Test Plan

We understand that a working code, is not only a program that compiles and display results on the console.

To prove that a program works, we would have to exhaustive test the application and also manually calculate the results to check if you could rely in its accuracy.

The assignment required us to create a program that would read multiple weather data .csv files, and would display a menu interface which a user is able to insert dates and get back a whole set of results based on the matching of the date given by the user and the set of data analysed.

So, among other requirements, to determine if the code is fully working the user can run the program which would read multiple files, and have the following options displayed:

1 - max wind speed for year and month
2 - Average wind speed for year
3 - Total solar radiation for year
4 - Average wind speed + solar radiation for year
5 - Max solar radiation in a day
6 - exit program

The user then chose an option, insert a date and get back the result. However, because of the magnitude of the files, each containing over 50000 lines, it would be nearly impossible to really check manually if the result that was given by the program was accurate.

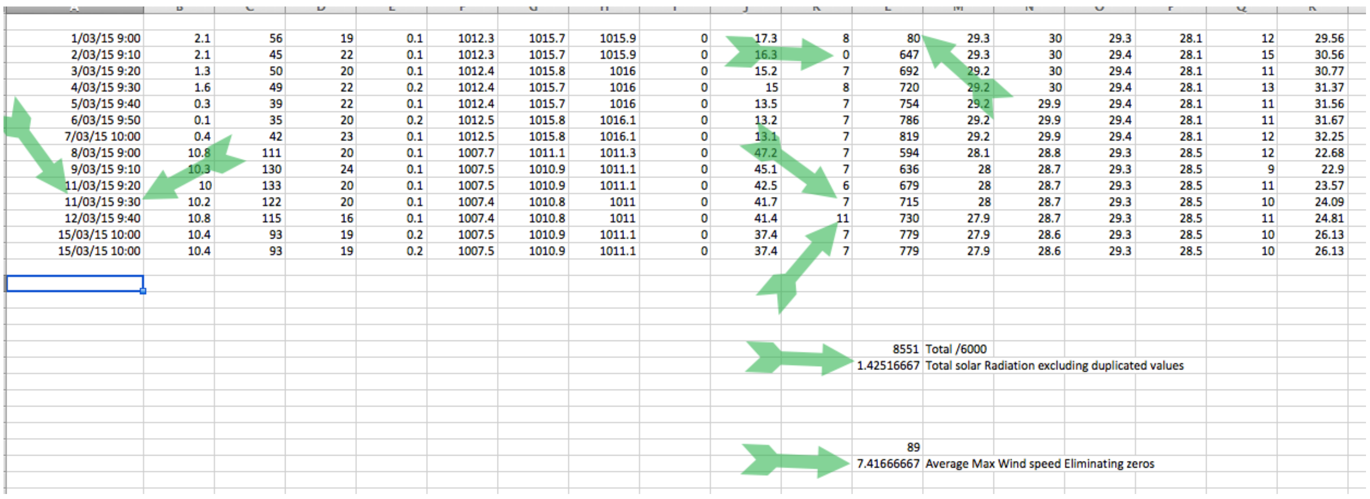
Testing Strategy

So my first approach of testing was to create a smaller version of the weather data file, with less register, so I could manually check the results on an excel spreadsheet and compare to results given by the application.

In order to have an accurate result some cases should be regarded, for example:

- registers with value zero should not be counted
- repeated registers should not be counted.
- Solar radiation less than 100 should not be counted
- The max solar radiation for a day should not have repeated times

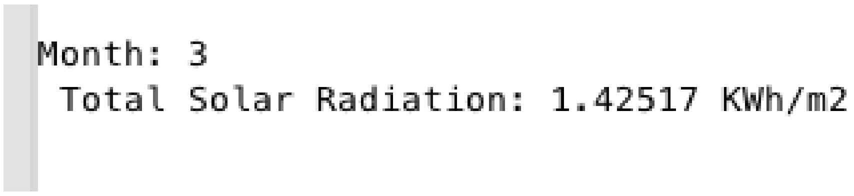
All these conditions were successful tested in a smaller version of the weather data file, as you can check below.



A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
1/03/15 9:00	2.1	56	19	0.1	1012.3	1015.7	1015.9	0	17.3	8	80	29.3	30	29.3	28.1	12	29.56
2/03/15 9:10	2.1	45	22	0.1	1012.3	1015.7	1015.9	0	16.3	0	647	29.3	30	29.4	28.1	15	30.56
3/03/15 9:20	1.3	50	20	0.1	1012.4	1015.8	1016	0	15.2	7	692	29.2	30	29.4	28.1	11	30.77
4/03/15 9:30	1.6	49	22	0.2	1012.4	1015.7	1016	0	15	8	720	29.2	30	29.4	28.1	13	31.37
5/03/15 9:40	0.3	39	22	0.1	1012.4	1015.7	1016	0	13.5	7	754	29.2	29.9	29.4	28.1	11	31.56
6/03/15 9:50	0.1	35	20	0.2	1012.5	1015.8	1016.1	0	13.2	7	786	29.2	29.9	29.4	28.1	11	31.67
7/03/15 10:00	0.4	42	23	0.1	1012.5	1015.8	1016.1	0	13.1	7	819	29.2	29.9	29.4	28.1	12	32.25
8/03/15 9:00	10.8	111	20	0.1	1007.7	1011.1	1011.3	0	47.2	7	594	28.1	28.8	29.3	28.5	12	22.68
9/03/15 9:10	10.3	130	24	0.1	1007.5	1010.9	1011.1	0	45.1	7	636	28	28.7	29.3	28.5	9	22.9
11/03/15 9:20	10	133	20	0.1	1007.5	1010.9	1011.1	0	42.5	6	679	28	28.7	29.3	28.5	11	23.57
11/03/15 9:30	10.2	122	20	0.1	1007.4	1010.8	1011	0	41.7	7	715	28	28.7	29.3	28.5	10	24.09
12/03/15 9:40	10.8	115	16	0.1	1007.4	1010.8	1011	0	41.4	11	730	27.9	28.7	29.3	28.5	11	24.81
15/03/15 10:00	10.4	93	19	0.2	1007.5	1010.9	1011.1	0	37.4	7	779	27.9	28.6	29.3	28.5	10	26.13
15/03/15 10:00	10.4	93	19	0.2	1007.5	1010.9	1011.1	0	37.4	7	779	27.9	28.6	29.3	28.5	10	26.13
											8551	Total /6000					
											1.42516667	Total solar Radiation excluding duplicated values					
											89						
											7.41666667	Average Max Wind speed Eliminating zeros					

The .csv file above represent the data collected from two different files, as you can see the last register which was repeated was excluded from the calculation.

Also the first line has a solar radiation of 80, so under 100, hence not used in the calculation of the solar radiation variation.

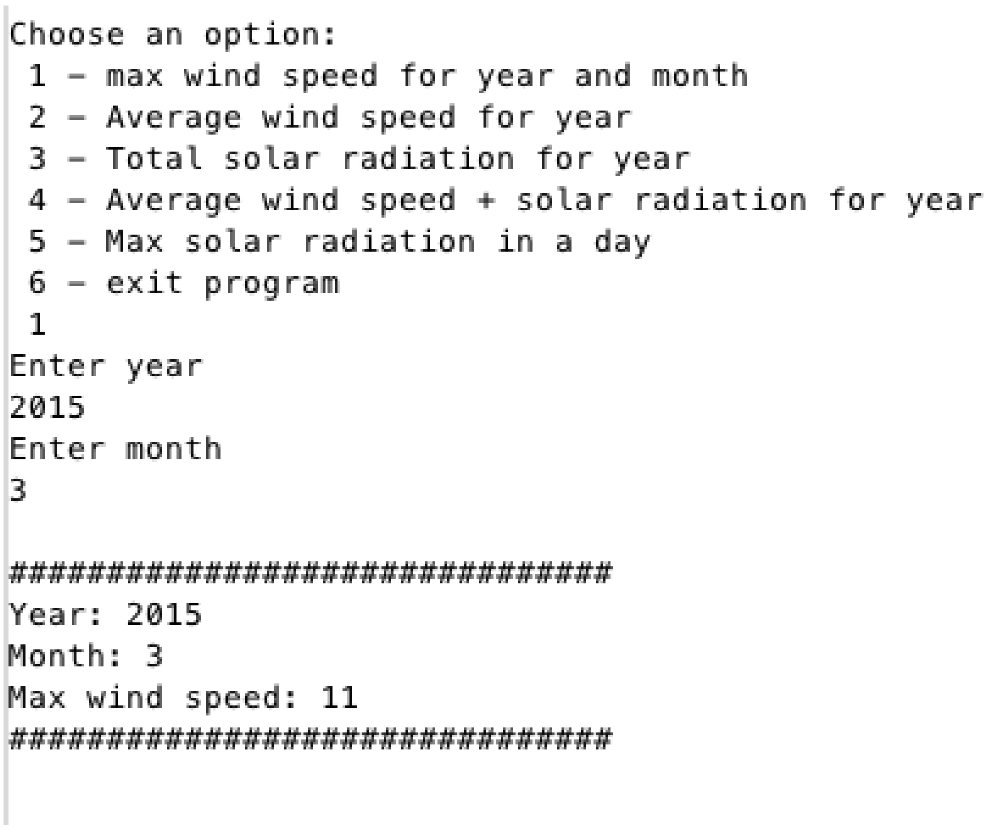


```
Month: 3
 Total Solar Radiation: 1.42517 KWh/m2
```

We can see that the program's has the same result has we manually tested on excel spreadsheet.

In the second line we have a zero wind speed, so that line was excluded from the calculation of average wind speed.

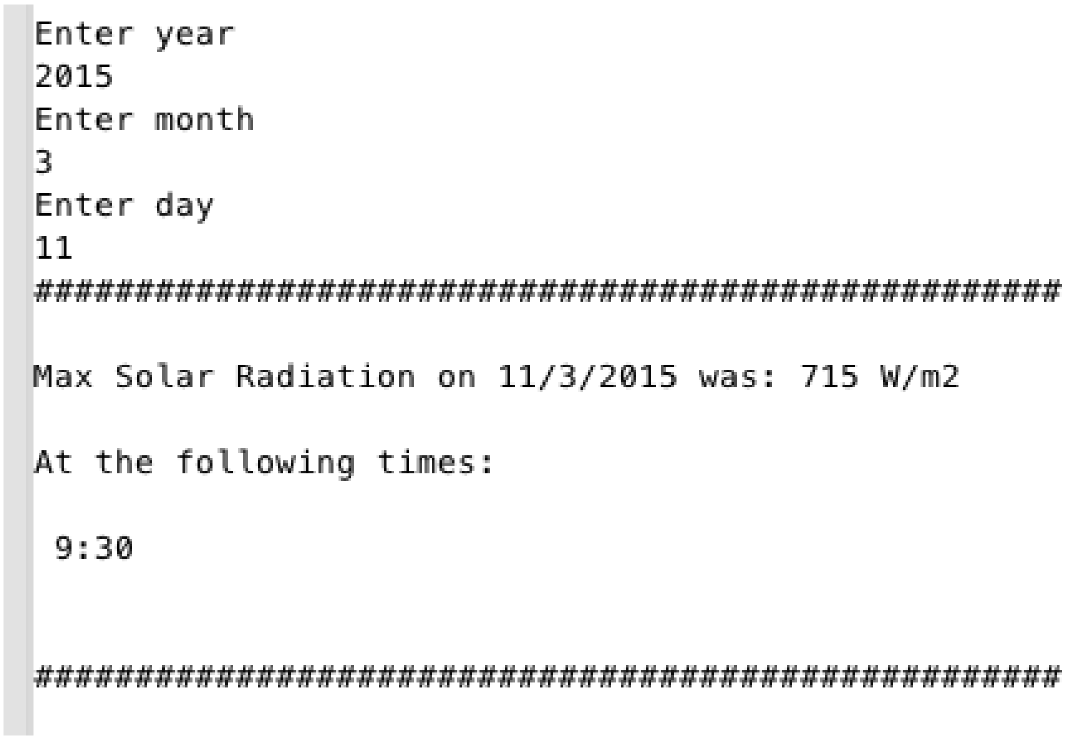
Again the program picked the expected value of 11.



```
Choose an option:
 1 - max wind speed for year and month
 2 - Average wind speed for year
 3 - Total solar radiation for year
 4 - Average wind speed + solar radiation for year
 5 - Max solar radiation in a day
 6 - exit program
 1
Enter year
2015
Enter month
3

###############################
Year: 2015
Month: 3
Max wind speed: 11
###############################
```

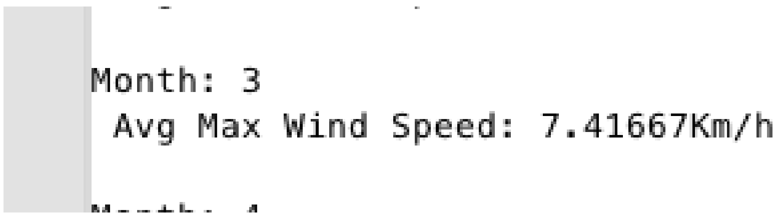
In this file we can see that 11 was the highest wind speed in the month.

In the option 5 of the menu where we have to check the highest wind speed for a date which includes day, we can see that when we have 2 days the same like in this case 11/03/2015, the program still gives the expected result.



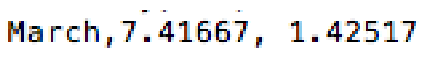
```
Enter year
2015
Enter month
3
Enter day
11
#####################################################

Max Solar Radiation on 11/3/2015 was: 715 W/m2

At the following times:

 9:30


#####################################################
```

The average didn’t count zeros neither duplicates and printed the expected result.



```
Month: 3
 Avg Max Wind Speed: 7.41667Km/h
```

The success of the results extended to the option 4 which we had to print the results on a file.



```
March,7.41667, 1.42517
```

So apart from manually testing the results, we have to make sure that all methods of our classes are tested in classes designed for this end.

Another important part of the testing is the validation of the data insert by the user, despite the importance of this set of tests, I neglected them purely because of lack of time. So, in order to the program to run properly we expect the user to type numbers that make sense, instead of weird characters.

Below is a list of all classes and methods that should be tested beforehand and the respective results of the testing process.

Name	Type	Description	Pass/ Fail
Vector		Array that stores unlimited number of a generic type	□
List*	elemType	This is a list of a certain generic type	□
Length	Int	This is the current size of the array Vector	□
maxSize	Int	This is the max size of the array Vector	□
Vector(int)	Vector	Standard constructor of Vector takes a interger as paramenter	□
Vector	Vector	Vector destructor	□
insertEnd(const elemType)	elemType	Insert a generic element to the end of the Vector	□
retrieveAt(int, elemType)	elemType	Gets a index position and return the element of the array Vector	□

Resize(int)	void	Function that doubles the size of vector when it reaches it capacity	▯
Metadata		Class that works as an object to hold the variables of the csv lines	▯
m_windSpeed	Double	Holds the wind speed max for a specific register	▯
M_solarRad	Double	Holds the solar radiation registered for a sigle register	▯

M_dateObject	Date	-	Object of type date that will be to hold a date while a read the file csv	▯
Metadata()	Metadata	+	Constructor for metadata, don't have paramenters, used in case just need to create an object without passing the values	▯
Metadata(Date, Time, double, double)	Metadata	+	Constructor that holds parameters	▯
getWindSpeed()	Double	+	Function that returns wind speed for a specific register	▯
getSolarRad()	Double	+	Function that returns solar radiation from a register	▯
getDate()	Date	+	Returns a date of a weather register	▯
getTime()	Time	+	Returns time of a specific weather event	▯
setWindSpeed(double)	Void	+	Set the wind speed of a register Metadata	▯
setSolarRad(double)	Void	+	Set solar radiation of a metadata register	▯
setDate(Date)	Void	+	Set the date of a metadata register	▯
setTime(Time)	Void	+	Set the time of a metadata Register	▯
Time			Class that collects time of a metadata register	▯
M_hour	Int	-	Variable that holds the hour of a register	▯
M_minute	Int	-	Variable that holds the minutes of a register	▯
Time()	Time	+	Default constructor of class Time	▯

Time(int, int)	Time	+	Constructor of Time object that takes parameters	▯
getHour()	Int	+	Returns an hour of an event	▯
getMinute()	Int	+	Returns a minute of an event	▯
setHour(int)	Void	+	Set an hour of a register	▯
setMinute(int)	Void	+	Set the minute of a regiter	▯
Date			Class date that gets a set of day, month and year	▯
M_day	Int	-	Day variable is the day of a register	▯
M_month	Int	-	Month of a register	▯
M_year	Int	-	Year of a register metadata	▯
Date()	Date	+	Default constructor of Date object	▯
Date(int, int, int)	Date	+	Date constructor that takes parameters	▯
getDay()	Int	+	Function that returns the day of a register	▯
getMonth()	Int	+	Function that returns the month of an event	▯
getYear()	Int	+	Return the year	▯
setDay(int)	Void	+	Set the day of a specific register	▯
setMonth(int)	Void	+	Set the month of a metadata register	▯
setYear(int)	Void	+	Set the year of a register	▯