Benjamin Hamon

Sophie Huart

Alexandre Fischer

Sébastien Corbeau

Alain Le

Jeremy Moriaux

Objet avancé

Programmation générique

# Contexte du projet

L’objectif du projet est de créer une bibliothèque logicielle C++ en utilisant les principes de programmation générique et des techniques d’implantations associées vues en cours (traits, méta-programmation, tag dispatching, EDL, etc…).

Notre sujet porte sur les conteneurs multidimensionnels. L’objectif est de trouver des concepts pour gérer et implanter des conteneurs de dimension N quelconque. Pour cela, il faut fournir une implantation d’un tableau à N dimensions et d’en inférer les extensions nécessaires pour gérer de manière homogène d’autres conteneurs en mode N dimensions. Ainsi la librairie finale mettra à disposition de l’utilisateur un certains nombre de fonctions génériques de traitement de tableaux multidimensionnels et un mécanisme d’extension pour intégrer facilement des conteneurs personnalisés.

# Organisation

Nous avons créé un dépôt GITHUB : https://github.com/bhamon/genericprogramming

Le dépôt est divisé en plusieurs parties :

* Les répertoires « include » et « src » : ils contiennent les sources de notre librairie. Nous avons défini le namespace « pps::range » pour tous les objets de la librairie.
* Le répertoire « tests » : il contient les différents tests unitaires de la librairie. Chaque fichier cpp se compile séparément. Le fichier « tests.h » contient un ensemble de fonctions utilitaires pour réaliser ces tests.
* Les fichiers Makefile : ils permettent de compiler les tests ainsi que la librairie.

# Démarche

## Premiers pas

Un tableau multidimensionnel au sens large du terme peut prendre diverses formes en C++, comme le montre le code ci-dessous :

std::vector<int> t1;

std::list<std::vector<double\*\*>> t2;

char t3\*[20];

On constate d’entrée de jeux que l’on peut séparer les conteneurs en trois catégories :

* Les classes de conteneur génériques.
* Les tableaux natifs du langage.
* Les pointeurs sur des séries d’éléments contigus en mémoire.

Dans tous les cas, on peut dégager un modèle en s’inspirant de la « standard library », et plus particulièrement des iterators. Ce modèle sera spécialisé pour prendre en compte les différentes implémentations concrètes.

Ainsi, nous avons fait le choix de considérer une dimension d’un tableau comme un « range », c’est-à-dire une paire d’itérators. Les tableaux multidimensionnels sont donc considérés comme des range en cascade que l’on peut apparenter à un arbre, comme le montre la figure ci-dessous :

La spécification des types de données et des types d’iterators s’effectue par le biais d’un traits dont l’implémentation est localisée dans le fichier « traits.h ». Ce traits a été spécialisé pour les tableaux natifs, les pointeurs sur des éléments contigus en mémoire et les conteneurs de la STL.

La récupération de la paire d’iterators d’une dimension s’effectue par le biais des fonctions « begin » et « end », localisées dans les fichiers « begin.h » et « end.h ». De la même manière que pour le traits, la fonction « begin » a été spécialisée pour les tableaux natifs, les pointeurs et les conteneurs de la STL. Néanmoins, la fonction « end » n’a été spécialisée que pour les conteneurs de la STL, la récupération du nombre d’éléments présents dans un tableau natif et dans un espace mémoire pointé ne pouvant être effectué.

Grâce à ce panel, il est possible de gérer n’importe quel tableau multidimensionnel formé de tableaux natifs, de pointeurs et de conteneurs de la STL, comme le montre le fichier de test « test\_usage.cpp ».

## Extensibilité

La librairie permet de gérer tout type de conteneurs en les apparentant à des range. Pour intégrer un nouveau conteneur au sein de la librairie, il faut procéder en trois étapes simples :

* Spécialisation du traits pour le type utilisateur à gérer.
* Définition des fonctions begin (const et no-const) pour le type en question.
* Définition des fonctions end pour le type en question.

Un exemple d’extension est disponible dans le fichier de test « test\_extend.cpp ».

Si une extension venait à manquer pour un conteneur utilisateur, le code mis en place prévient l’utilisateur par le biais d’une static\_assert propre liée à un commentaire d’explications lui expliquant son erreur.

## Aide à la manipulation

En l’état, le parcours descendant dans les différentes dimensions du tableau requiert une écriture un peu lourde. De plus, il est impossible de connaitre le type de base manipulé par le tableau multidimensionnel. Pour résoudre ces problèmes, nous avons introduit un second traits disponible dans le fichier « traitsBaseType.h ».

Ce nouveau traits apporte donc la possibilité d’inspecter les types consécutifs d’un tableau multidimensionnel en ajoutant la notion de dimension. Un exemple est disponible dans le fichier de test « test\_manip.cpp ».