Purpose: Calculates the first quartile (Q1), median (Q2), and third quartile (Q3) of the dataset, and determines whether Q1 and Q3 can be calculated (known) or not.
precondition: data set must have at least 2 values
postcondition: returns Q1, Q2, Q3, and if Q1 and Q3 are known or unknown
Time Complexity: O(n)
Step 1: Set requirements for function and have printed errors
Step 2: Since Q2 is the same as median(), call to that function
Step 3: Calculate Q1 (lower half median)
    -split dataset into lower half (n / 2)
    -if its an odd number, Q1 is the middle element
    -if its even, Q1 is the average of two middle elements
Step 4: Calculate Q3 (upper half median)
    -If n is even number then start with n / 2
    -If n is odd number then start with (n / 2) + 1
    -Calculate upper half size = n - upperStart
    -if upper half has fewer than 2 values then Q3 is unknown
    -If its an odd count then Q3 is the middle element
    -If its an even count then Q3 is average of two middle elements
Step 5: Return results
    -Return structure

<u>Summary of cases</u>

- n = 0 or 1 → error (not enough data).
- n = 2 or 3 → Q1 and Q3 are both unknown, because each half only has 1 value.
- n ≥ 4 → Both Q1 and Q3 are known, because each half has at least 2 values.

```
FUNCTION quartilesCalculation() RETURNS QuartileValues

    n = size of dataset

    //takes care of invalid dataset size
    IF n == 0 THEN
        PRINT "Error: Dataset is empty"
    END IF

    IF  n == 1 THEN
        PRINT "Error: Requires at least 2 data values"
    END IF

    //Q2 is the same as the median of whole data set
```

```
Q2 = median(dataset)

//start of finding Q1
//Q1 = 0
q1Known = FALSE                //this is so we can print unknown
lowerHalfSize = n / 2

IF lowerHalfSize >= 2 THEN
        q1Known = TRUE
        IF lowerHalfSize is odd THEN
                Q1 = dataset[lowerHalfSize / 2]
        ELSE
                mid1 = dataset[(lowerHalfSize / 2) - 1]
                mid2 = dataset[lowerHalfSize / 2]
                Q1 = (mid1 + mid2) / 2.0
        END IF
END IF

//start of finding Q3
Q3 = 0
q3Known = FALSE

//find where the upper half starts
IF n is even THEN
        upperStart = n / 2
ELSE
        upperStart = (n / 2) + 1
END IF

upperHalfSize = n - upperStart

IF upperHalfSize >= 2 THEN
        q3Known = TRUE
        IF upperHalfSize is odd THEN
                Q3 = dataset[upperStart + (upperHalfSize / 2)
        ELSE
                mid1 = dataset[upperStart + (upperHalfSize / 2) - 1]
                mid2 = dataset[upperStart + (upperHalfSize / 2)]
                Q3 = (mid1 + mid2) / 2.0
        END IF
```

```
END IF

//return results
RETURN (Q1, Q2, Q3, q1Known, q3Known)

END FUNCTION
```

Purpose: Displays Q1, Q2, and Q3, showing their values if calculable or “unknown” if not.
precondition: data set must have at least 2 values
postcondition: displays quartiles and if known/unknown
Time Complexity: O(n)
Step 1: Call quartilesCalculation)_
    -Get QuartileValues structure containing Q1, Q2, Q3, q1Known, q3Known
Step 2: Set up display
    -Print “Quartiles: “
Step 3: Display Q1
    -if Q1 is known (there are enough numbers in the dataset) then print Q1 value, otherwise print “unknown”
Step 4: Display Q2
    -Always print Q2 value
Step 5: Display Q3
    -if Q3 is known (there are enough numbers in the dataset) then print Q3 value, otherwise print “unknown”

```
FUNCTION quartiles() RETURNS void

    q = quartilesCalculation()      //get quartile values

    PRINT “Quartiles: “

    //display Q1
    IF q.q1Known == TRUE THEN
        PRINT "Q1 --> " + q.Q1
    ELSE
        PRINT "Q1 --> unknown"
    END IF

    //display Q2
    PRINT "Q2 --> " + q.Q2
```

```
//display Q3
IF q.q3Known == TRUE THEN
	PRINT "Q3 --> " + q.Q3
ELSE
	PRINT "Q3 --> unknown"
END IF

END FUNCTION
```

Purpose: Calculates and displays the Interquartile Range (IQR), which measures the spread of the middle 50% of the data.
precondition: data set must have at least 2 values
postcondition: displays quartiles and if known/unknown
Time Complexity: O(n)
Step 1: Set size of data to n
Step 2: Call to quartilesCalculation() to have Q1, Q2, Q3 values
Step 3: If dataset is less than 4 then print “unknown”
Step 4: Calculate and display interquartile range
    -IQR = Q3 - Q1
    -Display interquartile range

```
FUNCTION interquartile() RETURNS void

	n = size of dataset
	q = quartilesCalculation()	//get quartile values

	IF n < 4 THEN
		PRINT "Interquartile Range = unknown"
	ELSE
		interquartileRange = q.Q3 - q.Q1
		PRINT "Interquartile Range = " + interquartileRange
	END IF

END FUNCTION
```

Purpose: Identifies and displays any data points considered outliers, defined as values lying outside the lower or upper fences

precondition: data set must have at least 2 values
postcondition: calculates IQR and finds upper and lower fence. displays outliers if not in that range
Time Complexity: O(n)
Step 1: Call to quartilesCalculation() to have Q1 and Q3 values
Step 2: Calculate IQR
    -IQR = Q3 - Q1
Step 3: Calculate fences
    -Upper Fence = Q3 + (1.5 * IQR)
    -Lower Fence = Q1 - (1.5 * IQR)
Step 4: Search for outliers
    -Loop through all data values in dataset
    -If a value is not within the range of lower fence to upper fence then it is an outlier
        -Display outlier
Step 5: Handle case of no outlier
    -If no values are outside the fences, then display "None"

```
FUNCTION outliers() RETURNS void
	q = quartilesCalculation()	//get quartile values
	n = size of dataset

	//calculate IQR
	IQR = q.Q3 - q.Q1

	//calculate fences
	upperFence = q.Q3 + (1.5 * IQR)
	lowerFence = q.Q1 - (1.5 * IQR)

	PRINT "Outliers = "

	searchOutlier = FALSE

	//check each value against the fences
	FOR each value in dataset DO
		IF value < lowerFence OR value > upperFence THEN
			PRINT value
			searchOutlier = TRUE
		END IF
```

```
        END FOR

        //if no outliers found
        IF searchOutlier == FALSE THEN
                PRINT “None”
        END IF

END FUNCTION
```

Purpose: Calculates and displays the sum of squared deviations of data values from the mean, a measure of total variation in the dataset.
precondition: data set must have at least 2 values
postcondition: calculates and displays the sum of squares
Time Complexity: O(n)
Step 1: Set size of data to n
Step 2: Handle too small dataset
    -If n < 2, display “unknown”
Step 3: Call to mean() function to calculate mean
Step 4: Call to sumPow() function to calculate
    -sum of squared difference = Σ(value - mean)^2
Step 5: Display result

```
FUNCTION sumOfSquares() RETURNS void

        n = size of dataset

        IF n < 2 THEN
                PRINT "Sum of Squares = unknown"
                RETURN
        END IF

        m = mean(dataset)       //compute the mean
        sos = sumPow(m , 2)  //compute the squared difference from mean
        PRINT "Sum of Squares = " + sos

END FUNCTION
```

precondition: data set must have at least 2 values
postcondition: calculates and displays mean absolute deviation
Purpose: Calculates and displays the Mean Absolute Deviation (MAD), which is the average of absolute deviations of data values from the mean.
Time Complexity: O(n)
Step 1: Set size of data to n
Step 2: Handle if dataset is too small
    -If n < 2, display “unknown”
Step 3: Call to mean() function to calculate
Step 4: Calculate absolute deviations
    -initalize mad = 0
    -For each value in dataset:
        -Find absolute difference between value and mean: |x - m|
        -Add result to mad
Step 5: Average the deviations
    -Divide mad by n to get the mean absolute deviation
        -mean absolute deviation / n
Step 6: Display Mean Absolute Deviation

```
FUNCTION meanAbsoluteDeviation() RETURNS void

    n = size of dataset

    IF n < 2 THEN
        PRINT "Mean Absolute Deviation = unknown"
        RETURN
    END IF

    m = mean(dataset)      //compute mean
    mad = 0.0

    FOR each value in dataset DO
        mad = mad + ABS(value - m)
    END FOR

    mad = mad / n          //average of absolute deviations

    PRINT "Mean = " + m
    PRINT "Mean Absolute Deviation = " + mad
```

END FUNCTION

Purpose: Calculates and displays the Root Mean Square (RMS), which is the square root of the average of squared data values
precondition: data set must have at least 2 values
postcondition: calculates and displays root mean square
Time Complexity: O(n)
Step 1: Set size of data to n
Step 2: Handle if dataset is too small
    -If n < 2, display “unknown”
Step 3: Square each value and sum
    -For each value in dataset:
        -calculate (value^2)
        -add to squaredValues Added
Step 4: Calculate average of squared values
    -Divide squaredValuedAdded by n
Step 5: Take square root
    -rms = sqrt(average of squared values)
Step 6: Display result

```
FUNCTION rootMeanSquare() RETURNS void

     n = size of dataset
    IF n < 2 THEN
        PRINT "Root Mean Square = unknown"
        RETURN
    END IF

    squaredValuesAdded = 0.0

    FOR each value in dataset DO
        squaredValuesAdded = squaredValuesAdded + (value ^ 2)
    END FOR

    rms = SQRT(squaredValuesAdded / n)

    PRINT "Root Mean Square = " + rms

END FUNCTION
```

Purpose: Calculates and displays the Standard Error of the Mean (SEM), which measures how far the sample mean is expected to vary from the true population mean.
precondition: data set must have at least 2 values
postcondition: calculates and displays standard error mean
Time Complexity: O(n)
Step 1: Set size of data to n
Step 2: Handle if dataset is too small
    -If n < 2, display "unknown"
Step 3: Call to standard deviation() function and calculate
Step 4: Calculate standard error mean
    -Divide standard deviation by square root of n
    -SEM = s / sqrt(n)
Step 5: Display standard error mean

```
FUNCTION standardErrorMean() RETURNS void

    n = size of dataset

    IF n < 2 THEN
        PRINT "Standard Error of the Mean = unknown"
        RETURN
    END IF

     s = standardDeviation()         //calculate standard deviation
    sem = s / SQRT(n)      //standard error mean formula

    PRINT "Standard Error of the Mean = " + sem

END FUNCTION
```