Écrire l'interface/la spécification et l'implémentation d'une classe **Point3D** qui permet d'instancier des points de 3 coordonnées de type **double**.
Voici les opérateurs à surdéfinir au minimum :

- l'addition + et la soustraction −, la multiplication * par un flottant (attention à bien gérer la **commutativité**), le flux de sortie <<, et le flux d'entrée >> ;

Écrire l'interface/la spécification et l'implémentation de la classe **TabDyn** (ci-dessous) qui permet d'instancier des tableaux dynamiques de **double**.
Voici les opérateurs à surdéfinir au minimum:

- l'affectation =, l'égalité ==, la différence !=, le flux de sortie <<, le flux d'entrée >>, les crochets [] ;

Bien réfléchir pour chaque opérateur s'il doit être une fonction membre ou une fonction amie.

Écrire un programme de test qui illustre le fonctionnement de chaque opérateur des classes **Point3D** et **TabDyn**.

```
// TabDyn.hpp
class TabDyn
{
   double* m_cases_1D ;  // tableau dynamique 1D ;
   int m_taille ;          // taille actuelle du tableau dynamique
   int m_capacite ;      // nombre de cases allouées/réservées à un instant t ; on a toujours
                         // m_capacite>=m_taille
  public :
     // Constructeurs
     TabDyn(int taille=3, double val=0) ;
     TabDyn(const TabDyn&) ;

     // Le destructeur
     ~TabDyn() ;

     // Accesseurs
     int getTaille() const { return m_taille; }
     int size() const { return m_taille; }

     // Modifieur
     bool resize(int nouvelle_taille, double val=0) ; // à utiliser pour redimensionner le tableau

     // Opérateurs
     TabDyn& operator = (const TabDyn&) ;
     double& operator[](int indice) ;
```

```
    const double& operator[](int indice) const ;
    friend ostream& operator<< (ostream&, const TabDyn&) ;
    ... // A vous de terminer la spécification
} ;
// TabDyn.cpp
bool TabDyn::resize(int nouvelle_taille, double val)
{
   if(nouvelle_taille==m_taille) return true ; // succès
   int min_tailles = (nouvelle_taille<m_taille)?nouvelle_taille:m_taille ;

   if( (nouvelle_taille>10) && (nouvelle_taille>m_capacite || nouvelle_taille < m_capacite/2) )
   { // cas où la réallocation est intéressante
      double* new_tab = new (std::nothrow) double[nouvelle_taille*2];
      if(new_tab==NULL)
      {
         cout << " La reallocation a echouee ! On conserve l'ancien tableau! " << endl ;
         return false ; // échec
      }

      // on recopie l'ancien tableau dans le nouveau
      for(int i=0; i<min_tailles; i++)
         new_tab[i] = m_cases_1D[i] ;

      delete [] m_cases_1D ;

      m_cases_1D = new_tab;

      m_capacite = nouvelle_taille*2 ;
   }

   // on complète les éventuelles cases restantes par la valeur
   for(int i=min_tailles; i<nouvelle_taille; i++) m_cases_1D[i] = val ;

   m_taille = nouvelle_taille ;

   return true ; // succès
}
...// A vous de terminer l'implémentation
```