de l'intervalle autorisé !

L'affectation est valide, mais comme le tableau n'est pas initialisé, `x` peut prendre n'importe quelle valeur

Validité des accès

Pour un tableau de taille n, un indice est valide s'il est dans l'intervalle $\{0, \cdots, n-1\}$.
Lors d'une affectation, le compilateur C++ ne vérifie pas si l'indice est valide : ce travail revient au programmeur.

Erreurs d'exécution

```
#include <iostream>
using namespace std;

int main()
{
int notes[3];
int x;


notes[3] = 10;

x = notes[0];

x = notes[3];


}
```

L'affectation à `notes[3]` est invalide, puisque l'index 3 est en dehors de l'intervalle autorisé !

L'affectation est valide, mais comme le tableau n'est pas initialisé, `x` peut prendre n'importe quelle valeur

Comme `notes[3]` n'est pas un élément du tableau, l'affectation à `x` est invalide !

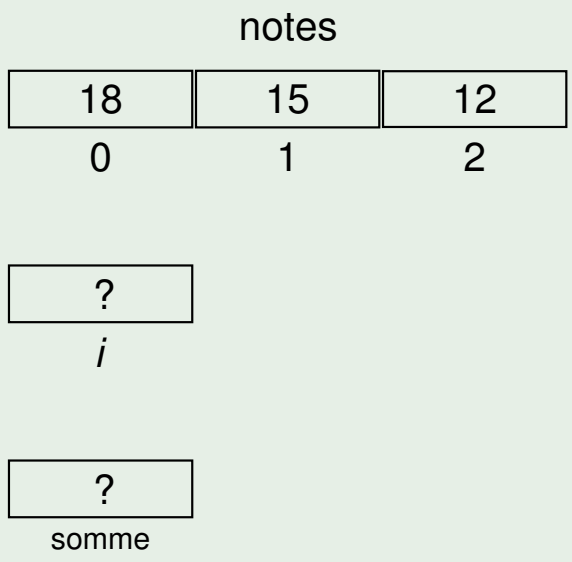
Parcours d'un tableau

Le parcours d'un tableau s'effectue en initialisant le compteur au premier index du tableau et en incrémentant le compteur jusqu'au dernier index du tableau

Parcours en pseudo-code

La moyenne des notes

```
variables
    entier notes[3] ← {18, 15, 12}
    entier i
    réel somme

début
    somme ← 0
    pour i ← 0 à 2 faire
        somme ← somme + notes[i]
    fin
    afficher somme / 3.0
fin
```



Déclaration des variables

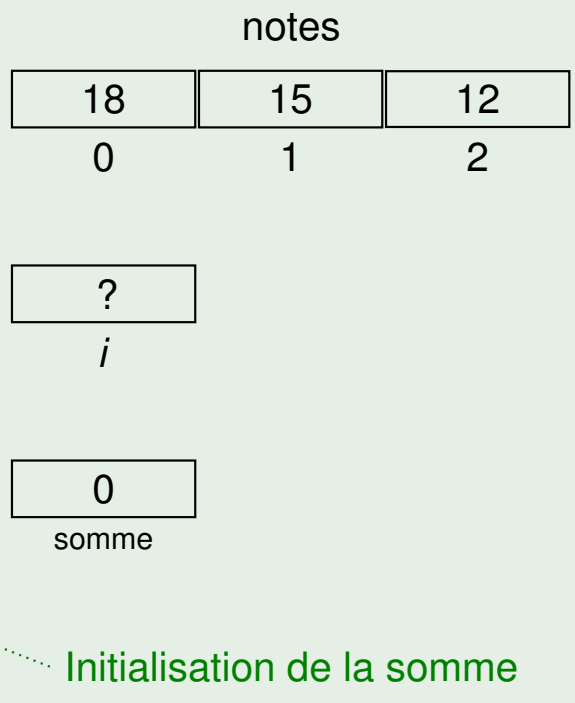
Parcours d'un tableau

Le parcours d'un tableau s'effectue en initialisant le compteur au premier index du tableau et en incrémentant le compteur jusqu'au dernier index du tableau

Parcours en pseudo-code

La moyenne des notes

```
variables
    entier notes[3] ← {18, 15, 12}
    entier i
    réel somme

début
    somme ← 0
    pour i ← 0 à 2 faire
        somme ← somme + notes[i]
    fin
    afficher somme / 3.0
fin
```



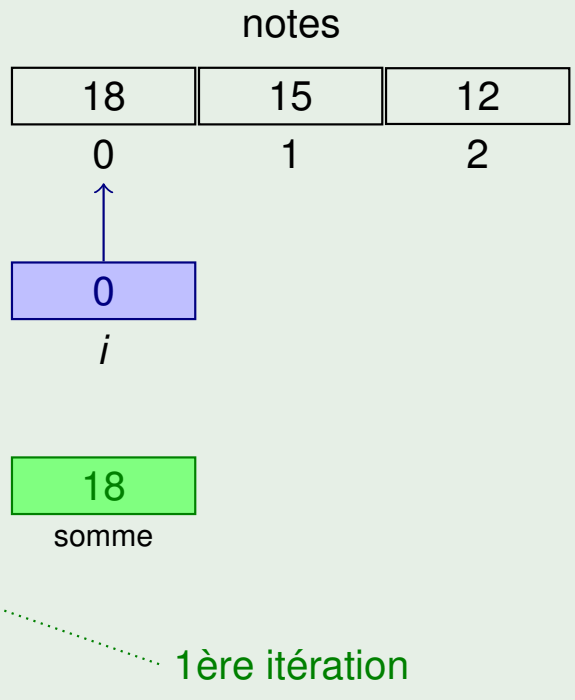
Parcours d'un tableau

Le parcours d'un tableau s'effectue en initialisant le compteur au premier index du tableau et en incrémentant le compteur jusqu'au dernier index du tableau

Parcours en pseudo-code

La moyenne des notes

```
variables
    entier notes[3] ← {18, 15, 12}
    entier i
    réel somme

début
    somme ← 0
    pour i ← 0 à 2 faire
        somme ← somme + notes[i]
    fin
    afficher somme / 3.0
fin
```



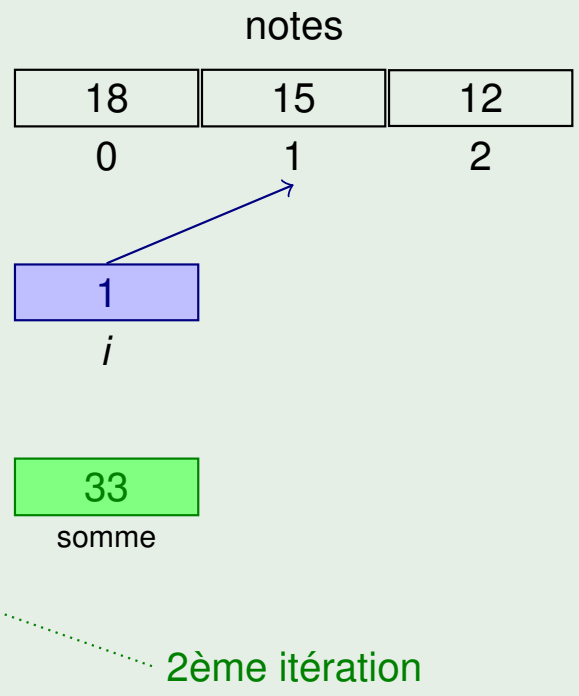
Parcours d'un tableau

Le parcours d'un tableau s'effectue en initialisant le compteur au premier index du tableau et en incrémentant le compteur jusqu'au dernier index du tableau

Parcours en pseudo-code

La moyenne des notes

```
variables
    entier notes[3] ← {18, 15, 12}
    entier i
    réel somme

début
    somme ← 0
    pour i ← 0 à 2 faire
        somme ← somme + notes[i]
    fin
    afficher somme / 3.0
fin
```



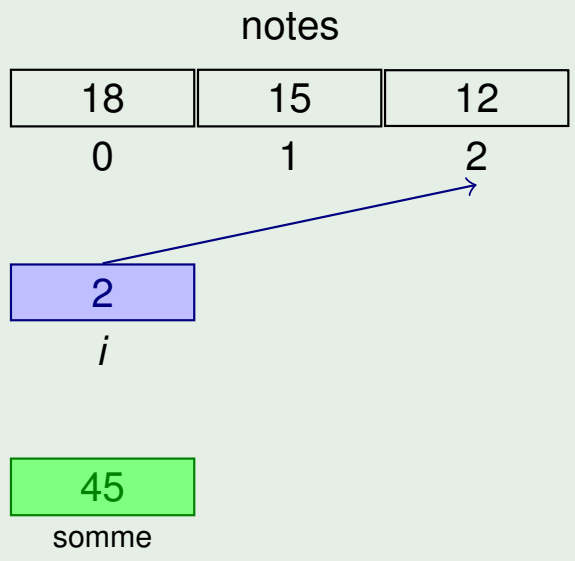
Parcours d'un tableau

Le parcours d'un tableau s'effectue en initialisant le compteur au premier index du tableau et en incrémentant le compteur jusqu'au dernier index du tableau

Parcours en pseudo-code

La moyenne des notes

```
variables
    entier notes[3] ← {18, 15, 12}
    entier i
    réel somme

début
    somme ← 0
    pour i ← 0 à 2 faire
        somme ← somme + notes[i]
    fin
    afficher somme / 3.0
fin
```



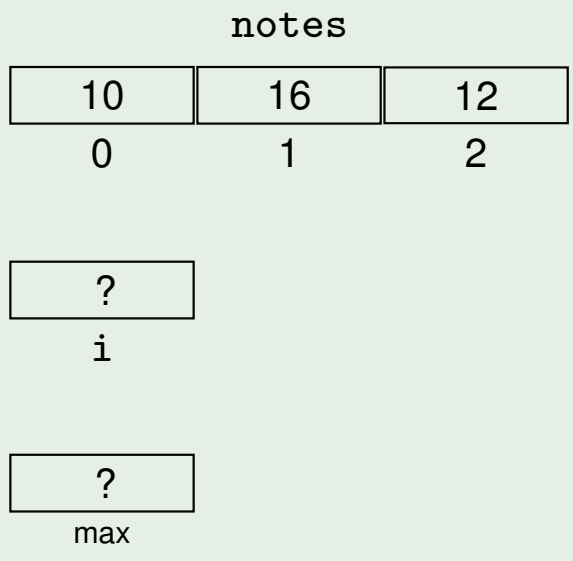
Parcours d'un tableau

Le parcours d'un tableau s'effectue en initialisant le compteur au premier index du tableau et en incrémentant le compteur jusqu'au dernier index du tableau

Parcours en C++

```
#include <iostream>
using namespace std;

int main()
{
int notes[3] = {10,16,12};
int i, max;


max = 0;
for (i=0; i<3; i++)
  {
  if (notes[i] > max)
    max = notes[i];
  }


cout << max;
return 0
}
```

Le maximum des notes



Déclaration des variables

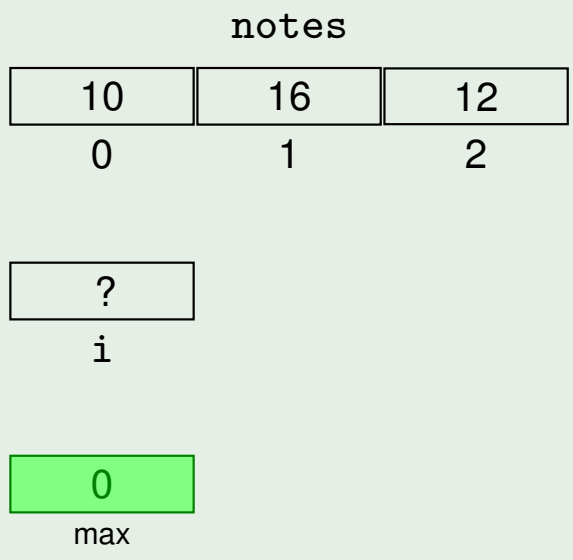
Parcours d'un tableau

Le parcours d'un tableau s'effectue en initialisant le compteur au premier index du tableau et en incrémentant le compteur jusqu'au dernier index du tableau

Parcours en C++

```
#include <iostream>
using namespace std;

int main()
{
int notes[3] = {10,16,12};
int i, max;


max = 0;
for (i=0; i<3; i++)
  {
  if (notes[i] > max)
    max = notes[i];
  }


cout << max;
return 0
}
```

Le maximum des notes

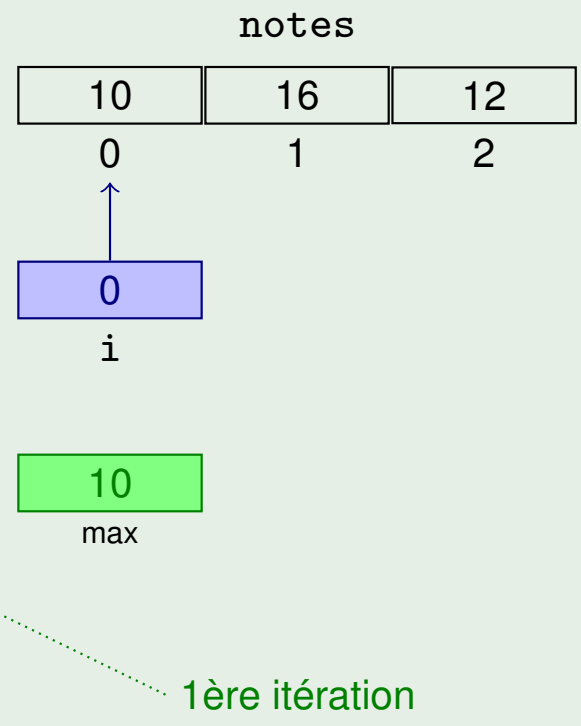


Parcours d'un tableau

Le parcours d'un tableau s'effectue en initialisant le compteur au premier index du tableau et en incrémentant le compteur jusqu'au dernier index du tableau

Parcours en C++

```
#include <iostream>
using namespace std;

int main()
{
int notes[3] = {10,16,12};
int i, max;


max = 0;
for (i=0; i<3; i++)
  {
  if (notes[i] > max)
    max = notes[i];
  }


cout << max;
return 0
}
```



Le maximum des notes

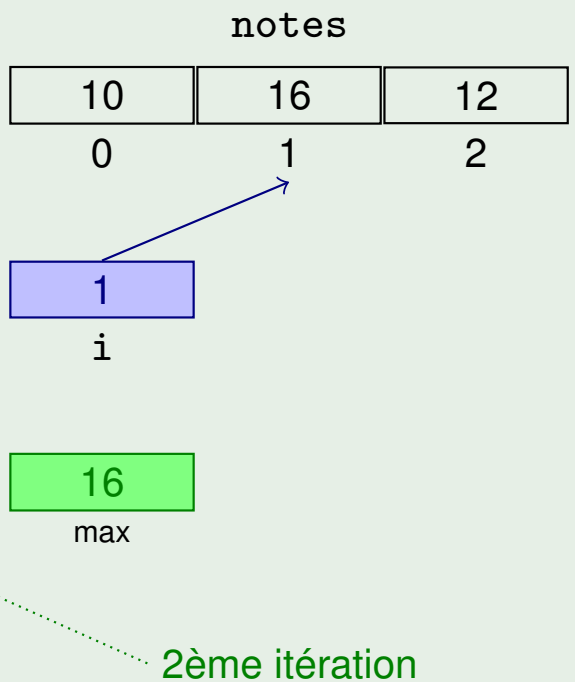
Parcours d'un tableau

Le parcours d'un tableau s'effectue en initialisant le compteur au premier index du tableau et en incrémentant le compteur jusqu'au dernier index du tableau

Parcours en C++

```
#include <iostream>
using namespace std;

int main()
{
int notes[3] = {10,16,12};
int i, max;


max = 0;
for (i=0; i<3; i++)
  {
  if (notes[i] > max)
    max = notes[i];
  }


cout << max;
return 0
}
```

Le maximum des notes

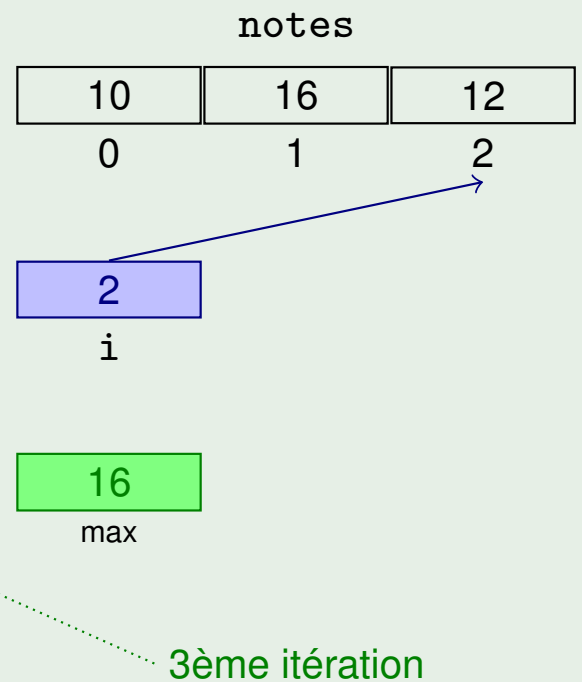


Parcours d'un tableau

Le parcours d'un tableau s'effectue en initialisant le compteur au premier index du tableau et en incrémentant le compteur jusqu'au dernier index du tableau

Parcours en C++

```
#include <iostream>
using namespace std;

int main()
{
int notes[3] = {10,16,12};
int i, max;


max = 0;
for (i=0; i<3; i++)
  {
  if (notes[i] > max)
    max = notes[i];
  }


cout << max;
return 0
}
```



Le maximum des notes

Validité des parcours

Le parcours total d'un tableau de taille n réalise n itérations depuis l'index 0 jusqu'à l'index $n-1$.

Erreurs de parcours

```
#include <iostream>
using namespace std;

int main()
{
int notes[5] = {10,12,08,14,16};
int i,somme;


somme = 0;
for (i=0; i<=5; i++)
  {
  somme += notes[i];
  }
cout << somme;


}
```

Validité des parcours

Le parcours total d'un tableau de taille n réalise n itérations depuis l'index 0 jusqu'à l'index $n-1$.

Erreurs de parcours

```
#include <iostream>
using namespace std;

int main()
{
int notes[5] = {10,12,08,14,16};
int i,somme;


somme = 0;
for (i=0; i<=5; i++)
  {
  somme += notes[i];
  }
cout << somme;


}
```

La condition `i <= 5` est invalide, puisque l'index 5 est en dehors de l'intervalle autorisé !

Validité des parcours

Le parcours total d'un tableau de taille n réalise n itérations depuis l'index 0 jusqu'à l'index $n-1$.

Erreurs de parcours

```
#include <iostream>
using namespace std;

int main()
{
int notes[5] = {10,12,08,14,16};
int i,somme;


somme = 0;
for (i=1; i<5; i++)
  {
  somme += notes[i];
  }
cout << somme;


}
```

Validité des parcours

Le parcours total d'un tableau de taille n réalise n itérations depuis l'index 0 jusqu'à l'index $n-1$.

Erreurs de parcours

```
#include <iostream>
using namespace std;

int main()
{
int notes[5] = {10,12,08,14,16};
int i,somme;


somme = 0;
for (i=1; i<5; i++)
  {
  somme += notes[i];
  }
cout << somme;


}
```

Même si l'instruction `for` est ici valide, la condition `i = 1` est incomplète pour faire la somme, puisque l'index 0 a été oublié !

Problème de la recherche

- Données : un tableau de n entiers naturels distincts, et une valeur v à trouver.
- Résultat : si le tableau contient un élément de valeur v, alors afficher l'index de cet élément ; sinon afficher "échec".

Recherche séquentielle

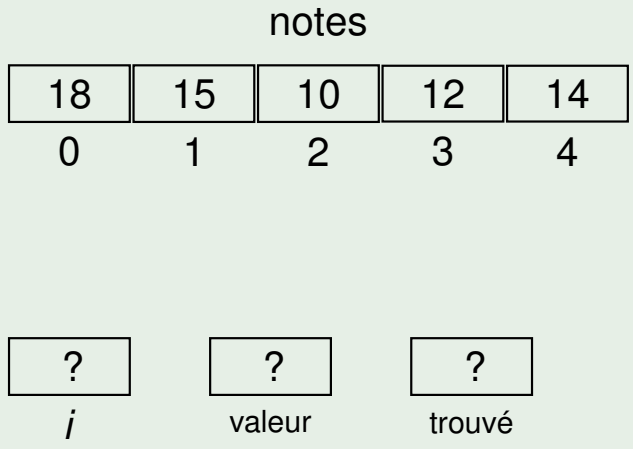
Rechercher dans un tableau non trié

```
variables
    entier notes[5] ← {18, 15, 10, 12, 14}
    entier i, valeur
    booléen trouvé

début
    lire valeur
    i ← 0
    répéter
        trouvé ← notes[i] = valeur
        i ← i + 1
    jusqu'à trouvé ou i = 5
    si trouvé alors
        afficher i − 1
    sinon
        afficher "échec"
    fin
fin
```

Recherche séquentielle

Rechercher dans un tableau non trié

```
variables
    entier notes[5] ← {18, 15, 10, 12, 14}
    entier i, valeur
    booléen trouvé

début
    lire valeur
    i ← 0
    répéter
        trouvé ← notes[i] = valeur
        i ← i + 1
    jusqu'à trouvé ou i = 5
    si trouvé alors
        afficher i − 1
    sinon
        afficher "échec"
    fin
fin
```

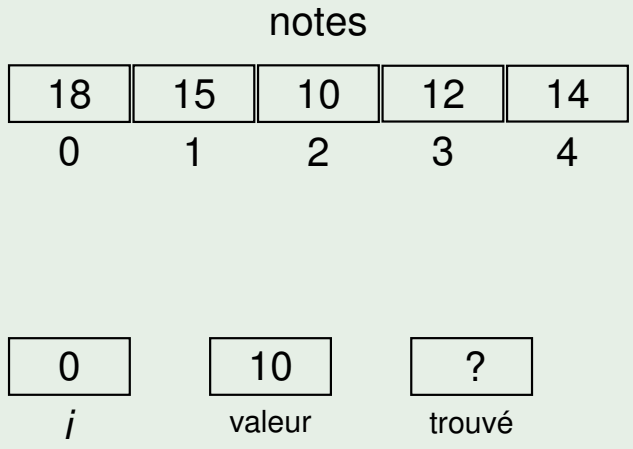


Déclaration des variables

Recherche séquentielle

Rechercher dans un tableau non trié

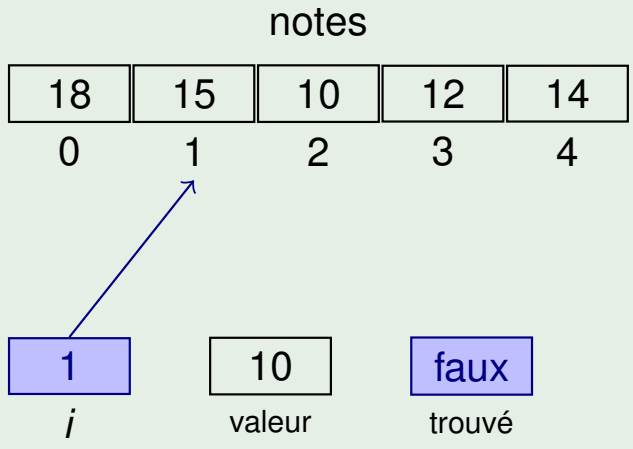
```
variables
    entier notes[5] ← {18, 15, 10, 12, 14}
    entier i, valeur
    booléen trouvé

début
    lire valeur
    i ← 0
    répéter
        trouvé ← notes[i] = valeur
        i ← i + 1
    jusqu'à trouvé ou i = 5
    si trouvé alors
        afficher i − 1
    sinon
        afficher "échec"
    fin
fin
```



Recherche séquentielle

Rechercher dans un tableau non trié

```
variables
    entier notes[5] ← {18,15,10,12,14}
    entier i, valeur
    booléen trouvé

début
    lire valeur
    i ← 0
    répéter
        trouvé ← notes[i] = valeur
        i ← i + 1
    jusqu'à trouvé ou i = 5
    si trouvé alors
        afficher i − 1
    sinon
        afficher "échec"
    fin
fin
```

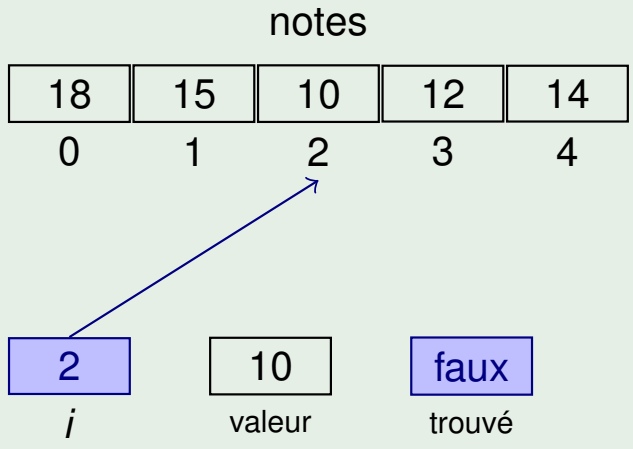


Fin de la 1ère itération

Recherche séquentielle

Rechercher dans un tableau non trié

```
variables
    entier notes[5] ← {18,15,10,12,14}
    entier i, valeur
    booléen trouvé

début
    lire valeur
    i ← 0
    répéter
        trouvé ← notes[i] = valeur
        i ← i + 1
    jusqu'à trouvé ou i = 5
    si trouvé alors
        afficher i − 1
    sinon
        afficher "échec"
    fin
fin
```

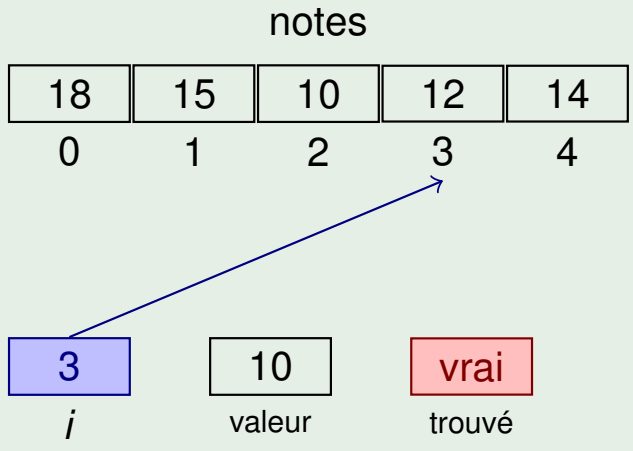


Fin de la 2ème itération

Recherche séquentielle

Rechercher dans un tableau non trié

```
variables
    entier notes[5] ← {18,15,10,12,14}
    entier i, valeur
    booléen trouvé

début
    lire valeur
    i ← 0
    répéter
        trouvé ← notes[i] = valeur
        i ← i + 1
    jusqu'à trouvé ou i = 5
    si trouvé alors
        afficher i − 1
    sinon
        afficher "échec"
    fin
fin
```

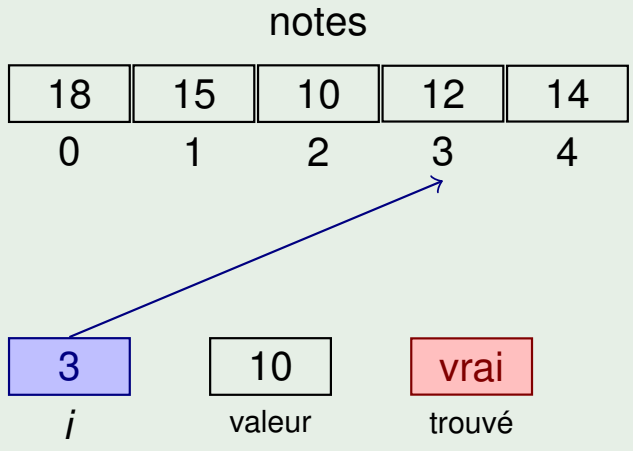


Fin de la 3ème itération

Recherche séquentielle

Rechercher dans un tableau non trié

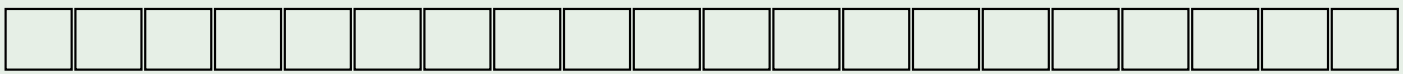
```
variables
    entier notes[5] ← {18, 15, 10, 12, 14}
    entier i, valeur
    booléen trouvé

début
    lire valeur
    i ← 0
    répéter
        trouvé ← notes[i] = valeur
        i ← i + 1
    jusqu'à trouvé ou i = 5
    si trouvé alors
        afficher i − 1
    sinon
        afficher "échec"
    fin
fin
```



$i < 5$ donc le programme retourne 2

Recherche dichotomique

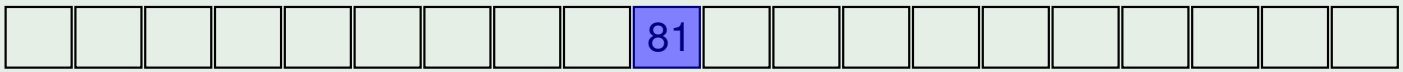
60



Supposons que l'on ait un grand tableau d'éléments **triés** du plus petit au plus grand. On cherche la valeur 60 dans le tableau.

Recherche dichotomique

60



On commence par regarder le milieu du tableau.

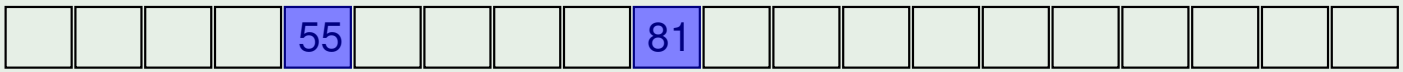
Recherche dichotomique

60

Comme la valeur recherchée est plus petite que 81 on relance la recherche dans la moitié gauche du tableau.

Recherche dichotomique

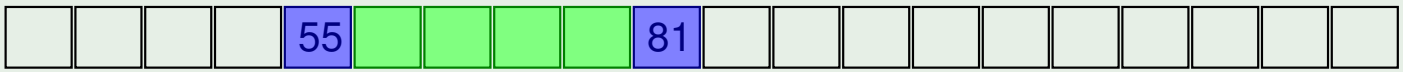
60



On regarde le milieu de cette portion.

Recherche dichotomique

60



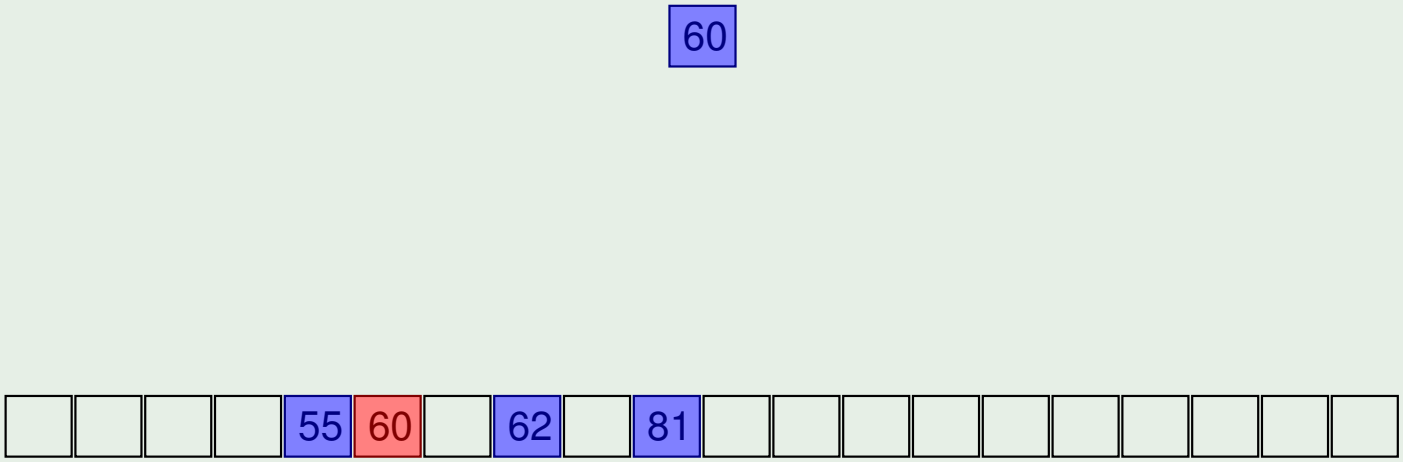
Comme la valeur recherchée est plus grande que 55 on relance la recherche dans la moitié droite de la portion.

Recherche dichotomique

60

Et ainsi de suite …

Recherche dichotomique



… jusqu'à tomber sur la valeur recherchée, ou jusqu'à obtenir une portion vide.

Recherche dichotomique

Rechercher dans un tableau trié

```
variables
    entier tableau[n] ← {···}
    entier i, gauche, droite, valeur

début
    lire valeur
    gauche ← 0
    droite ← n − 1
    répéter
        i ← (gauche + droite) / 2
        si valeur < tableau[i] alors
            droite ← i − 1
        sinon
            gauche ← i + 1
        fin
    jusqu'à (valeur = tableau[i]) ou (gauche > droite)
    si valeur = tableau[i] alors
        afficher i
    sinon
        afficher "échec"
    fin
fin
```

Recherche dichotomique

Rechercher dans un tableau trié

```
variables
    entier tableau[n] ← {···}
    entier i, gauche, droite, valeur

début
    lire valeur
    gauche ← 0
    droite ← n − 1
    répéter
        i ← (gauche + droite) / 2
        si valeur < tableau[i] alors
            droite ← i − 1
        sinon
            gauche ← i + 1
        fin
    jusqu'à (valeur = tableau[i]) ou (gauche > droite)
    si valeur = tableau[i] alors
        afficher i
    sinon
        afficher "échec"
    fin
fin
```

Bornes de la portion à rechercher

Recherche dichotomique

Rechercher dans un tableau trié

```
variables
    entier tableau[n] ← {···}
    entier i, gauche, droite, valeur

début
    lire valeur
    gauche ← 0
    droite ← n − 1
    répéter
        i ← (gauche + droite) / 2
        si valeur < tableau[i] alors
            droite ← i − 1
        sinon
            gauche ← i + 1
        fin
    jusqu'à (valeur = tableau[i]) ou (gauche > droite)
    si valeur = tableau[i] alors
        afficher i
    sinon
        afficher "échec"
    fin
fin
```

Bornes de la portion à rechercher

Milieu de la portion

Recherche dichotomique

Rechercher dans un tableau trié

```
variables
    entier tableau[n] ← {···}
    entier i, gauche, droite, valeur

début
    lire valeur
    gauche ← 0
    droite ← n − 1
    répéter
        i ← (gauche + droite) / 2
        si valeur < tableau[i] alors
            droite ← i − 1
        sinon
            gauche ← i + 1
        fin
    jusqu'à (valeur = tableau[i]) ou (gauche > droite)
    si valeur = tableau[i] alors
        afficher i
    sinon
        afficher "échec"
    fin
fin
```

Bornes de la portion à rechercher

Milieu de la portion

Si la valeur est plus petite que le milieu

Recherche dichotomique

Rechercher dans un tableau trié

```
variables
    entier tableau[n] ← {···}
    entier i, gauche, droite, valeur

début
    lire valeur
    gauche ← 0
    droite ← n − 1
    répéter
        i ← (gauche + droite) / 2
        si valeur < tableau[i] alors
            droite ← i − 1
        sinon
            gauche ← i + 1
        fin
    jusqu'à (valeur = tableau[i]) ou (gauche > droite)
    si valeur = tableau[i] alors
        afficher i
    sinon
        afficher "échec"
    fin
fin
```

Bornes de la portion à rechercher

Milieu de la portion

Si la valeur est plus petite que le milieu

alors rechercher dans la moitié gauche

Recherche dichotomique

Rechercher dans un tableau trié

```
variables
    entier tableau[n] ← {···}
    entier i, gauche, droite, valeur

début
    lire valeur
    gauche ← 0
    droite ← n − 1
    répéter
        i ← (gauche + droite) / 2
        si valeur < tableau[i] alors
            droite ← i − 1
        sinon
            gauche ← i + 1
        fin
    jusqu'à (valeur = tableau[i]) ou (gauche > droite)
    si valeur = tableau[i] alors
        afficher i
    sinon
        afficher "échec"
    fin
fin
```

Bornes de la portion à rechercher

Milieu de la portion

Si la valeur est plus petite que le milieu

alors rechercher dans la moitié gauche

sinon rechercher dans la moitié droite

Recherche dichotomique

Rechercher dans un tableau trié

```
variables
    entier tableau[n] ← {···}
    entier i, gauche, droite, valeur

début
    lire valeur
    gauche ← 0
    droite ← n − 1
    répéter
        i ← (gauche + droite) / 2
        si valeur < tableau[i] alors
            droite ← i − 1
        sinon
            gauche ← i + 1
        fin
    jusqu'à (valeur = tableau[i]) ou (gauche > droite)
    si valeur = tableau[i] alors
        afficher i
    sinon
        afficher "échec"
    fin
fin
```

Bornes de la portion à rechercher

Milieu de la portion

Si la valeur est plus petite que le milieu

alors rechercher dans la moitié gauche

sinon rechercher dans la moitié droite

continuer jusqu'à ce que la portion soit vide

Recherche dichotomique

Rechercher dans un tableau trié

```
variables
    entier tableau[n] ← {···}
    entier i, gauche, droite, valeur

début
    lire valeur
    gauche ← 0
    droite ← n − 1
    répéter
        i ← (gauche + droite) / 2
        si valeur < tableau[i] alors
            droite ← i − 1
        sinon
            gauche ← i + 1
        fin
    jusqu'à (valeur = tableau[i]) ou (gauche > droite)
    si valeur = tableau[i] alors
        afficher i
    sinon
        afficher "échec"
    fin
fin
```

Bornes de la portion à rechercher

Milieu de la portion

Si la valeur est plus petite que le milieu

alors rechercher dans la moitié gauche

sinon rechercher dans la moitié droite

continuer jusqu'à ce que la portion soit vide

ou jusqu'à trouver l'index de la valeur

Problème du tri

- Données : un tableau de n entiers naturels $\langle x_0, \cdots, x_{n-1} \rangle$
- Résultat : une permutation des éléments $\langle x'_0, \cdots, x'_{n-1} \rangle$ telle que $x'_0 \leq x'_1 \leq \cdots \leq x'_{n-1}$.

Tri par sélection

20	12	15	16	30	31	08	07	04	12	11	02	05	09	10	21	14	16	03	29

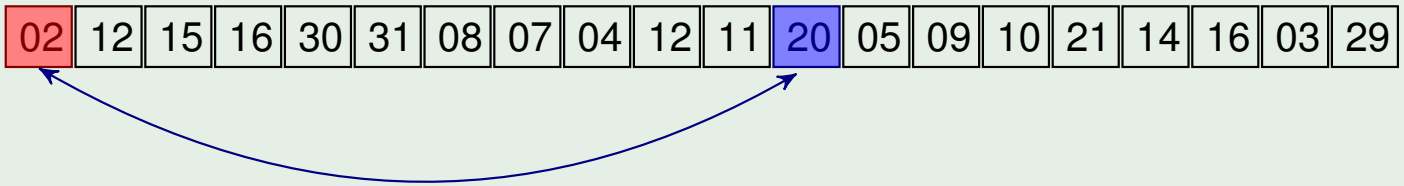
Supposons que l'on ait un tableau d'éléments non triés.

Tri par sélection

20	12	15	16	30	31	08	07	04	12	11	02	05	09	10	21	14	16	03	29

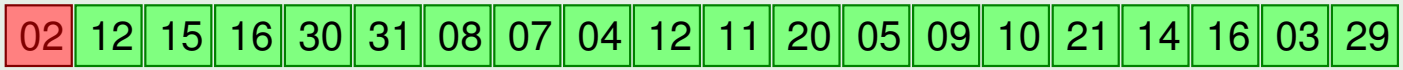
On commence par chercher le plus petit élément du tableau.

Tri par sélection



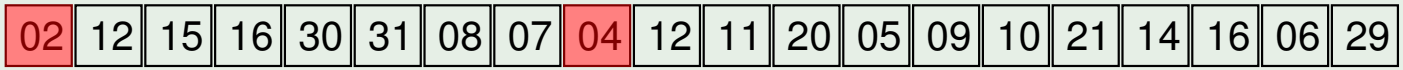
On le permute avec le premier élément du tableau, afin de le mettre en première position.

Tri par sélection



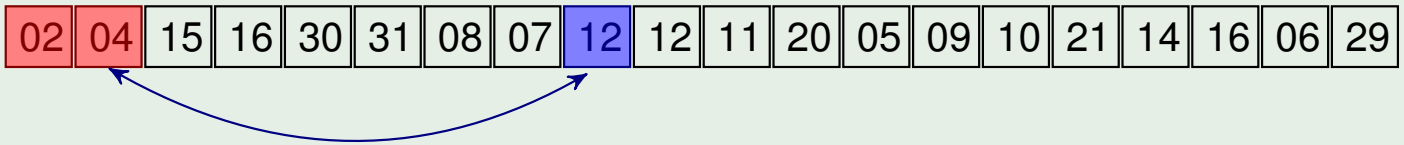
Comme le premier élément est bien rangé, on applique la même stratégie dans la portion du tableau commençant par le deuxième élément.

Tri par sélection



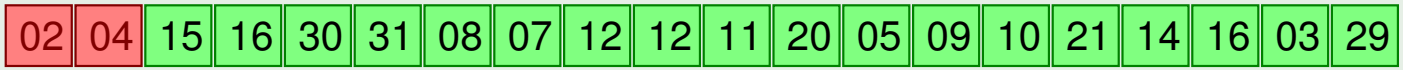
Donc on recommence par chercher le plus petit élément de cette portion,

Tri par sélection



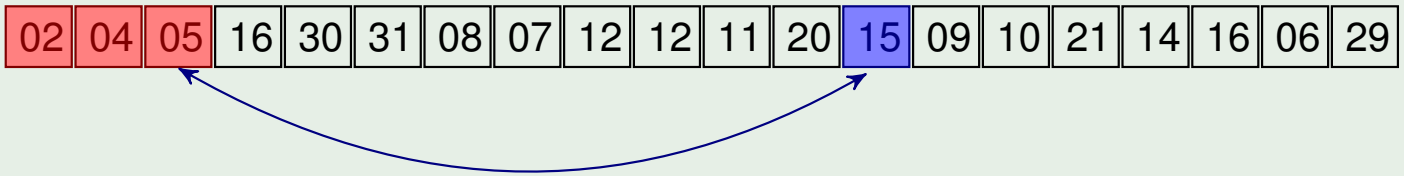
que l'on permute avec le deuxième élément.

Tri par sélection



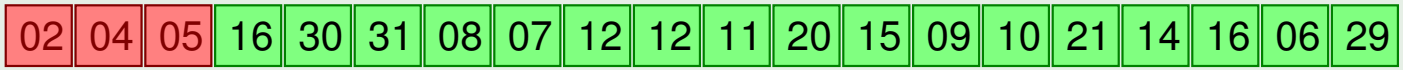
La même stratégie est appliquée sur la portion restante …

Tri par sélection



La même stratégie est appliquée sur la portion restante …

Tri par sélection



… et ainsi de suite jusqu'à atteindre la fin du tableau

Tri par sélection

Trier par selection

```
variables
    entier tableau[n] ← {···}
    entier i,j,t, min

début
    pour i ← 0 à n−1 faire
        min ← i
        pour j ← i+1 à n−1 faire
            si tableau[j] < tableau[min] alors
                min ← j
            fin
        fin
        t ← tableau[min]
        tableau[min] ← tableau[i]
        tableau[i] ← t
    fin
fin
```

Tri par sélection

Trier par selection

```
variables
    entier tableau[n] ← {···}
    entier i,j,t, min

début
    pour i ← 0 à n−1 faire
        min ← i
        pour j ← i+1 à n−1 faire
            si tableau[j] < tableau[min] alors
                min ← j
            fin
        fin
        t ← tableau[min]
        tableau[min] ← tableau[i]
        tableau[i] ← t
    fin
fin
```

Recherche l'index du plus petit élément de la portion courante

Tri par sélection

Trier par selection

```
variables
    entier tableau[n] ← {···}
    entier i,j,t, min

début
    pour i ← 0 à n − 1 faire
        min ← i
        pour j ← i + 1 à n − 1 faire
            si tableau[j] < tableau[min] alors
                min ← j
            fin
        fin
        t ← tableau[min]
        tableau[min] ← tableau[i]
        tableau[i] ← t
    fin
fin
```

Recherche l'index du plus petit élément de la portion courante

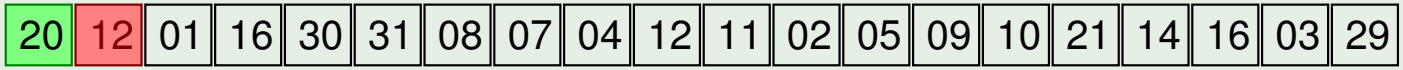
Permute le minimum avec l'élément le plus à gauche

Tri par insertion

20	12	01	16	30	31	08	07	04	12	11	02	05	09	10	21	14	16	03	29

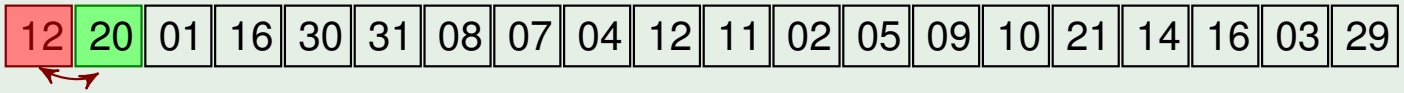
Le tri par insertion s'inspire de la manière dont on trie les cartes dans une main. Supposons que l'on ait un tableau d'éléments non triés.

Tri par insertion



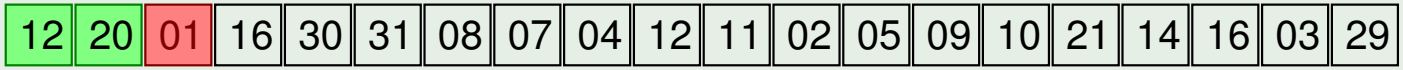
On commence par le deuxième élément du tableau.

Tri par insertion



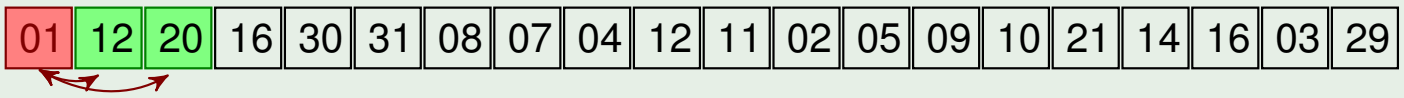
On l'insère à la bonne position sur la gauche en décalant les éléments plus grands.

Tri par insertion



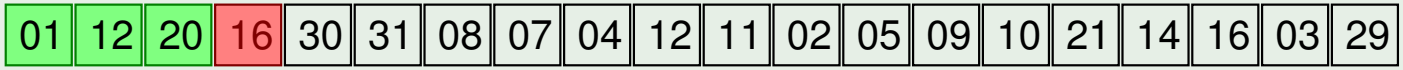
On applique la même stratégie au troisième élément du tableau,

Tri par insertion



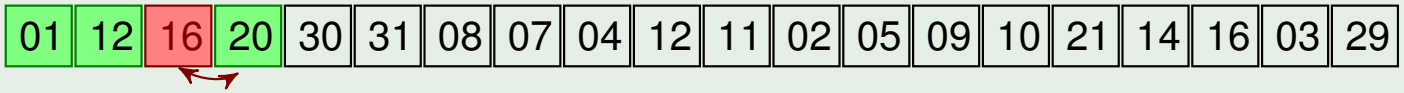
qu'on insère à la bonne position sur la gauche en décalant les éléments plus grands.

Tri par insertion



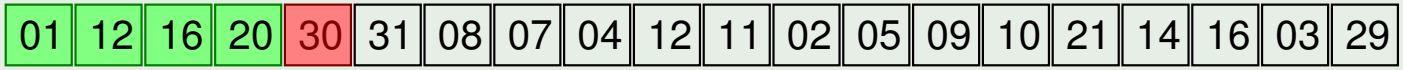
On applique la stratégie au quatrième élément du tableau,

Tri par insertion



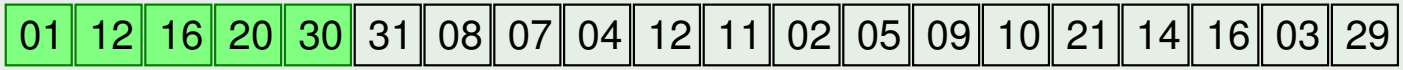
qu'on insère à la bonne position sur la gauche en décalant les éléments plus grands.

Tri par insertion



Comme la portion de gauche est toujours triée, il suffit d'appliquer le même principe jusqu'à atteindre la fin du tableau

Tri par insertion



Comme la portion de gauche est toujours triée, il suffit d'appliquer le même principe jusqu'à atteindre la fin du tableau

Tri par insertion

Trier par selection

```
variables
    entier tableau[n] ← {···}
    entier i,j,val

début
    pour i ← 1 à n − 1 faire
        val ← tableau[i]
        j ← i
        tant que (j > 0) et (tableau[j − 1] > val) faire
            tableau[j] ← tableau[j − 1]
            j ← j − 1
        fin
        tableau[j] ← val
    fin
fin
```

Tri par insertion

Trier par selection

```
variables
    entier tableau[n] ← {···}
    entier i,j,val

début
    pour i ← 1 à n − 1 faire
        val ← tableau[i]
        j ← i
        tant que (j > 0) et (tableau[j − 1] > val) faire
            tableau[j] ← tableau[j − 1]
            j ← j − 1
        fin
        tableau[j] ← val
    fin
fin
```

L'élément à insérer est stocké dans la variable "val"

Tri par insertion

Trier par selection

```
variables
    entier tableau[n] ← {···}
    entier i,j,val

début
    pour i ← 1 à n − 1 faire
        val ← tableau[i]
        j ← i
        tant que (j > 0) et (tableau[j − 1] > val) faire
            tableau[j] ← tableau[j − 1]
            j ← j − 1
        fin
        tableau[j] ← val
    fin
fin
```

L'élément à insérer est stocké dans la variable "val"

Les éléments de gauche plus grands que "val" sont décalés sur la droite

Tri par insertion

Trier par selection

```
variables
    entier tableau[n] ← {···}
    entier i,j,val

début
    pour i ← 1 à n − 1 faire
        val ← tableau[i]
        j ← i
        tant que (j > 0) et (tableau[j − 1] > val) faire
            tableau[j] ← tableau[j − 1]
            j ← j − 1
        fin
        tableau[j] ← val
    fin
fin
```

L'élément à insérer est stocké dans la variable "val"

Les éléments de gauche plus grands que "val" sont décalés sur la droite

L'élément "val" est inséré à position vacante

Résumé avec une mise en appétit de la complexité

Tableau

Suite consécutive d'éléments du même type, étant accessibles par leur index.

Résumé avec une mise en appétit de la complexité

Tableau

Suite consécutive d'éléments du même type, étant accessibles par leur index.

Algorithmes de recherche

Pour un tableau de n éléments,

- Recherche séquentielle : au plus n itérations
- Recherche dichotomique : au plus $\log_2 n$ itérations (mais il faut que le tableau soit trié)

Résumé avec une mise en appétit de la complexité

Tableau

Suite consécutive d'éléments du même type, étant accessibles par leur index.

Algorithmes de recherche

Pour un tableau de n éléments,

- Recherche séquentielle : au plus n itérations
- Recherche dichotomique : au plus $\log_2 n$ itérations (mais il faut que le tableau soit trié)

Algorithmes de tri

Pour un tableau de n éléments,

- Tri par sélection : au plus n^2 itérations
- Tri par insertion : au plus n^2 itérations
- Tri rapide : (étudié à la fin du semestre) au plus $n \log_2 n$ itérations