Lab Practice Problem 7

Filename: `practice7_surname.c`

In this activity, you will implement a program that determines whether a given 3×3 grid of positive integers forms a *magic square*. Since the grid size is 3×3, the values are the unique integers from 1 to 9. A magic square is a square grid in which the sums of all rows, columns, and both diagonals are equal. The following examples illustrate one that forms a magic square and another that does not.

8	1	6
3	5	7
4	9	2

Magic Square

1	2	3
4	5	6
7	8	9

Not Magic

Try It Out

Write a program that accepts a list of numbers representing a 3×3 grid and determines whether it forms a magic square. The following function prototypes are provided that define the **interfaces** you need to implement:

`int get_magic_constant(int n);`
Computes the expected sum (magic constant) for a square of size `n` using the formula $n(n^2+1)/2$.

`int is_magic_square(int grid[GRID_SIZE][GRID_SIZE], int size, int target);`
Calls all validation functions and returns `1` if all checks pass, or `0` otherwise.

`int validate_rows(int grid[GRID_SIZE][GRID_SIZE], int size, int target);`
Verifies that the sum of each row matches the given target.

`int validate_cols(int grid[GRID_SIZE][GRID_SIZE], int size, int target);`
Verifies that the sum of each column matches the given target.

`int validate_diag1(int grid[GRID_SIZE][GRID_SIZE], int size, int target);`
Verifies that the sum of the main diagonal (top-left to bottom-right) matches the given target.

`int validate_diag2(int grid[GRID_SIZE][GRID_SIZE], int size, int target);`
Verifies that the sum of the other diagonal (top-right to bottom-left) matches the given target.

In the `main()` function, use a loop to process all test cases and display the results.

Input

The first line of input contains an integer $0 \leq \mathsf{T} \leq 100$, the number of test cases. Each test case consists of nine unique positive integers ranging from 1 to 9. These values represent the elements of a 3×3 grid, filled in row by row. The first three numbers correspond to the first row, the next three to the second row, and the last three to the third row.

Output

For each test case, the program should print one line of output. The line begins with the test case number, followed by a colon and a space. Then, print `Magic` if the grid forms a magic square, otherwise print `Not Magic`.

Sample Input

```
2
8 1 6
3 5 7
4 9 2
1 2 3
4 5 6
7 8 9
```

Sample Output

```
1: Magic
2: Not Magic
```

Guide Questions

1. If the program were to be extended to handle a 4×4 or larger grid, what specific changes would need to be made to the implementation (e.g., in constants, loops, or function parameters)?
2. For a 4×4 magic square, what would the list of numbers be (i.e., the range of unique values used to fill the grid)? What is the corresponding magic constant for that grid size?
3. What would the function look like if the two diagonal validation functions were combined into a single function? Do you think this approach would be a good idea? Why or why not?
4. When you implemented the functions that check the diagonals, did you hard-code the computation of their sums? If so, do you think your solution would still work correctly for a grid of a different size? How could you modify your implementation to make it more generic and adaptable to grids of any dimension?

Sample Solution

The source code can be accessed here.