# Initiation à la programmation en C

# Correction du TP n°8

Antoine Miné

12 avril 2007

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

**Exercice 1**. Vecteurs.

```
vecteur init(int nb)
{
  int i;
  vecteur v = { nb, malloc( sizeof(double) * nb ) };
  assert( v.elems );
  for ( i=0; i<nb; i++ ) v.elems[i] = 0;
  return v;
}

void detruit(vecteur v)
{ free( v.elems ); }

vecteur add(vecteur a, vecteur b)
{
  int i;
  vecteur v;
  assert( a.nb == b.nb );
  v = init( a.nb );
  for ( i=0; i<a.nb; i++ )  v.elems[i] = a.elems[i] + b.elems[i];
  return v;
}

double mul(vecteur a, vecteur b)
{
  int i;
  double r = 0;
  assert( a.nb == b.nb );
  for ( i=0; i<a.nb; i++ )  r += a.elems[i] * b.elems[i];
  return r;
}

vecteur lit()
{
  int i;
  vecteur v;
  scanf( "%i", &v.nb );
  assert( v.nb>0 );
  v.elems = malloc( sizeof(double) * v.nb );
  assert( v.elems );
  for ( i=0; i<v.nb; i++ )  scanf( "%lf", &v.elems[i] );
  return v;
```

```
}

void affiche(vecteur a)
{
  int i;
  printf( "[ " );
  for ( i=0; i<a.nb; i++ )  printf( "%f ", a.elems[i] );
  printf( "]\n" );
}
```

Réponse aux questions :

1. `a.elems` n'étant pas alloué, `affiche(a)` provoquera une erreur à l'exécution.
2. `a.elems` et `b.elems` pointent sur le même bloc mémoire donc `b.elems[0] = 12` modifiera les deux vecteurs.
3. `detruit(b)` libère le bloc pointé par `a.elems` et `b.elems` donc `a.elems[0] = 12` provoquera une erreur.
4. Quand `f` retourne, la variable `a` est détruite mais le bloc pointé par `a.elems` n'est pas libéré. On n'a pas gardé d'autre pointeur vers ce bloc, il ne pourra donc jamais être libéré : on a une fuite de mémoire.
5. Le bloc alloué par `init(8)` n'est jamais libéré : on a une fuite de mémoire.

**Exercice 2**. Affichage à l'envers caractère par caractère.

```
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char* argv[])
{
  FILE* fichier;
  int i, taille;
  char* buf;

  /* ouverture */
  if ( argc < 2 ) { printf( "nom de fichier attendu\n" ); return 1; }
  fichier = fopen( argv[1], "r" );
  if ( ! fichier ) { printf( "ne peut ouvrir %s\n", argv[1] ); return 1; }

  /* saute à la fin du fichier pour déterminer sa taille */
  fseek( fichier, 0, SEEK_END );
  taille = ftell( fichier );

  /* retour au début du fichier */
  fseek( fichier, 0, SEEK_SET );

  /* lecture du fichier */
  buf = malloc( taille );
  if ( ! buf ) { printf( "échec de malloc\n" ); return 1; }
  for ( i=0; i<taille; i++ ) buf[i] = fgetc( fichier );

  /* écriture */
  for ( i=0; i<taille; i++ ) fputc( buf[ taille-i-1 ], stdout );
  printf("\n");

  return 0;
}
```

**Exercice 3**. Affichage à l'envers ligne par ligne.

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main(int argc, char* argv[])
{
  FILE* fichier;
  int i, taille = 0;
  char ** buf = NULL, ligne[256];

  /* ouverture */
  if ( argc < 2 ) { printf( "nom de fichier attendu\n" ); return 1; }
  fichier = fopen( argv[1], "r" );
  if ( ! fichier ) { printf( "ne peut ouvrir %s\n", argv[1] ); return 1; }

  /* lecture du fichier */
  while (1) {
    if (fgets( ligne, 256, fichier ) == NULL) break; /* fin de fichier */

    /* ajoute une entrée au tableau buf
       au premier appel (buf=NULL) c'est équivalent à un bête malloc
       lors des appels suivant, cela augmente la taille de buf
     */
    buf = realloc( buf, (taille+1) * sizeof(char*) );
    if (!buf) { printf( "échec de realloc\n"); return 1; }

    /* stoque un pointeur sur une copie de la ligne dans buf */
    buf[taille] = strdup( ligne );

    taille++;
  }

  /* écriture */
  for ( i=0; i<taille; i++ ) fputs( buf[ taille-i-1 ], stdout );

  return 0;
}
```

**Exercice 4**. Jeu de la vie (jeu de Conway).

```c
void init( monde* m, int lin, int col )
{
  int i;
  m->lin = lin; m->col = col;
  m->cases = malloc( lin*col );
  assert( m->cases );
  for ( i=0; i<lin*col; i++ ) m->cases[i] = 0;
}

char get( monde* m, int lin, int col )
{
  if (lin<0 || col<0 || lin>=m->lin || col>=m->col) return 0;
  return m->cases[ lin*m->col + col ];
}

void set( monde* m, int lin, int col, char c )
{
  if (lin<0 || col<0 || lin>=m->lin || col>=m->col) return;
  m->cases[ lin*m->col + col ] = c;
```

```
}

void affiche( monde* m )
{
  int i,j;
  for ( i=0; i<m->lin; i++ ) {
    for ( j=0; j<m->col; j++ ) {
      if ( get( m, i, j ) ) printf( "O" );
      else printf( " " );
    }
    printf("\n");
  }
}

void tick( monde* avant, monde* apres )
{
  int i,j;
  assert( avant->lin==apres->lin && avant->col==apres->col );
  for ( i=0; i<avant->lin; i++ )
    for ( j=0; j<avant->col; j++ ) {
      int nb =
        get( avant, i-1, j ) + get( avant, i-1, j-1 ) +
        get( avant, i+1, j ) + get( avant, i-1, j+1 ) +
        get( avant, i, j-1 ) + get( avant, i+1, j-1 ) +
        get( avant, i, j+1 ) + get( avant, i+1, j+1 );
      if ( get( avant, i, j) )
        if (nb<=1 || nb>=4) set( apres, i, j, 0 );
        else set( apres, i, j, 1 );
      else if (nb==3) set( apres, i, j, 1 );
        else set( apres, i, j, 0 );
    }
}
```