7-Semaine 7 - pointeurs et références

Résumé du cours

A pointer is a variable that contains the address of a variable. Pointers are much used in C, partly,because they are sometimes the only way to express acomputation, and partly because they usually lead to more compact and efficient code than can be obtained in other ways. Pointers and arrays are closely related.

- http://gitlab.krital.ovh:49892/komo/cpp/tree/master/btssn/1er/pointeurs [http://gitlab.krital.ovh:49892/komo/cpp/tree/master/btssn/1er/pointeurs]

Les pointeurs sont un type un peu particulier : ils permettent d'indiquer (« de pointer sur ») d'autres éléments du programme (variables ou fonctions). Ils font partie intégrante du langage.

- Déclaration : `type* nom;`
- Déclaration et initialisation : `type* nom(adresse);`
- Adresse d'une variable : `&variable`
- Accès au contenu d'une variable pointée par un pointeur :`*pointeur`
- Allocation mémoire :

```
pointeur = new type;
```

- ou avec initialisation :

```
pointeur = new type(valeur);
```

- Libération de la zone mémoire allouée :

```
delete pointeur;
```

Notions plus avancées

- Pointeur par le biais duquel la valeur ne peut être modifiée (pointeur « en lecture seulement ») :

```
const type* ptr;
```
ou
```
type const* ptr;
```

- Pointeur constant :

```
type* const ptr(adresse);
```

- Pointeur constant, en lecture seulement :

```
const type* const ptr(adresse);
```

- Pointeur sur une fonction :

```
function<type_retour(arguments…)> nom; // nécessite la bibliothèque <functional>
type_retour (*nom)(arguments…); // pointeur “à la C”
```

```
#include <iostream>

using namespace std;
int somme(int, int);
int soustraction(int, int);
int (*ptf)(int, int);

int main(int argc, char *argv[])
{
    ptf = somme;
    cout << "somme : " << (*ptf)(1, 2) << endl;

    ptf = soustraction;
    cout << "soustraction : " << (*ptf)(1, 2) << endl;


    return 0;
}

int somme(int v1, int v2){
    return v1 + v2;
}

int soustraction(int v1, int v2){
    return v1 - v2;
}
```

- Passer une fonction en argument à une fonction :

```
#include <iostream>

using namespace std;

void affiche(int (*f)(int, int), int, int);
int somme(int, int);
int soustraction(int, int);

int main(int argc, char *argv[])
{
    affiche(somme, 2, 3);
    affiche(soustraction, 2, 3);

    return 0;
}
```

```
void affiche(int (*f)(int a, int b), int a, int b){
    cout <<  f(a,b) << endl;
}

int somme(int v1, int v2){
    return v1 + v2;
}

int soustraction(int v1, int v2){
    return v1 - v2;
}
```

- [C++11] Pointeurs « intelligents » : `unique_ptr, shared_ptr` et `weak_ptr`.

En C++ existe aussi la notion de référence, à ne pas confondre avec la notion de pointeur : une référence est simplement un autre nom pour une variable ; elle n'est pas une indirection, ne peut pas être réaffectée (est toujours liée au même objet) et est toujours valide.

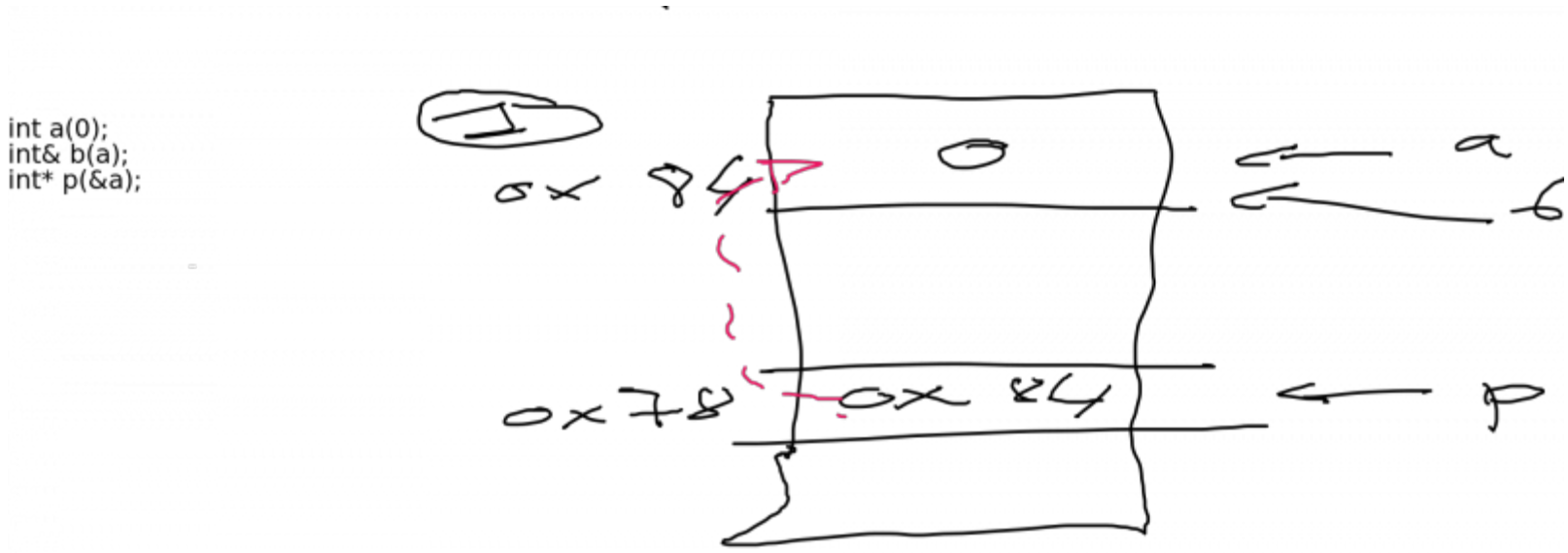
- Déclaration et initialisation (d'une référence) :

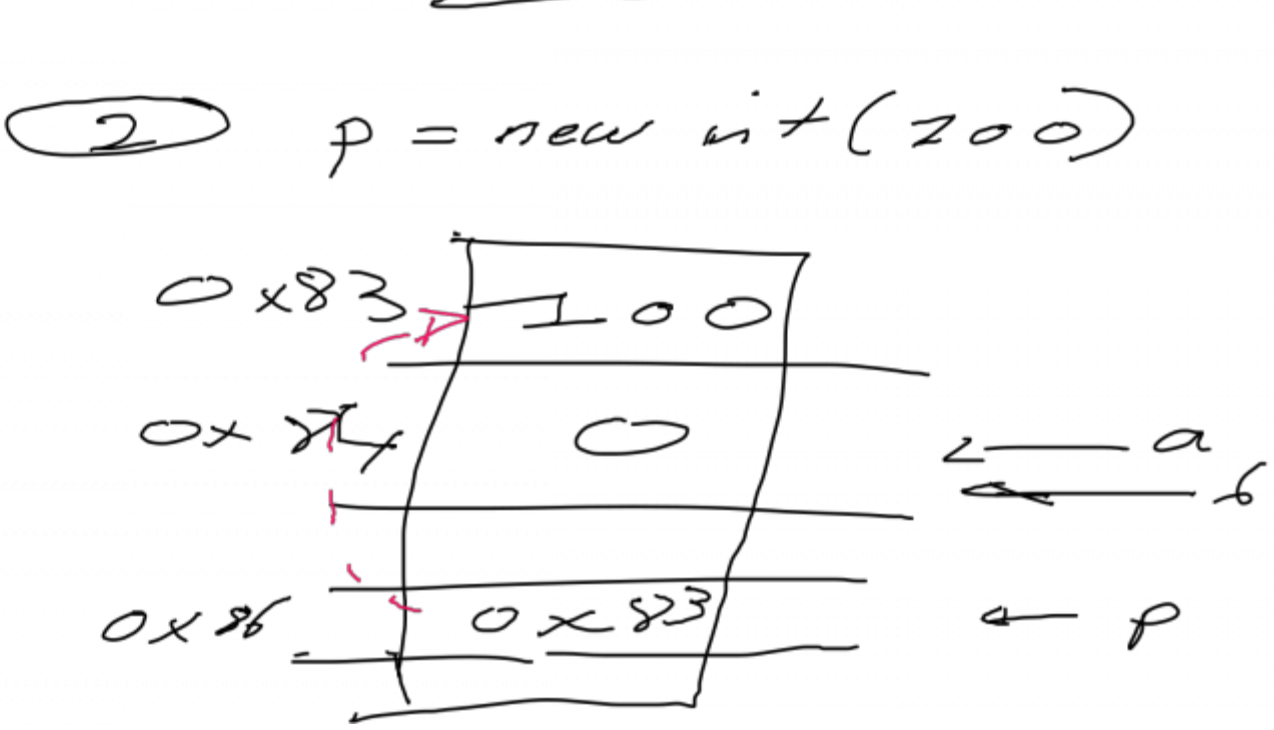
`type& référence(autre variable);`

Démonstration

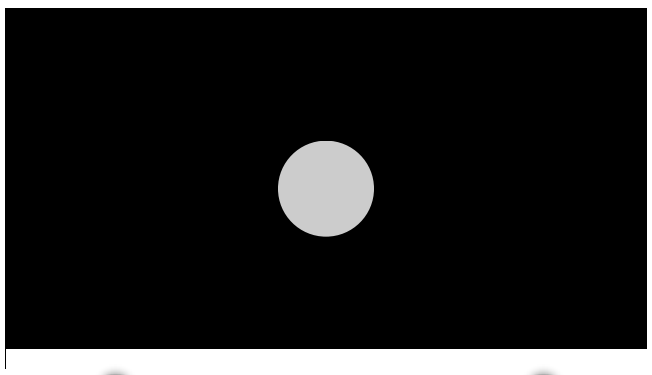
```
//============================================================================
// Name        : pointeurs.cpp
// Author      : komo
// Version     :
// Copyright   : Your copyright notice
// Description : Hello World in C++, Ansi-style
//============================================================================

#include <iostream>
using namespace std;

int func(int);
int rfunc(int&);
int pfunc(int*);

int main() {
        int a(0);
        int& b(a);
        int* p(&a);

cout << "a : " << a << endl;
cout << "b : " << b << endl;
cout << "p : " << p << endl;
cout << "*p : " << *p << endl;

cout << "--------------------------------" << endl;

/* 1 */
//a = 100;

/* 2 */
//b = 100;

/* 3 */
//*p = 100;

/* 4 */
p = new int(100);

/* 5 */
//int c = 100;
//b = c;

cout << "a : " << a << endl;
cout << "@a : " << &a << endl;
cout << "b : " << b << endl;
cout << "@b : " << &b << endl;
cout << "p : " << p << endl;
cout << "@p : " << &   p << endl;
cout << "*p : " << *p << endl;



        /*cout << "============= Passage par valeurs ===============" << endl;
        cout << "La valeur de a avant func : " << a << endl;
        cout << "La valeur de b avant func : " << b << endl;
        cout << "La valeur de a dans fun : " << func(a) << endl;
        cout << "La valeur de a après func : " << a << endl;
        cout << "La valeur de b après func : " << b << endl;


        cout << "============= Passage par référence ==============="  << endl;
        cout << "La valeur de a avant rfunc : " << a << endl;
        cout << "La valeur de b avant func : " << b << endl;
        cout << "La valeur de a dans rfun : " << rfunc(a) << endl;
        cout << "La valeur de a après rfunc : " << a << endl;
        cout << "La valeur de b après func : " << b << endl;


        cout << "============= Passage par @ ==============="  << endl;
        cout << "La valeur de a avant pfunc : " << a << endl;
        cout << "La valeur de b avant func : " << b << endl;
        cout << "La valeur de a dans pfun : " << pfunc(&a) << endl;
        cout << "La valeur de a après pfunc : " << a << endl;
        cout << "La valeur de b après func : " << b << endl;

        cout << "La valeur de *p après func : " << *p << endl;*/


        return 0;
}


int func(int a){
        a = 100;
```

```
        return a;
}

int rfunc(int& a){
        a = 200;
        return a;
}


int pfunc(int* a){
        *a = 300;
        return *a;
}
```





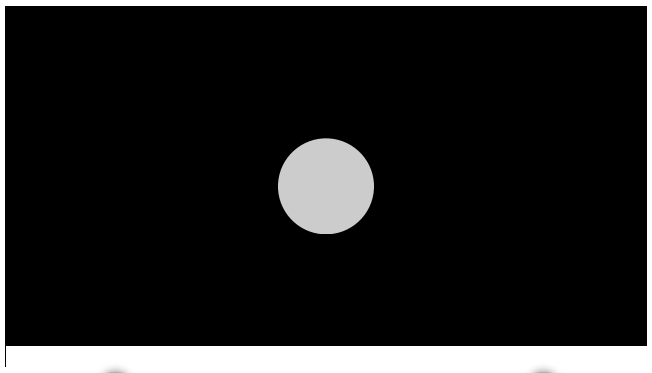
Cours



0:00 / 9:57

8 - 1 - Pointeurs et références - introduction [09-56]

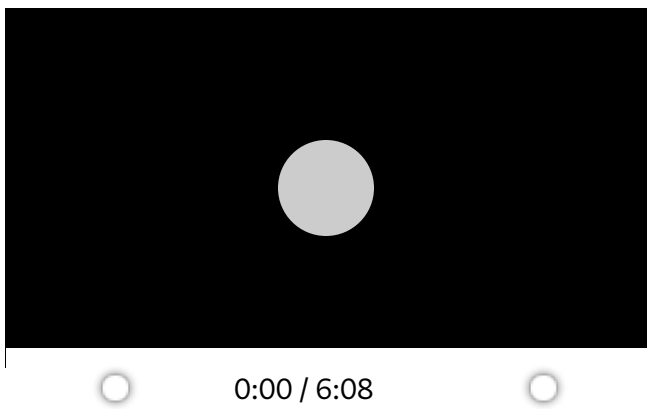
- Le cours en pdf



0:00 / 12:04

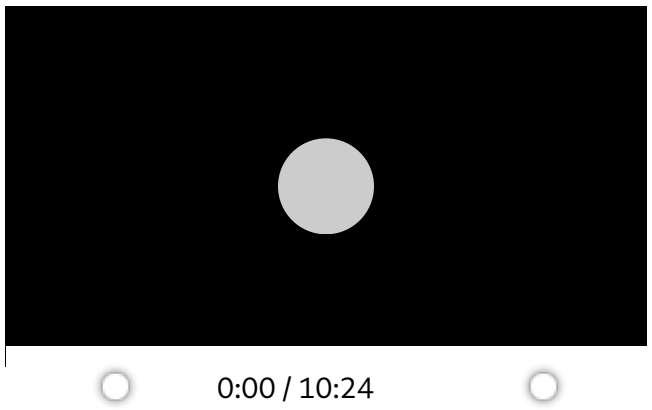
8 - 2 - Références [12-04]

- Le cours en pdf



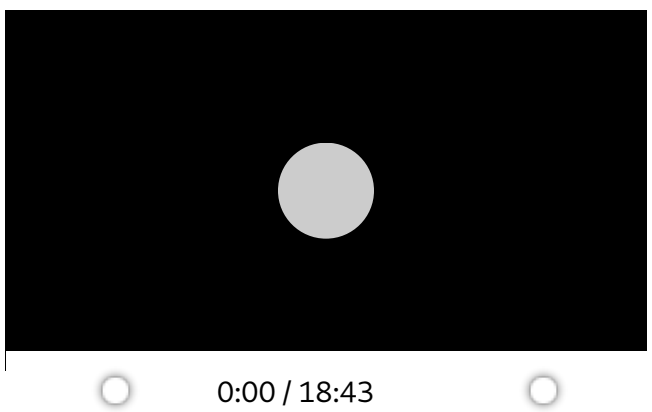
8 - 3 - Pointeurs - concept et analogie [06-07]

- Le cours en pdf



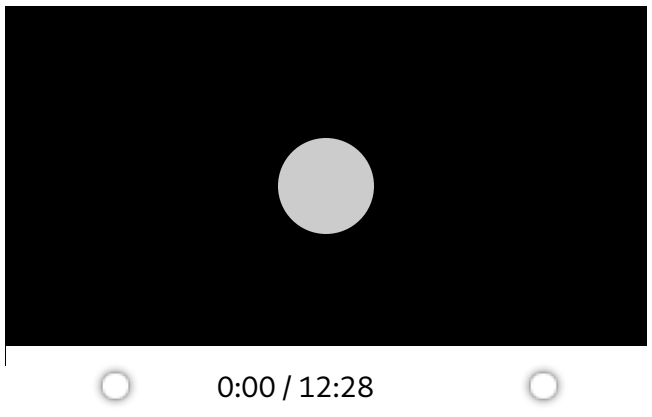
8 - 4 - Pointeurs - déclaration et opérateurs de base [10-24]

- Le cours en pdf



8 - 5 - Pointeurs - allocation dynamique [18-42]

- Le cours en pdf



8 - 6 - Pointeurs « intelligents » [12-28]

- Le cours en pdf

Tutoriel

Énoncé

Écrire une fonction qui ne renvoie aucune valeur et qui détermine la valeur maximale et la valeur minimale d'un tableau d'entiers (à un indice) de taille quelconque. On prévoira 4 arguments : le tableau, sa dimension, le maximum et le minimum. Pour chacun d'entre eux, on choisira le mode de transmission le plus approprié (par valeur ou par référence). Dans le cas où la transmission par référence est nécessaire, proposer deux solutions : l'une utilisant effectivement cette notion de référence, l'autre la « simulant » à l'aide de pointeurs.
Écrire un petit programme d'essai.

En C++, par défaut, les arguments sont transmis par valeur. Mais, dans le cas d'un tableau, cette valeur, de type pointeur, n'est rien d'autre que l'adresse du tableau.
Quant à la transmission par référence, elle n'a pas de signification dans ce cas. Nous n'avons donc aucun choix concernant le mode de transmission de notre tableau.

En ce qui concerne le nombre d'éléments du tableau, on peut indifféremment en transmettre l'adresse (sous forme d'un pointeur de type int *), ou la valeur ; ici, la seconde solution est la plus appropriée, puisque la fonction n'a pas besoin d'en modifier la valeur.

En revanche, en ce qui concerne le maximum et le minimum, ils ne peuvent pas être transmis par valeur, puisqu'ils doivent précisément être déterminés par la fonction. Il faut donc obligatoirement prévoir de passer :

- soit des références. L'en-tête de notre fonction se présentera ainsi :

```
void maxmin (int t[], int n, int & admax, int & admin)
```

- soit des pointeurs sur des entiers. L'en-tête de notre fonction se présentera ainsi :

```
void maxmin (int t[], int n, int * admax, int * admin)
```

L'algorithme de recherche de maximum et de minimum

- avec transmission par référence :

```
void maxmin (int t[], int n, int & admax, int & admin)
{
        int i;
        admax = t[1];
        admin = t[1];
        for (i=1; i<n; i++)
        { if (t[i] > admax) admax = t[i];
          if (t[i] < admin) admin = t[i]; }
}
```

- avec transmission par pointeurs :

```
void maxmin (int t[], int n, int * admax, int * admin)
{
        int i;
        *admax = t[1];
        *admin = t[1];
        for (i=1; i<n; i++)
        { if (t[i] > *admax) *admax = t[i];
          if (t[i] < *admin) *admin = t[i]; }
}
```

- Ici, si l'on souhaite éviter les « indirections » qui apparaissent systématiquement dans les instructions de comparaison, on peut travailler temporairement sur des variables locales à la fonction (nommées ici max et min). Cela nous conduit à une fonction de la forme suivante :

```
void maxmin (int t[], int n, int * admax, int * admin)
{
        int i, max, min;
        max = t[1];
        min = t[1];
        for (i=1; i<n; i++)
        { if (t[i] > max) max = t[i];
          if (t[i] < min) min = t[i]; }
        *admax = max;
        *admin = min;
}
```

- Voici un petit exemple de programme d'utilisation de la première fonction :

```
int main()
{
        void maxmin (int [], int, int &, int &);
        int t[8] = { 2, 5, 7, 2, 9, 3, 9, 4};
        int max, min;
        maxmin (t, 8, max, min);
        cout << "valeur maxi : " << max << "\n";
        cout << "valeur mini : " << min << "\n";
        return 0;
}
```

- Et voici le même exemple utilisant la seconde fonction :

```
int main()
{
        void maxmin (int [], int, int *, int *);
        int t[8] = { 2, 5, 7, 2, 9, 3, 9, 4};
        int max, min;
        maxmin (t, 8, &max, &min);
        cout << "valeur maxi : " << max << "\n";
        cout << "valeur mini : " << min << "\n";
        return 0;
}
```

Exercice

Un petit exercice simple pour manipuler les pointeurs comme sélecteurs d'objets.

Créez un nouveau fichier appelé `selecteur.cpp`.

Dans le `main()`, créez trois variables `valeur1` `valeur2` et `valeur3`, de type `double` que vous initialisez à des valeurs de votre choix.
Déclarez un pointeur `choix`, pointeur sur des double.

Demandez ensuite un nombre entre 1 et 3 à l'utilisateur et, en fonction de son choix, faites pointer choix sur `valeur1`, `valeur2` ou `valeur3`.
Pour finir affichez Vous avez choisi et la valeur choisie en utilisant le pointeur choix.
Remarque : il est évident que le but de cet exercice n'est que de manipuler les pointeurs. Hors de ces contraintes, la bonne solution pour faire la même chose serait bien sûr d'utiliser un tableau.

Devoir

- Énoncé