Prog. Or. Système - Correction série 05 : Pointeurs 1

Exercice 1 : Passage par référence
(fichier src/swap.c)

```
void echange(int* a, int* b)
{
  /* on sauvegarde la valeur pointée par a pour ne pas la perdre */
  int const copie = *a;

  /* la valeur pointée par a prend la valeur pointée par b       */
  *a = *b;

  /* la valeur pointée par b prend la valeur de copie            */
  *b = copie;
}
```

Pour plus de détails, voir le cours 8 (à venir).

Exercice 2 : Pointeurs
(fichier src/ptr.c)

```
#include <stdio.h>

int demander_nombre(int a, int b)
{
   int res;

   do {
       printf("Entrez un nombre entier compris entre %d et %d :",
              a, b);
       scanf("%d", &res);
    } while ((res < a) || (res > b));
    return res;
}

int main(void) {
  double valeur1 = 3.14159265358;
  double valeur2 = 1.42857142857;
  double valeur3 = -12.344556667;
  double* choix = NULL;    /* On initialise le pointeur sur NULLe part */

  switch (demander_nombre(1,3)) {
    case 1: choix = &valeur1; /* la valeur de choix est l'adresse de valeur1 */
```

```
      break;
    case 2: choix = &valeur2; /* la valeur de choix est l'adresse de valeur2 */
      break;
    case 3: choix = &valeur3; /* la valeur de choix est l'adresse de valeur3 */
      break;
  }

  printf("Vous avez choisi %lf\n",
         *choix /* on affiche la valeur pointée par choix */
        );
  return 0;
}
```

Exercice 3 : Explorer la mémoire (niveau 1)
Je présente ici deux variantes de l'affichage des bits d'un octet. Il y en a bien sûr beaucoup d'autres ! C'est surtout pour illustrer du point de vue pratique des opérations peu commentées en cours (opérations binaires sur la mémoire).

Une des solutions présente aussi l'opérateur ternaire `?:` : « `A ? B : C` » est similaire à « `if (A) { B } else { C }` ».

(fichier src/memory_view.c)

```
// C99
#include <stdio.h>

typedef unsigned char octet;

// =====================================================================
// version 1

static inline void affiche_bit(const octet c,
                               const octet position_pattern)
{
  putchar(c & position_pattern ? '1' : '0');
}

void affiche_binaire(const octet c) {
  for(octet mask = 0x80; mask; mask >>= 1)
    affiche_bit(c, mask);
}

/* version 2 : moins bonne que ci-dessus :
 * affiche les bits « à l'envers » et n'affiche
 * pas les 0 de poids fort.
 */
```

```
void affiche_binaire_2(octet c) {
  do {
    if (c & 1) putchar('1');
    else       putchar('0');
    c >>= 1; // ou c /= 2;
  } while (c);
}

// ========================================================================
void affiche(size_t i, octet c) {
  printf("%02zu : ", i);
  affiche_binaire(c);
  printf(" %3u ", (unsigned int) c);
  if ((c >= 32) && (c <= 126)) {
    printf("'%c'", c);
  }
  putchar('\n');
}

// ========================================================================
void dump_mem(const octet* ptr, size_t length)
{
  /* solution simple qui pourra être améliorée
   * lorsque nous aurons vu l'arithmétique des pointeurs
   */
  printf("A partir de %p :\n", ptr);
  for (size_t i = 0; i < length; ++i) {
    affiche(i, ptr[i]);
  }
}

// ========================================================================
int main(void)
{
  int a = 64 + 16;
  int b = -a;
  double x = 0.5;
  double y = 0.1;

  dump_mem( (octet*) &a, sizeof(a) );
  dump_mem( (octet*) &b, sizeof(b) );
  dump_mem( (octet*) &x, sizeof(x) );
  dump_mem( (octet*) &y, sizeof(y) );

  return 0;
}
```

Dernière mise à jour : Dernière mise à jour le 8 mars 2016

Last modified: Tue Mar 8, 2016