2022-10-07

1. ARRAY MANIPULATION
2. PASSING ARRAY TO FUNCTION
3. PRACTICE TEST

Presenter: Nguyen Khoa

1.1. ARRAY INITIALIZATION

```c
/**
 * Create a new 1-dimensional array with the given size
 * @param[in] _size the size of the array
 * @param[out] _ empty 1-dimensional array filled with 0
 */
int *i_arrayNew_1d(int _size)
{
    return (int *)calloc(_size, sizeof(int));
}
```

Note:
MUST HAVE
`#include <stdlib.h>`

```c
/**
 * Create a new 1-dimensional array with the given size filled values within range [min, max]
 * @param[in] _size the size of the array
 * @param[in] min the minimum value of the array
 * @param[in] max the maximum value of the array
 * @param[out] _ 1-dimensional array
 */
int *i_arrayNewRandom_1d(int _size, int min, int max)
{
    assert(min <= max);
    int *arr = i_arrayNew_1d(_size);
    for (int i = 0; i < _size; i++)
    {
        arr[i] = rand() % (max - min + 1) + min;
    }
    return arr;
}
```

Note:
MUST HAVE
`#include <assert.h>`

Note:
May need
`#include <assert.h>`
`srand(time(NULL));`

```c
#include <time.h>
#include <stdio.h>
#include <assert.h>
#include <stdlib.h>
```

```c
// set random seed
srand(time(NULL));

// initialization 1d
int size = 10;
int *arr = i_arrayNewRandom_1d(size, 10, 100);
```

1.1. ARRAY INITIALIZATION

```c
/**
 * Create a new 2-dimensional array with the given size filled with values within range [min, max]
 * @param[in] row number of rows
 * @param[in] col number of columns
 * @param[in] min minimum value of the array
 * @param[in] max maximum value of the array
 * @param[out] _ 2-dimensional array
 */
int **i_arrayNewRandom_2d(int row, int col, int min, int max)
{
    assert(min <= max);
    int **arr = i_arrayNew_2d(row, col);
    for (int i = 0; i < row; i++)
    {
        for (int j = 0; j < col; j++)
        {
            arr[i][j] = rand() % (max - min + 1) + min;
        }
    }
    return arr;
}
```

Note:
MUST HAVE
#include
<assert.h>

Note:
May need
#include
<assert.h>
srand(time(NULL));

Note:
MUST HAVE
#include
<stdlib.h>

```c
/**
 * Create a new 2-dimensional array with the given size
 * @param[in] row number of rows
 * @param[in] col number of columns
 * @param[out] _ empty 2-dimensional array filled with 0
 */
int **i_arrayNew_2d(int row, int col)
{
    int **matrix = (int **)calloc(row, sizeof(int *));
    for (int i = 0; i < row; i++)
    {
        matrix[i] = (int *)calloc(col, sizeof(int));
    }
    return matrix;
}
```

1.2. Accessing Elements of an Array

```
printf("i_arrayPrint_subscritable1d\n");
for (int i = 0; i < _size; i++)
{
    printf("%d ", arr[i]);
}
puts("\n");
```

```
printf("i_arrayPrint_pointer1d\n");
for (int i = 0; i < _size; i++)
{
    printf("%d ", *(arr + i));
}
puts("\n");
```

```
i_arrayPrint_subscritable1d
95 10 55 66 66 65 69 23 57 68

i_arrayPrint_pointer1d
95 10 55 66 66 65 69 23 57 68
```

1.2. ACCESSING ELEMENTS OF AN ARRAY

```c
printf("i_arrayPrint_subscritable2d\n");
for (int i = 0; i < row; i++)
{
    for (int j = 0; j < col; j++)
    {
        printf("%d ", arr[i][j]);
    }
    puts("");
}
puts("");
```

```c
printf("i_arrayPrint_pointer2d\n");
for (int i = 0; i < row; i++)
{
    for (int j = 0; j < col; j++)
    {
        printf("%d ", *(*(arr + i) + j));
    }
    puts("");
}
puts("");
```

```c
printf("i_arrayPrint_2das1d\n");
for (int i = 0; i < row; i++)
{
    for (int j = 0; j < col; j++)
    {
        printf("%d ", array[i * col + j]);
    }
    printf("\n");
}
```

1.2. Accessing Elements of an Array

```
i_arrayPrint_subscritable2d
24 70 68 99 13
20 67 10 74 43
41 57 10 33 39
45 26 27 55 20
79 94 70 59 49
```

```
i_arrayPrint_pointer2d
24 70 68 99 13
20 67 10 74 43
41 57 10 33 39
45 26 27 55 20
79 94 70 59 49
```

```
i_arrayPrint_2das1d
24 70 68 99 13
20 67 10 74 43
41 57 10 33 39
45 26 27 55 20
79 94 70 59 49
```

2.1. PASSING 1-DIMETIONAL ARRAY TO FUNCTION

```c
#include <stdio.h>

#define MAX 5

void printArray(int arr[MAX])
{
    for (int i = 0; i < MAX; i++)
    {
        printf("%d ", arr[i]);
    }
}

int main()
{
    int arr[] = {1, 2, 3, 4, 5};
    printArray(arr);
}
```

```c
#include <stdio.h>

#define MAX 5

void printArray(int arr[MAX])
{
    for (int i = 0; i < MAX; i++)
    {
        printf("%d ", *(arr + i));
    }
}

int main()
{
    int arr[] = {1, 10, 3, 9, 5};
    printArray(arr);
}
```

2.1. PASSING 1-DIMETIONAL ARRAY TO FUNCTION

```c
#include <stdio.h>

void printArray(int arr[], int _size)
{
    for (int i = 0; i < _size; i++)
    {
        printf("%d ", arr[i]);
    }
}

int main()
{
    int arr[] = {1, 10, 3, 9, 5};
    printArray(arr, sizeof(arr)/sizeof(arr[0]));
}
```

```c
#include <stdio.h>

void printArray(int *arr, int _size)
{
    for (int i = 0; i < _size; i++)
    {
        printf("%d ", arr[i]);
    }
}

int main()
{
    int arr[] = {1, 10, 3, 9, 5};
    printArray(arr, sizeof(arr)/sizeof(arr[0]));
}
```

2.2. PASSING 2-DIMETIONAL ARRAY TO FUNCTION

```c
#include <stdio.h>

#define M 3
#define N 3

void print(int arr[M][N])
{
    for (int i = 0; i < M; i++)
    {
        for (int j = 0; j < N; j++)
        {
            printf("%d ", arr[i][j]);
        }
        puts("");
    }
}

int main()
{
    int arr[][3] = {{1, 2, 3}, {4, 5, 6}, {7, 8, 9}};
    print(arr);
    return 0;
}
```

```c
#include <stdio.h>

#define M 3
#define N 3

void print(int arr[M][N])
{
    for (int i = 0; i < M; i++)
    {
        for (int j = 0; j < N; j++)
        {
            printf("%d ", *(*(arr + i) + j));
        }
        puts("");
    }
}

int main()
{
    int arr[][3] = {{1, 2, 3}, {4, 5, 6}, {7, 8, 9}};
    print(arr);
}
```

2.2. PASSING 2-DIMETIONAL ARRAY TO FUNCTION

```c
#include <stdio.h>

#define M 3
#define N 3

void print(int arr[M][N])
{
    for (int i = 0; i < M; i++)
    {
        for (int j = 0; j < N; j++)
        {
            printf("%d ", arr[i][j]);
        }
        puts("");
    }
}

int main()
{
    int arr[][3] = {{1, 2, 3}, {4, 5, 6}, {7, 8, 9}};
    print(arr);
    return 0;
}
```

```c
#include <stdio.h>

#define M 3
#define N 3

void print(int arr[M][N])
{
    for (int i = 0; i < M; i++)
    {
        for (int j = 0; j < N; j++)
        {
            printf("%d ", *(*(arr + i) + j));
        }
        puts("");
    }
}

int main()
{
    int arr[][3] = {{1, 2, 3}, {4, 5, 6}, {7, 8, 9}};
    print(arr);
}
```

2.2. PASSING 2-DIMETIONAL ARRAY TO FUNCTION

```c
#include <stdio.h>

#define N 3

void print(int arr[][N], int m)
{
    for (int i = 0; i < m; i++)
    {
        for (int j = 0; j < N; j++)
        {
            printf("%d ", arr[i][j]);
        }
        puts("");
    }
}

int main()
{
    int arr[][3] = {{1, 2, 3}, {4, 5, 6}, {7, 8, 9}};
    print(arr, 3);
    return 0;
}
```

```c
#include <stdio.h>

// n must be passed before the 2D array
void print(int m, int n, int arr[][n])
{
    for (int i = 0; i < m; i++)
    {
        for (int j = 0; j < n; j++)
        {
            printf("%d ", arr[i][j]);
        }
        puts("");
    }
}

int main()
{
    int arr[][3] = {{1, 2, 3}, {4, 5, 6}, {7, 8, 9}};
    print(3, 3, arr);
}
```

2.2. PASSING 2-DIMETIONAL ARRAY TO FUNCTION

```c
#include <stdio.h>

void print(int arr[][n], int m, int n)
{
    for (int i = 0; i < m; i++)
    {
        for (int j = 0; j < n; j++)
        {
            printf("%d ", arr[i][j]);
        }
        puts("");
    }
}
```

```c
int main()
{
    int arr[][3] = {{1, 2, 3}, {4, 5, 6}, {7, 8, 9}};
    print(arr, 3, 3);
}
```

```
.\passing_array2d_4.c:3:22: error: 'n' undeclared here (not in a function)
 void print(int arr[][n], int m, int n)
                      ^
.\passing_array2d_4.c: In function 'main':
.\passing_array2d_4.c:18:11: error: type of formal parameter 1 is incomplete
     print(arr, 3, 3);
           ^~~
```

2.2. PASSING 2-DIMETIONAL ARRAY TO FUNCTION

```c
#include <stdio.h>

void printarray(void *array, int row, int col)
{
    int *charArray = (int *)array;

    for (int i = 0; i < row; i++)
    {
        for (int j = 0; j < col; j++)
        {
            printf("%d ", charArray[i * col + j]);
        }
        printf("\n");
    }
}

int main()
{
    int array[2][3] = {{1, 2, 5}, {3, 4, 6}};
    printarray(array, 2, 3);
}
```

2.2. PASSING 2-DIMETIONAL ARRAY TO FUNCTION

```c
#include <stdio.h>
#include <stdlib.h>

int **i_arrayNew_2d(int row, int col)
{
    int **matrix = (int **)calloc(row, sizeof(int *));
    for (int i = 0; i < row; i++)
    {
        matrix[i] = (int *)calloc(col, sizeof(int));
    }
    return matrix;
}

int **i_arrayConvert_1To2(int *arr, int row, int col)
{
    int **matrix = i_arrayNew_2d(row, col);
    for (int i = 0; i < row; i++)
    {
        for (int j = 0; j < col; j++)
        {
            matrix[i][j] = arr[i * col + j];
        }
    }
    return matrix;
}
```

```c
void printarray(void *array, int row, int col)
{
    int *intArray = array;
    int **array_2d = i_arrayConvert_1To2(intArray, row, col);

    for (int i = 0; i < row; i++)
    {
        for (int j = 0; j < col; j++)
        {
            printf("%d ", array_2d[i][j]);
        }
        printf("\n");
    }
}

int main()
{
    int array[2][3] = {{1, 2, 5}, {3, 4, 6}};
    int row = 2;
    int col = 3;
    printarray(array, row, col);
}
```

2.2. PASSING 2-DIMETIONAL ARRAY TO FUNCTION

```c
#include <stdio.h>
#include <stdlib.h>

int **i_arrayNew_2d(int row, int col)
{
    int **matrix = (int **)calloc(row, sizeof(int *));
    for (int i = 0; i < row; i++)
    {
        matrix[i] = (int *)calloc(col, sizeof(int));
    }
    return matrix;
}

int **i_arrayConvert_1To2(int *arr, int row, int col)
{
    int **matrix = i_arrayNew_2d(row, col);
    for (int i = 0; i < row; i++)
    {
        for (int j = 0; j < col; j++)
        {
            matrix[i][j] = arr[i * col + j];
        }
    }
    return matrix;
}
```

```c
void printarray(void *array, int row, int col)
{
    int *intArray = array;
    int **array_2d = i_arrayConvert_1To2(intArray, row, col);

    for (int i = 0; i < row; i++)
    {
        for (int j = 0; j < col; j++)
        {
            printf("%d ", array_2d[i][j]);
        }
        printf("\n");
    }
}

int main()
{
    int array[2][3] = {{1, 2, 5}, {3, 4, 6}};
    int row = 2;
    int col = 3;
    printarray(array, row, col);
}
```

2.2. PASSING 2-DIMETIONAL ARRAY TO FUNCTION

```c
#include <stdio.h>
#include <stdlib.h>

int **i_arrayNew_2d(int row, int col)
{
    int **matrix = (int **)calloc(row, sizeof(int *));
    for (int i = 0; i < row; i++)
    {
        matrix[i] = (int *)calloc(col, sizeof(int));
    }
    return matrix;
}

void i_arrayPrint_2d(int **matrix, int row, int col)
{
    for (int i = 0; i < row; i++)
    {
        printf("\t");
        for (int j = 0; j < col; j++)
        {
            printf("%d\t", matrix[i][j]);
        }
        printf("\n");
    }
}
```

```c
int main()
{
    int row = 4;
    int col = 3;
    int **arr = i_arrayNew_2d(row, col);
    for (int i = 0; i < col; i++)
    {
        for (int j = 0; j < row; j++)
        {
            arr[j][i] = i * row + j + 1;
        }
    }
    i_arrayPrint_2d(arr, row, col);
}
```

```
1       5       9
2       6       10
3       7       11
4       8       12
```

2.2. PASSING 2-DIMETIONAL ARRAY TO FUNCTION

```c
#include <stdio.h>
#include <stdlib.h>

int **i_arrayNew_2d(int row, int col)
{
    int **matrix = (int **)calloc(row, sizeof(int *));
    for (int i = 0; i < row; i++)
    {
        matrix[i] = (int *)calloc(col, sizeof(int));
    }
    return matrix;
}
```

```c
void i_arrayPrint_2d(int **matrix, int row, int col)
{
    for (int i = 0; i < row; i++)
    {
        printf("\t");
        for (int j = 0; j < col; j++)
        {
            printf("%d\t", matrix[i][j]);
        }
        printf("\n");
    }
}
```

```c
int main()
{
    int arr[2][3] = {{1, 2, 5}, {3, 4, 6}};
    int row = 2;
    int col = 3;

    for (int i = 0; i < col; i++)
    {
        for (int j = 0; j < row; j++)
        {
            arr[j][i] = i * row + j + 1;
        }
    }
    i_arrayPrint_2d(arr, row, col);
}
```

```
.\passing_array2d_7.c: In function 'main':
.\passing_array2d_7.c:40:21: warning: passing argument 1 of 'i_arrayPrint_2d' from incompatible pointer type [-Wincompatible-pointer-types]
    i_arrayPrint_2d(arr, row, col);
                    ^~~
.\passing_array2d_7.c:14:6: note: expected 'int **' but argument is of type 'int (*)[3]'
 void i_arrayPrint_2d(int **matrix, int row, int col)
      ^~~~~~~~~~~~~~~
```

SUMMARY

1-Dimensional Array Allocation

```
int *arr = (int *)calloc(_size, sizeof(int));
```

Accessing Element 1-Dimensional Array

```
arr[i]
```

```
*(arr + i)
```

Accessing Element 2-Dimensional Array

```
arr[i][j]
```

```
arr[i * col + j]
```

```
*(*(arr + i) + j)
```

2-Dimensional Array Allocation

```
int **matrix = (int **)calloc(row, sizeof(int *));
for (int i = 0; i < row; i++)
{
    matrix[i] = (int *)calloc(col, sizeof(int));
}
return matrix;
```

Other

1. int matrix[n][m] is not a double pointer
2. When passing an array to function, the thing that is actually passed is the address of the first element of the array (aka pointer)

SUMMARY

Passing a 1-dimentional array to function

```c
#define MAX 5

void printArray(int arr[MAX])
```

```c
void printArray(int arr[], int _size)
```

```c
void printArray(int *arr, int _size)
```

Passing a 2-dimentional array to function

```c
int **arr
```

```c
void i_arrayPrint_2d(int **matrix, int row, int col)
```

```c
int array[2][3]
```

```c
void printarray(void *array, int row, int col)
{
    int *charArray = (int *)array;
```

```c
int array[2][3]
```

```c
// n must be passed before the 2D array
void print(int m, int n, int arr[][n])
```

3. PRACTICE TEST 1.1.

$$f(x) = \frac{x}{2} * (1 + x + 0.04 * x^3)$$

Example 1: $f(0.5) \approx 0.376250$

Example 2: $f(1.5) \approx 1.976250$

```
Function(x=0.500000) = 0.376250
Function(x=1.500000) = 1.976250
```

```c
#include <stdio.h>
#include <math.h>

double function(double x)
{
    return x / 2 * (1 + x + 0.04 * pow(x, 3));
}

int main()
{
    float x = 0.5;
    printf("Function(x=%f) = %f\n", x, function(x));
    x = 1.5;
    printf("Function(x=%f) = %f", x, function(x));
}
```

3. PRACTICE TEST 1.2.

```
// format of for loop
for (Initilization; Condition; Update)
```

$$\ln(0.5) \approx \sum_{i=1}^{n} -\frac{0.5^i}{i}$$

Example 1: $n = 10$, $result \approx -0.693065$

Example 2: $n = 100 \approx -0.693147$

```
ln(x=0.500000, loop=10) = -0.693065
ln(x=0.500000, loop=100) = -0.693147
```

```c
#include <stdio.h>
#include <math.h>

double ln(double x, int loop)
{
    double result = 0;
    for (int i = 1; i <= loop; i++)
    {
        result -= pow(x, i) / i;
    }
    return result;
}

int main()
{
    float x = 0.5;
    int loop = 10;
    printf("ln(x=%f, loop=%d) = %f\n", x, loop, ln(x, loop));
    loop = 100;
    printf("ln(x=%f, loop=%d) = %f", x, loop, ln(x, loop));
}
```

3. PRACTICE TEST 2.1.

Implement the function which receive **min_value**, **max_value** and an **array**, then modifying on this array to ensure that all elements in the array cannot be out of the range (**min_value** $\leq$ element $\leq$ **max_value**). If an element less than the min_value, you should replace it with **min_value**. Do the same for **max_value**.

Example 1:

input: min_value = 3, max_value=6, **x** = {**1**, **2**, 3, 4, 5, 6, **7**, **8**, **9**}

result: **x** = {3, 3, 3, 4, 5, 6, 6, 6, 6}

Example 2:

input: min_value = 11, max_value=33, **x** = {**4**, 12, 23, **4**, **35**, 16, **7**, **48**, 19}

result: **x** = {11, 12, 23, 11, 33, 16, 11, 33, 19}

3. PRACTICE TEST 2.1.

```c
int *i_arrayClip_1d(int *arr, int _size, int min_value, int max_value)
{
    int *result = i_arrayNew_1d(_size);
    for (int i = 0; i < _size; i++)
    {
        if (arr[i] < min_value)
        {
            result[i] = min_value;
        }

        else if (arr[i] > max_value)
        {
            result[i] = max_value;
        }

        else
        {
            result[i] = arr[i];
        }
    }
    return result;
}
```

3. PRACTICE TEST 2.1.

```c
void i_arrayClip_1d_inplace(int *arr, int _size, int min_value, int max_value)
{
    for (int i = 0; i < _size; i++)
    {
        if (arr[i] < min_value)
        {
            arr[i] = min_value;
        }

        else if (arr[i] > max_value)
        {
            arr[i] = max_value;
        }
    }
}
```

3. PRACTICE TEST 2.2.

Implement the function which receive **input_array**, **diff_array**, and **sign_array**. Use two pointers, one for **diff_array** and the other for **sign_array** to modify them. Elements in **diff_array** are results of difference between two consecutive element in the **input_array** (right element - left element). Elements in sign_array indicate that the difference is negative (< 0) = -1, or positive (≥ 0) = 1.

Example 1:

input: **input_array** = {1, 3, 7, 1, 2, 6, 0, 1}

result:

diff_array = {2, 4, -6, 1, 4, -6, 1}

sign_array = {1, 1, -1, 1, 1, -1, 1}

Example 2:

input: **input_array** = {11, 23, 7, 10, 21, 16, 40, 17}

result:

diff_array = {12, -16, 3, 11, -5, 24, -23}

sign_array = {1, -1, 1, 1, -1, 1, -1}

3. PRACTICE TEST 2.2.

```c
int *i_arrayDiff_1d(int *arr, int _size)
{
    int *result = i_arrayNew_1d(_size - 1);
    for (int i = 0; i < _size - 1; i++)
    {
        result[i] = arr[i + 1] - arr[i];
    }
    return result;
}
```

```c
int *i_arraySign_1d(int *arr, int _size)
{
    int *result = i_arrayNew_1d(_size);
    for (int i = 0; i < _size; i++)
    {
        if (arr[i] == 0)
        {
            result[i] = 0;
        }
        else
        {
            result[i] = (int)(arr[i] / abs(arr[i]));
        }
    }
    return result;
}
```

```c
int main()
{
    int arr[] = {1, 3, 7, 1, 2, 6, 0, 1};
    int length = sizeof(arr) / sizeof(arr[0]);
    i_arrayPrintCustom_1d(arr, length, "Original:\t", "", " \t");
    int *diff = i_arrayDiff_1d(arr, length);
    i_arrayPrintCustom_1d(diff, length - 1, "Diff:\t\t", "   ", " \t");
    int *sign = i_arraySign_1d(diff, length - 1);
    i_arrayPrintCustom_1d(sign, length - 1, "Sign:\t\t", "   ", " \t");
}
```

```
Original:       1       3       7       1       2       6       0       1
Diff:             2       4       -6      1       4       -6      1
Sign:             1       1       -1      1       1       -1      1
```

3. PRACTICE TEST 3.1.

Write a function to compute sum for odd elements in each row

Example 1:

$\mathbf{X} = \begin{bmatrix} \mathbf{1} & 4 & \mathbf{7} \\ 2 & \mathbf{5} & 8 \\ \mathbf{3} & 6 & \mathbf{9} \end{bmatrix}$ result is an array which has number of elements in array equal to number of rows: res = {8, 5, 12}

Example 2:

$\mathbf{X} = \begin{bmatrix} \mathbf{1} & 2 & \mathbf{3} \\ 4 & \mathbf{5} & 6 \\ \mathbf{7} & 8 & \mathbf{9} \\ 10 & \mathbf{11} & 12 \end{bmatrix}$ result is an array which has number of elements in array equal to number of rows: res = {4, 5, 16, 11}

3. PRACTICE TEST 3.1.

```c
int *i_arrayAddOddRowwise_2d(int **arr, int row, int col)
{
    int *result = i_arrayNew_1d(row);
    for (int i = 0; i < row; i++)
    {
        for (int j = 0; j < col; j++)
        {
            if (arr[i][j] % 2 == 1)
            {
                result[i] += arr[i][j];
            }
        }
    }
    return result;
}
```

```
The matrix:
        1       2       3
        4       5       6
        7       8       9
        10      11      12

The row-wise sum of odd elements:       4       5       16      11
```

```c
int main()
{
    int row = 4;
    int col = 3;
    int **arr = i_arrayNew_2d(row, col);
    for (int i = 0; i < row; i++)
    {
        for (int j = 0; j < col; j++)
        {
            arr[i][j] = i * col + j + 1;
        }
    }

    printf("The matrix:\n");
    i_arrayPrint_2d(arr, row, col);

    int *result = i_arrayAddOddRowwise_2d(arr, row, col);
    printf("\nThe row-wise sum of odd elements:\t");
    i_arrayPrint_1d(result, row);
}
```

3. PRACTICE TEST 3.2.

Write a function to compute sum of squared diagonal (column index = row index) in the matrix

Example 1:

$$\mathbf{X} = \begin{bmatrix} \mathbf{1} & 4 & 7 \\ 2 & \mathbf{5} & 8 \\ 3 & 6 & \mathbf{9} \end{bmatrix} \text{ result} = 1^2 + 5^2 + 9^2 = 107$$

Example 2:

$$\mathbf{X} = \begin{bmatrix} \mathbf{1} & 2 & 3 \\ 4 & \mathbf{5} & 6 \\ 7 & 8 & \mathbf{9} \\ 10 & 11 & 12 \end{bmatrix} \text{ result} = 1^2 + 5^2 + 9^2 = 107$$

3. PRACTICE TEST 3.2.

```c
int diagonalSquareSum(int **matrix, int row, int col)
{
    int sum = 0;
    for (int i = 0; i < row; i++)
    {
        for (int j = 0; j < col; j++)
        {
            if (i==j)
            {
                sum += matrix[i][j] * matrix[i][j];
            }
        }
    }
    return sum;
}
```

```c
int main()
{
    int row = 3;
    int col = 3;
    int **arr = i_arrayNew_2d(row, col);
    for (int i = 0; i < col; i++)
    {
        for (int j = 0; j < row; j++)
        {
            arr[j][i] = i * row + j + 1;
        }
    }

    printf("The matrix:\n");
    i_arrayPrint_2d(arr, row, col);

    printf("Sum of square diagonal = %d\n", diagonalSquareSum(arr, row, col));
}
```

```
The matrix:
        1       4       7
        2       5       8
        3       6       9
Sum of square diagonal = 107
```

3. PRACTICE TEST 4.1.

Write a function to compute sum for **even index** in each row

Example 1:

$\mathbf{X} = \begin{bmatrix} \mathbf{1} & 4 & \mathbf{7} \\ \mathbf{2} & 5 & \mathbf{8} \\ \mathbf{3} & 6 & \mathbf{9} \end{bmatrix}$ result is an array which has number of elements in array equal to number of rows: res = {8, 10, 12}

Example 2:

$\mathbf{X} = \begin{bmatrix} \mathbf{1} & 2 & \mathbf{3} \\ \mathbf{4} & 5 & \mathbf{6} \\ \mathbf{7} & 8 & \mathbf{9} \\ \mathbf{10} & 11 & \mathbf{12} \end{bmatrix}$ result is an array which has number of elements in array equal to number of rows: res = {4, 10, 16, 22}

3. PRACTICE TEST 4.1.

```c
int *i_arrayAddEvenIndicesRowwise_2d(int *arr, int row, int col)
{
    int *result = i_arrayNew_1d(row);
    for (int i = 0; i < row; i++)
    {
        for (int j = 0; j < col; j++)
        {
            if (j % 2 == 0)
            {
                result[i] += arr[i * col + j];
            }
        }
    }
    return result;
}
```

```c
int main()
{
    int row = 4;
    int col = 3;
    int *arr = i_arrayNew_1d(row * col);
    for (int i = 0; i < row; i++)
    {
        for (int j = 0; j < col; j++)
        {
            arr[i * col + j] = i * col + j + 1;
        }
    }
    printf("The matrix:\n");
    i_arrayPrint_2d(arr, row, col);

    int *result = i_arrayAddEvenIndicesRowwise_2d(arr, row, col);
    printf("The result:\n");
    i_arrayPrint_1d(result, row);
}
```

```
The matrix:
        1       2       3
        4       5       6
        7       8       9
        10      11      12
The result:
4       10      16      22
```

3. PRACTICE TEST 4.2.

Write a function to compute sum of **doubled squared diagonal** (column index = row index) in the matrix

Example 1:

$$\mathbf{X} = \begin{bmatrix} \mathbf{1} & 4 & 7 \\ 2 & \mathbf{5} & 8 \\ 3 & 6 & \mathbf{9} \end{bmatrix} \quad \text{result} = 2 * 1^2 + 2 * 5^2 + 2 * 9^2 = 214$$

Example 2:

$$\mathbf{X} = \begin{bmatrix} \mathbf{1} & 2 & 3 \\ 4 & \mathbf{5} & 6 \\ 7 & 8 & \mathbf{9} \\ 10 & 11 & 12 \end{bmatrix} \quad \text{result} = 2 * 1^2 + 2 * 5^2 + 2 * 9^2 = 214$$

3. PRACTICE TEST 4.2.

```c
int diagonalDoubleSquareSum(int *matrix, int row, int col)
{
    int sum = 0;
    for (int i = 0; i < row; i++)
    {
        for (int j = 0; j < col; j++)
        {
            if (i == j)
            {
                sum += 2 * matrix[i * col + j] * matrix[i * col + j];
            }
        }
    }
    return sum;
}
```

```c
int main()
{
    int row = 3;
    int col = 3;
    int *arr = i_arrayNew_1d(row * col);
    for (int i = 0; i < col; i++)
    {
        for (int j = 0; j < row; j++)
        {
            arr[j * row + i] = i * row + j + 1;
        }
    }
    printf("The matrix:\n");
    i_arrayPrint_2d(arr, row, col);

    printf("The result: \n");
    printf("%d\n", diagonalDoubleSquareSum(arr, row, col));
}
```

```
The matrix:
        1       4       7
        2       5       8
        3       6       9
The result: 214
```

THANK YOU FOR LISTENING