université de TOURS

Développement objet C++

Dominique H. Li

dominique.li@univ-tours.fr

Licence Informatique - Blois

Université de Tours

université de TOURS

```cpp
#include <iostream>

int main() {
    std::cout << "Hello, World !" << std::endl;
    return 0;
}
```

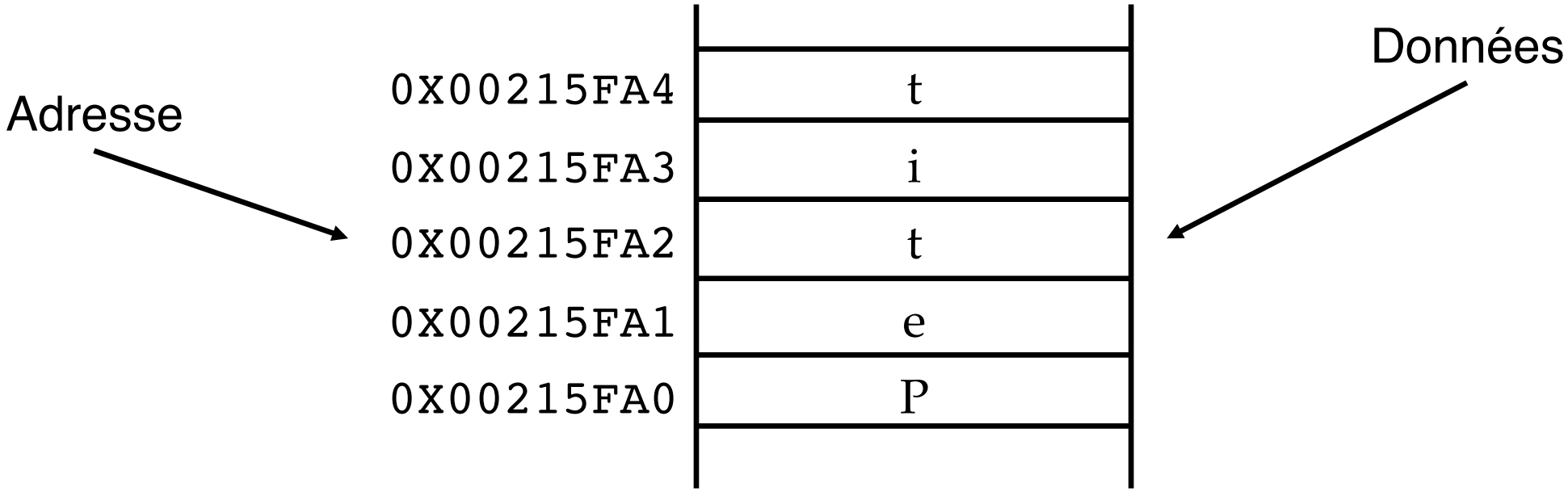
Utilisation et gestion de la mémoire

Développement objet C++

Utilisation de la mémoire

- Une *adresse mémoire* est un identifiant qui désigne une zone particulière de la mémoire physique où des données peuvent être lues et stockées
  - Temporaire (mémoire vive, RAM)
  - Durable (mémoire non volatile, disque dur)
- Une *variable* (un stockage temporaire de données) implique l'accès à la mémoire vive
- Un *tableau* est un ensemble d'éléments de même type stockés consécutivement en mémoire

Schéma de tableaux



Les chaînes de caractères de type tableau

- En C++ (initialement en C), une *chaîne de caractères* est effectivement un *tableau de caractères* de type **char**
- La fin de chaîne se détermine par le *caractère nul* **'\0'**
- Les fonctions définies dans l'en-tête **<cstring>** sont indispensables à toute manipulation des chaînes de caractères classiques

Les définitions de tableaux

```
double note[5];
int entier[2][5];
char ligne[80];

int nombre[] = {3,2,7,6,8};                    // sizeof(nombre) == 20
int groupe[5][2] = {{0,1}, {2,3}, {4,5}, {6,7}, {8,9}};
char alphabet[26] = {'A', 'B', 'C’};

char mot1[5] = "Hello";                        // ERREUR !!!
char mot2[6] = "Hello";
char mot[] = "Hello";                          // sizeof(mot) == 6
```

Accès aux éléments de tableau

- Chaque élément d'un tableau est accessible grâce à un indice entre crochets **[]**
- Les éléments d'un tableau de longueur *n* sont numérotés *de 0 à n-1*

Exemple : tableau

```
int chiffre[10];
int matrice[10][8];

for (int i = 0; i < 10; i++) {
        chiffre[i] = i;
        for (int j = 0; j < 8; j++) {
                matrice[i][j] = i * j;

        }
}
```

Exemple : utilisation de tableaux

```
char mot[5];

for (int i = 0; i < 5; i++) {
        mot[i] = 'A' + i;
}
std::cout << mot << std::endl;          // ATTENTION !!!

std::cout << "Entrer le mot : ";        // Entrer 3 lettres
std::cin >> mot;                        // ATTENTION !!!
std::cout << mot << std::endl;

std::cout << "Entrer le mot : ";        // Entrer 8 lettres
std::cin >> mot;                        // ATTENTION !!!
std::cout << mot << std::endl;
```

Les paramètres tableaux

- Concernant le passage des tableaux dans une fonction, c'est toujours l'*adresse du tableau* qui est transmis
- La taille du tableau ne sera jamais connue dans la fonction appelée
  - Gestion par des variables globales statiques
  - Gestion par une classe de tableau
- À l'intérieur de la fonction, toute modification d'un élément du tableau transmis agira sur l'original

Utilisation de paramètres tableaux

```
void init(int t[], int n) {                    // n : la taille du tableau
        for (int i = 0; i < n; ++i) {
                t[i] = 0;
        }
}

int main() {
        int tab[10];                           // La taille du tableau est 10
        init(tab, 10);                         // 10 est passé dans la fonction
        init(tab, 8);                          // Logiquement correcte
        init(tab, 11);                         // ERREUR !!!
}
```

Les pointeurs

- Un *pointeur* est en effet une adresse mémoire
- Une *variable de type pointeur* est une variable contenant l'adresse d'un objet
- L'adresse de l'emplacement mémoire d'un objet s'obtient grâce à l'*opérateur d'adressage* **&** suivi du nom de l'objet
- L'*opérateur d'indirection* * appliqué à un pointeur permet d'obtenir le contenu de l'adresse (donc l'objet pointé lui-même)

Déclaration de pointeurs

- Les pointeurs sont typés et sont liés au type de l'objet sur lequel ils pointent
- Les pointeurs et les constantes
  - Un pointeur peut être une constante
  - Un pointeur peut pointer vers une constante

Exemple : pointeurs

```
int *p1;                    // Un pointeur de type entier
double * p2;                // Un pointeur de type réel
char* p3;                   // Un pointeur de type caractère
std::string* p4, p5;        // Quel est le type de p5 ?


int *const p5 = &x;         // p5 est une constante
int const *p6 = n;          // n est une constante


int *****p5;                // Quel est le type de p5 ?
```

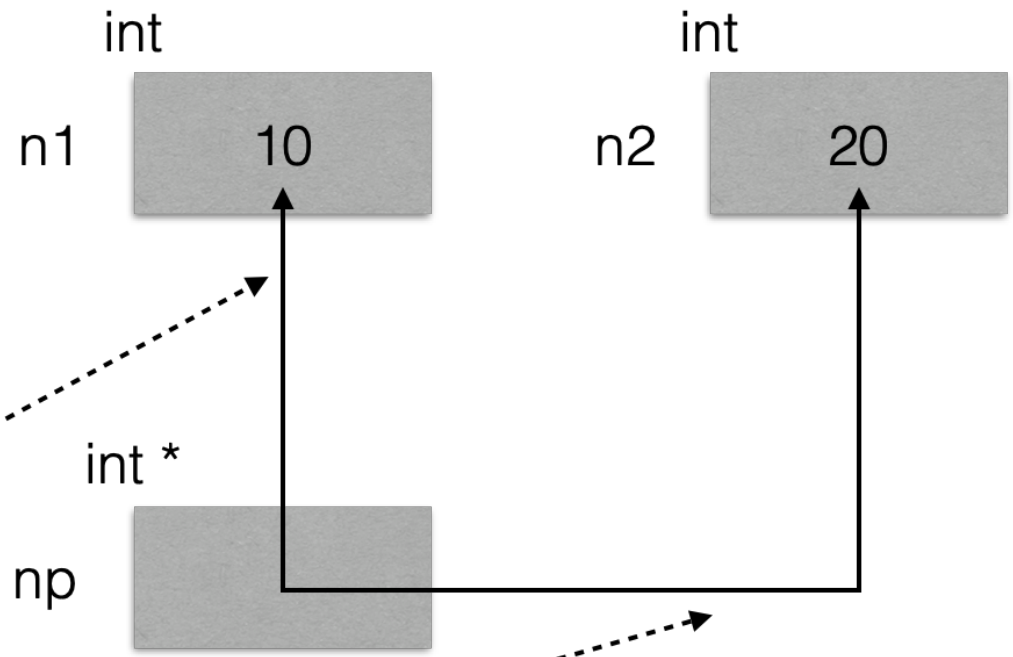
Affectation de pointeurs

```
int n1 = 10;
int n2 = 20;
```

```
int *np;
```

```
np = &n1;
```

```
np = &n2;
```



Exemple : utilisation de pointeurs

```
int n1, n2;          // Déclaration de deux entiers
int *np;             // Déclaration d'un pointeur de type int

n1 = 10;             // n1 contient la valeur 10
np = &n1;            // np contient l'adresse de n1

*np = 20;            // n1 devient égal à 20
n2 = *np;            // Affecter à n2 le contenu de l'objet pointé par np

n2 = *np++;          // Quelle est la valeur de n1 ?
n1 = *++np;          // Quelle est la valeur de n1 ?
np = 0;              // np ne pointe sur rien maintenant
```

Manipulation de pointeurs

- La valeur d'une variable de type pointeur est en effet un entier équivalent au type **unsigned long long int**
- Soit **ptr** un pointeur de type réel, l'expression **ptr + 1** représente l'adresse du réel suivant
- Il faut distinguer les expressions suivantes
  - **(*ptr)++** et **++(*ptr)**
  - ***(ptr++)** et ***(++ptr)**
  - ***ptr++**, **++*ptr** et ***++ptr**

Les pointeurs et les tableaux

- En C/C++, un *nom de tableau* est un *pointeur constant*
- Le nom d'un tableau représente en fait l'*adresse du premier élément* du tableau
- La manipulation des tableaux à l'aide des pointeurs est fréquente et plus rapide à l'exécution
  1. Déclarer un pointeur et lui donner l'adresse du premier élément du tableau
  2. Incrémenter l'adresse pour parcourir le tableau

Exemple : manipuler un tableaux par pointeur

```
int tab[10];
int *iptr;
iptr = tab;
```

```
for (int i = 0; i < 10; i++)        tab[i] = 0;                 // Classique

for (int i = 0; i < 10; i++)        *iptr++ = 0;

for (int i = 0; i < 10; i++)        *(tab + i) = 0;

for (int i = 0; i < 10; i++)        *(tab++) = 0;               // ERREUR !!!
```

Exemple : initialisation de tableaux par pointeur

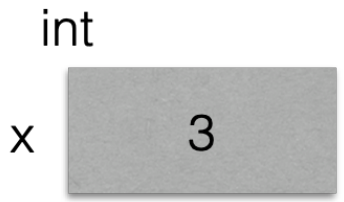
```
void init_par_tab(int t[], int n) {
        for (int i = 0; i < n; i++)
                t[i] = 0;
}

void init_par_ptr(int *t, int n) {
        for (int i = 0; i < n; i++, t++)
                *t = 0;
}

void init_ptr_tab(int *t, int n) {
        for (int i = 0; i < n; i++)
                t[i] = 0;
}
```

Les références

- En C++, une *référence* est un nom alternatif ou alias pour un objet
- Une *variable de type référence* doit obligatoirement être initiale lors de sa déclaration
  - Une référence ne peut être initialisée qu'une fois
  - Une référence désignera toujours le même objet
- Les références sont principalement utilisées dans la spécification des paramètres et des valeurs de retour de fonction

Déclaration et affectation de références

```
int x = 3;
```




```
int &r = x;
```


```
int y = 0;
```


```
r = y;
```

Les passages de paramètres

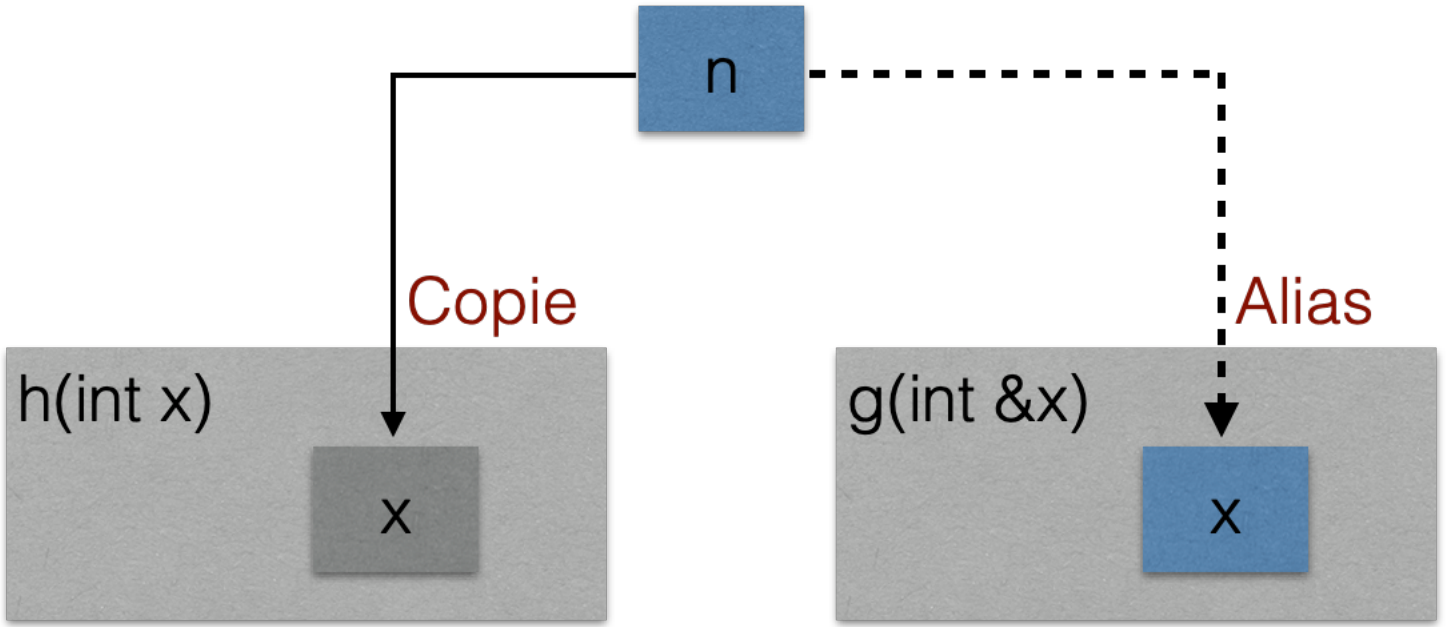
- En C++, il existe deux méthodes pour passer des variables en paramètre dans une fonction
  - Le passage de paramètres *par valeur*
  - Le passage de paramètres *par référence*
- Ces deux sortes de passages sont déterminés par la syntaxe utilisée lors de la définition de la liste de paramètres

Exemple : passages de paramètres

```
void pass_par_val(int x);          // Par valeur
void pass_par_ref(int &x);         // Par référence
void pass_par_ptr(int *x);         // Par valeur (de pointeur)
```

Par référence ou par valeur ?

```
h(int x) {
    // ...
}

g(int &x) {
    // ...
}

int n = 5;
h(n);
g(n);
```



Exemple : utilisation de paramètre valeur

```
void efg(std::string str) {
        str = "EFG";
}

int main() {
        std::string s = "ABC";
        efg(s);
        std::cout << s << std::endl;
}
```

Exemple : utilisation de paramètre référence

```
void efg(std::string &str) {
        str = "EFG";
}

int main() {
        std::string s = "ABC";
        efg(s);
        std::cout << s << std::endl;
}
```

Exemple : utilisation de paramètre pointeur

```
void efg(std::string *str) {
        *str = "EFG";
}


int main() {
        std::string s = "ABC";
        efg(&s);

        std::cout << s << std::endl;
}
```

Allocation statique de la mémoire

- Les objets déclarés dans un programme (*statiques*) se voient attribués un emplacement en mémoire pour y stocker leurs valeurs
  - Ces objets occupent la partie de mémoire appelée *pile* (*stack* en anglais)
  - La taille de ces objets doit être *connue à la compilation*, c'est le compilateur qui détermine leur emplacement mémoire
- Les *objets statiques* continuent à occuper de l'espace-mémoire jusqu'à ce qu'ils perdent leur validité

Allocation dynamique de la mémoire

- Il est important de pouvoir créer des objets au cours de l'exécution d'un programme (*dynamiques*) et de pouvoir détruire ces objets lorsqu'ils sont devenus inutiles
  - Ces objets occupent la partie de mémoire appelée *tas* (*heap* en anglais)
  - La taille de ces objets est *inconnue à la compilation*
- La durée de vie des *objets dynamiques* est gérée par le programmeur

Création d'objets dynamiques

- En C++, l'opérateur **new** permet de créer en cours d'exécution d'un programme un objet du type précisé
  - En cas de succès, **new** retourne un pointeur sur l'objet créé
  - En cas d'échec, **new** retourne le *pointeur nul* de valeur **0** ou **nullptr** (en C++11)
- L'opérateur **new** calcule automatiquement la taille de l'objet
- Par défaut, la zone de mémoire allouée *ne sera pas initialisée*

Exemple : utilisation de l'opérateur new

```
int *iptr1, *iptr2, *iptr3;

iptr1 = new int;
*iptr1 = 10;

iptr2 = new int(10);                    // Un entier de la valeur 10
iptr3 = new int[10];                    // Un tableau de 10 entiers

char *mot;
mot = new char[20];                     // Un tableau de 20 caractères

std::string *str = new std::string("coucou");
*str = "bonjour";
```

Destruction d'objets dynamiques

- Un objet dynamique existe depuis son point de création jusqu'à sa destruction explicite
- En C++, l'opérateur **delete** permet de détruite un objet créé dynamiquement
- L'opérateur **delete[]** est utilisé pour détruire les tableaux dynamiques
- **NE JAMAIS oublier de détruire des objets dynamiques !!!**

Exemple : utilisation de l'opérateur delete

```
int *iptr1, *iptr2;

iptr1 = new int;
*iptr1 = 10;
iptr2 = new int[10];                    // Un tableau de 10 entiers

char *mot;
mot = new char[20];                     // Un tableau de 20 caractères

delete iptr1;
delete[] iptr2;
delete[] mot;
```

Exemple : vérification de l'espace alloué

```
#include <iostream>
#include <new>

void err_mem(void) {
        std::cerr << "error: no enough memory!" << std::endl;
        exit(1);
}

int main() {
        std::set_new_handler(&err_mem);
        char *ptr = new char[1024 * 1024 * 1024 * 1024];
        delete[] ptr;
}
```

Exemple : un tableau dynamique

```
int main() {
        int *tableau;
        unsigned int taille;
        std::cout << "Entrer la taille du tableau : " << std::ends;
        std::cin >> taille;
        tableau = new int[taille];
        if (!tableau) {
                std::cerr << "erreur !" << std::endl;
        } else {
                for (int i = 0; i < taille; i++)
                        tableau[i] = i * i;
                delete[] tableau;
        }

}
```

Exemple : remplissage et copie de tableaux

```
#include <cstring>

#define SIZE 100

int main() {
        int *t1 = new int[SIZE];
        std::memset(t1, 0, SIZE * sizeof(int));          // Remplir 0 à t1
        int *t2 = new int[SIZE];
        std::memcpy(t2, t1, SIZE * sizeof(int));         // Copier t1 à t2
}
```