Exercice 1:

Ecrire un programme C qui :

- lit un chiffre positif (compris entre 0 et 9), et
- affiche la table de multiplication (jusqu'à 10) associée à ce chiffre.

Par exemple, si le chiffre lu est 3 alors le programme affichera :

$$
\begin{array}{lllll}
3 & \times & 0 & = & 0 \\
3 & \times & 1 & = & 3 \\
3 & \times & 2 & = & 6 \\
3 & \times & 3 & = & 9 \\
3 & \times & 4 & = & 12 \\
3 & \times & 5 & = & 15 \\
3 & \times & 6 & = & 18 \\
3 & \times & 7 & = & 21 \\
3 & \times & 8 & = & 24 \\
3 & \times & 9 & = & 27 \\
3 & \times & 10 & = & 30
\end{array}
$$

Exercice 2:

Ecrire une fonction C qui :

- prend en paramètre un entier naturel positif n,
- alloue, avec la fonction malloc, un espace mémoire pour stocker n éléments, chacun de taille sizeof(int) et
- retourne :
  - 0 si l'allocation a échouée,
  - 1 sinon.
- Le résultat de l'allocation est stocké dans une variable locale.
- Si l'allocation s'est bien déroulée, on demande d'imprimer l'adresse du premier octet de l'espace réservé puis de libérer l'espace allouée avant de retourner la valeur 1.
- Si l'allocation n'est pas bien déroulée, on demande d'imprimer un petit message d'erreur avant de retourner la valeur 0.

Exercice 3:

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 4:

Considérons de nouveau la fonction "absolu" et son utilisation dans le main.

```
#include <stdio.h>
int absolu (int a)
{
  if (a<0)
    return -a;
  else return a;
}
```

```
int main (void)
{
    int absolu (int a);
    int i=-20;
    printf ("La valeur absolue de i est : %d.\n ", absolu(i));
    return(0);
}
```

- Modifier le programme pour afficher :
  - l'adresse du paramètre a,
  - l'adresse de la variable i utilisée dans l'appel à la fonction "absolu" depuis le main, et
  - la valeur qu'a obtenu le paramètre a tout au début de l'exécution de la fonction "absolu".
- Que faut-il conclure?

Exercice 5:

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 6:

Ecrire un programme C (sans utiliser de tableaux ou d'allocations dynamiques)[1] qui :

- lit un nombre positif n inférieur ou égal à 8 (utiliser la boucle do ... while pour avoir la bonne valeur de n),
- lit n chiffres et
- affiche ces chiffres dans l'ordre inverse de la lecture.

Par exemple,

- Supposons que n=8.
- Supposons que les nombres lus sont :

  2 6 3 8 9 4 3 4

- alors le programme affichera :

  4 3 4 9 8 3 6 2

  Reprendre l'exercice en utilisant les allocations dynamiques.

[1] Indication : mémoriser les chiffres lus dans un entier de type "long int".

Exercice 7:

NB. Penser à écrire des petites fonctions ré-utilisables.

Un enseignant en informatique rédige chaque exercice dans un fichier indépendant. Il a nommé ses fichiers exo1.tex, exo2.tex, etc. De même, les solutions des exercices sont également rédigées dans des fichiers indépendants, nommés exo1-solution.tex, exo2-solution.tex ...

Pour inclure un exercice exo$i$ (ou sa solution) dans un TD ou TP, l'enseignant utilise la commande d'inclusion de fichiers (latex) :

\input "exo$i$.tex"

L'enseignant souhaite disposer d'un programme qui génère automatiquement les instructions d'inclusion de fichiers.

1. Ecrire une fonction C, qui :
   - n'admet pas de paramètres et ne retourne aucune valeur,
   - demande à l'utilisateur de rentrer un nombre n
   - affiche les instructions d'inclusion de fichiers de exo1.tex à exon.tex.
   - Par exemple, si n=3, la fonction affichera :
     \input "exo1.tex"
     \input "exo2.tex"
     \input "exo3.tex"

2. Modifier votre fonction, afin que l'utilisateur puisse préfixer son nombre du caractère 'a' ou 's' pour indiquer s'il souhaite ou non avoir les solutions.
   - Par exemple, si l'utilisateur rentre a2, la fonction affichera :
     \input "exo1.tex"
     \input "exo1-solution.tex"
     \input "exo2.tex"
     \input "exo2-solution.tex"
   - Par contre s'il rentre s2, la fonction affichera :
     \input "exo1.tex"
     \input "exo2.tex"

3. Modifier votre fonction pour afin que l'utilisateur puisse rentrer une liste d'exercices
   - séparés par une virgule,
   - chacun préfixé de 'a' ou de 's', et
   - la fonction s'arrête à la rencontre du caractère 'x'
   - Par exemple, si l'utilisateur rentre "a2,a5,s6,x" la fonction affichera :
     \input "exo2.tex"
     \input "exo2-solution.tex"
     \input "exo5.tex"
     \input "exo5-solution.tex"
     \input "exo6.tex"

4. Modifier votre fonction pour afin que l'utilisateur puisse rentrer une liste d'exercices sous forme d'intervalles
   - séparés par une virgule,
   - chacun intervalle est préfixé de 'a' ou de 's', et
   - la fonction s'arrête à la rencontre du caractère 'x'
   - Par exemple, si l'utilisateur rentre "a2-3,s6-6,x" la fonction affichera :
     \input "exo2.tex"
     \input "exo2-solution.tex"
     \input "exo3.tex"
     \input "exo3-solution.tex"
     \input "exo6.tex"

Exercice 8:

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.