Exercices pointeurs

Exercice 1

Vérifiez que vous avez compris le système d'adressage et de déréférencement en complétant le code suivant

```
int value = 42;
int * ptr = ...;                /* TODO : pointe sur value */
int ** pptr = ...;              /* TODO : pointe sur ptr */
int *** ppptr = ...;            /* TODO : pointe sur pptr */
printf("valeur via value : %d\n", valeur);
printf("valeur via ptr : %d\n", ...);
printf("valeur via pptr : %d\n", ...);
printf("valeur via ppptr : %d\n", ...);
```

Chaque printf devrait afficher la valeur 42

```
correction
int value = 42;
int * ptr = &value;             /* TODO : pointe sur value */
int ** pptr = &ptr;             /* TODO : pointe sur ptr */
int *** ppptr = &pptr;                /* TODO : pointe sur pptr */
printf("valeur via value : %d\n", valeur);
printf("valeur via ptr : %d\n", *ptr);
printf("valeur via pptr : %d\n", **pptr);
printf("valeur via ppptr : %d\n", ***ppptr);
```

Exercice 2

Soient les instructions suivantes

```
int x = 5;
int* pX = &x;
float y = 6.8;
float* pY = &y;
```

Ecrire les instructions permettant d'afficher

- les valeurs des variables x et y
- les adresses des variables x et y

- les valeurs des variables x et y en utilisant les pointeurs pX et pY
- les adresses des variables x et y en utilisant les pointeurs pX et pY

correction :

```
printf("valeur de x : %d\n", x);
printf("valeur de y : %g\n", y);
printf("adresse de x : %p\n", &x);
printf("adresse de y : %p\n", &y);
printf("valeur de x en utilisant pX : %d\n", *pX);
printf("valeur de y en utilisant pY : %g\n", *pY);
printf("adresse de x en utilisant pX : %p\n", pX);
printf("adresse de y en utilisant pY : %p\n", pY);
```

Exercice 3

Les deux séries d'instructions suivantes sont-elles correctes ? Pourquoi ?

```
int *ptr, nb=1;
*ptr=22;
ptr=&nb;
**ptr = 100;
```

```
int n1,n2;
int *ptr=&n1;
ptr=10;
n2=ptr+2;
ptr=&n2;
ptr = &2541;
```

correction bloc 1
**ptr = 100 interdit
car *ptr c'est 22, qui est un int, pas une adresse

correction bloc 2
ptr = 10 interdit
n2 = ptr+2 interdit car on affecterait une adresse dans un int
ptr = &2541 interdit car une valeur seule n'a pas d'adresse

Exercice 4

Soit le programme C suivant :

```c
#include <stdio.h>
int main(int argc, char ** argv) {
   int *aPtr;
   int a = 7;
   aPtr = &a;
   printf("\n%? %?", sizeof (a), sizeof (aPtr));
   printf("\n%?", &a);
   printf("\n%?", aPtr);
   printf("\n%?", &aPtr);
   printf("\n%?", a);
   printf("\n%?", *aPtr);
   printf("\n%?", *&aPtr);
   printf("\n%?", &*aPtr);
   printf("\n%?", *&a);
   printf("\n%?", &*a);

   return 0;
}
```

- Quels sont les codes formats à utiliser ?
- Y a-t-il une erreur ?
- Le taper ensuite et vérifier les résultats.

correction

```c
#include <stdio.h>
int main(int argc, char ** argv) {
  int *aPtr;
  int a = 7;
  aPtr = &a;
  printf("\n%lu %lu", sizeof (a), sizeof (aPtr));
  printf("\n%p", &a);
  printf("\n%p", aPtr);
  printf("\n%p", &aPtr);
  printf("\n%d", a);
  printf("\n%d", *aPtr);
  printf("\n%p", *&aPtr);
  printf("\n%p", &*aPtr);
  printf("\n%d", *&a);
  // printf("\n%?", &*a);// erreur

  return 0;
```

```
}
```

Exercice 5

Soient les déclarations suivantes :

```
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
b) *(P+2)
c) &P+1
d) &A[4]-3
e) A+3
f) &A[7]-P
g) P+(*P-10)
h) *(P+*(P+8)-A[7])

correction :

a) 14
b) 34
c) int** c'est l'adresse qui suit l'adresse du pointeur p
d) adresse de la deuxième case du tableau
e) adresse de la 4e case du tableau
f) 7
g) adresse de la 3e case du tableau
h) 23

Exercice 6

construisez une fonction

```
void addToVal(int val);
```

qui prend en paramètre un entier et ajoute 1 à val

construisez une autre fonction

```
void addToValInTab(int tab[], int size);
```

qui ajoute 1 à toutes les valeurs dans les tableau tab

puis créez un programme qui appelle les 2 fonctions ainsi, puis affiche les valeurs des variables après l'appel des fonctions

Les valeurs affectées plus tôt ont-elles évolué? Pourquoi ?

```c
#include <stdio.h>
int main() {
   int a = 1;
   int t[] = {1, 2, 3, 4};
   addToVal(a);
   addToValInTab(t, 4);
   printf("a: %d\n", a);
   for (int i = 0; i < 4; i++)
   {
       printf("t[%d] : %d\n", i, t[i]);
   }
   return 0;
}
```

correction

```c
void addToVal(int a) {
   a++;
}

void addToValInTab(int tab[],int size) {
   for (int i = 0; i < size; i++)
   {
       tab[i]++;
   }
}
```

Exercice 7

Soit le programme suivant

```c
#include <stdio.h>

int main() {
```

```
    int a = 1, b = 2;
    swap(&a, &b);
    printf("a : %d\n", a);
    printf("b : %d\n", b);
    return 0;
}
```

construire la fonction swap afin qu'elle inverse les valeurs dans a et b
l'objectif est que le programme affiche :
a : 2
b : 1

correction

```
void swap(int* pA, int* pB) {
    int temp = *pA;
    *pA = *pB;
    *pB = temp;
}
```

Exercice 8

soit l'extrait de code suivant

```
int t[] = {1, 2, 3, 4};
swapCells(t, 1, 3);
```

coder la fonction swapCells qui inverse le contenu des 2 cases du tableau aux 2 indices renseignés comme 2e et 3e paramètres

correction

```
void swapCells(int tab[], int i1, int i2) {
    int temp;
    temp = tab[i1];
    tab[i1] = tab[i2];
    tab[i2] = temp;
}
```

Exercice 9

Recopiez la fonction fun et la manière dont elle est appelée dans le main.

```c
int* fun() {
   int a = 5;
   return &a;
}

int main(int argc, char ** argv) {
   int* x = fun();
   printf("%d\n", *x);
   return 0;
}
```

qu'affiche votre programme ?
Pouvez-vous expliquer votre affichage ?

Si votre programme affiche la valeur 5, compléter votre code ainsi :

```c
int* fun() {
   int a = 5;
   return &a;
}

int trash() {
   int b = 999;
   return b;
}


int main(int argc, char ** argv) {
   int* x = fun();
   trash();
   printf("%d\n", *x);
   return 0;
}
```

Qu'affiche votre programme maintenant ? Pourquoi ? Comment corrigeriez vous votre code pour avoir un comportement prédictible?

correction : fun implique un comportement imprévisible :
La variable a est détruite et l'emplacement mémoire libérée à la fin de la fonction. l'adresse retournée existe, mais il n'y a plus de protection sur la valeur à l'intérieur.
Si la bonne valeur est affichée avec le premier code, c'est une coïncidence (cela veut dire que rien n'a été réécrit entre temps sur l'emplacement mémoire)

Avec le deuxième code, on constate que l'affichage n'est pas celui qu'on espère, car on force une réécriture dans la pile de mémoire

Pour corriger cela, il faudrait modifier la fonction fun pour que la variable qu'on affecte lui soit passée en paramètre (en pointeur) afin que l'emplacement mémoire ne soit pas libéré à la fin