Les bases de C++ par l'exemple

Philippe Laroque

février 2009

Résumé

Ce petit document décrit, à travers un exemple simple (une classe `Vecteur`), les mécanismes de base nécessaires à la programmation C++.

Les versions successives de l'application illustrent notamment la construction d'objets, l'utilisation des références (v1), la surcharge des opérateurs (v2), la nécessité des friends pour gérer des exceptions locales à l'encapsulation (v3), la gestion propre et efficace de la mémoire avec le destructeur et le copy-constructor (v4 et v5), l'utilisation des exceptions pour gérer les situations à problème (v6, v7 et v8) et enfin les types paramétrés (v9)

1 Le problème à résoudre et la version 1

Il s'agit de créer une classe `Vecteur`, qui représente la notion simple de vecteur de nombres réels. Les fonctionnalités demandées sont les suivantes :

- construction d'un vecteur d'une dimension (nombre de coordonnées) fixée en paramètre : constructeur `Vecteur(int)` ;
- méthode d'accès (lecture et écriture) à la n^{ieme} coordonnée (aucun test de débordement) : méthode `nieme(int)` ;
- méthode d'affichage du contenu du vecteur (sous la forme "$[x_1...x_n]$") : méthode `display()` ;
- méthode de calcul de la somme de deux vecteurs : méthode `somme(Vecteur)`[1] ;
- méthode de calcul du produit scalaire de deux vecteurs : méthode `scalaire(Vecteur)`.

Comme pour tout test unitaire, on a donc à écrire un ensemble de trois fichiers :

1. le fichier `Vecteur.h` contient la déclaration de la classe avec la structure des instances et le profil des méthodes ;
2. le fichier `Vecteur.C` contient le corps des méthodes de la classe ;
3. enfin, le fichier `tVecteur.C` contient le programme de test (main()).

1.1 Le fichier `Vecteur.h`

```
//////////////////////////////////////////////////////////////////////
/**
 * @file Vecteur.h
 * Etude des mécanismes de base de C++.
 *
 * $Id: Vecteur.h 1642 2009-02-12 16:25:33Z phil $
 *
 * On illustre ici, sur l'exemple simple d'un vecteur de nombres reels:
 * - Constructeur (avec arguments)
 * - Destructeur et copy-constructor
 * - surcharge d'opérateurs
 * - fonctions amies
```

[1] En effet, une fonction binaire se traduit par une méthode à un argument : `somme(v1, v2) ;` s'écrit en C++ `v1.somme(v2) ;`

```
 *
 * Version 1: seule difficulté = référence en retour de nieme()
 */
//////////////////////////////////////////////////////////////////////

#ifndef __Vecteur
#define __Vecteur

class Vecteur {

protected:
  double * _tab;                // le tableau des coordonnées
  int _dimension;               // la taille du tableau

public:
  Vecteur (int dim);            // on donne la taille du vecteur
  void display();               // [ x1 ... xN ]
  double& nieme(int i);         // accès à xi
  Vecteur somme(Vecteur v);     // somme de deux vecteurs
  double scalaire(Vecteur v);   // produit scalaire
};

#endif

/*****************************************************************************/
```

1.2 Le fichier `Vecteur.C`

```
//////////////////////////////////////////////////////////////////////
/**
 * @file Vecteur.C
 * Corps des méthodes de la classe Vecteur.
 *
 * $Id: Vecteur.C 1642 2009-02-12 16:25:33Z phil $
 *
 * Le cas est simplifié au maximum:
 * - tous les passages se font par valeur.
 * - pas de destructeur
 * - pas de surcharge d'opérateur
 * - pas de copy-constructor
 * - pas de gestion des débordements d'indice
 *
 * Version 1: seule difficulté = référence en retour de nieme()
 */
//////////////////////////////////////////////////////////////////////

#include <iostream>
using namespace std;
#include "Vecteur.h"

//////////////////////////////////////////////////////////////////////
/**
 * Construit le vecteur avec des 0.
 * @param dim la taille du vecteur
```

```
 ******************************/
//////////////////////////////////////////////////////////////////////
Vecteur :: Vecteur (int dim) {
  _dimension = dim;
  _tab = new double[dim];
  for (int i = 0; i<dim; i++)
    _tab[i] = 0;
}

//////////////////////////////////////////////////////////////////////
/**
 * Affiche le vecteur sous la forme [ x1 ... xN ].
 ******************************/
//////////////////////////////////////////////////////////////////////
void Vecteur :: display() {
  cout << "[ ";
  for (int i = 0; i < _dimension; i++)
    cout << _tab[i] << ' ';
  cout << "]\n";
}

//////////////////////////////////////////////////////////////////////
/**
 * Renvoie la nieme coordonnée du vecteur.
 * Aucune gestion des dépassements
 * @param i l'indice de l'élément cherché (entre 1 et dimension).
 ******************************/
//////////////////////////////////////////////////////////////////////
double& Vecteur :: nieme(int i) {
  return _tab[i-1];
}

//////////////////////////////////////////////////////////////////////
/**
 * Construit le vecteur avec des 0.
 * @param dim la taille du vecteur
 ******************************/
//////////////////////////////////////////////////////////////////////
Vecteur Vecteur :: somme(Vecteur v) {
  Vecteur res (_dimension);
  for (int i = 0; i<_dimension; i++)
    res._tab[i] = _tab[i] + v._tab[i];
  return res;
}

//////////////////////////////////////////////////////////////////////
/**
 * Construit le vecteur avec des 0.
 * @param dim la taille du vecteur
 ******************************/
//////////////////////////////////////////////////////////////////////
double Vecteur :: scalaire(Vecteur v) {
  double res = 0;
  for (int i = 0; i < _dimension; i++)
```

```
    res += _tab[i] * v._tab[i];
  return res;
}

//////////////////////////////////////////////////////////////////////
```

1.3 Le fichier tVecteur.C

```
//////////////////////////////////////////////////////////////////////
/**
 * @file tVecteur.C
 * Test de la classe Vecteur.
 *
 * $Id: tVecteur.C 1642 2009-02-12 16:25:33Z phil $
 *
 * - création de deux vecteurs
 * - modification de coordonnées
 * - affichage d'un vecteur
 * - affichage de la somme et du produit scalaire des deux vecteurs
 */
//////////////////////////////////////////////////////////////////////

#include <iostream>
using namespace std;
#include "Vecteur.h"

//////////////////////////////////////////////////////////////////////
/**
 * Test de la classe Vecteur.
 * - création de deux vecteurs
 * - modification de coordonnées
 * - affichage d'un vecteur
 * - affichage de la somme et du produit scalaire des deux vecteurs
 *
 ******************************/
int main() {
  cout << "sortie attendue:\n"
       << "[ 0 0 0 ]\n"
       << "[ 1 0 0 ]\n"
       << "[ 1 2 0 ]\n"
       << "0\n"
       << "\nsortie effective:\n";
  Vecteur v1(3), v2(3);
  v1.display();
  v1.nieme(1) = 1;
  v1.display();
  v2.nieme(2) = 2;
  v1.somme(v2).display();
  cout << v1.scalaire(v2) << endl;
  return 0;
}

//////////////////////////////////////////////////////////////////////
```

2 Version 2 - surcharge d'opérateurs

Dans cette version, on va utiliser les opérateurs du langage au lieu de méthodes nommées. Par exemple, au lieu d'écrire "`v1.somme(v2)`" on préfère écrire "`v1 + v2`".

On rappelle que pour surcharger un opérateur existant, il suffit de définir la méthode `operator<nomDeLOperateur>` en respectant bien sûr l'arité d'origine de l'opérateur. Par exemple, pour l'addition, on définit la méthode `operator+(Vecteur) ;`

Dans cette version, l'affichage reste la responsabilité de la méthode `display()`, car on ne peut surcharger l'opérateur `<<` en tant que membre (le premier opérande n'est pas un `Vecteur`, mais un flux de sortie – classe `ostream`).

2.1 Le fichier `Vecteur.h`

```
//////////////////////////////////////////////////////////////////////
/**
 * @file Vecteur.h
 * Etude des mécanismes de base de C++.
 *
 * $Id: Vecteur.h 1645 2009-02-16 08:40:30Z phil $
 *
 * On illustre ici, sur l'exemple simple d'un vecteur de nombres reels:
 * - Constructeur (avec arguments)
 * - Destructeur et copy-constructor
 * - surcharge d'opérateurs
 * - fonctions amies
 *
 * Version 2: surcharge d'opérateurs (hors friends)
 */
//////////////////////////////////////////////////////////////////////

#ifndef __Vecteur
#define __Vecteur

class Vecteur {

protected:
  double * _tab;                // le tableau des coordonnées
  int _dimension;               // la taille du tableau

public:
  Vecteur (int dim);            // on donne la taille du vecteur
  void display();               // [ x1 ... xN ]
  double& operator[](int i);    // accès à xi
  Vecteur operator+(Vecteur v); // somme de deux vecteurs
  double operator*(Vecteur v);  // produit scalaire
};

#endif

/*****************************************************************************/
```

2.2 Le fichier `Vecteur.C`

```
////////////////////////////////////////////////////////////////////
/**
 * @file Vecteur.C
 * Corps des méthodes de la classe Vecteur.
 *
 * $Id: Vecteur.C 1645 2009-02-16 08:40:30Z phil $
 *
 * Le cas est simplifié au maximum:
 * - tous les passages se font par valeur.
 * - pas de destructeur
 * - pas de surcharge d'opérateur
 * - pas de copy-constructor
 * - pas de gestion des débordements d'indice
 *
 * Version 2: surcharge d'opérateurs (hors friends)
 */
////////////////////////////////////////////////////////////////////

#include <iostream>
using namespace std;
#include "Vecteur.h"

////////////////////////////////////////////////////////////////////
/**
 * Construit le vecteur avec des 0.
 * @param dim la taille du vecteur
 ******************************/
////////////////////////////////////////////////////////////////////
Vecteur :: Vecteur (int dim) {
  _dimension = dim;
  _tab = new double[dim];
  for (int i = 0; i<dim; i++)
    _tab[i] = 0;
}

////////////////////////////////////////////////////////////////////
/**
 * Affiche le vecteur sous la forme [ x1 ... xN ].
 ******************************/
////////////////////////////////////////////////////////////////////
void Vecteur :: display() {
  cout << "[ ";
  for (int i = 0; i < _dimension; i++)
    cout << _tab[i] << ' ';
  cout << "]\n";
}

////////////////////////////////////////////////////////////////////
/**
 * Renvoie la nieme coordonnée du vecteur.
 * Aucune gestion des dépassements
 * @param i l'indice de l'élément cherché (entre 1 et dimension).
```

```
 ******************************/
//////////////////////////////////////////////////////////////////////
double& Vecteur :: operator[](int i) {
  return _tab[i-1];
}

//////////////////////////////////////////////////////////////////////
/**
 * Construit le vecteur avec des 0.
 * @param dim la taille du vecteur
 ******************************/
//////////////////////////////////////////////////////////////////////
Vecteur Vecteur :: operator+(Vecteur v) {
  Vecteur res (_dimension);
  for (int i = 0; i<_dimension; i++)
    res._tab[i] = _tab[i] + v._tab[i];
  return res;
}

//////////////////////////////////////////////////////////////////////
/**
 * Construit le vecteur avec des 0.
 * @param dim la taille du vecteur
 ******************************/
//////////////////////////////////////////////////////////////////////
double Vecteur :: operator*(Vecteur v) {
  double res = 0;
  for (int i = 0; i < _dimension; i++)
    res += _tab[i] * v._tab[i];
  return res;
}

//////////////////////////////////////////////////////////////////////
```

2.3 Le fichier `tVecteur.C`

```
//////////////////////////////////////////////////////////////////////
/**
 * @file tVecteur.C
 * Test de la classe Vecteur.
 *
 * $Id: tVecteur.C 1645 2009-02-16 08:40:30Z phil $
 *
 * - création de deux vecteurs
 * - modification de coordonnées
 * - affichage d'un vecteur
 * - affichage de la somme et du produit scalaire des deux vecteurs
 */
//////////////////////////////////////////////////////////////////////

#include <iostream>
using namespace std;
#include "Vecteur.h"
```

```
//////////////////////////////////////////////////////////////////////
/**
 * Test de la classe Vecteur.
 * - création de deux vecteurs
 * - modification de coordonnées
 * - affichage d'un vecteur
 * - affichage de la somme et du produit scalaire des deux vecteurs
 *
 ******************************/
int main() {
  cout << "sortie attendue:\n"
       << "[ 0 0 0 ]\n"
       << "[ 1 0 0 ]\n"
       << "[ 1 2 0 ]\n"
       << "0\n"
       << "\nsortie effective:\n";
  Vecteur v1(3), v2(3);
  v1.display();
  v1[1] = 1;
  v1.display();
  v2[2] = 2;
  (v1 + v2).display();
  cout << (v1 *v2) << endl;
  return 0;
}

//////////////////////////////////////////////////////////////////////
```

3 Version 3 - friends

On s'intéresse ici à la surcharge de l'opérateur d'affichage. Puisqu'il ne peut être surchargé comme membre, on va le définir hors des membres de la classe. Si l'on souhaite cependant accéder aux attributs du vecteur, il faut faire un accroc au principe d'encapsulation : on va définir l'opérateur surchargé comme *ami* (`friend`) de la classe `Vecteur`, et lui donner ainsi un accès aux attributs cachés, comme s'il s'agisssait d'une méthode de la classe.

3.1 Le fichier `Vecteur.h`

```
//////////////////////////////////////////////////////////////////////
/**
 * @file Vecteur.h
 * Etude des mécanismes de base de C++.
 *
 * $Id: Vecteur.h 1647 2009-02-16 09:06:38Z phil $
 *
 * On illustre ici, sur l'exemple simple d'un vecteur de nombres reels:
 * - Constructeur (avec arguments)
 * - Destructeur et copy-constructor
 * - surcharge d'opérateurs
 * - fonctions amies
 *
```

```
 * Version 1: seule difficulté = référence en retour de nieme()
 * Version 2: surcharge d'opérateurs (hors friends)
 * Version 3: surcharge des E/S avec les friends (commencer par un opérateur
 * hors de la classe, utilisant "[]", puis expliquer les "friends")
 */
//////////////////////////////////////////////////////////////////////

#ifndef __Vecteur
#define __Vecteur

#include <iostream>
using namespace std;

class Vecteur {

protected:
  double * _tab;                // le tableau des coordonnées
  int _dimension;               // la taille du tableau

public:
  Vecteur (int dim);            // on donne la taille du vecteur
  friend ostream& operator<<(ostream&, Vecteur);
                                // [ x1 ... xN ]
  double& operator[](int i);    // accès à xi
  Vecteur operator+(Vecteur v); // somme de deux vecteurs
  double operator*(Vecteur v);  // produit scalaire
};

#endif

/*************************************************************************/
```

3.2 Le fichier `Vecteur.C`

```
//////////////////////////////////////////////////////////////////////
/**
 * @file Vecteur.C
 * Corps des méthodes de la classe Vecteur.
 *
 * $Id: Vecteur.C 1647 2009-02-16 09:06:38Z phil $
 *
 * Dans cette version:
 * - tous les passages se font par valeur.
 * - pas de destructeur
 * - pas de copy-constructor
 * - pas de gestion des débordements d'indice
 *
 * Version 1: seule difficulté = référence en retour de nieme()
 * Version 2: surcharge d'opérateurs (hors friends)
 * Version 3: surcharge des E/S avec les friends
 */
//////////////////////////////////////////////////////////////////////

#include <iostream>
```

```
using namespace std;
#include "Vecteur.h"

////////////////////////////////////////////////////////////////////
/**
 * Construit le vecteur avec des 0.
 * @param dim la taille du vecteur
 ******************************/
////////////////////////////////////////////////////////////////////
Vecteur :: Vecteur (int dim) {
  _dimension = dim;
  _tab = new double[dim];
  for (int i = 0; i<dim; i++)
    _tab[i] = 0;
}

////////////////////////////////////////////////////////////////////
/**
 * Affiche le vecteur sous la forme [ x1 ... xN ].
 ******************************/
////////////////////////////////////////////////////////////////////
ostream& operator<<(ostream& o, Vecteur v) {
  o << "[ ";
  for (int i = 0; i < v._dimension; i++)
    o << v._tab[i] << ' ';
  return o << "]\n";
}

////////////////////////////////////////////////////////////////////
/**
 * Renvoie la nieme coordonnée du vecteur.
 * Aucune gestion des dépassements
 * @param i l'indice de l'élément cherché (entre 1 et dimension).
 ******************************/
////////////////////////////////////////////////////////////////////
double& Vecteur :: operator[](int i) {
  return _tab[i-1];
}

////////////////////////////////////////////////////////////////////
/**
 * Construit le vecteur avec des 0.
 * @param dim la taille du vecteur
 ******************************/
////////////////////////////////////////////////////////////////////
Vecteur Vecteur :: operator+(Vecteur v) {
  Vecteur res (_dimension);
  for (int i = 0; i<_dimension; i++)
    res._tab[i] = _tab[i] + v._tab[i];
  return res;
}

////////////////////////////////////////////////////////////////////
/**
```

```
 * Construit le vecteur avec des 0.
 * @param dim la taille du vecteur
 ******************************/
//////////////////////////////////////////////////////////////////////
double Vecteur :: operator*(Vecteur v) {
  double res = 0;
  for (int i = 0; i < _dimension; i++)
    res += _tab[i] * v._tab[i];
  return res;
}

//////////////////////////////////////////////////////////////////////
```

3.3 Le fichier tVecteur.C

```
//////////////////////////////////////////////////////////////////////
/**
 * @file tVecteur.C
 * Test de la classe Vecteur.
 *
 * $Id: tVecteur.C 1647 2009-02-16 09:06:38Z phil $
 *
 * - création de deux vecteurs
 * - modification de coordonnées
 * - affichage d'un vecteur
 * - affichage de la somme et du produit scalaire des deux vecteurs
 */
//////////////////////////////////////////////////////////////////////

#include <iostream>
using namespace std;
#include "Vecteur.h"

//////////////////////////////////////////////////////////////////////
/**
 * Test de la classe Vecteur.
 * - création de deux vecteurs
 * - modification de coordonnées
 * - affichage d'un vecteur
 * - affichage de la somme et du produit scalaire des deux vecteurs
 *
 ******************************/
int main() {
  cout << "sortie attendue:\n"
       << "[ 0 0 0 ]\n"
       << "[ 1 0 0 ]\n"
       << "[ 1 2 0 ]\n"
       << "0\n"
       << "\nsortie effective:\n";
  Vecteur v1(3), v2(3);
  cout << v1;
  v1[1] = 1;
  cout << v1;
```

```
  v2[2] = 2;
  cout << (v1 + v2);
  cout << (v1 * v2) << endl;
  return 0;
}

//////////////////////////////////////////////////////////////////////
```

4 Version 4 - gestion de la mémoire (1/2)

Dasn cette version on va résoudre un problème laissé en suspens dans les versions précédentes : le constructeur de Vecteur alloue un tableau physique en mémoire, or cet espace alloué n'est jamais libéré. Il faut donc lui adjoindre un destructeur, qui va récupérer, quand le vecteur disparaît, la mémoire allouée lors de son initialisation par le constructeur.

On essaiera d'abord d'exécuter le programme avec juste l'ajout du destructeur, pour constater que le fonctionnement est alors incorrect. Ceci est dû à un partage de tableaux entre plusieurs vecteurs, ce qui provoque une destructtion multiple de ces tableaux.

On ajoutera ensuite un *copy-constructor* pour assurer l'unicité du tableau physique. On remarque que le paramètre du *copy-constructor* est passé par *référence* : s'il était passé par valeur, l'appel au *copy-constructor* se solderait par une duplication du paramètre, qui ferait donc appel au *copy-constructor*, qui etc.

4.1 Le fichier `Vecteur.h`

```
//////////////////////////////////////////////////////////////////////
/**
 * @file Vecteur.h
 * Etude des mécanismes de base de C++.
 *
 * $Id: Vecteur.h 1649 2009-02-16 09:22:47Z phil $
 *
 * On illustre ici, sur l'exemple simple d'un vecteur de nombres reels:
 * - Constructeur (avec arguments)
 * - Destructeur et copy-constructor
 * - surcharge d'opérateurs
 * - fonctions amies
 *
 * Version 1: seule difficulté = référence en retour de nieme()
 * Version 2: surcharge d'opérateurs (hors friends)
 * Version 3: surcharge des E/S avec les friends (commencer par un opérateur
 * hors de la classe, utilisant "[]", puis expliquer les "friends")
 * Version 4: ajout du destructeur et du copy-constructor
 * (le destructeur seul provoque une erreur d'exécution)
 */
//////////////////////////////////////////////////////////////////////

#ifndef __Vecteur
#define __Vecteur

#include <iostream>
using namespace std;
```

```
class Vecteur {

protected:
  double * _tab;                // le tableau des coordonnées
  int _dimension;               // la taille du tableau

public:
  Vecteur (int dim);            // on donne la taille du vecteur
  ~Vecteur();                   // pour récupérer la mémoire
  Vecteur(const Vecteur&);      // pour dupliquer correctement un Vecteur
  friend ostream& operator<<(ostream&, Vecteur);
                                // [ x1 ... xN ]
  double& operator[](int i);    // accès à xi
  Vecteur operator+(Vecteur v); // somme de deux vecteurs
  double operator*(Vecteur v);  // produit scalaire
};

#endif

/*****************************************************************************/
```

4.2 Le fichier `Vecteur.C`

```
//////////////////////////////////////////////////////////////////////
/**
 * @file Vecteur.C
 * Corps des méthodes de la classe Vecteur.
 *
 * $Id: Vecteur.C 1649 2009-02-16 09:22:47Z phil $
 *
 * Dans cette version:
 * - tous les passages se font par valeur.
 * - pas de gestion des débordements d'indice
 *
 * Version 1: seule difficulté = référence en retour de nieme()
 * Version 2: surcharge d'opérateurs (hors friends)
 * Version 3: surcharge des E/S avec les friends
 * Version 4: ajout du destructeur et du copy-constructor
 * (le destructeur seul provoque une erreur d'exécution)
 */
//////////////////////////////////////////////////////////////////////

#include <iostream>
using namespace std;
#include "Vecteur.h"

//////////////////////////////////////////////////////////////////////
/**
 * Construit le vecteur avec des 0.
 * @param dim la taille du vecteur
 ******************************/
//////////////////////////////////////////////////////////////////////
Vecteur :: Vecteur (int dim) {
  _dimension = dim;
```

```
  _tab = new double[dim];
  for (int i = 0; i<dim; i++)
    _tab[i] = 0;
}

////////////////////////////////////////////////////////////////////
/**
 * Construit le vecteur a partir d'un vecteur existant (recopie).
 * @param v le vecteur à copier.
 ******************************/
////////////////////////////////////////////////////////////////////
Vecteur :: Vecteur (const Vecteur& v) {
  _dimension = v._dimension;
  _tab = new double[_dimension];
  for (int i = 0; i<_dimension; i++)
    _tab[i] = v._tab[i];
}
////////////////////////////////////////////////////////////////////
/**
 * Récupère la mémoire allouée à la création du vecteur.
 * On remplit la zone de "0" pour mettre en évidence le passage dans le
 * destructeur.
 ******************************/
////////////////////////////////////////////////////////////////////
Vecteur :: ~Vecteur() {
  for (int i = 0; i<_dimension; i++)
    _tab[i] = 0;
  delete[] _tab;
}

////////////////////////////////////////////////////////////////////
/**
 * Affiche le vecteur sous la forme [ x1 ... xN ].
 ******************************/
////////////////////////////////////////////////////////////////////
ostream& operator<<(ostream& o, Vecteur v) {
  o << "[ ";
  for (int i = 0; i < v._dimension; i++)
    o << v._tab[i] << ' ';
  return o << "]\n";
}

////////////////////////////////////////////////////////////////////
/**
 * Renvoie la nieme coordonnée du vecteur.
 * Aucune gestion des dépassements
 * @param i l'indice de l'élément cherché (entre 1 et dimension).
 ******************************/
////////////////////////////////////////////////////////////////////
double& Vecteur :: operator[](int i) {
  return _tab[i-1];
}

////////////////////////////////////////////////////////////////////
```

```
/**
 * Construit le vecteur avec des 0.
 * @param dim la taille du vecteur
 ******************************/
//////////////////////////////////////////////////////////////////////
Vecteur Vecteur :: operator+(Vecteur v) {
  Vecteur res (_dimension);
  for (int i = 0; i<_dimension; i++)
    res._tab[i] = _tab[i] + v._tab[i];
  return res;
}

//////////////////////////////////////////////////////////////////////
/**
 * Construit le vecteur avec des 0.
 * @param dim la taille du vecteur
 ******************************/
//////////////////////////////////////////////////////////////////////
double Vecteur :: operator*(Vecteur v) {
  double res = 0;
  for (int i = 0; i < _dimension; i++)
    res += _tab[i] * v._tab[i];
  return res;
}

//////////////////////////////////////////////////////////////////////
```

4.3 Le fichier `tVecteur.C`

```
//////////////////////////////////////////////////////////////////////
/**
 * @file tVecteur.C
 * Test de la classe Vecteur.
 *
 * $Id: tVecteur.C 1647 2009-02-16 09:06:38Z phil $
 *
 * - création de deux vecteurs
 * - modification de coordonnées
 * - affichage d'un vecteur
 * - affichage de la somme et du produit scalaire des deux vecteurs
 */
//////////////////////////////////////////////////////////////////////

#include <iostream>
using namespace std;
#include "Vecteur.h"

//////////////////////////////////////////////////////////////////////
/**
 * Test de la classe Vecteur.
 * - création de deux vecteurs
 * - modification de coordonnées
 * - affichage d'un vecteur
```

```
 * - affichage de la somme et du produit scalaire des deux vecteurs
 *
 ******************************/
int main() {
  cout << "sortie attendue:\n"
       << "[ 0 0 0 ]\n"
       << "[ 1 0 0 ]\n"
       << "[ 1 2 0 ]\n"
       << "0\n"
       << "\nsortie effective:\n";
  Vecteur v1(3), v2(3);
  cout << v1;
  v1[1] = 1;
  cout << v1;
  v2[2] = 2;
  cout << (v1 + v2);
  cout << (v1 * v2) << endl;
  return 0;
}

//////////////////////////////////////////////////////////////////////
```

5 Version 5 - gestion de la mémoire (2/2)

Il s'agit ici d'optimiser la gestion de la mémoire. On constate, en laissant un message de trace dans les constructeurs et dans le destructeur, que le programme de test provoque la construction de 7 objets (3 par appel à `Vecteur(int)`, 4 par appel à `Vecteur(const Vecteur&)`). Cela est dû au passage par valeur des vecteurs dans les méthodes de la classe. En modifiant le profil des méthodes pour qu'elles passent le vecteur par référence (`const`, puisqu'on ne veut pas modifier le vecteur dans le coprs de ces méthodes), on retombe à 3 objets créés !

5.1 Le fichier `Vecteur.h`

```
//////////////////////////////////////////////////////////////////////
/**
 * @file Vecteur.h
 * Etude des mécanismes de base de C++.
 *
 * $Id: Vecteur.h 1651 2009-02-16 09:34:14Z phil $
 *
 * On illustre ici, sur l'exemple simple d'un vecteur de nombres reels:
 * - Constructeur (avec arguments)
 * - Destructeur et copy-constructor
 * - surcharge d'opérateurs
 * - fonctions amies
 *
 * Version 1: seule difficulté = référence en retour de nieme()
 * Version 2: surcharge d'opérateurs (hors friends)
 * Version 3: surcharge des E/S avec les friends (commencer par un opérateur
 * hors de la classe, utilisant "[]", puis expliquer les "friends")
 * Version 4: ajout du destructeur et du copy-constructor
 * (le destructeur seul provoque une erreur d'exécution)
```

```
 * Version 5: passage par référence quand ça évite des duplications
 * (3 Vecteur(int) + 4 Vecteur(Vecteur) -> 3 Vecteur(int)!
 */
//////////////////////////////////////////////////////////////////////

#ifndef __Vecteur
#define __Vecteur

#include <iostream>
using namespace std;

class Vecteur {

protected:
  double * _tab;                // le tableau des coordonnées
  int _dimension;               // la taille du tableau

public:
  Vecteur (int dim);            // on donne la taille du vecteur
  ~Vecteur();                   // pour récupérer la mémoire
  Vecteur(const Vecteur&);      // pour dupliquer correctement un Vecteur
  friend ostream& operator<<(ostream&, const Vecteur&);
                                // [ x1 ... xN ]
  double& operator[](int i);    // accès à xi
  Vecteur operator+(const Vecteur& v);
                                // somme de deux vecteurs
  double operator*(const Vecteur& v);
                                // produit scalaire
};

#endif

/*****************************************************************************/
```

5.2 Le fichier `Vecteur.C`

```
//////////////////////////////////////////////////////////////////////
/**
 * @file Vecteur.C
 * Corps des méthodes de la classe Vecteur.
 *
 * $Id: Vecteur.C 1651 2009-02-16 09:34:14Z phil $
 *
 * Dans cette version:
 *
 * - pas de gestion des débordements d'indice
 *
 * Version 1: seule difficulté = référence en retour de nieme()
 * Version 2: surcharge d'opérateurs (hors friends)
 * Version 3: surcharge des E/S avec les friends
 * Version 4: ajout du destructeur et du copy-constructor
 * (le destructeur seul provoque une erreur d'exécution)
 * Version 5: passage par référence quand ça évite des duplications
 * (3 Vecteur(int) + 4 Vecteur(Vecteur) -> 3 Vecteur(int)!
```

```
 */
//////////////////////////////////////////////////////////////////////

#include <iostream>
using namespace std;
#include "Vecteur.h"

//////////////////////////////////////////////////////////////////////
/**
 * Construit le vecteur avec des 0.
 * @param dim la taille du vecteur
 ******************************/
//////////////////////////////////////////////////////////////////////
Vecteur :: Vecteur (int dim) {
  cerr << "Vecteur(int)!\n";
  _dimension = dim;
  _tab = new double[dim];
  for (int i = 0; i<dim; i++)
    _tab[i] = 0;
}

//////////////////////////////////////////////////////////////////////
/**
 * Construit le vecteur a partir d'un vecteur existant (recopie).
 * @param v le vecteur à copier.
 ******************************/
//////////////////////////////////////////////////////////////////////
Vecteur :: Vecteur (const Vecteur& v) {
  cerr << "Vecteur(Vecteur)!\n";
  _dimension = v._dimension;
  _tab = new double[_dimension];
  for (int i = 0; i<_dimension; i++)
    _tab[i] = v._tab[i];
}
//////////////////////////////////////////////////////////////////////
/**
 * Récupère la mémoire allouée à la création du vecteur.
 * On remplit la zone de "0" pour mettre en évidence le passage dans le
 * destructeur.
 ******************************/
//////////////////////////////////////////////////////////////////////
Vecteur :: ~Vecteur() {
  cerr << "~Vecteur()!\n";
  for (int i = 0; i<_dimension; i++)
    _tab[i] = 0;
  delete[] _tab;
}

//////////////////////////////////////////////////////////////////////
/**
 * Affiche le vecteur sous la forme [ x1 ... xN ].
 ******************************/
//////////////////////////////////////////////////////////////////////
ostream& operator<<(ostream& o, const Vecteur& v) {
```

```
  o << "[ ";
  for (int i = 0; i < v._dimension; i++)
    o << v._tab[i] << ' ';
  return o << "]\n";
}

//////////////////////////////////////////////////////////////////////
/**
 * Renvoie la nieme coordonnée du vecteur.
 * Aucune gestion des dépassements
 * @param i l'indice de l'élément cherché (entre 1 et dimension).
 ******************************/
//////////////////////////////////////////////////////////////////////
double& Vecteur :: operator[](int i) {
  return _tab[i-1];
}

//////////////////////////////////////////////////////////////////////
/**
 * Construit le vecteur avec des 0.
 * @param dim la taille du vecteur
 ******************************/
//////////////////////////////////////////////////////////////////////
Vecteur Vecteur :: operator+(const Vecteur& v) {
  Vecteur res (_dimension);
  for (int i = 0; i<_dimension; i++)
    res._tab[i] = _tab[i] + v._tab[i];
  return res;
}

//////////////////////////////////////////////////////////////////////
/**
 * Construit le vecteur avec des 0.
 * @param dim la taille du vecteur
 ******************************/
//////////////////////////////////////////////////////////////////////
double Vecteur :: operator*(const Vecteur& v) {
  double res = 0;
  for (int i = 0; i < _dimension; i++)
    res += _tab[i] * v._tab[i];
  return res;
}

//////////////////////////////////////////////////////////////////////
```

5.3 Le fichier tVecteur.C

```
//////////////////////////////////////////////////////////////////////
/**
 * @file tVecteur.C
 * Test de la classe Vecteur.
 *
 * $Id: tVecteur.C 1647 2009-02-16 09:06:38Z phil $
```

```
 *
 * - création de deux vecteurs
 * - modification de coordonnées
 * - affichage d'un vecteur
 * - affichage de la somme et du produit scalaire des deux vecteurs
 */
//////////////////////////////////////////////////////////////////////

#include <iostream>
using namespace std;
#include "Vecteur.h"

//////////////////////////////////////////////////////////////////////
/**
 * Test de la classe Vecteur.
 * - création de deux vecteurs
 * - modification de coordonnées
 * - affichage d'un vecteur
 * - affichage de la somme et du produit scalaire des deux vecteurs
 *
 ******************************/
int main() {
  cout << "sortie attendue:\n"
       << "[ 0 0 0 ]\n"
       << "[ 1 0 0 ]\n"
       << "[ 1 2 0 ]\n"
       << "0\n"
       << "\nsortie effective:\n";
  Vecteur v1(3), v2(3);
  cout << v1;
  v1[1] = 1;
  cout << v1;
  v2[2] = 2;
  cout << (v1 + v2);
  cout << (v1 * v2) << endl;
  return 0;
}

//////////////////////////////////////////////////////////////////////
```

6 Version 6 - Gestion des débordements d'indice

On utilise ici une variable de classe (`static`) pour gérer les débordements d'indice dans le vecteur. En effet, puisque l'opérateur `[]` retourne une référence, il faut retourner un objet licite en cas d'erreur. On aurait pu renvoyer la première coordonnée du vecteur, mais cela modifie le vecteur et ce n'est sans doute pas souhaitable ; on aurait aussi pu définir un attribut ordinaire destiné spécifiquement à cette situation, mais cela aurait consommé inutilement de la place mémoire : un seul objet suffit pour l'ensemble des vecteurs, d'où l'attribut `static`.

À noter : les attributs `static` sont définis hors de la classe, comme le corps des méthodes (donc dans le fichier "`.C`")

6.1 Le fichier Vecteur.h

```
//////////////////////////////////////////////////////////////////////
/**
 * @file Vecteur.h
 * Etude des mécanismes de base de C++.
 *
 * $Id: Vecteur.h 1653 2009-02-16 09:44:46Z phil $
 *
 * On illustre ici, sur l'exemple simple d'un vecteur de nombres reels:
 * - Constructeur (avec arguments)
 * - Destructeur et copy-constructor
 * - surcharge d'opérateurs
 * - fonctions amies
 *
 * Version 1: seule difficulté = référence en retour de nieme()
 * Version 2: surcharge d'opérateurs (hors friends)
 * Version 3: surcharge des E/S avec les friends (commencer par un opérateur
 * hors de la classe, utilisant "[]", puis expliquer les "friends")
 * Version 4: ajout du destructeur et du copy-constructor
 * (le destructeur seul provoque une erreur d'exécution)
 * Version 5: passage par référence quand ça évite des duplications
 * (3 Vecteur(int) + 4 Vecteur(Vecteur) -> 3 Vecteur(int)!
 * Version 6: gestion des débordements d'indice par une variable statique
 */
//////////////////////////////////////////////////////////////////////

#ifndef __Vecteur
#define __Vecteur

#include <iostream>
using namespace std;

class Vecteur {

protected:
  double * _tab;                // le tableau des coordonnées
  int _dimension;               // la taille du tableau
  static double Trash;          // "poubelle" pour les débordements d'indice

public:
  Vecteur (int dim);            // on donne la taille du vecteur
  ~Vecteur();                   // pour récupérer la mémoire
  Vecteur(const Vecteur&);      // pour dupliquer correctement un Vecteur
  friend ostream& operator<<(ostream&, const Vecteur&);
                                // [ x1 ... xN ]
  double& operator[](int i);    // accès à xi
  Vecteur operator+(const Vecteur& v);
                                // somme de deux vecteurs
  double operator*(const Vecteur& v);
                                // produit scalaire
};

#endif
```

```
/*************************************************************************/
```

6.2 Le fichier Vecteur.C

```
//////////////////////////////////////////////////////////////////////
/**
 * @file Vecteur.C
 * Corps des méthodes de la classe Vecteur.
 *
 * $Id: Vecteur.C 1653 2009-02-16 09:44:46Z phil $
 *
 * Dans cette version:
 *
 * - pas de gestion des débordements d'indice
 *
 * Version 1: seule difficulté = référence en retour de nieme()
 * Version 2: surcharge d'opérateurs (hors friends)
 * Version 3: surcharge des E/S avec les friends
 * Version 4: ajout du destructeur et du copy-constructor
 * (le destructeur seul provoque une erreur d'exécution)
 * Version 5: passage par référence quand ça évite des duplications
 * (3 Vecteur(int) + 4 Vecteur(Vecteur) -> 3 Vecteur(int)!
 * Version 6: gestion des débordements d'indice par une variable statique
 */
//////////////////////////////////////////////////////////////////////

#include <iostream>
using namespace std;
#include "Vecteur.h"


//////////////////////////////////////////////////////////////////////
/**
 * Permet de retourner un objet licite en cas de débordement d'indice
 * dans [].
 ******************************/
//////////////////////////////////////////////////////////////////////
double Vecteur::Trash = 0;

//////////////////////////////////////////////////////////////////////
/**
 * Construit le vecteur avec des 0.
 * @param dim la taille du vecteur
 ******************************/
//////////////////////////////////////////////////////////////////////
Vecteur :: Vecteur (int dim) {
  _dimension = dim;
  _tab = new double[dim];
  for (int i = 0; i<dim; i++)
    _tab[i] = 0;
}

//////////////////////////////////////////////////////////////////////
```

```
/**
 * Construit le vecteur a partir d'un vecteur existant (recopie).
 * @param v le vecteur à copier.
 ******************************/
//////////////////////////////////////////////////////////////////////
Vecteur :: Vecteur (const Vecteur& v) {
  _dimension = v._dimension;
  _tab = new double[_dimension];
  for (int i = 0; i<_dimension; i++)
    _tab[i] = v._tab[i];
}
//////////////////////////////////////////////////////////////////////
/**
 * Récupère la mémoire allouée à la création du vecteur.
 * On remplit la zone de "0" pour mettre en évidence le passage dans le
 * destructeur.
 ******************************/
//////////////////////////////////////////////////////////////////////
Vecteur :: ~Vecteur() {
  for (int i = 0; i<_dimension; i++)
    _tab[i] = 0;
  delete[] _tab;
}

//////////////////////////////////////////////////////////////////////
/**
 * Affiche le vecteur sous la forme [ x1 ... xN ].
 ******************************/
//////////////////////////////////////////////////////////////////////
ostream& operator<<(ostream& o, const Vecteur& v) {
  o << "[ ";
  for (int i = 0; i < v._dimension; i++)
    o << v._tab[i] << ' ';
  return o << "]\n";
}

//////////////////////////////////////////////////////////////////////
/**
 * Renvoie la nieme coordonnée du vecteur.
 * Aucune gestion des dépassements
 * @param i l'indice de l'élément cherché (entre 1 et dimension).
 ******************************/
//////////////////////////////////////////////////////////////////////
double& Vecteur :: operator[](int i) {
  if (i<1 || i>_dimension) {
    cerr << "débordement d'indice\n";
    return Trash;
  }
  return _tab[i-1];
}

//////////////////////////////////////////////////////////////////////
/**
 * Construit le vecteur avec des 0.
```

```
 * @param dim la taille du vecteur
 ******************************/
//////////////////////////////////////////////////////////////////////
Vecteur Vecteur :: operator+(const Vecteur& v) {
  Vecteur res (_dimension);
  for (int i = 0; i<_dimension; i++)
    res._tab[i] = _tab[i] + v._tab[i];
  return res;
}

//////////////////////////////////////////////////////////////////////
/**
 * Construit le vecteur avec des 0.
 * @param dim la taille du vecteur
 ******************************/
//////////////////////////////////////////////////////////////////////
double Vecteur :: operator*(const Vecteur& v) {
  double res = 0;
  for (int i = 0; i < _dimension; i++)
    res += _tab[i] * v._tab[i];
  return res;
}

//////////////////////////////////////////////////////////////////////
```

6.3 Le fichier `tVecteur.C`

```
//////////////////////////////////////////////////////////////////////
/**
 * @file tVecteur.C
 * Test de la classe Vecteur.
 *
 * $Id: tVecteur.C 1653 2009-02-16 09:44:46Z phil $
 *
 * - création de deux vecteurs
 * - modification de coordonnées
 * - affichage d'un vecteur
 * - affichage de la somme et du produit scalaire des deux vecteurs
 */
//////////////////////////////////////////////////////////////////////

#include <iostream>
using namespace std;
#include "Vecteur.h"

//////////////////////////////////////////////////////////////////////
/**
 * Test de la classe Vecteur.
 * - création de deux vecteurs
 * - modification de coordonnées et vérification du débordement
 * - affichage d'un vecteur
 * - affichage de la somme et du produit scalaire des deux vecteurs
 *
```

```
 ******************************/
int main() {
  cout << "sortie attendue:\n"
       << "[ 0 0 0 ]\n"
       << "[ 1 0 0 ]\n"
       << "[ 1 2 0 ]\n"
       << "débordement d'indice\n0\n"
       << "0\n"
       << "\nsortie effective:\n";
  Vecteur v1(3), v2(3);
  cout << v1;
  v1[1] = 1;
  cout << v1;
  v2[2] = 2;
  cout << (v1 + v2);
  cout << v1[10] << endl;
  cout << (v1 * v2) << endl;
  return 0;
}

//////////////////////////////////////////////////////////////////////
```

7 Version 7 - Les exceptions (1/2)

Dans cette version, on va définir des objets exceptions pour gérer plus proprement les situations "aux limites" (débordements d'indice, tailles de vecteur incompatibles etc.). Dans la mesure où aucun traitement spécial n'est fait dans le `main()` (*cf.* section suivante), le programme de cette version 7 se termine sur un message d'erreur.

7.1 Le fichier `Vecteur.h`

```
//////////////////////////////////////////////////////////////////////
/**
 * @file Vecteur.h
 * Etude des mécanismes de base de C++.
 *
 * $Id: Vecteur.h 1655 2009-02-16 10:04:20Z phil $
 *
 * On illustre ici, sur l'exemple simple d'un vecteur de nombres reels:
 * - Constructeur (avec arguments)
 * - Destructeur et copy-constructor
 * - surcharge d'opérateurs
 * - fonctions amies
 *
 * Version 1: seule difficulté = référence en retour de nieme()
 * Version 2: surcharge d'opérateurs (hors friends)
 * Version 3: surcharge des E/S avec les friends (commencer par un opérateur
 *      hors de la classe, utilisant "[]", puis expliquer les "friends")
 * Version 4: ajout du destructeur et du copy-constructor
 *      (le destructeur seul provoque une erreur d'exécution)
 * Version 5: passage par référence quand ça évite des duplications
 *      (3 Vecteur(int) + 4 Vecteur(Vecteur) -> 3 Vecteur(int)!
```

```
 * Version 6: gestion des débordements d'indice par une variable statique
 * Version 7: gestion des débordements / pbs de taille par exceptions
 *      (sauf main())
 */
//////////////////////////////////////////////////////////////////////

#ifndef __Vecteur
#define __Vecteur

#include <iostream>
using namespace std;

//////////////////////////////////////////////////////////////////////
/**
 * La classe IndiceException.
 * - utilisée lors des débordements d'indice.
 ******************************/
class IndiceException {
public:
  IndiceException(int i, int max) : indice(i), taille(max) {}
  const int indice;
  const int taille;
};

//////////////////////////////////////////////////////////////////////
/**
 * La classe TailleException.
 * - utilisée lors d'une opération sur deux vecteurs de taille distincte.
 ******************************/
class TailleException {
public:
  TailleException(int t1, int t2) : taille1(t1), taille2(t2) {}
  const int taille1;
  const int taille2;

};

//////////////////////////////////////////////////////////////////////
/**
 * La classe Vecteur
 *
 ******************************/
class Vecteur {

protected:
  double * _tab;                // le tableau des coordonnées
  int _dimension;               // la taille du tableau
  static double Trash;          // "poubelle" pour les débordements d'indice

public:
  Vecteur (int dim);            // on donne la taille du vecteur
  ~Vecteur();                   // pour récupérer la mémoire
  Vecteur(const Vecteur&);      // pour dupliquer correctement un Vecteur
  friend ostream& operator<<(ostream&, const Vecteur&);
```

```
  // [ x1 ... xN ]
  double& operator[](int i) throw (IndiceException);
  // accès à xi
  Vecteur operator+(const Vecteur& v) throw (TailleException);
  // somme de deux vecteurs
  double operator*(const Vecteur& v) throw (TailleException);
  // produit scalaire
};

#endif

/*****************************************************************************/
```

7.2 Le fichier Vecteur.C

```
//////////////////////////////////////////////////////////////////////
/**
 * @file Vecteur.C
 * Corps des méthodes de la classe Vecteur.
 *
 * $Id: Vecteur.C 1655 2009-02-16 10:04:20Z phil $
 *
 * Version 1: seule difficulté = référence en retour de nieme()
 * Version 2: surcharge d'opérateurs (hors friends)
 * Version 3: surcharge des E/S avec les friends
 * Version 4: ajout du destructeur et du copy-constructor
 *      (le destructeur seul provoque une erreur d'exécution)
 * Version 5: passage par référence quand ça évite des duplications
 *      (3 Vecteur(int) + 4 Vecteur(Vecteur) -> 3 Vecteur(int)!
 * Version 6: gestion des débordements d'indice par une variable statique
 * Version 7: gestion des débordements / pbs de taille par exceptions
 *     (sauf main())
 */
//////////////////////////////////////////////////////////////////////

#include <iostream>
using namespace std;
#include "Vecteur.h"


//////////////////////////////////////////////////////////////////////
/**
 * Permet de retourner un objet licite en cas de débordement d'indice
 * dans [].
 ******************************/
//////////////////////////////////////////////////////////////////////
double Vecteur::Trash = 0;

//////////////////////////////////////////////////////////////////////
/**
 * Construit le vecteur avec des 0.
 * @param dim la taille du vecteur
 ******************************/
//////////////////////////////////////////////////////////////////////
```

```
Vecteur :: Vecteur (int dim) {
  _dimension = dim;
  _tab = new double[dim];
  for (int i = 0; i<dim; i++)
    _tab[i] = 0;
}

////////////////////////////////////////////////////////////////////
/**
 * Construit le vecteur a partir d'un vecteur existant (recopie).
 * @param v le vecteur à copier.
 ******************************/
////////////////////////////////////////////////////////////////////
Vecteur :: Vecteur (const Vecteur& v) {
  _dimension = v._dimension;
  _tab = new double[_dimension];
  for (int i = 0; i<_dimension; i++)
    _tab[i] = v._tab[i];
}
////////////////////////////////////////////////////////////////////
/**
 * Récupère la mémoire allouée à la création du vecteur.
 * On remplit la zone de "0" pour mettre en évidence le passage dans le
 * destructeur.
 ******************************/
////////////////////////////////////////////////////////////////////
Vecteur :: ~Vecteur() {
  for (int i = 0; i<_dimension; i++)
    _tab[i] = 0;
  delete[] _tab;
}

////////////////////////////////////////////////////////////////////
/**
 * Affiche le vecteur sous la forme [ x1 ... xN ].
 ******************************/
////////////////////////////////////////////////////////////////////
ostream& operator<<(ostream& o, const Vecteur& v) {
  o << "[ ";
  for (int i = 0; i < v._dimension; i++)
    o << v._tab[i] << ' ';
  return o << "]\n";
}

////////////////////////////////////////////////////////////////////
/**
 * Renvoie la nieme coordonnée du vecteur.
 * Aucune gestion des dépassements
 * @param i l'indice de l'élément cherché (entre 1 et dimension).
 ******************************/
////////////////////////////////////////////////////////////////////
double& Vecteur :: operator[](int i) throw (IndiceException) {
  if (i<1 || i>_dimension) throw (IndiceException(i,_dimension));
  return _tab[i-1];
```

```
}

////////////////////////////////////////////////////////////////////
/**
 * Construit le vecteur avec des 0.
 * @param dim la taille du vecteur
 ******************************/
////////////////////////////////////////////////////////////////////
Vecteur Vecteur :: operator+(const Vecteur& v) throw (TailleException) {
  if (_dimension != v._dimension)
    throw (TailleException(_dimension, v._dimension));
  Vecteur res (_dimension);
  for (int i = 0; i<_dimension; i++)
    res._tab[i] = _tab[i] + v._tab[i];
  return res;
}

////////////////////////////////////////////////////////////////////
/**
 * Construit le vecteur avec des 0.
 * @param dim la taille du vecteur
 ******************************/
////////////////////////////////////////////////////////////////////
double Vecteur :: operator*(const Vecteur& v) throw (TailleException) {
  if (_dimension != v._dimension)
    throw (TailleException(_dimension, v._dimension));
  double res = 0;
  for (int i = 0; i < _dimension; i++)
    res += _tab[i] * v._tab[i];
  return res;
}

////////////////////////////////////////////////////////////////////
```

7.3 Le fichier tVecteur.C

```
////////////////////////////////////////////////////////////////////
/**
 * @file tVecteur.C
 * Test de la classe Vecteur.
 *
 * $Id: tVecteur.C 1655 2009-02-16 10:04:20Z phil $
 *
 * - création de deux vecteurs
 * - modification de coordonnées
 * - affichage d'un vecteur
 * - affichage de la somme et du produit scalaire des deux vecteurs
 */
////////////////////////////////////////////////////////////////////

#include <iostream>
using namespace std;
#include "Vecteur.h"
```

```
//////////////////////////////////////////////////////////////////////
/**
 * Test de la classe Vecteur.
 * - création de deux vecteurs
 * - modification de coordonnées et vérification du débordement
 * - affichage d'un vecteur
 * - affichage de la somme et du produit scalaire des deux vecteurs
 *
 ******************************/
int main() {
  cout << "sortie attendue:\n"
       << "[ 0 0 0 ]\n"
       << "[ 1 0 0 ]\n"
       << "[ 1 2 0 ]\n"
       << "terminate called after throwing an instance of 'IndiceException'\n"
       << "Aborted\n"           // exception non gérée
       << "\nsortie effective:\n";
  Vecteur v1(3), v2(3);
  cout << v1;
  v1[1] = 1;
  cout << v1;
  v2[2] = 2;
  cout << (v1 + v2);
  cout << v1[10] << endl;
  cout << (v1 * v2) << endl;
  return 0;
}

//////////////////////////////////////////////////////////////////////
```

8 Version 8 - Les exceptions (2/2)

Cette fois, on choisit de récupérer dans le main() l'exception liée au débordement d'indice. Même si le bloc `try` est suivi de deux `catch`, seul le `catch` correspondant à l'exception `IndiceException` est exécuté.

8.1 Le fichier `tVecteur.C`

```
//////////////////////////////////////////////////////////////////////
/**
 * @file tVecteur.C
 * Test de la classe Vecteur.
 *
 * $Id: tVecteur.C 1657 2009-02-16 10:17:08Z phil $
 *
 * - création de deux vecteurs
 * - modification de coordonnées
 * - affichage d'un vecteur
 * - affichage de la somme et du produit scalaire des deux vecteurs
 * Version 8: gestion des débordements dans main()
 */
```

```
//////////////////////////////////////////////////////////////////////

#include <iostream>
using namespace std;
#include "Vecteur.h"

//////////////////////////////////////////////////////////////////////
/**
 * Test de la classe Vecteur.
 * - création de deux vecteurs
 * - modification de coordonnées et vérification du débordement
 * - affichage d'un vecteur
 * - affichage de la somme et du produit scalaire des deux vecteurs
 *
 ******************************/
int main() {
  cout << "sortie attendue:\n"
       << "[ 0 0 0 ]\n"
       << "[ 1 0 0 ]\n"
       << "[ 1 2 0 ]\n"
       << "IndiceException: tentative d'accès à l'indice 10 "
       << "sur un vecteur de taille 3\n"// exception gérée
       << "\nsortie effective:\n";
  Vecteur v1(3), v2(3);
  cout << v1;
  try {
    v1[1] = 1;
    cout << v1;
    v2[2] = 2;
    cout << (v1 + v2);
    cout << v1[10] << endl;
    cout << (v1 * v2) << endl;
  }
  catch (TailleException te) {
    cerr << "Une TailleException\n";
    return 1;
  }
  catch(IndiceException ie) {
    cerr <<"IndiceException: tentative d'accès à l'indice "
         << ie.indice
         << " sur un vecteur de taille "
         << ie.taille << endl;
    return 2;
  }
  return 0;
}

//////////////////////////////////////////////////////////////////////
```

9 Version 9 - Types paramétrés

Dans cette dernière version, on rend le type des éléments du vecteur (jusque-là des `double`) paramètre du type `Vecteur`, en utilisant la notion de *template* de classe.

Deux remarques importantes (outre la lourdeur du code, qui saute immédiatement aux yeux !) :

1. Dans le cas des templates, le code de la classe générique doit être *intégralement* disponible lorsque le compilateur rencontre une demande d'instanciation du type : dans le `main()`, le compilateur doit créer une classe `Vecteur` contenant des `double` ; il doit donc générer, à partir du code intégral de la template, le code intégral de cette classe. C'est ce qui explique l'inclusion "inversée" des fichiers : le fichier ".h" fait un `#include` du fichier ".C", et le `main` constitue donc la seule unité de compilation (pas besoin de `makefile` dans ce cas).
2. Lorsqu'une template de fonction est intégrée dans une template de classe (c'est le cas de l'opérateur d'affichage ici), le paramètre de la généricité pour la fonction doit être distinct de celui utilisé pour la classe.

9.1 Le fichier `Vecteur.h`

```
//////////////////////////////////////////////////////////////////////
/**
 * @file Vecteur.h
 * Etude des mécanismes de base de C++.
 *
 * $Id: Vecteur.h 1659 2009-02-16 14:28:29Z phil $
 *
 * On illustre ici, sur l'exemple simple d'un vecteur:
 * - Constructeur (avec arguments)
 * - Destructeur et copy-constructor
 * - surcharge d'opérateurs
 * - fonctions amies
 * - types paramétrés
 *
 * Version 1: seule difficulté = référence en retour de nieme()
 * Version 2: surcharge d'opérateurs (hors friends)
 * Version 3: surcharge des E/S avec les friends (commencer par un opérateur
 *      hors de la classe, utilisant "[]", puis expliquer les "friends")
 * Version 4: ajout du destructeur et du copy-constructor
 *      (le destructeur seul provoque une erreur d'exécution)
 * Version 5: passage par référence quand ça évite des duplications
 *      (3 Vecteur(int) + 4 Vecteur(Vecteur) -> 3 Vecteur(int)!
 * Version 6: gestion des débordements d'indice par une variable statique
 * Version 7: gestion des débordements / pbs de taille par exceptions
 *      (sauf main(): v8)
 * Version 9: type paramétré
 */
//////////////////////////////////////////////////////////////////////

#ifndef __Vecteur
#define __Vecteur

#include <iostream>
using namespace std;

//////////////////////////////////////////////////////////////////////
```

```
/**
 * La classe IndiceException.
 * - utilisée lors des débordements d'indice.
 ******************************/
class IndiceException {
public:
  IndiceException(int i, int max) : indice(i), taille(max) {}
  const int indice;
  const int taille;
};

//////////////////////////////////////////////////////////////////////
/**
 * La classe TailleException.
 * - utilisée lors d'une opération sur deux vecteurs de taille distincte.
 ******************************/
class TailleException {
public:
  TailleException(int t1, int t2) : taille1(t1), taille2(t2) {}
  const int taille1;
  const int taille2;

};

//////////////////////////////////////////////////////////////////////
/**
 * La classe Vecteur
 *
 ******************************/
template <class Elt> class Vecteur {

protected:
  Elt * _tab;                // le tableau des coordonnées
  int _dimension;               // la taille du tableau
  static Elt Trash;          // "poubelle" pour les débordements d'indice

public:
  Vecteur (int dim);            // on donne la taille du vecteur
  ~Vecteur();                   // pour récupérer la mémoire
  Vecteur(const Vecteur<Elt>&); // pour dupliquer correctement un Vecteur

  // ATTENTION: la fonction "friend" doit etre redeclaree en template,
  // et le parametre doit etre different (!)
  // affiche [ x1 ... xn ]
  template <class E> friend ostream& operator<<(ostream&, const Vecteur<E> &);

  Elt& operator[](int i) throw (IndiceException);
                                // accès à xi
  Vecteur<Elt> operator+(const Vecteur<Elt>& v) throw (TailleException);
                                // somme de deux vecteurs
  Elt operator*(const Vecteur<Elt>& v) throw (TailleException);
                                // produit scalaire
};
```

```
// Code inline obligatoire pour les templates
#include "Vecteur.C"

#endif

/*****************************************************************************/
```

9.2 Le fichier Vecteur.C

```
//////////////////////////////////////////////////////////////////////
/**
 * @file Vecteur.C
 * Corps des méthodes de la classe Vecteur.
 *
 * $Id: Vecteur.C 1659 2009-02-16 14:28:29Z phil $
 *
 * Version 1: seule difficulté = référence en retour de nieme()
 * Version 2: surcharge d'opérateurs (hors friends)
 * Version 3: surcharge des E/S avec les friends
 * Version 4: ajout du destructeur et du copy-constructor
 *      (le destructeur seul provoque une erreur d'exécution)
 * Version 5: passage par référence quand ça évite des duplications
 *      (3 Vecteur(int) + 4 Vecteur(Vecteur) -> 3 Vecteur(int)!
 * Version 6: gestion des débordements d'indice par une variable statique
 * Version 7: gestion des débordements / pbs de taille par exceptions
 *     (sauf main(): v8)
 * Version 9: type paramétré
 */
//////////////////////////////////////////////////////////////////////


//////////////////////////////////////////////////////////////////////
/**
 * Permet de retourner un objet licite en cas de débordement d'indice
 * dans [].
 ******************************/
//////////////////////////////////////////////////////////////////////
template <class Elt> Elt Vecteur<Elt> :: Trash;

//////////////////////////////////////////////////////////////////////
/**
 * Construit le vecteur avec des 0.
 * @param dim la taille du vecteur
 ******************************/
//////////////////////////////////////////////////////////////////////
template <class Elt> Vecteur<Elt> :: Vecteur (int dim) {
  _dimension = dim;
  _tab = new Elt[dim];
  for (int i = 0; i<dim; i++)
    _tab[i] = 0;
}

//////////////////////////////////////////////////////////////////////
/**
```

```
 * Construit le vecteur a partir d'un vecteur existant (recopie).
 * @param v le vecteur à copier.
 ******************************/
//////////////////////////////////////////////////////////////////////
template <class Elt> Vecteur<Elt> :: Vecteur (const Vecteur& v) {
  _dimension = v._dimension;
  _tab = new Elt[_dimension];
  for (int i = 0; i<_dimension; i++)
    _tab[i] = v._tab[i];
}
//////////////////////////////////////////////////////////////////////
/**
 * Récupère la mémoire allouée à la création du vecteur.
 * On remplit la zone de "0" pour mettre en évidence le passage dans le
 * destructeur.
 ******************************/
//////////////////////////////////////////////////////////////////////
template <class Elt> Vecteur<Elt> :: ~Vecteur() {
  for (int i = 0; i<_dimension; i++)
    _tab[i] = 0;
  delete[] _tab;
}

//////////////////////////////////////////////////////////////////////
/**
 * Affiche le vecteur sous la forme [ x1 ... xN ].
 ******************************/
//////////////////////////////////////////////////////////////////////
template <class E> ostream& operator<<(ostream& o, const Vecteur<E>& v) {
  o << "[ ";
  for (int i = 0; i < v._dimension; i++)
    o << v._tab[i] << ' ';
  return o << "]\n";
}

//////////////////////////////////////////////////////////////////////
/**
 * Renvoie la nieme coordonnée du vecteur.
 * Aucune gestion des dépassements
 * @param i l'indice de l'élément cherché (entre 1 et dimension).
 ******************************/
//////////////////////////////////////////////////////////////////////
template <class Elt> Elt& Vecteur<Elt> :: operator[](int i)
  throw (IndiceException) {
  if (i<1 || i>_dimension) throw (IndiceException(i,_dimension));
  return _tab[i-1];
}

//////////////////////////////////////////////////////////////////////
/**
 * Construit le vecteur avec des 0.
 * @param dim la taille du vecteur
 ******************************/
//////////////////////////////////////////////////////////////////////
```

```
template <class Elt> Vecteur<Elt>
Vecteur<Elt> :: operator+(const Vecteur<Elt>& v)
  throw (TailleException) {
  if (_dimension != v._dimension)
    throw (TailleException(_dimension, v._dimension));
  Vecteur res (_dimension);
  for (int i = 0; i<_dimension; i++)
    res._tab[i] = _tab[i] + v._tab[i];
  return res;
}

//////////////////////////////////////////////////////////////////////
/**
 * Construit le vecteur avec des 0.
 * @param dim la taille du vecteur
 ******************************/
//////////////////////////////////////////////////////////////////////
template <class Elt> Elt
Vecteur<Elt> :: operator*(const Vecteur<Elt>& v) throw (TailleException) {
  if (_dimension != v._dimension)
    throw (TailleException(_dimension, v._dimension));
  Elt res = 0;
  for (int i = 0; i < _dimension; i++)
    res += _tab[i] * v._tab[i];
  return res;
}

//////////////////////////////////////////////////////////////////////
```

9.3 Le fichier `tVecteur.C`

```
//////////////////////////////////////////////////////////////////////
/**
 * @file tVecteur.C
 * Test de la classe Vecteur.
 *
 * $Id: tVecteur.C 1659 2009-02-16 14:28:29Z phil $
 *
 * - création de deux vecteurs
 * - modification de coordonnées
 * - affichage d'un vecteur
 * - affichage de la somme et du produit scalaire des deux vecteurs
 *
 * Version 1: seule difficulté = référence en retour de nieme()
 * Version 2: surcharge d'opérateurs (hors friends)
 * Version 3: surcharge des E/S avec les friends (commencer par un opérateur
 *      hors de la classe, utilisant "[]", puis expliquer les "friends")
 * Version 4: ajout du destructeur et du copy-constructor
 *      (le destructeur seul provoque une erreur d'exécution)
 * Version 5: passage par référence quand ça évite des duplications
 *      (3 Vecteur(int) + 4 Vecteur(Vecteur) -> 3 Vecteur(int)!
 * Version 6: gestion des débordements d'indice par une variable statique
 * Version 7: gestion des débordements / pbs de taille par exceptions
```

```
 * Version 8: gestion des débordements dans main()
 * Version 9: type paramétré
 */
//////////////////////////////////////////////////////////////////////

#include <iostream>
using namespace std;
#include "Vecteur.h"

//////////////////////////////////////////////////////////////////////
/**
 * Test de la classe Vecteur.
 * - création de deux vecteurs
 * - modification de coordonnées et vérification du débordement
 * - affichage d'un vecteur
 * - affichage de la somme et du produit scalaire des deux vecteurs
 *
 ******************************/
int main() {
  cout << "sortie attendue:\n"
       << "[ 0 0 0 ]\n"
       << "[ 1 0 0 ]\n"
       << "[ 1 2 0 ]\n"
       << "IndiceException: tentative d'accès à l'indice 10 "
       << "sur un vecteur de taille 3\n"// exception gérée
       << "\nsortie effective:\n";
  Vecteur<double> v1(3), v2(3);
  cout << v1;
  try {
    v1[1] = 1;
    cout << v1;
    v2[2] = 2;
    cout << (v1 + v2);
    cout << v1[10] << endl;
    cout << (v1 * v2) << endl;
  }
  catch (TailleException te) {
    cerr << "Une TailleException\n";
    return 1;
  }
  catch(IndiceException ie) {
    cerr <<"IndiceException: tentative d'accès à l'indice "
         << ie.indice
         << " sur un vecteur de taille "
         << ie.taille << endl;
    return 2;
  }
  return 0;
}

//////////////////////////////////////////////////////////////////////
```