Initiation à la programmation en C

TP n°8

Antoine Miné

12 avril 2007

Site du cours: `http://www.di.ens.fr/~mine/enseignement/prog2006/`

Exercice 1. Vecteurs.
On propose le type structuré suivant pour représenter des vecteurs :

```
typedef struct {
  int     nb;
  double* elems;
} vecteur;
```

où `nb` est la dimension du vecteur et `elems` pointe vers un bloc de mémoire dynamique assez grand pour contenir `nb` éléments de type `double`.
Écrivez une fonction de prototype :

```
vecteur init(int nb);
```

qui crée un nouveau vecteur de taille `nb`, initialisé à 0. Écrivez également une fonction :

```
void detruit(vecteur v);
```

qui libère la mémoire dynamique allouée pour `v`. Écrivez des fonctions pour calculer la somme et le produit scalaire de deux vecteurs (après avoir vérifié que leurs dimensions sont compatibles) :

```
vecteur add(vecteur a, vecteur b);
double  mul(vecteur a, vecteur b);
```

Enfin, écrivez les fonctions suivantes pour saisir un vecteur au clavier et afficher un vecteur à l'écran :

```
vecteur lit();
void    affiche(vecteur a);
```

Que font les fragments de programme suivant :

1. `vecteur a ; affiche(a) ;`
2. `vecteur a = init(3) ; vecteur b = a ; b.elems[0] = 12 ; affiche(a) ;`
3. `vecteur a = init(3) ; vecteur b = a ; detruit(b) ; a.elems[0] = 12 ;`
4. `void f() { vecteur a = init(99) ; }`
5. `vecteur a = init(8) ; a = lit() ; detruit(a) ;`

Exercice 2. Affichage à l'envers caractère par caractère.
Écrivez un programme qui affiche le contenu d'un fichier à l'envers. Pour cela on doit :
– ouvrir (`fopen`) le fichier de nom donné en argument en ligne de commande (`argv[1]`),
– déterminer sa taille (`fseek` et `ftell`),
– allouer un tableau dynamique de la taille correspondante (`malloc`),
– charger le fichier dans le tableau (`fgetc`),
– afficher le tableau du dernier au premier caractère.

Exercice 3. Affichage à l'envers ligne par ligne.
Cette fois, on souhaite afficher le fichier ligne par ligne, en commençant par la dernière ligne, chaque ligne étant affichée à l'endroit.
Pour cela, on commencera par lire le fichier en entier ligne par ligne (`fgets`). Les lignes sont stockées dans un tableau `tab` contenant des entrées de type `char*` : chaque entrée pointe vers une copie (`strdup`) de la ligne. Comme on ne sait pas *a priori* le nombre de lignes du fichier, on part d'un tableau dynamique de petite taille qu'on étend (`realloc`) si nécessaire.

Exercice 4. Jeu de la vie (jeu de Conway).
Le *jeu de la vie* se joue une sur matrice rectangulaire dont chaque case peut être occupée par une cellule ou libre. À chaque coup d'horloge, on applique les règles suivantes :
– toute cellule ayant un ou zéro voisin meurt (sa case devient libre),
– toute cellule ayant quatre voisins ou plus meurt,
– dans toute case libre entourée de exactement trois voisins, une cellule naît,
– les autres cases restent inchangées.
Par "voisin", nous entendons toute cellule adjacente horizontalement, verticalement ou en diagonale. Chaque case a donc huit cases voisines.
On propose la structure suivante pour représenter un "état" :

```
typedef struct {
  int   lin, col;
  char* cases;
} monde;
```

où `lin` et `col` sont le nombre de lignes et de colonnes de la matrice et `cases` est un tableau alloué dynamiquement de `lin` × `col` octets. L'état de la case à la ligne `l`, colonne `c` est indiqué par `cases[ col*l + c ]` : 0 indique une case libre et 1 une case occupée par une cellule.
Proposez des fonctions :

```
void init( monde* m, int lin, int col );
void affiche( monde* m );
```

qui initialisent un nouvel état (vide) et affichent un état à l'écran. Notez que, contrairement au premier exercice, on a choisi de passer la structure par référence et non par valeur.
Proposez des fonctions pour lire ou modifier l'état d'une case :

```
char get( monde* m, int lin, int col );
void set( monde* m, int lin, int col, int c );
```

On supposera que lire en dehors de la matrice renvoie toujours 0 tandis qu'écrire en dehors de la matrice n'a aucun effet.
Proposez enfin une fonction :

```
void tick( monde* avant, monde* apres );
```

qui fait avancer la simulation d'un pas de temps. `tick` lit l'état courant du monde dans `avant` et stocke le nouvel état dans `apres`.
On pourra alors utiliser le bout de programme suivant pour lancer une simulation :

```
monde m1, m2;
init( &m1, 70, 24 );
init( &m2, 70, 24 );
/* initialiser m1 avec des valeurs intéressantes */
while (1) {
  affiche( &m1 );
  tick( &m1, &m2 );
  affiche( &m2 );
  tick( &m2, &m1 );
}
```

On propose d'initialiser le monde `m1` avec un des motifs intéressants suivants (où `-` indique une case vide et `O` une case occupée) :

-OOO OOO-	OOO O-- -O-	-O--O O---- O---O OOOO-
champignon	glisseur	vaisseau spatial