1. Faites un programme qui crée un tableau **array** de 10 entiers, puis l'initialise de telle sorte que la valeur de chaque case du tableau soit égale à son indice : t[0] doit valoir 0, ..., t[i] doit valoir i, ... t[9] doit valoir 9.
   Utilisez pour cela une boucle **for**.
2. Faites un programme qui crée un tableau **array** de 8 flottants non initialisé, puis qui affiche la valeur de chaque case du tableau grâce à une boucle **for**.
   Que pouvez-vous dire sur ce programme ?
3. Faites un programme qui calcule les **11** premiers termes de la suite de Fibonacci et les stocke dans un tableau (grâce à une boucle *for*).

$$u_n = \begin{cases} 0 & \text{si } n = 0 \\ 1 & \text{si } n = 1 \\ u_{n-1} + u_{n-2} & \text{sinon} \end{cases}$$

4. Implémentez la fonction **est_egal**, dont prototype est donné ci-dessous. Cette fonction doit renvoyer vrai si et seulement si les deux tableaux passés en paramètre sont égaux. Deux tableaux sont égaux si et seulement si ils ont la même taille et le même contenu.

```
bool est_egal(int taille1, const float tab1[taille1],
              int taille2, const float tab2[taille2]);
```

5. Implémentez la fonction **vsum**, dont le prototype est donné ci-dessous. Cette fonction doit calculer la somme membre à membre de deux tableaux **tab1** et **tab2** passés en paramètre, en stockant le résultat dans le tableau **tab_res**.
   Par exemple, $vsum([1,2,3],[4,5,6])$ devrait donner [5,7,9].

```
void vsum(int taille1, const int tab1[taille1],
          int taille2, const int tab2[taille2],
          int taille_res, int tab_res[taille_res]);
```

6. Implémentez la fonction **est_trie** dont le prototype est donné ci-dessous. Cette fonction doit renvoyer vrai si et seulement si le tableau donné en paramètre est trié par ordre croissant (non strict).

```
bool est_trie(int taille, const int tab[taille]);
```

7. Implémentez la fonction **index_of** dont le prototype est donné ci-dessous. Cette fonction doit renvoyer l'indice de la première occurence de **valeur** dans le tableau **tab**.
   Si **tab** ne contient pas **valeur**, -1 doit être renvoyé.

```
int index_of(int taille, const int tab[taille], int valeur);
```

8. Implémentez la fonction **all_different** dont le prototype est donné ci-dessous. Cette fonction doit renvoyer vrai si et seulement si toutes les valeurs du tableau **tab** sont différentes.

```
bool all_different(int taille, const int tab[taille]);
```

9. Créez un programme qui demande à l'utilisateur un nombre positif $n \leq 10$, crée un tableau **array** à deux dimensions $n \times n$, l'initialise à la matrice identité d'ordre n, puis l'affiche. Résultat attendu :

```
$ ./program
Veuillez entrer n <= 10 : 12
Non. Veuillez entrer n <= 10 : 4
1000
0100
0010
0001
```

10. Une grille de morpion peut être représentée par un tableau 3×3 de caractères, où 'x' représente les pions d'un joueur, 'o' les pions de son adversaire et ' ' les cases vides.

    Créez une fonction **display_grid**, qui affiche une grille de morpion.
    L'affichage doit être de ce type :

```
+---+---+---+
| o |   | x |
+---+---+---+
| o | x |   |
+---+---+---+
| o |   | x |
+---+---+---+
```

11. Créez une fonction **empty_tiles** qui prend une grille de morpion en entrée, calcule les coordonnées de toutes les cases vides de la grille et retourne leur nombre.
    Pour cela, la fonction prend et modifie deux tableaux de taille 9 en entrée : un pour chaque type de coordonnée (un pour les abscisses, un pour les ordonnées). Sur la grille affichée en question précédente, la fonction devrait renvoyer 3, en modifiant coordx=[1,2,1,$\cdots$] et coordy=[0,1,2,$\cdots$].

12. Créez une fonction **is_won**, qui prend une grille de morpion en entrée. La fonction doit calculer si un joueur a gagné ou non. La fonction doit renvoyer un caractère :

$$is\_won(grid) = \begin{cases} \text{'x'} & \text{si le joueur 'x' a gagné} \\ \text{'o'} & \text{si le joueur 'o' a gagné} \\ 0 & \text{sinon} \end{cases}$$

13. Créez un programme qui joue au morpion, en appelant les fonctions précédentes. Ce programme doit jouer de manière aléatoire dans une des cases vides.