Colle n°1

Tous vos fichiers sont à renommer comme d'habitude (en y ajoutant votre nom en majuscule) et à envoyer par mail à `geraldine.olivier@ac-bordeaux.fr` à la fin de la colle.
Pensez à toutes les bonnes pratiques de programmation : réfléchir sur papier, tester dès que cela est possible, méthode des petits pas, programmation défensive...
Si vous avez terminé en moins d'1h, venez me demander un exercice supplémentaire.

Exercice 1 (Affichage d'un motif à l'écran).
On veut écrire en langage C une fonction `void draw(int n)` qui affiche sur la sortie standard une grille $n \times n$ de caractères de la manière suivante :

- si la position du caractère (i, j) dans la grille est telle que i et j n'ont aucun bit en commun, le code affiche `*` à cet endroit de la grille.
- sinon, le code affiche un caractère espace à cet endroit de la grille.

i et j sont des entiers encodés selon le type `int` du C. On rappelle que le OU bit à bit s'écrit — et que le ET bit à bit s'écrit &.

1. Dessiner la grille attendue pour $n = 16$.

2. Coder la fonction `draw` en C.

La forme obtenue s'appelle le triangle de Sierpinski.

Exercice 2 (Décomposition binaire d'un entier en OCaml).
Les fonctions de cet exercice sont à coder en langage OCaml dans un fichier nommé `dec2bin.ml`

1. Écrire une fonction `nombre_de_bits` qui calcule le nombre de bits minimal nécessaire à l'encodage d'un entier naturel.

2. Écrire une fonction itérative `dec2bin` qui renvoie un tableau de 0 et de 1 correspondant à la décomposition binaire d'un nombre entier donné

3. Créer une fonction `nombre_de_bits_rec` qui fait le même travail que `nombre_de_bits` mais est formulée sous forme récursive

4. (Question plus difficile, à traiter en dernier) : écrire la fonction récursive `dec2bin_rec`, qui fait le même travail que `dec2bin`, en utilisant des listes OCaml.

5. (Question encore plus difficile) : essayer de mettre ces deux fonctions sous forme récursive terminale. Vous créerez une 3eme version de vos deux fonctions appelées : `nombre_de_bits_rec_term` et `dec2bin_rec_term`. *Indication : fonction auxiliaire, accumulateurs...*

Exercice 3 (Statistiques descriptives).
Les fonctions suivantes seront codées en langage C dans un fichier `stats.c`.

1. Avec votre calculatrice ou sinon à la main, créer 2 séries de nombres flottants de tailles différentes et raisonnable. Calculer leur moyenne, leur écart-type, le premier quartile, la médiane et le dernier quartile.
2. Écrire un code qui prend en entrée une série de nombres réels donnée par l'utilisateur et qui calcule la moyenne de toutes les valeurs données. On nommera `moyenne` la fonction qui renvoie la moyenne de la série. On pourra utiliser la fonction de conversion `atof` qui convertit une chaîne de caractère en nombre flottant 32 bits.
3. Testez votre fonction sur 2 séries de différentes tailles. Validez les résultats sur les deux séries choisies à la première question.
4. Créer une fonction `ecart_type` qui calcule l'écart-type de la série. Validez les résultats sur les deux séries choisies à la première question. Utiliser la librairie `math.h` pour la racine carrée `sqrt`. Il faudra alors rajouter l'option `-lm` sur la ligne de commande lors de la compilation.
5. Créer une fonction `quartiles` qui renvoie le premier quartile, la médiane et le troisième quartile de la série. *Indication : Vous pourrez réutiliser le tri à bulle codé dans un précédent TP.*
   Rappel : le premier quartile d'une série la valeur de la série qui est telle que l'ensemble des valeurs inférieures à cette valeur représente au moins un quart du nombre total de valeurs de la série. Je vous laisse déduire la définition de la médiane et du troisième quartile. Pour la médiane, si le nombre de valeurs de la série est paire, on prend par convention la valeur moyenne entre les deux valeurs avant et après la moitié du nombre total de valeurs.