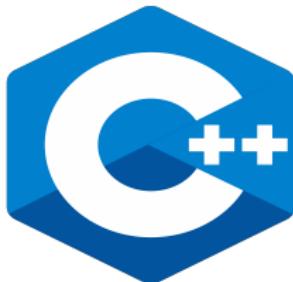


# C++ : fonctions

**Achref El Mouelhi**

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

[elmouelhi.achref@gmail.com](mailto:elmouelhi.achref@gmail.com)



# Plan

- 1 Introduction
- 2 Passage par valeur
- 3 Passage par référence
- 4 Passage par référence constante
- 5 Passage par adresse (pointeur)
- 6 Passage par pointeur constant
- 7 Variable locale statique
- 8 Fonction inline

## Une fonction ?

- Un bloc d'instructions exécuté lorsqu'il est appelé
- Pouvant recevoir 0, 1 ou plusieurs paramètres
- Pouvant retourner une valeur (résultat d'un certain calcul)

## Une fonction ?

- Un bloc d'instructions exécuté lorsqu'il est appelé
- Pouvant recevoir 0, 1 ou plusieurs paramètres
- Pouvant retourner une valeur (résultat d'un certain calcul)

## Syntaxe (de déclaration d'une fonction)

```
valeurDeRetour nomFonction ([les paramètres])  
{  
    bloc d'instructions = définition  
}
```

## Déclaration d'une fonction qui retourne une valeur entière

```
int somme(int a, int b)
{
    return a + b;
}
```

© Achref EL MOUADJI

## Déclaration d'une fonction qui retourne une valeur entière

```
int somme(int a, int b)
{
    return a + b;
}
```

### Remarque

- C++ est un langage séquentiel (fonctionnant de haut vers le bas)
- La fonction doit être déclarée avant qu'elle soit appelée

# C++

## Comment appeler une fonction ?

```
int main()
{
    int x = 2, y = 3;
    int resultat = somme (x, y);
    cout << "Le résultat est " << resultat << endl;
    return 0;
}
```



# C++

## Comment appeler une fonction ?

```
int main()
{
    int x = 2, y = 3;
    int resultat = somme (x, y);
    cout << "Le résultat est " << resultat << endl;
    return 0;
}
```



### Remarque

Si la fonction ne retourne rien, la valeur de retour à préciser sera `void`.

## Deux autres solutions pour l'emplacement de fonctions

- Déclarer le **prototype** de la fonction avant le `main` et la définir après.
- Déclarer la fonction dans un autre fichier et l'inclure dans le fichier contenant l'appel de la fonction (**à voir dans le prochain chapitre**).

# C++

## La deuxième solution

```
#include <iostream>

using namespace std;

int somme (int a, int b);

int main()
{
    int x = 2, y = 3;
    int resultat = somme (x, y);
    cout << "Le résultat est " << resultat << endl;
    return 0;
}

int somme (int a, int b)
{
    return a + b;
}
```

## C++ permet la surcharge (fonctions avec même nom et signatures différentes)

```
int somme (int a, int b);
int somme (int a, int b, int c);

int main()
{
    int x = 2, y = 3, z = 7;
    int resultat1 = somme (x, y);
    cout << "Le résultat est " << resultat1 << endl;

    int resultat2 = somme (x, y, z);
    cout << "Le résultat est " << resultat2 << endl;
    return 0;
}

int somme(int a, int b)
{
    return a + b;
}

int somme(int a, int b, int c)
{
    return a + b + c;
}
```

# C++

Il est possible de définir des valeurs par défaut pour les paramètres

```
int produit (int a = 0, int b = 1)
{
    return a * b;
}
```

# C++

Il est possible de définir des valeurs par défaut pour les paramètres

```
int produit (int a = 0, int b = 1)
{
    return a * b;
}
```

Exemple d'appel

```
cout << "produit sans paramètres " << produit() << endl;
// affiche produit sans paramètres 0
```

```
cout << "produit avec un paramètre " << produit(5) << endl;
// affiche produit avec un paramètre 5
```

## La fonction principale `main`

- Elle a une valeur de retour de type `int`
- Le programme fonctionne même si on ne retourne pas explicitement une valeur
- Le compilateur fait comme si la fonction se termine par une instruction `return` implicite `return 0`

# C++

## Exercice 1

- Écrire une fonction `repeter` qui
  - prend un premier paramètre `str` de type `string`
  - prend un deuxième paramètre `n` de type `int`
  - retourne une chaîne de caractère contenant `n` fois la concaténation de `str`
- La fonction doit fonctionner quel que soit le nombre de paramètres reçus (0, 1 ou 2).

## Résultat attendu

```
int main()
{
    cout << repeter() << endl;
    // n'affiche rien
    cout << repeter("bonjour") << endl;
    // affiche bonjour
    cout << repeter("bonjour", 3) << endl;
    // affiche bonjourbonjourbonjour
    return 0;
}
```

## Exercice 2

Écrire une fonction `evenFromArray` qui

- prend un premier paramètre obligatoire `vecteur` de type `vector<int>`
- prend un deuxième paramètre `opt` de type `bool` avec une valeur par défaut `true`
- retourne les valeurs paires de `vecteur` si `opt == true`, les valeurs impaires sinon.

## Résultat attendu

```
int main()
{
    vector<int> vec{1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
    auto pairs = evenFromArray(vec, true);
    for (int elt : pairs)
    {
        cout << elt << " ";
    }
    // affiche 2 4 6 8 10

    auto impairs = evenFromArray(vec, false);
    for (int elt : impairs)
    {
        cout << elt << " ";
    }
    // affiche 1 3 5 7 9

    return 0;
}
```

# C++

## Correction

```
vector<int> evenFromArray(vector<int> vecteur, bool opt = true)
{
    vector<int> result;
    for (int i = 0; i < vecteur.size(); i++)
    {
        if (vecteur[i] % 2 == opt)
        {
            result.push_back(vecteur[i]);
        }
    }
    return result;
}
```

## Exercice

Écrire une fonction qui permet de permuter le contenu de deux variables (de type entier par exemple) passées en paramètre

## Exercice

Écrire une fonction qui permet de permuter le contenu de deux variables (de type entier par exemple) passées en paramètre

Voici à quoi en pense la première fois

```
void permuter(int a, int b)
{
    int aux = a;
    a = b;
    b = aux;
}
```

# C++

En testant ce code, c'est la surprise

```
int main()
{
    int x = 2, y = 3;
    permuter(x, y);
    cout << x << " " << y << endl;
    // affiche 2 3
    return 0;
}
```

# C++

En testant ce code, c'est la surprise

```
int main()
{
    int x = 2, y = 3;
    permuter(x, y);
    cout << x << " " << y << endl;
    // affiche 2 3
    return 0;
}
```

Et pourtant, si on vérifie dans la fonction, les valeurs ont bien été permuteées

```
void permuter(int a, int b)
{
    int aux = a;
    a = b;
    b = aux;
    cout << a << " " << b << endl;
    // affiche 3 2
}
```

## Passage par valeur

- Les fonctions, par définition, reçoivent une copie du contenu de la variable
- Elles ne manipulent pas la case de mémoire de la variable envoyée
- Ceci s'appelle **passage par valeur**



# C++

## Passage par valeur

- Les fonctions, par définition, reçoivent une copie du contenu de la variable
- Elles ne manipulent pas la case de mémoire de la variable envoyée
- Ceci s'appelle **passage par valeur**

## Quelles solutions ?

- Passage par référence
- Passage par adresse (pointeur)

# C++

Pour passer les références de nos paramètres, on utilise &

```
void permuter(int &a, int &b)
{
    int aux = a;
    a = b;
    b = aux;
}
```

# C++

Pour passer les références de nos paramètres, on utilise &

```
void permuter(int &a, int &b)
{
    int aux = a;
    a = b;
    b = aux;
}
```

Pour tester, le contenu de la fonction principale (`main`) ne change pas

```
int main()
{
    int x = 2, y = 3;
    permuter(x, y);
    cout << x << " " << y << endl;
    // affiche 3 2
    return 0;
}
```

## Exercice

Écrire une fonction `minMax` qui

- prend un premier paramètre obligatoire `tab` de type tableau statique de 5 `int`
- prend un deuxième paramètre obligatoire `min` de type `int`
- prend un troisième paramètre obligatoire `max` de type `int`
- affecte à `min` (et respectivement `max`) la plus petite valeur du tableau (resp. la plus grande valeur du tableau) sans retourner de valeur.

## Résultat attendu

```
int main()
{
    int t[]{2, 3, 8, 5, 1};
    int min, max;
    minMax(t, min, max);
    cout << min << " : " << max << endl;
    // affiche 1 : 8
    return 0;
}
```

# C++

## Solution

```
void minMax(int tab[5], int &min, int &max)
{
    min = tab[0];
    max = tab[0];
    for (int i = 1; i < 5; i++)
    {
        if (tab[i] > max)
        {
            max = tab[i];
        }
        if (tab[i] < min)
        {
            min = tab[i];
        }
    }
}
```

## Compromis entre passage par valeur et par référence

- Le passage par valeur duplique le contenu de la variable
- Et si le contenu est trop long ? et qu'on ne veut pas modifier le contenu originel du paramètre
- Une solution consiste à passer le paramètre par référence (pour ne pas dupliquer le contenu) et le déclarer comme constante (pour conserver son contenu)

**Redéfinition de la fonction somme en utilisant les références constantes**

```
int somme(int const& a, int const& b)
{
    return a + b;
}
```

© Achref EL MOUELLI

# C++

**Redéfinition de la fonction somme en utilisant les références constantes**

```
int somme(int const& a, int const& b)
{
    return a + b;
}
```

**Le contenu de la fonction principale**

```
int main()
{
    int x = 2, y = 3;
    int resultat = somme (x, y);
    cout << "Le résultat est " << resultat << endl;
    return 0;
}
```

# C++

Pour avoir les adresses de nos paramètres, on utilise \* (le pointeur)

```
void permuter(int *a, int *b)
{
    int aux = *a;
    *a = *b;
    *b = aux;
}
```

# C++

Pour avoir les adresses de nos paramètres, on utilise \* (le pointeur)

```
void permuter(int *a, int *b)
{
    int aux = *a;
    *a = *b;
    *b = aux;
}
```

Pour utiliser les pointeurs, on doit envoyer des références

```
int main()
{
    int x = 2, y = 3;
    permuter(&x, &y);
    cout << x << " " << y << endl;
    // affiche 3 2
    return 0;
}
```

## Exercice (à refaire avec le passage par adresse)

Écrire une fonction `minMax` qui

- prend un premier paramètre obligatoire `tab` de type tableau statique de 5 `int`
- prend un deuxième paramètre obligatoire `min` de type `int`
- prend un troisième paramètre obligatoire `max` de type `int`
- affecte à `min` (et respectivement `max`) la plus petite valeur du tableau (resp. la plus grande valeur du tableau) sans retourner de valeur.

# C++

## Solution

```
void minMax(int tab[5], int *min, int *max)
{
    *min = tab[0];
    *max = tab[0];
    for (int i = 1; i < 5; i++)
    {
        if (tab[i] > *max)
        {
            *max = tab[i];
        }
        if (tab[i] < *min)
        {
            *min = tab[i];
        }
    }
}

int main()
{
    int t[]{2, 3, 8, 5, 1};
    int min, max;

    minMax(t, &min, &max);
    cout << min << " : " << max << endl;
    // affiche 1 : 8
    return 0;
}
```

## Remarques

- En C++, les tableaux sont passés par adresse (pointeur).
- Ce n'est pas le cas pour les vecteurs et les autres conteneurs **STL**.

## Exercice

Écrire une fonction `permuteArray` qui

- prend un premier paramètre obligatoire `tab` de type tableau statique de `int`
- prend un deuxième paramètre obligatoire `i` de type `int`
- prend un troisième paramètre obligatoire `j` de type `int`
- permute le contenu des éléments d'indice `i` et `j` sans rien retourner.

## C++

## Correction

```
void permuterArray(int tab[], int i, int j)
{
    int tmp = tab[i];
    tab[i] = tab[j];
    tab[j] = tmp;
}

int main()
{
    int t[]{2, 3, 8, 5, 1};
    permuterArray(t, 1, 3);
    for (int elt : t)
    {
        cout << elt << " ";
    }
    // affiche 2 5 8 3 1
    return 0;
}
```

## Compromis entre passage par valeur et par pointeur

- Le passage par valeur duplique le contenu de la variable
- Et si le contenu est trop long ? et qu'on ne veut pas modifier le contenu originel du paramètre
- Une solution consiste à passer le paramètre par adresse (pour ne pas dupliquer le contenu) et le déclarer comme constant (pour conserver son contenu)

**Redéfinition de la fonction somme en utilisant les pointeurs constants**

```
int somme (int const *a, int const *b)
{
    return *a + *b;
}
```

© Achref EL MOUELLI

# C++

**Redéfinition de la fonction somme en utilisant les pointeurs constants**

```
int somme (int const *a, int const *b)
{
    return *a + *b;
}
```

**Le contenu de la fonction principale**

```
int main()
{
    int x = 2, y = 3;
    int resultat = somme (&x, &y);
    cout << "Le résultat est " << resultat << endl;
    return 0;
}
```

# C++

Étant donnée la fonction suivante

```
void incrementI()
{
    int i = 0;
    i++;
    cout << i << " ";
}
```

# C++

Étant donnée la fonction suivante

```
void incrementI()
{
    int i = 0;
    i++;
    cout << i << " ";
}
```

Testons le code suivant

```
int main()
{
    incrementI();
    incrementI();
    incrementI();
}
// affiche 1 1 1
```

## Explication

- Par définition d'une fonction, les variables locales se réinitialisent à chaque appel
- Donc, à chaque appel, la variable `i` est initialisée à 0, incrémentée puis affichée

© Achref EL MOUELLI

# C++

## Explication

- Par définition d'une fonction, les variables locales se réinitialisent à chaque appel
- Donc, à chaque appel, la variable `i` est initialisée à 0, incrémentée puis affichée

## Question

Et si on veut préserver la valeur de l'appel précédent ?

# C++

## Explication

- Par définition d'une fonction, les variables locales se réinitialisent à chaque appel
- Donc, à chaque appel, la variable `i` est initialisée à 0, incrémentée puis affichée

## Question

Et si on veut préserver la valeur de l'appel précédent ?

## Réponse

On peut utiliser les variables statiques.

# C++

Déclarons la variable locale `i` comme étant statique

```
void incrementI()
{
    static int i = 0;
    i++;
    cout << i << " ";
}
```

# C++

Déclarons la variable locale `i` comme étant statique

```
void incrementI()
{
    static int i = 0;
    i++;
    cout << i << " ";
}
```

Vérifions maintenant l'exécution

```
int main()
{
    incrementI();
    incrementI();
    incrementI();
}
// affiche 1 2 3
```

## Fonction inline

- Permet d'indiquer au compilateur qu'il faut remplacer chaque appel de la fonction par son corps
- Permet d'optimiser le programme et d'accélérer son exécution si le nombre d'instruction est très petit

## Fonction inline

- Permet d'indiquer au compilateur qu'il faut remplacer chaque appel de la fonction par son corps
- Permet d'optimiser le programme et d'accélérer son exécution si le nombre d'instruction est très petit

### Exemple

```
inline int somme (int a, int b)
{
    return a + b;
}
```

## Fonction inline

- Permet d'indiquer au compilateur qu'il faut remplacer chaque appel de la fonction par son corps
- Permet d'optimiser le programme et d'accélérer son exécution si le nombre d'instruction est très petit

### Exemple

```
inline int somme (int a, int b)
{
    return a + b;
}
```

Rien ne change pour l'appel

```
int main()
{
    int x = 2, y = 3;
    cout << somme(x, y) << endl;
}
```