Analyse et Programmation Orientées Objets / C++

T.A.D. Vecteur

Types abstraits de données

Modèle de conception pour la programmation impérative

- Privilégier une modélisation fonctionnelle

- Notions de services rendus favorisant la réutilisabilité
- Encapsulation et structuration des données internes
- Structuration des traitements, corrélée aux données
- Signatures formelles des services indépendante de l'implémentation

Choix de conception

Indépendants du langage de programmation

- Toutes les données structurées par un type hétérogène

- Instanciation dynamique du type
- Constructeur : moyen unique d'instanciation
- Définition d'accesseurs de consultation
- Définition d'accesseurs de modification
- Définition d'opérateurs génériques
- Définition de services génériques

Header du T.A.D. Vecteur (1)

```
# include <stdio.h>
# include <math.h>
# include <malloc.h>
const double EPSILON= 0.0001;
struct sVecteur {
double m_x;
double m_y;
};
typedef struct sVecteur Vecteur;
```

Header du T.A.D. Vecteur (2)

```
// Constructeurs et destructeur
//
Vecteur* construire ();
Vecteur* construire (double, double);
Vecteur* construire (const Vecteur&);
bool     detruire   (Vecteur*);
```

Header du T.A.D. Vecteur (3)

```
// Accesseurs de consultation
//
inline double  abscisse (const Vecteur& V) {
    return V.m_x;
}
inline double  ordonnee (const Vecteur& V) {
    return V.m_y;
}
double norme (const Vecteur&);
double norme (const Vecteur*);
```

Header du T.A.D. Vecteur (4)

```
// Opérateurs
//
bool egal (const Vecteur&, const Vecteur&);
bool diff (const Vecteur&, const Vecteur&);

Vecteur* add (const Vecteur&, const Vecteur&);
Vecteur* sub (const Vecteur&, const Vecteur&);
Vecteur* sub (const Vecteur&);
Vecteur* mul (const Vecteur&, double);
double   prodS (const Vecteur&, const Vecteur&);
```

Header du T.A.D. Vecteur (5)

```
// Services
//
bool  colineaire  (const Vecteur&, const Vecteur&);
bool  orthogonal (const Vecteur&, const Vecteur&);
bool  memeSens (const Vecteur&, const Vecteur&);

char*  toString     (const Vecteur&);
```

Constructeurs (1)

```
# include "Vecteur.h "

Vecteur* construire () {
Vecteur* pResultat;

    pResultat= (Vecteur*) malloc(sizeof(Vecteur));
    if (pResultat= =NULL) return NULL;
    pResultat->m_x= 0;
    pResultat->m_y= 0;

    return pResultat;
}
```

Constructeurs (2)

```
Vecteur* construire (double x, double y) {
Vecteur* pResultat;

    pResultat= (Vecteur*) malloc(sizeof(Vecteur));
    if (pResultat= =NULL) return NULL;
    pResultat->m_x= x;
    pResultat->m_y= y;

    return pResultat;
}
```

Constructeur de copie et destructeur

```
Vecteur* construire (const Vecteur& V) {
      return construire (abscisse(V), ordonnee(V));
 }

bool detruire (Vecteur* pV) {

      if (pV==NULL) return false;
      free (pV);
      return true;
 }
```

Exemple d'opérateurs (1)

```
# include "Vecteur.h "
Vecteur* add (const Vecteur& V1, const Vecteur& V2) {
double x, y;

    x= abscisse (V1) + abscisse (V2);
    y= ordonnee (V1) + ordonnee (V2);

    return construire (x, y);
}
```

Exemple d'opérateurs (2)

```
# include "Vecteur.h "
Vecteur* mul (const Vecteur& V, double k) {
double x, y;

    x= k*abscisse (V);
    y= k*ordonnee (V);

    return construire (x, y);
}
```

Exemple de services

```
 # include "Vecteur.h "
bool colineaire (const Vecteur& V1,  const Vecteur& V2) {
double d1= abscisse(V1)*ordonnee(V2);
double d2= ordonnee(V1)*abscisse(V2);

      return fabs(d1-d2) < EPSILON;
 }
```

Problème permanent des erreurs d'arrondis !