Chapitre 9

Passage de paramètres

1. Référence

- Référence est un alias à un espace mémoire.
- Syntaxe: Type& var = valeur;

```
#include <iostream>

using namespace std ;

int main() {
     int i=0,j=20;
     int& valeur =i;
     cout << "i: " << i << " valeur: " <<valeur << endl;
     i = 50;
     cout << "i: " << i << " valeur: " <<valeur << endl;
     valeur = j;
     cout << "i: " << i << " valeur: " <<valeur << endl;
     return 0;
}
```

- En sortie:

```
i: 0 valeur: 0
i: 50 valeur: 50
i: 20 valeur: 20
```

- Les variables de référence doivent être initialisées au moment de leurs déclarations.

```
#include <iostream>

using namespace std ;

int main() {

     int i=0;

     int& valeur;     // Erreur valeur a été déclarée comme référence mais n'a
                      // pas été initialisée. La manière correcte est
                      //  int& valeur = i;

     cout << valeur << "  " << i << endl;
     return 0;
}
```


- Ne peuvent être réaffectées (réassignées) comme alias à d'autres variables.

```
#include <iostream>

using namespace std ;

int main() {

     int i=0, j=20;
     int& valeur = i;


     valeur = j;




     cout << valeur << "  " \
      << i << " " << j << endl;

     return 0;
}
```

La valeur a été déclarée comme référence alias à la variable i

Cette valeur n'a pas été réaffectée à j, elle est toujours alias de i . Cette ligne sert uniquement à affecter 20 (donc le contenu de j) à valeur (donc i)

\ permet de dire au compilateur que la ligne qui suit fait partie de l'instruction

- Une fois que la référence est déclarée alias d'une autre variable, toutes les opérations que l'on croit effectuer sur l'alias (c'est à dire la référence) sont en fait exécutées sur la variable d'origine.

```
#include <iostream>

using namespace std ;

int main() {

     int i=0;
     int& valeur = i;
     valeur++;
     cout << i << endl ;

     return 0;
}
```

valeur = i =0
valeur = 1
affiche: 1

- **Danger …**

```
// bugref.cpp

#include <iostream>

using namespace std ;

int main() {
     double i=0;
     int& valeur = i;
     valeur = 250;
     cout << i << endl ;
     return 0;
}
```

valeur = i =0
valeur = 250
affiche: 0

- Le compilateur compile le programme sans erreurs et génère les warnings suivants:

```
bugref.cpp: In function `int main()':
bugref.cpp:5: warning: converting to `int' from `double'
bugref.cpp:5: warning: initializing non-const `int &' with
     `double' will use a temporary
```

- Le compilateur a créé une variable temporaire (signalée dans le second warning), comme suit:

`double i=0;` `int temp = i;`	**conversion double vers int et initialisation de temp avec la valeur de i**
`int& valeur = temp;`	**valeur est une référence à temp (mais pas i)**

- Modifier **`valeur`** revient à modifier **`temp`** mais pas **`i`** car il n'y a aucun lien entre **`i`** et **`temp`**.
- De ce fait, faire attention de déclarer le type de la référence identique à la variable référencée, pour éviter des surprises lors de l'exécution de votre programme.

Utilité:
- En général, pour éviter d'utiliser les pointeurs.
- En particulier, lors des passages de paramètres (des classes, des structures) afin d'éliminer la surcharge causée par la copie de grandes quantités de données.

2. Passage de paramètres

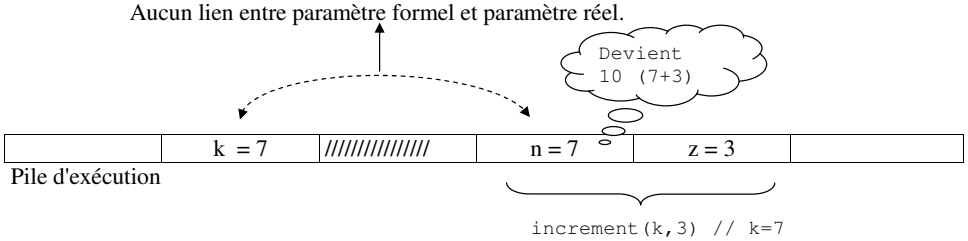
- Pour rappel, il y a 2 types de paramètres:
  - paramètres réels: dans l'appel de fonction
  - paramètres formels: dans l'entête de la fonction
- En C++, 3 types de passage de paramètres:
  - par valeur (existe aussi dans les langages C et Java[1])
  - par adresse (existe aussi dans le langage C)
  - par référence (uniquement dans les langages C++ et Java[1])

[1] Java : passage par valeur pour les primitifs et par référence pour les objets.

2.1. Passage de paramètres par valeurs

```
#include <iostream>
using namespace std ;
void increment(int n,int z) {
     n=n+z;
}
int main() {
     int k=7;
     increment(k,3);
     cout << k << endl; // affiche 7 mais pas 10 (i.e. 7+3)
     return 0;
}
```

Aucun lien entre paramètre formel et paramètre réel.



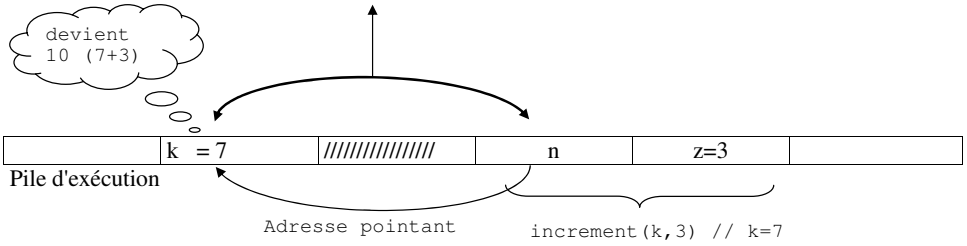
Paramètre formel **`n`** est initialisé avec la valeur du paramètre réel **`k=7`**.

2.2. Passage de paramètres par adresses

- Pour modifier le paramètre réel, on passe son adresse plutôt que sa valeur.

```
#include <iostream>
using namespace std ;
void increment(int *n,int z) {
     *n= *n + z;
}

int main() {
     int k=7;
     increment(&k,3);
     cout << k << endl; // affiche 10 car c'est l'adresse qui a été passée.
     return 0;
}
```

- Il y a un lien entre paramètre formel et paramètre réel.



- Paramètre formel **`n`** est initialisé avec le contenu de l'adresse pointant sur **`k`**.
- Il y a un lien potentiel entre les paramètres formels de la fonction et les paramètres réels.
- La gestion des adresses est effectuée par le programmeur (lourde et difficile à lire).

2.3. Passage de paramètres par référence

- En C++ le compilateur peut se charger lui même de la gestion des adresses.
- Le paramètre formel est un alias de l'emplacement mémoire du paramètre réel.

```
#include <iostream>

using namespace std;

void increment(int& n,int z) {
n= n + z;
}

int main() {
     int k=7;
     increment(k,3);
     cout << k << endl;  // affiche 10 car c'est l'adresse qui a été passée.
     return 0;
}
```

2.4 Exemple du swap

- Que va afficher le programme suivant sur la sortie standard? Réécrire le code suivant dans les 2 cas suivants:
  - `-1- void swap(int a,int& b){…}`
  - `-2- void swap(int& a,int& b){…}`

```
#include <iostream>

using namespace std ;

void swap(int a,int b){
     int c;
     c=a;a=b;b=c;
}

int main() {

     int i=4,j=7;

     swap(i,j);

     cout << i << " " << j << endl;

     return 0;
}
```

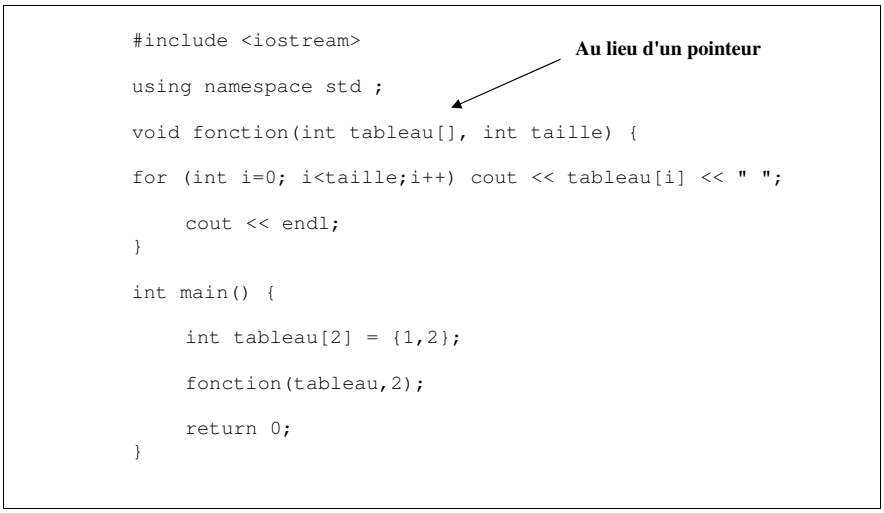

3. Passage d'un tableau

- 2 configurations possibles:

3.1. Par pointeur

```
#include <iostream>

using namespace std;

void fonction(int* tableau, int taille) {

     for (int i=0; i<taille;i++) cout << tableau[i] << " ";

     cout << endl;
}

int main() {

     int tableau[2] = {1,2};

     fonction(tableau,2);

     return 0;
}
```

3.2. Par semi-référence



4. Quand utiliser une référence

(voir aussi P. Prados @ http://perso.club-internet.fr/pprados/Langage/CPP/ref/ref.html)

4.1. Comme attribut

- Revoir le paragraphe sur les références.

4.2. Comme un paramètre

- Une question difficile à trancher :
  - Si le paramètre ne doit être consulté que par la méthode, il faut recevoir une référence (assez souvent constante).
  - Si le paramètre reçu par la méthode peut être absent, i.e. l'utilisateur peut envoyer la valeur NULL, il faut recevoir dans ce cas un pointeur:

```
void f(const TEST& x){ // x est une référence
(if (&x == NULL) … // À ne pas faire … référence ne peut pas être NULL.
}
```

- Utilisez la surcharge …

```
void f(const TEST& x){ // traite le cas d'un passage par référence
     // …
}
void f(const void* x){ // traite le cas d'un pointeur NULL

}
```

- Essayez de cogiter sur un exemple complet.

4.3. En retour d'une méthode (ou fonction)

```
#include <iostream>

using namespace std ;
int& fonction(int &a) {
     a += 10;
     return a;
}
int main() {
     int val,test = 20;
     val = fonction(test);
     cout << "val: " << val << " " \
            << "test: " << test << endl; // affiche: val: 30 test: 30
     fonction(test)  = 250;
     cout << test << endl; // affiche: 250
     return 0;
}
```

Danger …

```
int& exemple() {

     int i=10; // i est une variable locale de la fonction exemple
               // durée de vie entre {}

     return i; // exemple va retourner un alias sur une variable
               // qui va cesser d'exister!
}
```