USE DD/MM/YYYY INPUT AND PRINT NEXT DATE

Below is a clear, beginner-friendly explanation of the flow of the program, followed by a brief explanation of the code so you can easily prepare a document.

✅ FLOW (How the Program Works — Step by Step)

1. **User inputs a date**
   You provide:
   - Day (DD)
   - Month (MM)
   - Year (YYYY)

   Example: `28 / 02 / 2024`

2. **The program creates an array of days in each month**
   A list of days for all 12 months is stored inside the function:

   ```
   [31,28,31,30,31,30,31,31,30,31,30,31]
   ```

3. **It checks if the given year is a leap year**
   If it is a leap year:
   - February gets **29 days** instead of 28.

4. **The program increases the day by 1**
   Example:
   28 becomes 29.

5. **It checks whether the new day is valid**
   If the day becomes greater than the number of days in that month:
   - Day resets to 1
   - Month increases by 1
6. **It checks if the month became 13**
   If month goes beyond 12:
   - Month becomes 1
   - Year increases by 1
     (because it’ s a new year)
7. **The function returns the next date**
   Output is in **(day, month, year)** format.

✅ BRIEF EXPLANATION OF THE CODE (Line-by-Line but Simple)

Here is the same code with a simple explanation:

✔ `pragma solidity ^0.8.0;`

Tells Remix to use Solidity version 0.8 or above.

✔ `contract NextDate { ... }`

Defines a contract named **NextDate**.

✔ `function getNextDate(uint day, uint month, uint year) public pure returns (...)`

A function that:

- Takes `day`, `month`, `year` as inputs

- Returns the next date
- `pure` means it does not read/write blockchain storage

✓ Days of each month

```
uint[12] memory daysInMonth = [
    uint(31), uint(28), uint(31), uint(30),
    uint(31), uint(30), uint(31), uint(31),
    uint(30), uint(31), uint(30), uint(31)
];
```

Stores the number of days in each month in an array.

✓ Leap year check

```
if ((year % 400 == 0) || (year % 4 == 0 && year % 100 != 0)) {
    daysInMonth[1] = 29;
}
```

If the year is a leap year:

- February (index 1) becomes 29 days.

✓ Increase day

Adds 1 to the day.

✓ Handle month overflow

```
if (day > daysInMonth[month - 1]) {
    day = 1;
    month++;
```

If the new day is more than the allowed days in that month:

- Reset day to **1**

- Move to the next month

✓ Handle year overflow

```
if (month > 12) {
    month = 1;
    year++;
}
```

If month becomes 13:

- Reset to January
- Increase year

✓ Return the next date

```
return (day, month, year);
```

Returns all three values.

Solidity Code for NextDate Contract

```
 // SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

contract NextDate {

    function getNextDate(uint day, uint month, uint year) public pure returns
(uint, uint, uint) {

        // Days in each month (uint required for all)
        uint[12] memory daysInMonth = [
            uint(31), uint(28), uint(31), uint(30),
            uint(31), uint(30), uint(31), uint(31),
            uint(30), uint(31), uint(30), uint(31)
        ];

        // Leap year check → February = 29 days
        if ((year % 400 == 0) || (year % 4 == 0 && year % 100 != 0)) {
            daysInMonth[1] = 29;
        }

        // Increase day
        day++;

        // If day exceeds the month limit
        if (day > daysInMonth[month - 1]) {
            day = 1;
            month++;

            // If month exceeds 12 → new year
            if (month > 12) {
                month = 1;
                year++;
            }
        }

        return (day, month, year);
    }
}
```