## 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:

Écrire une fonction en langage C qui prend en paramètres deux entiers positifs $a$ et $b$, et qui affiche l'ensemble des chiffres qu'ils ont en commun. Si aucun chiffre n'est commun, la fonction affichera "Désolé, pas de chiffres en commun". La fonction ne retourne pas de valeur.

**Exemples :**

- Si $a = 19097$ et $b = 27349$, la fonction affichera $\{7, 9\}$, car ce sont les seuls deux chiffres qui apparaissent à la fois dans $a$ et $b$.
- Si $a = 2097$ et $b = 456666$, la fonction affichera "Désolé, pas de chiffres en commun", car aucun chiffre n'est commun entre $a$ et $b$.

## Exercice 3:

Écrire une fonction en C qui lit une suite de caractères et retourne 1 si les caractères lus sont triés par ordre croissant. La fonction retourne 0 sinon. La lecture se fait caractère par caractère où seul getchar() est autorisé. La saisie se termine par le caractère '$', qui ne doit pas être pris en compte dans le traitement. La fonction n'admet pas d'arguments.

**Exemples :**

- Si l'utilisateur saisit abeg$, alors la fonction retournera 1.
- Si l'utilisateur saisit DFAZE$, alors la fonction retournera 0.

## Exercice 4:

Sans utiliser les opérations bit à bit, écrire une fonction récursive en C qui prend en paramètre un entier positif $nb$ et retourne le nombre de bits égaux à 1 dans sa représentation binaire. Par exemple, pour l'entier passé en paramètre 14, qui s'écrit 1110 en binaire, la fonction doit retourner 3.

## Exercice 5:

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 6:

Ecrire une fonction C qui prend en paramètre un entier positif $n$ et retourne le nombre de diviseurs stricts (inférieurs à n) du nombre $n$. Par exemple,

- si le nombre n est égal à 12, alors la fonction retourne 5 (car le nombre 12 admet 5 diviseurs stricts 1, 2, 3, 4, 6).

## Exercice 7:

Écrire une fonction qui prend en paramètre un entier positif $a$ et un indice $n$, et qui inverse le n-ième bit de cet entier (change 0 en 1 ou 1 en 0). L'indice $n$ est compris entre 0 et (sizeof($a$)*8)-1. La fonction doit retourner le nouvel entier après l'inversion du bit.

**Exemple :**

Si l'entier est $a = 10$ (en binaire 1010) et que l'on souhaite inverser le 2ème bit (en partant de 0), alors la fonction retournera 14 (en binaire 1110).

## Exercice 8:

- 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 9:

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 10:

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 $\epsilon$ 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 $\epsilon = 0.001$.