TD n°2 - Pointeurs

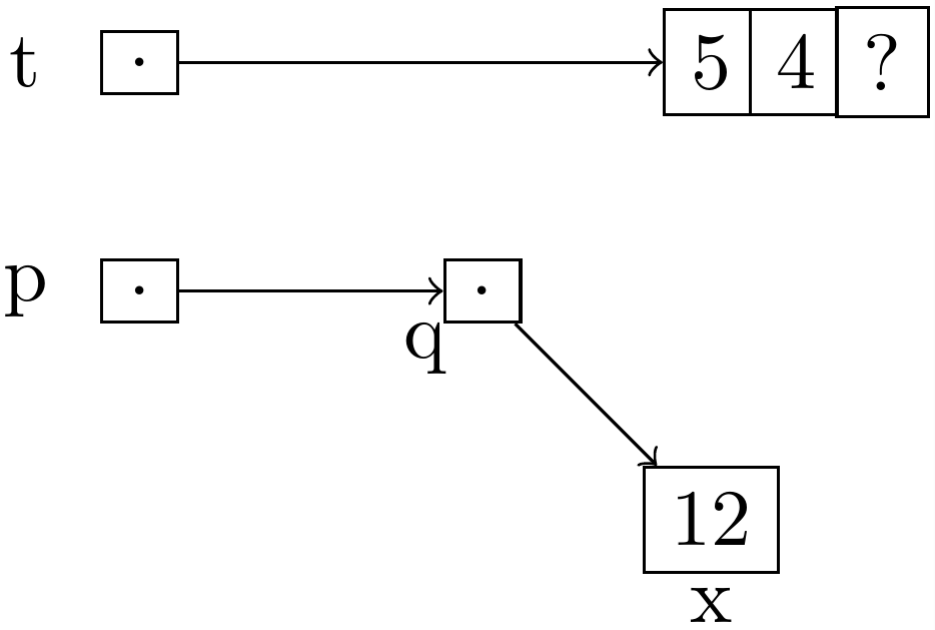
Exercice 1

Indiquer l'évolution, instruction par instruction, de l'état de la mémoire lorsqu'on exécute toutes les instructions suivantes :

```
int n = 1;
int m = 7;
int* p = &n;
int* q = &m;
n += 1;
*q = *p + 3;
*p += 2;
p = q;
*p += 1;
if (p == q) {*q += 1;}
if (p != NULL) {*p += 1;}
int** pp = &p;
int** qq = &q;
*pp = NULL;
if (p != NULL) {*p += 1;}
if (qq != NULL && *qq != NULL) {**qq *= 2;}
```

Exercice 2

Écrire une suite d'instructions en C (la plus simple possible) qui permet d'obtenir l'état suivant de la mémoire. on se placera dans la fonction `main`.



Exercice 3

On propose le programme suivant pour échanger les valeurs de deux variables :

```
void swap(int x, int y){
 int tmp = x;
 x = y;
 y = tmp;
}

int main(){
int x = 3;
int y = 5;
swap(x,y);
}
```

1. Si on regarde les valeurs de x et y après `swap(x,y)`, on observe que x vaut 3 et y vaut 5. Pourquoi ?
2. Écrire un fonction `void swap_bis(int* a, int* b)` qui prend en entrée des pointeurs et échange les valeurs de *a et *b.

3. Écrire une fonction `void minimax(int* a, int* b)` qui met le minimum entre *a et *b dans *a et le maximum dans *b

Exercice 4

On considère la fonction suivante :

```
void mystere(int* x, int* y) {
  *x = *x - *y;
  *y = *x + *y;
  *x = *y - *x;
}
```

1. D'après vous, que fait cette fonction pour deux pointeurs entiers x et y donnés ? Tester sur des exemples puis le prouver.
   Indications de rédaction : On nommera x_0 et y_0 les valeurs des variables $*x$ et $*y$ avant le début du code et x_i et y_i ces mêmes valeurs après la i-ème instruction. Il faut établir le lien entre x_0, y_0, x_3 et y_3.
2. Supposons maintenant qu'on exécute le code suivant :

   ```
   int x3 = 3;
   mystere(&x3, &x3);
   printf("x3 = %d\n", x3);
   ```

   Que se passe-t'il ? Adaptez votre preuve.
   Remarque : Lors d'une preuve de programme, on doit toujours s'intéresser aux cas limites des entrées, c'est à dire les entrées qui donnent (ou sont susceptibles de donner) un fonctionnement différent de la normale. Par exemple le fait que deux pointeurs qui ont l'air différents pointent vers la même case.

Exercice 5

On a expliqué en cours que les pointeurs servent pour écrire des fonction qui modifient des variable qui existent en dehors de la fonction. Une autre utilité des pointeurs est de renvoyer plusieurs choses dans une fonction.

En C, une fonction ne peut renvoyer qu'une seule chose. Si on souhaite obtenir plusieurs informations de la fonction, on peut lui donner des pointeurs à "remplir".

Par exemple, on peut écrire une fonction `int div_euclidienne(int a, int b, int* r)` qui renvoie le quotient de la division euclidienne de a par b et stocke le reste dans la case vers laquelle r pointe. De cette façon, en créant r avant l'appel de fonction, le reste peut être conservé après la fin de la fonction.

Plus précisément le `main` doit ressembler à ceci :

```
int main(){
   int* r = malloc(sizeof(int)); //On crée un pointeur vers une nouvelle case
   div_euclidienne(123,6,r);
   printf("%d", *r); //Affiche la valeur du reste
}
```

1. Écrire cette fonction.
2. Écrire une fonction qui prend en entrée un tableau, sa taille et un élément x et renvoie à la fois le nombre de fois où x apparait et aussi l'indice de sa première occurence.