Support de cours

Cours:

Initiation à la programmation (en C++)

Vidéo:

Expressions (partie 4)

Concepts (extraits des sous-titres générés automatiquement) :

Fonctions mathématiques. Sinus d'un angle. Fonction sin. Fonctions trigonométriques. Dièse include. Fonction puissance. Début du programme. Aide de cette expression. Logarithme népérien. X puissance. Lettres capitales. Fonction exponentielle. Sinus de cet angle. Base des logarithmes naturels. Constantes mathématiques.

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

vers la vidéo

EPFL

Expressions et opérateurs
(Partie 4)

Initiation à la programmation (C++)

Vincent Lepetit, Jean-Cédric Chappelier et Jamila Sam

...

notes

résumé

0m 0s

Fonctions mathématiques

EPFL

La bibliothèque standard du C++ fournit les fonctions mathématiques usuelles.
Par exemple:

```
#include <iostream>
#include <cmath>

using namespace std;

int main()
{
  double angle;
  double s;

  angle = 10 * 3.14159 / 180;
  s = sin(angle);
```

il faut ajouter cette ligne pour pouvoir utiliser les fonction mathématiques

→ `sin(angle)` calcule le sinus de la valeur contenue dans la variable `angle`, en radians.

On peut également utiliser des fonctions mathématiques dans des expressions, pour cela il faut ajouter la ligne #include (dièse include) cmath

notes

résumé

0m 1s

Fonctions mathématiques

La bibliothèque standard du C++ fournit les fonctions mathématiques usuelles.
Par exemple:

```
#include <iostream>
#include <cmath>

using namespace std;

int main()
{
  double angle;
  double s;

  angle = 10 * 3.14159 / 180;
  s = sin(angle);
```

il faut ajouter cette ligne pour pouvoir utiliser les fonction mathématiques

→ `sin(angle)` calcule le sinus de la valeur contenue dans la variable `angle`, en radians.

au début du programme, et si par exemple, je veux calculer le sinus d'un angle, je vais pouvoir utiliser la fonction sin comme ici. Au passage, les fonctions trigonométriques en C++ attendent des angles en radians. Donc si j'ai un angle de 10 degrés, comme ici, je vais le convertir en radians à l'aide de cette expression

notes

résumé

0m 13s

Fonctions mathématiques

EPFL

- `sin` les fonctions trigonométriques fonctionnent en radians
- `cos`
- `tan`
- `asin` sinus inverse ou arc sinus
- `acos`
- `atan`
- `atan2` `atan2(y, x)` fournit la valeur de l'arc-tangente de `y / x`
- `sinh` sinus hyperbolique ou *sh*
- `cosh`
- `tanh`
- `exp`
- `log` logarithme népérien ou *ln*
- `log10` logarithme à base 10 ou *log*
- `pow` `pow(x, y)` fournit la valeur de x^y
- `sqrt` racine carrée
- `ceil` `ceil(x)` renvoie le plus petit entier qui ne soit pas inférieur à `x`: `ceil(2.6) = 3`
- `floor` `floor(x)` renvoie le plus grand entier qui ne soit pas supérieur à `x`: `floor(2.6) = 2`
- `abs` valeur absolue

et je vais donner sa valeur à la fonction sin pour qu'elle me renvoie le sinus de cet angle de 10 degrés. Voilà, ce n'est pas plus difficile que ça. Voici une liste assez complète des fonctions mathématiques dont on dispose en C++, les fonctions les plus intéressantes sont sans doute les fonctions trigonométriques sin, cos et tan qui attendent, je vous le rappelle, des angles en radians, la fonction exponentielle se note exp, le logarithme népérien qui se note ln en mathématiques, se note log en C++. La fonction puissance "x puissance y" se note pow: x, y en C++,

notes

résumé

0m 37s

Constantes mathématiques

Bien que non standard, les constantes suivantes sont souvent définies par les compilateurs usuels:

- `M_PI` π
- `M_E` e (= 2.71828, base des logarithmes naturels);
- ...

la racine carrée se note sqrt pour square root, la valeur absolue se note abs. On dispose également de plusieurs constantes mathématiques ;

notes

résumé

1m 25s

Constantes mathématiques

Bien que non standard, les constantes suivantes sont souvent définies par les compilateurs usuels:

- `M_PI` π
- `M_E` e (= 2.71828, base des logarithmes naturels);
- ...

alors, même si elle ne sont pas définies officiellement par le standard du C++, la plupart des compilateurs les définissent.

notes

résumé

1m 37s

```cpp
#include <iostream>
#include <cmath>
using namespace std;

int main()
{
  double angle_en_degres;

  cout << "Entrez un angle en degres: " << endl;
  cin >> angle_en_degres;

  double angle_en_radians(M_PI * angle_en_degres / 180);

  cout << "Sa valeur en radians vaut " << angle_en_radians << endl;
  cout << "Son cosinus vaut " << cos(angle_en_radians) << endl;

  return 0;
}
```

$$\text{angle en radians} = \frac{\pi \text{ angle en degrés}}{180}$$

Par exemple, nous avons Pi qui se note M_PI (M underscore PI) en lettres capitales, et e qui est la base des logarithmes naturels, qui s'écrit M_E (M underscore E) également en lettres capitales. Par exemple, si j'ai un angle exprimé en degrés

notes

résumé

1m 44s

EPFL

```cpp
#include <iostream>
#include <cmath>
using namespace std;

int main()
{
  double angle_en_degres;

  cout << "Entrez un angle en degres: " << endl;
  cin >> angle_en_degres;

  double angle_en_radians(M_PI * angle_en_degres / 180);

  cout << "Sa valeur en radians vaut " << angle_en_radians << endl;
  cout << "Son cosinus vaut " << cos(angle_en_radians) << endl;

  return 0;
}
```

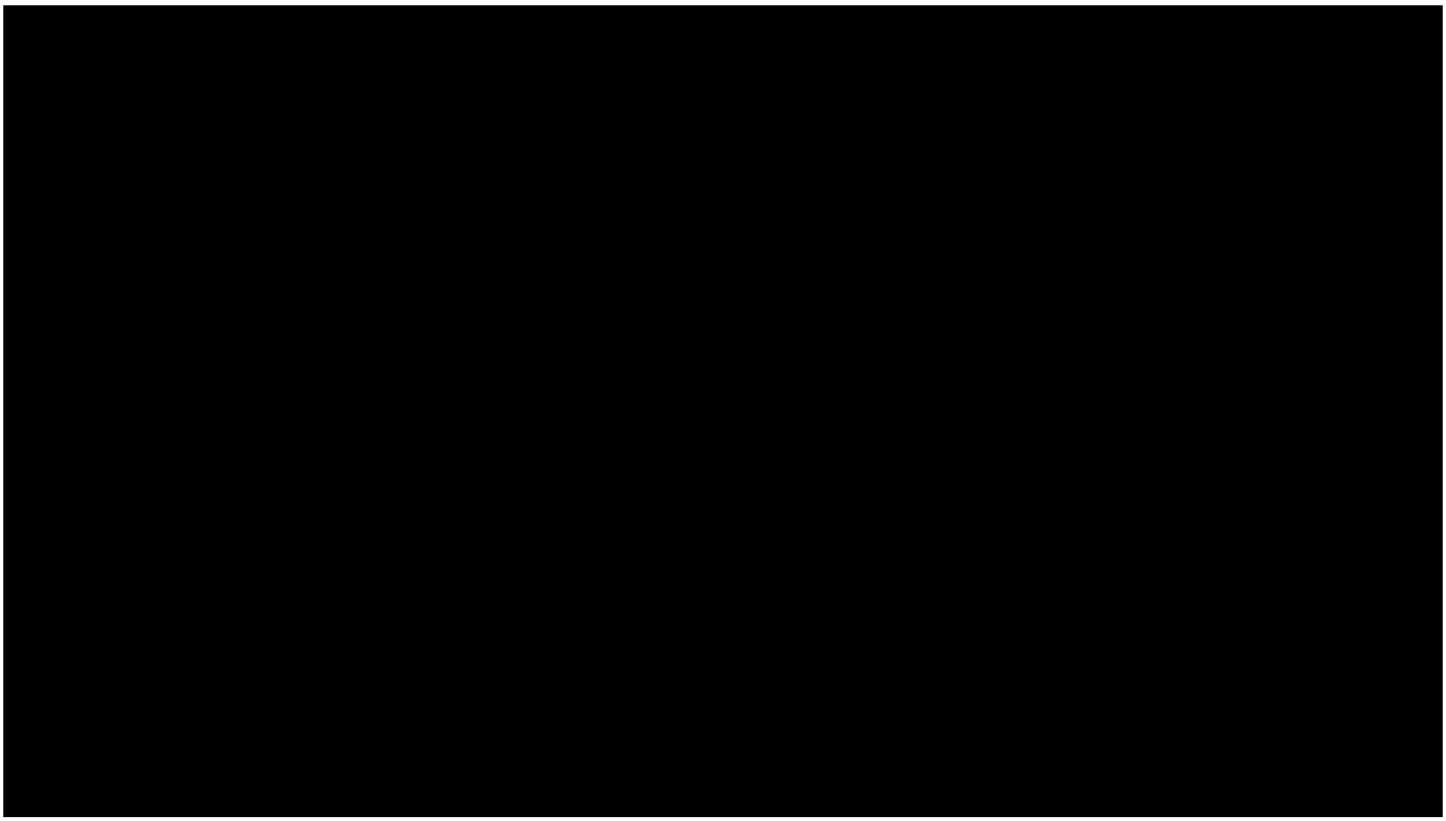
$$\text{angle en radians} = \frac{\pi \text{ angle en degrés}}{180}$$

que je veux convertir en radians, je peux utiliser cette formule-ci. C'est-à-dire multiplier l'angle en degrés par Pi et diviser par 180. En C++, cela va s'écrire de cette façon-ci, j'obtiens la valeur de Pi en écrivant M_PI je peux multiplier par l'angle, diviser par 180

notes

résumé

2m 1s



et notez au passage que comme cette expression est de type double, je n'ai pas de problème de la division entière lors de cette division. entière lors de cette division.

notes

résumé

2m 25s