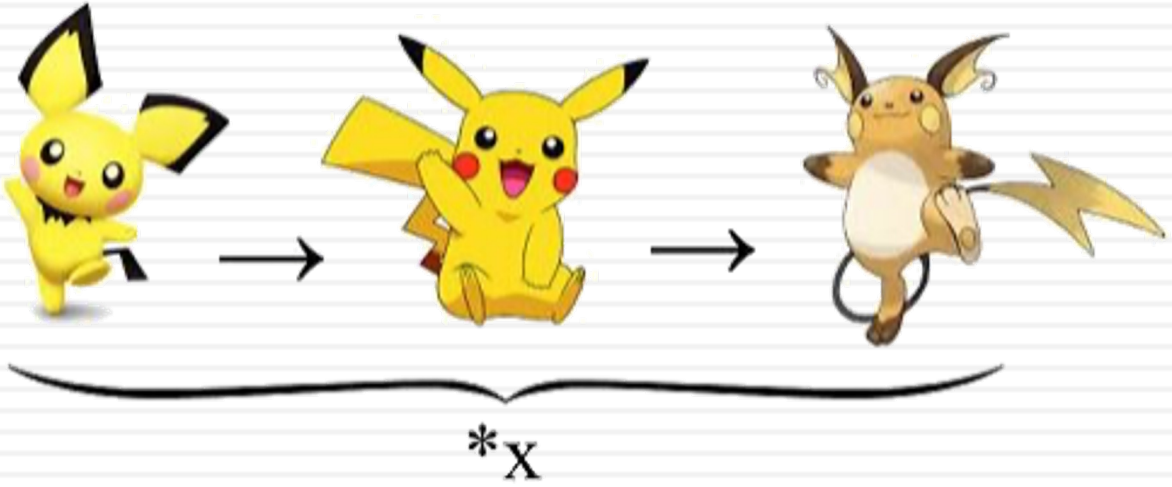
R1.01
DEV1

Les pointeurs ne manquent pas d'adresse



Etienne Carnovali

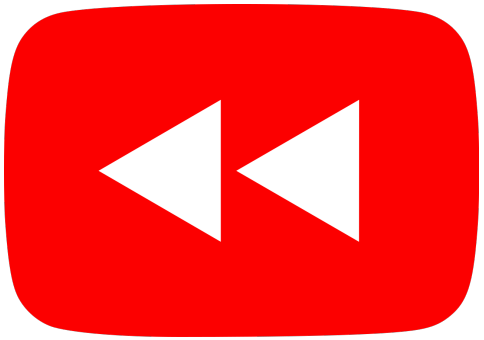
Jean-Michel Bohé

Marwa Hamdi

Ronan Champagnat

Au cours précédent

- Les fonctions
- La portée des variables
- Les références



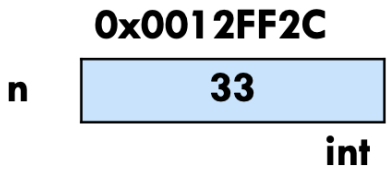
Aujourd'hui

- Les pointeurs :
  - Qu'est-ce que c'est ?
  - Comment on les utilisent ?
  - À quoi ça sert ?
  - Quelques pointeurs particuliers

Rappel : Déclaration d'une variable


```
int n = 33 ;        // int -> 4 octets (processeur 32 et 64 bits)
```

- Une variable c'est :
  - Un nom
  - Un type
  - Une valeur
  - Une adresse



. . .	
0x0012FF2C	33
0x0012FF2D	
0x0012FF2E	
0x0012FF2F	
0x0012FF30	
0x0012FF31	
. . .	

Rappel : Opérateur de référencement &

- Pour obtenir l'adresse d'une variable :

```
int main()
{
    int n = 33;
    cout << "n = " << n << endl;
    cout << "&n = " << &n << endl;
    return 0;
}
```

```
n = 33
&n = 0x16d5d7688
```

Les pointeurs *

- Les adresses sont des **nombres**. Vous connaissez plusieurs types permettant de stocker des nombres : `int`, `unsigned int`, `double`.
- C'est possible de stocker une adresse dans une variable, mais pas avec les types que vous connaissez. Il nous faut utiliser un type un peu particulier : le **pointeur**.
- → **Un pointeur est une variable qui contient l'adresse d'une autre variable.**

Déclarez un pointeur

- Pour déclarer un pointeur il faut, comme pour les variables, deux choses :
  - Un nom.
  - Un type.

- Pour le nom, il n'y a rien de particulier à signaler. Les mêmes règles que pour les variables s'appliquent. Ouf !

- Le type d'un pointeur a une petite subtilité. Il faut indiquer quel est le type de variable dont on veut stocker l'adresse, et ajouter une étoile `*`.

Déclarez un pointeur


```
int *pointeur;
```

- Ce code déclare un pointeur qui peut contenir l'adresse d'une variable de type `int`.
- On peut également écrire `int* pointeur` (avec l'étoile collée au mot `int`).

```
int* pointeur;
```

- Mais cette notation a un léger inconvénient car elle ne permet pas de déclarer plusieurs pointeurs sur la même ligne, comme ceci :

```
int* pointeur1, pointeur2, pointeur3;
```

- Si l'on procède ainsi, seul `pointeur1` sera un pointeur, les deux autres variables seront des entiers tout à fait standard.

Déclarez un pointeur

- On peut bien sûr faire cela pour n'importe quel type :

```
double *pointeurA;
//Un pointeur qui peut contenir l'adresse d'un nombre à virgule


unsigned int *pointeurB;
//Un pointeur qui peut contenir l'adresse d'un nombre entier positif


string *pointeurC;
//Un pointeur qui peut contenir l'adresse d'une chaîne de caractères


const int *pointeurE;
//Un pointeur qui peut contenir l'adresse d'un nombre entier constant
```

Déclarez un pointeur

- Pour le moment, ces pointeurs ne contiennent aucune adresse connue.
- Si vous essayez d'utiliser le pointeur, vous ne savez pas quelle case de la mémoire vous manipulez.
- Ce peut être n'importe quelle case, par exemple celle qui contient votre mot de passe Windows, ou celle stockant l'heure actuelle.
- **Il ne faut donc *jamais* déclarer un pointeur sans lui donner d'adresse.**

Déclarez un pointeur

- **Par conséquent, il faut toujours déclarer un pointeur en lui donnant la valeur `nullptr`.**

```
int *pointeur = nullptr;
double *pointeurA = nullptr;
unsigned int *pointeurB = nullptr;
string *pointeurC = nullptr;
const int *pointeurD = nullptr;
```

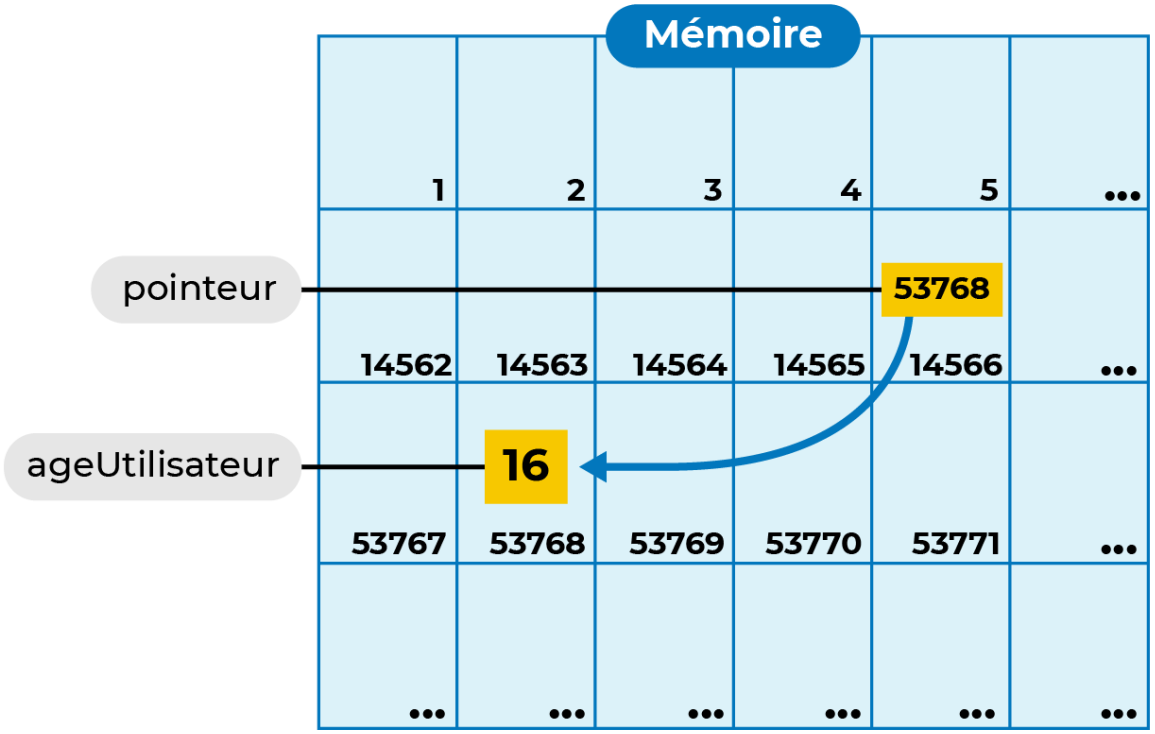
- Lorsque vous créez un pointeur contenant l'adresse `nullptr`, cela signifie qu'il ne contient l'adresse d'aucune case.

Stockez une adresse

- Maintenant qu'on a la variable, il n'y a plus qu'à y mettre une valeur.

```
int main()
{
    int ageUtilisateur = 16;
    //Une variable de type int

    int *pointeur = nullptr;
    //Un pointeur pouvant contenir l'adresse d'un nombre entier

    pointeur = &ageUtilisateur;
    //On met l'adresse de 'ageUtilisateur' dans le pointeur 'ptr'

    return 0;
}
```

Stockez une adresse



- Dans la case mémoire n°14566, il y a une variable nommée `pointeur` qui a pour valeur l'adresse 53768, c'est-à-dire l'adresse de la variable `ageUtilisateur`

Affichez l'adresse

- Comme pour toutes les variables, on peut afficher le contenu d'un pointeur :

```
int main()
{
    int ageUtilisateur = 16;
    int *ptr = nullptr;

    ptr = &ageUtilisateur;

    cout << "L'adresse de 'ageUtilisateur' est : " << &ageUtilisateur << endl;
    cout << "La valeur de pointeur est : " << ptr << endl;

    return 0;
}
```

```
L'adresse de 'ageUtilisateur' est : 0x2ff18
La valeur de pointeur est : 0x2ff18
```

Accédez à la valeur pointée

- Il faut utiliser l'étoile * sur le pointeur pour afficher la valeur de la variable pointée :

```
int main()
{
    int ageUtilisateur = 16;
    int *pointeur = nullptr;

    pointeur = &ageUtilisateur;

    cout << "La valeur est : " << *pointeur << endl;

    return 0;
}
```

```
La valeur est : 16
```

L'opérateur de déréférencement *

- En utilisant l'étoile `*` , on accède à la valeur de la variable pointée.

- C'est ce qui s'appelle "déréférencer" un pointeur.

Synthèse sur la notation

- Pour une variable `int nombre` :
  - `nombre` permet d'accéder à la **valeur** de la variable ;
  - `&nombre` permet d'accéder à l'**adresse** de la variable.

- Sur un pointeur `int *pointeur` :
  - `pointeur` permet d'accéder à la **valeur du pointeur**, c'est-à-dire à l'adresse de la variable pointée ;
  - `*pointeur` permet d'accéder à la **valeur de la variable pointée**.

Envoyez un pointeur à une fonction

- Simplement en précisant que le paramètre est un pointeur

```
void triplePointeur(int *pointeurSurNombre);

int main()
{
    int nombre = 5;

    triplePointeur(&nombre); // On envoie l'adresse de nombre à la fonction
    cout << nombre; // On affiche la variable nombre. La fonction a
directement modifié la valeur de la variable car elle connaissait son adresse

    return 0;
}

void triplePointeur(int *pointeurSurNombre)
{
    *pointeurSurNombre *= 3; // On multiplie par 3 la valeur de nombre
}
```

Envoyez un pointeur à une fonction

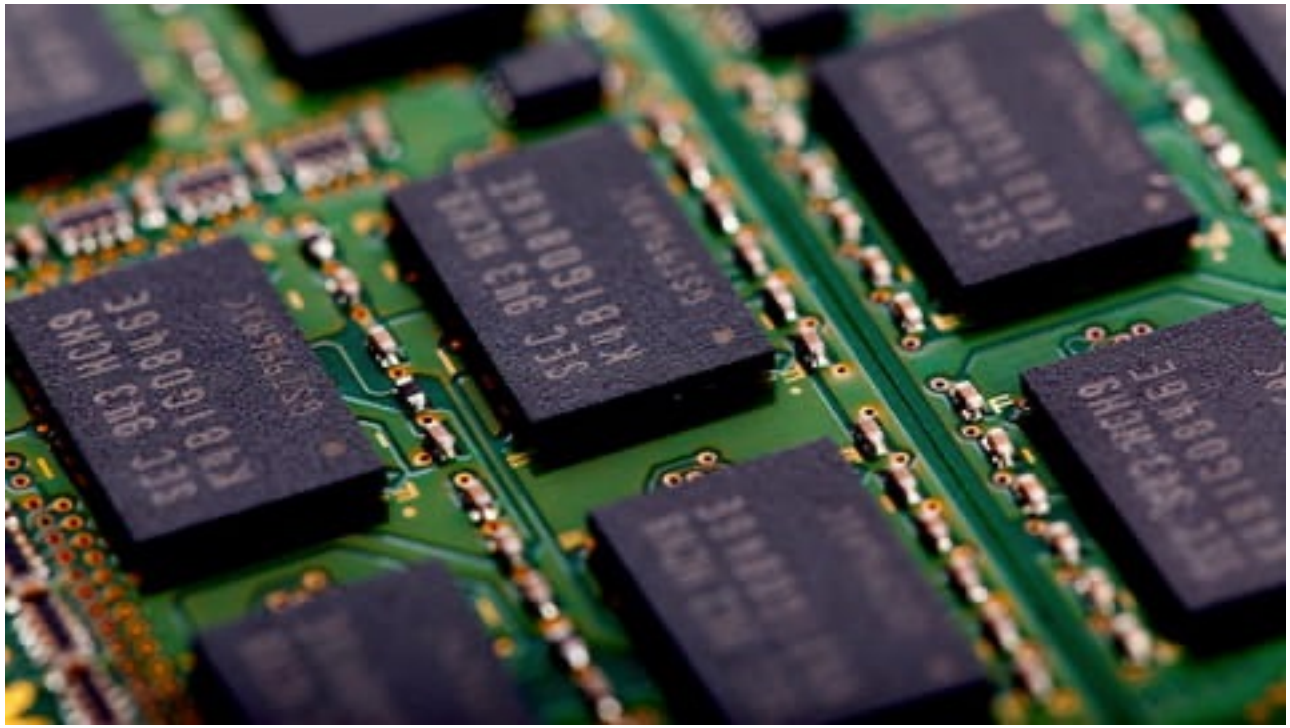
- Alternative : ajoutez un pointeur dans la fonction `main`

```
void triplePointeur(int *pointeurSurNombre);

int main()
{
    int nombre = 5;
    int *pointeur = &nombre; // pointeur prend l'adresse de nombre

    triplePointeur(pointeur); // On envoie pointeur (l'adresse de nombre) à
la fonction
    cout << *pointeur; // On affiche la valeur de nombre avec *pointeur

    return 0;
}

void triplePointeur(int *pointeurSurNombre)
{
    *pointeurSurNombre *= 3; // On multiplie par 3 la valeur de nombre
}
```

Mais à quoi ça sert les pointeurs ?

Cas d'usage n°1

- Gérer la création et la destruction des cases mémoire



Faites de l'allocation dynamique

- Lors de la déclaration d'une variable, le programme effectue deux étapes :
  1. Il demande à l'ordinateur de lui fournir une zone dans la mémoire. En termes techniques, on parle d'**allocation** de la mémoire.
  2. Il remplit cette case avec la valeur fournie. On parle alors d'**initialisation** de la variable.

- De même, lorsque l'on arrive à la fin d'une fonction, le programme rend la mémoire utilisée à l'ordinateur. C'est ce qu'on appelle la **libération** de la mémoire.

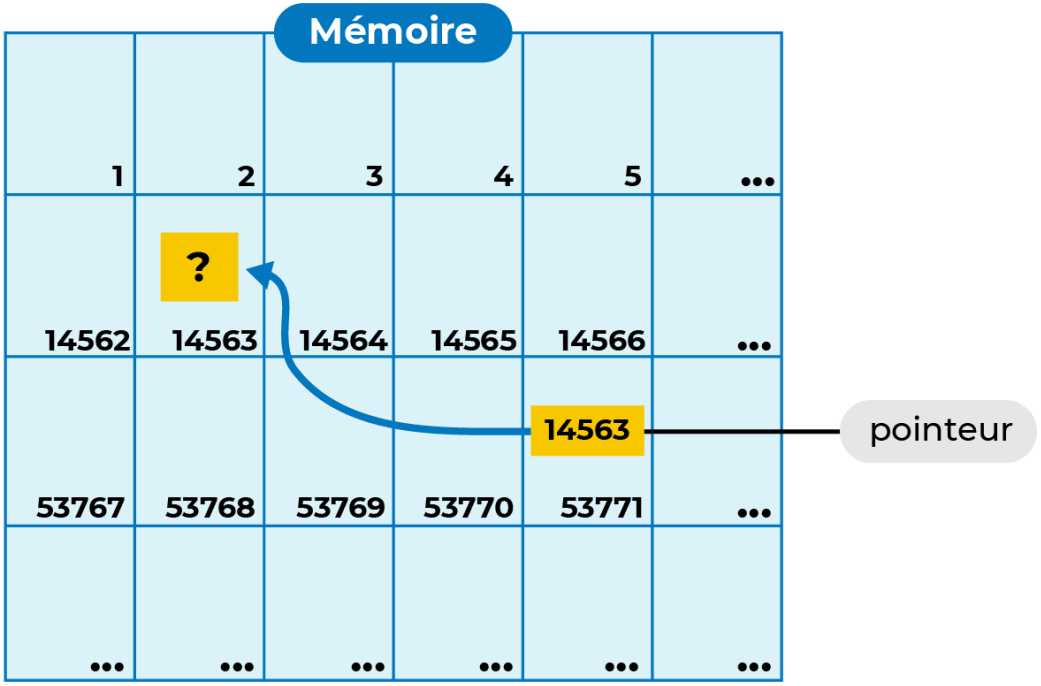
Allouez un espace mémoire

- Tout ceci se faisait automatiquement, nous allons maintenant apprendre à le faire manuellement
- Pour demander manuellement une case dans la mémoire, il faut utiliser l'opérateur `new`.
- `new` demande une case à l'ordinateur, et renvoie un pointeur pointant vers cette case

```
int *pointeur = nullptr;
pointeur = new int;
```

Allouez un espace mémoire

- La case 14563 qui contient une variable de type `int` non initialisée.
- La case 53771 qui contient un pointeur pointant sur la variable.

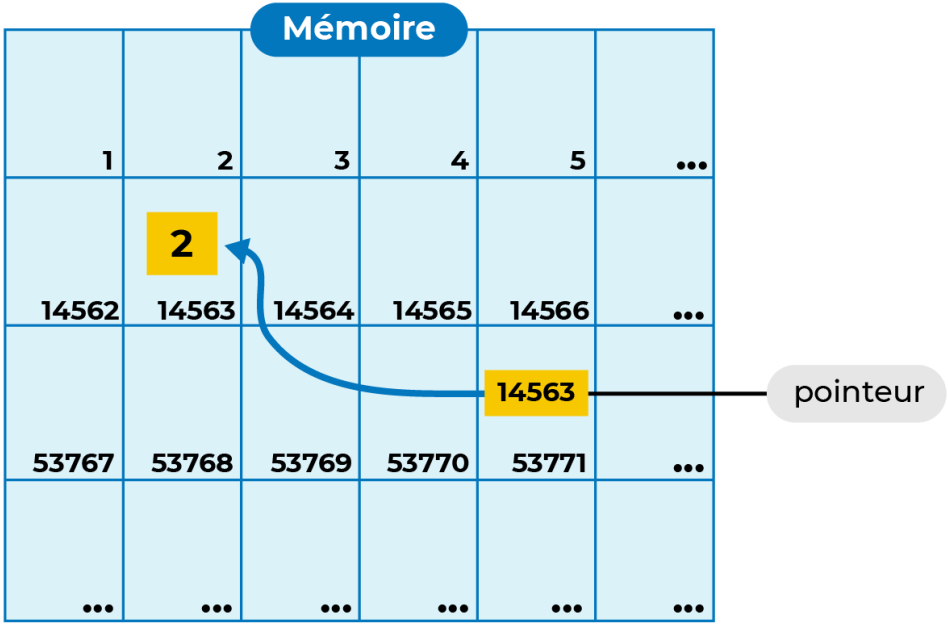


- Rien de neuf. Mais le point important, c'est que la variable dans la case 14563 **n'a pas de nom**. Le seul moyen d'y accéder est donc de passer par le pointeur.

Allouez un espace mémoire

- Une fois allouée manuellement, la variable s'utilise comme n'importe quelle autre. On doit juste se rappeler qu'il faut y accéder par le pointeur, en le déréférençant.

```
int main()
{
    int *pointeur = nullptr;
    pointeur = new int;


    *pointeur = 2;  //On accède à
la case mémoire pour en modifier la
valeur

   return 0;
}
```



Attention aux fuites de mémoire

- Si vous changez la valeur du pointeur, vous perdez le seul moyen d'accéder à cette case mémoire.
- Vous ne pourrez donc plus l'utiliser ni la supprimer ! Elle sera définitivement perdue, mais elle continuera à prendre de la place.
- C'est ce qu'on appelle une **fuite de mémoire**.

Libérez la mémoire

- Une fois que l'on n'a plus besoin de la case mémoire, il faut la rendre à l'ordinateur.
- Cela se fait via l'opérateur `delete`.

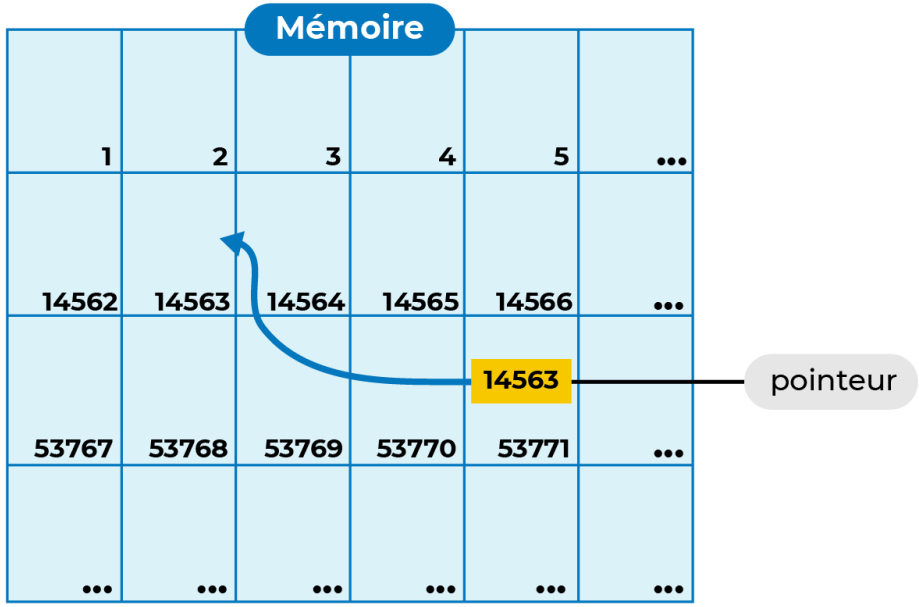
```
int *pointeur = nullptr;
pointeur = new int;

delete pointeur;  //On libère la case mémoire
```

- La case est alors rendue à l'ordinateur qui pourra l'employer à autre chose.
- Le pointeur, lui, existe toujours et il pointe toujours sur la case, mais vous n'avez plus le droit de l'utiliser

Libérez la mémoire

- Après avoir fait appel à `delete`, il est donc essentiel de supprimer cette "flèche" en mettant le pointeur à l'adresse `nullptr`.
- Ne pas le faire est une cause très courante de plantage des programmes.

```
int *pointeur = nullptr;
pointeur = new int;

delete pointeur;     //On libère la case mémoire
pointeur = nullptr;
//On indique que le pointeur ne pointe plus vers rien
```



Exemple d'utilisation


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

int main()
{
   int* pointeur = nullptr;
   pointeur = new int;

   cout << "Quel est votre age ? ";
   cin >> *pointeur;
   //On écrit dans la case mémoire pointée par le pointeur 'pointeur'

   cout << "Vous avez " << *pointeur << " ans." << endl;
   //On utilise à nouveau *pointeur

   delete pointeur;   //Ne pas oublier de libérer la mémoire
   pointeur = nullptr;       //Et de faire pointer le pointeur vers rien

   return 0;
}
```

Tableau dynamique


```
pointeur=new type[taille];
```

- L'opérateur `new[]` permet d'allouer une zone mémoire pouvant stocker taille éléments de type `type`, et retourne l'adresse de cette zone.
- Le paramètre `taille` est un entier qui peut être quelconque (variable, constante, expression).
- `new` renverra un pointeur vers un `type`.
- La variable pointeur est donc du type `type *`.
- Les cases du tableau seront numérotées de `0` à `taille-1` et on y accédera comme un tableau statique.
- S'il n'y a pas assez de mémoire disponible, `new` renvoie le pointeur `NULL`.

```
delete[] pointeur;
```

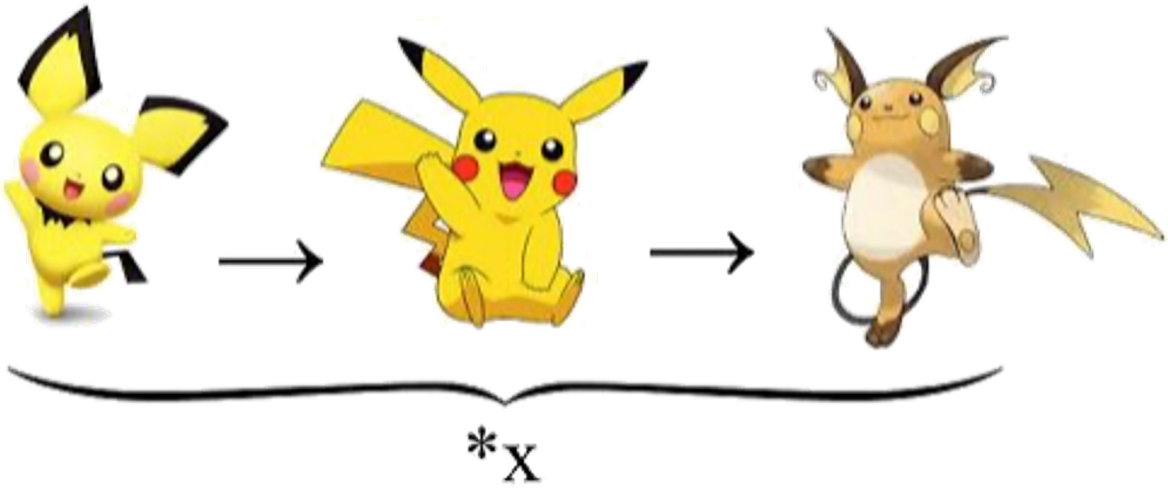
- L'opérateur `delete[]` permet de détruire un tableau précédemment alloué grâce à `new[]`.

Exemple d'utilisation


```
int main()
{
    int size;

    cout << "Tapez la taille du tableau : ";
    cin >> size;

    int *array = nullptr;
    array = new int[size];      //Création du tableau

    for (int i = 0; i < size; i++)      //Remplissage du tableau
        array[i] = i * i;

    for (int i = 0; i < size; i++)      //Affichage du tableau
       cout << array[i] << endl;

    delete[] array;     //Libération de l'espace avec l'opérateur delete[]
    array = nullptr;

    return 0;
}
```

Cas d'usage n°2

- Sélectionner une valeur parmi plusieurs options



Exemple d'utilisation

- Dans un jeu de stratégie du type Warcraft, dans un combat chaque personnage a une cible précise.

Exemple d'utilisation

- Chaque personnage possède un pointeur qui pointe vers sa cible. Il a ainsi un moyen de savoir qui viser et attaquer.

```
Personnage *cible;  //Un pointeur qui pointe sur un autre personnage
```

- Quand il n'y a pas de combat en cours, le pointeur pointe vers l'adresse `nullptr`, il n'a pas de cible.
- Quand le combat est engagé, le pointeur pointe vers un ennemi.
- Enfin, quand cet ennemi meurt, on déplace le pointeur vers une autre adresse, c'est-à-dire vers un autre personnage.
- Le pointeur est donc réellement utilisé ici comme une flèche reliant un personnage à son ennemi.

Pointeurs sur structure

- Un pointeur de structure se crée de la même manière qu'un pointeur de `int`, de `double` ou de n'importe quelle autre type de base :

```
Coordonnees *point = nullptr;
```

- Pour accéder à une composante de la structure il ne faut pas faire comme cela :

```
*point.x = 0;
```

- Mais comme cela :

```
(*point).x = 0;
point->x = 0;      //Syntaxe à privilégier
```

Exemple d'utilisation


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

struct Coordonnees
{
    int x, y;
};

void initialiserCoordonnees(Coordonnees *point)
{
    point->x = point->y = 0;
}

int main()
{
    Coordonnees monPoint;
    Coordonnees *pointeur = &monPoint;

    initialiserCoordonnees(pointeur);
    return 0;
}
```

Pointeurs de fonctions

- Il est possible de faire des pointeurs de fonctions.
- Un pointeur de fonction contient l'adresse du début du code binaire constituant la fonction.
- Pour déclarer un pointeur de fonction, il suffit de considérer les fonctions comme des variables.

```
type (*identificateur)(paramètres);
```

Exemple d'utilisation


```
#include <iostream>

using namespace std;

/* Définit plusieurs fonctions travaillant sur des entiers : */
int somme(int nb1, int nb2){
    return nb1+nb2;
}

int multiplication(int nb1, int nb2){
    return nb1*nb2;
}

int quotient(int nb1, int nb2){
    return nb1/nb2;
}

int modulo(int nb1, int nb2){
    return nb1%nb2;
}
```

Exemple d'utilisation


```
//Initialise le tableau de pointeur de fonctions
int (*ftab[])(int, int) = {somme, multiplication, quotient, modulo};

int main(void)
{
    int nb1,nb2,selection;

    cout << "Entrez deux entiers : ";
    cin >> nb1 >> nb2;          /* Demande les deux entiers i et j. */

    cout << endl << "Entrez la fonction : ";
    cin >> selection;          /* Demande la fonction à appeler. */

    if (selection < 4 && selection >= 0)
        cout << endl << "Résultat : " << (*(ftab[selection]))(nb1,nb2);
    else
        cout << endl << "Mauvais numéro de fonction." << endl;
    return 0;
}
```

Danger sur l'utilisation des pointeurs

- VEILLEZ À TOUJOURS INITIALISER LES POINTEURS QUE VOUS UTILISEZ.
- VÉRIFIEZ QUE TOUTE DEMANDE D'ALLOCATION MÉMOIRE A ÉTÉ SATISFAITE.
- PENSEZ A LIBERER LA MÉMOIRE ALLOUEE MANUELLEMENT
- LORSQU'ON UTILISE UN POINTEUR, IL FAUT VÉRIFIER S'IL EST VALIDE


KEEP
CALM
AND
MAY THE FORCE
BE WITH YOU