

Support de cours

Cours:

Introduction à la programmation orientée objet (en C++)

Vidéo:

INIT-CPP-7_4-PointeursDeclarationetOperateurs-pt1

Concepts (extraits des sous-titres générés automatiquement) :

Valeur d'un type particulier. Nom du type. Exemple int. Adresse d'une zone mémoire. Second temps. Moyen d'une valeur spéciale. Syntaxe du langage c. Valeur de type double. Première ligne. Adresse de la zone mémoire. Opérateurs fondamentaux. Valeur du pointeur. Identificateur de la variable. Pointeurs. Identificateur du pointeur.



[vers la recherche de séquences vidéo](#)

(dans Introduction à la programmation orientée objet (en C++).)



[vers la vidéo](#)

Center for Digital Education. Plus de matériel de soutien pédagogique ici :

<https://www.epfl.ch/education/educational-initiatives/cede/educational-technologies-gallery/boocs-en/>

Pointeurs : déclaration et opérateurs de base

(Partie 1)

Initiation à la programmation (C++)

Vincent Lepetit, Jean-Cédric Chappelier et Jamila Sam

...

notes

résumé

0m 0s





Avec notre petite analogie du carnet d'adresses,

notes

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

résumé

0m 1s



Les pointeurs (3) : la pratique

La déclaration d'un pointeur se fait selon la syntaxe suivante :

```
type* identificateur;
```

Cette instruction déclare une variable de nom *identificateur* de type pointeur sur une valeur de type *type*.

Exemple : `int* ptr;`

déclare une variable `ptr` qui pointe sur une valeur de type `int`.

L'initialisation d'un pointeur se fait selon la syntaxe suivante :

```
type* identificateur(adresse);
```

Exemples :

```
int* ptr(nullptr);
```

```
int* ptr(&i);
```

```
int* ptr(new int(33));
```

vous connaissez maintenant l'essentiel des manipulations que l'on peut faire sur des pointeurs. Nous allons maintenant aborder l'ensemble de ces manipulations les unes après les autres et voir comment concrètement elles se pratiquent dans la syntaxe du langage C++..

notes

résumé

0m 5s



Les pointeurs (3) : la pratique

La déclaration d'un pointeur se fait selon la syntaxe suivante :

```

int x;
type* identificateur;
int* ptr;

```

Cette instruction déclare une variable de nom *identificateur* de type pointeur sur une valeur de type *type*.

Exemple : `int* ptr;`

déclare une variable *ptr* qui pointe sur une valeur de type *int*.

L'initialisation d'un pointeur se fait selon la syntaxe suivante :

```

type* identificateur(adresse);
int x(4);

```

Exemples :

```

int* ptr(nullptr);
int* ptr(&i);
int* ptr(new int(33));

```

ne pointe vers rien

Pour déclarer une variable en C++ nous savons que nous pouvons avoir recours à la syntaxe suivante. Donc nom du type, par exemple `int` pour des entiers suivi de l'identificateur associé à la variable, à savoir « `x` ». Pour un pointeur, c'est une notation équivalente, sauf que le type va indiquer qu'il s'agit d'un pointeur sur une valeur d'un type particulier. Donc si par exemple je veux déclarer un pointeur sur un entier, le type associé sera « `int*` » pour dire pointeur sur entier suivi de l'identificateur du pointeur, donc de la variable de type pointeur associé. De même si je veux déclarer un pointeur sur un double j'aurais recours à la syntaxe suivante. Donc j'écrirais `double*` comme type suivi de l'identificateur de la variable « `ptr` ». Donc ici, j'ai déclaré une variable « `ptr` » qui est de type pointeur sur une valeur de type double. Ceci est pour la déclaration tout court. Si je veux déclarer-initialiser, la syntaxe est analogue à ce que je connais jusqu'ici. Par exemple, si je veux écrire, déclarer et initialiser une variable « `x` » de type entier initialisée à 4, j'ai recours à cette syntaxe-là. Et bien, la syntaxe est tout à fait analogue pour les pointeurs. Je dois indiquer entre parenthèses la valeur du pointeur. Et cette valeur doit être une adresse. Donc ici très concrètement, trois exemples de déclaration-initialisation d'une variable de type pointeur. Ici nous anticipons un petit peu sur ce qui va suivre dans cette séquence et dans les suivantes, à savoir comment obtenir l'adresse d'une zone mémoire nommée, ici l'adresse de la zone mémoire appelée « `i` », de la variable « `i` ». Comment allouer dynamiquement un emplacement mémoire capable de contenir une donnée d'un type particulier, ici un entier. Donc bien évidemment, nous allons revoir ces notions beaucoup plus en détails le moment venu. Première ligne ici, je suis

notes

résumé

0m 17s



Les pointeurs (3) : la pratique

La déclaration d'un pointeur se fait selon la syntaxe suivante :

```

    int x;
    type* identificateur;
    int* ptr;
  
```

Cette instruction déclare une variable de nom *identificateur* de type pointeur sur une valeur de type *type*.

Exemple : `int* ptr;`

déclare une variable `ptr` qui pointe sur une valeur de type `int`.

L'initialisation d'un pointeur se fait selon la syntaxe suivante :

```

    type* identificateur(adresse);
    int x(4);
  
```

Exemples :

```

    int* ptr(nullptr);
    int* ptr(&i);
    int* ptr(new int(33));
  
```

ne pointe vers rien

en train de déclarer-initialiser une variable « ptr » de type pointeur, sur un entier. Donc la variable « ptr » est capable de contenir l'adresse d'un entier et nous initialisons cette variable au moyen d'une valeur spéciale qui est `nullptr` qui signifie que le pointeur « ptr » ne pointe vers rien. Donc rappelez-vous la petite analogie avec le carnet d'adresses. Ici nous avons une page du carnet d'adresses qui est vide, qui est initialisée à – qui est effacée, qui ne contient rien. Deuxième instruction, j'initialise un pointeur « ptr » au moyen de l'adresse d'une zone existante, d'une variable existante en mémoire. Donc nous allons revenir sur ce point un peu plus tard

notes

résumé

Opérateurs sur les pointeurs

C++ possède deux *opérateurs* particuliers en relation avec les pointeurs : `&` et `*`.

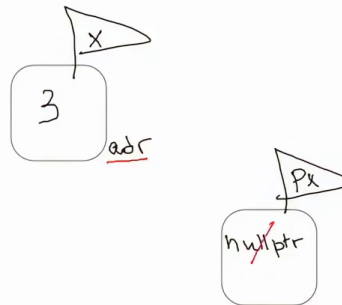
`&` est l'opérateur qui

retourne l'adresse mémoire de la valeur d'une variable

Si `x` est de type `type`, `&x` est de type `type*` (pointeur sur `type`).

Exemple :

```
int x(3);
int* px(nullptr);
px = &x;
```



C++ met à disposition deux opérateurs fondamentaux pour la manipulation de pointeurs. L'opérateur « & », l'opérateur « * ». Le rôle de l'opérateur « & » est de retourner l'adresse mémoire de la valeur d'une variable. Examinons ce que cela signifie sur un exemple concret. Donc ici nous avons un petit programme qui commence par déclarer une variable « x » de type entier et l'initialise à 3. Donc nous avons cette situation mémoire et nous savons que à chaque variable d'un programme est associée une adresse en mémoire. Notons cette adresse « adr ». Dans un second temps nous déclarons une seconde variable « px » qui cette fois est de type pointeur sur un entier. Donc elle est capable de contenir l'adresse d'une zone mémoire contenant un entier. Donc nous avons cette situation mémoire. Et ici dans un premier temps nous initialisons la variable « px » au moyen de la variable spéciale nullptr. Ce qui veut dire concrètement que « px » est capable de pointer sur une zone mémoire contenant un entier, mais à ce stade ne pointe vers rien. Elle ne pointe sur rien du tout. Ensuite, troisième ligne du programme. Et c'est là qu'intervient notre petit opérateur « & ». Nous affectons à la variable « px » l'adresse de la variable « x ». Nous avons vu que cette adresse ici est « adr »,

notes

résumé

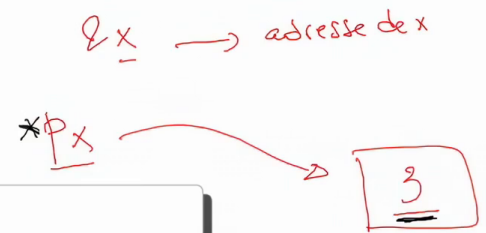
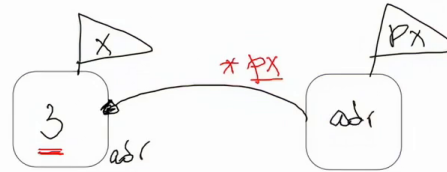
3m 5s



Opérateurs sur les pointeurs (2)

* est l'opérateur qui **retourne la valeur pointée par une variable pointeur**.

Si `px` est de type `type*`, `(*px)` est la valeur de type `type` pointée par `px`. Exemple :



```
int x(3);           // x est de type entier
int* px(nullptr);  // px est un pointeur sur entier

px = &x;           // px pointe sur la variable x
cout << *px        // affiche la valeur pointee par px : 3
    << endl;
```

Note : `*&i` est donc strictement équivalent à `i`

donc nous allons nous trouver dans cette situation. Nous allons affecter à « `px` » l'adresse de « `x` ». Donc nous établissons le lien entre le pointeur et la zone pointée, au moyen de cette affectation. Donc pour résumer, « `px` » retourne l'adresse associée à la variable « `x` ». L'opérateur « `*` » permet de retourner la valeur pointée par une variable de type pointeur. Tout à l'heure nous avons vu l'opérateur « `&` ». Donc « `&x;` » où « `x` » est une variable, et va retourner l'adresse de « `x` » en mémoire. Maintenant imaginons que nous ayons un pointeur « `px` ». Ce dernier pointe, contient l'adresse d'une zone mémoire contenant une certaine valeur, par exemple 3. Et nous souhaitons à partir du pointeur, retrouver la valeur pointée. À ce moment-là il faut utiliser l'opérateur « `*` » appliqué à une variable de type pointeur et c'est ce qui va nous permettre de retrouver la valeur pointée par ce pointeur. Donc si nous reprenons notre petit exemple de la séquence précédente. Nous déroulons les premières instructions et nous nous trouvons la situation suivante. Une variable « `x` » initialisée à 3 de type entier et qui a une adresse. Une seconde variable « `px` » qui est de type pointeur sur un entier qu'on commence par initialiser à `nullptr`, ce qui veut dire qu'elle ne pointe sur rien. Et à laquelle nous affectons par la suite l'adresse de « `x` » ce qui va permettre d'établir le lien entre le pointeur et la variable pointée. Ensuite, dernière instruction, et c'est là que nous faisons appel à notre fameux petit opérateur « `*` » qui va permettre justement d'établir le lien entre le pointeur et la valeur pointée. Donc ici « `*px` » nous appliquons « `*` » au pointeur, va nous permettre de retrouver à

notes

résumé

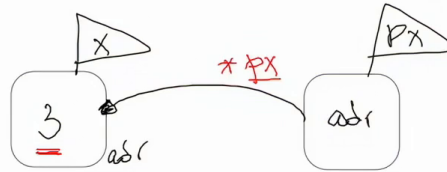
4m 25s



Opérateurs sur les pointeurs (2)

***** est l'opérateur qui **retourne la valeur pointée par une variable pointeur**.

Si `px` est de type `type*`, `(*px)` est la valeur de type `type` pointée par `px`. Exemple :



```
int x(3);           // x est de type entier
int* px(nullptr);  // px est un pointeur sur entier

px = &x;           // px pointe sur la variable x
cout << *px        // affiche la valeur pointee par px : 3
    << endl;
```

Note : `*&i` est donc strictement équivalent à `i`

partir du pointeur, la valeur pointée par ce pointeur

notes

résumé

Imaginez maintenant que j'ai une variable de type int nommée « i » qui contient une certaine valeur et qui a une adresse en mémoire. Si j'écris « &i; », j'ai en fait en retour une valeur de type pointeur sur entier qui a pour valeur l'adresse « adr » Si j'écris maintenant « *&i; », je vais appliquer « * » à ce pointeur qui va me retourner la valeur contenue à l'adresse et donc exactement 3. Donc écrire « *&i; » est donc strictement équivalent à écrire « i » puisque j'accède directement à la valeur contenue dans « i » à la valeur contenue dans « i »

6m 13s

