QExtend - Tâche #1298

Évolution # 700 (Assigné): Classes conteneurs 2D

Array

03/18/2011 03:42 PM - gbdivers

Status:	Assigné	**Start date:**	03/18/2011
Priority:	Normal	**Due date:**	03/20/2011
Assignee:	gbdivers	**% Done:**	80%
Category:	Core	**Estimated time:**	0.00 hour
Target version:	Version 0.0.2	**Spent time:**	2.00 hours

Description

Création des classes "Array" : StaticArray1D, DynamicArray1D, StaticArray2D, DynamicArray2D, StaticArray3D, DynamicArray3D

History

#1 - 03/18/2011 03:54 PM - yan

Quel est la différence entre
StaticArray1D, StaticArray2D, StaticArray3D ou DynamicArray1D,DynamicArray2D, , DynamicArray3D ?
La dimension ne pourrait'il pas être un paramètre template?

#2 - 03/18/2011 04:04 PM - gbdivers

C'est bien la dimension qui change (1, 2 ou 3D). Static et dynamic correspondent si c'est un tableau de taille fixe ou non.
Il y a donc des correspondance (volontairement, pour des questions d'homogénéité) entre certaines classes et ce que propose Qt et boost. En particulier StaticArray1D correspond à boost::array et DynamicArray1D à QVector (avec quelques différences, évidemment).

Pour les tableaux à N dimensions, il y a déjà boost::multiarray. Mais les tableaux 2D et 3D étant très utilisés (pour les jeux en particulier) et boost::multiarray ayant une syntaxe un peu lourde, ces classes permettent surtout de simplifier la syntaxe et de proposer un Qt-style

#3 - 03/18/2011 04:08 PM - yan

ma question été surtout pour mettre la dimention d'un parametre template ;)
On vpourra en reparler avec la v1

#4 - 03/22/2011 09:50 PM - gbdivers

Plein de questions sur ce que vous préférez :

- utilisation du type size_t pour les index (comme dans la stl) ou int (comme Qt) ? Càd "inline T& at(size_type i);" ou "inline T& at(int i);" ?

- utilisation de Allocator() (comme dans la stl) ou pas (comme dans Qt) ?

- mettre le destructeur en virtuel (Qt style) ou non (stl style) ?

- pour les tableaux de taille fixe, on exclue les tableaux de taille 0 ou non ?

- en analysant le code de la stl et Qt, j'ai trouvé différentes implémentations pour une même fonction. Par exemple pour fill() :

```
for (size_type i = 0; i < N; ++i)
    elems[i] = value;
```

ou

```
for (iterator it = begin(); it !=  end(); ++it)
```

```
*it = value;
```

ou

```
T* first = elems;
T* last = elems + N;
while (first != last)
    *--it = value;
```

- pour les fonctions de recherche d'items (indexOf, lastIndexOf), vous préférez retourner une paire d'indice ("std::pair<int,int>") ou passer les indices en référence ("indexof(int& i, int& j)")

Vous avez des préférences ?

#5 - 03/22/2011 09:55 PM - dourouc05

Je dirais de t'orienter plus sur du Qt-style. Pour le dernier point, je m'orienterais plus vers les deux dernières options.

#6 - 03/22/2011 09:59 PM - gbdivers

yan a écrit :

> ma question été surtout pour mettre la dimention d'un parametre template ;)
> On vpourra en reparler avec la v1

Au fait, pour répondre à ton message...
tu parles d'avoir un template du type :
template<typename T, size DIM> array;
array<int, 2> -> tableau d'entier à 2D
array<double, 3> -> tableau de double en 3D
?

C'est ce que fait boost pour multiarray mais je trouve l'écriture très lourde du coup (utilisation de boost::extends)
On utilise souvent des tableaux à 2 et 3D pour qu'il soit intéressant de les implémenter tel quel

#7 - 03/22/2011 10:01 PM - yan

gbdivers a écrit :

> mais je trouve l'écriture très lourde du coup (utilisation de boost::extends)
> On utilise souvent des tableaux à 2 et 3D pour qu'il soit intéressant de les implémenter tel quel

ok, comme tu préfère

#8 - 03/22/2011 10:06 PM - yan

gbdivers a écrit :

> - utilisation du type size_t pour les index (comme dans la stl) ou int (comme Qt) ? Càd "inline T& at(size_type i);" ou "inline T& at(int i);" ?

size_t

> - utilisation de Allocator() (comme dans la stl) ou pas (comme dans Qt) ?

pourquoi pas.

> - mettre le destructeur en virtuel (Qt style) ou non (stl style) ?

je dirais non. Es ce que cela à un sens d'en hériter?

> - pour les tableaux de taille fixe, on exclue les tableaux de taille 0 ou non ?

oui? ça sert à quoi une taille de 0?

Sinon, je pense que ces classes devraient être compatible stl.

#9 - 03/22/2011 10:15 PM - gbdivers

> - mettre le destructeur en virtuel (Qt style) ou non (stl style) ?

je dirais non. Es ce que cela à un sens d'en hériter?

C'est ce que fait Qt, par exemple pour pouvoir créer QStack en héritant de QVector ou QStringList en héritant de QList

> - pour les tableaux de taille fixe, on exclue les tableaux de taille 0 ou non ?

oui? ça sert à quoi une taille de 0?

A rien :) il suffit juste d'ajouter un assert(N != 0) dans les constructeurs

> Sinon, je pense que ces classes devraient être compatible stl.

Tu penses aux itérateurs ? normalement, c'est la cas (sauf oublie). Autre chose ?

Pour les autres questions, je vois que les avis diverges... on verra à l'utilisation alors

#10 - 03/22/2011 11:08 PM - yan

gbdivers a écrit :

> Tu penses aux itérateurs ? normalement, c'est la cas (sauf oublie). Autre chose ?

Spécialiser std::swap pour ces classes.
Après je ne voie pas trop. Peut être des itérateurs type java comme ce que fournie Qt?
Pour les classes dynamique, tu va utiliser quoi pour la mémoire un std::vector? un pointeur?

Utilisation du COW ?

#11 - 03/22/2011 11:15 PM - gbdivers

yan a écrit :

> gbdivers a écrit :
>
> > Tu penses aux itérateurs ? normalement, c'est la cas (sauf oublie). Autre chose ?
>
> Spécialiser std::swap pour ces classes.
> Après je ne voie pas trop. Peut être des itérateurs type java comme ce que fournie Qt?
> Pour les classes dynamique, tu va utiliser quoi pour la mémoire un std::vector? un pointeur?
>
> Utilisation du COW ?

j'utilise pas de conteneur en interne (pour des raisons de performances principalement et pour avoir un meilleur controle). Pas de COW prévu (pour des raisons de performances aussi... mais c'est à voir)

#12 - 03/22/2011 11:24 PM - yan

gbdivers a écrit :

> j'utilise pas de conteneur en interne (pour des raisons de performances principalement et pour avoir un meilleur controle).

En quoi un std::vector serait plus lent qu'un tableau dynamique?

#13 - 03/23/2011 10:47 AM - gbdivers

yan a écrit :

> gbdivers a écrit :
>
> > j'utilise pas de conteneur en interne (pour des raisons de performances principalement et pour avoir un meilleur controle).
>
> En quoi un std::vector serait plus lent qu'un tableau dynamique?

A cause des tests internes (et détails d'implémentation) qui sont réalisé plusieurs fois (par exemple, lors d'un accès aux données avec value(i,j), je teste que les bornes sont correctes par rapport à N et M, puis l'index calculé i+j*N est vérifié de nouveau par rapport à N*M) Cela dépend évidement des implémentations de la stl et des options de compilation
Mais quitte à devoir écrire une partie, autant tout réécrire

Autre question :
lors de la création d'un static array, pensez vous que c'est gênant ces syntaxes :

```
StaticArray1D<int, 10> array(5); -> taille 10 remplit avec des 5
DynamicArray1D<int> array(5); -> taille 5 remplit avec des 0
Pour avoir une taille de 10 remplit avec des 0 -> DynamicArray1D<int> array(10, 5);
```

Risque de confusion du paramètre donné dans le constructeur ? (taille dans un cas, élément par défaut dans l'autre)

#14 - 03/23/2011 10:58 AM - gbdivers

Autre question :

boost utilise des vues pour extraire une partie des données d'un tableau (par exemple sur un tableau 2D de taille (10,10), on va extraire une vue, càd un autre tableau 2D, de taille (3,2) coorrespondant aux colonnes 2 à 5 et aux lignes 3 à 5)

Perso j'avais utiliser des itérateurs spécifiques :

```
VerticalIterator // parcours la tableau par colonne
HorizontalIterator // parcours le tableau par ligne
Iterator it = array.iterator(i, j); // récupère un itérateur à la position (i,j)
std::foreach(array.iterator(i, j), array.iterator(i2, j2), fct); // utilisation d'un algo std
```

J'ai implémenté les itérateurs mais si les vues vous semble aussi intéressantes, je verrai si je les implémente également (il faudrait que je regarde comment boost fait, pour éviter les copies du tableau)

#15 - 03/23/2011 11:17 AM - gbdivers

Pour l'utilisation de std::size_t ou de int, je vais faire :
- paramètre de template : size_t
- paramètre de fonction : int (comme Qt)

#16 - 03/23/2011 01:41 PM - gbdivers

yan a écrit :

> gbdivers a écrit :
>
> > j'utilise pas de conteneur en interne (pour des raisons de performances principalement et pour avoir un meilleur controle).
>
> En quoi un std::vector serait plus lent qu'un tableau dynamique?

Bon, correction. Finalement je vais utiliser un std::vector (paramètre template, ce qui permet d'utiliser un QVector si on veut, pour avoir le COW)
Ce que j'avais écrit fonctionner ("tomber en marche") mais je catchais pas les exceptions sur les new, je ne réallouais pas une mémoire de taille plus importante à chaque resize, etc.

#17 - 03/23/2011 01:58 PM - yan

gbdivers a écrit :

> Bon, correction. Finalement je vais utiliser un std::vector (paramètre template, ce qui permet d'utiliser un QVector si on veut, pour avoir le COW)
> Ce que j'avais écrit fonctionner ("tomber en marche") mais je catchais pas les exceptions sur les new, je ne réallouais pas une mémoire de taille plus importante à chaque resize, etc.

Le gros avantage que je vois sont les iterateur déjà définie et optimisé pour les algo std.
Par contre tu ne pourra pas donner une mémoire qui serait interfacé par cette classe.

Pour les staticArray, ca me semble bien et la taille mémoire est correcte (sizeof(StaticArray<char,3>) == 3 ;)). On peut donc caster sans problème.
Faudrait ajouter des define quand on veux boost ou non.

#18 - 03/23/2011 02:16 PM - gbdivers

> Le gros avantage que je vois sont les iterateur déjà définie et optimisé pour les algo std.

oui, j'avais du réécrire les itérateurs, ce qui alourdissait la classe

> Par contre tu ne pourra pas donner une mémoire qui serait interfacé par cette classe.

c'est à dire ? un code exemple ?

> Pour les staticArray, ca me semble bien et la taille mémoire est correcte (sizeof(StaticArray<char,3>) == 3 ;)). On peut donc caster sans problème. Faudrait ajouter des define quand on veux boost ou non.

oui pour les defines. Pour le moment, j'ai ajouté plusieurs include, qui ne servent pas forcement tous. a faire le tri

#19 - 03/23/2011 02:27 PM - yan

gbdivers a écrit :

> c'est à dire ? un code exemple ?

Je veux juste que tu ne pourra interfacer une mémoire avec ce type de classe.
Par exemple, si on considère DynamicArray2D comme une image, il peux être iteressant de lui fournir la mémoire à utiliser et éviter des copy.

Mais c'est peut être u autre type de classe.

#20 - 03/23/2011 03:16 PM - gbdivers

Utiliser un vector pour accéder à une mémoire peut être problématique. Par exemple, que faire si on fait un resize sur le vector ? il faut pouvoir appeler le resize de l'image aussi... idem dans l'autre sens, il faut que le vector soit mis à jour en cas de resize de l'image... donc il faut que le conteneur contienne un accès à l'image et ce n'est plus un conteneur générique.

Peut être à la rigueur une classe dérivée. Mais pour le moment, ce n'est pas au programme (et quel serait l'avantage par rapport à QImage ?)

#21 - 03/23/2011 07:58 PM - gbdivers

Note : pour le support du type de conteneur
- http://en.wikipedia.org/wiki/Substitution_failure_is_not_an_error
- http://www.boost.org/doc/libs/1_46_1/libs/utility/enable_if.html

#22 - 03/23/2011 10:50 PM - yan

gbdivers a écrit :

> Note : pour le support du type de conteneur
> - http://en.wikipedia.org/wiki/Substitution_failure_is_not_an_error
> - http://www.boost.org/doc/libs/1_46_1/libs/utility/enable_if.html

Penser à completer src/core/templatetools.hpp qui contient déjà une structure pour SFINAE.
Le reste est du test.

#23 - 03/24/2011 12:33 AM - gbdivers

Je n'ai pas précisé : pour le moment, je pense laisser tel quel, càd sans ajouter SFINAE donc en ne prenant en charge que std::vector, QVector et QList (qui, étrangement, possède l'opérateur [])

L'ajout de cette possibilité est reporté aux prochaines versions (ou plus tard...), en fonction des besoins.

<mode digression>Je dois comprendre que tu connaissais, non ? Tu l'as trouvé dans le Vandevoorde ou tu connaissais avant ? Tu l'utilises où le templatetool.hpp ? J'ai vraiment encore beaucoup de choses à apprendre :((surtout en méta)</mode digression>

#24 - 03/24/2011 09:16 AM - yan

gbdivers a écrit :

> Je n'ai pas précisé : pour le moment, je pense laisser tel quel, càd sans ajouter SFINAE donc en ne prenant en charge que std::vector, QVector et QList (qui, étrangement, possède l'opérateur [])

QList est un vecteur et non une list chainé ;). Dans la pluspart des cas elle est plus adapté que QVector.

> <mode digression>Je dois comprendre que tu connaissais, non ? Tu l'as trouvé dans le Vandevoorde ou tu connaissais avant ? Tu l'utilises où le templatetool.hpp ? J'ai vraiment encore beaucoup de choses à apprendre :((surtout en méta)</mode digression>

oui mais je n'ai pas encore tous compris :P
Je m'en sert pour savoir si la classe hérite de QObject ou non pour utilise un QWeakPointeur sur les QObject. C'ets dans les smartpointeur.
Petite discution
http://www.developpez.net/forums/d906935/c-cpp/cpp/langage/specialisation-template-fonction-lheritage/

#25 - 03/24/2011 11:34 AM - gbdivers

Je suis complètement à la ramasse niveau lecture...
Le SFINAE est abordé dans le Vandevoorde à la page 106. Je n'ai pas encore eu le temps d'aller jusque là :(

#26 - 03/24/2011 01:08 PM - gbdivers

Je viens de lire le post sur le forum que tu avais posé concernant SFINAE... en fait, on se pose des questions similaires... mais tu me devances d'un an :)

#27 - 03/24/2011 02:35 PM - yan

gbdivers a écrit :

> Je viens de lire le post sur le forum que tu avais posé concernant SFINAE... en fait, on se pose des questions similaires... mais tu me devances d'un an :)

^^ mais bon tu va facilement me rattraper