Travaux dirigés C++ n°4

Informatique

—IMAC 2e année—

Les templates

Ce TD traite des templates en C++, qui permettent d'écrire du code générique (sans spécifier le type) que le compilateur utilisera pour générer le code demandé en fonction des types nécessaires.

▸ Exercice 1. *Fonction template*

1. *Soit la fonction template* `getMinimum` *:*

```
template<typename T>
T getMinimum(const T&a, const T&b)
{
  return  a < b ? a : b;
  // equivalent to :
  // if (a < b) return a;
  // else return b;
}
```

   Faites un nouveau programme et copiez cette fonction. Testez-la en l'appelant avec :

   - *deux entiers (e.g 42 et 57)*
   - *deux flottants (e.g 4.2 et 5.7)*
   - *deux chars (e.g 'v' et 'n')*

   Normalement, tout se passe bien, le compilateur a généré le code pour chaque type demandé.

2. *Que se passe-t-il si on teste cette fonction avec un entier et un flottant (e.g 42 et 5.7)? Proposez une solution pour résoudre le problème que vous aurez détecté.*

3. *Appelez la fonction* `getMinimum` *avec les paramètres 'a' et 'Z'. Le résultat espéré est 'a' et pourtant, la fonction nous renvoie 'Z'. Comment expliquez-vous ce résultat? Codez une solution pour résoudre ce problème (vous pourrez jeter un œil aux fonctions* `toupper` *ou* `tolower` *).*

▸ **Exercice 2.** ***Classe template***

1. *Reprenez votre classe* `VectorD` *des TP précédents. Transformez la pour qu'elle devienne une classe template. Vous devrez alors modifier toutes ses méthodes. Rappelez-vous qu'avec les templates, les méthodes doivent être codées dans le fichier* `VectorD.hpp`*.*

2. *Testez votre classe avec plusieurs types (* `int`*,* `float`*, ...). Notez que le nom de la classe n'est plus adapté à sa fonction.*

3. *Codez un constructeur permettant d'initialiser un* `VectorD` *à partir d'un* `VectorD` *d'un autre type, e.g :*

```
VectorD<float> A(3, 4.2); // dimension 3, all values initialized to 4.2
VectorD<int> B(A);        // should contain [4, 4, 4]
```

▸ **Exercice 3.** ***Template de type et de valeurs***

1. *Créez une classe template* `Tableau` *permettant de gérer un tableau dont la taille maximale* `N` *et le type* `T` *sont connus à la compilation. Les variables* `N` *et* `T` *sont les paramètres du template. Cette classe possède aussi deux attributs :* `m_data` *pour stoquer les données sous forme d'un tableau statique de taille* `N` *et* `m_size` *qui correspond au nombre d'entrées stoquées à un instant (*$\texttt{m\_size} \leq \texttt{N}$*). En effet, comme un* `std::vector`*, on pourra faire des* `push` *et des* `pop`*, mais le nombre de varaibles stockées ne pourra jamais dépasser la taille maximum* `N`*.*
   À noter que l'intérêt d'avoir la taille maximum `N` *en template permet d'éviter de faire de l'allocation dynamique pour* `m_data`*.*

2. *Ajoutez les méthodes suivantes :*

   - `isEmpty` *: retourne* `true` *si le tableau est vide,* `false` *sinon.*
   - `getSize` *: retourne le nombre d'éléments courant.*
   - `getMaxSize` *retourne le nombre d'éléments maximum.*
   - `getFirst` *: retourne une référence sur le premier élément.*
   - `getLast` *: retourne une référence sur le dernier élément.*
   - *l'opérateur* `[]` *pour pouvoir lire et modifier un élément du tableau.*
   - `push` *: insère un élément en fin de tableau.*
   - `pop` *: supprime le dernier élément du tableau.*
   - *l'opérateur* `<<` *pour afficher le tableau.*