Support de cours

Cours:

Initiation à la programmation (en C++)

Vidéo:

Typedef - alias de types (partie 2)

Concepts (extraits des sous-titres générés automatiquement) :

Seule ligne. Type entier. Manipulation des vecteurs. Distance d. Type double. Produit vectoriel. Nom clair. Concept de vecteur. Ligne de votre typedef. Programme plein d'entiers. Possible grâce. Meilleure identification des concepts. Déclarations de tableaux. Distances d'un seul coup. Concept de matrice.

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

vers la vidéo

EPFL

Typedef : alias de types
(Partie 2)

Initiation à la programmation (C++)

Vincent Lepetit, Jean-Cédric Chappelier et Jamila Sam

...

notes

résumé

0m 0s

Alias de types

EPFL

De telles définitions de nouveaux noms de types sont particulièrement utiles, pour :

- *bien définir les types* des objets que l'on manipule
- les *paramètres de fonctions*
- les déclarations de *tableaux*

Ces alias de types, ces typedef, sont utiles pour trois choses : tout d'abord, les déclarations de tableaux comme nous venons de le voir ce qui permet non seulement d'améliorer la lecture, l'écriture ou la manipulation des vecteurs, des matrices mais aussi de clairement identifier les concepts, le concept de vecteur, le concept de matrice.

notes

résumé

0m 1s

Alias de types

EPFL

De telles définitions de nouveaux noms de types sont particulièrement utiles, pour :

- *bien définir les types* des objets que l'on manipule
  Exemple :

```
typedef int Distance;

Distance ma_longueur(0);
```

Distance d; Distance d2;
int nb; int temp;

  - meilleure identification des « concepts »
    Si tout est `int`, on ne distingue plus les `distance`s des `volume`s, des `couleur`s, etc.
  - changements ultérieurs de types plus faciles
    par exemple, les distances deviennent des `double`
- les *paramètres de fonctions*
- les déclarations de *tableaux*

Les typedef permettent de bien définir, de bien identifier les concepts et c'est peut-être là l'utilisation la plus importante des typedef pour bien identifier les objets que l'on manipule. Prenons un exemple, supposons que vous ayez un programme dans lequel vous avez besoin de manipuler les distances. Et vous avez décidé au départ que c'était de type entier, donc, vous avez une distance d vous l'avez déclaré comme un entier. Vous avez par exemple une distance d2 que vous avez déclaré comme un entier et supposons aussi que vous ayez des nombres, que vous ayez compté un nombre nb de type entier. Vous avez aussi des températures etc... Vous avez dans votre programme plein d'entiers mais qui représentent fondamentalement des concepts différents, des distances, des températures, des nombres et supposons maintenant que vous décidiez de modifier votre programme et que les distances deviennent de type double. La question c'est de savoir parmi tous ces int que vous avez partout dans votre programme, lesquels doivent être changés en double et lesquels doivent encore garder int ? Si vous aviez défini un typedef, la solution aurait été beaucoup plus simple. Si vous avez comme ça un typedef ici, typedef int Distance, ce qui signifie que le terme distance ici le mot "distance" représente un entier, vous pouvez tout à fait l'utiliser comme un nouveau type, un alias de type pour représenter des longueurs, la variable ma longueur. A ce moment-là, dans l'exemple précédent, le d et le d2 auraient été des distances et si vous avez besoin de modifier les distances

notes

résumé

0m 25s

Alias de types

EPFL

De telles définitions de nouveaux noms de types sont particulièrement utiles, pour :

- *bien définir les types* des objets que l'on manipule
  Exemple :

```
typedef int Distance;

Distance ma_longueur(0);
```

  - meilleure identification des « concepts »
    Si tout est `int`, on ne distingue plus les `distance`s des `volume`s, des `couleur`s, etc.
  - changements ultérieurs de types plus faciles
    par exemple, les distances deviennent des `double`
- les *paramètres de fonctions*
- les déclarations de *tableaux*

pour qu'elles ne soient plus des entiers, mais qu'elles soient des double, vous avez une seule ligne à modifier, c'est la ligne de votre typedef. Vous remplacez int par double et en faisant ceci, en ayant modifié une seule ligne, la ligne du typedef alors toutes vos distances d'un seul coup deviennent de type double et vous n'avez pas changé ni les nombres d'un côté, ni les températures de l'autre. L'utilisation des typedef comme ceci permet une meilleure identification des concepts et ça c'est très important pour rendre votre programme beaucoup plus clair, beaucoup plus intelligible, mais aussi et en même temps quelque chose qui est aussi important de permettre les changements ultérieurs comme la modification de int en double ici pour les distances, de rendre ces modifications beaucoup plus faciles.

notes

résumé

2m 1s

Alias de types

EPFL

De telles définitions de nouveaux noms de types sont particulièrement utiles, pour :

- *bien définir les types* des objets que l'on manipule
- les *paramètres de fonctions*

  Écriture plus claire, plus compacte et plus systématique

  Exemple :

```
typedef vector<double> Vecteur;

Vecteur produit_vectoriel(Vecteur, Vecteur);
```

- les déclarations de *tableaux*

vector<double> produit_vectoriel(vector<double>, vector<d

Un cas particulier de l'identification des concepts à l'aide de typedef c'est de faciliter l'écriture des paramètres des fonctions. On a une écriture du coup beaucoup plus claire beaucoup plus compacte des fonctions. Prenons l'exemple d'une fonction qui serait le produit vectoriel, produit vectoriel entre deux vecteurs. Si on écrit comme ceci, on voit clairement que ce sont deux vecteurs qui retournent un vecteur. Ceci étant possible grâce à la ligne ici, que l'on aura mis au début de son programme typedef vector Vecteur et il est beaucoup plus clair de lire que le produit vectoriel retourne un vecteur et prend deux vecteurs plutôt que d'avoir à écrire vectorproduit_vectoriel (vector,vector);

notes

résumé

2m 47s

Alias de types

EPFL

De telles définitions de nouveaux noms de types sont particulièrement utiles, pour :

- *bien définir les types* des objets que l'on manipule
- les *paramètres de fonctions*
- les déclarations de *tableaux*

ce qui est quand même nettement moins lisible et compréhensible.

notes

résumé

3m 37s

Pour toutes ces raisons, et fondamentalement pour la première bien identifier les concepts, je vous encourage vraiment à utiliser les typedef. Utilisez-les au maximum. A chaque fois que vous pouvez donner un nom clair, intelligible à un type, cela améliore largement votre programme. cela améliore largement votre programme.

notes

résumé

3m 43s