Université de Strasbourg
UFR de mathématique et d'informatique

L2 informatique
automne 2016

Structures de données et algorithmes

TP2 : Un peu d'algèbre linéaire

Echauffement

On considère la spécification des vecteurs 3D vue en cours et qu'on rappelle ici :

```
Spéc VECT-3D étend NAT + FLOAT
  Sorte Vect3d
  Opérations
      <_,_,_>       : Float Float Float -> Vect3d
      _:x  _:y _:z  : Vect3d -> Float              ** coordonnées
      _+_           : Vect3d Vect3d -> Vect3d
      _*_           : Float Vect3d -> Vect3d       ** produit par un nombre flottant
      _·_           : Vect3d Vect3d -> Float       ** produit scalaire
   Vars a b c r s t : Float
   Axiomes
   <a, b, c> :x = a .      <a, b, c> :y = b .        <a, b, c> :z = c .
   <a, b, c> + <r, s, t> = <a+r, b+s, c+t> .
   a*<r, s, t> = <a*r, a*s, a*t> .
   <a, b, c>.<r,s,t> = a*r + b*s + c*t .
fin spéc
```

1. Implantez cette spécification avec la structure de données suivante

```
// fichier vect3d.h
// ...
// protection contre la double inclusion (include guard)

typedef struct {
               float x; float y; float z; } vect3d;
```

Le produit vectoriel de deux vecteurs `<a,b,c>` et `<r,s,t>` en dimension 3 est un vecteur de coordonnées `bt-sc`, `rc-at` et `as-rb`.

2. Ajoutez des fonctions pour la saisie d'un vecteur 3D par l'utilisateur et son affichage.

3. Écrivez un programme de test qui vérifie notamment que le produit vectoriel w de deux vecteurs u et v est orthogonal à ces deux vecteurs. Compilez et testez votre implantation.

Vecteurs en dimension arbitraire donnée

On considère à nouveau la spécification vue en travaux dirigés dont la signature est rappelée ici :

```
 Spéc VECT[ N : Nat ] étend NAT + FLOAT
Sorte Vect
Opérations
          nul           :                 ->   Vect    ** vecteur nul
          _ :: _ <- _   : Vect Nat Float  ->   Vect    ** affectation
          _ [ _ ]       : Vect Nat        ->   Float   ** ième coordonnée
          _+_           : Vect Vect       ->   Vect    ** somme de deux vecteurs
          _*_           : Float Vect      ->   Vect    ** produit par un scalaire
          _·_           :  Vect Vect      ->   Float   ** produit scalaire
fin spéc
```

On veut faire une implantation efficace de cette spécification et on va utiliser des tableaux sans aucune encapsulation. On considère donc le type de données suivant :

```
// fichier vect.h
// ... include guard

#define N 5

typedef float fnum;     // qu'on pourra remplacer par d'autres types
                        // comme des doubles, des rationnels, des complexes ...

typedef fnum vect[N];
// ...
```

1. Avec ces définitions expliquez pourquoi une fonction de prototype

```
vect nul();
```

ne pas pas être programmée en C et d'une manière générale pourquoi une fonction ne peut pas retourner un tableau. Comment peut-on contourner le problème ?

Dans la suite, on choisit une programmation uniquement par effet de bord et on considère donc que le type concret pour `vect` est mutable. Ce qui nous permet de compléter le fichier header comme suit :

```
void v_nul(vect v);                     // rend v égal au vecteur nul
void v_modif_i(vect v, int i, fnum x);  // v :: i <- x
fnum v_ieme(vect v, int i);             // v[i] ... on pourra aussi s'autoriser v[i]
void v_somme(vect v, vect w, vect r);   // r = v + w
void v_prode(fnum a, vect v, vect r);   // r = a*v
fnum v_prod_scal(vect v, vect w);       // produit scalaire

void v_cp(vect u, vect r);              // r = u ... à quoi sert cette fonction ?
void v_print(vect v);                   // affichage
void v_read(vect v);                    // saisie
```

2. Complétez le fichier header avec les préconditions et programmez toutes ces fonctions. Quelle est l'utilité de la fonction `v_cp()` ?

3. Dans un fichier de nom `test_v.c`, faites un programme qui teste ces fonctions en vérifiant sur des exemples que les axiomes de la spécification vue en TD sont vérifiés. Compilez et exécutez.

Matrices

D'un point de vue structures de données, on considère des matrices `N` $\times$ `M` comme des vecteurs de vecteurs (mais évidemment, des fonctions comme le produit scalaire n'ont *a priori* pas de sens pour les matrices). On considère donc le fichier header :

```
// fichier mat.h

// ... include guard

#include "vect.h"
#define M 5

typedef vect mat[M];

void m_nulle(mat m);
void m_modif_ij(mat m, int i, int j, fnum x); // idem pour les deux fonctions
fnum m_ijeme(mat m, int i, int j);            // on pourra utiliser m[j][i]
                                              // ... attention à l'échange i et j
```

```
void v_prodm(mat m, vect v, vect r);          // r = m.v
void m_prodm(mat m1, mat m2, mat r);          // r = m1.m2

void m_cp(mat m, mat r);
void m_print(mat m);
void m_read(mat m);
```

Comme pour la section précédente, programmez et testez. En particulier, vérifiez les éventuels effets de bord avec l'appel `m_prodm(A,A,A)` où `A` est une matrice quelconque.