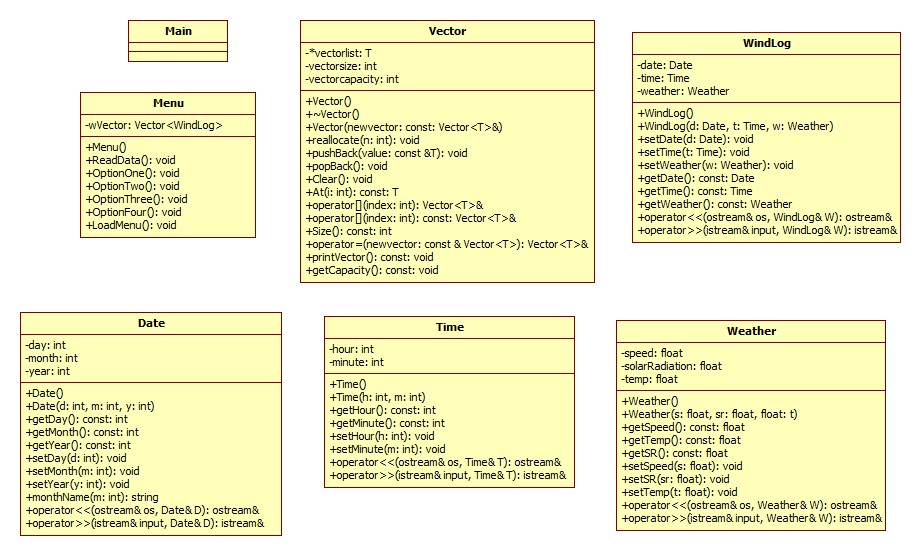
# **ICT283 Assignment 1 Documentation**

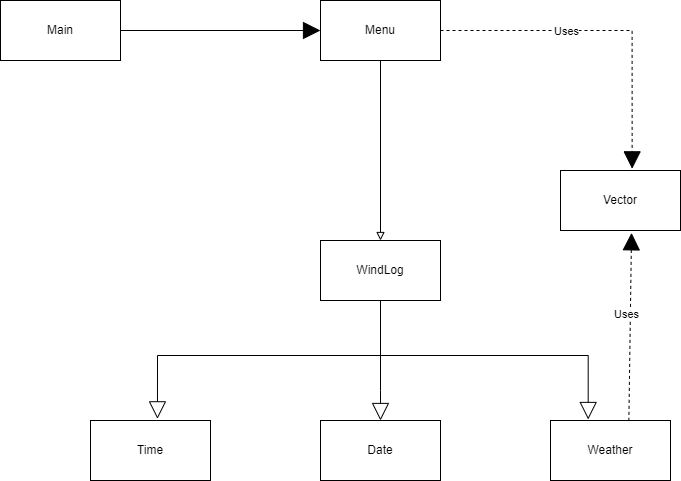
Name: Ang Jin Wei

Student Number: 34792195

Email: [34792195@student.murdoch.edu.au](mailto:34792195@student.murdoch.edu.au)

# **UML Diagram**





# **Data Dictionary**

**Vector**

| **Name** | **Type** | **Protection** | **Description** | **Rationale** |
| --- | --- | --- | --- | --- |
| Vector |  |  | The vector class, to create vectors, store data in vectors, modify data in vectors and retrieve data in vectors. | To create a vector of various classes to store information in. |
| Vector() | Constructor | + | Default constructor for Vector |  |
| Vector(const Vector<T> & newvector) | Constructor | + | Copy constructor for Vector |  |
| ~Vector() | Destructor | + | Destructor for Vector |  |
| reallocate(int n) | void | + | Reallocate size of vector where n = new size and capacity of vector. | To change the size and capacity of the vector so that it can contain more elements. |
| pushBack(const T& value) | void | + | Add element “value” to the back of the vector, reallocating vector size and capacity when needed. | So that new elements can be added to the vector. |
| popBack() | void | + | Remove element from the back of the vector. | So that an object can be removed from the vector. |
| Clear() | void | + | Clear the vector. |  |
| At(int n) | T | + | Get element where n = index from the vector. | So that we can access a particular element by the index. |
| operator[](int index) | Vector<T>& | + | Return element from index. | So that we can access a particular element by the index. |
| Size() | int | + | Return size of vector. |  |
| operator=(const Vector<T> & newvector) | Vector<T>& | + | = operator for Vector, copy a vector. |  |
| printVector() | void | + | Print the vector. |  |
| getCapacity() | void | + | Return the capacity of the vector. |  |
| vectorlist | T\* | - | The vector list. |  |
| vectorsize | int | - | The size of the vector. |  |
| vectorcapacity | int | - | The capacity of the vector. |  |

**Menu**

| **Name** | **Type** | **Protection** | **Description** | **Rationale** |
| --- | --- | --- | --- | --- |
| Menu |  |  | Provides the menu and functions for the program. | To provide a simple menu for the user and also to execute various functions required by the program. |
| Menu() | Constructor | + | Default constructor for Menu |  |
| ReadData() | void | + | Read data from data file and push data into a vector. |  |
| OptionOne() | void | + | Get user input on month and year and print out average windspeed and standard deviation. | Each option will only handle one requirement of the program, this is to ensure the ease of making changes in the future by sticking to an object-oriented program. |
| OptionTwo() | void | + | Get user input on year and print out average temperature and standard deviation. | Each option will only handle one requirement of the program, this is to ensure the ease of making changes in the future by sticking to an object-oriented program. |
| OptionThree() | void | + | Get user input on year and print out total solar radiation in kWh/m^2. | Each option will only handle one requirement of the program, this is to ensure the ease of making changes in the future by sticking to an object-oriented program. |
| OptionFour() | void | + | Get user input on year and output the data for the year to a CSV file. | Each option will only handle one requirement of the program, this is to ensure the ease of making changes in the future by sticking to an object-oriented program. |
| LoadMenu() | void | + | Load menu and get user input on menu option. | Each option will only handle one requirement of the program, this is to ensure the ease of making changes in the future by sticking to an object-oriented program. |
| wVector | Vector<WindLog> | - | A vector to store WindLog objects. | The use of vector here to store WindLog objects will enable ease of access to a large amount of data. |

**WindLog**

| **Name** | **Type** | **Protection** | **Description** | **Rationale** |
| --- | --- | --- | --- | --- |
| WindLog |  |  | Stores the data including the date, time and weather. | By having a separate class “WindLog”, encapsulation will be further enforced and it will be easier to make changes to the program in the future. |
| WindLog() | Constructor | + | Default constructor for WindLog. |  |
| WindLog(Date d, Time t, Weather w) | Constructor | + | Parametized constructor for WindLog. |  |
| setDate(Date d) | void | + | Set the date object for WindLog where d = date object. |  |
| setTime(Time t) | void | + | Set the time object for WindLog where t = time object. |  |
| setWeather(Weather w) | void | + | Set the weather object for WindLog where w = weather object. |  |
| getDate() | Date | + | Returns the date object. |  |
| getTime() | Time | + | Returns the time object. |  |
| getWeather() | Weather | + | Returns the weather object. |  |
| date | Date | - | Date object |  |
| time | Time | - | Time object |  |
| weather | Weather | - | Weather object |  |
| operator<<(ostream& os, const WindLog & W) | ostream& |  | Ostream operator for WindLog. |  |
| operator>>(istream&  input, WindLog & W) | istream& |  | Istream operator for WindLog. |  |

**Date**

| **Name** | **Type** | **Protection** | **Description** | **Rationale** |
| --- | --- | --- | --- | --- |
| Date |  |  | A date object that stores information of the date (day, month, year). | By having a Date object that solely handles the date aspect of the data will further enforce encapsulation and also provide ease of making changes to the program in the future. |
| Date() | Constructor | + | Default constructor for Date. |  |
| Date(int d, int m, int y) | Constructor | + | Parametized constructor for Date. |  |
| getDay() | int | + | Returns “day” of Date. |  |
| getMonth() | int | + | Returns “month” of Date. |  |
| getYear() | int | + | Returns “year” of Date. |  |
| setDay(int d) | void | + | Set “day” of Date where d = day. |  |
| setMonth(int m) | void | + | Set “month” of Date where m = month. |  |
| setYear(int y) | void | + | Set “year” of Date where y = year. |  |
| monthName(int m) | string | + | Returns month name in alphabets where m = month in numbers. | This is to fulfil the requirement of the program where the month should be printed out in alphabets instead of numbers. |
| day | int | - | The day of the date. |  |
| month | int | - | The month of the date. |  |
| year | int | - | The year of the date. |  |
| operator<<(ostream & os, Date & D) | ostream& |  | Ostream operator for Date. |  |
| operator>>(istream & input, Date & D) | istream& |  | Istream operator for Date. |  |

**Time**

| **Name** | **Type** | **Protection** | **Description** | **Rationale** |
| --- | --- | --- | --- | --- |
| Time |  |  | A time object that stores information on the time (hours and minutes) | By having a Time object that solely handles the time aspect of the data will further enforce encapsulation and also provide ease of making changes to the program in the future. |
| Time() | Constructor | + | Default constructor for Time. |  |
| Time(int h, int m) | Constructor | + | Parametized constructor for Time. |  |
| getHour() | int | + | Returns “hour” of Time. |  |
| getMinute() | int | + | Returns “minute” of Time. |  |
| setHour(int h) | void | + | Set “hour” of Time where h = hour. |  |
| setMinute(int m) | void | + | Set “minute” of Time where m = minute. |  |
| hour | int | - | The hour of the Time. |  |
| minute | int | - | The minute of the Time. |  |
| operator<<(ostream& os, Time& T) | ostream& |  | Ostream operator for Time. |  |
| operator>>(istream& input, Time& T) | istream& |  | Istream operator for Time. |  |

**Weather**

| **Name** | **Type** | **Protection** | **Description** | **Rationale** |
| --- | --- | --- | --- | --- |
| Weather |  |  | A weather object that stores information on the weather (windspeed, solar radiation, temperature) | By having a Weather object that solely handles the weather aspect of the data will further enforce encapsulation and also provide ease of making changes to the program in the future. |
| Weather() | Constructor | + | Default constructor for weather. |  |
| Weather(float s, float sr, float t) | Constructor | + | Parametized constructor for weather. |  |
| getSpeed() | float | + | Returns the “speed” of the weather. |  |
| getTemp() | float | + | Returns the “temp” of the weather. |  |
| getSR() | float | + | Returns the “solarRadiation” of the weather. |  |
| setSpeed(float s) | void | + | Set the “speed” of the weather where s = speed. |  |
| setSR(float sr) | void | + | Set the “solarRadiation” of the weather where sr = solarRadiation. |  |
| setTemp(float t) | void | + | Set the “temp” of the weather where t = temp. |  |
| speed | float | - | The wind speed. |  |
| solarRadiation | float | - | The solar radiation. |  |
| temp | float | - | The temperature. |  |
| operator<<(ostream& os, const Weather & W) | ostream& |  | Ostream operator for Weather. |  |
| operator>>(istream& input, Weather & W) | istream& |  | Istream operator for Weather. |  |

# **Algorithm**

**Main**

1. Declare main function.
2. Initialize Menu object.
3. Run ReadData() function from Menu to read the data.
4. Run LoadMenu() function from Menu to load the menu.

**Menu**

**ReadData()**

1. Set a DATA\_PATH where the data\_source text file is.
2. Initialize input stream.
3. Get the data file name from DATA\_PATH.
4. Open the data file using the data file name.
5. Skip the first line of the data file as it is the header.
6. Read the data file line-by-line and push the data into a WindLog object.
7. Push the WindLog object into a vector.

**OptionOne()**

1. Get user input on a month and year, validating the month using a while loop where the month should be between 1 and 12.
2. Loop through the vector to find data matching the month and year given by the user.
3. If the month and year match, add 1 to count and add the speed in that particular data to the sum of speed.
4. Get average speed by dividing the sum of the speed by the count.
5. Loop through the vector to find data matching the month and year given by the user again.
6. Calculate the standard deviation by square rooting the square of the deviation of all speed.
7. Display the average speed and standard deviation of the user’s selected month and year.
8. Display no data if there’s no data matched in the loop.

**OptionTwo()**

1. Get user input on the year.
2. Loop through 1 to 12 for the months.
3. For every month, loop through the vector to find data matching the year given by the user.
4. If data matches, add the temperature to the sum and add 1 to count.
5. Calculate the average by dividing the sum by the count.
6. Loop through the vector again to calculate the standard deviation.
7. For each loop of the month, print out the month name, the average temperature of the month and the standard deviation.
8. If there is no data for the month, print no data.
9. Repeat for all 12 loops of the month.

**OptionThree()**

1. Get user input on the year.
2. Loop through 1 to 12 for the months.
3. For every month, loop through the vector to find data matching the year given by the user.
4. If data matches, and if the solar radiation is over 100 W/m^2, add the solar radiation to the sum and add 1 to count.
5. Convert the sum of the solar radiation to kWh/m^2 by dividing it by 6, then dividing it by 1000.
6. For every loop of the month, display the name of the month and the total solar radiation in kWh/m^2.
7. If there is no data for the given month, display no data.

**OptionFour()**

1. Initialize and define an OUTPUT\_PATH.
2. Initialize an output stream.
3. Get user input on the year.
4. Open the file defined as the OUTPUT\_PATH.
5. Output the year given by the user to the file.
6. Loop through 1 to 12 for the months.
7. For every month, loop through the vector to find data matching the year given by the user and the month matching the loop.
8. Check if solar radiation is over 100, if it is, add it to the sum of Solar Radiation.
9. Add matching speed and temperature to the sum of speed and the sum of temperature.
10. If sum of speed, sum of temp and sum of solar radiation is more than 0, add 1 to year count to indicate that there’s data for the year.
11. If year count is more than 0, loop through 1 to 12 for the months again.
12. Loop through the vector to find matching data according to user given year and corresponding month of the loop.
13. Add up all the speed and temperature of matching data and adding 1 to count for every matching data.
14. Get average of speed, average of temperature by dividing them by the count.
15. Loop through the vector again and add the solar radiation of matching data to the sum of solar radiation if the solar radiation is over 100 W/m^2.
16. Add the square of the deviation of speed to speedSD.
17. Add the square of the deviation of temp to tempSD.
18. Calculate the standard deviation of speed and temperature by square rooting speedSD and tempSD.
19. Convert the solar radiation to kWh/m^2 by dividing it by 6 then dividing it by 1000.
20. If count is over 0, output the month’s name and check if certain data is missing.
21. If the data is missing, output nothing, if the data is existing, output the data.
22. Lastly, if the year count is 0, output No Data.
23. Display message to user stating the data is successfully output to the OUTPUT\_PATH.

**LoadMenu()**

1. Initialize choice = 0.
2. Display the menu options.
3. While loop to check for choice not equal to 5.
4. Run the option corresponding to the user’s input.
5. If choice is not one of the case (1,2,3,4,5), display Invalid option.
6. If choice is 5, display a farewell message and exit the program.

**Date**

**monthName(int m)**

1. Switch case (m)
2. Return the corresponding month in alphabets according to m.

**Weather**

**Istream operator >>**

1. Define the NUM\_OF\_COLUMNS in the data file excluding the date/time.
2. Define the indexes of the columns you want to access.
3. For loop, looping through the columns.
4. Push data of each line into a vector.
5. Set the elements using the vector and the indexes defined earlier on.

**Vector**

**reallocate(int n)**

1. Define and initialize a new array.
2. Check if n is smaller than the current vector size.
3. If n is smaller than the current vector size, set vector size to n.
4. For loop through the vector size.
5. Set element in the corresponding index to the new array.
6. Delete the current vector list.
7. Set vector list to new array.
8. Set capacity of vector to n.

**pushBack(const T& value)**

1. Check if vector size is more than vector capacity.
2. If so, run reallocate where n is vector capacity + (vector capacity/2) to increase the capacity to fit the new element.
3. Add the element to the back of the vector.
4. Add 1 to the size of vector.

**operator= (const Vector<T>& newvector)**

1. Check if current vector is not equal to newvector.
2. Set vector size to newvector’s size.
3. Set vector capacity to newvector’s capacity.
4. Initialize a new array.
5. Loop through a for loop according to the vector size.
6. Add each element of newvector according to the index to vector.

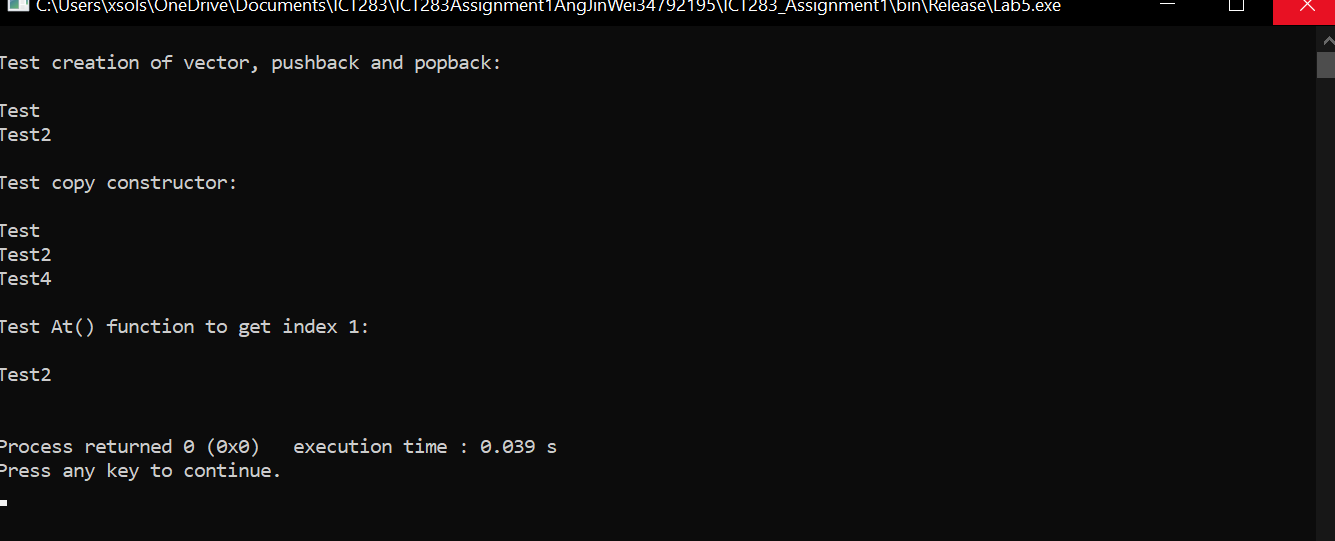
# 

# **Test-Plan**

| **Test ID** | **Description** | **Test Data** | **Expected Output** | **Pass/Fail** |
| --- | --- | --- | --- | --- |
| 1 | Test initializing of a new vector. | Vector<string> newvector | - | Pass |
| 2 | Test pushBack() function of vector. | newvector.pushBack("Test");  newvector.pushBack("Test2");  newvector.pushBack("Test3"); | Test,Test2,Test3 to be pushed to newvector. | Pass |
| 3 | Test popBack() function of vector. | newvector.popBack() | Test3 should be removed from newvector. | Pass |
| 4 | Test copy constructor of vector | Vector<string> copyvector = Vector<string>(newvector); | - | Pass |
| 5 | Test pushBack() function of copied vector and At() function. | copyvector.pushBack("Test4");  copyvector.At(1) | Test4 should be pushed to copyvector only.  Test2 should be displayed. | Pass |
| 6 | Test reading of data from data\_source.txt | string const DATA\_PATH = "Data/data\_source.txt"; | Data should be read and menu should be successfully displayed | Pass |
| 7 | Test getting average speed and standard deviation of data. | Speed = 5  Speed = 7 | Average should be 12/2 = 6 and standard deviation should be 1. | Pass |
| 8 | Test getting average temperature and standard deviation of data. | Temp = 10  Temp = 15  Temp = 5  In April 2014. | Average should be 30 / 3 = 10 and SD should be 4.082. | Pass |
| 9 | Test calculating total Solar Radiation in kWh/m^2 | SR = 123  SR = 98  In May 2014. | Total SR should be (123/6)/1000 = 0.0205 kWh/m^2 because SR below 100 should be ignored from the calculation. | Pass |
| 10 | Test if output successfully to WindTempSolar.csv. | Using MetData\_Mar01-2014-Mar01-2015-ALL.csv file. | Should be successfully output with the output requirements stated in the assignment. | Pass |
| 11 | Test if OptionOne() only shows data for the year and month specified. | Using MetData\_Mar01-2014-Mar01-2015-ALL.csv file.  May 2014. | It will only display average speed and standard deviation for results in May 2014. | Pass |
| 12 | Test if OptionTwo() shows all the data for each month in the year specified. | Using MetData\_Mar01-2014-Mar01-2015-ALL.csv file.  2014. | It will display all the average temperature and standard deviation for results in 2014 for each month. | Pass |
| 13 | Test if OptionThree() only show data for each month for the year specified. | Using MetData\_Mar01-2014-Mar01-2015-ALL.csv file.  2015. | Should only display three months of data in 2015. | Pass |
| 14 | Test if OptionFour() only output data for the given year. | Using MetData\_Mar01-2014-Mar01-2015-ALL.csv file.  2015. | Should only output three months of data in 2015. | Pass |

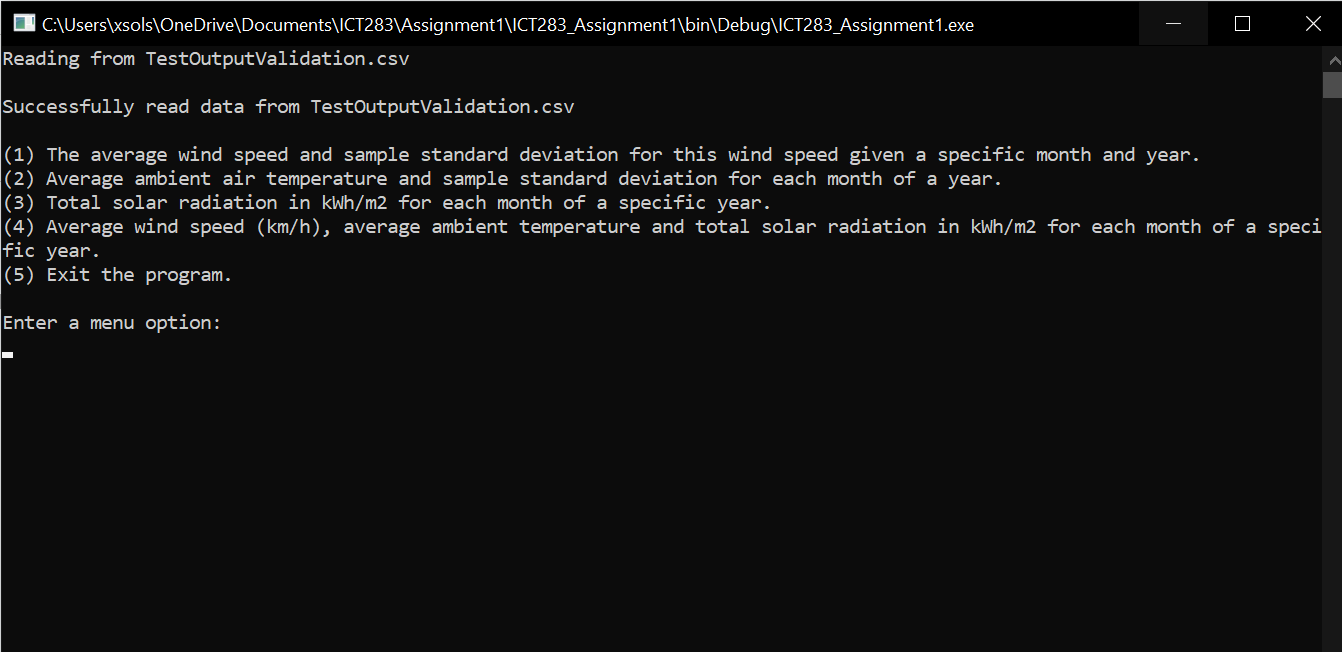
# **Test Plan Outputs**

**Test ID 1,2,3,4,5**

****

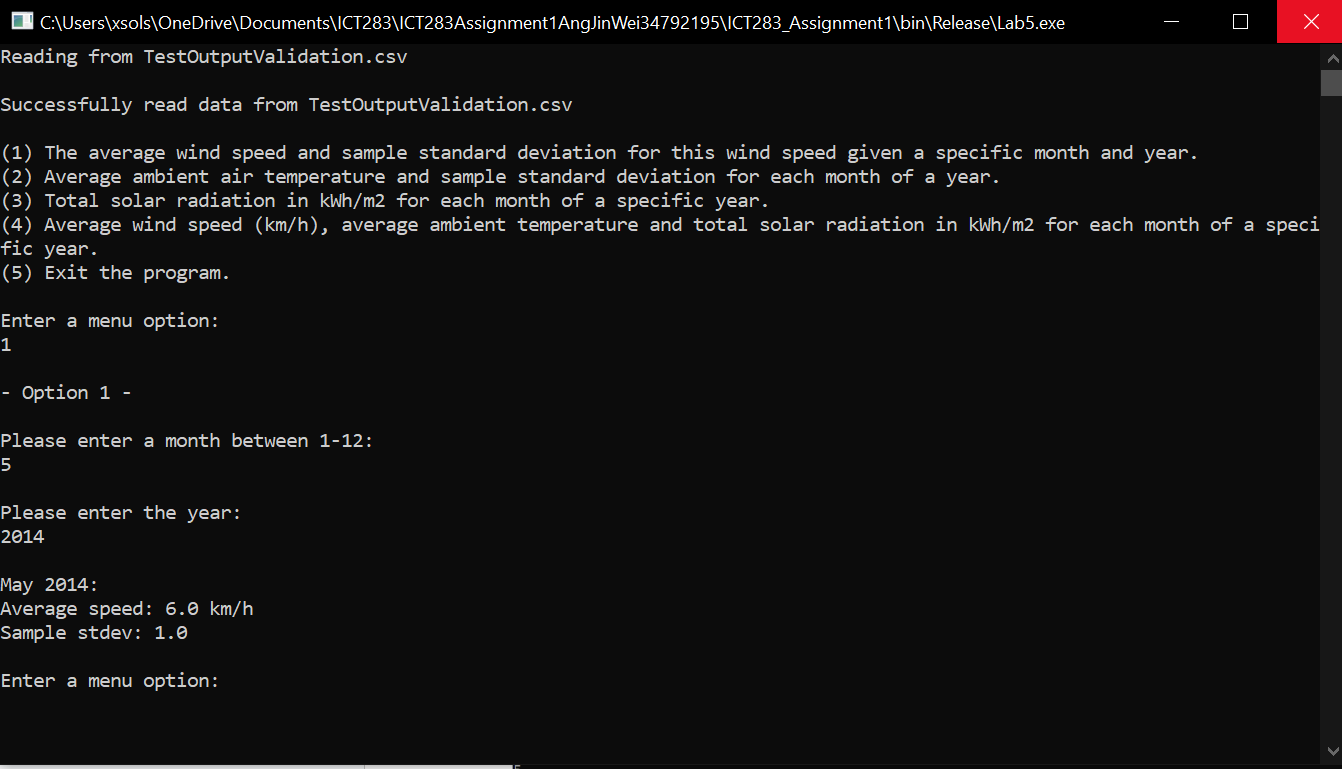
Newvector created, 3 elements pushed to the vector, last element (Test3) removed. Copy constructor works. At(1) successfully get index 1 of vector.

**Test ID 6**

****

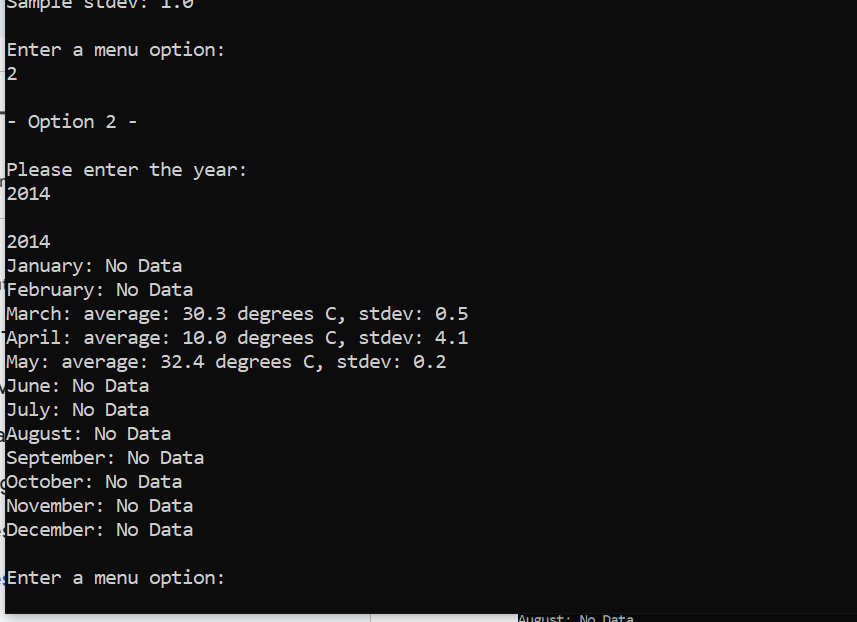
Data successfully read and menu is displayed.

**Test ID 7**

****

Average speed and standard deviation successfully calculated.

**Test ID 8**

****

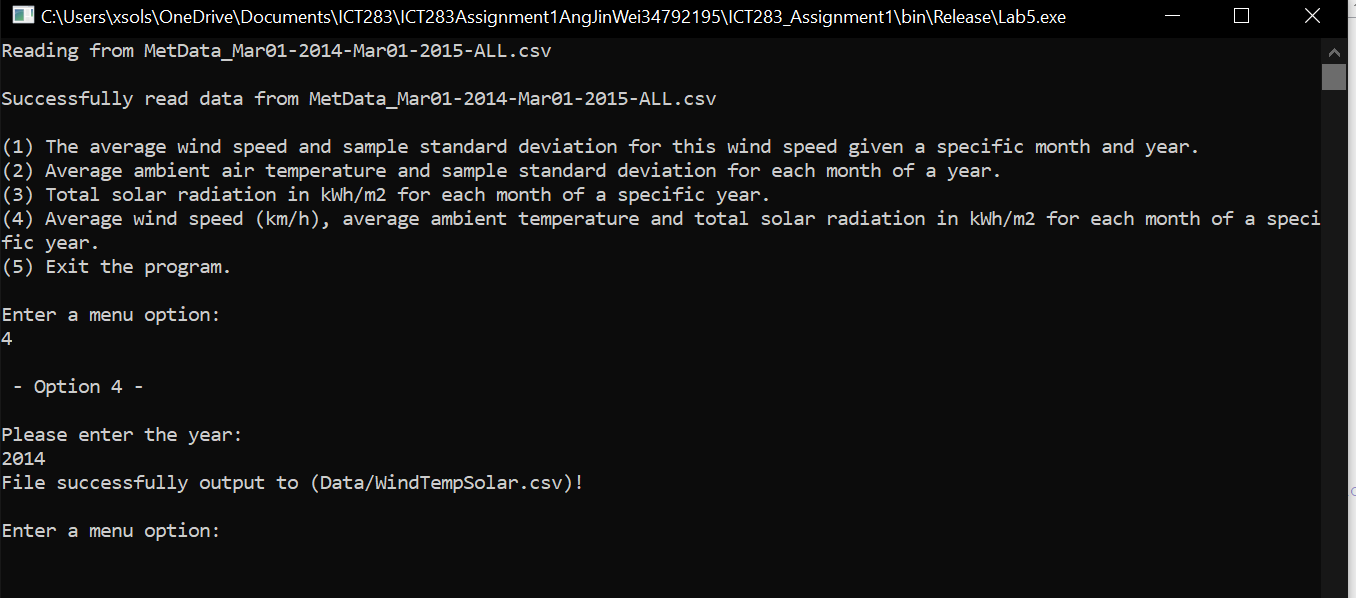
Average temp and standard deviation successfully calculated.

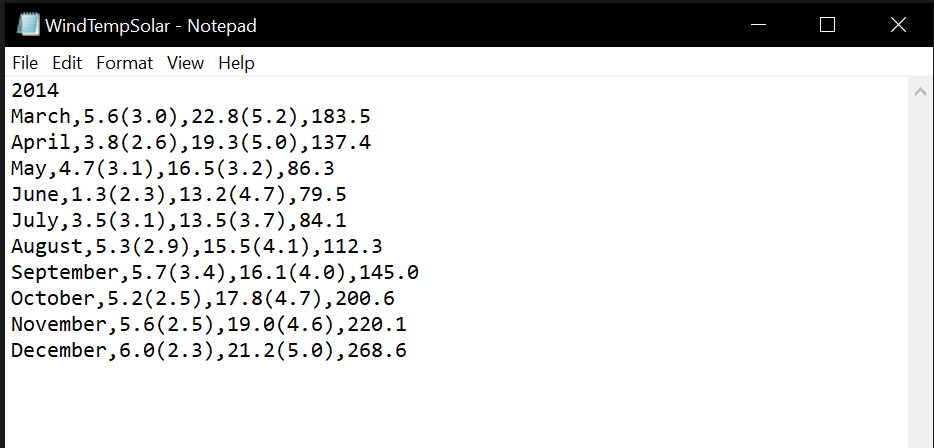
**Test ID 9**

****

Solar Radiation successfully calculated and displayed as kWh/m^2.

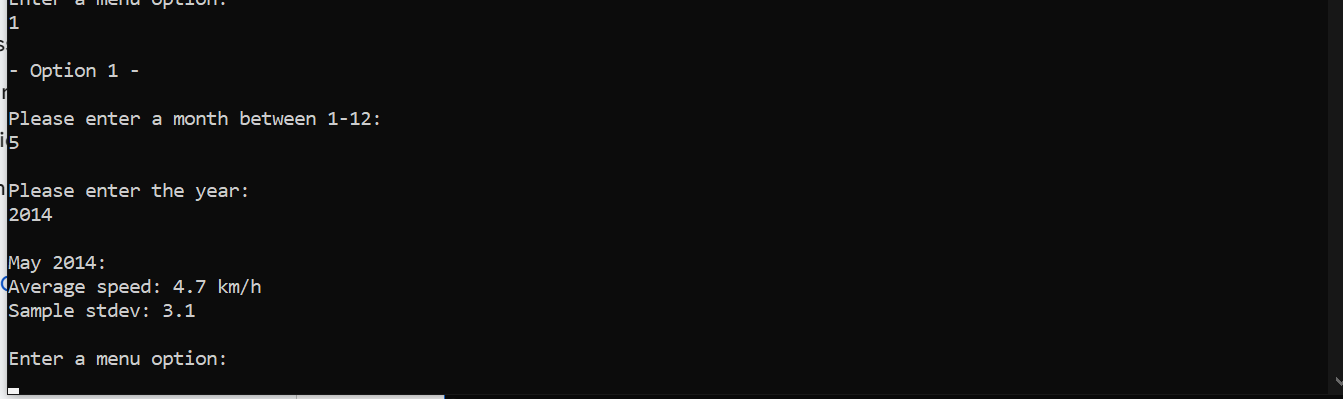
**Test ID 10**

****

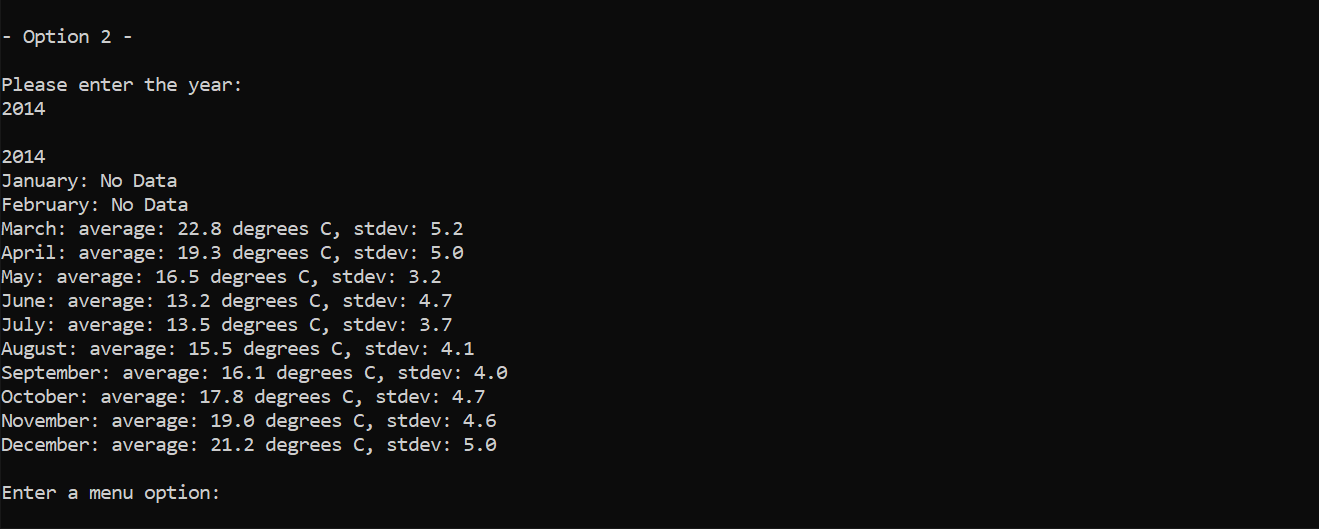
****

Data successfully output according to the requirements in the assignment.

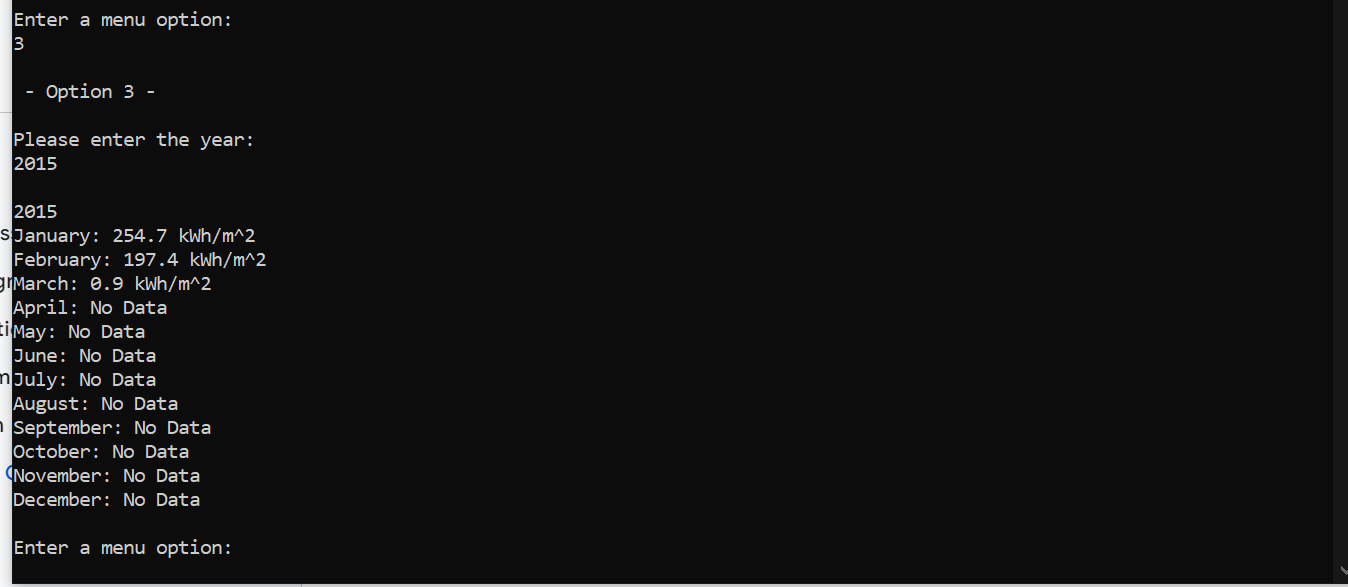
**Test ID 11**

****

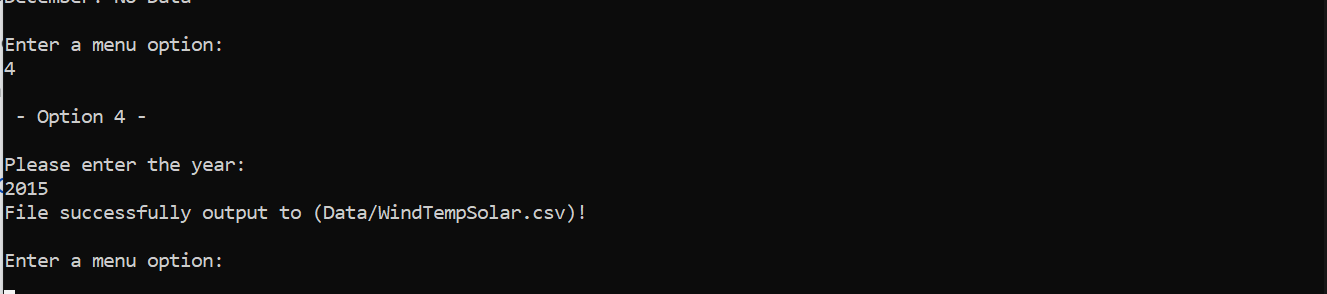
**Test ID 12**

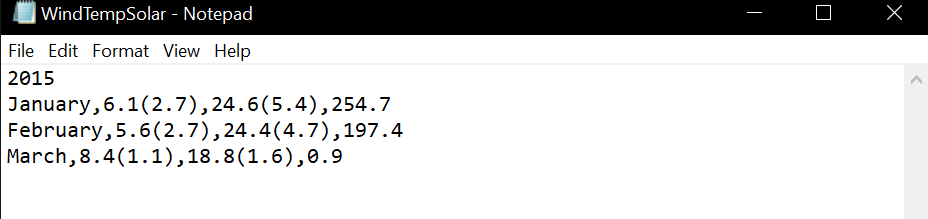
****

**Test ID 13**

****

**Test ID 14**

****

****

# 