Exercice1 :

On aimerait implémenter un programme qui va utiliser un thread pour calculer la somme d'un tableau d'entiers. Le thread devra donc s'occuper de calculer la somme et le programme principal (main) en affichera le résultat.
Pour cette implémentation, aucun argument ne sera passé au thread et celui-ci ne retournera aucune valeur. Le tableau d'entiers sera entré sur la ligne de commande (chaque entier séparé par un espace) et la somme sera stockée dans une variable globale.

Exercice2 :

Nous souhaitons implémenter un programme fonctionnellement identique à l'exercice 1, cependant nous n'utiliserons cette fois-ci aucune variable globale. De plus, le thread retournera la somme du tableau d'entiers à l'aide du mot clé return.

Exercice3 :

Ecrire un programme calculant la somme des entiers de 1 à N à l'aide de M threads. Chaque thread calculera la somme d'un sous-ensemble de ces entiers et la somme globale sera obtenue en calculant la somme des résultats intermédiaires de chaque thread.

Les entiers sont répartis uniformément entre les threads comme suit (exemple avec 3 threads) :

Thread 1 : 1, 4, 7, ...
Thread 2 : 2, 5, 8, ...
Thread 3 : 3, 6, 9, ...

Le programme doit lancer M threads, attendre qu'ils se terminent, faire la somme des résultats intermédiaires et afficher le résultat. Les valeurs N et M seront passées en ligne de commande.

Exercice4 :

1. Intuitivement, que devrait afficher le programme ci-dessous ?
2. Pensez-vous qu'il existe un problème avec l'implémentation du programme ? Vérifiez votre réponse en compilant et exécutant le programme

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <pthread.h>

#define NUM_THREADS     8

void *thread(void *thread_id) {
        int id = *((int *) thread_id);
        sleep(1); // 1s
        printf("Thread %d\n", id);
        return NULL;
}
int main() {
        pthread_t threads[NUM_THREADS];
        int tab[NUM_THREADS];
        int i;
        for (i = 0; i < NUM_THREADS; i++) {
                tab[i] = i;
                int code = pthread_create(&threads[i], NULL, thread, &tab[i]);
                if (code != 0) {
                        fprintf(stderr, "pthread_create failed!\n");
                        return EXIT_FAILURE;


        }



        pthread_exit(NULL);
        return EXIT_SUCCESS;
```