Functions and Pointers

Exercise 1:

Create a function that returns the factorial of a number.
Then, using that function, calculate the sum $S = 1! + 2! + 3! + \ldots + n!$
Now, do it using a void function.
Bonus question: do it without a function using the optimal way.

Exercise 2:

- Write a function to check if a number is even or odd.
- Now, in the main function, create a system where you read an array of integers, and use the previous function to set up the odd numbers on the left, and the even numbers on the right.

Exercise 3:

We want to create a Tic-Tac-Toe (XO) program.

- Create a function that displays the board, knowing that the "board" is a matrix.
- Create a function that checks if the board is in a win state, draw state, or losing state.
- Create a function to check if the chosen slot is empty.
- Create a function that lets the computer choose a random slot to play.
- In the main function:
  - Create an interface that lets you choose the slot to play.
  - After that, use the previous functions to create the game Tic-Tac-Toe.

Hint: use a `current` variable to store the current player (human/computer.)

Exercise 4:

Calculate the sum: $S = 1! + 3! + 5! + \ldots + (2n + 1)!$ using both function types.

Exercise 5:

Create the `pow` and `powf` functions considering that the power is an integer, and could either be positive, negative, or zero.

Exercise 6:

Part A:
Write a program that fills an array `T` of strictly positive integers of size n ($n \leq 100$), then display the greatest prime element in `T` if it exists (without using a function).

Part B:

1) Write a function `Read_Array` allowing to read an array `V` of `N` strictly positive integers.
2) Write a function `Prime` allowing to verify if an integer `x` is prime or not.
3) Write a function `Get_First_Prime` that uses the function `Prime` to get the position of the first prime element in an array `V` of `N` positive integers.
4) Write a main function that uses the previous functions to:
   a) Read an array `T` of integers of size n ($n \leq 100$).
   b) Display the greatest prime element in `T` if it exists.

Exercise 7:

A classic problem, Two Sum. Given an array `V`, create a function that sets up an array of the indices (indexes) of the first pair of elements that add up to a target variable.
Ex: `int V[] = {1, 2, 6}  target = 7` the indices returned: `0` and `2`, since $1 + 6 = 7$.
Bonus Question (Not Necessary): Now, create a function that sets up a matrix of all the possible combinations for that target. Repetition is allowed.

Exercise 8:

- Knowing that a function is declared by `void read(int **V, int n)` where `V` is an array of integers, and `n` is its size, define that function to read the entire array.

Exercise 9 (Quiz):

- Having an array `int V[] = {1, 2, 3}`, `printf("%d", *V++)` prints out:

  a. `1` b. `2` c. `Error`

- A pointer to a pointer to the first element of an array `V` is defined as:

  a. `&V` b. `V` c. `*V`

- Having double pointer `int **V`, we want to dereference it, increment by one, and get the address again.

  a. `&((*V)++)` b. `&(V++)` c. `V++` d. `&(*V)++`

- What is the output of:

```
int arr[] = {5, 10, 15};
int *p = arr + 1;
printf("%d\n", *(p + 1));
```

  a. `Garbage Value` b. `10` c. `15`

Exercise 10:

Say we have `typedef struct{ char name[50]; int age;} Student`.

- Create an array of 3 students (reading their data isn't necessary.)
- Define a pointer to that array.
- Now, imagine a year has passed. Add 1 to every student's age using the said pointer using a function.

Exercise 11 (Number Theory):

I/
A palindrome number is a number that can be read from left to right and right to left. Ex: $n = 123$, rev $= 321$, so 123 is not a palindrome. $n = 1001$, rev $= 1001$, so n is a palindrome. $n = -100$, rev $= 001$-, so n is not a palindrome.

- Create a function that reverses an integer.
- Create a function that checks if a number is a palindrome or not.
- In the main function, read an integer $n < 1000$, and check if it's a palindrome or not, and display the result.

II/
An Armstrong (narcissistic) number is a number that is equal to the sum of the digits powered to the number of digits. Ex: $153 = 1^3 + 5^3 + 3^3$. Negative numbers are not Armstrong numbers. `<math.h>` is included.

- Create a function that calculates the number of digits using any method you know.
- Create a function that checks if a number is an Armstrong number or not.
- In the main function, read an integer $n < 1000$, and check if it's an Armstrong number or not, then display the result.

III/
Mersenne prime: prime of the form $2^p - 1$ where p is prime (e.g., $31 = 2^5 - 1$).

- Create a function that checks if a number is prime.
- Create a function that checks if that prime is a Mersenne prime, and return the number p.
- In the main function, read an integer n and check if the number is prime, then check if it's a Mersenne prime and display the integer p.