COURS C++ ISI 2025

Chapitre 2 : Les Pointeurs

Stéphanos Langate

Stephzoux@gmail.com

Chapitre 2 : Les pointeurs

Un pointeur est une variable qui contient l'adresse mémoire d'une autre variable. Une variable classique contient une valeur, tandis qu'un pointeur contient l'adresse où cette valeur est stockée.

I. Déclaration d'un pointeur

Syntaxe : type* nom_pointeur;

Exemple : int* ptr; déclare un pointeur vers un entier

go.cpp

```cpp
#include <iostream>
using namespace std;

int main() {
    int a = 10;
    int* p = &a;
    cout << "Valeur de a : " << a << endl;
    cout << "Adresse de a (&a) : " << &a << endl;
    cout << "Valeur du pointeur p : " << p << endl;
    cout << "Valeur pointee (*p) : " << *p << endl;

    return 0;
}
```

II. Modifier la valeur via un pointeur

go.cpp

```cpp
#include <iostream>
using namespace std;

int main() {
    int dk = 5;
    int* st = &dk;

    *st = 20;  // on modifie la valeur pointée

    cout << "Nouvelle valeur de a : " << dk << endl; // affiche 20
    return 0;
}
```

Comme st pointe vers dk, modifier *st revient à modifier dk

III. Pointeurs et tableaux

Un tableau est étroitement lié aux pointeurs :

Start here × go.cpp ×

```cpp
#include <iostream>
using namespace std;

int main() {
    int T[3] = {10, 20, 30};
    int* p = T;   // équivaut à &T[0]

    cout << *p << endl;       // 10
    cout << *(p + 1) << endl; // 20
    cout << *(p + 2) << endl; // 30

}
```

Le nom d'un tableau (T) agit comme un pointeur vers son premier élément.

IV. Pointeurs et fonctions

Les pointeurs permettent de passer des arguments par adresse à une fonction.

Start here × go.cpp ×

```cpp
#include <iostream>
using namespace std;

void increment(int* p) {
    (*p)++;
}

int main() {
    int x = 10;
    increment(&x);
    cout << x; // affiche 11
    return 0;
}
```

V. Pointeur vers pointeur

On peut avoir des pointeurs vers d'autres pointeurs :

Start here × go.cpp ×

```cpp
#include <iostream>
using namespace std;

void increment(int* p) {
    (*p)++;
}

int main() {
    int a = 5;
    int* p = &a;
    int** pp = &p;

    cout << **pp; // 5

}
```

VI. Pointeur nul et pointeur non initialisé

int* p = nullptr; pointeur vide

int* p ; pointeur non initialisé, pas recommandable

Toujours initialiser un pointeur :

❖ Allocation dynamique

L'allocation dynamique consiste à réserver de la mémoire pendant l'exécution du programme (et non à la compilation).

Elle permet de créer des variables, tableaux ou structures dont la taille n'est pas connue à l'avance.

En C++, cela se fait avec les opérateurs :

new => pour allouer

delete => pour libérer

Start here × go.cpp ×

```cpp
#include <iostream>
using namespace std;

int main() {
    int* p = new int;    // allocation d'un entier
    *p = 10;             // affectation
    cout << "Valeur : " << *p << endl;
    delete p; // libération de la mémoire
    return 0;
}
```

new int ; réserve de la mémoire pour un entier

*p = 10 ; stocke la valeur 10 à cette adresse

delete p ; libère la mémoire (important pour éviter les fuites mémoire)

Start here | *go.cpp

```cpp
#include <iostream>
using namespace std;
int main() {
    int n;
    cout << "Entrez la taille du tableau : ";
    cin >> n;
    int* tab = new int[n];
    for (int i = 0; i < n; i++) {
        cout << "tab[" << i << "] = ";
        cin >> tab[i];
    }

    cout << "\nContenu du tableau : ";
    for (int i = 0; i < n; i++)
        cout << tab[i] << " ";

    delete[] tab;
    return 0;
}
```

new type[n] ; alloue un tableau dynamique

delete[] ptr ; libère la mémoire du tableau