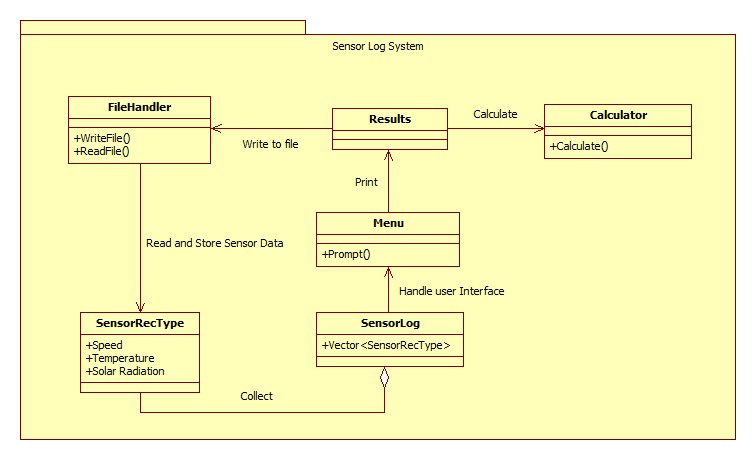
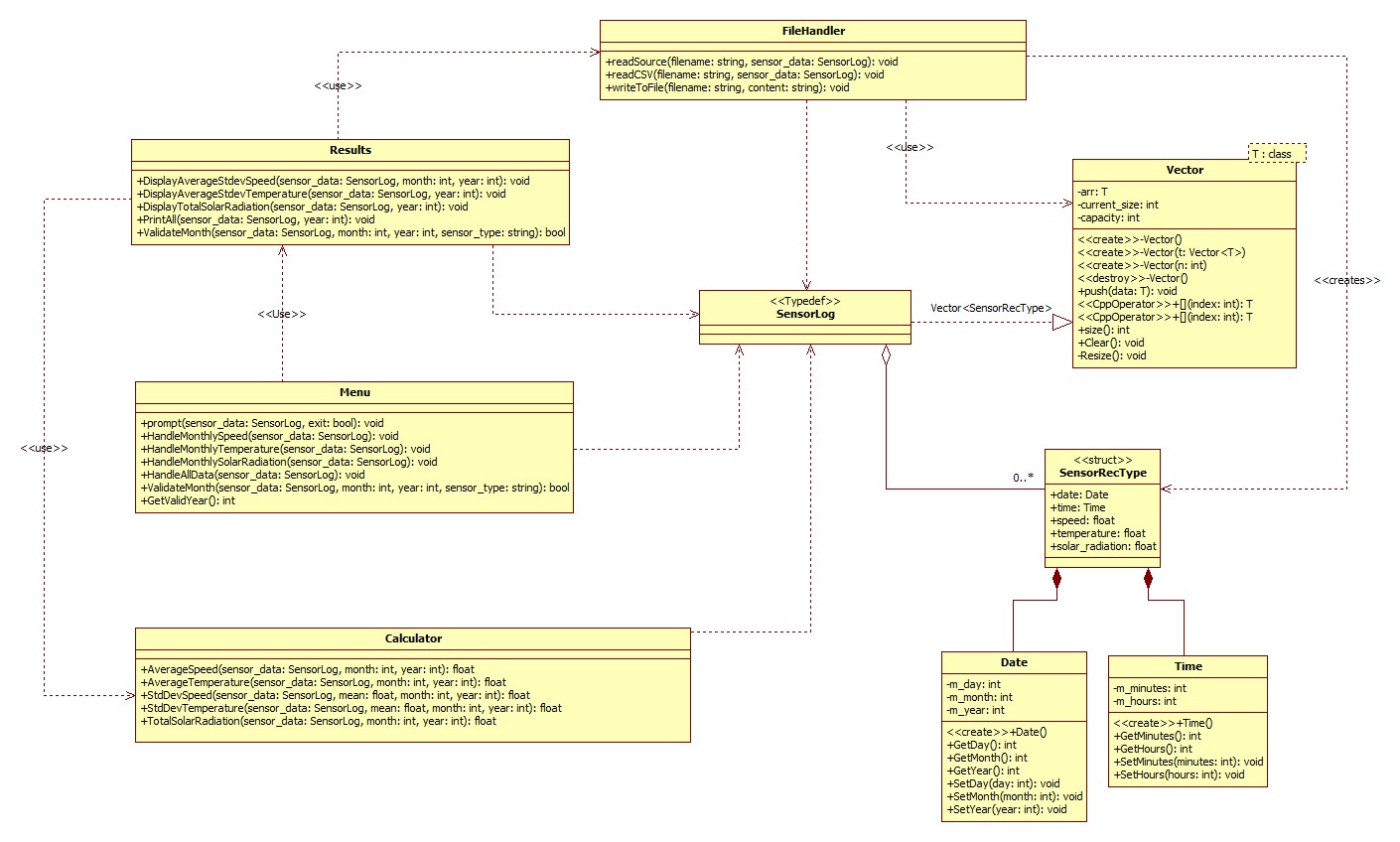
**UML**

**High Level:**



**Low Level**



**Data Dictionary**

**Vector.h**

|  |  |  |  |
| --- | --- | --- | --- |
| **Function/Member** | **Description** | **Type** | **Protection** |
| **Vector()** | Initialize an empty vector with default capacity, and current\_size and allocates initial memory. | Constructor | public |
| **Vector(const Vector <T> t)** | Copy constructor, creates a deep copy of the vector. | Copy  Constructor | public |
| **Vector(int n)** | Creates a vector with an initial capacity defined by the int parameter. | Constructor | public |
| **~Vector()** | Destructor, destroy all memory used by the vector | Destructor | public |
| **const T operator[int index] const** | Provides const/read-only access to element specified by the integer index parameter | Template class <T> | public |
| **T operator[int index]** | Provides access to element specified by the integer index parameter and allows for modification. | Template class <T> | public |
| **push(T data)** | Add an element at the end of the Vector and calls a Resize routine if number of elements exceed capacity. | void | public |
| **Resize()** | Double the capacity of the vector when called. | void | private |
| **Clear()** | Removes all elements from the vector and resets the vector to it’s initial empty state. | void | public |
| **arr** | Array that holds the elements | Template class <T> | private |
| **current\_size** | Number of elements currently | Int | private |
| **capacity** | Total capacity of allocated memory | Int | private |

**Vector Rationale**

The **arr** private member allows the class to have an array to store elements into. The **current\_size** and **capacity**, allows the class to track if it is time to resize the array. **Clear()** lets users reset the vector, this lets us reuse the Vector object without creating a new instance of it. **Resize()** is what makes our Vector a dynamic array, when it is called, it will resize the max capacity to allow for more data to be stored. The **push(T data)** allows for insertion of data into the dynamic array, and it also checks if **Resize()** should be called. The **operator[]** allows access to elements inside our Vector, The **const operator[] () const** is the same, but only provides read-only access which allows us to pass a const version of the vector into a parameter, this is helpful to prevent accidental modifications. **~Vector** is destructor that can prevent memory leaks. **Vector(int n)** is useful if you know how much data is going to be inside the Vector, this will reduce the number of times the vector needs to resize, and theoretically improve performance. **Vector(const Vector<T> &other** is a copy constructor the deep copies, which prevents issues when one vector is modified while the reference exists. **Vector()** is the default constructor that initializes the capacity to 1 to minimize memory usage for empty vectors.

**SensorRecType.h**

|  |  |  |  |
| --- | --- | --- | --- |
| **Function/Member** | **Description** | **Type** | **Protection** |
| **date** | Date of the sensor reading | Date | public |
| **time** | Time of the sensor reading | Time | public |
| **speed** | Wind Speed in m/s | float | public |
| **temperature** | Temperature in degrees Celsius | float | public |
| **solar\_radiation** | Solar Radiation measurement in W/m^2 | float | public |

**SensorRecType Rationale**

This vector struct provides the storage for individual sensor readings. It contains **date** and **time** for the Date and time the sensor was recorded. It also contains the Speed, Temperature , Solar Radiation readings which is a requirement for this program.

**Date.h**

|  |  |  |  |
| --- | --- | --- | --- |
| **Function/Member** | **Description** | **Type** | **Protection** |
| **Date()** | Constructor that initializes Date with default values | int | public |
| **GetDay()** | Returns m\_day value | int | public |
| **GetMonth()** | Returns m\_month value | int | public |
| **GetYear()** | Returns m\_year value | int | public |
| **SetDay(int day)** | Sets m\_day value | void | public |
| **SetMonth(int month)** | Sets m\_month value | void | public |
| **SetYear(int year)** | Sets m\_year value | void | public |
| **m\_day** | Contains the day value | int | private |
| **m\_month** | Contains the month value | int | private |
| **m\_year** | Contains the year value | int | private |

**Date Rationale**

Is a class that contains Setters and Getters to store or retrieve Day, Month, Year values of the recorded sensor readings.

**Time.h**

|  |  |  |  |
| --- | --- | --- | --- |
| **Function/Member** | **Description** | **Type** | **Protection** |
| **Time()** | Constructor that initializes Date with default values | int | public |
| **GetMinutes()** | Returns m\_minutes value | int | public |
| **GetHours()** | Returns m\_hours value | int | public |
| **SetMinutes(in)** | Returns m\_minutes value | int | public |
| **SetHours()** | Sets m\_hours value | void | public |
| **m\_minutes** | Contains the minute value | int | private |
| **m\_hours** | Contains the hour value | int | private |

**Time Rationale**

Is a class that contains Setters and Getters to store or retrieve hours and minute values of the recorded sensor readings.

**Calculator.h**

|  |  |  |  |
| --- | --- | --- | --- |
| **Function/Member** | **Description** | **Type** | **Protection** |
| **AverageSpeed (**  **Vector<float> &data) const;** | Calculates the average speed for the given month and year | float | Public |
| **AverageTemperature (**  **const Vector <float>&data) const** | Calculates the temperature speed for the given month and year | float | Public |
| **StdDevSpeed (**  **const Vector <float> &data, float mean) const** | Calculates the sample standard deviation of speed for the given month and year | float | Public |
| **StdDevTemperature (**  **const Vector<float> &data, float mean) const** | Calculates the sample standard deviation of temperature for the given month and year | float | Public |
| **TotalSolarRadiation(**  **const Vector<float> &data**  **) const** | Calculates the Total Solar Radiation for the given month and year | float | Public |

**Calculator Rationale**

Each function of this class has different statistical operations for the calculation of each sensor reading, Speed, Temperature, Solar Radiation. These statistical calculations is a requirement.

**Menu.h**

|  |  |  |  |
| --- | --- | --- | --- |
| **Function/Member** | **Description** | **Type** | **Protection** |
| **prompt(**  **const SensorLog &sensor\_data, bool &exit) const** | Displays main menu and handles user input | void | public |
| **HandleMonthlySpeed (const SensorLog &sensor\_data** | Gets and validate user input for specific month and year | void | public |
| **GetValidYear() const** | Gets and validates year input from user | int | public |

**Menu Rationale**

This class is responsible for user interaction. The **prompt()** function is responsible for displaying the Menu prompt, which handles user input and processes it into the function the user wishes to perform. The **HandleMonthlySpeed ()** is afunction is responsible for handling the user input for month, the rationale for creating a function like this is to maintain readable code. **GetValidYear()** is a function similar to **HandleMonthlySpeed()** where it prompts user and validates if the preconditions are correct before returning the year the user entered. My rationale for that function is to allow user to input a year without breaking the program, because if I use an int type to store user input, and they accidentally typed a non-digit character, the program will go into an infinite loop and break.

**Results.h**

|  |  |  |  |
| --- | --- | --- | --- |
| **Function/Member** | **Description** | **Type** | **Protection** |
| **DisplayAverageStdevSpeed(**  **const SensorLog &sensor\_data, const int month,**  **const int year) const** | Calculates and displays average wind speed and standard deviation for specified month/year | void | public |
| **DisplayAverageStdevTemperature(**  **const SensorLog &sensor\_data,**  **const int year) const** | Calculates and displays average temperature and standard deviation for each month of specified year | void | public |
| **DisplayTotalSolarRadiation(**  **const SensorLog &sensor\_data,**  **const int year) const;** | Calculates and displays total solar radiation for each month of specified year | void | public |
| **PrintAll(**  **const SensorLog &sensor\_data, const int year) const** | Writes wind speed, temperature and solar radiation data to WindTempSolar.csv | void | public |
| **ValidateMonth(**  **const SensorLog &sensor\_data, const int month,**  **const int year,**  **const string sensor\_type) const** | Checks if data exists for specified month and year | bool | public |

**Results Rationale**

Results class is responsible for displaying aggregated results by calling the Calculator class to perform a calculation. Each function Except for ValidateMonth() performs functions that is part of the requirement. ValidateMonth() is a helper function that checks if data exists. Helper functions reduce code duplication and allows for a more readable code.

**FileHandler.h**

|  |  |  |  |
| --- | --- | --- | --- |
| **Function/Member** | **Description** | **Type** | **Protection** |
| **readSource(**  **const string &filename, SensorLog &sensor\_data) const** | Reads a source file containing a list of csv filenames | void | public |
| **readCSV(**  **const string &filename, SensorLog &sensor\_data) const** | Reads and parses a CSV file containing sensor measurements | void | public |
| **writeToFile(const string &filename,**  **const string &content) const** | Writes content to a specified file | void | public |

**FileHandler Rationale**

This is responsible for any file handling operations. **readSource()** reads a list of filenames which can be processed into the vector struct by calling **readCSV**, this parses and inserts the sensor data and date into the struct which will then be pushed into a vector. The **writeToFile()** allows the insertion of data to be processed into a external csv file.

**Algorithm**

FUNCTION AverageSpeed(data)

Initialize sum to 0.0

Initialize data\_count to 0

FOR each value IN data

Add value to sum

Increment data\_count

END FOR

IF data\_count is less than 2 THEN

RETURN 0

ELSE

Calculate average = sum / data\_count

Round average to 1 decimal place

RETURN average

END IF

END FUNCTION

———————

FUNCTION AverageTemperature(data)

Initialize sum to 0.0

Initialize data\_count to 0

FOR each value IN data

Add value to sum

Increment data\_count

END FOR

IF data\_count is less than 2 THEN

RETURN 0

ELSE

Calculate average = sum / data\_count

Round average to 1 decimal place

RETURN average

END IF

END FUNCTION

————————-

FUNCTION StdDevSpeed(data, mean)

Initialize sum\_square\_diff to 0.0

Initialize data\_count to 0

FOR each value IN data

Calculate diff = value - mean

Add (diff \* diff) to sum\_square\_diff

Increment data\_count

END FOR

IF data\_count is less than 2 THEN

RETURN 0

ELSE

Calculate standard\_deviation = square\_root(sum\_square\_diff / (data\_count - 1))

Round standard\_deviation to 1 decimal place

RETURN standard\_deviation

END IF

END FUNCTION

—————————

FUNCTION StdDevTemperature(data, mean)

Initialize sum\_square\_diff to 0.0

Initialize data\_count to 0

FOR each value IN data

Calculate diff = value - mean

Add (diff \* diff) to sum\_square\_diff

Increment data\_count

END FOR

IF data\_count is less than 2 THEN

RETURN 0

ELSE

Calculate standard\_deviation = square\_root(sum\_square\_diff / (data\_count - 1))

Round standard\_deviation to 1 decimal place

RETURN standard\_deviation

END IF

END FUNCTION

———————————————-

FUNCTION TotalSolarRadiation(data)

Initialize totalRadiation to 0.0

Initialize data\_count to 0

FOR each value IN data

IF value is greater than or equal to 100 THEN

Convert from W/m² to kWh/m²: kWh\_conversion = value \* (10/60) / 1000

Add kWh\_conversion to totalRadiation

Increment data\_count

END IF

END FOR

IF data\_count is less than 2 THEN

RETURN 0

ELSE

Round totalRadiation to 1 decimal place

RETURN totalRadiation

END IF

END FUNCTION

————————————-

FUNCTION writeToFile(filename, content)

Initialize outputFile as file output stream

Open outputFile with filename

IF outputFile is not open THEN

Output error message to error stream: "Error: Could not open file 'filename' for writing."

RETURN

END IF

Write content to outputFile

Close outputFile

END FUNCTION

————————————

FUNCTION readSource(filename, sensor\_data)

Initialize file as file input stream with filename

IF file is not open THEN

Output error message to error stream: "Error Opening file filename"

RETURN

END IF

Initialize temp\_name as string

WHILE getline from file into temp\_name succeeds

Call readCSV(temp\_name, sensor\_data)

END WHILE

Close file

END FUNCTION

——————

FUNCTION readCSV(filename, sensor\_data)

Initialize variables:

token as empty string

line as empty string

S\_Index as -1 // Speed column index

T\_Index as -1 // Temperature column index

SR\_Index as -1 // Solar Radiation column index

vecHeader as empty Vector of strings

Open file with path "data/" + filename

IF file is not open THEN

Output error message to error stream: "Error opening file filename"

RETURN

END IF

// Process header row

Read first line from file into line

Create string stream ss from line

WHILE getline from ss into token with delimiter ',' succeeds

Add token to vecHeader

END WHILE

Clear ss stream

// Find column indices in header

FOR i from 0 to vecHeader.size() - 1

Set header to vecHeader[i]

IF header equals "S" THEN

Set S\_Index to i

END IF

IF header equals "T" THEN

Set T\_Index to i

END IF

IF header equals "SR" THEN

Set SR\_Index to i

END IF

END FOR

// Process data rows

WHILE getline from file into line succeeds

IF line is empty OR line contains only whitespace THEN

CONTINUE to next iteration

END IF

Create string stream ss from line

Initialize variables for current row:

s\_speed, s\_temperature, s\_solar\_radiation, skip as empty strings

sDate, sTime, sDay, sMonth, sYear, sHour, sMinute as empty strings

day, year, month, hours, minutes as 0

f\_speed, f\_temperature, f\_solar\_radiation as 0.0

// Process each column in the row

FOR i from 0 to vecHeader.size() - 1

IF i equals 0 THEN

// Process date/time column

Read from ss into sDate until space

Read from ss into sTime until comma

Create dateStream from sDate

Read from dateStream into sDay until '/'

Read from dateStream into sMonth until '/'

Read from dateStream into sYear to end

Create timeStream from sTime

Read from timeStream into sHour until ':'

Read from timeStream into sMinute to end

ELSE IF i equals S\_Index THEN

// Process speed column

Read from ss into s\_speed until comma

IF s\_speed is empty THEN

Set s\_speed to "0"

END IF

ELSE IF i equals T\_Index THEN

// Process temperature column

Read from ss into s\_temperature until comma

IF s\_temperature is empty THEN

Set s\_temperature to "0"

END IF

ELSE IF i equals SR\_Index THEN

// Process solar radiation column

Read from ss into s\_solar\_radiation until comma

IF s\_solar\_radiation is empty THEN

Set s\_solar\_radiation to "0"

END IF

ELSE

// Skip unused column

Read from ss into skip until comma

END IF

END FOR

// Convert string values to numeric

Convert sDay to integer day

Convert sMonth to integer month

Convert sYear to integer year

Convert sHour to integer hours

Convert sMinute to integer minutes

IF s\_speed is not empty THEN

Convert s\_speed to float f\_speed

ELSE

Set f\_speed to 0.0

END IF

IF s\_temperature is not empty THEN

Convert s\_temperature to float f\_temperature

ELSE

Set f\_temperature to 0.0

END IF

IF s\_solar\_radiation is not empty THEN

Convert s\_solar\_radiation to float f\_solar\_radiation

ELSE

Set f\_solar\_radiation to 0.0

END IF

// Create Date object

Create date object

Set date.day to day

Set date.month to month

Set date.year to year

// Create Time object

Create time object

Set time.hours to hours

Set time.minutes to minutes

// Create SensorRecType record

Create tempData as SensorRecType

Set tempData.date to date

Set tempData.time to time

Set tempData.speed to f\_speed

Set tempData.temperature to f\_temperature

Set tempData.solar\_radiation to f\_solar\_radiation

// Add to sensor\_data collection

Add tempData to sensor\_data

END WHILE

Close file

END FUNCTION

———————————————-

FUNCTION DisplayAverageStdevSpeed(sensor\_data, month, year)

Initialize calculate as Calculator object

Initialize average to 0.0

Initialize standard\_deviation to 0.0

Initialize data as empty Vector of floats

IF ValidateMonth(sensor\_data, month, year, "S") THEN

FOR each record IN sensor\_data

IF record.date.month equals month AND record.date.year equals year THEN

Add record.speed to data

END IF

END FOR

Set average to calculate.AverageSpeed(data)

Set standard\_deviation to calculate.StdDevSpeed(data, average)

Output month, year, average speed, and standard deviation

ELSE

Output "month year: No Data"

END IF

END FUNCTION

——————————-

FUNCTION DisplayAverageStdevTemperature(sensor\_data, year)

Initialize calculate as Calculator object

Initialize average to 0.0

Initialize standard\_deviation to 0.0

Initialize month to 0

Initialize month\_exist to false

Initialize data as empty Vector of floats

FOR each record IN sensor\_data

IF record.date.year equals year THEN

Set month\_exist to true

EXIT loop

END IF

END FOR

IF month\_exist THEN

Output year

FOR month\_index from 1 to 12

Set month to month\_index

Clear data

FOR each record IN sensor\_data

IF record.date.month equals month AND record.date.year equals year THEN

Add record.temperature to data

END IF

END FOR

IF ValidateMonth(sensor\_data, month, year, "T") THEN

Set average to calculate.AverageTemperature(data)

Set standard\_deviation to calculate.StdDevTemperature(data, average)

Output month name, average temperature, and standard deviation

ELSE

Output "month name: No Data"

END IF

END FOR

ELSE

Output "year: No data"

END IF

END FUNCTION

————————————

FUNCTION DisplayTotalSolarRadiation(sensor\_data, year)

Initialize calculate as Calculator object

Initialize month to 0

Initialize total\_solar\_radiation to 0.0

Initialize data as empty Vector of floats

Initialize month\_exist to false

FOR each record IN sensor\_data

IF record.date.year equals year THEN

Set month\_exist to true

EXIT loop

END IF

END FOR

IF month\_exist THEN

Output year

FOR month\_index from 1 to 12

Set month to month\_index

Clear data

FOR each record IN sensor\_data

IF record.date.month equals month AND record.date.year equals year THEN

Add record.solar\_radiation to data

END IF

END FOR

IF ValidateMonth(sensor\_data, month, year, "SR") THEN

Set total\_solar\_radiation to calculate.TotalSolarRadiation(data)

Output month name, total\_solar\_radiation

ELSE

Output "month name: No Data"

END IF

END FOR

ELSE

Output "year: No data"

END IF

END FUNCTION

——————————————————-

FUNCTION PrintAll(sensor\_data, year)

Initialize calculate as Calculator object

Initialize handleFile as FileHandler object

Initialize output as string output stream

Initialize average\_speed, stdev\_speed, average\_temperature, stdev\_temperature, total\_solar\_radiation to 0.0

Initialize month to 0

Initialize month\_exist to false

Initialize speed\_data, temperature\_data, solar\_radiation\_data as empty Vectors of floats

FOR each record IN sensor\_data

IF record.date.year equals year THEN

Set month\_exist to true

EXIT loop

END IF

END FOR

IF month\_exist THEN

Add year to output stream

FOR month\_index from 1 to 12

Set month to month\_index

Clear speed\_data, temperature\_data, solar\_radiation\_data

Initialize empty\_data to 0

FOR each record IN sensor\_data

IF record.date.month equals month AND record.date.year equals year THEN

Add record.speed to speed\_data

Add record.temperature to temperature\_data

Add record.solar\_radiation to solar\_radiation\_data

END IF

END FOR

Set average\_speed to calculate.AverageSpeed(speed\_data)

Set stdev\_speed to calculate.StdDevSpeed(speed\_data, average\_speed)

Set average\_temperature to calculate.AverageTemperature(temperature\_data)

Set stdev\_temperature to calculate.StdDevTemperature(temperature\_data, average\_temperature)

Set total\_solar\_radiation to calculate.TotalSolarRadiation(solar\_radiation\_data)

Add month name to output stream

IF average\_speed not equals 0 THEN

Add average\_speed to output stream

ELSE

Increment empty\_data

END IF

IF stdev\_speed not equals 0 THEN

Add stdev\_speed to output stream

ELSE

Increment empty\_data

END IF

IF average\_temperature not equals 0 THEN

Add average\_temperature to output stream

ELSE

Increment empty\_data

END IF

IF stdev\_temperature not equals 0 THEN

Add stdev\_temperature to output stream

ELSE

Increment empty\_data

END IF

IF total\_solar\_radiation not equals 0 THEN

Add total\_solar\_radiation to output stream

ELSE

Increment empty\_data

END IF

IF empty\_data equals 5 THEN

Add "No Data" to output stream

END IF

Add newline to output stream

END FOR

Call handleFile.writeToFile("WindTempSolar.csv", output.string)

END IF

END FUNCTION

——————————————-FUNCTION ValidateMonth(sensor\_data, month, year, sensor\_type)

IF sensor\_type equals "S" THEN

FOR each record IN sensor\_data

IF record.date.month equals month AND

record.date.year equals year AND

record.speed not equals 0 THEN

RETURN true

END IF

END FOR

END IF

IF sensor\_type equals "T" THEN

FOR each record IN sensor\_data

IF record.date.month equals month AND

record.date.year equals year AND

record.temperature not equals 0 THEN

RETURN true

END IF

END FOR

END IF

IF sensor\_type equals "SR" THEN

FOR each record IN sensor\_data

IF record.date.month equals month AND

record.date.year equals year AND

record.solar\_radiation not equals 0 THEN

RETURN true

END IF

END FOR

END IF

RETURN false

END FUNCTION

——————————————————

FUNCTION int\_to\_month(month)

Initialize month\_arr as array of strings:

["", "January", "February", "March", "April", "May", "June",

"July", "August", "September", "October", "November", "December"]

RETURN month\_arr[month]

END FUNCTION