Support de cours

Cours:

Initiation à la programmation (en C++)

Vidéo:

Références (partie 1)

Concepts (extraits des sous-titres générés automatiquement) :

Variable de type entier. Variable i. Travers de la variable. Nouvelle variable. Exemple val. Référence j. Cas de droite. Fois i. Cas de la référence. Fonction f. Déclaration d'une référence. Juste i. Cas. Référence. Cas de gauche.

vers la recherche de séquences vidéo
(dans Initiation à la programmation (en C++).)

vers la vidéo

EPFL

Références
(Partie 1)

Initiation à la programmation (C++)

Vincent Lepetit, Jean-Cédric Chappelier et Jamila Sam

...

notes

résumé

0m 0s

Le type « référence »

Une **référence** est un *autre nom* pour un objet existant, un *synonyme*, un *alias*

Une référence, c'est simplement juste un autre nom, un synonyme, un alias,

notes

résumé

0m 1s

Le type « référence »

EPFL

Une **référence** est un *autre nom* pour un objet existant, un *synonyme*, un *alias*

Une référence permet donc de désigner un objet indirectement

C'est exactement ce que l'on utilise lors d'un *passage par référence*

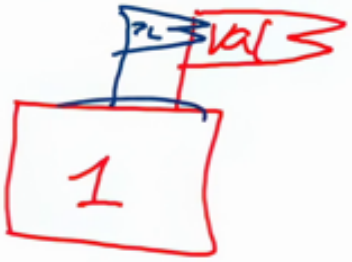
La déclaration d'une référence se fait selon la syntaxe suivante :

```
type& nom_reference(identificateur);
```

Après une telle déclaration, *nom_reference* peut être utilisé partout où *identificateur* peut l'être.

Exemple :

```
int  val(1);
int& x(val);
```



pour un objet qui existe déjà en mémoire. Si par exemple j'ai déjà un objet, une variable de type entier qui s'appelle par exemple val et qui contient la valeur 1. Je peux lui donner un autre nom, par exemple x, pour ce même endroit, cette même variable en mémoire, ce sera une référence. Une référence ça sert donc juste à désigner un autre objet qui est en mémoire. C'est exactement ce que l'on utilisait lorsque l'on faisait des passages par référence. Si on avait pas exemple une fonction f qui prenait par référence un entier a, qu'on écrivait comme ceci, avec le signe &, ici. On faisait bien un passage par référence si quelque part, plus tard, dans le programme, j'avais déclaré par exemple un entier que j'appelle b, que j'initialise à 3, et que je faisais un appel à f(b). b était passé par référence à la fonction f au travers de la variable a, paramètre de la fonction f. Ce passage par référence permettait que ce soit le même objet qui soit désigné à l'extérieur et à l'intérieur de la fonction. C'est de ça dont il s'agit quand on parle de référence. La syntaxe très générale de déclaration d'une référence est la suivante. On va pouvoir généraliser cette syntaxe, utilisée par le passage par référence dans les fonctions, en ajoutant le signe & derrière le type puis le nom d'une référence. Ce n'est pas une nouvelle variable. C'est juste une nouvelle étiquette sur une variable qui existe. Puis ici, on la lie à un identificateur. Si je reprends un exemple, avec une variable qui s'appelle val qui est de type entier, qui contient la valeur 1. Je peux alors à ce moment là, déclarer ici une référence que j'appelle x et qui est un autre nom

notes

résumé

0m 9s

Le type « référence »

EPFL

Une **référence** est un *autre nom* pour un objet existant, un *synonyme*, un *alias*

Une référence permet donc de désigner un objet indirectement

C'est exactement ce que l'on utilise lors d'un *passage par référence*

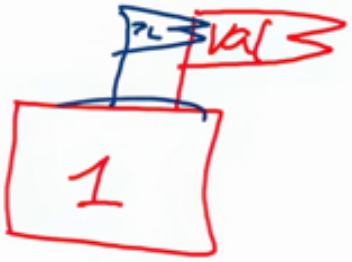
La déclaration d'une référence se fait selon la syntaxe suivante :

```
type& nom_reference(identificateur);
```

Après une telle déclaration, *nom_reference* peut être utilisé partout où *identificateur* peut l'être.

Exemple :

```
int  val(1);
int& x(val);
```



sur la variable val. Cette déclaration, cette syntaxe, fait que x est un autre nom pour la variable val Ça ne fait pas une nouvelle variable x dans laquelle je copierai val, mais ça crée un nouveau drapeau, un nouveau nom, une nouvelle étiquette,

notes

résumé

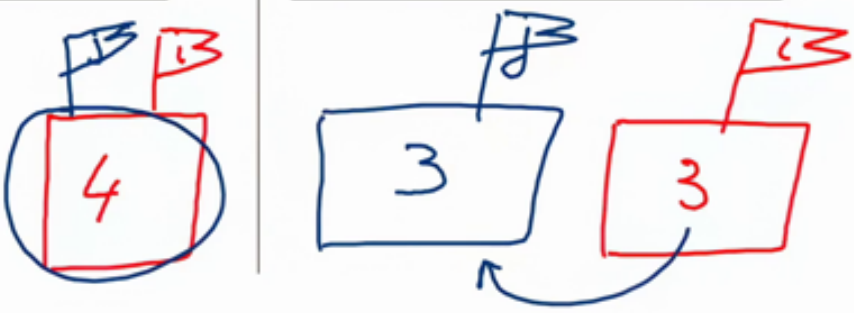
Attention aux pièges !

EPFL

- signification de l'opérateur = :

```
int i(3);
int& j(i); // alias
/* i et j sont la MÊME *
 * case mémoire        */
i = 4; // j AUSSI vaut 4
j = 6; // i AUSSI vaut 6
```

```
int i(3);
int j(i); // copie
/* i et j vivent leur *
 * vie séparément     */
i = 4; // j vaut encore 3
j = 6; // i vaut encore 4
```



pour ce qui existe déjà en mémoire, qui est la variable val. A partir de là, je peux aussi bien désigner cette variable par x que par val Ces deux noms vont désigner la même variable. Attention donc aux confusions, aux pièges, liés au fait qu'une référence n'est pas une variable. C'est juste une étiquette sur une variable. Par exemple, au piège lié au signe = Comparons deux bouts de code, ici assez similaires où l'on a une variable i initialisée à 3 et où l'on a dans un cas une référence j qui est une référence, un alias, sur i, et dans l'autre cas, ici, une seconde variable j Qu'est-ce qu'il va se passer ? Dans les deux cas, nous avons une variable i qui est initialisée et qui contient la valeur 3. Dans le cas de la référence nous avons, par ailleurs, en plus une référence j c'est à dire un autre nom pour la même variable, au même endroit en mémoire. Alors que dans le cas de droite, nous avons ici une variable j et donc une seconde variable est très différent. Dans ce cas là, ce qui va se passer, c'est qu'on initialise j avec la valeur de i. On copie la valeur de i dans j. Alors que dans le cas de gauche, dans le cas des références, c'est le même objet, la même variable qui est référencée. Ça a pour conséquence que si on affecte 4 à i dans le cas de la référence, ce qu'il va se passer, c'est que à la fois i et j vont avoir la valeur 4. J vaudra aussi 4.

notes

résumé

2m 37s

Alors que dans le cas de droite, ici nous avons juste i, la copie ayant été effectuée, cette flèche n'a plus lieu d'être. La copie est terminée. C'est donc seulement i, i et lui seul, qui vaut 4. j continue à avoir une valeur 3. De même si on modifie j, dans le cas de la référence, modifier j revient à modifier aussi i. Donc ici si je modifie j alors i vaut aussi 6. Alors que dans le cas de deux variables complètement différentes, si je modifie j alors i n'est pas modifié et i vaut toujours 4. et i vaut toujours 4.

notes

résumé

4m 25s