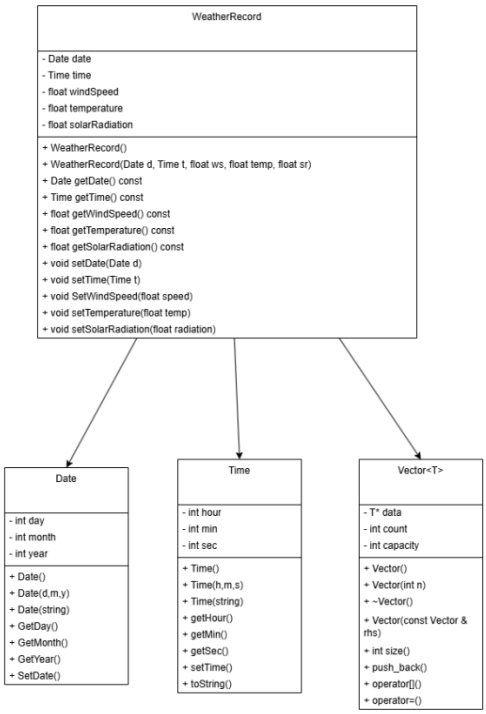
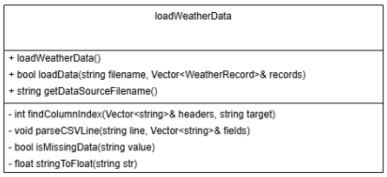
ICT283 Assignment1

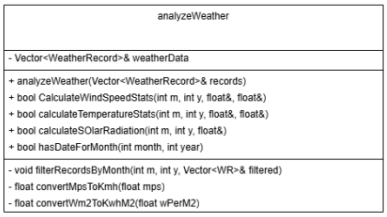
Name: Dhruv Goswami
Student ID: 35373785

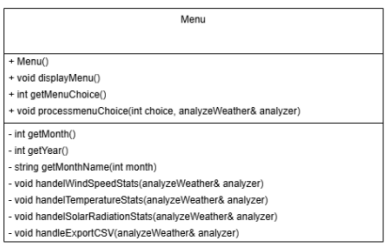
Contents

UML


WeatherRecord
- Date date
- Time time
- float windSpeed
- float temperature
- float solarRadiation
+ WeatherRecord()
+ WeatherRecord(Date d, Time t, float ws, float temp, float sr)
+ Date getDate() const
+ Time getTime() const
+ float getWindSpeed() const
+ float getTemperature() const
+ float getSolarRadiation() const
+ void setDate(Date d)
+ void setTime(Time t)
+ void SetWindSpeed(float speed)
+ void setTemperature(float temp)
+ void setSolarRadiation(float radiation)
Date
- int day
- int month
- int year
+ Date()
+ Date(d,m,y)
+ Date(string)
+ GetDay()
+ GetMonth()
+ GetYear()
+ SetDate()
Time
- int hour
- int min
- int sec
+ Time()
+ Time(h,m,s)
+ Time(string)
+ getHour()
+ getMin()
+ getSec()
+ setTime()
+ toString()
Vector<T>
- T* data
- int count
- int capacity
+ Vector()
+ Vector(int n)
+ ~Vector()
+ Vector(const Vector &rhs)
+ int size()
+ push_back()
+ operator[]()
+ operator=()


loadWeatherData
+ loadWeatherData()
+ bool loadData(string filename, Vector<WeatherRecord>& records)
+ string getDataSourceFilename()
- int findColumnIndex(Vector<string>& headers, string target)
- void parseCSVLine(string line, Vector<string>& fields)
- bool isMissingData(string value)
- float stringToFloat(string str)


analyzeWeather
- Vector<WeatherRecord>& weatherData
+ analyzeWeather(Vector<WeatherRecord>& records)
+ bool CalculateWindSpeedStats(int m, int y, float&, float&)
+ bool calculateTemperatureStats(int m, int y, float&, float&)
+ bool calculateSOlarRadiation(int m, int y, float&)
+ bool hasDateForMonth(int month, int year)
- void filterRecordsByMonth(int m, int y, Vector<WR>& filtered)
- float convertMpsToKmh(float mps)
- float convertWm2ToKwhM2(float wPerM2)


Menu
+ Menu()
+ void displayMenu()
+ int getMenuChoice()
+ void processmenuChoice(int choice, analyzeWeather& analyzer)
- int getMonth()
- int getYear()
- string getMonthName(int month)
- void handelWindSpeedStats(analyzeWeather& analyzer)
- void handelTemperatureStats(analyzeWeather& analyzer)
- void handelSolarRadiationStats(analyzeWeather& analyzer)
- void handleExportCSV(analyzeWeather& analyzer)

Data Dictionary

Date Class

Name	Type	Protection	Description	Rationale
Date	Class		Manages date information including day, month and year	Encapsulated date info and provides validity checks for each field
day	int	-	The day of the date	Storage for the day info
month	int	-	The month of the date	Storage for month info
year	Int	-	The year of the date	Storage for year info
Date()	procedure	+	Default constructor initializing day, month and year to default values	Provides a default state for creating the object
Date(int d, int m, int y)	procedure	+	Parameterized constructor initializing with provided values	Provides an easy way to create a date object with values
Date(string dateStr)	procedure	+	Constructor to initialize date from string format DD/MM/YYYY	Allows creation from string input
Getday()	int	+	Retrieves the current day of the date	Accessor method for day value
getMonth()	int	+	Retrives the current month of the date	Accessor mothod for month values
getYear()	int	+	Retrives the current year of the date	Accessor method for the year values
setDate(int d, int m, int y)	procedure	+	Sets complete date with provided valuies	Mutator method for all date components

Time Class

Name	Type	Protection	Description	Rationale
Time	Class		Manages time infor including hours, minutes and seconds	Enxapsulates time info and provides validity checks for each field
hour	int	-	The hour of the time	Storage for hours in 24 hour format
min	int	-	The minutes of the time	Storage for minutes
sec	Int	-	The seconds of the time	Storage for seconds
Time()	procedure	+	Default constructor initializing hour, minute and second to 0	Provides a default state for creating the object
Time(string timeStr)	procedure	+	Constructor to initialize time from string format HH:MM:SS	Allows creation from string
getHour()	int	+	Retrieves the current hour of the time	Accessor method for hour value
getMin()	int	+	Retrieves the current minute of the time	Accessor method for minute value
getSec()	int	+	Retrieves the current second of the time	Accessor method for seconds value
setTime(int h, int m, int s)	Procedure	+	Sets complete time with provided values	Mutator method for all time components
toString()	string	+	Converts the time to a string in format HH:MM:SS	Provides string representation

Vector Class

Name	Type	Protection	Description	Rationale
Vector	Class		A custom vector class that manages a dynamic array of type T	Storage for data structure similar to a stack with encapsulates a pointer
data	T*	-	Pointer to the dynamic array storing elements of type T	Allows us to dynamically allocate memory to store multiple values
Count	int	-	The current number of elements stored in vector	Used for getting information on how many items are currently in the array
capacity	Int	-	The max number of elements that can be stored before resizing is required	Used for checking how many items can be added to the array of more memory needs to be allocated
Vector()	procedure	+	Default constructore initializes on empty vector with default capacity	Default state with a capacity of 10
Vector(const Vector<T> & rhs)	procedure	+	Copy constructor initializes a new vector as deep copy of an existing vector	Ensures a deep copy when trying to create a new vector based on another vector using copy method
Vector (int n)	procedure	+	Parameterized constructor that initializes the vector with a given capapcity	Provides a method for creating a vector with an initial capacity

~Vector()	procedure	+	Destructor, releases the dynamically allocated memory for the array	Required for freeing ip memory consumed by the pointer when the vector goes out of scope
Size()	Int	+	Returns the current number of elements in the vector	Provides access to current size
Push_back(const T & element)	procedure	+	Adds an element to the end of the vector	Provides a method for adding items to the vector while increasing capacity if required
Operator[ ](int index)	T &	+	Overloaded index operator for accessing elements by index	Provides indexed access to vector elements, allowing access to elements
Operator = (const Vector<T> & rhs)	Vector<T>&	+	Overloaded assignment operator for deep copying another vector	Allows assignment of vectors while performing a deep copy
Resize()	procedure	-	Resizes the vector when the size exceedes the current capacity	Dynamically adjusts the memory allocation to fit new elements

WeatherRecord Class

Name	Type	Protection	Description	Rationale
WeatherRecord			Encapsulates a single weather measurement record.	Provides structured storage for weather data points
date	Date	-	Date of the weather measurement.	Storage for when measurement was taken
Time	Time	-	Time of the weather measurement.	Storage for specific time of measurement
windspeed	Float	-	Wind speed in meters per second.	Storage for wind speed data
Temperature	Float	-	Temperature in degrees Celsius.	Storage for temperature data
solarRadiation	Float	-	Solar radiation in watts per square meter.	Storage for solar radiation data
WeatherRecord()	Procedure	+	Default constructor initializing all values to defaults.	Provides default state for object creation
WeatherRecord(Date d, Time t, float ws, float temp, float sr)	Procedure	+	Parameterized constructor with all weather data.	Allows creation with complete data set
getDate()	Date	+	Returns the date of the measurement.	Accessor for date information
getTime()	Time	+	Returns the time of the measurement.	Accessor for time information

getWindSpeed()	float	+	Returns wind speed value.	Accessor for wind speed data
getTemperature()	Float	+	Returns temperature value.	Accessor for temperature data
getSolarRadiation()	float	+	Returns solar radiation value.	Accessor for solar radiation data
setDate(Date d)	procedure	+	Sets the date of measurement.	Mutator for date information
setTime(Time t)	procedure	+	Sets the time of measurement.	Mutator for time information
setWindSpeed(float speed)	procedure	+	Sets wind speed value.	Mutator for wind speed data
setTemperature(float temp)	Procedure	+	Sets temperature value.	Mutator for temperature data
setSolarRadiation(float radiation)	procedure	+	Sets solar radiation value.	Mutator for solar radiation data

Psudocode

High Level

Main.cpp

```
START
    OUTPUT "Weather Data Analysis Program"
    OUTPUT "Loading data..."

    DECLARE dataLoader AS loadWeatherData
    DECLARE dataFile AS STRING
    dataFile = dataLoader.getDataSourceFilename()

    IF dataFile IS EMPTY THEN
        OUTPUT "Failed to load data source filename."
        RETURN 1
    ENDIF

    DECLARE records AS Vector<WeatherRecord>
    IF NOT dataLoader.loadData(dataFile, records) THEN
        OUTPUT "Failed to load weather data from " + dataFile
        RETURN 1
    ENDIF

    OUTPUT "Data loaded successfully. Number of records: " + records.size()
```

```
    DECLARE analyzer AS analyzeWeather(records)
    DECLARE menu AS Menu
    DECLARE choice AS INTEGER

    DO
        menu.displayMenu()
        choice = menu.getMenuChoice()
        menu.processmenuChoice(choice, analyzer)
    WHILE choice != 5

    RETURN 0
END
```

Low Level

Date.cpp

```
PROCEDURE Date() : Date
    day ← 1
    month ← 1
    year ← 2000
ENDPROCEDURE

PROCEDURE Date(d : INTEGER, m : INTEGER, y : INTEGER) : Date
    day ← d
    month ← m
    year ← y
ENDPROCEDURE

PROCEDURE Date(dateStr : STRING) : Date
    DECLARE ss AS STRINGSTREAM(dateStr)
    DECLARE slash AS CHAR
    ss >> day >> slash >> month >> slash >> year
ENDPROCEDURE

FUNCTION GetDay() : INTEGER
    RETURN day
ENDFUNCTION

FUNCTION GetMonth() : INTEGER
    RETURN month
```

```
ENDFUNCTION


FUNCTION GetYear() : INTEGER
    RETURN year
ENDFUNCTION


PROCEDURE SetDate(d : INTEGER, m : INTEGER, y : INTEGER)
    day ← d
    month ← m
    year ← y
ENDPROCEDURE
```

Time.cpp

```
PROCEDURE Time() : Time
    hour ← 0
    min ← 0
    sec ← 0
ENDPROCEDURE

PROCEDURE Time(timeStr : STRING) : Time
    DECLARE ss AS STRINGSTREAM(timeStr)
    DECLARE sep AS CHAR
    ss >> hour >> sep >> min >> sep >> sec
ENDPROCEDURE

FUNCTION getHour() : INTEGER
    RETURN hour
ENDFUNCTION

FUNCTION getMin() : INTEGER
    RETURN min
ENDFUNCTION

FUNCTION getSec() : INTEGER
    RETURN sec
ENDFUNCTION

PROCEDURE setTime(h : INTEGER, m : INTEGER, s : INTEGER)
```

```
    hour ← h
    min ← m
    sec ← s
ENDPROCEDURE

FUNCTION toString() : STRING
    DECLARE oss AS OSTRINGSTREAM
    oss << SETW(2) << SETFILL('0') << hour << ":"
        << SETW(2) << SETFILL('0') << min << ":"
        << SETW(2) << SETFILL('0') << sec
    RETURN oss.str()
ENDFUNCTION
```

Vector.cpp

```
PROCEDURE Vector(size : INTEGER = 10)
    count ← 0
    capacity ← size
    data ← NEW T[capacity]
ENDPROCEDURE

PROCEDURE Vector(other : Vector<T>)
    count ← other.count
    capacity ← other.capacity
    data ← NEW T[capacity]
    FOR i = 0 TO count - 1 DO
        data[i] ← other.data[i]
    NEXT i
ENDPROCEDURE

PROCEDURE ~Vector()
    DELETE[] data
ENDPROCEDURE

FUNCTION size() : INTEGER
    RETURN count
ENDFUNCTION

PROCEDURE push_back(value : T)
    IF count == capacity THEN
```

```
        CALL resize()
    ENDIF
    data[count] ← value
    count ← count + 1
ENDPROCEDURE

FUNCTION operator[](index : INTEGER) : T&
    ASSERT(index >= 0 AND index < count)
    RETURN data[index]
ENDFUNCTION

PROCEDURE resize()
    capacity ← capacity * 2
    DECLARE newData AS T*
    newData ← NEW T[capacity]
    FOR i = 0 TO count - 1 DO
        newData[i] ← data[i]
    NEXT i
    DELETE[] data
    data ← newData
ENDPROCEDURE
```

loadWeatherData.cpp

```
FUNCTION getDataSourceFilename() : STRING
    DECLARE sourceFile AS IFSTREAM("data/data_source.txt")
    IF NOT sourceFile THEN
        OUTPUT "cannot open data_source.txt"
        RETURN ""
    ENDIF

    DECLARE filename AS STRING
    GETLINE(sourceFile, filename)
    sourceFile.close()

    RETURN "data/" + filename
ENDFUNCTION

FUNCTION loadData(filename : STRING, records : Vector<WeatherRecord>) : BOOLEAN
    DECLARE file AS IFSTREAM(filename)
    IF NOT file THEN
        OUTPUT "Cannot open the file " + filename
        RETURN FALSE
    ENDIF

    DECLARE headerLine AS STRING
    IF NOT GETLINE(file, headerLine) THEN
        OUTPUT "Cannot read the header line from the file"
        RETURN FALSE
```

```
ENDIF

DECLARE headers AS Vector<STRING>
CALL parseCSVLine(headerLine, headers)

DECLARE wastIndex AS INTEGER = findColumnIndex(headers, "WAST")
DECLARE sIndex AS INTEGER = findColumnIndex(headers, "S")
DECLARE tIndex AS INTEGER = findColumnIndex(headers, "T")
DECLARE srIndex AS INTEGER = findColumnIndex(headers, "SR")

IF wastIndex == -1 OR sIndex == -1 OR tIndex == -1 OR srIndex == -1 THEN
   OUTPUT "Cannot find required columns in the header file"
   RETURN FALSE
ENDIF

DECLARE line AS STRING
WHILE GETLINE(file, line) DO
   DECLARE fields AS Vector<STRING>
   CALL parseCSVLine(line, fields)

   IF fields.size() <= MAX(wastIndex, sIndex, tIndex, srIndex) THEN
      CONTINUE
   ENDIF

   IF isMissingData(fields[sIndex]) OR
     isMissingData(fields[tIndex]) OR
```

```
            isMissingData(fields[srIndex]) THEN
            CONTINUE
        ENDIF

        DECLARE datetime AS STRING = fields[wastIndex]
        DECLARE dtStream AS STRINGSTREAM(datetime)
        DECLARE dateStr, timeStr AS STRING
        dtStream >> dateStr >> timeStr

        DECLARE date AS Date(dateStr)
        DECLARE time AS Time(timeStr)

        DECLARE windSpeed AS FLOAT = stringToFloat(fields[sIndex])
        DECLARE temperature AS FLOAT = stringToFloat(fields[tIndex])
        DECLARE solarRadiation AS FLOAT = stringToFloat(fields[srIndex])

        IF solarRadiation >= 100.0 THEN
            DECLARE record AS WeatherRecord(date, time, windSpeed, temperature, solarRadiation)
            records.push_back(record)
        ENDIF
    ENDWHILE

    file.close()
    RETURN TRUE
ENDFUNCTION
```

analyzeWeather.cpp

```
FUNCTION CalculateWindSpeedStats(month : INTEGER, year : INTEGER, meanSpeed : FLOAT,
stdev : FLOAT) : BOOLEAN
    DECLARE filteredRecords AS Vector<WeatherRecord>
    CALL filterRecordsByMonth(month, year, filteredRecords)

    IF filteredRecords.size() == 0 THEN
        RETURN FALSE
    ENDIF

    DECLARE windSpeeds AS Vector<FLOAT>
    FOR i = 0 TO filteredRecords.size() - 1 DO
        DECLARE speedKmh AS FLOAT = convertMpsToKmh(filteredRecords[i].getWindSpeed())
        windSpeeds.push_back(speedKmh)
    NEXT i

    meanSpeed ← statistics::calculateMean(windSpeeds)
    stdev ← statistics::calculateStandardDeviation(windSpeeds, meanSpeed)
    RETURN TRUE
ENDFUNCTION

FUNCTION calculateTemperatureStats(month : INTEGER, year : INTEGER, meanTemp : FLOAT,
stdev : FLOAT) : BOOLEAN
    DECLARE filteredRecords AS Vector<WeatherRecord>
    CALL filterRecordsByMonth(month, year, filteredRecords)

    IF filteredRecords.size() == 0 THEN
```

```
        RETURN FALSE
    ENDIF

    DECLARE temperatures AS Vector<FLOAT>
    FOR i = 0 TO filteredRecords.size() - 1 DO
        temperatures.push_back(filteredRecords[i].getTemperature())
    NEXT i

    meanTemp ← statistics::calculateMean(temperatures)
    stdev ← statistics::calculateStandardDeviation(temperatures, meanTemp)
    RETURN TRUE
ENDFUNCTION

FUNCTION calculateSOlarRadiation(month : INTEGER, year : INTEGER, totalRadiation : FLOAT) :
BOOLEAN
    DECLARE filteredRecords AS Vector<WeatherRecord>
    CALL filterRecordsByMonth(month, year, filteredRecords)

    IF filteredRecords.size() == 0 THEN
        RETURN FALSE
    ENDIF

    totalRadiation ← 0.0
    FOR i = 0 TO filteredRecords.size() - 1 DO
        DECLARE radiationKwh AS FLOAT =
convertWm2ToKwhM2(filteredRecords[i].getSolarRadiation())
        totalRadiation ← totalRadiation + radiationKwh
```

```
    NEXT i


    RETURN TRUE
ENDFUNCTION


PROCEDURE filterRecordsByMonth(month : INTEGER, year : INTEGER, filteredRecords :
Vector<WeatherRecord>)
    FOR i = 0 TO weatherData.size() - 1 DO
        IF weatherData[i].getDate().GetMonth() == month AND weatherData[i].getDate().GetYear()
== year THEN
            filteredRecords.push_back(weatherData[i])
        ENDIF
    NEXT i
ENDPROCEDURE


FUNCTION convertMpsToKmh(mps : FLOAT) : FLOAT
    RETURN mps * 3.6
ENDFUNCTION


FUNCTION convertWm2ToKwhM2(wPerM2 : FLOAT) : FLOAT
    RETURN wPerM2 * (1.0 / 6.0) / 1000.0
ENDFUNCTION
```

Menu.cpp

```
PROCEDURE displayMenu()
    OUTPUT "Weather Data Analysis Menu"
    OUTPUT "1. Analyze Wind Speed Statistics"
    OUTPUT "2. Analyze Temperature Statistics"
    OUTPUT "3. Analyze Solar Radiation Statistics"
    OUTPUT "4. Export Data to CSV"
    OUTPUT "5. Exit"
ENDPROCEDURE

FUNCTION getMenuChoice() : INTEGER
    DECLARE choice AS INTEGER
    OUTPUT "Enter your choice (1-5): "
    INPUT choice
    RETURN choice
ENDFUNCTION

PROCEDURE processmenuChoice(choice : INTEGER, analyzer : analyzeWeather)
    SWITCH choice
    CASE 1:
        CALL handelWindSpeedStats(analyzer)
    CASE 2:
        CALL handelTemperatureStats(analyzer)
    CASE 3:
        CALL handelSolarRadiationStats(analyzer)
    CASE 4:
```

```
        CALL handleExportCSV(analyzer)
    CASE 5:
        OUTPUT "Exiting the program."
    DEFAULT:
        OUTPUT "Invalid choice. Please try again."
    ENDSWITCH
ENDPROCEDURE

FUNCTION getMonth() : INTEGER
    DECLARE month AS INTEGER
    OUTPUT "Enter month (1-12): "
    INPUT month
    WHILE month < 1 OR month > 12 DO
        OUTPUT "Invalid month. Please enter a valid month (1-12): "
        INPUT month
    ENDWHILE
    RETURN month
ENDFUNCTION

FUNCTION getYear() : INTEGER
    DECLARE year AS INTEGER
    OUTPUT "Enter year (e.g., 2023): "
    INPUT year
    RETURN year
ENDFUNCTION
```

Statistics.cpp

```
FUNCTION calculateMean(data : Vector<FLOAT>) : FLOAT
    IF data.size() == 0 THEN
        RETURN 0.0
    ENDIF

    DECLARE sum AS FLOAT = 0.0
    FOR i = 0 TO data.size() - 1 DO
        sum ← sum + data[i]
    NEXT i

    RETURN sum / data.size()
ENDFUNCTION

FUNCTION calculateStandardDeviation(data : Vector<FLOAT>, mean : FLOAT) : FLOAT
    IF data.size() <= 1 THEN
        RETURN 0.0
    ENDIF

    DECLARE sumSquareDiff AS FLOAT = 0.0
    FOR i = 0 TO data.size() - 1 DO
        DECLARE diff AS FLOAT = data[i] - mean
        sumSquareDiff ← sumSquareDiff + (diff * diff)
    NEXT i

    RETURN SQRT(sumSquareDiff / (data.size() - 1))
```

```
ENDFUNCTION


FUNCTION calculateSum(data : Vector<FLOAT>) : FLOAT
    DECLARE sum AS FLOAT = 0.0
    FOR i = 0 TO data.size() - 1 DO
        sum ← sum + data[i]
    NEXT i
    RETURN sum
ENDFUNCTION
```

Test Plan

Data Class Testing

Date test plan

Test	Description	Actual Test Data	Expected output	Pass/Fail
1	Check that the default constructor initializes the date with default values.	NA = Default constructor	Day = 1, Month = 1, Year = 2000	Pass
2	Check that the parameterized constructor initializes the date with specific values.	Day = 8, Month = 5, Year = 2004	Day = 8, Month = 5, Year = 2004	Pass
3	Check that the string constructor parses date correctly from DD/MM/YYYY format.	"15/06/2023"	Day = 15, Month = 6, Year = 2023	Pass
4	Check that GetDay() returns the correct day value.	Date(25, 12, 2023)	GetDay() returns 25	Pass
5	Check that GetMonth() returns the correct month value.	Date(25, 12, 2023)	GetMonth() returns 12	Pass
6	Check that GetYear() returns the correct year value.	Date(25, 12, 2023)	GetYear() returns 2023	Pass
7	Check that SetDate() correctly	SetDate(1, 1, 2025)	Day = 1, Month = 1, Year = 2025	Pass

	updates all date components.			
8	Check boundary day value (31st).	Day = 31, Month = 1, Year = 2023	GetDay() returns 31	Pass
9	Check boundary month value (December).	Day = 15, Month = 12, Year = 2023	GetMonth() returns 12	Pass
10	Check that string parsing handles single digit values.	"5/3/2023"	Day = 5, Month = 3, Year = 2023	Pass

Date Test Results

```
--------------------------------------------------------------------------------
Test 1: Default Constructor
--------------------------------------------------------------------------------
Date object successfully created
Day: 1 Month: 1 Year: 2000
--------------------------------------------------------------------------------
```

```
--------------------------------------------------------------------------------
Test 2: Parameterized Constructor
--------------------------------------------------------------------------------
Date object successfully created with day = 8, month = 5, year = 2004
Day: 8 Month: 5 Year: 2004
--------------------------------------------------------------------------------
```

```
--------------------------------------------------------------------------------
Test 3: String Constructor
--------------------------------------------------------------------------------
Creating date from string: "15/06/2023"
Day: 15 Month: 6 Year: 2023
--------------------------------------------------------------------------------
```

```
--------------------------------------------------------------------------------
Test 4: GetDay() Method
--------------------------------------------------------------------------------
Date created with day = 25
GetDay() returned: 25
--------------------------------------------------------------------------------
```

```
--------------------------------------------------------------------------------
Test 5: GetMonth() Method
--------------------------------------------------------------------------------
Date created with month = 12
GetMonth() returned: 12
--------------------------------------------------------------------------------
```

```
--------------------------------------------------------------------------------
Test 6: GetYear() Method
--------------------------------------------------------------------------------
Date created with year = 2023
GetYear() returned: 2023
--------------------------------------------------------------------------------
```

```
--------------------------------------------------------------------------------
Test 7: SetDate() Method
--------------------------------------------------------------------------------
Setting date to: 1/1/2025
Day: 1 Month: 1 Year: 2025
--------------------------------------------------------------------------------
```

```
--------------------------------------------------------------------------------
Test 8: Boundary Day Value
--------------------------------------------------------------------------------
Setting boundary day: 31
GetDay() returned: 31
--------------------------------------------------------------------------------
```

```
--------------------------------------------------------------------------------
Test 9: Boundary Month Value
--------------------------------------------------------------------------------
Setting boundary month: 12
GetMonth() returned: 12
--------------------------------------------------------------------------------
```

```
--------------------------------------------------------------------------------
Test 10: Single Digit String Parsing
--------------------------------------------------------------------------------
Parsing string: "5/3/2023"
Day: 5 Month: 3 Year: 2023
================================================================================
```

Time Class Testing

Time Test Plan

Test	Description	Actual test Data	Expected output	Pass/ fail
1	Check that the default constructor initializes the time with default values.	NA – Default constructor	Hour = 0, Minute = 0, Second = 0	Pass
2	Check that the string constructor initializes the time with specific values from HH:MM:SS format.	"14:30:45"	Hour = 14, Minute = 30, Second = 45	Pass
3	Check that getHour() returns the correct hour value.	Time("09:15:30")	getHour() returns 9	Pass
4	Check that getMin() returns the correct minute value.	Time("09:15:30")	getMin() returns 15	Pass
5	Check that getSec() returns the correct second value.	Time("09:15:30")	getSec() returns 30	Pass
6	Check that setTime() correctly updates all time components.	setTime(23, 59, 59)	Hour = 23, Minute = 59, Second = 59	Pass
7	Check that toString() returns correct formatted string.	Time(8, 5, 3)	toString() returns "08:05:03"	Pass
8	Check boundary hour value (23).	Time("23:00:00")	getHour() returns 23	Pass

9	Check boundary minute value (59).	Time("12:59:00")	getMin() returns 59	Pass
10	Check boundary second value (59).	Time("12:30:59")	getSec() returns 59	Pass

Time test results

```
--------------------------------------------------------------------------------
Test 1: Default Constructor
--------------------------------------------------------------------------------
Time object successfully created
Hour: 0 Minute: 0 Second: 0
--------------------------------------------------------------------------------
```

```
--------------------------------------------------------------------------------
Test 2: String Constructor
--------------------------------------------------------------------------------
Time object successfully created from string "14:30:45"
Hour: 14 Minute: 30 Second: 45
--------------------------------------------------------------------------------
```

```
--------------------------------------------------------------------------------
Test 3: getHour() Method
--------------------------------------------------------------------------------
Time created from "09:15:30"
getHour() returned: 9
--------------------------------------------------------------------------------
```

```
--------------------------------------------------------------------------------
Test 4: getMin() Method
--------------------------------------------------------------------------------
Time created from "09:15:30"
getMin() returned: 15
--------------------------------------------------------------------------------
```

```
--------------------------------------------------------------------------------
Test 5: getSec() Method
--------------------------------------------------------------------------------
Time created from "09:15:30"
getSec() returned: 30
--------------------------------------------------------------------------------
```

```
--------------------------------------------------------------------------------
Test 6: setTime() Method
--------------------------------------------------------------------------------
Setting time to: 23:59:59
Hour: 23 Minute: 59 Second: 59
--------------------------------------------------------------------------------
```

```
--------------------------------------------------------------------------
Test 7: toString() Method
--------------------------------------------------------------------------
Time(8, 5, 3) toString()
Result: "08:05:03"
--------------------------------------------------------------------------
```

```
--------------------------------------------------------------------------
Test 8: Boundary Hour Value
--------------------------------------------------------------------------
Time created with hour = 23
getHour() returned: 23
--------------------------------------------------------------------------
```

```
--------------------------------------------------------------------------
Test 9: Boundary Minute Value
--------------------------------------------------------------------------
Time created with minute = 59
getMin() returned: 59
--------------------------------------------------------------------------
```

```
--------------------------------------------------------------------------
Test 10: Boundary Second Value
--------------------------------------------------------------------------
Time created with second = 59
getSec() returned: 59
==========================================================================
```

Vector template Class testing

Vector test plan

Test	Description	Actual test data	Expected output	Pass/ fail
1	Check that the default constructor initialises the vector with default values.	NA – Default constructor	Size = 0, Capacity = 10	Pass
2	Check that the parameterized constructor initialises the vector with a specific capacity.	Capacity = 5	Size = 0, Capacity = 5	Pass
3	Check the push_back() operation and size tracking.	Push 10, 20, 30	Size after push = 3	Pass
4	Check the copy constructor for deep copy of vector elements.	Vector 1: push 5, 10; copy to Vector 2	Vector 1 and 2 are identical after copy	Pass
5	Check the assignment operator for deep copy of vector elements.	Vector 1: push 5, 10; assign to Vector 2	Vector 1 and 2 are identical after assignment	Pass
6	Check the resize() function when pushing more elements than the initial capacity.	Push 15 elements to capacity 10 vector	Capacity increases, Size = 15	Pass
7	Check that accessing vector elements by index works correctly.	Push 100, 200, 300; access vec[1]	Returns 200	Pass
8	Check the push_back() operations with	Push WeatherRecord objects	Size reflects added	Pass

	WeatherRecord objects.		WeatherRecord objects	
9	Check that vector of custom objects resizes correctly.	Push more WeatherRecord than initial capacity	Vector resizes and stores all objects	Pass
10	Check destructor properly deallocates memory.	Create vector, go out of scope	No memory leaks	Pass

Vector Test Results

```
--------------------------------------------------------------------------------
Test 1: Default Constructor
--------------------------------------------------------------------------------
Vector<int> object successfully created
Size: 0 Capacity: 10
--------------------------------------------------------------------------------
```

```
--------------------------------------------------------------------------------
Test 2: Parameterized Constructor
--------------------------------------------------------------------------------
Vector<int> object created with initial capacity = 5
Size: 0 Capacity: 5
--------------------------------------------------------------------------------
```

```
--------------------------------------------------------------------------------
Test 3: Push and Size
--------------------------------------------------------------------------------
Elements pushed: 10, 20, 30
Size after push: 3
--------------------------------------------------------------------------------
```

```
--------------------------------------------------------------------------------
Test 4: Copy Constructor
--------------------------------------------------------------------------------
Vector copied successfully
Original and copy have identical elements
--------------------------------------------------------------------------------
```

```
--------------------------------------------------------------------------------
Test 5: Assignment Operator
--------------------------------------------------------------------------------
Vector 1 and Vector 2 should now be identical.
vec1[0]: 5 vec2[0]: 5
--------------------------------------------------------------------------------
```

```
--------------------------------------------------------------------------
Test 6: Resize
--------------------------------------------------------------------------
Pushed 15 elements into the vector.
Size: 15 Capacity: 20
--------------------------------------------------------------------------
```

```
--------------------------------------------------------------------------
Test 7: Element Access
--------------------------------------------------------------------------
Pushed elements: 100, 200, 300
vec[1] = 200
--------------------------------------------------------------------------
```

```
--------------------------------------------------------------------------
Test 8: Push Date Objects
--------------------------------------------------------------------------
Dates pushed: Date(1,1,2000), Date(2,2,2021)
Size after push: 2
--------------------------------------------------------------------------
```

```
--------------------------------------------------------------------------
Test 9: Push Time Objects
--------------------------------------------------------------------------
Times pushed: Time(10,30), Time(12,0)
Size after push: 2
--------------------------------------------------------------------------
```

```
--------------------------------------------------------------------------
Test 10: Resize with Date Objects
--------------------------------------------------------------------------
Pushed 5 Date objects into vector with capacity 3.
Size: 5 Capacity: 6
==========================================================================
```

Main Testing Plan

Test	Description	Actual Test Data	Expected Output	Pass/ Fail
1	Test if "data_source.txt" was found	"data/data_source.txt"	Program continues as expected	Pass
2	Test if "data_source.txt" is missing	No file input	File not found	Pass
3	Test if the file in data_source.txt was found	"sample_data.csv"	Program continues as expected	Pass
4	Test if the file in data_source.txt was not found	Empty filename	File not found	Pass
5	Test if headers were read correctly	"sample_data.csv"	4 headers read	Pass
6	Test if data was loaded	"sample_data.csv"	Data lines read from file	Pass
7	Menu displays correctly	runMenu(weather_data);	Menu displays	Pass
8	Terminate menu when 5 is selected	Choice = 5	Program terminates	Pass
9	An invalid option is selected for main	Choice = 0	Menu redisplays and program continues	Pass
10	Option 1 is selected: month and year are in data	Choice = 1, Month = 3, Year = 2014	Wind speed statistics displayed	Pass
11	Option 1 is selected: month and year are not in data	Choice = 1, Month = 5, Year = 1990	No data for May 1990	Pass

12	Option 1 is selected: invalid month input	Choice = 1, Month = -1, Year = 2020	Invalid month handled gracefully	Fail
13	Option 1 is selected: invalid year input	Choice = 1, Month = 3, Year = -1	Invalid year handled gracefully	Fail
14	Option 2 is selected: Year in data	Choice = 2, Year = 2014	Temperature statistics for all months	Pass
15	Option 2 is selected: Year not in data	Choice = 2, Year = 1990	No data for all months	Pass
16	Option 2 is selected: invalid year input	Choice = 2, Year = -1	Invalid year handled gracefully	Fail
17	Option 3 is selected: Year in data	Choice = 3, Year = 2014	Solar radiation totals displayed	Pass
18	Option 3 is selected: Year not in data	Choice = 3, Year = 1990	No data for all months	Pass
19	Option 3 is selected: invalid year input	Choice = 3, Year = -1	Invalid year handled gracefully	Fail
20	Option 4 is selected: Year in data no missing values	Choice = 4, Year = 2014	CSV file created with complete data	Pass
21	Option 4 is selected: Year in data with missing values	Choice = 4, Year = 2024	CSV file created with missing values handled	Pass
22	Option 4 is selected: Year not in data	Choice = 4, Year = 1990	CSV file shows "No Data mate"	Pass

Main Test Results

```
================================================================================
Test 1
================================================================================
Reading data_source.txt... Source file: sample_data.csv
Extracting headers from data/sample_data.csv... 4 headers read from file
Extracting data lines from data/sample_data.csv... 3 read from file.
================================================================================
```

```
================================================================================
Test 2
================================================================================
Reading data_source.txt... File not found
Process returned -1 (0xFFFFFFFF)    execution time : 0.457 s
Press any key to continue.
================================================================================
```

```
================================================================================
Test 3
================================================================================
Reading data_source.txt... Source file: sample_data.csv
Extracting headers from data/sample_data.csv... 4 headers read from file
Extracting data lines from data/sample_data.csv... 3 read from file.
================================================================================
```

```
================================================================================
Test 4
================================================================================
Reading data_source.txt... Source file:
Extracting headers from data/... File not found
Process returned -1 (0xFFFFFFFF)    execution time : 0.414 s
================================================================================
```

```
================================================================================
Test 5
================================================================================
Extracting headers from data/sample_data.csv... 4 headers read from file
================================================================================
```

```
================================================================================
Test 6
================================================================================
Extracting data lines from data/sample_data.csv... 3 read from file.
================================================================================
```

```
Test 7
================================================================================
Weather Data Analysis Menu
1. Analyze Wind Speed Statistics
2. Analyze Temperature Statistics
3. Analyze Solar Radiation Statistics
4. Export Data to CSV
5. Exit
Enter your choice: ================================================================
Test 8
================================================================================
```

```
Test 8
================================================================================
Weather Data Analysis Menu
Enter your choice: 5
Exiting the program.
Process returned 0 (0x0)    execution time : 360.127 s
================================================================================
```

```
================================================================================
Test 9
================================================================================
Enter your choice: 0
Invalid option. Please choose a number between 1 and 5.
Weather Data Analysis Menu
================================================================================
```

```
================================================================================
 Test 10
 ================================================================================
 Display the wind speed data for a specific month and year.
 Enter the month: 3
 Enter the year: 2014
 March 2014:
     Average wind speed: 7.2 km/h
     Standard Deviation: 2.9
 ================================================================================
```

```
================================================================================
Test 11
================================================================================
Display the wind speed data for a specific month and year.
Enter the month: 5
Enter the year: 1990
May 1990: No Data
================================================================================
```

```
================================================================================
Test 12
================================================================================
Display the wind speed data for a specific month and year.
Enter the month: -1
Enter the year: 2020
Invalid month 2020: No Data
================================================================================
```

```
================================================================================
Test 13
================================================================================
Display the wind speed data for a specific month and year.
Enter the month: 3
Enter the year: -1
March -1: No Data
================================================================================
```

```
================================================================================
Test 14
================================================================================
Display the average ambient air temperature for each month in a specified year.
Enter the year: 2014
2014:
    January: No Data
    February: No Data
    March: average: 21.7 degrees C, stdev: 0.1
================================================================================
```

```
================================================================================
Test 15
================================================================================
Display the average ambient air temperature for each month in a specified year.
Enter the year: 1990
1990:
    January: No Data
    February: No Data
    March: No Data
================================================================================
```

```
================================================================================
Test 16
================================================================================
Display the average ambient air temperature for each month in a specified year.
Enter the year: -1
-1:
    January: No Data
    February: No Data
================================================================================
```

```
================================================================================
Test 17
================================================================================
Display the total solar radiation for each month in a specified year.
Enter the year: 2014
2014:
    January: No Data
    February: No Data
    March: 0.05 kWh/m^2
================================================================================
```

```
================================================================================
Test 18
================================================================================
Display the total solar radiation for each month in a specified year.
Enter the year: 1990
1990:
    January: No Data
    February: No Data
================================================================================
```

```
================================================================================
Test 19
================================================================================
Display the total solar radiation for each month in a specified year.
Enter the year: -1
-1:
    January: No Data
    February: No Data
================================================================================
```

```
================================================================================
Test 20
================================================================================
Export Data to CSV
Enter the year: 2014
Data exported to WindTempSolar.csv successfully.
File contents: 2014
March,7.2(2.9),21.7(0.1),0.05
================================================================================
```

```
================================================================================
Test 21
================================================================================
Export Data to CSV
Enter the year: 2024
Data exported to WindTempSolar.csv successfully.
File contents: 2024
================================================================================
```

```
==========================================================================
Test 22
==========================================================================
Export Data to CSV
Enter the year: 1990
Data exported to WindTempSolar.csv successfully.
File contents: 1990
No Data mate
==========================================================================
```