Programmation système et réseaux
Corrigé du TP 6: Pointeurs, allocation dynamique

Arnaud Giersch Guillaume Latu

Consigne. Pour les allocations mémoire, on utilisera la fonction xmalloc suivante :

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

void *xmalloc(size_t size)
{
    void *ret;
    if ((ret = malloc(size)) == NULL) {
        perror("xmalloc()");
        exit(EXIT_FAILURE);
    }
    return ret;
}
```

1. Dans cette question et la suivante nous allons manipuler des vecteurs et des matrices de réels double précision. Ci-après deux fonctions réalisant l'allocation d'une matrice de n lignes et k colonnes (c.-à-d. une matrice $n \times k$).

   ```
   double **creationMatrice1(int n, int k)
   {
       double **a, **pa;
       a = xmalloc(n * sizeof(double *));
       a[0] = xmalloc(n * k * sizeof(double));
       /* pour i = 1.. n-1, a[i] = a[i-1] + k */
       for(pa = a+1; pa < a+n; pa++)
           *pa = *(pa-1) + k;
       return a;
   }

   double **creationMatrice2(int n, int k)
   {
       double **a, **pa;
       a = xmalloc(n * sizeof(double *));
       for(pa = a; pa < a+n; pa++)
           *pa = xmalloc(k * sizeof(double));
       return a;
   }
   ```

   Quelle(s) différence(s) y a-t-il entre ces deux fonctions d'allocation ? Écrire les fonctions de désallocation correspondantes.

   Correction : *Dans la première fonction, la mémoire allouée pour chaque ligne de la matrice est contiguë. Dans la seconde fonction, les lignes étant allouée une à une, elles peuvent être dans différentes zones de la mémoire.*

   ```
   void liberationMatrice1(double **a, int n)
   {
       free(a[0]);
   ```

```
    free (a);
}

void liberationMatrice2 (double **a, int n)
{
    double **pa;
    for (pa = a; pa < a+n; pa++)
        free (*pa);
    free (a);
}
```

2. Écrire les fonctions ayant le prototype suivant :

```
/* affiche les éléments du vecteur "a" de dimension "k".
 */
void afficheVecteur (double *a, int k);

/* affiche les éléments de la matrice "a" de dimension "n * k".
 */
void afficheMatrice (double **a, int n, int k);

/* initialise les éléments de la matrice "a" de dimension "n * k";
   tous les éléments d'une même ligne seront égaux ; le réel choisi
   pour initialiser une ligne sera pris au hasard avec la fonction
   "drand48()".
*/
void initMatrice (double **a, int n, int k);

/* multiplie la matrice "a" de dimension "n * k" avec le vecteur "v"
   de dimension "k"; la fonction renverra en sortie le résultat de la
    multiplication .
*/
double * multiplieMatVect (double **a, int n, int k, double *v);
```

Vous utiliserez les fonctions précédentes dans le programme suivant que vous ferez fonctionner :

```
#define _XOPEN_SOURCE       /* pour srand48 */
#include <stdio.h>
#include <stdlib.h>

int main (void)
{
    double v[] = {0.2, 0.2, 0.2, 0.2, 0.2};
    int n = 6;
    int k = sizeof v / sizeof v[0];
    double **a = creationMatrice2 (n, k);
    double *res;

    /* initialisation du generateur pseudo-aleatoire "drand48()" */
    srand48 (2);
    initMatrice (a, n, k);
    printf ("La matrice : \n");
    afficheMatrice (a, n, k);
    res = multiplieMatVect (a, n, k, v);
    printf ("Le resultat : \n");
    afficheVecteur (res, n);

    liberationVecteur (res);
    liberationMatrice2 (a, n);
```

```
    return 0;
}
```

Correction :

```
double * creationVecteur (int k)
{
    return xmalloc (k * sizeof (double));
}

void liberationVecteur (double *v)
{
    free (v);
}

void afficheVecteur (double *a, int k)
{
    double *col;
    for (col = a; col < a + k; col++)
        printf ("%f\t", *col);
    printf ("\n");
}

void afficheMatrice (double **a, int n, int k)
{
    double **ligne;
    for (ligne = a; ligne < a + n; ligne++)
        afficheVecteur (* ligne, k);
}

void initMatrice (double **a, int n, int k)
{
    double *col;
    double **ligne;
    double val;
    for (ligne = a; ligne < a + n; ligne++) {
        val = drand48 ();
        for (col = * ligne; col < * ligne + k; col++)
            *col = val;
    }
}

double *multiplieMatVect (double **a, int n, int k, double *v)
{
    double *col;
    double **ligne;
    double *pres, * pv, * res;
    pres = res = creationVecteur (n);
    for (ligne = a; ligne < a + n; ligne++, pres++) {
        *pres = 0.0;
        for (col = * ligne, pv = v; col < * ligne + k; col++, pv++)
            *pres = * pres + *pv * *col;
    }
    return res;
}
```

3. **Pointeurs de fonctions.** La fonction qsort fait partie de la bibliothèque standard C. Elle réalise un tri sur un tableau d'éléments de type quelconque. Pour cela, on doit fournir à qsort, un pointeur sur une fonction permettant de comparer deux éléments entre eux (cf. manuel). Son prototype est le suivant

```
void qsort (void    *leTableau,   /* tableau à trier */
            size_t   nbElts,      /* nombre d'éléments dans le tableau */
            size_t   tailleElt,   /* taille d'un élément */
            int      (*compar)(const void *, const void *));
                                  /* fonction de comparaison */
```

Voici un exemple d'utilisation qui trie une liste d'entiers :

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

int compEntiers (const void *g, const void *d)
{
    int gi = *(int *)g;
    int di = *(int *)d;
    if (gi < di)  return -1;
    if (gi == di) return 0;
    return 1;
}

int main (void)
{
    int tableau [] = {0, 4, 2, 9, -1, 8};
    int taille = sizeof tableau / sizeof tableau[0];
    int i = 0;
    qsort (tableau, taille, sizeof tableau[0], &compEntiers);
    for (i = 0; i < taille; i++)
        printf ("%d\n", tableau[i]);
    return 0;
}
```

Réalisez deux programmes utilisant la fonction qsort :
– l'un triera un tableau de réels ;
– l'autre triera un tableau de chaînes de caractères.

Correction :

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

#define TAILLE(t) ((sizeof t) / (sizeof (t)[0]))

int cmp_double (const void *g, const void *d)
{
    double gg = *(double *)g;
    double dd = *(double *)d;
    if (gg < dd) return -1;
    if (gg > dd) return 1;
    return 0;
}

int cmp_string (const void *g, const void *d)
{
    return strcmp (*(char **)g, *(char **)d);
}

int main (void)
{
    double a_double [] = {5.3, 1.5, 9.6, -3.14, 7.7, -12.9};
```

```
    int s_double = TAILLE (a_double);
    char * a_string [] = { "il", "etait", "une", "fois", "qsort"};
    int s_string = TAILLE (a_string);
    int i;

    qsort (a_double, s_double, sizeof (double), cmp_double);
    qsort ( a_string , s_string , sizeof (char *), cmp_string);

    for (i = 0; i < s_double; i++)
        printf ("%g%s", a_double[i], (i < s_double - 1)? " ; ": "\n");
    for (i = 0; i < s_string ; i++)
        printf ("\"%s\"%s", a_string[i], (i < s_string - 1)? " ; ": "\n");

    return 0;
}
```

4. On souhaite proposer à des utilisateurs la fonctionnalité suivante : manipuler un vecteur de réels dont la taille est variable, sans qu'ils aient besoin de réaliser de malloc. Pour cela, vous allez coder deux fonctions : getElement et setElement qui permettent de lire ou d'écrire la valeur du i^e élément du vecteur.

   La fonction setElement réalisera automatiquement un realloc si l'utilisateur essaye d'affecter une valeur au-delà de la taille actuelle du vecteur. La taille courante du vecteur sera stockée dans une variable globale de type size_t. Voici le prototype des fonctions que vous devez coder :

```
/* accède en lecture au i ème élément du vecteur dont le type est ( float *) */
float getElement ( size_t i );

/* accède en écriture au i-ème élément du vecteur :
   - si ( i > taille ) un realloc est réalisé ,
   - si le realloc echoue , la fonction retourne 0,
   - si l' affectation du i ème élément réussie , retourne 1
*/
int setElement ( size_t i , float val );

/* libère l'espace mémoire alloué */
void freeElements (void);
```

   Vous essaierez votre couple de fonctions avec le programme suivant :

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

int main(void)
{
    size_t n, i;
    for (n = 1 ; n <= 1000000000; n *= 10 ) {
        printf ("n = %lu\n", (unsigned long)n);
        if (! setElement (n , 3.0 * n)) {
            printf ("! setElement\n");
            break;
        }
        printf ("valeurs du vecteur : ");
        for (i = 1; i <= n ; i *= 10 )
            printf ("%g%s", getElement (i), ((i==n)?"\n":", "));
    }
    freeElements ();
    return 0;
}
```

   Correction :

```
static size_t taille = 0;
static float *tampon = NULL;

float getElement ( size_t i )
{
    return ( i < taille )? tampon[i ]: 0.0;
}

int setElement ( size_t i , float val )
{
    if ( i >= taille ) {
        float *tampon_nouv;
        size_t taille_nouv ;
        size_t taille_nouv_reelle ;

        taille_nouv = ( taille == 0)? 1: taille ;
        taille_nouv_reelle = taille_nouv * sizeof *tampon;
        while ( taille_nouv <= i ) {
            size_t tmp = 2 * taille_nouv_reelle ;
            if ( tmp > taille_nouv_reelle ) {
                taille_nouv = 2 * taille_nouv ;
                taille_nouv_reelle = tmp;
            } else {
                /* overflow */
                return 0;
            }
        }

        fprintf ( stderr , "# taille : %lu -> %lu (%lu)\n",
                  (unsigned long) taille , ( unsigned long) taille_nouv ,
                  (unsigned long) taille_nouv_reelle );
        tampon_nouv = realloc (tampon, taille_nouv_reelle );
        fprintf ( stderr , "# realloc : %p -> %p\n",
                  (void *)tampon, (void *)tampon_nouv);
        if ( tampon_nouv == NULL) {
            /* realloc a échoué */
            return 0;
        }
        tampon = tampon_nouv;
        taille = taille_nouv ;
    }
    tampon[i ] = val ;
    return 1;
}

void freeElements (void)
{
    free (tampon);
    tampon = NULL;
    taille = 0;
}
```