Exercice 1 (Affichage) :

1. Afficher un triangle rectangle isocèle avec des étoiles ; vous demanderez à l'utilisateur la hauteur. Par exemple, avec une hauteur de 4 :

```
Output
*
**
***
****
```

2. Afficher un échiquier de taille 8×8. Pour afficher des cases blanches et noires, vous pouvez utiliser le caractère Unicode correspondant, dans notre cas `\u25A1` pour une case blanche et `\u25A0` pour une noire.

Exercice 2 (Polynômes du second degré) : On peut représenter un polynôme du second degré $aX^2 + bX + c$ sous la forme d'un triplet de valeurs (a, b, c).

1. Écrire une fonction `discriminant` qui prend en paramètres trois valeurs entières (a, b, et c), et retourne le discriminant Δ de l'équation $aX^2 + bX + c = 0$.
2. Écrire une fonction `nb_solutions` qui prend en paramètres trois valeurs entières (a, b, et c), et retourne le nombre de solutions de l'équation $aX^2 + bX + c = 0$. Cette fonction devra utiliser la fonction définie à la question précédente.
3. Écrire une fonction `est_solution` qui prend en paramètres trois valeurs entières (a, b, et c), ainsi qu'une valeur x, et retourne `true` si x est solution de $aX^2 + bX + c = 0$, et `false` sinon. On importera la librairie `stdbool.h`.

Exercice 3 (Fonctions impératives et récursives) :

1. Écrire une fonction `somme` de manière impérative, telle que `somme(n)` renvoie la somme des entiers de 1 à n.
2. Écrire une fonction `somme_rec` calculant la même chose, mais fonctionnant de manière récursive.
3. Écrire une fonction `factorielle` de manière impérative, telle que `factorielle(n)` renvoie $n!$.
4. Écrire une fonction `factorielle_rec` calculant la même chose, mais fonctionnant de manière récursive.
5. Tester les valeurs renvoyées par `factorielle(-1)` et `factorielle_rec(-1)`. Si vous observez un problème, comment y remédier ?
6. Écrire deux fonctions mutuellement récursives `pair` et `impair` qui s'appellent l'une l'autre pour déterminer si un entier n est pair ou impair.
7. Écrire une fonction `pgcd` telle que `pgcd(a,b)` renvoie le PGCD des entiers a et b.
   Rappel : $PGCD(a, 0) = a$; $PGCD(a, b) = PGCD(b, a \mod b)$.
8. Écrire une fonction `fibo_rec` de manière récursive, telle que `fibo_rec(n)` renvoie le n-ème terme de la suite de Fibonacci.
9. Calculer `fibo_rec(12)`. Qu'observez-vous ?

Exercice 4 (Tableaux) :

1. Écrire une fonction de signature `void init_tab(int *t, int n)` qui prend en paramètres un tableau `t` et sa taille n, et demande à l'utilisateur de saisir une à une les valeurs à mettre dans le tableau.
2. Écrire une fonction de signature `int somme_tab(int *t, int n)` qui prend en paramètres un tableau `t` et sa taille n, et revoie la somme des éléments du tableau.
3. En déduire une fonction `float moyenne_tab(int *t, int n)` qui renvoie la moyenne des valeurs de `t`.
4. Écrire une fonction `void minmax_tab(int *t, int n)` qui affiche la valeur minimale et la valeur maximale contenue dans `t`.
5. Écrire une fonction `void copie_tab(int *t1, int *t2, int n)` qui copie le contenu du tableau `t1` dans le tablau `t2` (tous les deux de taille n).

Exercice 5 (Matrices) : Il est possible de représenter en C une matrice par l'intermédiaire d'un tableau à deux dimensions (autrement dit : un tableau de tableaux).

Exemple

```c
int matrix[10][20]; // un matrice 10 * 20
```

Si une fonction ne doit fonctionner que pour une taille fixée, on peut la préciser dans l'entête.

Exemple

```c
void g(int m[10][20])
{
  ...
}
```

1. Écrire une fonction qui affiche une matrice 3×3.
2. Écrire un programme demandant à l'utilisateur de remplir une matrice 3×3 avec des entiers dans $[\![1, 9]\!]$, puis qui vérifie si c'est un **carré magique** (les sommes des chiffres sur les lignes, les colonnes, et le diagonales sont égales).
3. Écrire une fonction qui affiche une matrice 5×5.
4. Écrire une fonction qui met dans toutes les cases d'une matrice 5×5 la même valeur n.
5. Écrire une fonction qui prend en paramètres une matrice 5×5 (initialisée avec des zéros), et la remplit avec les valeurs d'un triangle de Pascal de 5 lignes.