Support de cours

Cours:

Initiation à la programmation (en C++)

Vidéo:

Références (partie 2)

Concepts (extraits des sous-titres générés automatiquement) :

Entier i. Travers de l'objet. Case mémoire i. Variable j. Sens de const. Variable i. Deuxième petit piège. Const référence. Mot clé. Vrais pointeurs. Espace mémoire. Travers de cette étiquette. Grosse différence. Séquence d'introduction. Fois i.

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

vers la vidéo

EPFL

Références
(Partie 2)

Initiation à la programmation (C++)

Vincent Lepetit, Jean-Cédric Chappelier et Jamila Sam

...

notes

résumé

0m 0s

Deuxième petit piège, avec lequel on a souvent des confusions,

notes

résumé

0m 1s

Attention aux pièges !

EPFL

- sémantique de `const` :

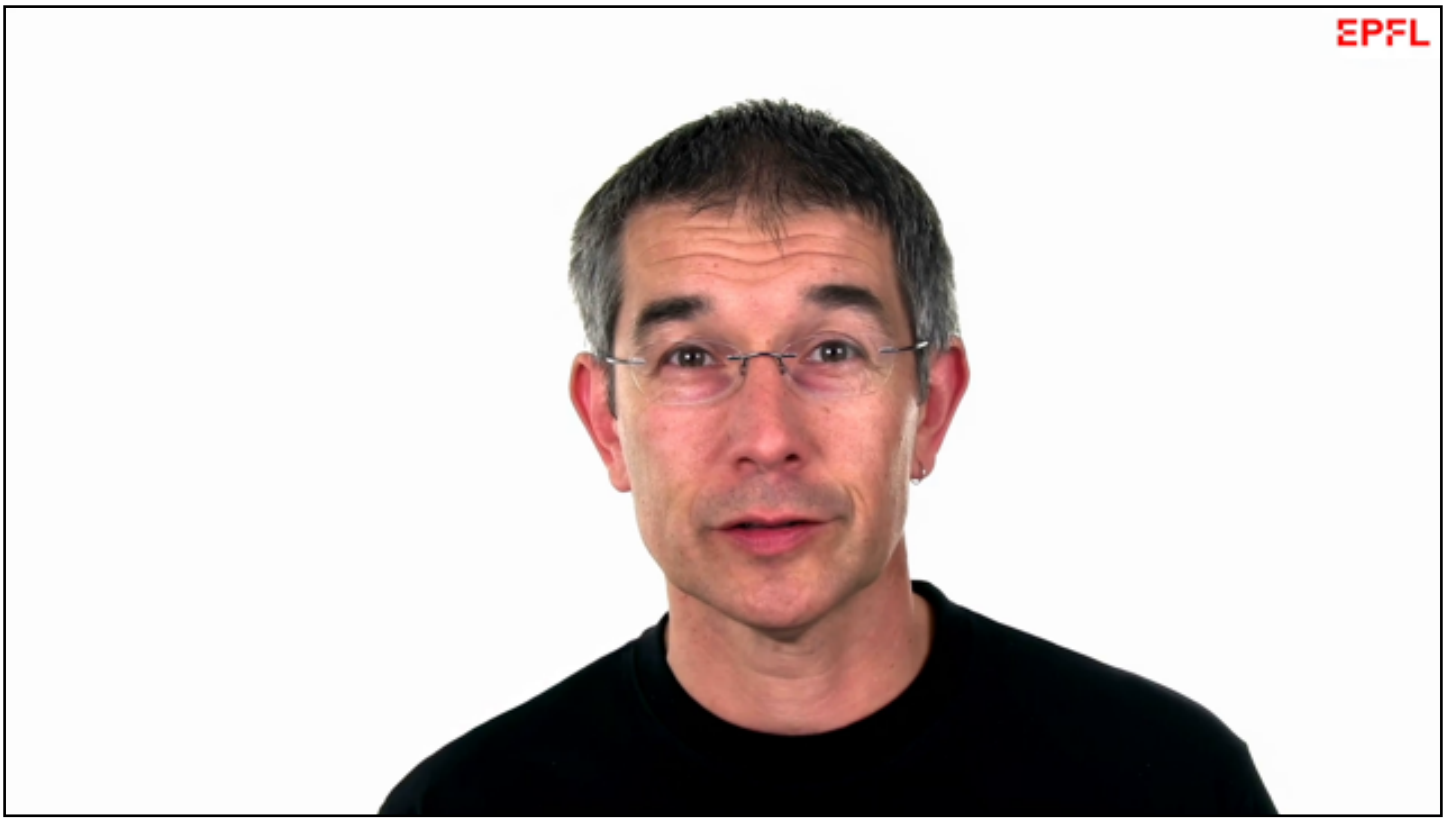
```
int i(3);
const int& j(i); /* i et j sont les mêmes.                    *
                  * On ne peut pas changer la valeur VIA J *
                  * (mais on peut le faire par ailleurs).  */
j = 12; // NON
i = 12; // OUI, et j AUSSI vaut 12 !
```

c'est le sens de const. on avait déjà insisté plusieurs fois sur le fait que const voulait dire qu'au travers de l'objet qu'on a qualifié de const on ne pouvait pas le modifier. Mais cet objet pouvait être par ailleurs modifié. C'est le cas avec des références. Regardons ici toujours avec un entier i qui est initialisé à la valeur 3. On déclare ici une const référence sur i que l'on appelle j. C'est à dire que l'on a défini ici un autre nom, un autre drapeau, une autre étiquette. Au travers de cette étiquette, je vais la dessiner plus grasse, on n'a pas le droit de modifier le contenu. Ça ne veut pas dire que le contenu ne peut pas être modifié par ailleurs. Si je veux modifier le contenu de cette même variable en mémoire qui s'appelle i et j, par exemple, par une affectation en voulant mettre 12. Je ne pourrais pas. Cette ligne va être interdite par le compilateur. Elle est interdite parce que, ici, j est const, c'est à dire qu'au travers de j je n'ai pas le droit de modifier. Cela n'empêche pas qu'au travers de i je peux modifier cet espace mémoire et j'ai tout à fait le droit de faire : i = 12 ce qui va me mettre 12 dans la case mémoire i qui est aussi la case mémoire qui s'appelle j. j vaut aussi 12. Tout ça pour récapituler, il faut bien faire attention que le mot clé, le mot réservé const s'applique à l'objet, au nom dont on parle et pas forcément à l'objet en mémoire qui est derrière. Ça s'applique à l'objet au travers de ce nom

notes

résumé

0m 5s



et non pas à l'objet en tant que tel dans l'absolu. J'ai dit dans la séquence d'introduction

notes

résumé

1m 49s

Spécificités des références (1/2)

Une référence :

- doit absolument être initialisée (vers un objet existant) :

```
int i;
int& ri(i); // OK
int& rj;    // NON, la référence rj doit être liée à un objet !
```

que les références n'étaient pas des vrais pointeurs en tant que tel, ce sont des objets assez séparés. C'est pour ça, d'ailleurs, qu'elles ont un nom séparé qu'on appelle référence. Je voudrais donc insister sur les spécificités de ces références.

notes

résumé

1m 53s

Spécificités des références (1/2)

EPFL

Une référence :

- doit absolument être initialisée (vers un objet existant) :

```
int i;
int& ri(i); // OK
int& rj;    // NON, la référence rj doit être liée à un objet !
```

La première de ces spécificités, c'est qu'une référence doit absolument toujours être initialisée. Elle désigne toujours un objet existant. Alors qu'un pointeur pourrait pointer vers quelque chose qui n'existe pas. Ici, j'ai déclaré de nouveau une variable i de type entier, et ici je déclare une référence ri sur l'objet, la variable i C'est à dire que c'est un autre nom pour i. Mais je ne peux pas déclarer une référence rj qui ne serait liée à aucun objet. Ceci est invalide.

notes

résumé

2m 6s

Spécificités des références (1/2)

EPFL

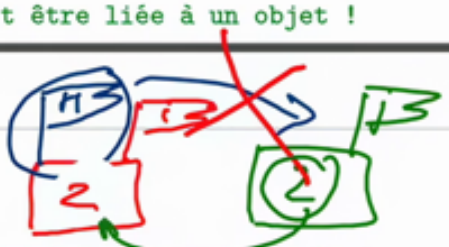
Une référence :

- doit absolument être initialisée (vers un objet existant) :

```
int i;
int& ri(i); // OK
int& rj;    // NON, la référence rj doit être liée à un objet !
```

- ne peut être liée qu'à un seul objet :

```
int i;
int& ri(i);
int j(2);
ri = j;  /* ne veut pas dire que ri est maintenant un alias de j, *
          * mais que i prend la valeur de j !!                    */
j = 3;
cout << i << endl; // affiche 2
```

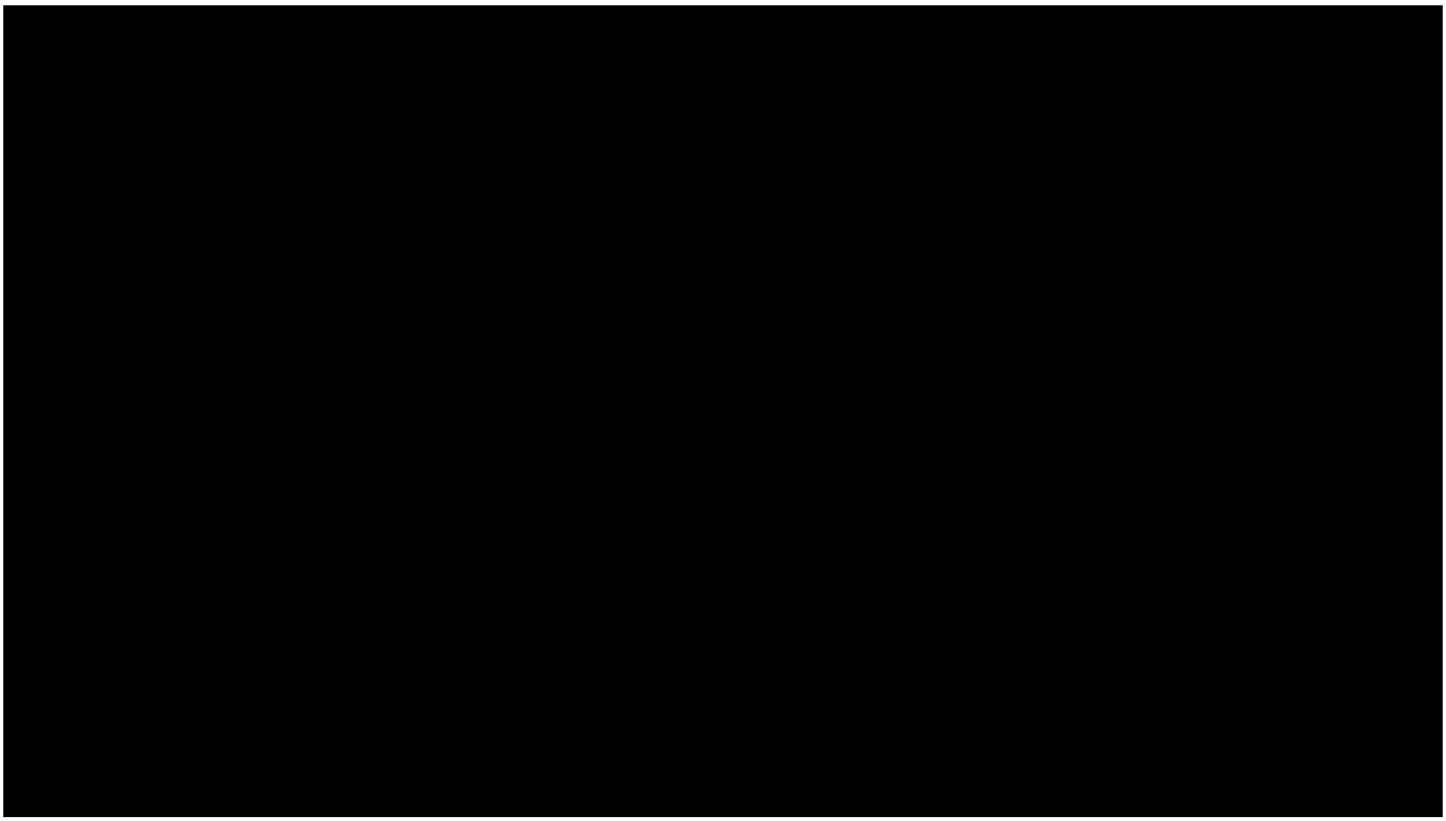


C'est une référence de quoi ? On ne sait pas vers quoi c'est une référence. On n'a pas le droit de déclarer des références qui ne réfèrent rien. Ça perdrait tout son sens. De la même façon, une référence ne peut être liée qu'à un seul objet. C'est aussi une grosse différence par rapport aux pointeurs. Je reprends notre exemple avec notre variable i. Ici, une référence ri qui référence cette variable i Ici, je déclare une autre variable j que j'initialise à 2. Je ne peux pas utiliser ri qui est une référence vers i. Elle a déjà été utilisée. Elle référence un objet i. Je ne peux pas déplacer l'étiquette vers j. Si je faisais le dessin, j'ai la variable i qui contient une valeur, on ne sait pas laquelle, ça n'a pas été initialisé. Je lui ai attaché la référence ri. Puis j'ai déclaré une autre variable j que j'ai initialisée à 2. Je ne peux pas déplacer une référence, ça n'est pas possible. On n'a pas le droit de faire ce genre de chose. Et pourtant la ligne, ici, ri = j compile. Elle est tout à fait licite en C++. Qu'est-ce que ça fait à ce moment là ? Revenons aux fondamentaux, ri, qu'est-ce que c'est ? C'est un autre nom pour i. C'est donc comme si j'avais écrit i = j Si j'avais écrit i = j, qu'est-ce qu'il se passe ? On recopie la valeur de j dans i et donc ici, de la même façon, on va recopier la valeur de j dans ri. Donc, cette ligne ici, ri = j, ne veut pas du tout dire,

notes

résumé

2m 37s



qu'on déplace l'étiquette de ri vers j et qu'on a une autre référence vers l'objet j mais veut dire que l'on fait une copie de l'objet j vers l'objet dont un des noms possibles est ri. La preuve, si je modifie la valeur de j, par exemple si je lui mets la valeur 3, alors à ce moment là et si j'affiche ensuite ici la valeur de i vous allez voir que ça va afficher... Je vous encourage à tester ce programme de votre côté. Vous allez voir que ça affiche 2. 2 parce que c'est bien la valeur à laquelle on a affecté la variable, qui s'appelle et à la fois i et à la fois ri. et à la fois ri.

notes

résumé

4m 25s