TP3: Corrigé

MP2I Lycée Pierre de Fermat
guillaume.rousseau@ens-lyon.fr

Exercice 1

Q1. A l'exécution, le programme affiche deux adresses mémoires (qui ne sont pas les mêmes à chaque exécution). Ces deux adresses, affichées en hexadécimal, diffèrent systématiquement de deux octets, par exemple :

```
L'adresse de x est 0x7ffc13963228,
celle de y est     0x7ffc1396322c
```

C'est logique étant donné que le type int prend 32 bits, soit 4 octets : en mémoire, `x` prend les octets `0x7ffc13963228` à `0x7ffc1396322b`, et `y` est stockée juste après, dans les octets `0x7ffc1396322c` à `0x7ffc1396322f`.

Q2. En remplaçant le type int par long int, on voit que les variables prennent maintenant 8 octets (car les adresses diffèrent de 8), ce qui est logique car nous avons dit que le type long int prend 64 bits.

Exercice 2

Q1. Oui.

Q2. Même en modifiant `x`, la variable `y` n'est pas affectée. C'est logique : les deux variables sont indépendantes, elles correspondent à des zones mémoires différentes, donc en modifier une n'a aucun effet sur l'autre. L'instruction `int y = *px;` ne fait que recopier la valeur de `x` dans la case mémoire `y`.

Exercice 3

Q1. Le programme affiche "Segmentation fault", ou "Erreur de segmentation" : ce message indique que l'on n'a pas le droit de lire à l'adresse NULL.

Q2. En demandant au programme de lire l'adresse 25 :

```c
#include <stdio.h>

int main(){
  int* p = 25;
  printf("L'entier stocké à l'adresse numéro 25 est %d\n", *p);
  return 0;
}
```

On obtient la même erreur : interdit de lire à l'adresse numéro 25 ! De manière

générale, on ne peut pas **choisir** les adresses où l'on stocke les données (en tout cas pas à notre niveau). Lorsque l'on crée une variable `int x` ou `float y`, le programme leur réserve le bon nombre d'octets dans un emplacement arbitraire que l'on ne peut pas (a priori) prédire.

Exercice 4

Q1. Oui.

Q2. Cette méthode simple d'échange en 3 temps est assez classique : on utilise une variable pour stocker temporairement la valeur pointée par `pa` afin de pouvoir l'écraser :

```c
/* Échange le contenu des cases mémoires pointées par pa et pb */
void echange(float* pa, float* pb){
  float temporaire = *pa;
  *pa = *pb;
  *pb = temporaire;
}
```

Q3. Cette fonction doit à la fois renvoyer le nombre de solutions ET utiliser les pointeurs donnés en entrée pour "renvoyer" également les solutions :

```c
/* Résout l'équation quadratique aX^2 + bX + c = 0
   et renvoie le nombre de solutions réelles (0, 1 ou 2).
   Stocke également la ou les racines réelles dans les zones
   pointées par x1 et x2. */
int quad_solve(float a, float b, float c, float* x1, float* x2){
  float delta = b*b - 4*a*c;
  if (delta < 0){
    return 0;
  } else if (delta == 0){
    *x1 = -b / (2*a);
    return 1;
  } else { // delta négatif
    *x1 = (-b + sqrt(delta))/(2*a);
    *x2 = (-b - sqrt(delta))/(2*a);
    return 2;
  }
}
```

En C, on ne peut pas renvoyer plusieurs valeurs à la fois, contrairement au Python où l'on pourrait renvoyer des couples, triplets, etc... En revanche, les fonctions peuvent prendre en entrée des pointeurs qui leurs permettent de modifier la mémoire, et donc de simuler le renvoi de plusieurs valeurs, comme c'est le cas dans la fonction `quad_solve`.

Voici un exemple de main utilisant les trois fonctions de cet exercice. On remarque que pour la fonction `quad_solve`, on doit utiliser la valeur renvoyée (0, 1 ou 2) pour savoir si l'on peut utiliser les variables `x1, x2`. Par exemple, si la fonction renvoie `0`, alors `x1, x2` ne contiennent pas d'informations intéressantes.

```c
int main(){
  // test incrémenter
  int x = 3;
  printf("Avant: %d\n", x);
  incrementer(&x);
  printf("Après: %d\n", x);

  // test échange
  float a = 3.3, b = 9.47;
  printf("Avant échange: a = %f, b = %f\n", a, b);
  echange(&a, &b);
  printf("Après échange: a = %f, b = %f\n", a, b);


  // test quad_solve
  float x1, x2;

  // X^2 + 2X + 1: une solution
  int n1 = quad_solve(1, 2, 1, &x1, &x2);
  assert(n1 == 1);
  printf("Solution de X^2 + 2X + 1 = 0: %f\n", x1);

  // X^2 + 2X - 3 = (X - 1)(X+3)
  int n2 = quad_solve(1, 2, -3, &x1, &x2);
  assert(n2 == 2);
  printf("Solutions de X^2 + 2X - 3 = 0: %f et %f\n", x1, x2);

  // X^2 5 = 0: pas de solution réelle
  int n3 = quad_solve(1, 0, 5, &x1, &x2);
  assert(n3 == 0);

  return 0;
}
```