

TP n° 4 de C++

Classe Pile - Vecteur et généricité élémentaire

Vecteur dynamique

« Une classe générique se construit plus facilement à partir d'une classe simple conçue sur un type de base. » B.S.

Stack
- <code>_size</code> : entier - <code>_sp</code> : pointeur courant de la pile - <code>_base</code> : <i>base de la pile</i>
+ <code>Stack ()</code> + <code>Stack (const Stack & inStack)</code> + <code>~ Stack ()</code> + opérateur d'affectation + <code>push(const char inChar)</code> + <code>pop() : char</code> + <code>top() const : char</code> + <code>getNbElements() const : int</code>

- 1) Ecrire le code C++ d'une classe Stack de caractères avec comme attributs une taille `_size`, un pointeur de pile `_sp`, et un pointeur sur la base de la pile : `_base`. On prévoit les méthodes suivantes, implémentées de façon déportée, en plus de celles exigées par la forme canonique de Coplien :
 - la méthode `push()` pour empiler un élément ;
 - la méthode `pop()` qui dépile l'élément situé au sommet ;
 - la méthode `top()` qui renvoie une référence sur l'élément (deux versions à prévoir) ;
 - une méthode `getNbElements()` qui retourne le nombre courant d'éléments présents dans la pile.

Ecrire un programme principal qui teste cette classe.

- 2) Ecrire une classe pile générique et implémenter les méthodes de façon déportées. Ecrire un programme principal dans un fichier séparé qui teste plusieurs instances. Suivez bien le guide de style et les conseils pour savoir quel type de fichier utiliser et inclure.

- 3) Pour ceux qui n'ont pas fait le supplément du dernier TP écrire une classe vecteur de réels double précision sur le modèle de la classe String du TP précédent. Tester et étudier l'option `-MM` du compilateur `g++` (avec la commande `g++ -MM *.cpp`) puis proposer un `makefile`. Tester dans le programme principal les différentes méthodes du `DoubleVector` une à une. Prenez le temps de les développer et de les tester au fur et à mesure.

DoubleVector
- <code>_size</code> : entier - <code>_pData</code> : tableau de <i>double</i>
+ <code>DoubleVector()</code> + <code>DoubleVector(int inSize)</code> + <code>DoubleVector(double * inDArray, int inSize)</code> + <code>DoubleVector(const DoubleVector & inDVector)</code> + <code>~DoubleVector()</code> + opérateur d'affectation + opérateurs d'indexation <code>[]</code> + opérateur d'addition

- 4) Ecrire le code C++ d'une classe `Vector` template en réutilisant le code précédent. C'est dans le fichier `Main.cpp` que l'on réalisera l'instanciation et l'utilisation de la classe instanciée. Vous testerez les méthodes sur des vecteurs d'entiers et de réels.
- 5) Modifier ce vecteur pour qu'il ait le comportement "dynamique" suivant. Lorsque le vecteur est plein et qu'un utilisateur désire encore ajouter un élément, il y a une nouvelle allocation dynamique de mémoire (Type amortis : nouvelle taille = ancienne-taille + $\frac{1}{2}$ * ancienne-taille). Tester ce nouveau vecteur générique.

TIPS pour les classes génériques

Conseil : Les définitions de classes génériques (partie déclaration des classes et implémentation des méthodes) devraient se retrouver dans deux fichiers :

- un « `.hpp` » classique pour la déclaration de la classe template et
- un « `.ipp` » pour la déclaration d'implémentation template (méthodes templates déportées) qui sera inclut à la fin du fichier « `.hpp` ».

Pour les pros : de l'art d'étudier les symboles...

Les templates permettent de proposer des modèles de classes. Nous allons étudier le mécanisme de génération des classes à partir des templates à partir de cet exemple.

- 1) Créez un fichier `main.cpp` dans lequel vous inclurez le fichier définissant l'une des meta-classes précédemment définies (`TemplateStack` ou `TemplateVector`). Le fichier ne contiendra rien d'autre que cette inclusion et la définition de la fonction `int main(int, char**)`, vide de tout contenu. Compilez le fichier `main.cpp`.
- 2) L'outil Unix `nm` permet de lister les symboles présents dans un fichier. Le C++ générant des symboles difficilement intelligibles pour un humain, il convient de coupler l'utilisation de cet outil à celle de `c++filt`. Vérifiez les symboles présents dans votre programme comme suit :
nm exécutable | c++filt
- 3) Dans la fonction `main`, instanciez un template issu d'une des meta-classes que vous avez précédemment développée et créez un objet de cette nouvelle classe. Compilez votre code et recommencez la manœuvre avec `nm`. Quelles différences constatez-vous ? (utilisez `diff` entre les deux sorties aux besoins)
- 4) Appelez à présent des méthodes sur l'objet créé et faites de même avec un objet issu d'une instantiation différente du template. Après avoir observé la sortie de `nm`, concluez sur le code généré par les templates.