Support de cours

Cours:

Initiation à la programmation (en C++)

Vidéo:

Itérations - approfondissement et exemples (partie 1)

Concepts (extraits des sous-titres générés automatiquement) :

Instruction d'incrémentation. Boucle for. Variable i. Fin de la ligne. Instruction for. Fois vraie. Exemples d'autres formes de boucles. Sortir de la boucle. Lignes de codes. Intérieur de la boucle. Tour de boucle. Nouvelle version du code. Première instruction cout. Tel cas. K égal.

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

vers la vidéo

EPFL

Itérations : approfondissement et exemples
(Partie 1)

Initiation à la programmation (C++)

Vincent Lepetit, Jean-Cédric Chappelier et Jamila Sam

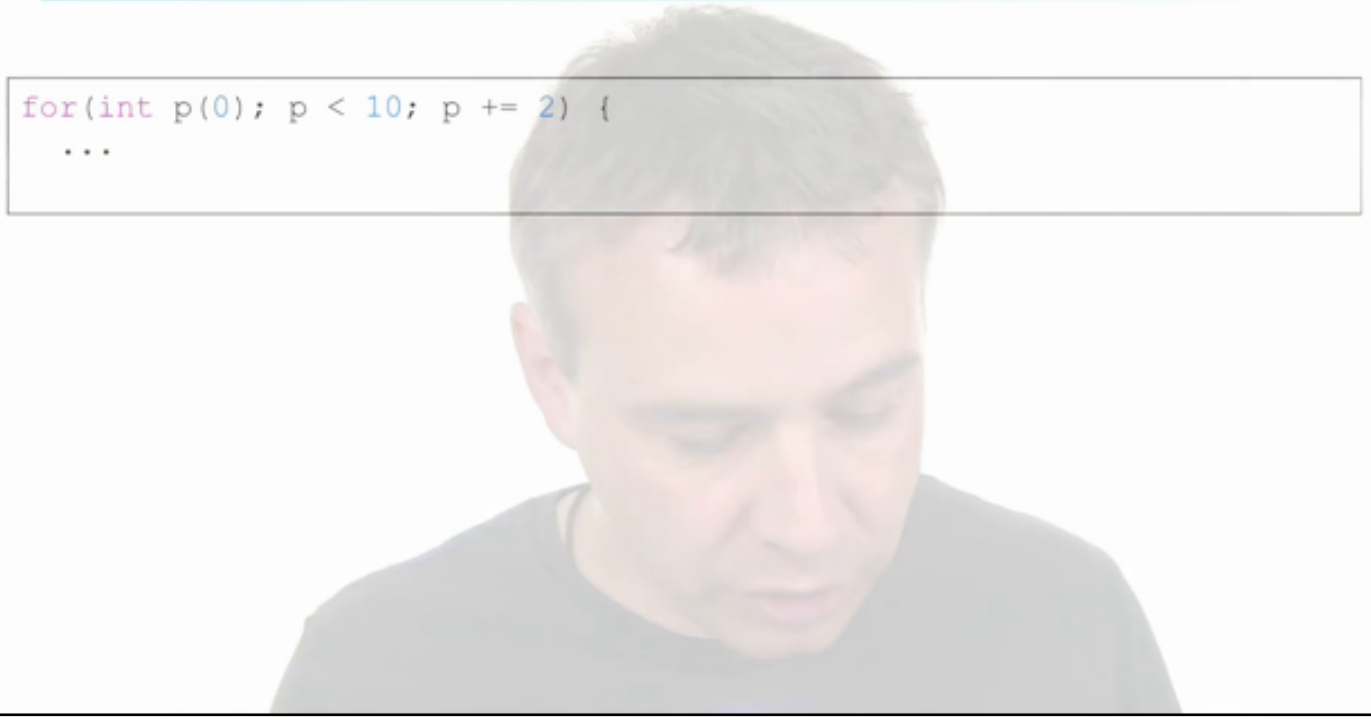
...

notes

résumé

0m 0s

Exemples d'autres formes de boucles `for`



Commençons par voir des exemples d'autres formes de

notes

résumé

0m 1s

Exemples d'autres formes de boucles `for`

```
for(int p(0); p < 10; p += 2) {
   ...
```

boucles for. Dans cette boucle for ici, j'ai déclaré une variable que j'ai appelée p, qui va être initialisée à zéro. La condition est p strictement inférieur à 10, et l'instruction d'incrémentation est p += 2. Alors je vous rappelle qu'écrire

notes

résumé

0m 8s

Exemples d'autres formes de boucles `for`

```
for(int p(0); p < 10; p += 2) {
   ...
```

p += 2, c'est équivalent à écrire p = p + 2, p va donc être initialisé à zéro. À la fin du premier tour de boucle, on va ajouter 2 à p ; donc p va prendre la valeur 2. Ensuite p va prendre la valeur 4 ; ainsi de suite, jusqu'à ce qu'il prenne la valeur 10. Dans ce cas, la condition va devenir fausse, et on va sortir de la boucle. Donc dans

notes

résumé

0m 26s

Exemples d'autres formes de boucles `for`

EPFL

```
for(int p(0); p < 10; p += 2) {
   ...
```

p += 2
p = p + 2

p : 0 2 4 6 8

la boucle, p va prendre les valeurs 0, 2, 4,

notes

résumé

0m 57s

Exemples d'autres formes de boucles `for`

EPFL

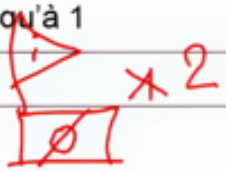
```
for(int p(0); p < 10; p += 2) {
  ...
```

la variable `p` prendra les valeurs de 0, 2, 4, 6, 8 (`p += 2` est équivalent à `p = p + 2`)

```
for(int k(10); k > 0; --k) {
  ...
```

la variable `k` prendra les valeurs 10, 9, 8 ... jusqu'à 1

```
for(int i(0); i >= 0; ++i) {
  ...
```



6, et 8. Considérons maintenant cette instruction for, qui déclare une variable k initialisée à 10, la condition est maintenant k est strictement supérieur à zéro, donc notez bien que le sens de comparaison a changé ici, et l'instruction d'incrémentation est --k, qui est équivalente à k égal k moins 1, c'est-à-dire qu'on va retrancher 1 à la variable k à chaque tour de boucle. Donc k est initialisé à la valeur 10, à la fin du premier tour de boucle, on va retrancher 1 à k, donc k va prendre la valeur 9 ; ensuite k prendra la valeur 8, jusqu'à ce que k prenne la valeur 0. Dans ce cas, la condition ici deviendra fausse, et on va sortir de la boucle. Donc à l'intérieur de la boucle, k prend les valeurs 10, 9, etc. jusque 1. Considérons maintenant ce troisième exemple qui déclare une variable i initialisée à zéro, la condition est i supérieur ou égal à zéro, et l'instruction d'incrémentation est ++i, c'est-à-dire qu'on va ajouter 1 à la variable i à chaque tour de boucle. i commence donc avec la valeur zéro. À la fin du premier tour de boucle, on va ajouter 1 à i. i va donc prendre la valeur 1, cette condition va être vraie. On va donc continuer et à la fin du deuxième tour de boucle, on va ajouter 1 à i qui va prendre la valeur 2. Cette condition va être encore une fois vraie, et on va continuer. Et vous pouvez constater que, du moins dans le principe,

notes

résumé

1m 3s

Pas de point-virgule (;) à la fin de l'instruction `for`

Les instructions suivantes n'affichent qu'une seule fois la chaine "bonjour":

```
for(int i(0); i < 10; ++i);
  cout << "bonjour" << endl;
```

Le point-virgule seul est considéré comme une instruction (qui ne fait rien).

i va prendre toutes les valeurs positives possibles sans que cette condition devienne fausse, et la boucle va être répétée indéfiniment. Alors pourquoi «du moins dans le principe» ? Pour des raisons techniques que je ne vais pas détailler, ce n'est pas complètement vrai ; mais on va tout de même répéter la boucle un très grand nombre de fois. Une boucle for peut donc ne pas s'arrêter, c'est ce qui se produit quand la condition est toujours vraie. Plusieurs causes sont possibles. Par exemple comme dans l'exemple précédent, peut-être qu'on s'est trompé sur la condition. Alors, encore une fois, dans un tel cas, en fait la boucle va tout de même s'arrêter, mais après un très grand nombre d'itérations. Une deuxième cause possible est de se tromper sur l'instruction de l'incrémentation, comme ici. J'ai ici incrémenté la variable j, alors que ma condition est sur la variable i. Si je ne modifie pas la valeur de la variable i dans le bloc de la boucle, comme i est initialisé à zéro, cette condition va être toujours vraie. Donc en pratique, si vous vous retrouvez dans un tel cas, où une boucle for ne s'arrête pas, vous pouvez toujours utiliser la combinaison Control C pour arrêter votre programme. Considérons maintenant quelques erreurs commises fréquemment par des débutants. Notez tout d'abord qu'il n'y a pas de point-virgule à la fin de la ligne qui commence par le mot-clé for, c'est-à-dire qu'il n'y a pas de point-virgule ici. Il se trouve que si vous mettez tout de même un point-virgule, le compilateur va accepter votre programme, mais votre programme va se comporter de façon un peu surprenante, et en fait, ce code, avec un point-virgule,

notes

résumé

2m 56s

Attention aux accolades

```
for(int i(0); i < 5; ++i)
  cout << "i = " << i << endl;
  cout << "Bonjour" << endl;
```

va n'afficher qu'une seule fois le mot "bonjour". Pourquoi ? Parce que le point-virgule qui, ici est considéré comme une instruction ; une instruction qui ne fait rien. C'est-à-dire que ce code ici va être interprété comme, tout d'abord, une boucle for dont le corps de la boucle est constitué par le point-virgule. Donc cette boucle for va répéter dix fois l'instruction vide, c'est-à-dire ne rien faire du tout. Et l'instruction avec le cout va être considérée comme en dehors de la boucle for, et donc va n'être exécutée qu'une seule fois. Considérez maintenant ces trois lignes de

notes

résumé

4m 49s

Attention aux accolades

```
for(int i(0); i < 5; ++i)
    cout << "i = " << i << endl;
    cout << "Bonjour" << endl;
```

affiche:

```
i = 0
i = 1
i = 2
i = 3
i = 4
Bonjour
```

codes, ces lignes vont en fait produire l'affichage suivant : c'est-à-dire que la première instruction cout va être répétée cinq fois, et la deuxième instruction cout va n'être exécutée qu'une seule fois. Alors, pourquoi ?

notes

résumé

5m 35s

Attention aux accolades

```
for(int i(0); i < 5; ++i){
  cout << "i = " << i << endl;
  cout << "Bonjour" << endl;
}
```

affiche:

```
i = 0
i = 1
i = 2
i = 3
i = 4
Bonjour
```

i = 0
Bonjour
i = 1
Bonjour
...

Interprétation:

```
for(int i(0); i < 5; ++i)
  cout << "i = " << i << endl;
cout << "Bonjour" << endl;
```

Parce qu'en fait, ce code-là est interprété de la façon suivante : c'est-à-dire que la première instruction cout à l'intérieur de la boucle for, la deuxième instruction cout après la boucle for, et donc ne va pas être répétée. En fait, attention ! Il ne suffit pas de décaler ces lignes pour qu'elles soient toutes les deux à l'intérieur de la boucle for. Si vous voulez que ces deux lignes soient effectivement répétées, il faut rajouter une accolade avant la première instruction, et une accolade fermante après la dernière instruction. Donc, cette nouvelle version du code va effectivement produire l'affichage i=0 Bonjour, i=1 Bonjour, etc.

notes

résumé

5m 53s

Evitez de modifier une variable compteur à l'intérieur d'une boucle `for`

```
for(int i(0); i < 10; ++i) {
  ...
  if (...)
    --i; // !!!
}
```

Enfin, évitez de modifier la variable qui sert à contrôler le nombre

notes

résumé

6m 49s

Evitez de modifier une variable compteur à l'intérieur d'une boucle `for`

```
for(int i(0); i < 10; ++i) {
  ...
  if (...)
    --i; // !!!
}
```

1. Ça ne fera sans doute pas ce que vous voulez : n'oubliez pas que la boucle `for`, de son côté, incrémente la variable `i`.

de tours de boucle à l'intérieur du corps de la boucle ; c'est-à-dire évitez de faire comme dans cet exemple, qui déclare au début de la boucle for une variable i, et qui modifie la variable i ici, dans le corps de la boucle. Alors pourquoi ?

notes

résumé

6m 50s

Tout d'abord parce ça ne va sans doute pas faire ce que vous voulez. N'oubliez-pas que la boucle for va modifier également, de son côté, la variable i à l'aide de l'instruction d'incrémentation, et, deuxièmement, un relecteur risque de ne pas s'apercevoir que vous modifiez la valeur de la variable i à l'intérieur du corps de la boucle. l'intérieur du corps de la boucle.

notes

résumé

7m 14s