Exercice 1:

Ecrire une fonction qui prend en paramètre un entier positif n et retourne 1 si le nombre n est uniforme. La fonction retourne 0 sinon. Un nombre est dit uniforme s'il est une répétition d'un même chiffre. Par exemple les nombres 111, 33333, 7777 sont des nombres uniformes alors que les nombres 123, 3332, 8879179 ne le sont pas.

Exercice 2:

Ecrire une fonction récursive qui réalise la fonction de Hanoï vue en cours. Tester votre fonction avec n=2, n=3, n=10, n=50 (Hum!)

Exercice 3:

Écrire un programme en C qui :

- lit un entier n,
- lit n entiers positifs ou nuls, puis les stocke en mémoire grâce à une allocation dynamique,
- affiche d'abord le nombre d'entiers nuls,
- affiche ensuite les nombres pairs,
- et enfin, affiche les nombres impairs.

Exercice 4:

On se donne une table (que l'on appellera baguenaudière) à n cases, chacune peut contenir un pion. Chaque case est numérotée de "1" à "n". La baguenaudière peut être vide ou pleine. Le but du jeu est de remplir la baguenaudière si elle est vide, ou de la vider si elle est pleine, en respectant les règles suivantes :

- Pour la case 1, on peut enlever un pion ou mettre un pion sans contrainte.
- Pour la case 2, on peut enlever un pion ou mettre un pion uniquement si la case 1 est pleine
- De manière générale, pour la case i (différente de 1 et 2), on peut enlever un pion ou mettre un pion si :
  - La case $i-1$ est pleine (contient un pion)
  - Toutes les cases de 1 à $i-2$ sont vides.

Ecrire les fonctions remplir et vider. On utilisera la récursivité croisée pour écrire vos fonctions.

Exercice 5:

Écrire une fonction en langage `C` ayant l'entête suivante :

```
int trois_lettres_consecutives(void);
```

Cette fonction lit une suite de lettres minuscules saisies au clavier, terminée par le caractère `$`. Elle doit lire les caractères un par un à l'aide de `getchar()`, sans utiliser de tableau.

La saisie peut contenir :

- 0, 1, 2 ou plusieurs lettres avant le caractère `$` ;
- uniquement des lettres minuscules (aucune autre vérification n'est nécessaire).

La fonction doit ensuite vérifier s'il existe trois lettres consécutives dans l'ordre alphabétique (par exemple : `abc`, `cde`, `xyz`, etc.).

Elle doit retourner :

- `1` s'il existe trois lettres consécutives ;
- `0` sinon.

Exercice 6:

1. En utilisant les allocations dynamiques, on vous demande d'écrire un programme C qui :

   - lit un mot (caractère par caractère et en utilisant seulement l'instruction de lecture getchar()) de n lettres alphabétiques et
   - affiche d'abord le nombre de lettres minuscules ainsi que les lettres minuscules, puis
   - affiche le nombre de lettres majuscules ainsi que les lettres majuscules.

   Par exemple,

   - si $n = 5$ et le mot lu est : "AccBE" alors le programme affichera "ccABE".

Exercice 7:

- Écrire une fonction en C qui prend en paramètres deux entiers positifs `a` et `b`, et retourne `1` si `a` est stricteemnt plus grand que `b`, `-1` si `a` est strictement plus petit que `b`, et `0` s'ils sont égaux.

  Exemples :

  - Si `a = 16` et `b = 15`, la fonction retourne `1`.
  - Si `a = 20` et `b = 30`, la fonction retourne `-1`.
  - Si `a = 18` et `b = 18`, la fonction retourne `0`.

- Reprendre l'exercice précédent, mais cette fois-ci, aucun opérateur arithmétique (`+`, `-`, etc.) ni de comparaison (`==`, `>`, `<`, `>=`, `<=`) ne doit être utilisé.

Exercice 8:

Le but de cet exercice est d'illustrer qu'un problème peut avoir des solutions différentes

- Ecrire une fonction C qui prend en paramètre un entier positif b et retourne un nombre positif a tel que b=a*a (on suppose qu'un tel nombre a existe). Les variables a et b sont déclarés comme unsigned long int.
  Tester votre programme avec b=64 et b=766680721.

- Ecrire une fonction C qui prend en paramètre deux nombres flottant x et y et retourne 1 si $x-\epsilon \leq y \leq x+\epsilon$. La fonction retourne 0 sinon. La constante ϵ est définie avec la directive #define.

- Ecrire une fonction C qui prend en paramètre un entier réel positif b et retourne un nombre positif a tel que $(a*a)-\epsilon \leq b \leq (a*a)+\epsilon$ (on suppose qu'un tel nombre a existe). Dans cette question, les variables a et b sont déclaré comme des "double" ou "float".
  Tester votre programme avec b=98.71 et $\epsilon = 0.001$.

Exercice 9:

Cet exercice (un peu difficile) a un double objectif :

- Illustrer l'utilisation de la récursivité,
- exploiter le fait qu'un tableau est un pointeur particulier pour résoudre notre problème.

Présentons d'abord le problème à résoudre. Il s'agit de résoudre une version simplifiée du jeu des chiffres qui se définit comme suit :

- on se donne une suite de n nombres positifs terminée par -1. Cette suite de nombres sera stockée dans un tableau, appelé *tnombres*.
- on se donne un nombre positif t, et
- le but est de voir s'il existe un sous-ensemble d'éléments de *tnombres* tel que leur somme est égale à t.

On vous demande :

- de déclarer la constante n grâce à la directive #define.

- d'écrire une fonction ** récursive **, appelée jeux_chiffres, qui a l'entête suivante :

$$int\ jeux\_chiffres\ (int\ t, int\ tnombres[])$$

- Cette fonction :
  - admet deux paramètres un entier t et un tableau tnombres d'entiers,
  - retourne 1 si le problème admet une solution et 0 sinon.
  - Si la fonction retourne 1, on demande aussi d'afficher une des solutions.

Exemple :

- Supposons que notre tableau contient les éléments $\{12, 5, 3, 7, -1\}$ (rappel -1 ne fait pas partie des nombres mais juste indiquer la fin de la suite des nombre).
- Supposons que $t = 23$. La fonction retourne 0 pour indiquer l'absence de solution.
- Maintenant si $t = 24$ la fonction retourne 1 et affichera les nombres 7, 5 et 12.

NB. Des indications pour écrire cette fonction sont disponibles dans une feuille séparée.

Indications : Voici quelques indications pour résoudre cet exercice difficile.

- Notez que tous les nombres sont positifs.
- Notez que dans la fonction jeux_chiffres la taille n'est pas précisée. Ce qui signifie que l'on peut passer en paramètres un tableau de taille n, ou un tableau de taille n-1 ou même un tableau de taille 1. Regarder comment vous pouvez concrètement passer un sous-tableau.
- Comme un tableau est un pointeur, on peut facilement définir un sous-tableau de *tnombres*.
- Exploiter le principe de la récursivité : pour écrire la fonction avec les paramètres (t et *tnombres*) vous pouvez considérer comme acquis la même fonctions mais pour des valeurs inférieurs à t ou pour tout sous-tableau de *tnombres*.
- Il ne vous reste qu'à écrire la fonction et commencer par identifier les conditions d'arrêt de votre fonction récursive.