TP n°10b : Un vecteur générique

Partie 1 : Une classe de vecteur générique

Dans cette partie, nous allons reprendre (dans un nouveau projet) la classe Vecteur du premier TP et la rendre générique. Attention, dans cette partie, nous ne reprenons que la classe mais pas les diverses fonctions qui ont pu être définies.

Question 1 : Quels sont le ou les paramètres génériques de la classe ?

Question 2 : Transformez la classe afin de la rendre générique.

Question 3 : Quelles sont les contraintes associées au paramètre générique ?

Question 4 : Créez un programme client demandant la saisie de deux vecteurs de « float » et affichant le résultat de l'addition.

Question 5 : Dans votre programme principal, ajoutez une classe de vecteurs de « std ::string » et effectuez le même traitement. Quelle est la signification de l'addition dans ce cadre ?

Partie 2 : Des fonctions génériques

Question 6 : Au premier TP sur les vecteurs, nous avons défini des opérateurs pour afficher un vecteur (opérateur <<) et lire un vecteur (opérateur >>). Reprenez ces opérateurs afin de les rendre compatibles avec votre classe de vecteur générique.

Question 7 : Quelles sont les nouvelles contraintes associées au paramètre générique ?

Question 8 : Reprenez votre programme principal de manière à ce que ce dernier utilise les opérateurs << et >> pour les affichages et les saisies.

Question 9 : Au premier TP, nous avons défini, sous la forme d'une fonction, l'opérateur * en charge de calculer le produit scalaire entre deux vecteurs. Reprenez cet opérateur et rendez le générique. Recompilez votre programme principal.

Question 10 : Modifiez les traitements effectués dans votre programme principal afin d'afficher la valeur du produit scalaire des vecteurs saisis (vecteur de « float » et de « std ::string »). Normalement, vous devriez voir une erreur apparaître, pourquoi ? Supprimez le traitement provoquant cette erreur. Que pouvez vous déduire sur ce que fait le compilateur lorsque vous utilisez la généricité ?

Partie 3 : Les types

Question 11 : les types « float » et « int » sont compatibles entre eux (autrement dit, vous pouvez les additionner, les multiplier…). Pouvez vous calculer le produit scalaire d'un vecteur de « float » avec un vecteur de « int » ? Si vous n'êtes pas certain du résultat, ajoutez cette opération dans votre programme principal et regardez.

Question 12 : En modifiant votre opérateur de calcul de produit scalaire, proposez une solution permettant de résoudre ce problème.

Question 13 : Testez votre solution en modifiant votre programme principal afin de saisir un vecteur de « int » (*a*) et un vecteur de « float » (*b*). Puis affichez le résultat de l'opération *a*b* puis de l'opération *b*a*. Saisissez des nombres à virgule pour initialiser le vecteur *b*. Que constatez vous au niveau du résultat ? Pourquoi ?

Dernière partie.

Le problème de typage pour lequel vous venez de proposer une solution peut se poser dans de nombreux autres cas. A titre d'exemple, la modification que vous venez de faire pour l'opérateur de calcul de produit scalaire, devrait être faite pour l'opérateur d'addition, d'affectation… Cependant, C++ dispose d'un mécanisme particulier pouvant vous permettre de résoudre ce problème. Voici une explication sur un exemple de ce mécanisme.

Soit la classe A, disposant d'un constructeur prenant en paramètre une instance d'objet de type B.

```
class A
{
public :
        A(const B & b) ;
} ;
```

Soit une fonction prenant en paramètre un objet de type A et **ne le modifiant pas**.

```
void func(const A & a) ;
```

Soit le programme suivant :

```
{
        B b ;           // Déclaration d'une variable de type B.
        func(b) ;       // Appel de la fonction avec un objet de type B.
}
```

L'appel de la fonction « func » avec un objet de type B en paramètre devrait générer une erreur puisque cette dernière n'est définie que pour le type A. Et bien non, tout cela compile…En réalité, le compilateur, dans ce genre de cas, essaye de trouver un constructeur lui permettant de créer à partir de l'objet de type B, un objet de type A. Si ce dernier existe (ce qui est le cas dans notre exemple), le compilateur transforme **automatiquement** le code de la manière suivante :

```
{
        B b ;           // Déclaration d'une variable de type B.
        func( A(b) ) ;  // Appel de la fonction en « transformant » un objet de type B
                        // en objet de type A.
}
```

Question 14 : En se basant sur cette notion, la solution à notre problème consistant à autoriser l'affectation, l'addition… entre vecteurs de « int » et de « float » ou entre tous vecteurs de deux types compatibles, réside dans la création d'un constructeur de copie générique pour la classe . Proposer une implémentation de ce constructeur.

Question 15 : Testez votre constructeur en modifiant votre programme principal de manière à ce que ce dernier utilise les opérations d'addition et d'affectation entre vecteurs de « int » et de « float ». Normalement tout cela devrait fonctionner…

Remarque : Nous vous avons présenté ce mécanisme afin que vous sachiez qu'il existe et que le compilateur l'utilise à votre insu. Cependant, il n'est pas toujours souhaitable de disposer de telles fonctionnalités car parfois, cela peut être cause d'erreurs dans vos programmes. A titre d'exemple, vous pouvez actuellement additionner un vecteur avec un entier. Dans ce cas, le compilateur ne génèrera pas d'erreur mais appellera le constructeur de vecteur à un paramètre et dans ce cas, l'entier sera interprété comme la dimension du vecteur… Cela est-il souhaitable ? Pas vraiment à vrai dire… Pour parer à cette éventualité, le langage C++ dispose d'un mot clé : « **explicit** » qui doit se trouver en tête de la déclaration de vos constructeurs et qui permet de dire au compilateur de ne jamais utiliser ce constructeur de manière implicite. Ce mot clé peut vous éviter de nombreux désagréments…