// documentation for sinogaya_header.h

```
#ifndef SINOGAYA_STATS_H
#define SINOGAYA_STATS_H

// Function declarations

/**
 * Function to input an array from the user.
 * @param array Pointer to the array where the input will be stored.
 * @param size The size of the array.
 */
void inputArray(int *array, int size);

/**
 * Function to calculate the mean of an array.
 * @param array Pointer to the array of integers.
 * @param size The size of the array.
 * @return The mean of the array as a double.
 */
double calculateMean(int *array, int size);

/**
 * Function to calculate the median of an array.
 * @param array Pointer to the array of integers.
 * @param size The size of the array.
 * @return The median of the array as a double.
 */
double calculateMedian(int *array, int size);
```

```c
/**
 * Function to calculate the mode of an array.
 * @param array Pointer to the array of integers.
 * @param size The size of the array.
 * @return The mode of the array as an integer. Returns -1 if no mode is found.
 */
int calculateMode(int *array, int size);

/**
 * Function to calculate the variance of an array.
 * @param array Pointer to the array of integers.
 * @param size The size of the array.
 * @param mean The mean of the array.
 * @return The variance of the array as a double.
 */
double calculateVariance(int *array, int size, double mean);

/**
 * Function to calculate the standard deviation of an array.
 * @param variance The variance of the array.
 * @return The standard deviation of the array as a double.
 */
double calculateStandardDeviation(double variance);

#endif
```

// documentation for sinogaya_implentation.c

```c
#include <stdio.h>

#include <stdlib.h>

#include <math.h>

#include "sinogaya_stats.h"


/**

 * Function to input an array from the user.

 * @param array Pointer to the array where the input will be stored.

 * @param size The size of the array.

 */

void inputArray(int *array, int size) {

    printf("Enter %d numbers:\n", size);

    for (int i = 0; i < size; i++) {

        if (scanf("%d", &array[i]) != 1) {

            printf("Invalid input. Please enter integers only.\n");

            exit(EXIT_FAILURE);

        }

    }

}


/**

 * Function to calculate the mean of an array.

 * @param array Pointer to the array of integers.

 * @param size The size of the array.

 * @return The mean of the array as a double.

 */
```

```
double calculateMean(int *array, int size) {
    int sum = 0;
    for (int i = 0; i < size; i++) {
        sum += array[i];
    }
    return (double)sum / size;
}

/**
 * Function to calculate the median of an array.
 * @param array Pointer to the array of integers.
 * @param size The size of the array.
 * @return The median of the array as a double.
 */
double calculateMedian(int *array, int size) {
    // Sorting the array (ascending order)
    for (int i = 0; i < size - 1; i++) {
        for (int j = 0; j < size - i - 1; j++) {
            if (array[j] > array[j + 1]) {
                int temp = array[j];
                array[j] = array[j + 1];
                array[j + 1] = temp;
            }
        }
    }

    // Finding the median
    if (size % 2 == 0) {
        return (double)(array[size / 2 - 1] + array[size / 2]) / 2.0;
    } else {
        return array[size / 2];
    }
}

/**
 * Function to calculate the mode of an array.
 * @param array Pointer to the array of integers.
 * @param size The size of the array.
 * @return The mode of the array as an integer. Returns -1 if no mode is found.
 */
int calculateMode(int *array, int size) {
    int maxValue = 0, maxCount = 0, i, j;

    for (i = 0; i < size; ++i) {
        int count = 0;
        for (j = 0; j < size; ++j) {
            if (array[j] == array[i])
                ++count;
        }

        if (count > maxCount) {
            maxCount = count;
            maxValue = array[i];
        }
    }

    return maxCount > 1 ? maxValue : -1; // Return -1 if no mode found
}
```

```
/**
 * Function to calculate the variance of an array.
 * @param array Pointer to the array of integers.
 * @param size The size of the array.
 * @param mean The mean of the array.
 * @return The variance of the array as a double.
 */
double calculateVariance(int *array, int size, double mean) {
    double variance = 0;
    for (int i = 0; i < size; i++) {
        variance += pow(array[i] - mean, 2);
    }
    return variance / size;
}

/**
 * Function to calculate the standard deviation of an array.
 * @param variance The variance of the array.
 * @return The standard deviation of the array as a double.
 */
double calculateStandardDeviation(double variance) {
    return sqrt(variance);
}

// Main function
int main() {
    int size;
```

```
printf("Enter the size of the array: ");
if (scanf("%d", &size) != 1 || size <= 0) {
    printf("Invalid input. Please enter a positive integer.\n");
    return 1;
}

int *array = (int *)malloc(size * sizeof(int));
if (array == NULL) {
    printf("Memory allocation failed.\n");
    return 1;
}

// Input array elements
inputArray(array, size);

// Calculate mean
double mean = calculateMean(array, size);
printf("Mean: %.2f\n", mean);

// Calculate median
double median = calculateMedian(array, size);
printf("Median: %.2f\n", median);

// Calculate mode
int mode = calculateMode(array, size);
if (mode != -1) {
    printf("Mode: %d\n", mode);
} else {
    printf("Mode: No mode found\n");
    }

    // Calculate variance
    double variance = calculateVariance(array, size, mean);
    printf("Variance: %.2f\n", variance);

    // Calculate standard deviation
    double standardDeviation = calculateStandardDeviation(variance);
    printf("Standard Deviation: %.2f\n", standardDeviation);

    free(array);
    return 0;
}
```

//documentation for sinogaya_test.c

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "sinogaya_stats.h"

/**
 * Function to input an array from the user.
 * @param array Pointer to the array where the input will be stored.
 * @param size The size of the array.
 */
void inputArray(int *array, int size) {
    printf("Enter %d numbers:\n", size);
    for (int i = 0; i < size; i++) {
        if (scanf("%d", &array[i]) != 1) {
            printf("Invalid input. Please enter integers only.\n");
            exit(EXIT_FAILURE);
        }
    }
}

/**
 * Function to calculate the mean of an array.
 * @param array Pointer to the array of integers.
 * @param size The size of the array.
 * @return The mean of the array as a double.
 */
double calculateMean(int *array, int size) {
    int sum = 0;
    for (int i = 0; i < size; i++) {
        sum += array[i];
    }
    return (double)sum / size;
}

/**
 * Function to calculate the median of an array.
 * @param array Pointer to the array of integers.
 * @param size The size of the array.
 * @return The median of the array as a double.
 */
```

```
double calculateMedian(int *array, int size) {
    // Sorting the array (ascending order)
    for (int i = 0; i < size - 1; i++) {
        for (int j = 0; j < size - i - 1; j++) {
            if (array[j] > array[j + 1]) {
                int temp = array[j];
                array[j] = array[j + 1];
                array[j + 1] = temp;
            }
        }
    }

    // Finding the median
    if (size % 2 == 0) {
        return (double)(array[size / 2 - 1] + array[size / 2]) / 2.0;
    } else {
        return array[size / 2];
    }
}

/**
 * Function to calculate the mode of an array.
 * @param array Pointer to the array of integers.
 * @param size The size of the array.
 * @return The mode of the array as an integer. Returns -1 if no mode is found.
 */
int calculateMode(int *array, int size) {
    int maxValue = 0, maxCount = 0, i, j;
    for (i = 0; i < size; ++i) {
        int count = 0;
        for (j = 0; j < size; ++j) {
            if (array[j] == array[i])
                ++count;
        }

        if (count > maxCount) {
            maxCount = count;
            maxValue = array[i];
        }
    }

    return maxCount > 1 ? maxValue : -1; // Return -1 if no mode found
}

/**
 * Function to calculate the variance of an array.
 * @param array Pointer to the array of integers.
 * @param size The size of the array.
 * @param mean The mean of the array.
 * @return The variance of the array as a double.
 */
double calculateVariance(int *array, int size, double mean) {
    double variance = 0;
    for (int i = 0; i < size; i++) {
        variance += pow(array[i] - mean, 2);
    }
    return variance / size;
```

```c
}

/**
 * Function to calculate the standard deviation of an array.
 * @param variance The variance of the array.
 * @return The standard deviation of the array as a double.
 */
double calculateStandardDeviation(double variance) {
    return sqrt(variance);
}

// Main function
int main() {
    int size;

    printf("Enter the size of the array: ");
    if (scanf("%d", &size) != 1 || size <= 0) {
        printf("Invalid input. Please enter a positive integer.\n");
        return 1;
    }

    int *array = (int *)malloc(size * sizeof(int));
    if (array == NULL) {
        printf("Memory allocation failed.\n");
        return 1;
    }

    // Input array elements
    inputArray(array, size);
    // Calculate mean
    double mean = calculateMean(array, size);
    printf("Mean: %.2f\n", mean);

    // Calculate median
    double median = calculateMedian(array, size);
    printf("Median: %.2f\n", median);

    // Calculate mode
    int mode = calculateMode(array, size);
    if (mode != -1) {
        printf("Mode: %d\n", mode);
    } else {
        printf("Mode: No mode found\n");
    }

    // Calculate variance
    double variance = calculateVariance(array, size, mean);
    printf("Variance: %.2f\n", variance);

    // Calculate standard deviation
    double standardDeviation = calculateStandardDeviation(variance);
    printf("Standard Deviation: %.2f\n", standardDeviation);

    free(array);
    return 0;
}
```