Corrigé série 9.1 : Pointeurs

Exercice 1

Quelles sont les valeurs finales des variables a, b et c après exécution des deux séquences de programmes suivantes :

a)

```
int a, b, c, *ptr;
ptr = &a;
*ptr = 4;
b = a + 5;
ptr = &b;
c = *ptr;
```

a=4, b=9, c=9

b)

```
char a, b, c, *ptr;
a = b = 3;
ptr = &a;
c = *ptr += 2;
ptr = &c;
++(*ptr);
```

a=5,b=3,c=6

Exercice 2

Si champ représente un tableau et si ptr est un pointeur du même type, pourquoi les instructions suivantes sont-elles équivalentes ?

```
ptr = champ;
```

```
ptr = &champ[0];
```

- Le mot *champ* représente l'adresse du tableau. C'est aussi l'adresse du premier élément.
- Le premier élément du tableau est *champ[0]*
- L'adresse du premier élément du tableau est *&champ[0]* ou *champ.* ptr = champ indique que *ptr* pointe sur le premier élément *champ[0].*

Exercice 3

Trouver les erreurs ou les incohérences figurant dans les deux extraits de programmes suivant :

a)

```
char x[] = "abcd";
char y[] = "efgh";
char *ptr = x;
x = y;
y = ptr;
```

Programme a)
On ne peut pas affecter une nouvelle adresse à une variable désignant un tableau. Les instructions **x = y** et **y = ptr** contreviennent à ce principe. Seuls les pointeurs peuvent être réaffectés.

b)

```
int nombre1, nombre2;
int *ptr = &nombre1;
ptr = 25;
nombre2 = ptr + 6;
ptr = &nombre2;
```

Programme b)
Syntaxiquement ce programme est correct, par contre il va « planter » lors de l'execution car on a spécifié une adresse 25 qui ne correspond à rien
Une correction possible :
Dans les lignes **ptr = 25** et **nombre2 = ptr + 6**, l'opérateur * a été oublié.

Source : http://www.ltam.lu/Tutoriel_Ansi_C/

Exercice 4
Un programme C contient les instructions suivantes :

```
char  *pc,          c[10];
int   *pi1, *pi2, i1[10], i2 = 7;
float *pf1, *pf2, f1[10], f2 = 3.14;
```

On supposera que **c** se trouve à l'adresse 900, **i1** à l'adresse 1000, **i2** à l'adresse 1100, **f1** à l'adresse 1200 et **f2** à l'adresse 1300.
Pour **chacune** des 11 instructions ci-dessus, donner la valeur assignée au membre de gauche. Expliquer chacune de vos réponses.

```
  pc = c;           // Adr. de la variable c => 900
  printf("pc : %u\n", pc);

  pi1 = i1;         // Adr. de la variable i1 => 1000
  printf("pi1 : %u\n", pi1);

  pi2 = &i2;        // Adr.e de la variable i2 => 1100
  printf("pi2 : %u\n", pi2);

  pf1 = f1;         // Adr. de la variable f1 => 1200
  printf("pf1 : %u\n", pf1);

  pf2 = &f2;        // Adr. de la variable f2 => 1300
  printf("pf2 : %u\n", pf2);

  pc += 6;          // pc prend la valeur pc + 6 = 906
                    // (1 octet par caractère)
  printf("pc : %u\n", pc);

  pi1 += 6;         // pi1 prend la valeur pi1 + 24 = 1024
                    // (6 * 4 octets par valeur int)
  printf("pi1 : %u\n", pi1);

  pf1 += 6;         // pf1 prend la valeur pf1 + 24 = 1224
                    // (6 * 4 octets par valeur float)
  printf("pf1 : %u\n", pf1);

  *pf2 += 5;        // La valeur pointée par pf2 (f2) prend
                    // la valeur f2 + 5 = 8.14
  printf("*pf2 : %f\n", *pf2);

  *pi2 -= 2;        // La valeur pointée par pi2 (i2) prend
                    // la valeur i2 - 2 = 5
printf("*pi2 : %d\n", *pi2);

  pi1 -= *pi2;      // pi1 prend la valeur pi1 - 20 = 1004
                    // (*pi2 vaut 5 et on a 4 octets par int)
  printf("pi1 : %u\n", pi1);
```

Source : http://www.ltam.lu/Tutoriel_Ansi_C/

Exercice 5

Soit le programme ci-dessous. Supposons que la variable **a** se trouve à l'adresse `0x0012ff88`, la variable **ptr1** à l'adresse `0x0012ff84` et la variable **ptr2** à l'adresse `0x0012ff80`.

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

int main (void)
{
    int a = 1 ;

    int*  ptr1 ;
    int** ptr2 ;

    ptr1 = &a ;
    ptr2 = &ptr1 ;

    printf("&a    : %d\n", &a);
    printf("a     : %d\n", a);

    printf("&ptr1 : %d\n", &ptr1);
    printf("ptr1  : %d\n", ptr1);
    printf("*ptr1 : %d\n", *ptr1);

    printf("&ptr2 : %d\n", &ptr2);
    printf("ptr2  : %d\n", ptr2);
    printf("*ptr2 : %d\n", *ptr2);
    printf("**ptr2: %d\n", **ptr2);
  system("pause");

  return 0;
}
```

Quelle sera la sortie de ce programme ?

```
&a    : 0x0012ff88 // Adresse de a
a     : 1          // Valeur de a
&ptr1 : 0x0012ff84 // Adresse de ptr1
ptr1  : 0x0012ff88 // Valeur de ptr1, adresse de a
*ptr1 : 1          // Valeur pointée par ptr1, valeur de a
&ptr2 : 0x0012ff80 // Adresse de ptr2
ptr2  : 0x0012ff84 // Valeur de ptr2, adresse de ptr1
*ptr2 : 0x0012ff88 // Valeur pointée par ptr2, valeur de ptr1,
                   // adresse de a
**ptr2: 1          // Valeur de a
```

Source : http://www.ltam.lu/Tutoriel_Ansi_C/

Exercice 6

Ecrire un programme qui lit deux tableaux a et b et leurs tailles n et m au clavier et qui ajoute les éléments de **b** à la fin de **a**. Utiliser le formalisme pointeur à chaque fois que cela est possible. Les tableaux seront déclarés avec une taille permettant d'introduire jusqu'à 50 valeurs dans a et b, puis de réaliser l'opération demandée.

```
#include <stdio.h>
int main()
{
 /* Déclarations */
 int a[100], b[50];                    /* tableaux */
 int n, m;                             /* dimensions des tableaux
*/
 int i;                                /* indice courant */

 /* Saisie des données */
 printf("Dimension du tableau A (max.50) : ");
 scanf("%d", &n );
 for (i=0; i<n; i++){
   printf("Elément %d : ", i);
   scanf("%d", a+i);
 }
 printf("Dimension du tableau b (max.50) : ");
 scanf("%d", &m );
 for (i=0; i<m; i++){
   printf("Elément %d : ", i);
   scanf("%d", b+i);
 }

 /* Affichage des tableaux */
 printf("Tableau donné a :\n");
 for (i=0; i<n; i++)
      printf("%d ", *(a+i));
 printf("\n");
 printf("Tableau donné b :\n");
 for (i=0; i<m; i++)
      printf("%d ", *(b+i));
 printf("\n");

 /* Copie de b à la fin de a */
 for (i=0; i<m; i++)
       *(a+n+i) = *(b+i);
 /* Nouvelle dimension de a */
 n += m;

  /* Affichage du résultat */
 printf("Tableau résultat a :\n");
 for (i=0; i<n; i++)
       printf("%d ", *(a+i));
 printf("\n");

 return 0;
}
```

Source : http://www.ltam.lu/Tutoriel_Ansi_C/

Exercice 7

ptr est un pointeur sur une variable de type int. Initialement ptr pointe sur la base d'un tableau **int**. Quel élément du tableau est référencé par chacune des trois expressions suivantes : le 2ème élément du tableau dans les 3 cas.

```
ptr[1]          *(ptr+1)   *(++ptr)
```

Laquelle de ces trois opérations modifie le pointeur ? la 3ème

Exercice 8

Trouver l'erreur qui se cache dans la séquence de programme suivante :

```
int tablo[20], *ptr = tablo, n;
for (n = 0; n < 20; --n) *(ptr++) = 2;
```

La boucle devrait être exécutée pour **n** variant de **0 à 19**. Mais la variable **n** est **décrémentée** à chaque itération. Le décompte se fait donc en décroissant et la boucle tournera plus longtemps que prévu (n devient de plus en plus négatif).
L'instruction ***(ptr++) = 2;** est particulièrement **dangereuse**. Elle ne se contente pas d'affecter une valeur aux 20 éléments du tableau, mais elle déclenche des écritures dans la mémoire qui se trouve au-delà. Si des données sensibles se trouvent stockées à cet endroit, la conséquence peut être un arrêt du système.

Source : http://www.ltam.lu/Tutoriel_Ansi_C/

Exercice 9
Soit le programme suivant :

```
#include <stdio.h>

int lectureTaille( float * );

int main(void)
{
   float tableau[ 20 ];

   printf("La taille du tableau en octets est %d"
          "\nLa taille renvoyée par lectureTaille est %d\n",
          sizeof( tableau ),
          lectureTaille( tableau ));

   return 0 ;
}

int lectureTaille( float *ptr )
{
   return sizeof( ptr );
}
```

Ecrire la sortie de ce programme dans le rectangle ci-dessous :

```
La taille du tableau en octets est 80
La taille renvoyée par lectureTaille est 4
```

tableau est un nom qui identifie une zone mémoire fixe (adresse constante) allouée automatiquement pour 20x4 bytes, l'opérateur sizeof renvoie donc 80 dans le premier cas.

La fonction **lectureTaille** renvoie la taille de son paramètre ptr: une variable locale de type pointeur contenant une adresse (32 bits ici) --> `sizeof (ptr)` vaut 4 bytes.

Source : http://www.ltam.lu/Tutoriel_Ansi_C/

Exercice 10

Ecrire une fonction `void Echange(...)` qui permette l'échange du contenu de 2 variables déclarées dans le programme principal. Faire un programme pour la tester

```
#include <stdio.h>

void echange( int *a, int *b );

int main(void)
{
  int a = 5, b = 10 ;

  printf("Before: a: %d, b: %d\n", a, b);
  echange(&a,&b) ;
  printf("After:  a: %d, b: %d\n", a, b);

  return 0 ;
}

void echange( int *a, int *b )
{
  int tmp;

  tmp=*a;
  *a=*b;
  *b=tmp;
}
```

Source : http://www.ltam.lu/Tutoriel_Ansi_C/

Exercice 11

Soit le programme suivant :

```
int main()
{
    int A = 1;
    int B = 2;
    int C = 3;
    int *P1, *P2;

    P1 = &A;
    P2 = &C;
    *P1 = (*P2)++;
    P1 = P2;
    P2 = &B;
    *P1 -= *P2;
    ++(*P2);
    *P1 *= *P2;
    A = ++(*P2) * *P1;
    P1 = &A;
    *P2 = *P1 /= *P2;
    return 0;
}
```

Compléter le tableau ci-dessous pour chaque instruction du programme :

	A	**B**	**C**	**P1**	**P2**
Initialisation	1	2	3	?	?
P1 = &A	1	2	3	&A	?
P2 = &C	1	2	3	&A	**&C**
*P1 = (*P2)++	**3**	2	**4**	&A	&C
P1 = P2	3	2	4	**&C**	&C
P2 = &B	3	2	4	&C	**&B**
*P1 -= *P2	3	2	**2**	&C	&B
++(*P2)	3	**3**	2	&C	&B
*P1 *= *P2	3	3	**6**	&C	&B
A = ++(*P2) * *P1	**24**	**4**	6	&C	&B
P1 = &A	24	4	6	**&A**	&B
*P2 = *P1 /= *P2	**6**	**6**	6	&A	&B

Source : http://www.ltam.lu/Tutoriel_Ansi_C/

Exercice 12

Soit **p** un pointeur qui 'pointe' sur un tableau **a** :

```
int a[] = {12, 23, 34, 45, 56, 67, 78, 89, 90};
int *p;
p = a;
```

Quelles valeurs ou adresses fournissent ces expressions:

a)	`*p + 2`	**14**
b)	`*(p+2)`	**34**
c)	`&a[4] - 3`	&a[1]
d)	`a + 3`	&a[3]
e)	`&a[7] - p`	7 *sizeof(int)
f)	`p + (*p - 10)`	&a[2]
g)	`*(p + *(p+8)-a[7])`	23

Pour une valeur, on écrira par exemple: 90
Pour une adresse, on écrira par exemple l'adresse de la composante: &A[8]

Source : http://www.ltam.lu/Tutoriel_Ansi_C/