Support de cours

Cours:

Initiation à la programmation (en C++)

Vidéo:

Expressions (partie 1)

Concepts (extraits des sous-titres générés automatiquement) :

Symbole étoile. Type double. Déclaration de cette façon. Valeur littérale. Côté des opérateurs. Cas de la division. Règle générale. Cadres opérateurs usuels. Notation anglo-saxonne. Propre type. Valeurs littérales. Valeur de la variable n. Droite du signe. Valeur. Expressions.

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

vers la vidéo

EPFL

Expressions et opérateurs
(Partie 1)

Initiation à la programmation (C++)

Vincent Lepetit, Jean-Cédric Chappelier et Jamila Sam

...

notes

résumé

0m 0s

Revenons maintenant sur les expressions et opérateurs

notes

résumé

0m 1s

Expressions et Opérateurs

A droite du signe égal dans une affectation se trouve une **expression**:

```
nom_de_variable = expression;
```

Une expression calcule une valeur, qui doit être de même type que la variable.

Une expression peut être simplement une valeur littérale:

```
4
3.14
```

ou une formule qui met en oeuvre des opérateurs:

```
n * n
n * (n + 1) + 3 * n - 2
```

que vous avez déjà vus dans les vidéos précédentes. Une expression apparaît par exemple dans une affectation, comme ici.

notes

résumé

0m 5s

Expressions et Opérateurs

A droite du signe égal dans une affectation se trouve une **expression**:

```
nom_de_variable = expression;
```

Une expression calcule une valeur, qui doit être de même type que la variable.

Une expression peut être simplement une valeur littérale:

```
4
3.14
```

ou une formule qui met en oeuvre des opérateurs:

```
n * n
n * (n + 1) + 3 * n - 2
```

Dans ce cas, l'expression est forcément à droite du signe =, puisqu'elle calcule une valeur, cette valeur doit être du même type que la variable, qui se trouve à gauche du signe =. Puisqu'on va donner la valeur calculée par l'expression à la variable. Une expression peut être simplement une valeur littérale comme 4 ou 3,14 Notez au passage qu'on n'écrit pas 3 virgule 14 mais bien 3 point 14 puisque le C++ utilise la notation anglo-saxonne. Mais en règle générale, une expression peut être une formule, comme ici, où je multiplie la valeur de la variable n avec elle-même, j'obtiens donc n^2, simplement. La multiplication se note avec le symbole étoile *

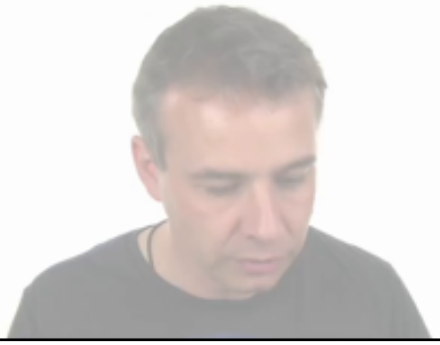
notes

résumé

0m 15s

Les valeurs littérales et leurs types

- `1` est de type `int`;



et je peux avoir dans mon expression d'autres symboles mathématiques comme l'addition, la soustraction. Je peux également utiliser des parenthèses, c'est donc quelque chose d'assez général.

notes

résumé

1m 13s

Les valeurs littérales et leurs types

- `1` est de type `int`;
- `1.0` est de type `double`;

Il est important d'être conscient, quand on écrit des expressions, que les valeurs littérales ont leur propre type, exactement comme les variables. Par exemple, si j'écris 1 simplement, ce 1 est une valeur littérale de type int. Si maintenant, j'écris 1.0, ce 1.0 sera de type double.

notes

résumé

1m 25s

Les valeurs littérales et leurs types

- `1` est de type `int`;
- `1.0` est de type `double`;
- `1.` est équivalent à `1.0`, et donc de type `double`. On peut écrire:

```
double x(1.);
```
au lieu de
```
double x(1.0);
```

Il vaut mieux écrire `1.0` plutôt que `1.` puisque c'est plus lisible

- On peut utiliser la notation scientifique, par exemple écrire `2e3` pour 2×10^3, c'est-à-dire 2000.

De façon générale: aeb vaut $a \times 10^b$. Par exemple:

```
double x(1.3e3);
```
→ `x` vaut $1.3\times10^3 = 1.3 \times 1000 = 1300$

```
double y(1.3e-3);
```
→

Notez au passage que l'on peut écrire tout simplement 1. tout court, au lieu de 1.0 et ce 1. sera donc aussi de type double, ça veut dire que je peux écrire une déclaration de cette façon-ci avec seulement 1., qui est équivalente à cette déclaration-ci mais évidemment, écrire 1.0 sera toujours plus lisible que d'écrire 1. tout court, donc préférez la notation 1.0 On peut également utiliser la notation scientifique, c'est-à-dire que l'on peut écrire 2e3 pour 2 multiplié par 10 puissance 3 c'est-à-dire 2000, donc 2e3 vaut 2000. Et de façon générale, aeb vaut a multiplié par 10 puissance b. Par exemple, dans cette déclaration, j'initialise la variable x à 1,3 multiplié par 10 puissance 3, c'est-à-dire 1300

notes

résumé

1m 49s

Opérateurs

On dispose des 4 opérateurs usuels:

+ pour l'addition;

− pour la soustraction;

* pour la multiplication;

/ pour la division.

Dans cette déclaration-ci, j'initialise ma variable y à 1,3 multiplié par 10 puissance -3 c'est-à-dire que y vaut 0,0013. Du côté des opérateurs, on dispose des cadres opérateurs usuels, c'est-à-dire l'addition qui se note avec le symbole + (plus), la soustraction qui se note avec le symbole - (moins), j'ai dit que la multiplication se notait avec le caractère * (étoile) et la division se note avec le caractère / (slash).

notes

résumé

3m 1s

Opérateurs

On dispose des 4 opérateurs usuels:

\+ pour l'addition;

\- pour la soustraction;

* pour la multiplication;

/ pour la division.

Attention: si la division se fait entre `int`, il s'agit de la division entière.
Par exemple:
`1 / 2` vaut `0`
`5 / 2` vaut `2`

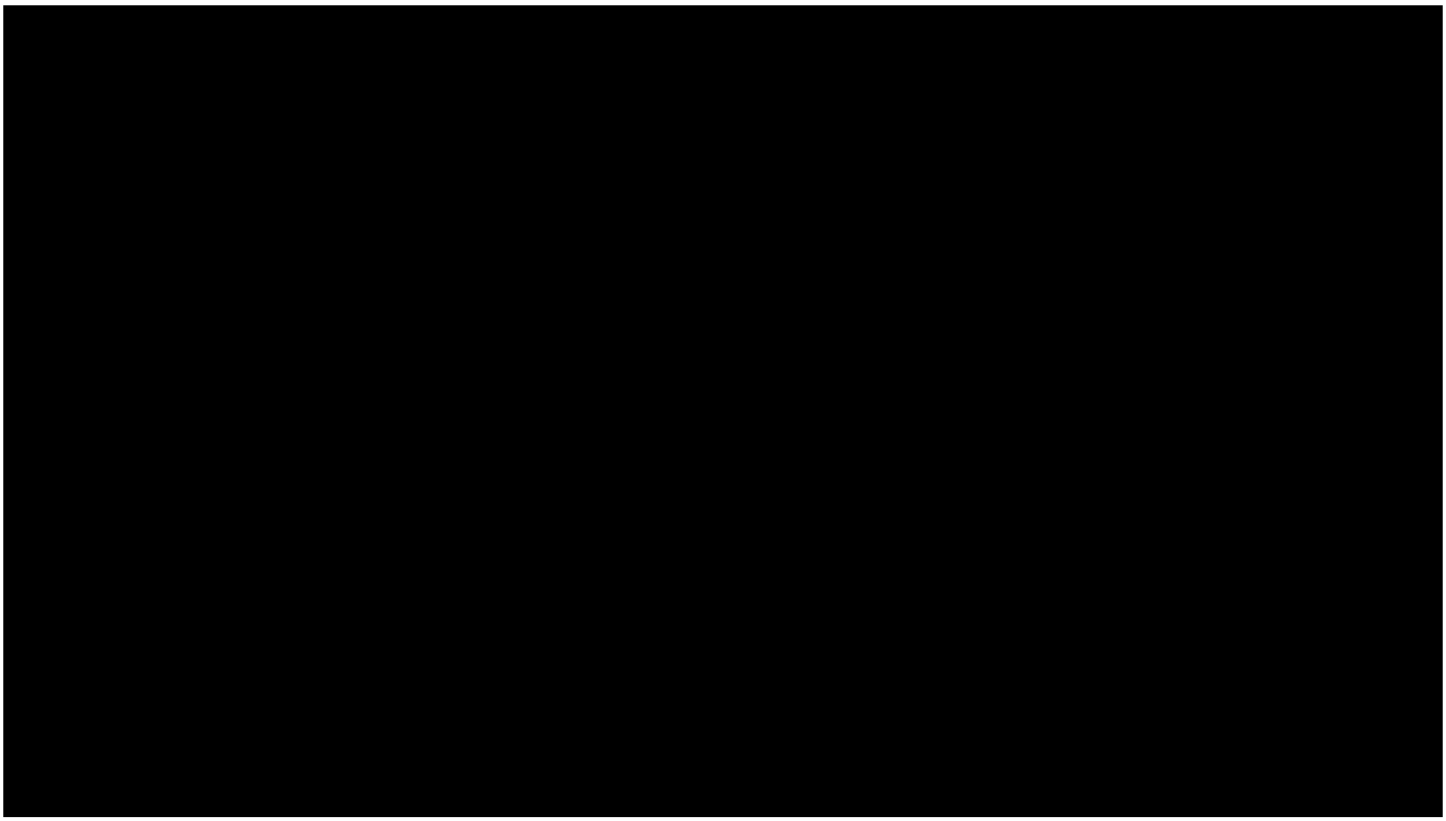
mais
`1 / 2.0` vaut bien `0.5`

Mais attention, il y a un piège dans le cas de la division, si les deux valeurs qui interviennent dans la division sont de type int, il s'agit de la division entière. Par exemple, dans cette division, 1 et 2 sont deux valeurs littérales qui sont toutes les deux de type int et il s'agit donc de la division entière et on va obtenir 0. Alors, pourquoi ? Parce que 1 est égal à 0 fois 2 et il reste 1 donc le résultat de la division entière est ce 0. Un autre exemple dans le cas de 5 divisé par 2, je vais obtenir 2,

notes

résumé

3m 37s



parce que 5 est égal à 2 fois 2 et il reste 1 et le résultat de la division est ce 2. Si en revanche, une des deux valeurs qui apparaissent dans la division est de type double, l'autre valeur va tout d'abord être convertie en double, c'est-à-dire que ce 1 va être converti en un double que je peux noter 1.0 On va obtenir la division 1.0 divisé par 2.0 et donc obtenir, comme on aurait pu s'y attendre, 0.5. 0.5.

notes

résumé

4m 13s