Exercice 1 :

Ecrire une fonction **récursive**

- qui prend en paramètre un tableau d'entiers t (tous positifs sauf le dernier élément qui est égal -1) et un entier positif nbre et
- qui retourne le nombre d'occurence de l'entier nbre dans le tableau t.

Exercice 2 :

Que retourne comme valeur la fonction "mystere2" suivante pour

- a=14, b=14.
- a=20, b=55.
- a=30, b=12.

Expliquer votre réponse.

```
int mystere2 (unsigned int a, unsigned int b){
    unsigned char a_ce_stade=0;
    if (! (a^b)) return 0;
    while (a && b){
        if ((a&1) && (!(b&1))) a_ce_stade=1;
        else if (!(a&1) && ((b&1))) a_ce_stade=-1;
        a>>=1;
        b>>=1;
    }
    if (a) return 1;
        else if (b) return -1;
            else return a_ce_stade;
}
```

Exercice 3 :

Exécuter le programme suivant :

```
1
2
3  #include <stdio.h>
4  int main(void)
5  {
6      int t[3]={10,-20,320};
7      printf ("l'adresse de t est  : %p \n", &t);
8      printf ("l'adresse de t[0] est  : %p \n", &t[0]);
9      printf ("la valeur de t est  : %p \n", t);
10     return 0;
11 }
```

Que faut-il conclure ?

Exercice 4 :

Ecrire une fonction **itérative**

- qui prend en paramètre un tableau d'entiers t (tous positifs sauf le dernier élément qui est égal -1) et
- qui imprime **de manière inversée** les nombres positifs du tableau t (du dernier nombre au premier nombre).

La fonction ne retourne pas de valeurs (de type void).

Exercice 5 :

- Ecrire une fonction **récursive** qui :
  - prend en paramètre un tableau d'entiers (tous positifs sauf le dernier élément qui est égal -1) et
  - retourne la somme des entiers positifs qui le composent.
- Réécrivez la fonction précédente sans utiliser les symboles [] utilisés dans les tableaux.
- Tester votre fonction depuis le main, en utilisant dans un premier temps un tableau, puis une allocation dynamique.

Exercice 6 :

Exécuter le programme suivant :

```
1
2
3  #include <stdio.h>
4  int main(void)
5  {
6      int t[3]={10,-20,320};
7      printf ("La traille de &t est  : %lu \n", sizeof(&t));
8      printf ("La traille de t[0] est  : %lu \n", sizeof(t[0]));
9      printf ("La traille de &t[0] est  : %lu \n", sizeof(&t[0]));
10     printf ("La traille de t est  : %lu \n", sizeof(t));
11     return 0;
12 }
```

Que faut-il conclure ?

Exercice 7:

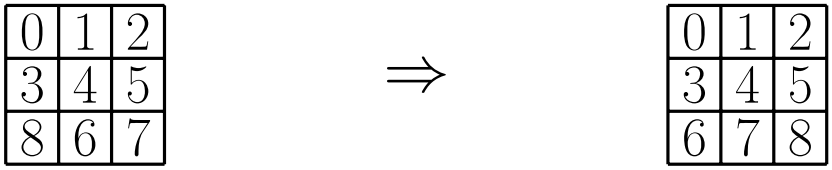
- En utilisant les opérations bit à bit, écrire une fonction itérative en C qui prend en paramètre un entier positif *nb* et retourne 1 si le nombre passé en paramètre est une puissance de 2. La fonction retourne 0 sinon.
  Par exemple, si l'entier passé en paramètre est 14, alors la fonction retournera 0. Par contre, si l'entier passé en paramètre est 16, alors la fonction retournera 1
- Reprendre la question précédente sans utiliser d'opérateurs arithmétiques ni d'opérateurs de comparaison.

Exercice 8:

Écrire une fonction **récursive** en C qui prend en paramètre un entier positif n et retourne le nombre de bits à 1 dans la représentation binaire de cet entier.

Exercice 9:

On souhaiterait écrire des fonctions qui aideraient à résoudre le problème du taquin $k*k$.
Le but est, étant donné, un taquin de départ ou initial (exemple ci-dessous) de donner une séquence de déplacements de la case "vide" (représentée ici par le nombre 0) qui permettrait d'atteindre un taquin but (exemple ci-dessous).



On suppose que les entiers des taquins à deux dimensions seront représentés dans un espace mémoire contigu de k*k éléments (une seule dimension). Le but aussi est d'écrire des fonctions simples pour passer d'une représentation à deux dimensions (plus intuitive) à une représentation à une dimension, et vice versa.

1. Déclarer une constante k (par exemple k est égale à 3) grâce à la directive #define.
2. Ecrire une fonction qui :
   - lit (avec la fonction scanf) $k*k$ entiers positifs (associés à un taquin à résoudre),
   - les stocke dynamiquement dans un espace mémoire (grâce à malloc) et
   - retourne l'adresse mémoire de cet espace.
3. Cette fonction sera utilisée par le main pour initialiser
   - le taquin initial, où la séquence à lire est :
     0 1 2 3 4 5 8 6 7
   - et le taquin but, où la séquence à lire est :
     0 1 2 3 4 5 6 7 8
4. Ecrire une fonction qui prend en paramètres l'adresse où sont stockés les entiers positifs d'un taquin et affiche le taquin en deux dimensions. Dans notre exemple, l'affichage attendu pour la taquin initial est :

   0 1 2
   3 4 5
   8 6 7

5. Ecrire une fonction
   - qui prend en paramètre une variable de type pointeur qui représente l'adresse où sont stockés les entiers positifs d'un taquin et
   - qui retourne 1 si chacun des entiers est compris entre 0 (dans la suite représentera le blanc) et ($k*k$-1).
     Elle retournera 0 sinon. Dans notre exemple, si k=3 la fonction retournera 1 si les nombres sont compris entre 0 et 8 et retournera 0 sinon.
6. Ecrire une fonction qui échange deux entiers passés en paramètres. La fonction ne retourne pas de valeur.
7. Ecrire une fonction, appelée est_but, qui
   - prend en paramètre deux pointeurs (taquin départ et taquin but),
   - et retourne 1 si les deux taquins sont égaux et retourne 0 sinon.
8. Ecrire une fonction, appelée position_du_vide, qui :
   - prend en paramètre une variable pointeur qui représente l'adresse où sont stockés les entiers positifs d'un taquin, et
   - deux pointeurs vers des entiers qui vont contenir les coordonnées (numéro de la ligne et numéro de la colonne) de la case vide (représentée par l'entier 0).
   - La fonction ne retourne pas de valeur.
9. Ecrire une fonction, appelée resultat_dune_etape, qui
   - prend en paramètre un pointeur qui contient l'adresse où sont stockés les entiers positifs d'un taquin, et
   - un caractère c qui peut-être sous la forme de l'une des lettres suivantes : 'b', 'h', 'g' et 'd'. Ces lettres indiquent respectivement un déplacement de la case vide vers le bas, vers le haut, vers la gauche et vers la droite.
   - La fonction retourne 1 si le déplacement est possible et retourne 0 (hors cadre) sinon.
10. Reprendre la question suivante où
    - dans l'hypothèse où le déplacement est possible, la fonction modifie le taquin selon les directives du caractère c.
11. Tester vos fonctions depuis le programme principal main avec l'exemple ci-dessus.