Support de cours

Cours:

Initiation à la programmation (en C++)

Vidéo:

Arguments par défaut et surcharge (partie 1)

Concepts (extraits des sous-titres générés automatiquement) :

Paramètres de cette fonction. Prototype de fonctions. Deuxième paramètre de la fonction. Ligne de plusieurs caractères. Tel paramètre. Appel de ma fonction. Déclaration du paramètre. Deuxième argument. Type char. Premier paramètre. Ligne de cette façon. Compilateur c. Seul argument. Prototype de la fonction. Liste des paramètres de la fonction.

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

vers la vidéo

EPFL

Fonctions : arguments par défaut et surcharge
(Partie 1)

Initiation à la programmation (C++)

Vincent Lepetit, Jean-Cédric Chappelier et Jamila Sam

...

notes

résumé

0m 0s

Arguments par défaut

EPFL

Lors de son prototypage, une fonction peut donner des valeurs par défaut à ses paramètres.
Il n'est alors pas nécessaire de fournir d'argument à ces paramètres lors de l'appel de la fonction.

La syntaxe d'un paramètre avec valeur par défaut est :

```
type identificateur = valeur
```

Attention : Les paramètres avec valeur par défaut doivent apparaître en dernier dans la liste des paramètres d'une fonction.

Terminons par deux notions qui sont assez facile à comprendre. Tout d'abord, les arguments par défaut. On peut, dans un prototype de fonctions, donner des valeurs par défaut à certains des paramètres de cette fonction. Et dans ce cas là, il n'est plus nécessaire de fournir de valeurs, c'est à dire d'arguments à ces paramètres quand on appelle la fonction. Pour pouvoir définir un tel paramètre avec une valeur par défaut, il suffit, lors de la déclaration du paramètre dans le prototype de la fonction,

notes

résumé

0m 1s

Arguments par défaut : exemple

Exemple :

```
void affiche_ligne(char elt, int nb = 5);

int main() {
  affiche_ligne('*');
  affiche_ligne('+', 8);
  return 0;
}

void affiche_ligne(char elt, int nb) {
  for(int i(0); i < nb; ++i) {
    cout << elt;
  }
  cout << endl;
}
```

Résultat :

```
*****
++++++++
```

Lors de l'appel `affiche_ligne('*')`, la valeur par défaut 5 est utilisée ; c'est strictement équivalent à `affiche_ligne('*', 5)`

Lors de l'appel `affiche_ligne('+', 8)`, la valeur explicite 8 est utilisée.

d'ajouter le signe '=' (égal) suivi de la valeur par défaut qu'on veut donner à ce paramètre. Un peu comme dans une affectation. Et attention, ces paramètres, avec valeur par défaut doivent forcément apparaître en dernier dans la liste des paramètres de la fonction. Prenons l'exemple d'une fonction qui va afficher une ligne de plusieurs caractères. Le prototype d'une telle fonction se trouve ici, le nom de la fonction est affiche_ligne. Cette fonction a deux paramètres : le premier paramètre va permettre de définir le caractère qu'on veut répéter sur la ligne. Le type char, sert simplement à dire que le paramètre elt va contenir un caractère. Le deuxième paramètre de la fonction : le nombre de fois qu'on veut répéter le caractère, et vous pouvez constater que ce paramètre a une valeur par défaut qui est "5". Ce qui veut dire que quand j'appelle la fonction affiche_ligne de cette façon ci, c'est à dire, avec un seul argument où le premier argument est le caractère '*' (étoile), le paramètre elt va contenir le caractère étoile. Comme, je n'ai pas de deuxième argument à cet appel de ma fonction affiche_ligne, le deuxième paramètre nb va prendre la valeur par défaut qui est 5 . Et donc, cet appel à la fonction affiche_ligne va m'afficher une ligne de 5 étoiles (*) . Maintenant, le deuxième appel à ma fonction affiche_ligne ici qui a cette fois deux arguments où le premier argument est le caractère '+' (plus) et le deuxième argument est la valeur 8. Dans ce cas là, le paramètre elt va contenir le caractère '+' et le deuxième paramètre nb va non pas prendre la valeur par défaut 5, mais la valeur qui est passée en argument ici, c'est à dire 8. Et donc cet appel à ma fonction affiche_ligne

notes

résumé

0m 37s

Arguments par défaut : Remarques

EPFL

- Les arguments par défaut se spécifient dans le **prototype** et non pas dans la définition de la fonction
- Lors de l'appel à une fonction avec plusieurs paramètres ayant des valeurs par défaut, les arguments omis doivent être les derniers et omis **dans l'ordre** de la liste des paramètres.

Exemple :

```
void f(int i, char c = 'a', double x = 0.0);
```

```
f(1)          → correct (vaut f(1, 'a', 0.0))
f(1, 'b')     → correct (vaut f(1, 'b', 0.0))
f(1, 3.0)     → incorrect !
f(1, , 3.0)   → incorrect !
f(1, 'b', 3.0) → correct
```

va m'afficher huit fois le caractère '+' .

notes

résumé

3m 1s

Arguments par défaut : Remarques

EPFL

- Les arguments par défaut se spécifient dans le prototype et non pas dans la définition de la fonction
- Lors de l'appel à une fonction avec plusieurs paramètres ayant des valeurs par défaut, les arguments omis doivent être les derniers et omis dans l'ordre de la liste des paramètres.

Exemple :

```
void f(int i, char c = 'a', double x = 0.0);
```

```
f(1)          → correct (vaut f(1, 'a', 0.0))
f(1, 'b')     → correct (vaut f(1, 'b', 0.0))
f(1, 3.0)     → incorrect !
f(1, , 3.0)   → incorrect !
f(1, 'b', 3.0) → correct
```

Deux remarques sur cette notion d'arguments par défaut, les arguments aux valeurs par défaut doivent être spécifiés dans le prototype de la fonction et non pas dans la définition de la fonction. De plus, lors de l'appel, à une fonction avec plusieurs paramètres

notes

résumé

3m 5s

Arguments par défaut : Remarques

EPFL

- Les arguments par défaut se spécifient dans le prototype et non pas dans la définition de la fonction
- Lors de l'appel à une fonction avec plusieurs paramètres ayant des valeurs par défaut, les arguments omis doivent être les derniers et omis dans l'ordre de la liste des paramètres.

Exemple :

```
void f(int i, char c = 'a', double x = 0.0);
```

```
f(1)          → correct (vaut f(1, 'a', 0.0))
f(1, 'b')     → correct (vaut f(1, 'b', 0.0))
f(1, 3.0)     → incorrect !
f(1, , 3.0)   → incorrect !
f(1, 'b', 3.0) → correct
```

ayant des valeurs par défaut, les arguments omis doivent être les derniers et omis dans l'ordre de la liste des paramètres. Je vais prendre l'exemple de cette fonction qui a un premier paramètre qui s'appelle "i" qui n'a pas de valeur par défaut, un deuxième paramètre "c" qui a le caractère "a" par défaut. Et un troisième paramètre " x " qui a la valeur 0 par défaut. Si maintenant j'appelle ma fonction " f " de cette façon ci, c'est à dire avec un seul argument, le paramètre "i" va prendre la valeur 1 le paramètre "c" va prendre le caractère par défaut c'est à dire "a", et le paramètre "x" va prendre la valeur, par défaut, la sienne qui est 0. Je peux également appeler ma fonction "f" de cette façon ci, avec deux arguments, "i" va prendre

notes

résumé

3m 24s

Arguments par défaut : Remarques

EPFL

- Les arguments par défaut se spécifient dans le prototype et non pas dans la définition de la fonction
- Lors de l'appel à une fonction avec plusieurs paramètres ayant des valeurs par défaut, les arguments omis doivent être les derniers et omis dans l'ordre de la liste des paramètres.

Exemple :

```
void f(int i, char c = 'a', double x = 0.0);
```

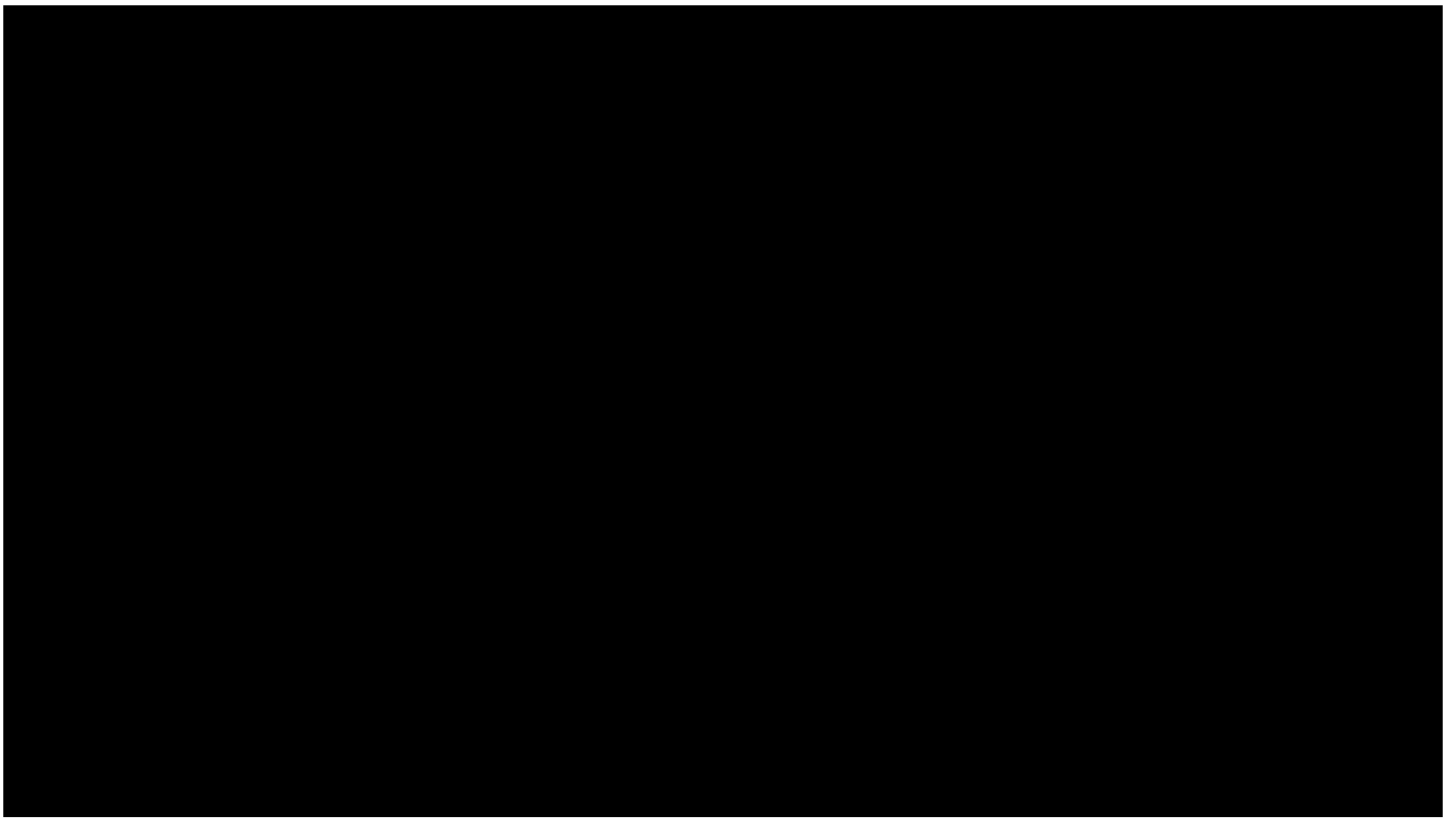
```
f(1)          → correct (vaut f(1, 'a', 0.0))
f(1, 'b')     → correct (vaut f(1, 'b', 0.0))
f(1, 3.0)     → incorrect !
f(1, , 3.0)   → incorrect !
f(1, 'b', 3.0) → correct
```

la valeur du premier argument c'est à dire 1, "c" va contenir le deuxième argument c'est à dire le caractère "b", et le troisième paramètre "x" va continuer à prendre sa valeur par défaut c'est à dire 0. Maintenant si j'essaye d'appeler ma fonction " f " de cette façon là, le compilateur va refuser l'appel parce que le deuxième argument ici est de type double, le compilateur va essayer d'affecter cette valeur de type double au deuxième paramètre "c" qui est de type char. Donc cette appel ci va être refusé par le compilateur. Maintenant on peut être tentés

notes

résumé

4m 37s



d'appeler la fonction " f " de cette façon ci, c'est à dire, en omettant le deuxième paramètre. Malheureusement, le compilateur C++ va refuser un tel appel. Et finalement, si j'essaye d'appeler ma fonction " f " de cette façon ci, avec ces trois arguments, qu'est ce qu'il va se passer ? Le paramètre "i" va prendre la valeur 1, le paramètre "c", va contenir le caractère "b" , et le paramètre "x" va prendre la valeur du troisième argument qui est la valeur 3. qui est la valeur 3.

notes

résumé

5m 25s