Support de cours

Cours:

Initiation à la programmation (en C++)

Vidéo:

Passage des arguments (partie 1)

Concepts (extraits des sous-titres générés automatiquement) :

Séquence vidéo précédente. Types de passage d'arguments. Liste des paramètres. Moment de cet appel. Nom x. Simples valeurs. Argument d'appel de la fonction. Petit programme principal. En-tête de la fonction d'une façon. Séquence précédente. Fonction. Zone locale. Zone mémoire. Symbole particulier. Type de retour void.

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

vers la vidéo

EPFL

Fonctions : passage des arguments
(Partie 1)

Initiation à la programmation (C++)

Vincent Lepetit, Jean-Cédric Chappelier et Jamila Sam

...

notes

résumé

Dans une séquence vidéo précédente, nous avons examiné les cinq étapes

notes

résumé

0m 1s

Le passage des arguments (1)

EPFL

Que peut-on garantir sur la valeur d'une variable passée en argument lors d'un appel de fonction ?

Par exemple :

```
void f(int x) {
  x = x + 1;
  cout << "x=" << x;
}
int main() {
  int val(1);
  f(val);
  cout << " val=" << val << endl;
  return 0;
}
```

Que vaut `val` ? Qu'affiche le programme ?

qui se déroulent dans le cas le plus général lorsqu'on appelle une fonction Dans les exemples vus jusqu'alors, les arguments passés à la fonction étaient soit de simples valeurs, soit des expressions à évaluer. Nous allons aujourd'hui examiner un peu plus finement ce qui se passe lorsque les arguments passés à la fonction sont des variables. Partons d'un exemple concret. Nous avons ici un petit programme principal dont la première instruction consiste à déclarer une variable « val » et l'initialiser avec la valeur 1. La 2e instruction de ce programme appelle la fonction « f » qui est définie ici, en lui fournissant justement en argument, la variable « val » que nous venons juste de déclarer. Nous avons vu dans une séquence précédente qu'au moment de cet appel il y a une mise en correspondance entre l'argument d'appel de la fonction et la liste des paramètres attendus par la fonction. Donc ici nous avons deux entités, nous avons une variable « x » utilisée par la fonction. Et il se trouve que la fonction « f » altère cette variable, la modifie en l'incrémentant d'une unité. La question que nous nous posons maintenant est la suivante : Est-ce que « x » et « val » sont la même zone mémoire ?

notes

résumé

0m 5s

Le passage des arguments (2)

On distingue 2 types de passages d'arguments :

passage par valeur

passage par référence

```
void f (int x){
    x=x+1;
}
....
int main (){
    int val(1);
    f(val);
}
```

Ce qui veut dire concrètement que l'altération de "x" aura une incidence sur « val » ou non.

notes

résumé

1m 13s

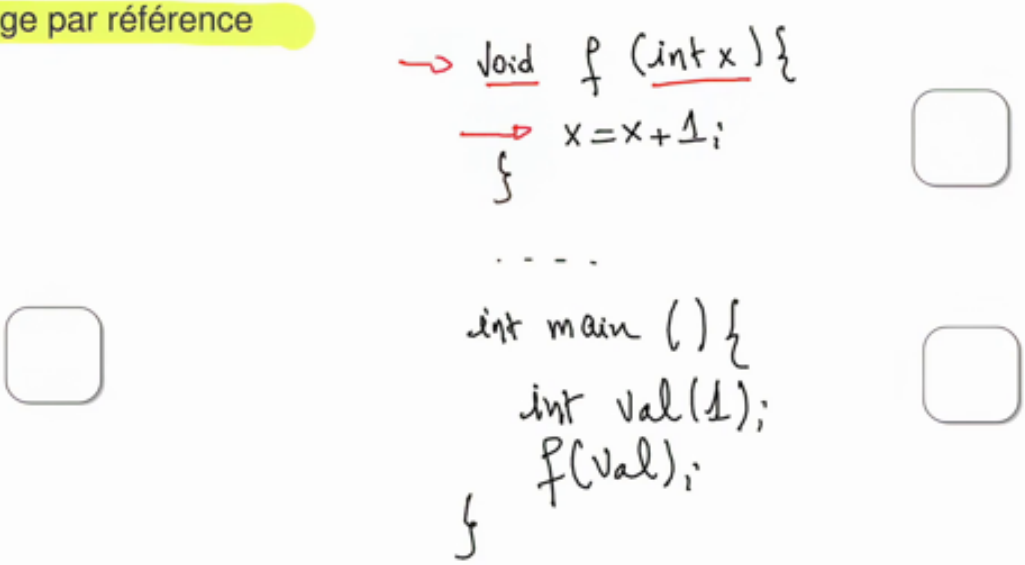
Le passage des arguments (2)

EPFL

On distingue 2 types de passages d'arguments :

passage par valeur

passage par référence



Il existe en programmation deux types de passage d'arguments. Le passage par valeur et le passage par référence. Donc examinons sur le même exemple que vu précédemment ce qui se produit dans l'un ou l'autre des cas. Donc nous avons une petite fonction « f » qui prend en paramètre un entier « x » Elle a pour type de retour void ce qui veut dire simplement qu'elle ne retourne aucune valeur, d'ailleurs il n'y a pas d'instruction return.

notes

résumé

1m 20s

Le passage des arguments (2)

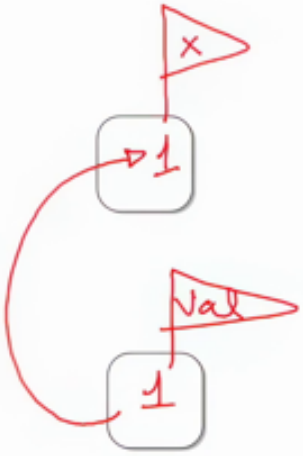
EPFL

On distingue 2 types de passages d'arguments :

passage par valeur

passage par référence

```
void f (int x){
    x=x+1;
}
....
int main (){
    int val(1);
    f(val);
}
```



Et ce que réalise cette fonction est d'incrémenter la valeur de son paramètre « x » égal à « x+1 ». Si une fonction utilise le passage par valeur, cela signifie que le paramètre « x » de la fonction correspond à une zone locale à la fonction. Ce qui signifie que, au moment où on invoque la fonction en lui passant en argument une variable, variable qui est définie à un autre endroit du programme comme par exemple ici, dans le programme principal. Au moment où on appelle la fonction, on va copier la valeur de la variable passée en argument dans la zone locale à la fonction.

notes

résumé

1m 49s

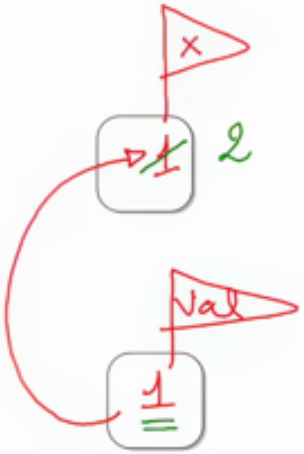
Le passage des arguments (2)

On distingue 2 types de passages d'arguments :

passage par valeur

passage par référence

```
f(int & x)
void f (int x){
  x=x+1;
}
....
int main (){
  int val(1);
  f(val);
}
```



Ce qui signifie que lorsque j'exécute ce traitement, je vais altérer la zone locale dans laquelle est stockée la valeur de la variable passée en argument. Ceci correspond au passage par valeur. Pour indiquer que l'on souhaite utiliser un passage par référence, on va spécifier l'en-tête de la fonction d'une façon un peu particulière en C++ Donc on va indiquer par un symbole particulier, le « & » que la fonction « f » travaille par référence. Lorsqu'on définit l'en-tête de la fonction de cette façon là, on indique qu'au moment de l'appel

notes

résumé

2m 25s

Le passage des arguments (2)

EPFL

On distingue 2 types de passages d'arguments :

passage par valeur

passage par référence

la variable locale associée à un argument passé par référence correspond à une référence sur l'objet associé à l'argument lors de l'appel.

☞ Une modification effectuée à l'intérieur de la fonction *se répercute* alors *à l'extérieur* de la fonction.

Le passage par référence doit être explicitement indiqué en utilisant le symbole `&` après le type ; par exemple : `double& x`.

« x » le paramètre de la fonction est une référence à la variable passée en argument, donc est simplement un autre nom pour la variable « val » passée en argument. Ici la variable « val » valait donc 1, et le nom x référence le même emplacement mémoire qui est le même que celui occupé par « val ». Et dans le cas présent, l'incrémentation de « x » va correspondre aussi à une incrémentation de « val ». Lors d'un passage par valeur, la fonction va travailler sur une copie de l'argument, ce qui signifie que les modifications que l'on fait à l'intérieur de la fonction ne sont pas répercutées à l'extérieur de la fonction. On travaille sur une zone locale, et l'argument n'est pas altéré. Lorsqu'on utilise le passage par référence, on indique que la variable locale qui est passée en argument correspond à une référence sur l'objet associé à l'argument lors de l'appel, et donc la différence principale c'est que toute modification effectuée à l'intérieur de la fonction se répercute aussi, est visible aussi, une fois que la fonction a terminé son exécution. En C++, on indique qu'une fonction pour l'un de ses paramètres utilise

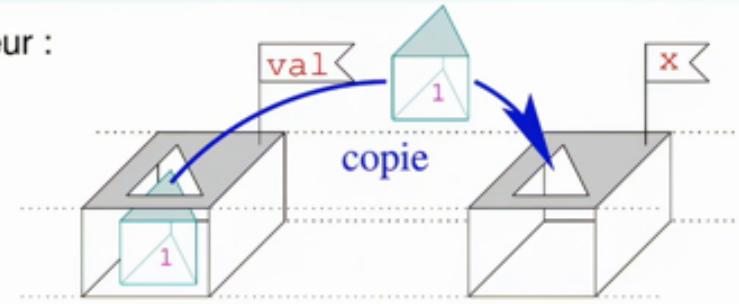
notes

résumé

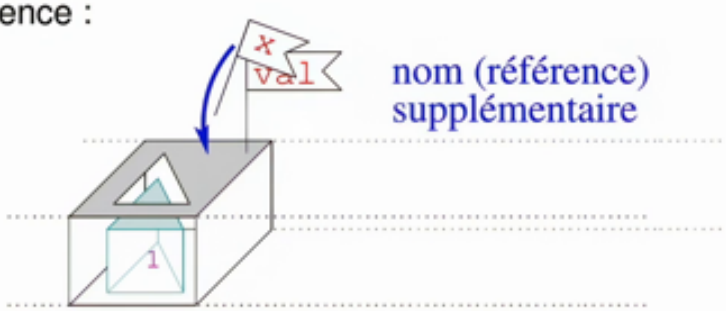
3m 1s

Passages d'argument : schéma

Passage par valeur :



Passage par référence :



le passage par référence en utilisant un symbole particulier qui est le « & ». Typiquement, au moment du prototypage de la fonction, on va indiquer que le paramètre « x » est passé par référence, ici en l'occurrence il s'agit d'un double qui est passé par référence. Pour résumer, lors d'un passage par valeur, le paramètre de la fonction est une zone locale à la fonction et toute altération n'a d'incidence que sur la zone locale, ne se répercute pas sur la variable passée en argument. Lors d'un passage par référence, on indique que le paramètre de la fonction n'est autre qu'un nom supplémentaire pour la variable passée en argument et toute altération du paramètre altérera aussi la variable passée en argument. altérera aussi la variable passée en argument.

notes

résumé

4m 13s