Filière Mécatronique – 2017/2018
Programmation en C++
Fonctions : Passage des arguments

Prof. *A. DARGHAM*

Passage des arguments par valeur

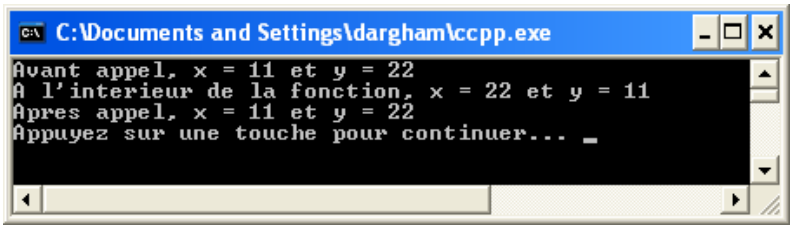
- **Exemple** :

```
#include <iostream>
using namespace std;
void permuter(int x, int y)
{
   int temp;
   temp = x;
   x = y;
   y = temp;
   cout << "A l'interieur de la fonction,  x = " << x;
   cout << " et y = " << y << endl;
}
```

Passage des arguments par valeur

```
int main()
{
   int x = 11, y = 22;

   cout << "Avant appel, x = " << x << " et y = " << y;
   cout << endl;
   permuter(x, y);
   cout << "Apres appel, x = " << x << " et y = " << y;
   cout << endl;
   system("pause");
   return 0;
}
```

Passage des arguments par valeur



Passage des arguments par valeur

- **Avant l'appel**, les **paramètres effectifs** sont x = 11 et y = 22.
- **Lors de l'appel de la fonction** « permuter » :
  - Le **paramètre formel** x est **initialisé avec la valeur** 11.
  - Le **paramètre formel** y est **initialisé avec la valeur** 22.
- **À l'intérieur de la fonction** « permuter » :
  - Les **paramètres formels** x et y **ont été bien échangés**.
- **Après l'appel**, les **paramètres effectifs** sont x = 11 et y = 22 : ils **n'ont pas été modifiés**.

Passage des arguments par valeur

- Le **paramètre formel x** se **comporte comme** une **variable locale** de la fonction.
- Le **paramètre formel x** de la fonction **n'a rien avoir avec** le **paramètre effectif x**, sauf que **la valeur de celle-ci est utilisée pour l'initialiser**.
- Même chose pour y.
- C'est le **passage d'arguments par valeur**.
- **Règle** :

  Une **fonction ne peut pas modifier** un **argument** si ce dernier est **passé par valeur**.

Passage des arguments par adresse

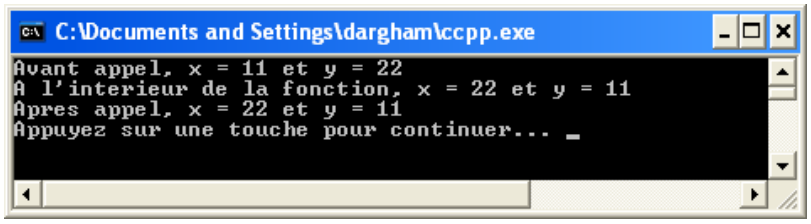
- **Exemple** :

```
#include <iostream>
using namespace std;
void permuter(int * x, int * y)
{
    int temp;
    temp = *x;
    *x = *y;
    *y = temp;
    cout << "A l'interieur de la fonction,  x = ";
    cout << *x << " et y = " << *y << endl;
}
```

Passage des arguments par adresse

```
int main()
{
   int x = 11, y = 22;

   cout << "Avant appel, x = " << x << " et y = " << y;
   cout << endl;
   permuter(&x, &y);
   cout << "Apres appel, x = " << x << " et y = " << y;
   cout << endl;
   system("pause");
   return 0;
}
```

Passage des arguments par adresse



Passage des arguments par adresse

- **Avant l'appel**, les **paramètres effectifs** sont x = 11 et y = 22.
- **Lors de l'appel de la fonction** « permuter » :
  - Le **paramètre formel** x est **initialisé avec l'adresse du paramètre effectif** x.
  - Le **paramètre formel** y est **initialisé avec l'adresse du paramètre effectif** y.
- **À l'intérieur de la fonction** « permuter » :
  - Les **paramètres formels** x et y **ont été bien échangés**.
- **Après l'appel**, les **paramètres effectifs** sont x = 22 et y = 11 : ils **ont été bien modifiés**.

Passage des arguments par adresse

- Comme le **paramètre formel x** de la fonction est un **pointeur**, sa **valeur sera toujours l'adresse du paramètre effectif** x (notez **&x**, lors de l'appel de la fonction).
- **Toute modification du paramètre formel affectera le paramètre effectif**.
- C'est le **passage d'arguments par adresse**.
- **Règle** :

  Une **fonction peut modifier** un **argument** si ce dernier est **passé par adresse**.

Passage des arguments par référence

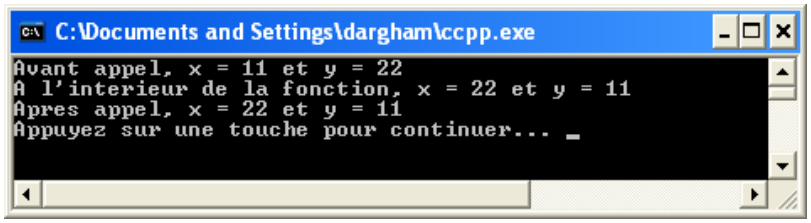
- **Exemple** :

```
#include <iostream>
using namespace std;
void permuter(int & x, int & y)
{
   int temp;
   temp = x;
   x = y;
   y = temp;
   cout << "A l'interieur de la fonction,  x = ";
   cout << x << " et y = " << y << endl;
}
```

Passage des arguments par référence

```
int main()
{
   int x = 11, y = 22;

   cout << "Avant appel, x = " << x << " et y = " << y;
   cout << endl;
   permuter(x, y);
   cout << "Apres appel, x = " << x << " et y = " << y;
   cout << endl;
   system("pause");
   return 0;
}
```

Passage des arguments par référence



Passage des arguments par référence

- **Avant l'appel**, les **paramètres effectifs** sont x = 11 et y = 22.
- **Lors de l'appel de la fonction** « permuter » :
  - Le **paramètre formel** x est **initialisé avec une référence sur le paramètre effectif** x.
  - Le **paramètre formel** y est **initialisé avec une référence du paramètre effectif** y.
- **À l'intérieur de la fonction** « permuter » :
  - Les **paramètres formels** x et y **ont été bien échangés**.
- **Après l'appel**, les **paramètres effectifs** sont x = 22 et y = 11 : ils **ont été bien modifiés**.

Passage des arguments par référence

- Comme le **paramètre formel x** de la fonction est une **référence**, sa **valeur sera toujours l'adresse du paramètre effectif** x.
- **Toute modification du paramètre formel affectera le paramètre effectif**.
- C'est le **passage d'arguments par référence**.
- **Règle** :

  Une **fonction peut modifier** un **argument** si ce dernier est **passé par référence**.

Cas des tableaux

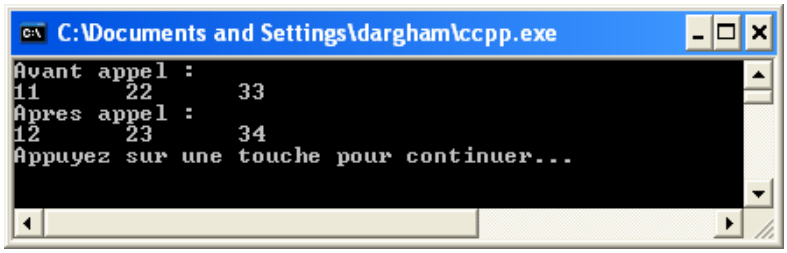
- **Exemple** :

```
#include <iostream>
using namespace std;
void ajouter_un(int t[ ], int n)
{
   int i;

   for(i = 0; i < n; i++)
      t[i]++;
}
```

Cas des tableaux

```
int main()
{
   int a[3] = {11, 22, 33};

   cout << "Avant appel : " << endl;
   for(int i = 0; i < 3; i++)
      cout << a[i] << "\t";
   ajouter_un(a, 3);
   cout << "\nApres appel : " << endl;
   for(int i = 0; i < 3; i++)
      cout << a[i] << "\t";
   cout << endl;
   system("pause");
   return 0;
}
```

Cas des tableaux



Cas des tableaux

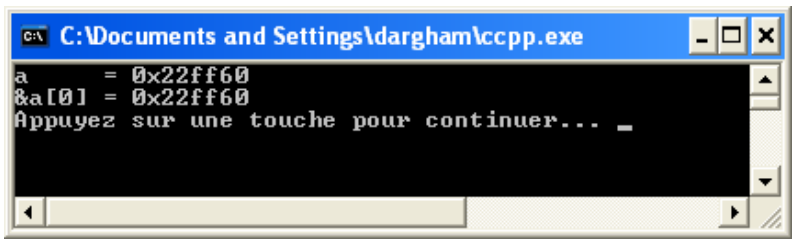
- Nous remarquons que **le tableau passé a été bien modifié par la fonction**.
- Cela est logique, puisque **le nom d'un tableau est un pointeur** (**sur le premier élément du tableau**).
- En fait, pour tout tableau « `tab` » :

```
tab ≡ &tab[0]
```

Cas des tableaux

```
#include <iostream>
using namespace std;
int main()
{
   int a[3] = {11, 22, 33};

   cout << "a      = " << a << endl;
   cout << "&a[0] = " << &a[0] << endl;
   system("pause");
   return 0;
}
```

Cas des tableaux



Cas des tableaux

- Nous remarquons que **le tableau passé a été bien modifié par la fonction**.
- Cela est logique, puisque **le nom d'un tableau est un pointeur** (**sur le premier élément du tableau**).
- En fait, pour tout tableau « `tab` » :

```
tab ≡ &tab[0]
```

Cas des tableaux

- Pour **interdire une fonction de modifier son argument qui est un tableau**, on **fait précédé le type du tableau par** le **mot-clé const** :

```
void f(const int t[ ], int n)
{
   /* code de la fonction */
   t[i]++;  /* erreur */
}
```

Résumé du passage d'arguments

Fonction	Comportement
`f(int n)`	Passage **par valeur** : f ne peut pas modifier n
`f(int *n)`	Passage **par adresse** : f peut modifier n
`f(int &n)`	Passage **par référence** : f peut modifier n
`f(int t[])`	Passage **d'un tableau** : f peut modifier t
`f(const int t[])`	Passage **d'un tableau protégé** : f ne peut pas modifier t