Write a function that prints the sum of the two diagonals of a square matrix of integers.

Prototype: void print_diagsums(int *a, int size);
Format: see example
You are allowed to use the standard library
Note that in the following example we are casting an int[][] into an int*. This is not something you should do. The goal here is to make sure you understand how an array of array is stored in memory.

```
julien@ubuntu:~/0x07$ cat 8-main.c
#include "main.h"
#include <stdio.h>

/**
* main - check the code
*
* Return: Always 0.
*/
int main(void)
{
int c3[3][3] = {
{0, 1, 5},
{10, 11, 12},
{1000, 101, 102},
};
int c5[5][5] = {
{0, 1, 5, 12124, 1234},
{10, 11, 12, 123521, 12512},
{1000, 101, 102, 12545, 214543435},
{100, 1012451, 11102, 12545, 214543435},
{10, 12401, 10452, 11542545, 1214543435},
};
print_diagsums((int *)c3, 3);
print_diagsums((int *)c5, 5);
return (0);
}
julien@ubuntu:~/0x07$ gcc -Wall -pedantic -Werror -Wextra -std=gnu89
8-main.c 8-print_diagsums.c -o 8-print_diagsums
julien@ubuntu:~/0x07$ ./8-print_diagsums
113, 1016
1214556093, 1137318
```

solution

```
void print_diagsums(int *a, int size)
```

```
{
int i;
int sum_main = 0;
int sum_secondary = 0;

for (i = 0; i < size; i++)
{
sum_main += a[i * size + i];
sum_secondary += a[i * size + (size - 1 - i)];
}

printf("%d, %d\n", sum_main, sum_secondary);
}
```

explain the task and break down the solution

Let's break down the task and the provided solution step by step:

The Task:

The objective is to print the sum of the two diagonals of a square matrix of integers. The matrix is passed as an argument in the form of a one-dimensional integer pointer (despite its two-dimensional nature).

The two diagonals of a square matrix are:

1. Main diagonal: Starts from the top-left and ends at the bottom-right.
2. Secondary diagonal: Starts from the top-right and ends at the bottom-left.

The Prototype:

The function `print_diagsums` accepts two arguments:

- `int *a`: A pointer to the first element of the matrix. As given in the example, a two-dimensional matrix is type-cast to a one-dimensional pointer, which means we have to manually perform the two-dimensional indexing.
- `int size`: The size of one dimension of the square matrix (since it's a square, both dimensions are the same).

The Solution:

1. Initialize two variables `sum_main` and `sum_secondary` to store the sums of the main and secondary diagonals, respectively.
2. Traverse through the matrix:
    3. The loop runs from `0` to `size-1`, with `i` being the current index.
    4. To compute the main diagonal sum:
        5. The expression `i * size + i` provides the linear index corresponding to `matrix[i][i]` in a two-dimensional representation.
    6. To compute the secondary diagonal sum:
        7. The expression `i * size + (size - 1 - i)` provides the linear index corresponding to `matrix[i][size-1-i]` in a two-dimensional representation.
8. Use the `printf` function to print the sums in the required format.

Explanation of the Solution:

1. Variables Declaration:

```
int i;
int sum_main = 0;
int sum_secondary = 0;
```

2. Loop to calculate the sums:

```
for (i = 0; i < size; i++)
{
    sum_main += a[i * size + i];
    sum_secondary += a[i * size + (size - 1 - i)];
}
```

Here's how the indexing works in this loop:

3. For a 3x3 matrix:
   4. Main diagonal: `(0,0)`, `(1,1)`, `(2,2)`
   5. Secondary diagonal: `(0,2)`, `(1,1)`, `(2,0)`

This converts into linear indexes for a one-dimensional representation as:

6. Main diagonal: `0`, `4`, `8`
7. Secondary diagonal: `2`, `4`, `6`

8. Print the Results:

```
printf("%d, %d\n", sum_main, sum_secondary);
```

When the function is executed with the provided sample matrices, it computes the sums of the diagonals and prints them in the specified format.