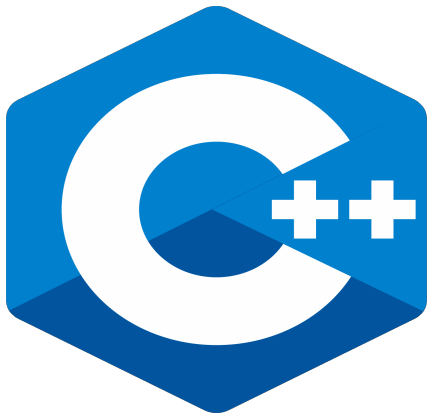
C++ : tableaux

Achref El Mouelhi

Docteur de l'université d'Aix-Marseille
Chercheur en programmation par contrainte (IA)
Ingénieur en génie logiciel

elmouelhi.achref@gmail.com



Plan

1. Introduction
2. Tableaux statiques
   - Tableau avec opérateur [ ]
   - Classe `array`
3. Paramètres du programme principal
4. `tuple`
5. STL Containers
   - `vector`
   - `set`
   - `map`

Plan

6 STL Algorithms

- `max_element` et `min_element`
- `sort`
- `reverse`
- `count`
- `find`
- `max` et `min`
- `accumulate`

C++

Tableaux

- Structure de données
- Permettant de sauvegarder simultanément plusieurs valeurs
- Chaque élément a un indice statique ou personnalisé

C++

Deux types de tableaux en **C++**

- statique : taille et type fixes
- dynamique : taille extensible et/ou types différents

C++

Tableaux statiques

- Tous les éléments ont le même type
- Il faut préciser une taille qu'on ne peut dépasser
- Deux types de tableau statique :
  - un tableau qu'on manipule avec les opérateurs `[ ]`
  - un tableau objet de la classe `array`

Déclaration d'un tableau : syntaxe

```
type nomTableau [taille];
```

C++

Déclaration d'un tableau : syntaxe

```
type nomTableau [taille];
```

Déclaration d'un tableau d'entier de taille 2 : exemple

```
int tab [2];
```

C++

Remarques

- Chaque élément du tableau est accessible via un indice : sa position dans le tableau
- Le premier élément est d'indice `0`
- Le dernier élément est d'indice `taille - 1`

C++

Remarques

- Chaque élément du tableau est accessible via un indice : sa position dans le tableau
- Le premier élément est d'indice `0`
- Le dernier élément est d'indice `taille - 1`

Affectation de valeurs au tableau

```
tab[0] = 5;
tab[1] = 3;
```

On peut faire une déclaration + une initialisation

```
int  tab [2] = {5, 3};
```

C++

On peut faire une déclaration + une initialisation

```
int  tab [2] = {5, 3};
```

Ou en plus simple

```
int  tab[]{5, 3};
```

C++

On peut faire une déclaration + une initialisation

```
int  tab [2] = {5, 3};
```

Ou en plus simple

```
int  tab[]{5, 3};
```


Comme si la taille du tableau a été fixée à deux.

C++

Problème

Il n'existe aucune fonction en **C++** qui retourne la taille d'un tableau (comme en **Java**, **C#**, **Python**...)

C++

Problème

Il n'existe aucune fonction en **C++** qui retourne la taille d'un tableau (comme en **Java**, **C#**, **Python**...)

Solution

Utiliser la fonction `sizeof()` pour calculer la taille d'un tableau.

C++

Problème

Il n'existe aucune fonction en **C++** qui retourne la taille d'un tableau (comme en **Java**, **C#**, **Python**...)

Solution

Utiliser la fonction `sizeof()` pour calculer la taille d'un tableau.

Pour connaître la taille d'un tableau, on divise le nombre d'octets réservés au tableau par le nombre d'octets réservés à un de ses éléments

```
cout << sizeof(tab) / sizeof(tab[0]) << endl;
// affiche 2
```

C++

On peut aussi omettre la taille

```
int  tab [] = {5, 3};
```

C++

On peut aussi omettre la taille

```
int  tab [] = {5, 3};
```

Ou aussi

```
int  tab [] {5, 3};
```

C++

On peut aussi omettre la taille

```
int  tab [] = {5, 3};
```

Ou aussi

```
int  tab [] {5, 3};
```

Il est aussi de préciser une taille qu'on n'utilise pas à l'initialisation

```
int  tab [4] {5, 3};
tab[2] = 8;
tab[3] = 1;
```

C++

Contrairement à certains langages de programmation (**Java**, **C#**...)

- Dépasser la taille déclarée d'un tableau est **syntaxiquement** correcte.
- Cependant, ceci peut générer une erreur à l'exécution.

C++

Contrairement à certains langages de programmation (**Java**, **C#**...)

- Dépasser la taille déclarée d'un tableau est **syntaxiquement** correcte.
- Cependant, ceci peut générer une erreur à l'exécution.

Le code suivant ne génère pas d'erreur

```
tab[2] = 10;
```

Afficher le tableau comme toute autre variable donne le résultat suivant

```
cout << tab << endl;
// affiche 0xf2893ff910
```

C++

Afficher le tableau comme toute autre variable donne le résultat suivant

```
cout << tab << endl;
// affiche 0xf2893ff910
```


En **C++**, on ne peut afficher le contenu de tableau sans le parcourir élément par élément.

C++

Pour parcourir et afficher le contenu d'un tableau

```
for (int i = 0; i < 4; i++)
{
  cout << tab[i] << endl;
}
```

C++

Pour parcourir et afficher le contenu d'un tableau

```
for (int i = 0; i < 4; i++)
{
  cout << tab[i] << endl;
}
```

Où encore la version simplifiée (`foreach`)

```
for (int elt : tab)
{
  cout << elt << endl;
}
```

C++

Le tableau est un pointeur sur la première case

```
cout << *tab  << endl;
// équivalent à
 cout << tab[0] << endl;
```

C++

Le tableau est un pointeur sur la première case

```
cout << *tab  << endl;
// équivalent à
 cout << tab[0] << endl;
```

On peut utiliser le pointeur pour accéder à un élément du tableau d'indice `i`

```
cout << *(tab + i) << endl;
// équivalent à
 cout << tab[i] << endl;
```

C++

Le tableau est un pointeur sur la première case

```
cout << *tab  << endl;
// équivalent à
 cout << tab[0] << endl;
```

On peut utiliser le pointeur pour accéder à un élément du tableau d'indice `i`

```
cout << *(tab + i) << endl;
// équivalent à
 cout << tab[i] << endl;
```

Attention

```
cout << *tab + i  << endl;
// équivalent à
 cout << tab[0] + i << endl;
```

C++

Exercice

Écrire un programme **C++** qui

1. demande à l'utilisateur de saisir le nombre de notes : `n`
2. demande à l'utilisateur de remplir un tableau avec `n` notes comprises entre 0 et 20
3. affiche le max, le min et la moyenne de notes

Pour déclarer un tableau à deux dimensions (matrice) composé de 2 ligne et 3 colonnes

```
int mat[2][3];
```

Pour déclarer un tableau à deux dimensions (matrice) composé de 2 ligne et 3 colonnes

```
int mat[2][3];
```

Pour déclarer et initialiser un tableau à deux dimensions

```
int mat[2][3] = {
    {1, 2, 3},
    {4, 5, 6}
};
```

C++

Exercice 1

Écrire un programme **C++** qui remplit le tableau précédent avec des valeurs saisies par l'utilisateur (afficher ensuite le contenu).

C++

Exercice 2

Écrire un programme **C++** qui calcule

1. la somme de deux matrices carrées,
2. le produit de deux matrices carrées.

C++

Classe `array` (**C++ 11**)

- définie dans **STL** : Standard Template Library
- pouvant fonctionner avec les opérateurs `[ ]`
- possibilité de récupérer la taille avec la méthode `size()`

C++

Avant utilisation, il faut inclure

```
#include <array>
```

C++

Avant utilisation, il faut inclure

```
#include <array>
```

Déclaration d'un tableau de taille 4

```
array<int, 4> tab = {4, 7, 1, 5};
```

C++

Avant utilisation, il faut inclure

```
#include <array>
```

Déclaration d'un tableau de taille 4

```
array<int, 4> tab = {4, 7, 1, 5};
```

Pour connaître la taille

```
cout << tab.size() << endl;
// affiche 4
```

C++

Avant utilisation, il faut inclure

```
#include <array>
```

Déclaration d'un tableau de taille 4

```
array<int, 4> tab = {4, 7, 1, 5};
```

Pour connaître la taille

```
cout << tab.size() << endl;
// affiche 4
```

Pour parcourir et afficher les éléments d'un tableau

```
for (int i = 0; i < tab.size(); i++)
    cout << tab[i] << " ";
// affiche 4 7 1 5
```

On peut aussi utiliser la méthode `at()` pour récupérer un élément

```
for (int i = 0; i < tab.size(); i++)
    cout << tab.at(i) << " ";
// affiche 4 7 1 5
```

On peut aussi utiliser la méthode `at()` pour récupérer un élément

```
for (int i = 0; i < tab.size(); i++)
    cout << tab.at(i) << " ";
// affiche 4 7 1 5
```

Ou aussi la fonction `get`

```
cout << get<1>(tab) << endl;
// affiche 7
```

C++

Autres méthodes de la classe `array`

- `t1.swap(t2)` : échange tous les éléments d'un tableau `t1` avec les éléments du tableau `t2`.
- `t.max_size()` : méthode définie pour tous les conteneurs. Pour `array`, aucune différence avec `size()`.
- `t.empty()` : retourne `true` si le tableau est vide, `false` sinon.
- `t.fill(x)` : affecte la valeur de `x` à tous les éléments du tableau.
- ...

C++

Structure du programme principal contenant la fonction `main`

```
#include <iostream>

using namespace std;

int main()
{
    return 0;
}
```

C++

La fonction `main` peut aussi avoir des paramètres

```
#include <iostream>

using namespace std;

int main(int argc, char *argv[])
{
    return 0;
}
```

C++

La fonction `main` peut aussi avoir des paramètres

```
#include <iostream>

using namespace std;

int main(int argc, char *argv[])
{
    return 0;
}
```


Explication

- `argc` : nombre de paramètres passés à la fonction `main`
- `argv` : tableau contenant les paramètres passés à la fonction `main`

C++

Question

Comment passer des paramètres à la fonction `main`

C++

Question

Comment passer des paramètres à la fonction `main`

Réponse

Au moment où on lance l'exécution avec la commande `./main`

C++

Affichons tous les paramètres reçus dans `main`

```
#include <iostream>

using namespace std;

int main(int argc, char *argv[])
{
    for (int i = 0; i < argc; i++)
    {
        cout << argv[i] << " ";
    }
    return 0;
}
```

C++

Compilons notre code avec la commande suivante

```
g++ main.cpp -o main
```

C++

Compilons notre code avec la commande suivante

```
g++ main.cpp -o main
```

Exécutons le fichier généré avec les trois paramètres suivants

```
./main 3 4 5
```

Résultat

```
3 4 5
```

C++

Exercice

Écrire un programme **C++** qui affiche la somme de paramètres positifs reçus par la fonction `main`.

C++

tuple

- Objet pouvant contenir des éléments de type différent
- Introduit dans **C++11**
- Taille statique

C++

Pour utiliser les tuples, il faut inclure

```
#include<tuple>
```

C++

Pour utiliser les tuples, il faut inclure

```
#include<tuple>
```

Déclaration d'un vecteur : syntaxe

```
tuple<type1, type2, ...> nom_tuple;
```

C++

Exemple

```
tuple<int, string, string, char> personne;
```

C++

Exemple

```
tuple<int, string, string, char> personne;
```

Pour affecter des valeurs à un tuple

```
personne = make_tuple(100, "wick", "john", 'h');
```

C++

Exemple

```
tuple<int, string, string, char> personne;
```

Pour affecter des valeurs à un tuple

```
personne = make_tuple(100, "wick", "john", 'h');
```

On peut aussi fusionner déclaration et initialisation

```
tuple<int, string, string, char> personne(100, "wick", "john", 'h');
```

C++

Pour accéder aux éléments d'un tuple

```
cout << get<0>(personne) << " " << get<1>(personne) << " ";
cout << get<2>(personne) << " " << get<3>(personne) << endl;
// affiche 100 wick john h
```

C++

Pour accéder aux éléments d'un tuple

```
cout << get<0>(personne) << " " << get<1>(personne) << " ";
cout << get<2>(personne) << " " << get<3>(personne) << endl;
// affiche 100 wick john h
```

`get` **peut être aussi utilisé pour modifier les éléments d'un tuple**

```
get<1>(personne) = "sophie";
get<3>(personne) = 'f';
cout << get<1>(personne) << " " << get<3>(personne) << endl;
// affiche  sophie f
```

C++

Pour accéder aux éléments d'un tuple

```
cout << get<0>(personne) << " " << get<1>(personne) << " ";
cout << get<2>(personne) << " " << get<3>(personne) << endl;
// affiche 100 wick john h
```

`get` **peut être aussi utilisé pour modifier les éléments d'un tuple**

```
get<1>(personne) = "sophie";
get<3>(personne) = 'f';
cout << get<1>(personne) << " " << get<3>(personne) << endl;
// affiche  sophie f
```

Pour la taille d'un tuple

```
cout << tuple_size<decltype(personne)>::value << endl;
// affiche  4
```

C++

Pour décomposer un tuple, on utilise la fonction `tie`

```
int num;
string nom, prenom;
char genre;

tie(num, nom, prenom, genre) = personne;
cout << num << " " << nom << " " << prenom << " " << genre << endl;
// affiche 100 sophie john f
```

C++

Pour décomposer un tuple, on utilise la fonction `tie`

```
int num;
string nom, prenom;
char genre;

tie(num, nom, prenom, genre) = personne;
cout << num << " " << nom << " " << prenom << " " << genre << endl;
// affiche 100 sophie john f
```


On ne peut itérer sur un tuple.

C++

Tableaux dynamiques (conteneurs)

- Pouvant être utilisés comme les tableaux statiques
- Taille extensible
- inclus dans le composant **Containers** de **STL**

C++

Tableaux dynamiques (conteneurs)

- Pouvant être utilisés comme les tableaux statiques
- Taille extensible
- inclus dans le composant **Containers** de **STL**

Deux catégories de conteneur

- Séquentiel : les éléments sont rangés les uns à coté des autres
- Associatif : un ensemble clé / valeur

C++

Les tableaux séquentiels

- `vector`
  - très proche des tableaux statiques
  - très couteux en insertion et suppression
  - adapté pour la recherche d'un élément
- `list`
  - utilisant le principe de double chaînage
  - plus efficace pour les opérations d'insertion et suppression
  - la recherche se fait dans un ordre séquentiel
- `queue` (file)
- `stack` (pile)

C++

Les tableaux associatifs

- `map` : la clé est unique et les éléments sont triés
- `set` : une map dont la clé = la valeur
- `multiset` : un set avec possibilité de doublons pour la clé
- `multimap` : une map avec possibilité de doublons pour la clé
- `unordered_set`
- `unordered_multiset`
- `unordered_map`
- `unordered_multimap`

Pour utiliser les vecteurs, il faut inclure

```
#include <vector>
```

C++

Pour utiliser les vecteurs, il faut inclure

```
#include <vector>
```

Déclaration d'un vecteur : syntaxe

```
vector<type> nomVecteur;
```

C++

Pour utiliser les vecteurs, il faut inclure

```
#include <vector>
```

Déclaration d'un vecteur : syntaxe

```
vector<type> nomVecteur;
```

Exemple

```
vector<int> vecteur;
```

C++

Pour insérer des éléments dans le vecteur

```
vecteur.push_back(2);
vecteur.push_back(5);
vecteur.push_back(8);
vecteur.push_back(4);
```

C++

Pour insérer des éléments dans le vecteur

```
vecteur.push_back(2);
vecteur.push_back(5);
vecteur.push_back(8);
vecteur.push_back(4);
```

Ou tout simplement

```
vector<int> vecteur = {2, 5, 8, 4};
```

C++

Pour insérer des éléments dans le vecteur

```
vecteur.push_back(2);
vecteur.push_back(5);
vecteur.push_back(8);
vecteur.push_back(4);
```

Ou tout simplement

```
vector<int> vecteur = {2, 5, 8, 4};
```

Ou aussi

```
vector<int> vecteur{2, 5, 8, 4};
```

Pour connaître le nombre d'éléments d'un vecteur

```
cout << vecteur.size() << endl;
// affiche 4
```

C++

Pour connaître le nombre d'éléments d'un vecteur

```
cout << vecteur.size() << endl;
// affiche 4
```

`max_size` **retourne le nombre maximum d'éléments qu'on peut stocker dans un set**

```
cout << vecteur.max_size() << endl;
// affiche 2305843009213693951
```

C++

Pour accéder à un élément du vecteur d'indice `i`

```
cout << vecteur[i] << endl;
```

C++

Pour accéder à un élément du vecteur d'indice `i`

```
cout << vecteur[i] << endl;
```

Ou aussi

```
cout << vecteur.at(i) << endl;
```

C++

Pour accéder à un élément du vecteur d'indice `i`

```
cout << vecteur[i] << endl;
```

Ou aussi

```
cout << vecteur.at(i) << endl;
```

Un raccourci pour accéder au premier élément

```
cout << vecteur.front() << endl;
```

C++

Pour accéder à un élément du vecteur d'indice `i`

```
cout << vecteur[i] << endl;
```

Ou aussi

```
cout << vecteur.at(i) << endl;
```

Un raccourci pour accéder au premier élément

```
cout << vecteur.front() << endl;
```

Un raccourci pour accéder au dernier élément

```
cout << vecteur.back() << endl;
```

C++

Pour parcourir et afficher tous les éléments d'un vecteur

```
for(int i = 0; i < vecteur.size(); i++)
{
    cout << vecteur[i] << " ";
}
```

C++

Pour parcourir et afficher tous les éléments d'un vecteur

```
for(int i = 0; i < vecteur.size(); i++)
{
    cout << vecteur[i] << " ";
}
```

Ou avec le `for` simplifié

```
for (int elt: vecteur)
{
    cout << elt << endl;
}
```

C++

Pour parcourir et afficher tous les éléments d'un vecteur

```
for(int i = 0; i < vecteur.size(); i++)
{
    cout << vecteur[i] << " ";
}
```

Ou avec le `for` simplifié

```
for (int elt: vecteur)
{
    cout << elt << endl;
}
```

Remarque

Possibilité de parcourir un vecteur avec un itérateur.

C++

Itérateur (Iterator)

- Composant de la **STL**
- Utilisé pour pointer une adresse mémoire relative à un élément d'un **Container** (**Vector** ou autre)
- Solution très optimale utilisée pour réduire le temps d'exécution

Pour déclarer un itérateur

```
vector<int>::iterator it = vecteur.begin();
```

C++

Pour déclarer un itérateur

```
vector<int>::iterator it = vecteur.begin();
```

Ou en plus simple

```
auto it = vecteur.begin();
```

C++

Pour récupérer le premier élément du vecteur en utilisant l'itérateur

```
cout << *it << endl;
// affiche 2
```

C++

Pour récupérer le premier élément du vecteur en utilisant l'itérateur

```
cout << *it << endl;
// affiche 2
```

Pour récupérer le deuxième élément du vecteur en utilisant l'itérateur

```
cout << *(it + 1) << endl;
// affiche 5
```

C++

Pour récupérer le premier élément du vecteur en utilisant l'itérateur

```
cout << *it << endl;
// affiche 2
```

Pour récupérer le deuxième élément du vecteur en utilisant l'itérateur

```
cout << *(it + 1) << endl;
// affiche 5
```

Ou en utilisant la fonction `advance` en indiquant le nombre de position

```
advance(it, 2);
cout << *it << endl;
// affiche 8
```

C++

On peut aussi utiliser `next` et `prev` qui retournent un nouvel itérateur pointant sur la position demandé

```
cout << *(next(it, 2)) << endl;
// affiche 8
```

C++

On peut aussi utiliser `next` et `prev` qui retournent un nouvel itérateur pointant sur la position demandé

```
cout << *(next(it, 2)) << endl;
// affiche 8
```

Pour récupérer le dernier élément du vecteur en utilisant l'itérateur : la position retournée par `end` est située juste après le vecteur

```
it = vecteur.end();
cout << *(it - 1) << endl;
// affiche 4
```

Pour parcourir un vecteur avec un itérateur

```
for(auto it = vecteur.begin(); it != vecteur.end(); it++)
{
    cout << *it << ' ' ;
}
```

C++

Exercice 1

Écrire un programme **C++** qui

1. demande à l'utilisateur de remplir un vecteur avec des nombres différents de zéro, il s'arrête à la saisie de zéro,
2. demande à l'utilisateur de saisir une valeur à chercher dans le vecteur,
3. affiche la position de cette valeur si elle est dans le vecteur, -1 sinon.

Proposer deux algorithmes de recherche : un premier avec itérateur et un deuxième sans.

C++

Solution avec itérateur

```
int main()
{
    vector<int> vecteur;
    int n;
    cout << "saisir un nombre != de 0" << endl;
    cin >> n;

    while (n != 0)
    {
        vecteur.push_back(n);
        cout << "saisir un nombre  != de 0" << endl;
        cin >> n;
    }

    cout << "quelle valeur vous cherchez ?" << endl;
    int val, found = -1;
    cin >> val;

    for (auto it = vecteur.begin(); it != vecteur.end(); it++)
    {
        if (*it == val)
        {
            found = it - vecteur.begin();
        }
    }

    cout << found << endl;
    return 0;
}
```

C++

Solution sans itérateur

```
int main()
{
    vector<int> vecteur;
    int n;
    cout << "saisir un nombre != de 0" << endl;
    cin >> n;

    while (n != 0)
    {
        vecteur.push_back(n);
        cout << "saisir un nombre != de 0" << endl;
        cin >> n;
    }

    cout << "quelle valeur vous cherchez ?" << endl;
    int val, found = -1;
    cin >> val;

    for (int i = 0; i < vecteur.size(); i++)
    {
        if (vecteur[i] == val)
        {
            found = i;
        }
    }

    cout << found << endl;
    return 0;
}
```

Pour supprimer le dernier élément

```
vecteur.pop_back();
```

Pour supprimer le dernier élément

```
vecteur.pop_back();
```

Pour supprimer l'élément du vecteur d'indice `i`

```
vecteur.erase(vecteur.begin() + i)
```

C++

Pour supprimer le dernier élément

```
vecteur.pop_back();
```

Pour supprimer l'élément du vecteur d'indice `i`

```
vecteur.erase(vecteur.begin() + i)
```

Pour supprimer les trois premiers éléments

```
vecteur.erase(vecteur.begin(), vecteur.begin() + 3)
```

C++

Exercice 1

Écrire un programme **C++** qui

1. demande à l'utilisateur de remplir un vecteur avec des nombres différents de zéro, il s'arrête à la saisie de zéro
2. demande à l'utilisateur de saisir une valeur à supprimer du vecteur
3. supprime la première occurrence de cette valeur du vecteur
4. affiche le nouveau vecteur (après suppression de la valeur demandée)

Solution

```
int main()
{
    vector<int> vecteur;
    int n;
    cout << "saisir un nombre != de 0" << endl;
    cin >> n;

    while (n != 0)
    {
        vecteur.push_back(n);
        cout << "saisir un nombre  != de 0" << endl;
        cin >> n;
    }

    cout << "quelle valeur vous cherchez ?" << endl;
    int val, found = -1;
    cin >> val;

    for (auto it = vecteur.begin(); it != vecteur.end(); it++)
    {
        if (*it == val)
        {
            vecteur.erase(it);
        }
    }

    for (int elt : vecteur)
    {
        cout << elt << " ";
    }
    return 0;
}
```

C++

Les vecteurs peuvent être utilisés comme les tableaux statiques

```
// déclarer un vecteur de taille 4
vector <int> vecteur (4);

// affecter 4 valeurs au vecteur
vecteur[0] = 2;
vecteur[1] = 5;
vecteur[2] = 8;
vecteur[3] = 4;

for (int elt: vecteur)
{
  cout << elt << " ";
}
// affiche 2 5 8 4
```

C++

Si on dépasse la taille du vecteur, ça génère pas d'erreur mais les éléments ne seront pas pris en compte

```
vecteur[4] = 1;
vecteur[5] = 7;

for (int elt: vecteur)
{
  cout << elt << " ";
}
// affiche 2 5 8 4
//Les deux derniers éléments ne seront pas affichés
```

C++

Deux solutions pour étendre la taille d'un tableau

- D'une manière explicite en changeant la taille du tableau
- D'une manière implicite en ajoutant les éléments avec `push_back()`

Première solution

```
vecteur.resize(6);
vecteur[4] = 1;
vecteur[5] = 7;

for (int elt: vecteur)
{
  cout << elt << " ";
}
// affiche 2 5 8 4 1 7
```

C++

Deuxième solution

```
vecteur.push_back(1);
vecteur.push_back(7);

for (int elt: vecteur)
{
  cout << elt << " ";
}
// affiche 2 5 8 4 1 7
```

C++

Pour extraire un sous-vecteur

```
vector<int> vecteur{2, 5, 8, 4};
vector<int> sousVecteur(vecteur.begin() + 1, vecteur.begin() + 3);

for (int elt : sousVecteur)
{
    cout << elt << " ";
}
// affiche 5 8
```

C++

Pour extraire un sous-vecteur

```
vector<int> vecteur{2, 5, 8, 4};
vector<int> sousVecteur(vecteur.begin() + 1, vecteur.begin() + 3);

for (int elt : sousVecteur)
{
    cout << elt << " ";
}
// affiche 5 8
```

Ou en utilisant la fonction `copy`

```
vector<int> vecteur{2, 5, 8, 4};
vector<int> sousVecteur(2);
copy(vecteur.begin() + 1, vecteur.begin() + 3, sousVecteur.begin());
for (int elt : sousVecteur)
{
    cout << elt << " ";
}
// affiche 5 8
```

C++

Étant donnée la liste suivante

```
vector<int> vecteur{1, 2, 7, 2, 1, 3, 9, 2, 4, 2, 6, 3};
```

C++

Étant donnée la liste suivante

```
vector<int> vecteur{1, 2, 7, 2, 1, 3, 9, 2, 4, 2, 6, 3};
```

Exercice

Écrire un programme **C++** qui permet d'extraire un sous-vecteur contenant les éléments du vecteur précédent situés entre le premier et le dernier 2.

C++

Solution

```
int main()
{
    vector<int> vecteur{1, 2, 7, 2, 1, 3, 9, 2, 4, 2, 6, 3};
    int start = -1, end;
    for (int i = 0; i < vecteur.size(); i++)
    {
        if (vecteur[i] == 2)
        {
            if (start == -1)
            {
                start = i;
            }
            else
            {
                end = i;
            }
        }
    }

    vector<int> subVector(start, end + 1);

    for (int i : subVector)
    {
        cout << i << " ";
    }
    return 0;
}
```

C++

set

- Ensemble trié
- Pas de doublons
- Pas d'accès aux éléments via l'indice
- Clé = valeur

Pour utiliser les sets, il faut inclure

```
#include <set>
```

C++

Pour utiliser les sets, il faut inclure

```
#include <set>
```

Déclaration d'un set : syntaxe

```
set <type> nomSet;
```

C++

Pour utiliser les sets, il faut inclure

```
#include <set>
```

Déclaration d'un set : syntaxe

```
set <type> nomSet;
```

Exemple

```
set <int> ensemble;
```

C++

Pour insérer des éléments dans le set

```
ensemble.insert(2);
ensemble.insert(5);
ensemble.insert(8);
ensemble.insert(4);
```

C++

Pour insérer des éléments dans le set

```
ensemble.insert(2);
ensemble.insert(5);
ensemble.insert(8);
ensemble.insert(4);
```

Ou tout simplement

```
set <int> ensemble = {2, 5, 8, 4};
```

C++

Pour insérer des éléments dans le set

```
ensemble.insert(2);
ensemble.insert(5);
ensemble.insert(8);
ensemble.insert(4);
```

Ou tout simplement

```
set <int> ensemble = {2, 5, 8, 4};
```

Ou encore

```
set <int> ensemble{2, 5, 8, 4};
```

Pour connaître le nombre d'élément d'un set

```
cout << ensemble.size() << endl;
// affiche 4
```

C++

Pour parcourir et afficher tous les éléments d'un set avec le for simplifié

```
for (int elt: ensemble)
{
    cout << elt << endl;
}
```

C++

Pour parcourir et afficher tous les éléments d'un set avec le for simplifié

```
for (int elt: ensemble)
{
    cout << elt << endl;
}
```

Le résultat est ordonné

```
2
4
5
8
```

C++

Pour parcourir et afficher tous les éléments d'un set avec le for simplifié

```
for (int elt: ensemble)
{
    cout << elt << endl;
}
```

Le résultat est ordonné

```
2
4
5
8
```

Ou avec les itérateurs

```
for(auto it = ensemble.begin(); it != ensemble.end(); it++)
{
    cout << *it << ' ' ;
}
```

C++

Impossible d'ajouter un élément déjà présent dans le set

```
ensemble.insert(4);
for (int elt: ensemble)
{
    cout << elt << endl;
}
```


Le résultat est le même et la valeur 4 est présente une seule fois

```
2
4
5
8
```

Pour supprimer le premier élément d'un set

```
ensemble.erase(ensemble.begin())
```

Pour supprimer le premier élément d'un set

```
ensemble.erase(ensemble.begin())
```

Pour supprimer la valeur 5

```
ensemble.erase(5)
```

C++

map

- Ensemble de pair (couple) : clé - valeur
- Clé unique
- Pas d'accès aux éléments via l'indice

Pour utiliser les dictionnaires, il faut inclure

```
#include <map>
```

C++

Pour utiliser les dictionnaires, il faut inclure

```
#include <map>
```

Déclaration d'une map : syntaxe

```
map<key_type, value_type> map_name;
```

C++

Pour utiliser les dictionnaires, il faut inclure

```
#include <map>
```

Déclaration d'une map : syntaxe

```
map<key_type, value_type> map_name;
```


Exemple

```
map<int, string> dictionnaire;
```

Pour ajouter des éléments

```
dictionnaire.emplace(1, "FCB");
dictionnaire.emplace(2, "RM");
```

Pour ajouter des éléments

```
dictionnaire.emplace(1, "FCB");
dictionnaire.emplace(2, "RM");
```

Pour initialiser un dictionnaire à la déclaration

```
map<int, string> dictionnaire {
    { 1, "FCB" },
    { 2, "RM" }
};
```

Impossible d'ajouter un élément avec une clé qui existe

```
dictionnaire.emplace(2, "MU");
```

C++

Impossible d'ajouter un élément avec une clé qui existe

```
dictionnaire.emplace(2, "MU");
```

On peut aussi ajouter des éléments ainsi

```
dictionnaire[4] = "MU";
```

C++

Pour parcourir et afficher les éléments d'une `map`

```
for (pair<int, string> elt: dictionnaire)
{
    cout << "cle " << elt.first << " " ;
    cout << "valeur " << elt.second << endl;
}
```

C++

Pour parcourir et afficher les éléments d'une `map`

```
for (pair<int, string> elt: dictionnaire)
{
    cout << "cle " << elt.first << " " ;
    cout << "valeur " << elt.second << endl;
}
```

Ou en utilisant un itérateur

```
for (auto it = dictionnaire.begin(); it != dictionnaire.end(); it++)
{
    cout << "cle " << it->first << " " ;
    cout << "valeur " << it->second << endl;
}
```

Pour supprimer l'élément ayant la clé 3

```
dictionnaire.erase(3);
```

C++

Pour supprimer l'élément ayant la clé 3

```
dictionnaire.erase(3);
```

Pour chercher un élément selon une clé

```
auto it = dictionnaire.find(1);
// retourne un itérateur sur map
if (it != dictionnaire.end())
{
    cout << "trouvée";
}
else
{
    cout << "non trouvée";
}
// affiche trouvée
```

C++

Étant donné le map suivant

```
map<string, int> repetition;
repetition.emplace("Java", 2);
repetition.emplace("PHP", 5);
repetition.emplace("C++", 1);
repetition.emplace("HTML", 4);
```

Exercice 1

Écrire un programme **C++** qui permet de répéter l'affichage de chaque clé de ce dictionnaire autant de fois que la valeur associée

Résultat attendu (l'ordre n'a pas d'importance) :

```
JavaJava PHPPHPPHPPHPPHP C++ HTMLHTMLHTMLHTML
```

C++

Considérons le map suivant

```
map<int, string> fcb;
fcb.emplace(10, "Messi");
fcb.emplace(9, "Suarez");
fcb.emplace(23, "Umtiti");
fcb.emplace(4, "Rakitic");
```


Exercice 2

Écrire un programme **C++** qui demande à l'utilisateur de saisir un nom (valeur) et affiche le numéro associé à ce nom (clé) si ce dernier existe, -1 sinon.

C++

Algorithm

- Composant de la **STL**
- Contenant un ensemble de fonctions d'agrégation et de recherche pour les vecteurs et les autres structures de données

C++

Algorithm

- Composant de la **STL**
- Contenant un ensemble de fonctions d'agrégation et de recherche pour les vecteurs et les autres structures de données

Pour consulter la liste d'algorithmes implémentés

`https://en.cppreference.com/w/cpp/algorithm`

C++

Commençons par l'inclure dans le programme

```
#include <algorithm>
```

C++

Commençons par l'inclure dans le programme

```
#include <algorithm>
```

Considérons le vecteur suivant

```
vector<int> vecteur{2, 5, 8, 4};
```

C++

Pour chercher le max d'un vecteur, on utilise `max_element` qui retourne un itérateur sur la valeur recherchée

```
cout << *max_element(vecteur.begin(), vecteur.end()) << endl;
// affiche 8

cout << *max_element(vecteur.begin(), vecteur.begin() + 2) <<
   endl;
// affiche 5
```

C++

Pour chercher le max d'un vecteur, on utilise `max_element` qui retourne un itérateur sur la valeur recherchée

```
cout << *max_element(vecteur.begin(), vecteur.end()) << endl;
// affiche 8

cout << *max_element(vecteur.begin(), vecteur.begin() + 2) <<
   endl;
// affiche 5
```

Pour chercher le min d'un vecteur, on utilise `min_element` qui retourne un itérateur sur la valeur recherchée

```
cout << *min_element(vecteur.begin(), vecteur.end()) << endl;
// affiche 2

cout << *min_element(vecteur.begin(), vecteur.begin() + 2) <<
   endl;
// affiche 2
```

C++

Pour trier les éléments d'un vecteur, on utilise `sort`

```
sort(vecteur.begin(), vecteur.end());

for (int elt : vecteur)
{
    cout << elt << " ";
}
// affiche 2 4 5 8
```

C++

Pour inverser les éléments d'un vecteur, on utilise `reverse`

```
reverse(vecteur.begin(), vecteur.end());

for (int elt : vecteur)
{
    cout << elt << " ";
}
// affiche 4 8 5 2
```

C++

Pour compter le nombre d'occurrence d'un élément dans un vecteur, on utilise `count`

```
cout << count(vecteur.begin(), vecteur.end(), 2) << " ";
// affiche 1

cout << count(vecteur.begin(), vecteur.end(), -2) << " ";
// affiche 0

cout << count(vecteur.begin(), vecteur.begin() + 2, 2) << " ";
// affiche 1
```

C++

Pour chercher un élément dans un vecteur, on utilise `find`

```
if (find(vecteur.begin(), vecteur.end(), 2) != vecteur.end())
{
    cout << "trouvé" << endl;
}
else
{
    cout << "non trouvé" << endl;
}
// affiche trouvée
```

C++

Pour trouver le max ou le min de deux valeurs

```
cout << max(2, 5) << endl;
// affiche 5

cout << max(5, 2) << endl;
// affiche 5

cout << min(2, 5) << endl;
// affiche 2

cout << min(5, 2) << endl;
// affiche 2
```

C++

accumulate

- Fonction définie dans `numeric`
- Permettant de réaliser plusieurs opérations définies dans la librairie `Functional` de la **STL**.

C++

Pour calculer la somme et le produit de tous les éléments d'un vecteur

```
int sum = accumulate(vecteur.begin(), vecteur.end(), 0);
cout << sum << endl;
// affiche 19

int product = accumulate(vecteur.begin(), vecteur.end(), 1, multiplies<
   int>());
// affiche 320
cout << product << endl;
```

C++

Pour calculer la somme et le produit de tous les éléments d'un vecteur

```
int sum = accumulate(vecteur.begin(), vecteur.end(), 0);
cout << sum << endl;
// affiche 19

int product = accumulate(vecteur.begin(), vecteur.end(), 1, multiplies<
   int>());
// affiche 320
cout << product << endl;
```


N'oublions pas d'inclure `numeric`

```
#include <numeric>
```

C++

Pour consulter la liste de toutes les autres fonctions

`https://en.cppreference.com/w/cpp/utility/functional`