Support de cours

Cours:

Initiation à la programmation (en C++)

Vidéo:

Utilisation des vector (partie 1)

Concepts (extraits des sous-titres générés automatiquement) :

Tableau dynamique. Erreur fatale. Copie de tab1. Erreur d'accès. Défaut de segmentation. Tableau vide. Calculs différents. Programmeurs débutants. Taille du tableau. Erreur de segmentation. Tableau d'entiers. Syntaxe d'affectation habituelle. Syntaxe suivante. Erreur juste. Exemple typique.

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

vers la vidéo

EPFL

Tableaux : utilisation des `vector`

(Partie 1)

Initiation à la programmation (C++)

Vincent Lepetit, Jean-Cédric Chappelier et Jamila Sam

...

notes

résumé

0m 0s

Affectation globale d'un tableau

EPFL

Tout tableau (qui n'a pas été déclaré comme constant) peut être modifié par une **affectation globale** du tableau en tant que tel.

On parle ici de l'affectation d'un tableau *complet*, dans sa totalité.

La syntaxe est :

tableau1 = *tableau2* ;

où *tableau2* est un tableau de même type que *tableau1*.

Exemple :

```
vector<int> tab1({ 1, 2, 3 });
vector<int> tab2;
...
tab2 = tab1 ; // copie de tout tab1 dans tab2
```

Lorsque nous avons déclaré et initialisé un tableau dynamique, nous voudrons bien sûr l'utiliser. Pour ce faire, nous pouvons soit utiliser le tableau globalement, ou élément par élément. Utiliser un tableau globalement, ce n'est pas très fréquent, mais je peux être utile de temps en temps, nous allons simplement attribuer globalement un tableau existant au tableau que nous voulons manipuler. Cela se fera donc simplement avec la syntaxe d'affectation habituelle (=). Prenons un exemple. Supposons que nous ayons un tableau tab1, que nous avons déclaré à l'avance, et initialisé ici avec trois valeurs.

notes

résumé

0m 1s

Affectation globale d'un tableau

EPFL

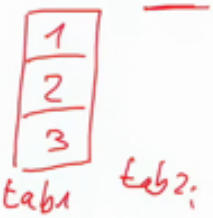
Tout tableau (qui n'a pas été déclaré comme constant) peut être modifié par une **affectation globale** du tableau en tant que tel.

On parle ici de l'affectation d'un tableau *complet*, dans sa totalité.

La syntaxe est :

tableau1 = *tableau2* ;

où *tableau2* est un tableau de même type que *tableau1*.



Exemple :

```
vector<int> tab1({ 1, 2, 3 });
vector<int> tab2;
...
tab2 = tab1 ; // copie de tout tab1 dans tab2
```

C'est un tableau d'entiers. Donc ce tableau tab1 contiendra, au début, trois entiers, qui ont pour valeurs 1, 2, 3, en raison de cette initialisation ici. Nous déclarons alors ici un tableau dynamique tab2, qui est le tableau vide. La déclaration que nous avons faite ici, comme elle n'a rien après elle, déclare un tableau initialisé sur un tableau vide. C'est un tableau sans éléments. Nous ferons ensuite des calculs différents, et enfin, à un certain moment, nous devrons peut-être faire une copie de tab1 dans tab2,

notes

résumé

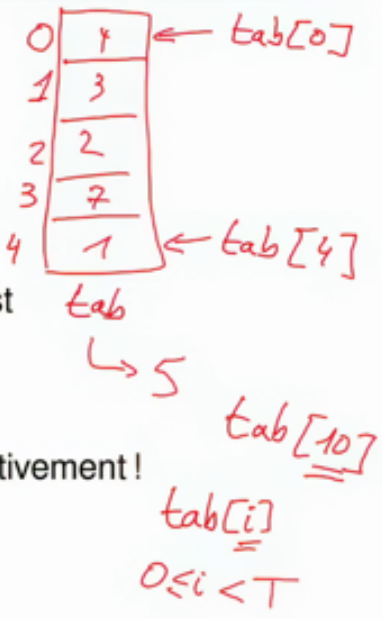
0m 37s

Accès direct aux éléments d'un tableau

EPFL

Le i+1ème élément d'un tableau `tab` est référencé par

`tab[i]`



Attention ! Les indices correspondant aux éléments d'un tableau de taille `T` varient entre `0` et `T-1`

Le 1er élément d'un tableau `tab` précédemment déclaré est donc `tab[0]` et son 10e élément est `tab[9]`

Attention ! Il n'y a pas de contrôle de débordement !!
Il est impératif que l'élément que vous référencez existe effectivement !
Sinon risque de *Segmentation Fault* !

Exemple (à ne pas suivre !) d'erreur classique :

```
vector<double> v;
v[0] = 5.4;  // NON !! v[0] n'existe pas encore !
```

nous allons donc faire une copie de tab1 dans tab2, ce qui signifie que maintenant, tab2 contiendra les éléments 1, 2, 3. Encore une fois, c'est une copie, donc à partir de cette ligne, tab1 et tab2 continueront de vivre séparément, si je modifie l'onglet 1, il n'y aura aucune modification sur l'onglet 2. Si une modification est apportée ici, il n'y a aucune modification dans l'onglet 2. tab1 et tab2 sont deux tableaux complètement séparés. C'est pour l'affectation globale d'un tableau. Mais en général, nous préférerons manipuler les tableaux élément par élément. Pour désigner l'élément d'un tableau, nous utiliserons la syntaxe suivante : le nom du tableau, suivi de, entre parenthèses ([ ]), l'index de l'élément que nous voulons sélectionner. Attention, les index des éléments d'un tableau, sont compris entre zéro et la taille du tableau moins un, ils commencent à zéro et allez à la taille du tableau moins un. Si par exemple, j'ai un tableau avec cinq éléments, qui contient 4, 3, 2, 7, 1, l'élément 4 ici, est l'élément avec la position zéro dans le tableau. Si ce tableau est nommé tab, j'utiliserai tab[0], pour désigner ce 4. Ensuite, j'ai tab[1], ici j'ai tab[2], tab[3], tab[4]. Ce tableau a une taille de cinq. Le dernier élément du tableau est tab[5 moins un], à savoir tab[4]. Les index vont de zéro à la taille du tableau moins un. Le premier élément du tableau est donc bien désigné par tab[0], et le dixième élément du tableau est désigné par tab[9], comme si nous commencions à zéro. Soyez également prudent avec le fait qu'il n'y a aucun contrôle du tout si vous êtes dans une situation de dépassement de tableau. Si par exemple, vous écrivez avec l'exemple précédent tab[10], rien ne vous dira que vous avez ici dépassé la taille du tableau. Alors soyez prudent lorsque vous écrivez quelque chose comme tab[i], pour être sûr que i est

notes

résumé

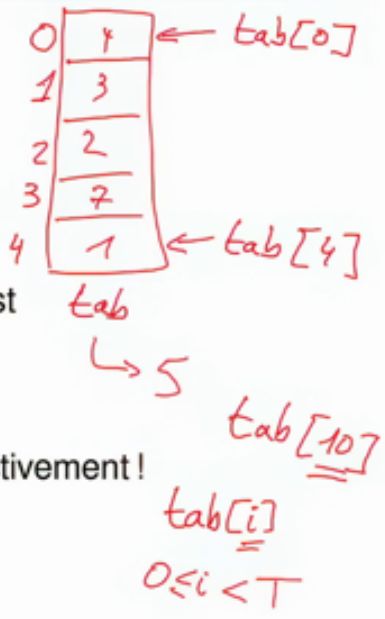
1m 13s

Accès direct aux éléments d'un tableau

EPFL

Le i+1ème élément d'un tableau `tab` est référencé par

`tab[i]`



Attention ! Les indices correspondant aux éléments d'un tableau de taille `T` varient entre `0` et `T-1`

Le 1er élément d'un tableau `tab` précédemment déclaré est donc `tab[0]` et son 10e élément est `tab[9]`

Attention ! Il n'y a pas de contrôle de débordement !!
Il est impératif que l'élément que vous référencez existe effectivement !
Sinon risque de *Segmentation Fault* !

Exemple (à ne pas suivre !) d'erreur classique :

```
vector<double> v;
v[0] = 5.4;  // NON !! v[0] n'existe pas encore !
```

compris entre zéro et la taille du tableau moins un, ou est strictement inférieur à la taille du tableau, sinon vous risquez de rencontrer cette erreur fatale, que nous appelons défaut de segmentation, ce qui indique une erreur d'accès à la mémoire. Je vous le dis, vous pouvez le rencontrer, car ce n'est même pas garanti que l'erreur de segmentation se produit, l'erreur est là, mais ce n'est même pas sûr que le programme le détectera. Quoi qu'il en soit, si la détection se produit, à ce moment-là, vous aurez une erreur de segmentation, avec ici l'exemple typique, que l'on rencontre beaucoup dans les programmes pour programmeurs débutants, qui est la suivante : nous déclarons un tableau dynamique, de doubles, que je nomme ici v, qui est initialisé à vide,

notes

résumé

comme il n'y a rien après la déclaration, nous initialisons donc un tableau vide, donc ce tableau est strictement vide, il n'y a rien dedans. C'est un tableau avec zéro élément. En général, le programmeur qui a écrit un programme, fera rarement l'erreur juste après, mais l'erreur est toujours de ce type, il écrira quelque chose comme v[0] = avec une affectation, pour essayer de mettre 5.4 dans le tableau, mais comme ce tableau est vide, nous n'avons même pas av[0], v[0] n'existe pas dans le tableau vide, donc ici vous aurez une erreur de segmentation. donc ici vous aurez une erreur de segmentation.

notes

résumé

4m 13s