

# **Mechanical Engineering Case Study**

## **Flight Wind Speed Simulator**



**Dhiyaa Al Jorf**

**da2863**

ENGR-UH 1000 Computer Programming for Engineers  
Assignment 1

## Step 1 - Problem Identification and Statement:

The objective is to develop a program that can simulate wind speeds to be used in a flight simulator. The program should include noise and be able to randomly generate storms and microbursts in the simulation. The user should be able to adjust the speed, duration, and probability values with ease.

## Step 2 – Gathering Information & Input/Output Description:

The software will be given 1 configuration file and must generate 4 files according to the following steps:

1. The software assumes a file exists with the simulation configuration. The file, named “simulationConfiguration.txt”, should store the parameters for the wind simulation. The first line stores the wind simulation configurations, namely the average speed, gust value, duration, and step size). The second line stores the storm simulation parameters, namely the storm probability, the minimum and maximum storm amplitude, and the minimum and maximum storm duration values. The third line stores the microburst parameters, namely the microburst probability, the minimum and maximum burst amplitude, and the minimum and maximum microburst duration values. Here is the structure of the configuration file:

Average Wind Speed, gust value, simulation duration, step size  
Storm probability (Pstorm), min and max storm amplitude, and min and max storm duration  
Microburst probability (Pburst), min and max microburst amplitude, min and max microburst duration

2. The program reads the average wind speed, the gust value, the duration and step size of the wind speed simulation from the “simulationConfiguration.txt” file (first line). The program should generate a data file named “WindSpeedData.txt” that contains the simulated wind speeds. The time should always start with 0 seconds. Each line of the “WindSpeedData.txt” file should contain the time in seconds and the corresponding wind speed. The wind speed is a random number between the (average speed + gust value) and the (average speed - gust value).

3. The wind speed simulation must include storm simulation. Assuming that the possibility of encountering a storm at each time step is Pstorm, the program must add a storm magnitude (a random number between the minimum and maximum storm amplitude values) to the simulated wind speed for a duration T (a random number between the minimum and maximum storm duration). The storm parameters are read from the “simulationConfiguration.txt” configuration file (second line). The storm simulation data must be stored in the file “StormData.txt”. The file “StormData.txt” stores the storm magnitude and not the sum of the storm magnitude and wind speed. Adding the files is done in the last step of the program.

4. The storm simulation must also include a simulation of microbursts. Microburst are simulated using three parameters: the possibility of encountering a microburst at each

time step in a storm (Pburst), the burst amplitude (a random number between the minimum and maximum microburst amplitude values), and the duration of the burst (a random number between the minimum and maximum storm duration values). The burst parameters are read from the “simulationConfiguration.txt” configuration file (third line). The microburst data must be stored in the file “BurstData.txt”. Similarly to the previous step, the file “BurstData.txt” stores the burst magnitude and not the sum of the burst magnitude, storm magnitude, and wind speed.

5. The software should generate the final wind simulation by combining the wind speed, storm, and burst data (three files created during steps 2 to 4), and save the result in a data file named “WindSimulationData.txt”. The right-most column in the files includes a binary value signifying whether a storm is active at that time (1 means active while 0 means inactive).

### **Data Validation:**

Before generating the file, the input in the configuration file must be validated according to the following constraints:

Wind Configuration Constraints:

1. Average wind speed must non-negative and greater than the gust speed to avoid having (average speed - gust value) negative.
2. Gust speed must be non-negative as speed can never be negative.
3. The duration and step size must be non-zero positive for the data to sample different times.

Storm Configuration Constraints:

1. The storm probability must be non-negative (0 probability should give us no storms) and written in decimal form (40% -> 0.4)
2. Storm amplitude must be positive (min and max both should be positive) and the max amplitude must be greater than the minimum.
3. Storm duration must be positive (min and max both should be positive) and the max duration must be greater than the minimum.

Microburst Configuration Constraints:

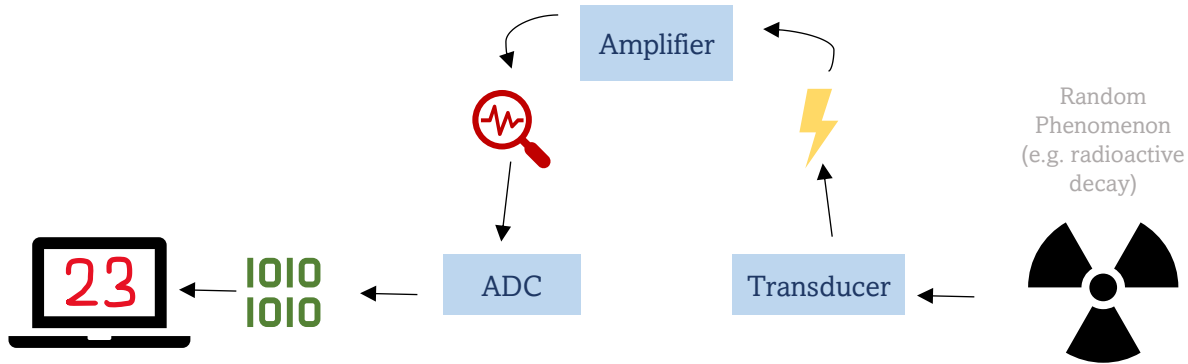
1. The microburst probability must be non-negative (0 probability should give us no storms) and written in decimal form (40% -> 0.4)
2. Microburst amplitude must be positive (min and max both should be positive) and the max amplitude must be greater than the minimum.
3. Microburst duration must be positive (min and max both should be positive) and the max duration must be greater than the minimum.

### **How Computers Generate Random Numbers:**

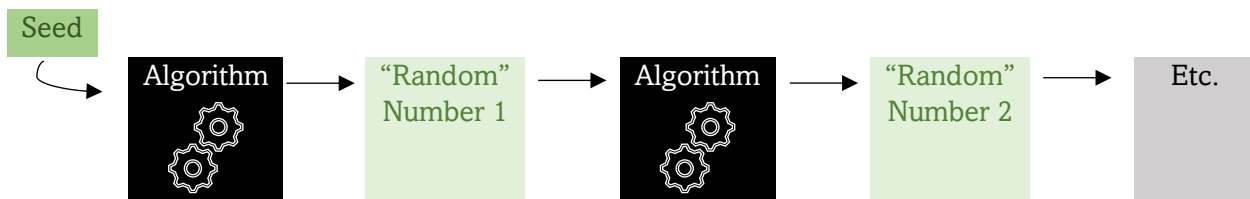
Since computers follow structured, rigid codes, it may seem counterintuitive for them to generate randomness. Computer-generated random numbers are grouped into two categories:

1. “True” Random Numbers: These numbers are not dependent on any algorithms and are rather generated by measuring some outside random phenomenon such as the radioactive decay of an atom or the random fluctuations in atmospheric pressure/temperature. The

process (also represented in the figure) goes as follows: A transducer converts the phenomenon being observed into electrical signals. These signals are later edited by the amplifier to amplify the amplitude of the random fluctuations of the data. Finally, the signals are translated into computer-understandable format (binary) with the help of an analogue-to-digital converter (ADC). Thus, these numbers exhibit no patterns or structure and are truly random. These numbers are indispensable in situations where random numbers must be unpredictable, e.g. in the process of data encryption (Hoffman, Herzstein).



2. Pseudorandom Numbers: They are an alternative to true random numbers and use an algorithm to produce “random” numbers. Before the algorithm is used, it must be seeded with a number, which is then mathematically manipulated into a random number, and that number is used as the seed for the next number of the sequence (As shown in the figure below). If the same seed is used twice, the sequence of random numbers will be the same, and that’s why these numbers are called “pseudo”-random. These numbers can’t be used in situations where true randomness is required as in the aforementioned case of encryption due to their predictable nature (knowing the algorithm and the seed is all one needs to know the stream of random numbers). Their true use lies in simulations, games, etc. where random numbers only need to *seem* random to the user and their predictability affects nothing of the experience (Herzstein).



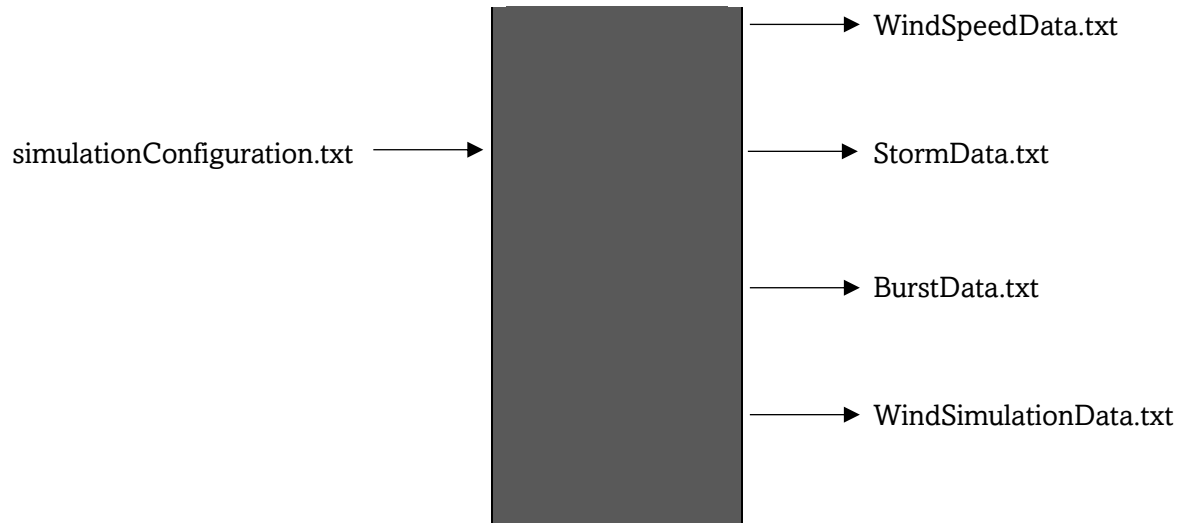
For our case, we will be using pseudorandom numbers as they suit our purposes (creating seemingly random numbers).

### **Input/Output Description:**

**Input:** simulationConfiguration.txt

**Output:**

- WindSpeedData.txt
- StormData.txt
- BurstData.txt
- WindSimulationData.txt

**I/O Diagram:**

## Step 3 - Test Cases & Algorithm:

**Test Cases:****Case 1: Invalid Input**

In the case of invalid input (simulationConfiguration.txt includes data against the constraints defined earlier), the program must stop all execution and inform user of all errors present in the document.

**Case 2: simulationConfiguration.txt Doesn't Exist**

In the case of the configuration file not being created, the program should display an error message and end execution.

**Case 3: Sample Simulation 1**

Average wind speed = 10; Gust speed = 2; Time = 1hr (3600s); Step-size = 10s

Storm Probability = 2.5%; Min & Max Amplitude: 15 - 25; Min & Max Storm Duration: 250s – 350s

Microburst Probability = 2.5%; Min & Max Amplitude: 30 – 50; Min & Max Microburst Duration: 1s – 3s

Every simulation will look different, but the output is expected to include some storms, some microbursts, be 1hr long, and show the values every 10 seconds. Each file's data should not go below or above the minimum or maximum amplitudes respectively. 2 Examples will be shown for this case to demonstrate the randomness of the data.

#### **Case 4: Sample Simulation 2**

Average wind speed = 20; Gust speed = 5; Time = 600s; Step-size = 0.5s

Storm Probability = 5%; Min & Max Amplitude: 20 - 40; Min & Max Storm Duration: 50s – 100s

Microburst Probability = 5%; Min & Max Amplitude: 40 – 60; Min & Max Microburst Duration: 1s – 5s

Every simulation will look different, but the output is expected to include more storms and microbursts than the previous one, be 600s long, and show the values every 0.5 seconds. Each file's data should not go below or above the minimum or maximum amplitudes respectively. 2 Examples will be shown for this case to demonstrate the randomness of the data.

#### **Case 5: Sample Simulation With No Storms (0% Storm Probability)**

If the storm probability is 0, there should be no storm or burst peaks (since bursts only occur during storms) and the final file should look the same as the WindSpeedData.txt file.

#### **Case 6: Sample Simulation Always with Storms (100% Storm Probability)**

If the storm probability is 100%, there should always be a storm.

#### **The Right-Most Column of WindSimulationData.txt**

In the final file (WindSimulationData.txt), the right-most column shows if a storm is happening or not. It should be 1 when there is an ongoing storm and 0 when there isn't.

## **Algorithm Design**

*Define ROW1 as 4, ROW2\_3 as 5, W as 20, and DELTA as 0.000001*

```
configure(windConfig, stormConfig, burstConfig)
    Assign 0 to errorCount
    Create an input file stream fileReader
    Open "simulationConfiguration.txt" in input mode
    If fileReader fails at opening the file
        PrintError "ERROR: Failed to open file . . .", Newline
        Exit with code -1
    Assign 0 to i
    Repeat while i < ROW1
        Read the next integer from fileReader into windConfig[i]
    Assign 0 to i
```

```

Repeat while i < ROW2_3
    Read the next integer from fileReader into stormConfig[i]
Assign 0 to i
Repeat while i < ROW2_3
    Read the next integer from fileReader into burstConfig[i]
If windConfig[0] is less than windConfig[1] OR windConfig[0] is less than 0
    PrintError "Invalid Input: Wind speed has to be non-negative and less than the gust
    speed . . .", Newline
    Increment errorCount
If windConfig[1] is less than 0
    PrintError "Invalid Input: Gust speed has to be non-negative . . .", Newline
    Increment errorCount
If windConfig[2] is less than or equal to 0
    PrintError "Invalid input: Time must be positive . . .", Newline
    Increment errorCount
If windConfig[3] is less than or equal to 0
    PrintError "Invalid input: Step-size must be positive . . .", Newline
    Increment errorCount
If stormConfig[0] is less than 0
    PrintError "Invalid input: Probability must be non-negative . . .", Newline
    Increment errorCount
If stormConfig[1] is less than or equal to 0
    PrintError "Invalid input: Min storm amplitude must be positive . . .", Newline
    Increment errorCount
If stormConfig[2] is less than or equal to 0 OR stormConfig[2] is less than stormConfig[1]
    PrintError "Invalid input: Maximum storm amplitude must be positive and greater
    than the minimum amplitude . . .", Newline
    Increment errorCount
If stormConfig[3] is less than or equal to 0
    PrintError "Invalid input: Min storm duration must be positive . . .", Newline
    Increment errorCount
If stormConfig[4] is less than or equal to 0 OR stormConfig[4] < stormConfig[3]
    PrintError "Invalid input: Max storm duration must be positive and greater than the
    minimum storm duration . . .", Newline
    Increment errorCount
If burstConfig[0] is less than 0
    PrintError "Invalid input: Probability must be non-negative . . .", Newline
    Increment errorCount
If burstConfig[1] is less than or equal to 0
    PrintError "Invalid input: Min burst amplitude must be positive . . .", Newline
    Increment errorCount
If burstConfig[2] is less than or equal to 0 OR burstConfig[2] is less than burstConfig[1]
    PrintError "Invalid input: Maximum burst amplitude must be positive and greater
    than the minimum amplitude . . .", Newline
    Increment errorCount
If burstConfig[3] is less than or equal to 0
    PrintError "Invalid input: Min burst duration must be positive . . .", Newline
    Increment errorCount

```

*If burstConfig[4] is less than or equal to 0 OR burstConfig[4] < burstConfig[3]*  
     *PrintError "Invalid input: Max burst duration must be positive and greater than the minimum burst duration . . .", Newline*  
     *Increment errorCount*  
*Close fileReader*  
*Return errorCount*

*generateSpeedData(windConfig)*

*Create an output file stream writer*  
     *Open "WindSpeedData.txt" into writer in the truncate mode*  
     *If writer fails at opening the file*  
         *PrintError "ERROR: Failed to open file . . .", Newline*  
         *Exit with code -1*  
     *Print "Time (s)", Space, "Wind-Speed", Newline into writer*  
     *Assign 0 to double t*  
     *Repeat while t is less than or equal to windConfig[2] + DELTA*  
         *Print t, Space, windConfig[0] + (randomly generated double between -windConfig[1] and windConfig[1]), Newline*  
         *Increment t by step*  
     *Close writer*

*generateStormData(stormConfig, duration, step)*

*Create an output file stream writer*  
     *Open "StormData.txt" into writer in the truncate mode*  
     *If writer fails at opening the file*  
         *PrintError "ERROR: Failed to open file . . .", Newline*  
         *Exit with code -1*  
     *Print "Time (s)", Space, "Storm-Magnitude", Newline into writer*  
     *Assign 0 to t*  
     *Repeat while t is less than or equal to duration + DELTA*  
         *If a randomly generated double between 0 and 1 is less than stormConfig[0] OR stormConfig[0] is equal to 1*  
             *Assign a randomly generated double between stormConfig[3] and stormConfig[4] to T*  
             *Assign t to initialT*  
             *Repeat while (t is less than or equal to T + initialT + DELTA) AND (t is less than or equal to duration + DELTA)*  
                 *Print t, Space, randomly generated double between stormConfig[1], stormConfig[2] into writer*  
                 *Increment t by step*  
                 *Decrement t by step*  
             *Otherwise*  
                 *Print t, Space, 0, Newline into writer*  
                 *Increment t by step*  
     *Close writer*

*generateBurstData(stormConfig, duration, step)*

*Create an output file stream writer*



```

Open "BurstData.txt" into writer in the truncate mode
If writer fails at opening the file
    PrintError "ERROR: Failed to open file . . .", Newline
    Exit with code -1
Create an input file stream reader
Open "StormData.txt" into reader in the at the end mode
If reader fails at opening the file
    PrintError "ERROR: Failed to open file . . .", Newline
    Exit with code -1
Move the cursor in reader to a position 2*W after the beginning of the file
Print "Time (s)", Space, "Burst-Magnitude", Newline into writer
Assign 0 to t, 0 to x, 0 to s
Read the next double from reader into x
Read the next double from reader into s
Repeat while t is less than or equal to duration + DELTA
    If NOT ((s is less than DELTA) AND (s is greater than -DELTA))
        If a randomly generated double between 0 and 1 is less than burstConfig[0]
        or burstConfig[0] is equal to 1
            Assign a randomly generated double between burstConfig[3] and
            burstConfig[4] to T
            Assign t to initialT
            Repeat while (t is less than or equal to T + initialT + DELTA) AND (t
            is less than or equal to duration + DELTA) AND NOT ((s is less than
            DELTA) AND (s is greater than -DELTA))
                Print t, Space, randomly generated double between
                burstConfig[1], burstConfig[2] into writer
                Read the next double from reader into x
                Read the next double from reader into s
                Increment t by step
            Decrement t by step
        Otherwise
            Print t, Space, 0, Newline into writer
            Read the next double from reader into x
            Read the next double from reader into s
    Otherwise
        Print t, Space, 0, Newline into writer
        Read the next double from reader into x
        Read the next double from reader into s
    Increment t by step
Close writer
Close reader
generateSimulationData(duration, step)
Create an output file stream writer
Open "WindSimulationData.txt" into writer in the truncate mode
If writer fails at opening the file
    PrintError "ERROR: Failed to open file . . .", Newline
    Exit with code -1
Create an input file stream wind

```

Open "WindSpeedData.txt" into wind in the at the end mode  
 Create an input file stream storm  
 Open "StormData.txt" into storm in the at the end mode  
 Create an input file stream burst  
 Open "BurstData.txt" into burst in the at the end mode  
 If (wind fails at opening the file) OR (storm fails at opening the file) OR (burst fails at opening the file)  
     PrintError "ERROR: Failed to open file . . .", Newline  
     Exit with code -1  
 Assign 0 to t, 0 to windS, 0 to stormM, 0 to burstM, 0 to x  
 Assign false to pres  
 Move the cursor in wind to a position 2\*W after the beginning of the file  
 Move the cursor in storm to a position 2\*W after the beginning of the file  
 Move the cursor in burst to a position 2\*W after the beginning of the file  
 Print "Time (s)", Space, "Speed", Space, "Storm Present?", Newline into writer  
 Repeat while t is less than or equal to duration + DELTA  
     Read next two doubles into x and windS from wind  
     Read next two doubles into x and stormM from storm  
     Read next two doubles into x and burstM from burst  
     If NOT((stormM is greater than -DELTA) AND (stormM is less than DELTA))  
         Assign true to pres  
     Otherwise  
         Assign false to pres  
     Print t, Space, (windS + stormM + burstM), Space, pres, Newline  
     Increment t by step  
 Close writer  
 Close wind  
 Close storm  
 Close burst

Main()

Seed the random number generator with the current time  
 Create an array windConfig of type double of ROW1 elements  
 Create an array stormConfig of type double of ROW2\_3 elements  
 Create an array burstConfig of type double of ROW2\_3 elements  
 Assign configure(windConfig, stormConfig, burstConfig) to errorCount  
 If errorCount is greater than 0  
     PrintError errorCount, " errors found in simulationConfiguration.txt . . .", Newline  
     PrintError "Please perform necessary edits and rerun the program . . .", Newline  
     Exit with error errorCount  
 Print "Configuration performed successfully . . .", Newline  
 generateSpeedData(windConfig)  
 Print "WindSpeedData.txt created successfully . . .", Newline  
 generateStormData(stormConfig, windConfig[2], windConfig[3])  
 Print "StormData.txt created successfully . . .", Newline  
 generateBurstData(burstConfig, windConfig[2], windConfig[3])  
 Print "BurstData.txt created successfully . . .", Newline  
 generateSimulationData(windConfig[2], windConfig[3])

*Print "WindSimulationData.txt generated successfully . . .", Newline*  
*Print "-----", Newline*  
*Print "Thanks for using our services . . .", Newline*  
*End program*

## Step 4 – C++ Code:

```
#include <iostream> // To display messages into the output stream
#include <fstream> // To read and write into files
#include <ctime> // To seed the random function
#include <iomanip> // To organize the data in tabular form
#define ROW1 4 // Length of the windConfig array
#define ROW2_3 5 // Length of the stormConfig and burstConfig arrays
#define W 20 // Width to be used in the setw() function
#define DELTA 0.000001 // To account for precision in double/float variables

using namespace std;

// Function prototypes
int configure(double windConfig[], double stormConfig[], double burstConfig[]);

void generateSpeedData(const double windConfig[]);

void generateStormData(const double stormConfig[], double duration, double step);

void generateBurstData(const double info[], double duration, double step);

void generateSimulationData(double duration, double step);

int main() {
    srand(time(0)); // Seeding the random function
    // Creating arrays for storing configuration data
    double windConfig[ROW1] = { 0.0 };
    double stormConfig[ROW2_3] = { 0.0 };
    double burstConfig[ROW2_3] = { 0.0 };
    int errorCount = configure(windConfig, stormConfig, burstConfig); // Configuring
and validating data
    if (errorCount > 0) {
        cerr << errorCount << " errors encountered in simulationConfiguration.txt .
. . " << endl;
        cerr << "Please perform necessary edits and rerun the program . . ." <<
endl;
        exit(errorCount);
    }
    cout << "Configuration performed successfully . . ." << endl;
    // Create and fill WindSpeedData.txt
    generateSpeedData(windConfig);
    cout << "WindSpeedData.txt generated successfully . . ." << endl;
    // Create and fill StormData.txt
    generateStormData(stormConfig, windConfig[2], windConfig[3]);
    cout << "StormData.txt generated successfully . . ." << endl;
    // Create and fill BurstData.txt
    generateBurstData(burstConfig, windConfig[2], windConfig[3]);
    cout << "BurstData.txt generated successfully . . ." << endl;
    // Create and fill WindSimulationData.txt
    generateSimulationData(windConfig[2], windConfig[3]);
    cout << "WindSimulationData.txt generated successfully . . ." << endl;
    cout << endl << "-----" <<
endl;
    cout << "Thanks for using our services . . ." << endl;
    return(0);
}
```

```

}

// The configure function reads all the user-specified constraints and values from the
// file simulationConfiguration.txt and validates said data
int configure(double windConfig[], double stormConfig[], double burstConfig[]) {
    int errorCount(0);
    ifstream fileReader;
    fileReader.open("simulationConfiguration.txt", ios::in);
    if (fileReader.fail()) {
        cerr << "ERROR: Failed to open file . . ." << endl;
        exit(-1);
    }
    // Reading wind speed configuration data
    for (int i = 0; i < ROW1; i++) {
        fileReader >> windConfig[i];
    }
    // Reading storm configuration data
    for (int i = 0; i < ROW2_3; i++) {
        fileReader >> stormConfig[i];
    }
    // Reading burst configuration data
    for (int i = 0; i < ROW2_3; i++) {
        fileReader >> burstConfig[i];
    }
    // Validating all configuration data based on constraints mentioned in the lab
report
    if (windConfig[0] < windConfig[1] || windConfig[0] < 0) {
        cerr << "Invalid input: Wind speed has to be non-negative and less than the
gust speed . . ." << endl;
        errorCount++;
    }
    if (windConfig[1] < 0) {
        cerr << "Invalid input: Gust speed has to be non-negative . . ." << endl;
        errorCount++;
    }
    if (windConfig[2] <= 0) {
        cerr << "Invalid input: Time must be positive . . ." << endl;
        errorCount++;
    }
    if (windConfig[3] <= 0) {
        cerr << "Invalid input: Step-size must be positive . . ." << endl;
        errorCount++;
    }
    if (stormConfig[0] < 0) {
        cerr << "Invalid input: Probability must be non-negative . . ." << endl;
        errorCount++;
    }
    if (stormConfig[1] <= 0) {
        cerr << "Invalid input: Min storm amplitude must be positive . . ." <<
endl;
        errorCount++;
    }
    if (stormConfig[2] <= 0 || stormConfig[2] < stormConfig[1]) {
        cerr << "Invalid input: Maximum amplitude must be positive and greater than
the minimum amplitude . . ." << endl;
        errorCount++;
    }
}

```

```

        if (stormConfig[3] <= 0) {
            cerr << "Invalid input: Min storm duration must be positive . . ." << endl;
            errorCount++;
        }
        if (stormConfig[4] <= 0 || stormConfig[4] < stormConfig[3]) {
            cerr << "Invalid input: Max storm duration must be positive and greater
than the minimum duration . . ." << endl;
            errorCount++;
        }
        if (burstConfig[0] < 0) {
            cerr << "Invalid input: Probability must be non-negative . . ." << endl;
            errorCount++;
        }
        if (burstConfig[1] <= 0) {
            cerr << "Invalid input: Min storm amplitude must be positive . . ." <<
endl;
            errorCount++;
        }
        if (burstConfig[2] <= 0 || burstConfig[2] < burstConfig[1]) {
            cerr << "Invalid input: Maximum amplitude must be positive and greater than
the minimum amplitude . . ." << endl;
            errorCount++;
        }
        if (burstConfig[3] <= 0) {
            cerr << "Invalid input: Min storm duration must be positive . . ." << endl;
            errorCount++;
        }
        if (burstConfig[4] <= 0 || burstConfig[4] < burstConfig[3]) {
            cerr << "Invalid input: Max strom duration must be positive and greater
than the minimum duration . . ." << endl;
            errorCount++;
        }
        fileReader.close();
        return errorCount;
    }

// The generateSpeedData function creates the WindSpeedData.txt file which stores a
random value of speed within the user-specified constraints for each time in the duration
void generateSpeedData(const double windConfig[]) {
    ofstream writer("WindSpeedData.txt", ios::trunc);
    if (writer.fail()) {
        cerr << "ERROR: Failed to open file . . ." << endl;
        exit(-1);
    }
    // Prepare column headers
    writer << setw(W) << left << "Time (s)" << setw(W) << "Wind-Speed" << endl;
    for (double t = 0; t <= (windConfig[2]) + DELTA; t += windConfig[3]) {
        writer << left << setw(W) << t << setw(W) << ((double)rand() / RAND_MAX - 0.5) * 2 * windConfig[1] +
windConfig[1] and +windConfig[1] (-gust and +gust)*" << endl;
    }
    writer.close();
}

void generateStormData(const double stormConfig[], double duration, double step) {
    ofstream writer("StormData.txt", ios::trunc);
    if (writer.fail()) {
        cerr << "ERROR: Failed to open file . . ." << endl;
    }
}

```

```

        exit(-1);
    }
    // Prepare column headers
    writer << setw(W) << left << "Time (s)" << setw(W) << "Storm-Magnitude" << endl;
    for (double t = 0; t <= duration + DELTA; t+=step) {
        if (((double)rand() / RAND_MAX < stormConfig[0] || stormConfig[0] ==1) { //
Generates a storm based on the specified probability from the configuration file
            double T = (((double)rand() / RAND_MAX) * (stormConfig[4] -
stormConfig[3]) + stormConfig[3]); // Random value between stormConfig[3] and
stormConfig[4] (Min and max duration)
            double initialT = t;
            for (t = initialT; t <= T + initialT + DELTA && t <= duration +
DELTA; t+=step) {
                writer << left << setw(W) << t << setw(W) << (((double)rand()
/ RAND_MAX) * (stormConfig[2] - stormConfig[1]) + stormConfig[1]) /*Random value between
stormConfig[1] and stormConfig[2] (Min and max amplitude)*/ << endl;
            }
            t-=step; // Before the loop exits, it will increment to the next t
value. This is to counteract this incrementation.
        }
        else
            writer << left << setw(W) << t << setw(W) << 0 << endl;
    }
    writer.close();
}

void generateBurstData(const double burstConfig[], double duration, double step) {
    ofstream writer("BurstData.txt", ios::trunc);
    if (writer.fail()) {
        cerr << "ERROR: Failed to open file . . ." << endl;
        exit(-1);
    }
    ifstream reader("StormData.txt", ios::ate); // The file needs to be opened in the
ios::ate mode to allow for skipping the column headers
    if (reader.fail()) {
        cerr << "ERROR: Failed to open file . . ." << endl;
        exit(-1);
    }
    reader.seekg(2*W, ios::beg); // Places cursor after the column headers to begin
reading the double values
    writer << setw(W) << left << "Time (s)" << setw(W) << "Burst-Magnitude" << endl;
    double x(0), s(0); // x is a rubbish variable that will be use to skip through the
unwanted data
    reader >> x >> s;
    for (double t = 0; t <= duration + DELTA; t += step) {
        if (!(s < DELTA && s > -DELTA)) { // Equivalent to s != 0 but accounts for
uncertainty of double values. Checks if there is a storm, and if so, a microburst can
occur
            if (double)rand() / RAND_MAX < burstConfig[0] || burstConfig[0] ==
1) { // Generates a microburst according to the probability defined in the configuration
file
                double T = (((double)rand() / RAND_MAX) * (burstConfig[4] -
burstConfig[3]) + burstConfig[3]); // Random value between burstConfig[3] and
burstConfig[4] (Min and max duration)
                double initialT = t;

```

```

        for (t = initialT; t <= T + initialT + DELTA && t <= duration
+ DELTA && !(s<DELTA && s>-DELTA) /*Equivalent to s != 0 but accounts for uncertainty of
double values*/; t += step) {
            writer << left << setw(W) << t << setw(W) <<
(((double)rand() / RAND_MAX) * (burstConfig[2] - burstConfig[1]) + burstConfig[1])
/*Random value between burstConfig[1] and burstConfig[2] (Min and max amplitude)*/ <<
endl;
            reader >> x >> s;
        }
        t -= step; // Before the loop exits, it will increment to the
next t value. This is to counteract this incrementation.
    }
    else {
        writer << setw(W) << left << t << setw(W) << 0 << endl;
        reader >> x >> s;
    }
}
else {
    writer << setw(W) << left << t << setw(W) << 0 << endl;
    reader >> x >> s;
}
}
writer.close();
reader.close();
}

void generateSimulationData(double duration, double step) {
    ofstream writer("WindSimulationData.txt", ios::trunc);
    // The files need to be opened in the ios::ate mode to allow for skipping the
column headers
    ifstream wind("WindSpeedData.txt", ios::ate);
    ifstream storm("StormData.txt", ios::ate);
    ifstream burst("BurstData.txt", ios::ate);
    if (wind.fail() || storm.fail() || burst.fail() || writer.fail()) {
        cerr << "ERROR: Failed to open file . . ." << endl;
        exit(-1);
    }
    double x(0), windS(0), stormM(0), burstM(0); // x is a rubbish variable that will
be used to store unneeded data
    bool pres(false);
    // Move all cursors in front of the column headers to begin reading the doubles
    wind.seekg(2 * W, ios::beg);
    storm.seekg(2 * W, ios::beg);
    burst.seekg(2 * W, ios::beg);
    writer << setw(W) << left << "Time (s)" << setw(W) << "Speed" << setw(W) << "Storm
present?" << endl;
    for (double t = 0; t <= duration + DELTA; t+=step) {
        wind >> x >> windS;
        storm >> x >> stormM;
        burst >> x >> burstM;
        if (!(stormM > -DELTA && stormM < DELTA)) // checks if a storm is present
which will be printed into the WindSimulation.txt file
            pres = true;
        else
            pres = false;
        writer << setw(W) << left << t << setw(W) << (windS + stormM + burstM) <<
setw(W) << pres << endl;
    }
}

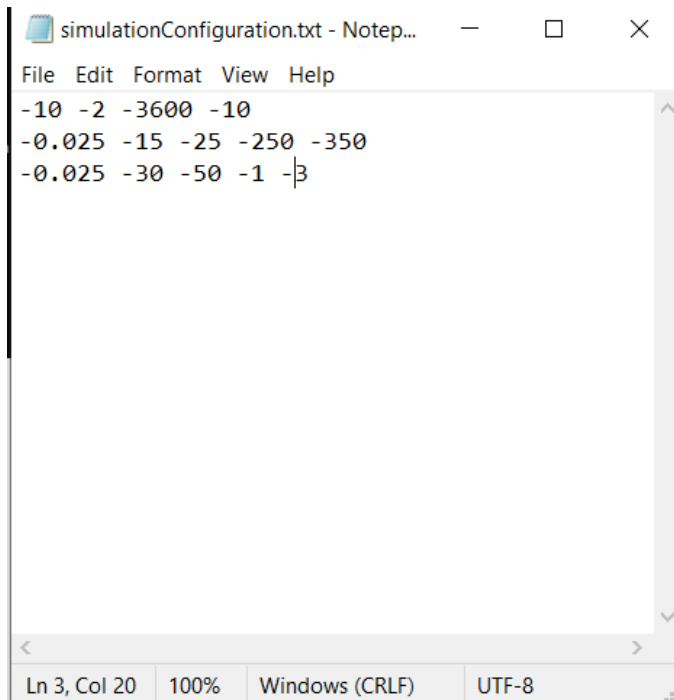
```



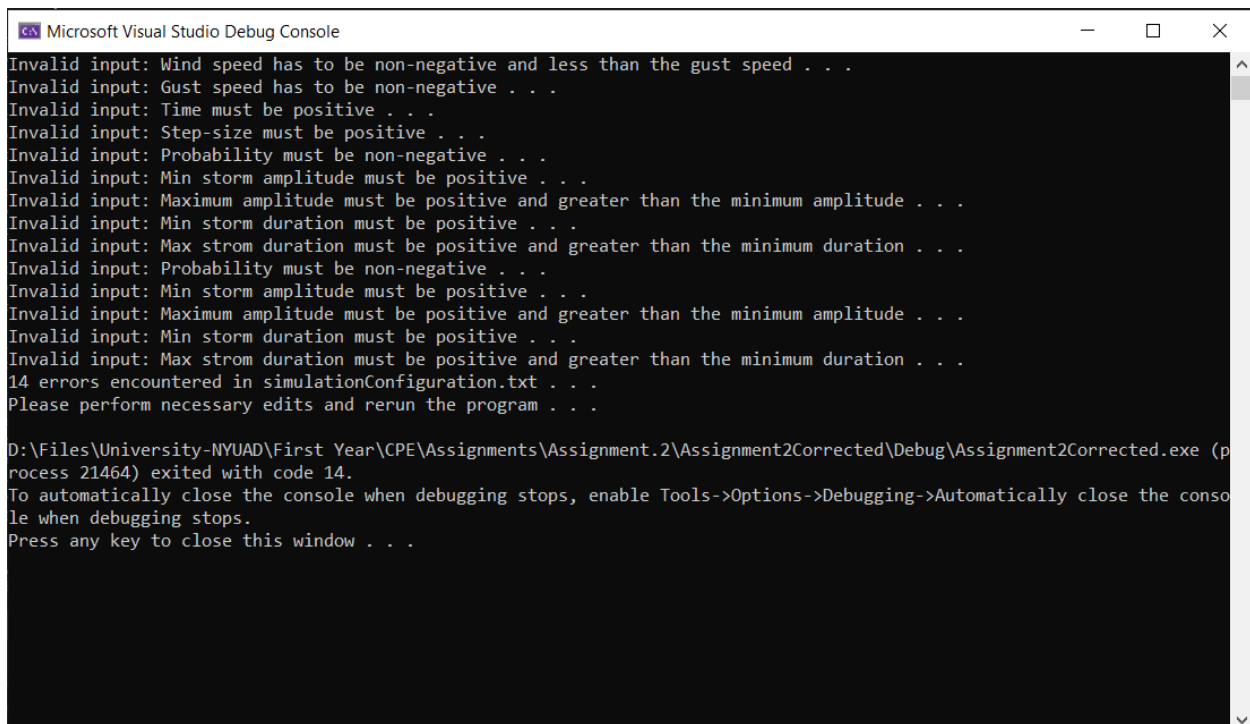
```
    }  
    writer.close();  
    wind.close();  
    storm.close();  
    burst.close();  
}
```

## Step 5 – Software Testing & Verification:

### Case 1: Invalid Input



```
simulationConfiguration.txt - Notep...
File Edit Format View Help
-10 -2 -3600 -10
-0.025 -15 -25 -250 -350
-0.025 -30 -50 -1 -3
Ln 3, Col 20 100% Windows (CRLF) UTF-8
```



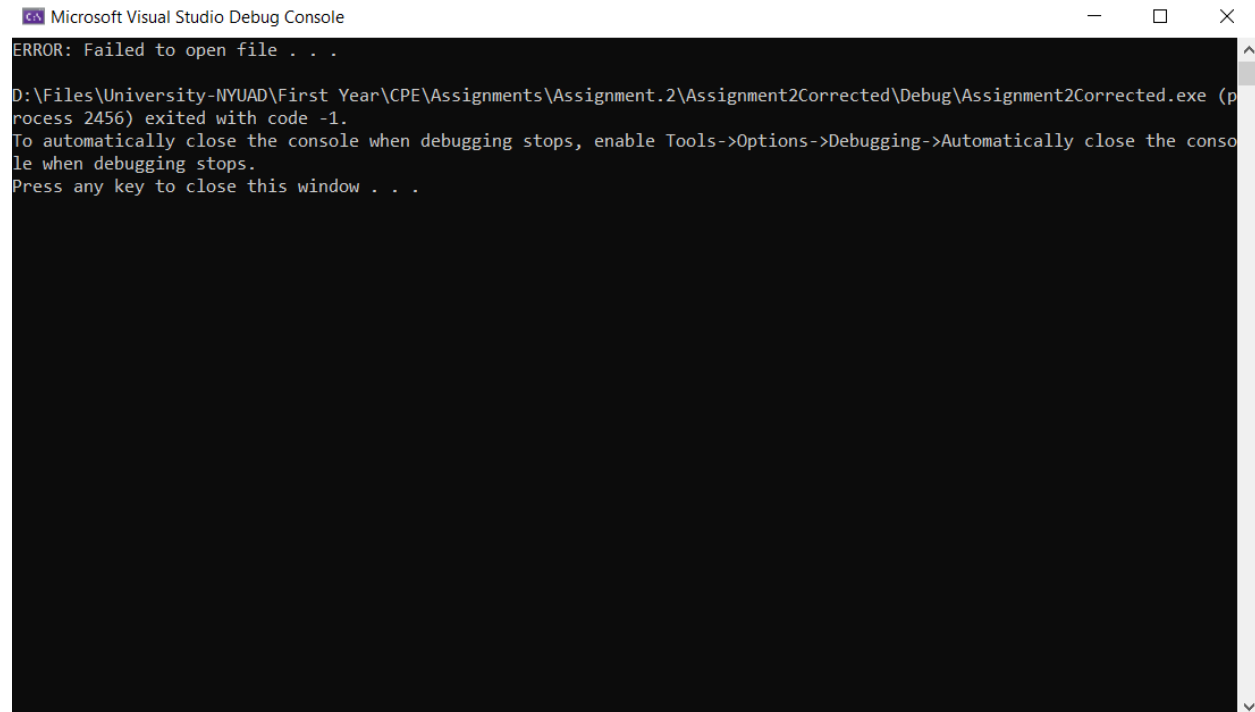
```
Microsoft Visual Studio Debug Console
Invalid input: Wind speed has to be non-negative and less than the gust speed . . .
Invalid input: Gust speed has to be non-negative . . .
Invalid input: Time must be positive . . .
Invalid input: Step-size must be positive . . .
Invalid input: Probability must be non-negative . . .
Invalid input: Min storm amplitude must be positive . . .
Invalid input: Maximum amplitude must be positive and greater than the minimum amplitude . . .
Invalid input: Min storm duration must be positive . . .
Invalid input: Max storm duration must be positive and greater than the minimum duration . . .
Invalid input: Probability must be non-negative . . .
Invalid input: Min storm amplitude must be positive . . .
Invalid input: Maximum amplitude must be positive and greater than the minimum amplitude . . .
Invalid input: Min storm duration must be positive . . .
Invalid input: Max storm duration must be positive and greater than the minimum duration . . .
14 errors encountered in simulationConfiguration.txt . . .
Please perform necessary edits and rerun the program . . .

D:\Files\University-NYUAD\First Year\CPE\Assignments\Assignment.2\Assignment2Corrected\Debug\Assignment2Corrected.exe (process 21464) exited with code 14.
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the console when debugging stops.
Press any key to close this window . . .
```

Program execution halted + Errors displayed => this test case passes

## Case 2: simulationConfiguration.txt Doesn't Exist:

Output Window:

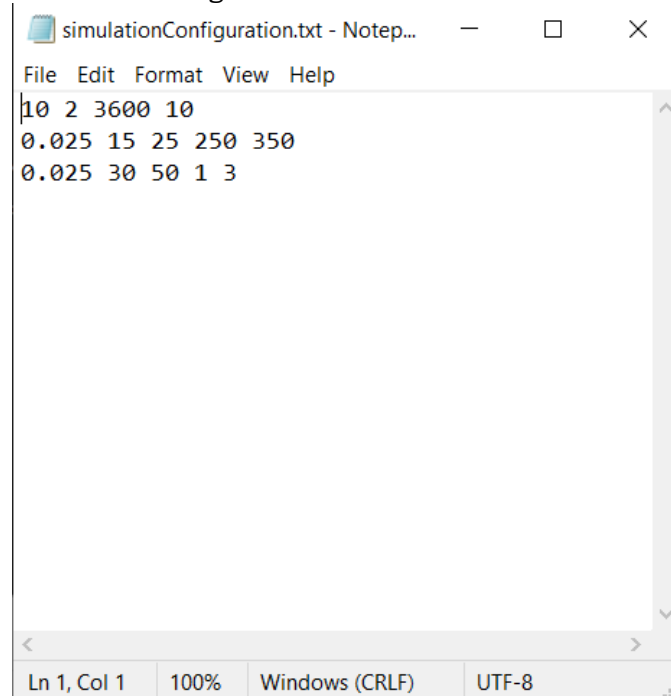


The screenshot shows the Microsoft Visual Studio Debug Console window. The title bar reads "Microsoft Visual Studio Debug Console". The console output is as follows:

```
ERROR: Failed to open file . . .  
  
D:\Files\University-NYUAD\First Year\CPE\Assignments\Assignment.2\Assignment2Corrected\Debug\Assignment2Corrected.exe (process 2456) exited with code -1.  
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the console when debugging stops.  
Press any key to close this window . . .
```

## Case 3: Sample Simulation 1

simulationConfiguration.txt:



The screenshot shows a Notepad++ window titled "simulationConfiguration.txt - Notep...". The menu bar includes File, Edit, Format, View, and Help. The text content of the file is:

```
10 2 3600 10  
0.025 15 25 250 350  
0.025 30 50 1 3
```

The status bar at the bottom indicates "Ln 1, Col 1", "100%", "Windows (CRLF)", and "UTF-8".

## Output Window:

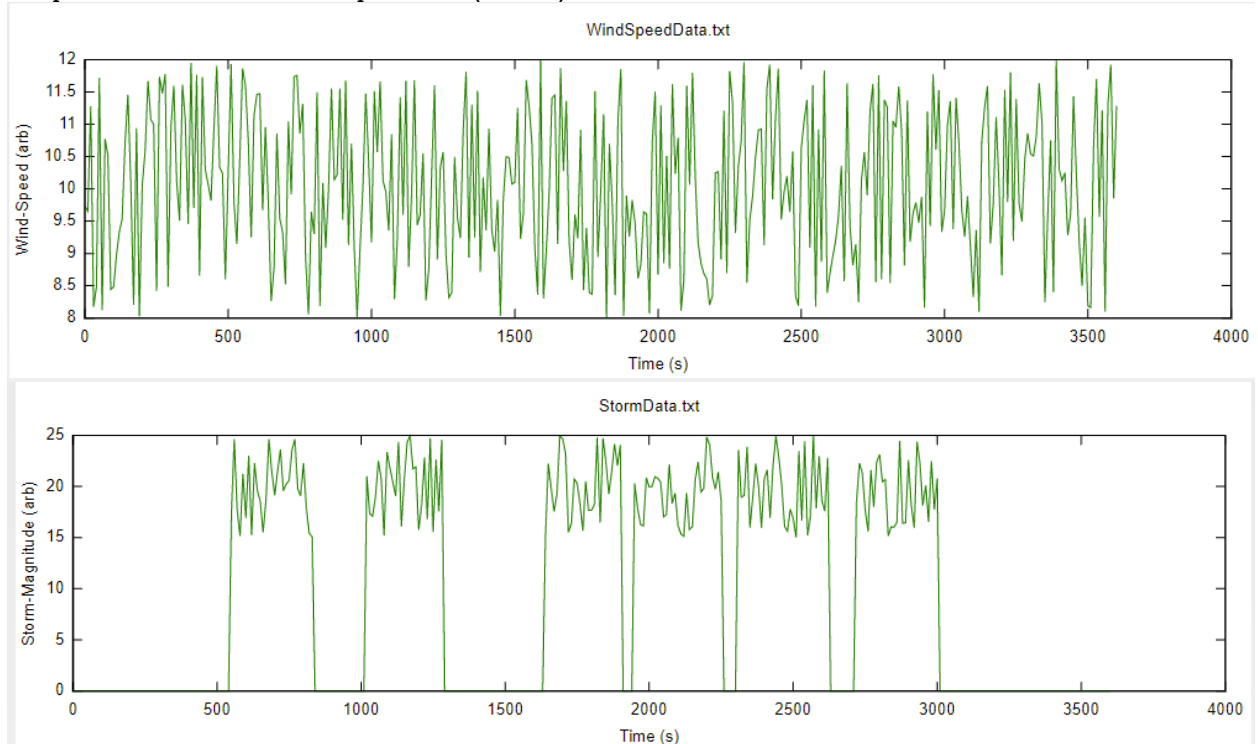
```
Microsoft Visual Studio Debug Console

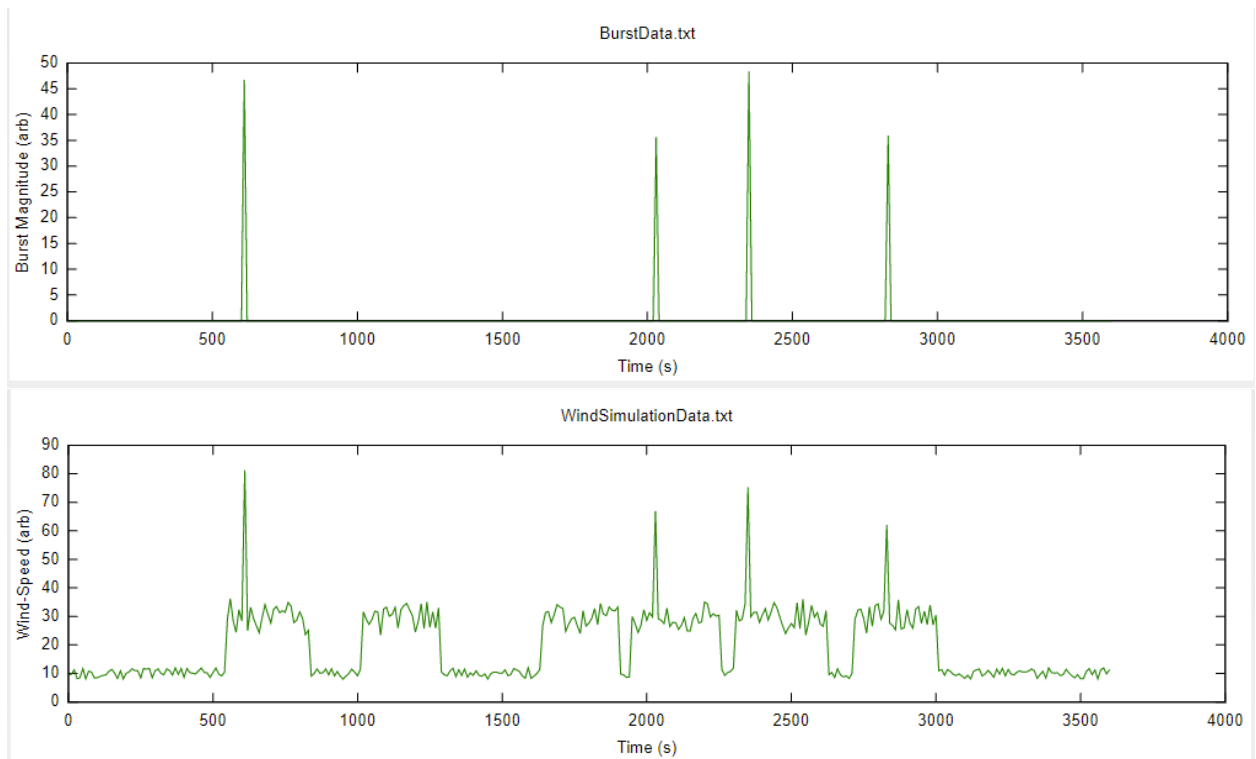
Configuration performed successfully . . .
WindSpeedData.txt generated successfully . . .
StormData.txt generated successfully . . .
BurstData.txt generated successfully . . .
WindSimulationData.txt generated successfully . . .

-----
Thanks for using our services . . .

D:\Files\University-NYUAD\First Year\CPE\Assignments\Assignment.2\Assignment2Corrected\Debug\Assignment2Corrected.exe (p
rocess 28748) exited with code 0.
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the cons
ole when debugging stops.
Press any key to close this window . . .
```

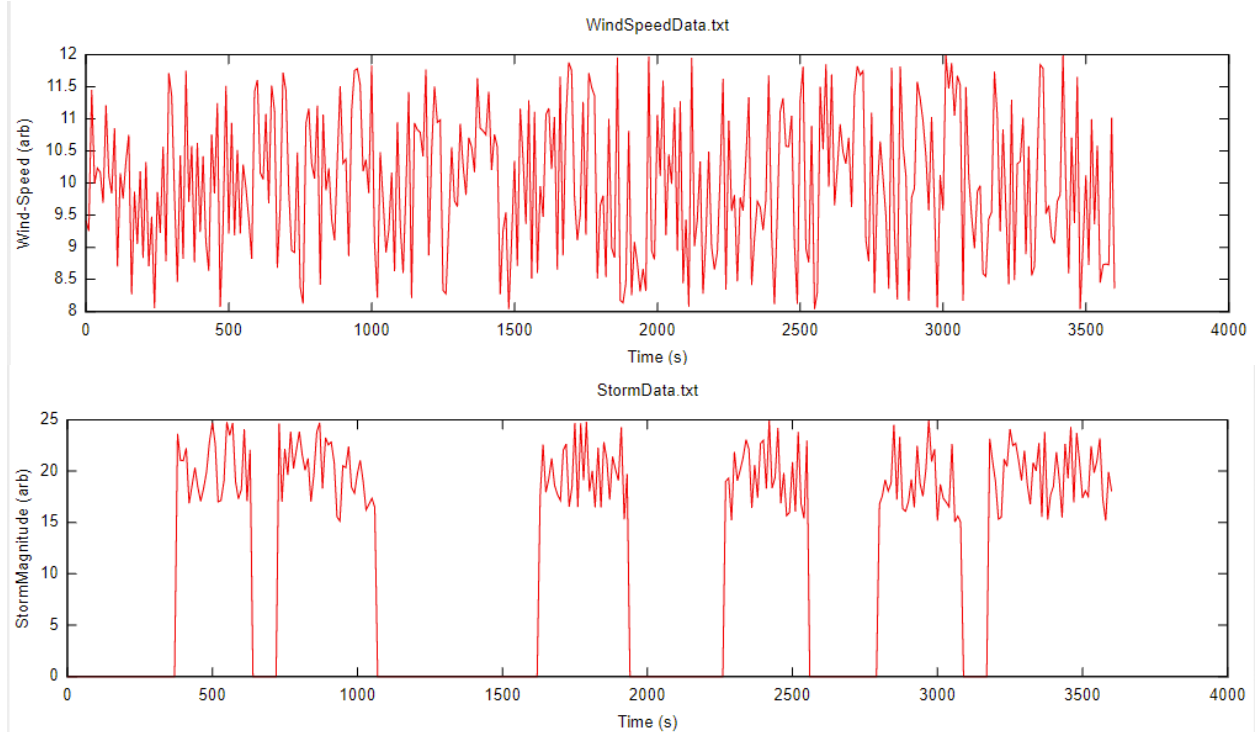
## Graphs Generated from Output Files (Test 1):

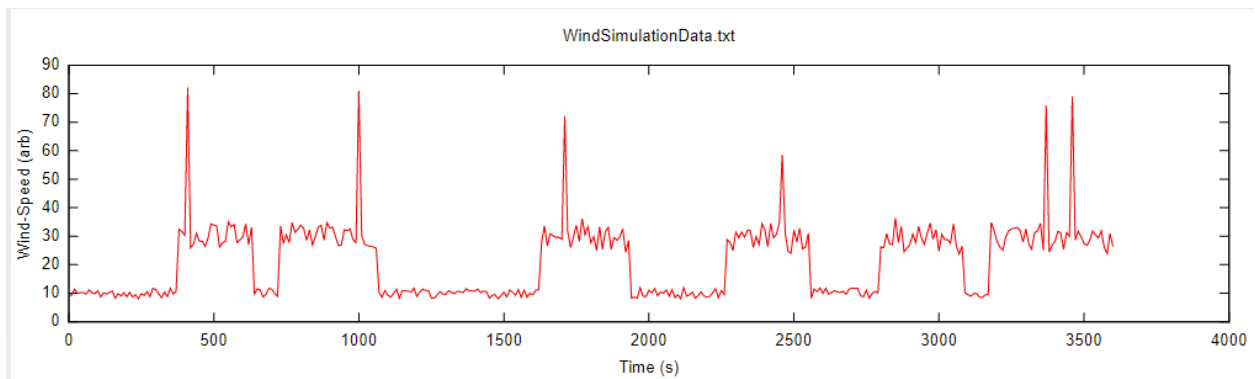
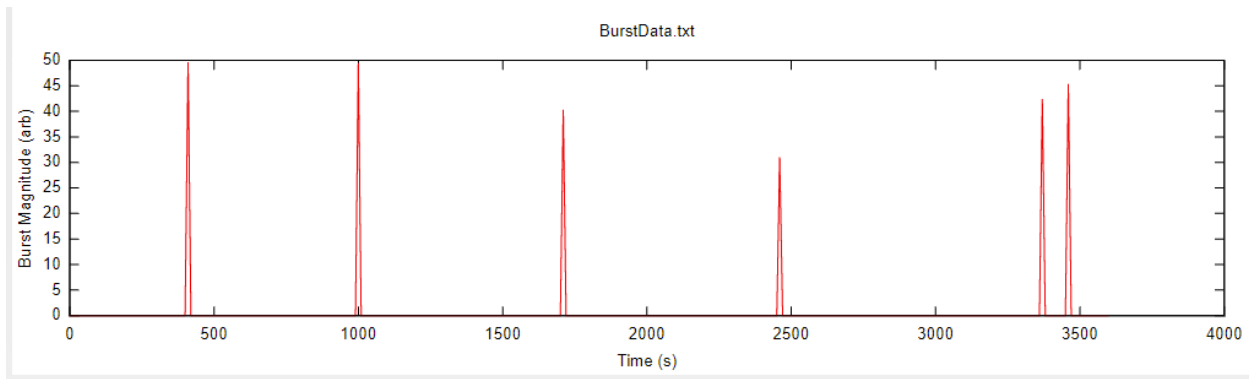




(WindSimulationData.txt includes the binary value to show when the storm occurs but that couldn't be represented graphically. However, it was correctly showing when a storm was happening and when it wasn't)

### Graphs Generated from Output Files (Test 2):



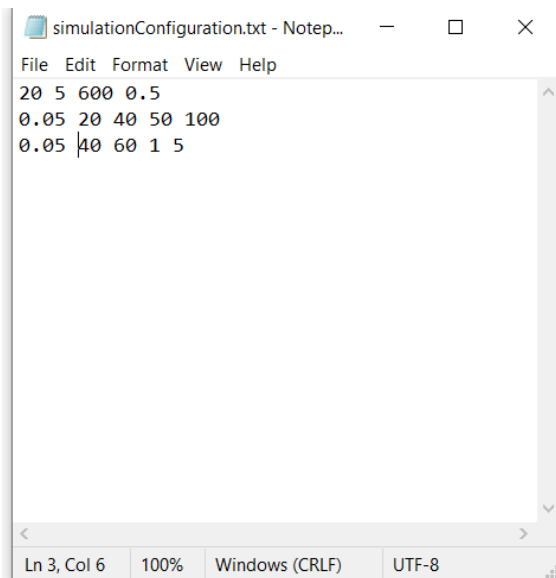


(WindSimulationData.txt includes the binary value to show when the storm occurs but that couldn't be represented graphically. However, it was correctly showing when a storm was happening and when it wasn't)

Since the output (though random) is as expected, this test case passes.

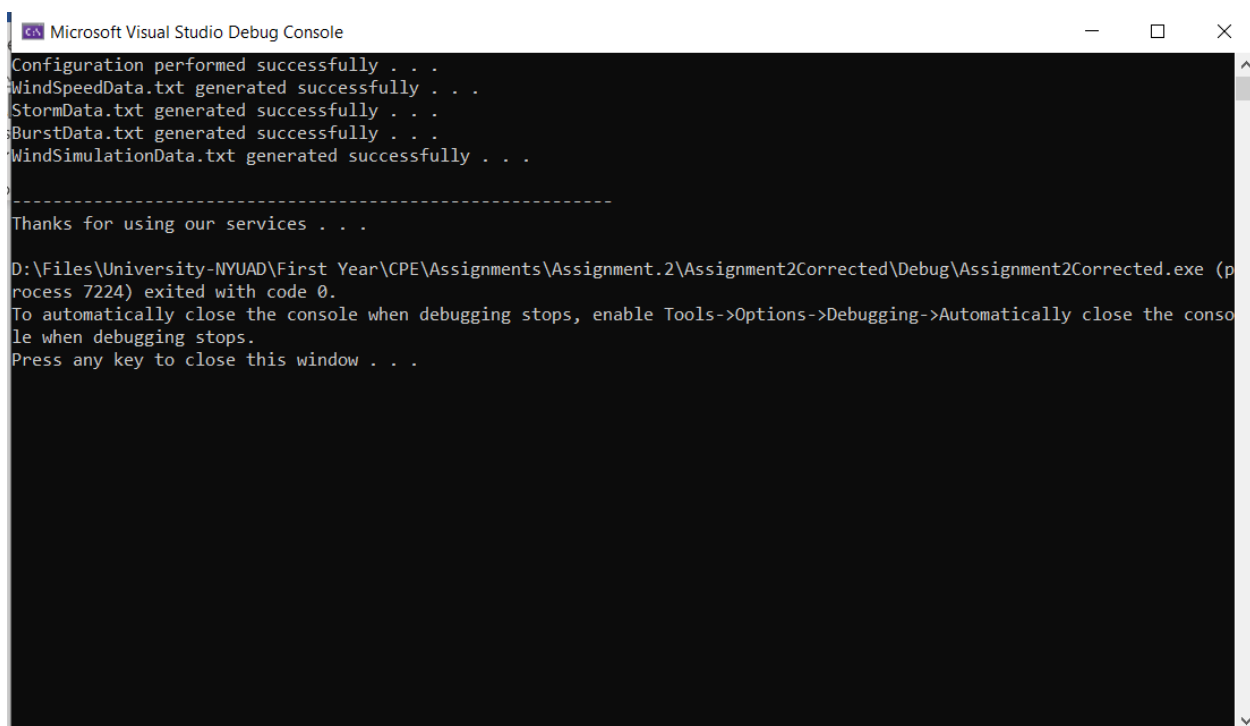
#### Case 4: Sample Simulation 2

simulationConfiguration.txt:



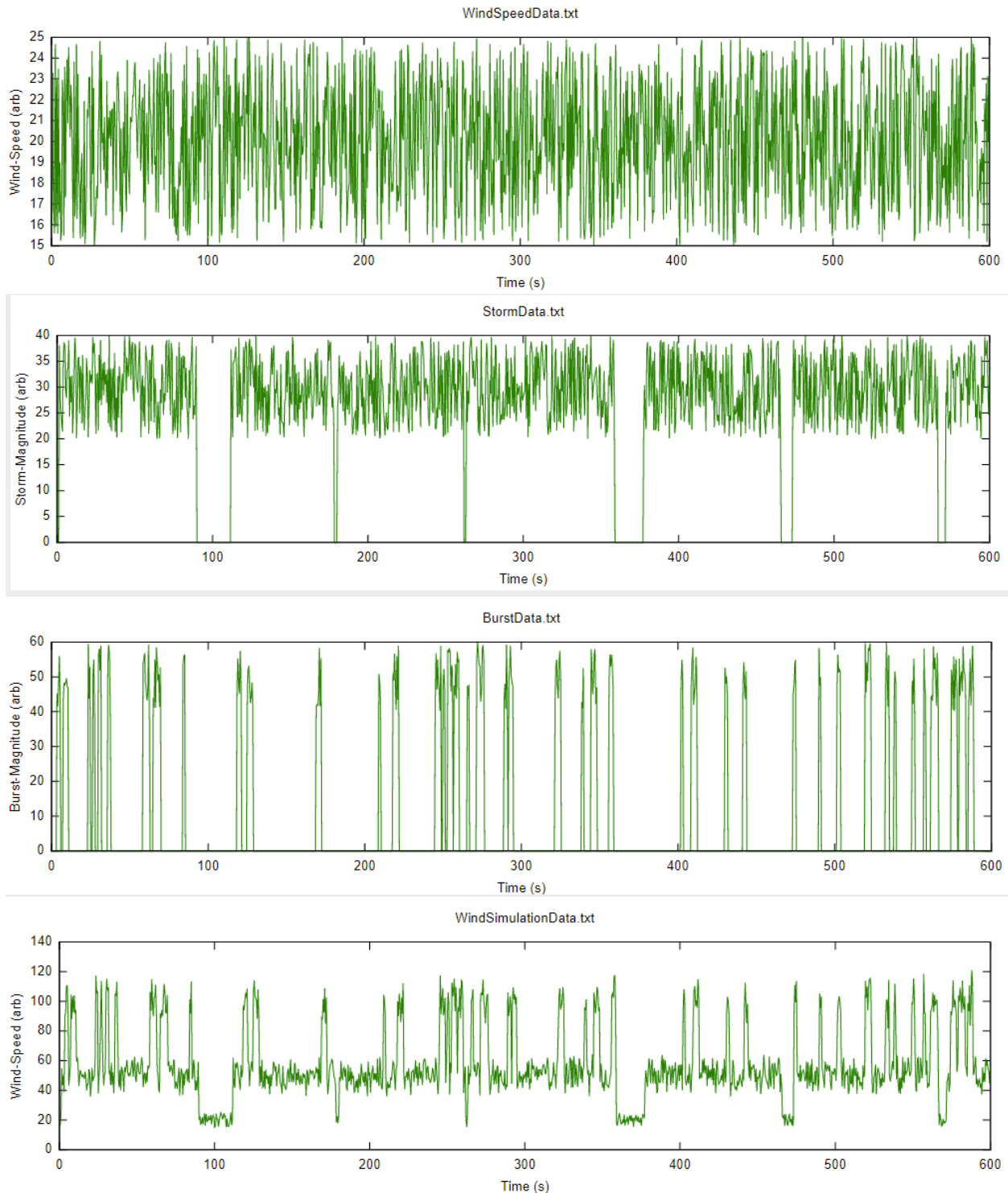
```
simulationConfiguration.txt - Notep...
File Edit Format View Help
20 5 600 0.5
0.05 20 40 50 100
0.05 40 60 1 5
Ln 3, Col 6 100% Windows (CRLF) UTF-8
```

Output Screen:



```
Microsoft Visual Studio Debug Console
Configuration performed successfully . . .
WindSpeedData.txt generated successfully . . .
StormData.txt generated successfully . . .
BurstData.txt generated successfully . . .
WindSimulationData.txt generated successfully . . .
-----
Thanks for using our services . . .
D:\Files\University-NYUAD\First Year\CPE\Assignments\Assignment.2\Assignment2Corrected\Debug\Assignment2Corrected.exe (process 7224) exited with code 0.
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the console when debugging stops.
Press any key to close this window . . .
```

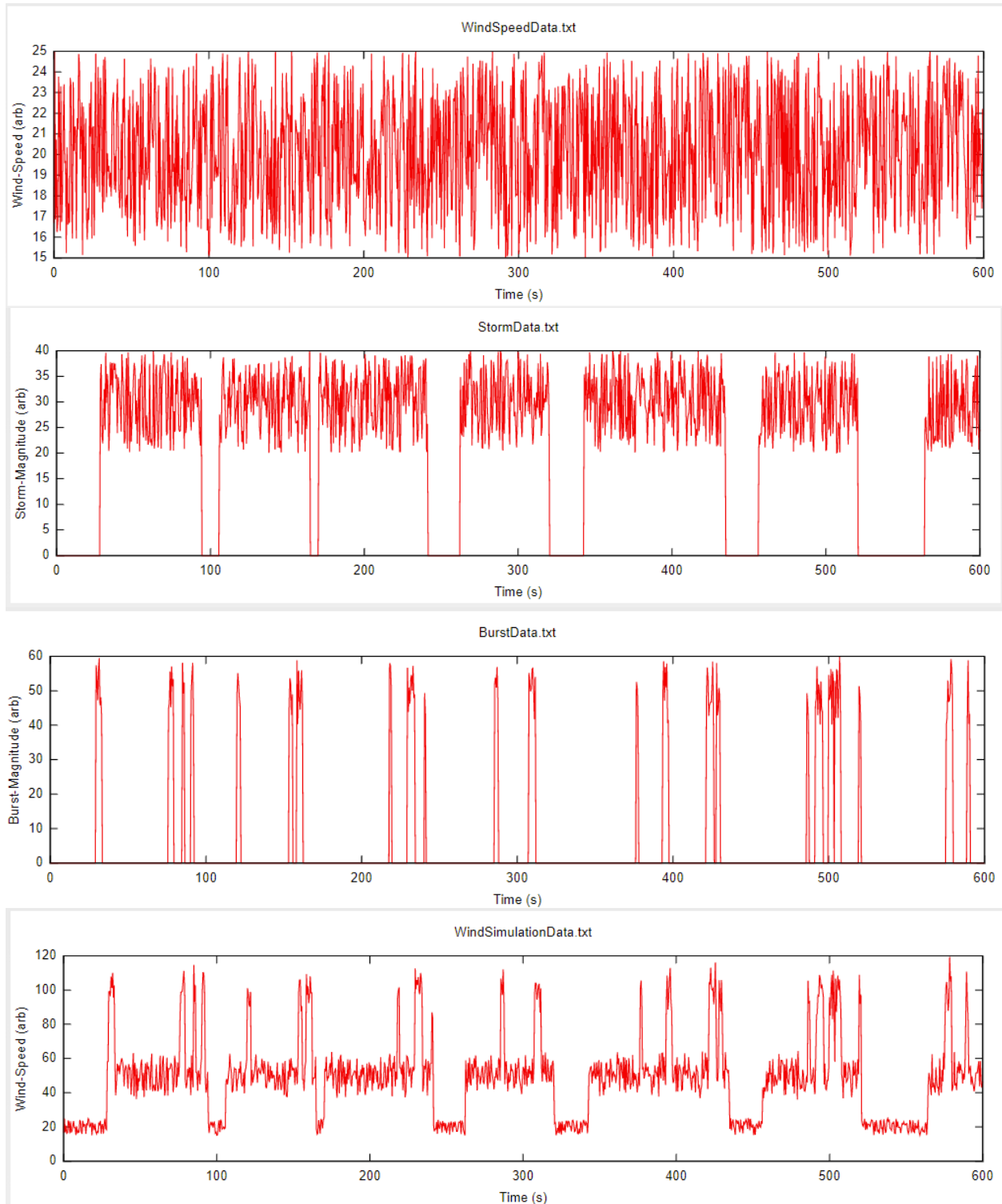
Graphs Generated from Output Files (Test 1):



(WindSimulationData.txt includes the binary value to show when the storm occurs but that couldn't be represented graphically. However, it was correctly showing when a storm was happening and when it wasn't)

Graphs Generated from Output Files (Test 2):



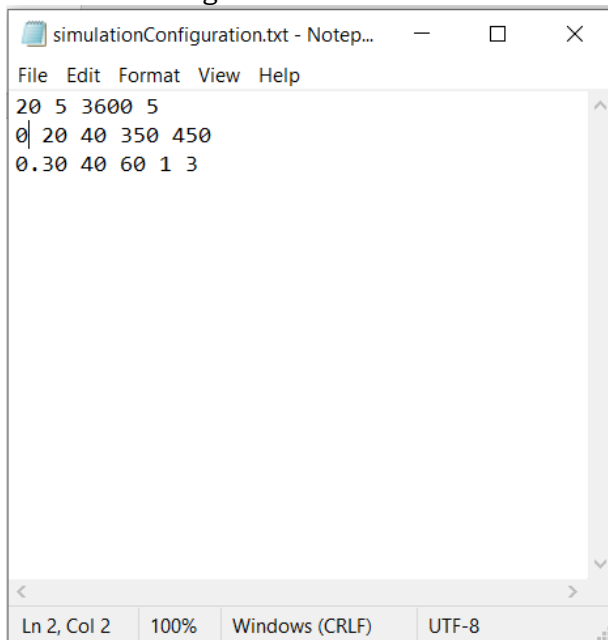


(WindSimulationData.txt includes the binary value to show when the storm occurs but that couldn't be represented graphically. However, it was correctly showing when a storm was happening and when it wasn't)

Since the output (though random) is as expected, this test case passes.

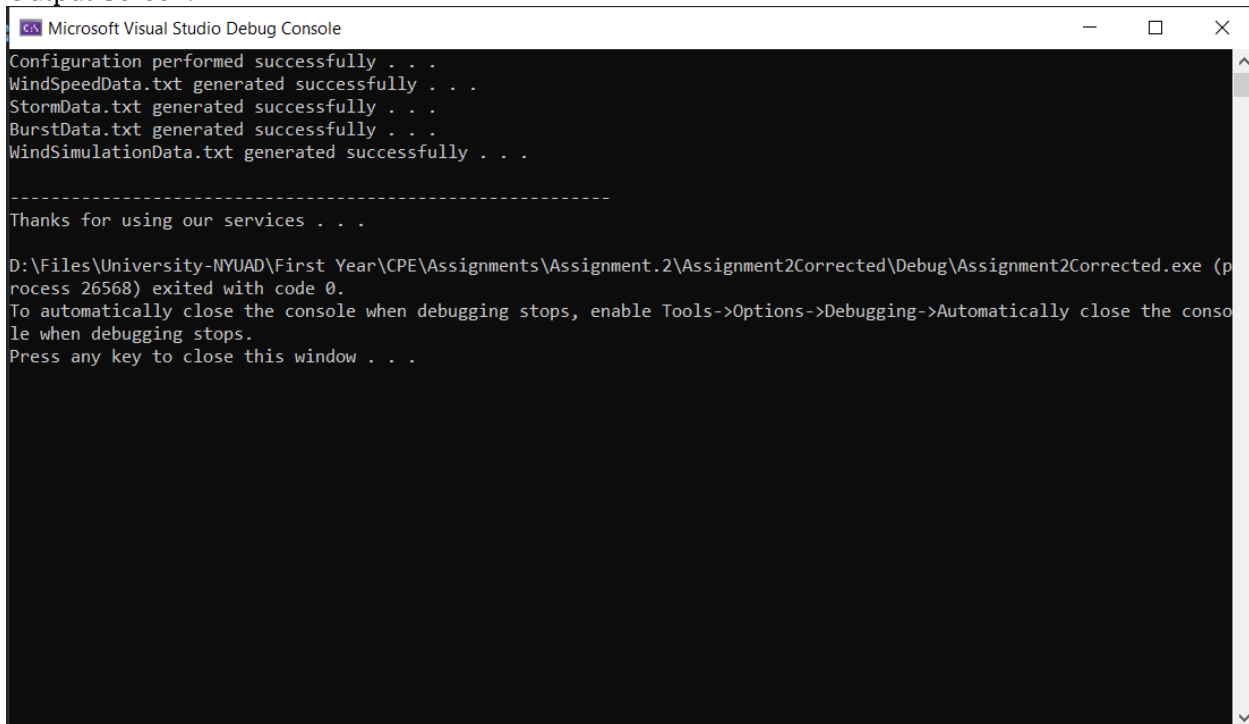
### Case 5: Sample Simulation with No Storms (0% Storm Probability):

simulationConfiguration.txt:



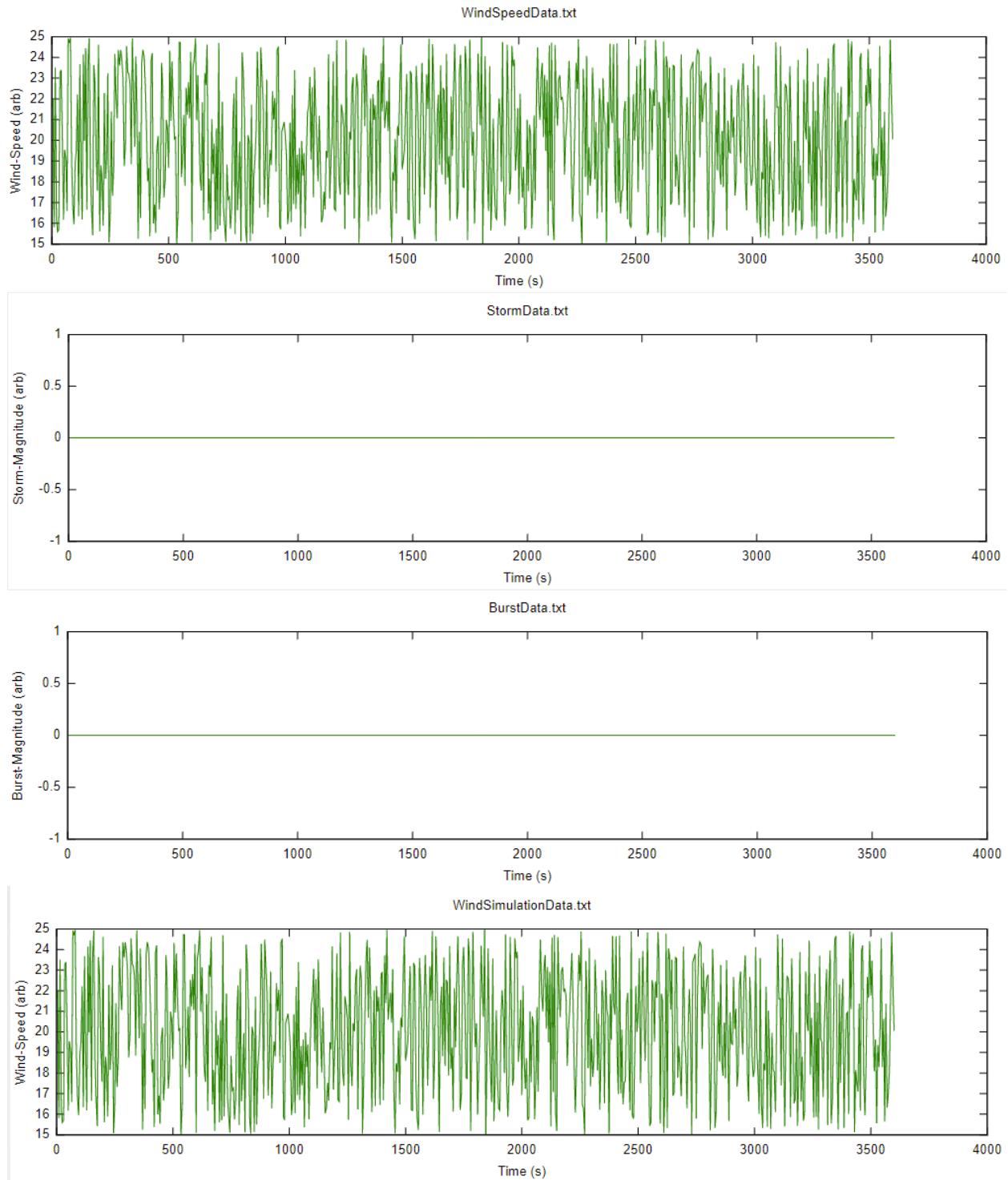
```
simulationConfiguration.txt - Notep...
File Edit Format View Help
20 5 3600 5
0 20 40 350 450
0.30 40 60 1 3
Ln 2, Col 2 100% Windows (CRLF) UTF-8
```

Output Screen:



```
Microsoft Visual Studio Debug Console
Configuration performed successfully . . .
WindSpeedData.txt generated successfully . . .
StormData.txt generated successfully . . .
BurstData.txt generated successfully . . .
WindSimulationData.txt generated successfully . . .
-----
Thanks for using our services . . .
D:\Files\University-NYUAD\First Year\CPE\Assignments\Assignment.2\Assignment2Corrected\Debug\Assignment2Corrected.exe (process 26568) exited with code 0.
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the console when debugging stops.
Press any key to close this window . . .
```

Graphs from output files:

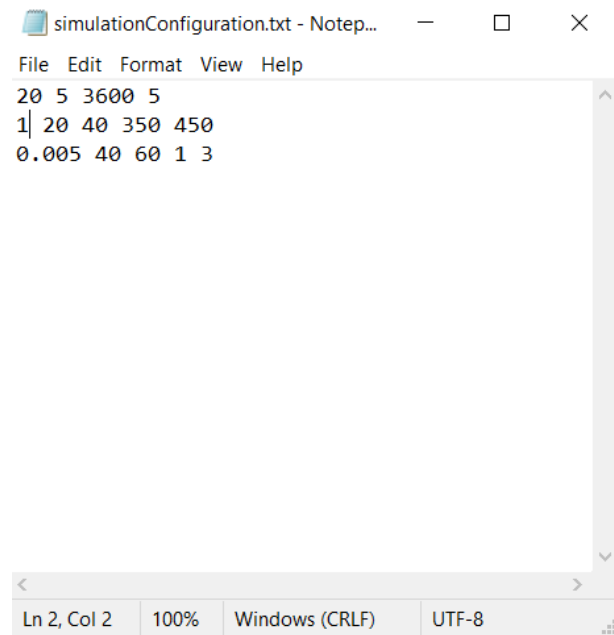


(WindSimulationData.txt includes the binary value to show when the storm occurs but that couldn't be represented graphically. However, it was correctly showing that a storm never occurs)

No storms or microbursts + WindSimulationData.txt is the same as WindSpeedData.txt => This case passes.

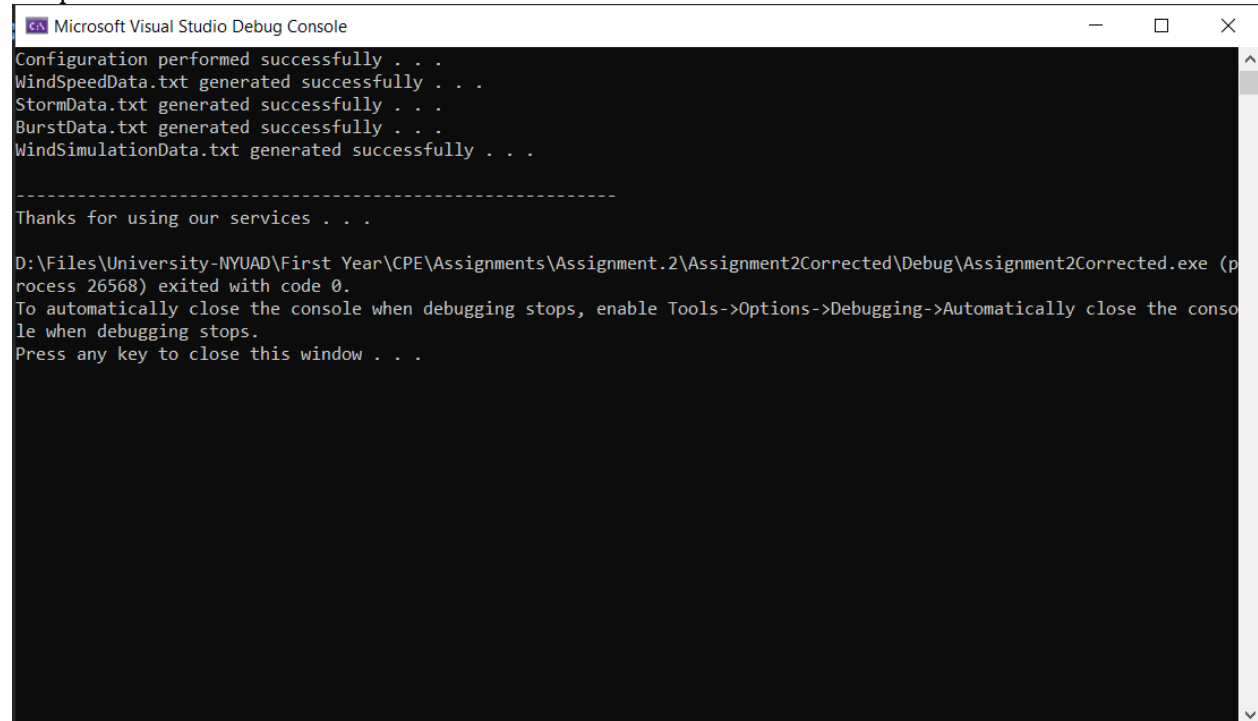
### Case 6: Sample Simulation Always with Storm (100% Storm Probability):

simulationConfiguration.txt:



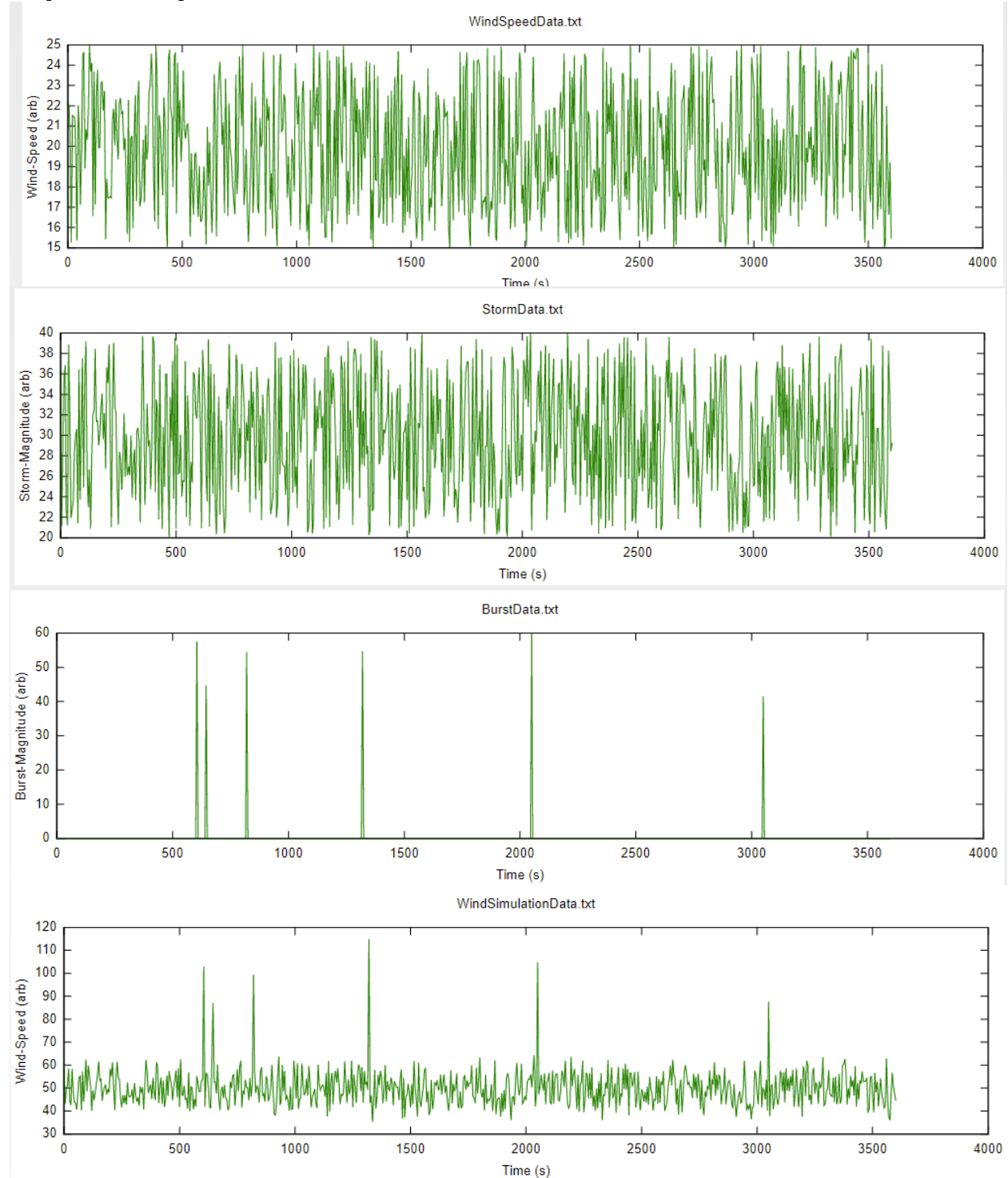
```
simulationConfiguration.txt - Notep...
File Edit Format View Help
20 5 3600 5
1| 20 40 350 450
0.005 40 60 1 3
Ln 2, Col 2 100% Windows (CRLF) UTF-8
```

Output Screen:



```
Microsoft Visual Studio Debug Console
Configuration performed successfully . . .
WindSpeedData.txt generated successfully . . .
StormData.txt generated successfully . . .
BurstData.txt generated successfully . . .
WindSimulationData.txt generated successfully . . .
-----
Thanks for using our services . . .
D:\Files\University-NYUAD\First Year\CPE\Assignments\Assignment.2\Assignment2Corrected\Debug\Assignment2Corrected.exe (p
rocess 26568) exited with code 0.
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the conso
le when debugging stops.
Press any key to close this window . . .
```

## Graphs from Output Files:

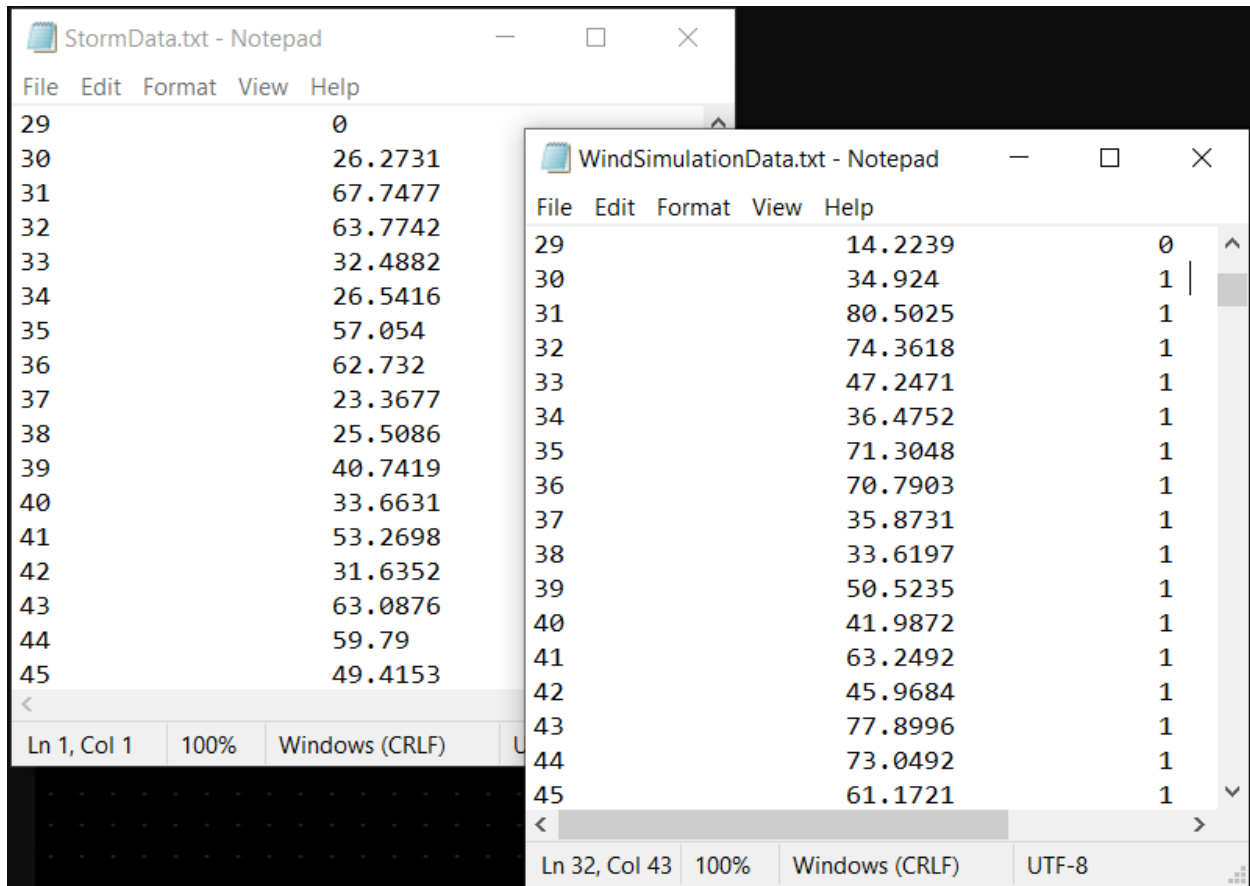


(WindSimulationData.txt includes the binary value to show when the storm occurs but that couldn't be represented graphically. However, it was correctly that there was always a storm)

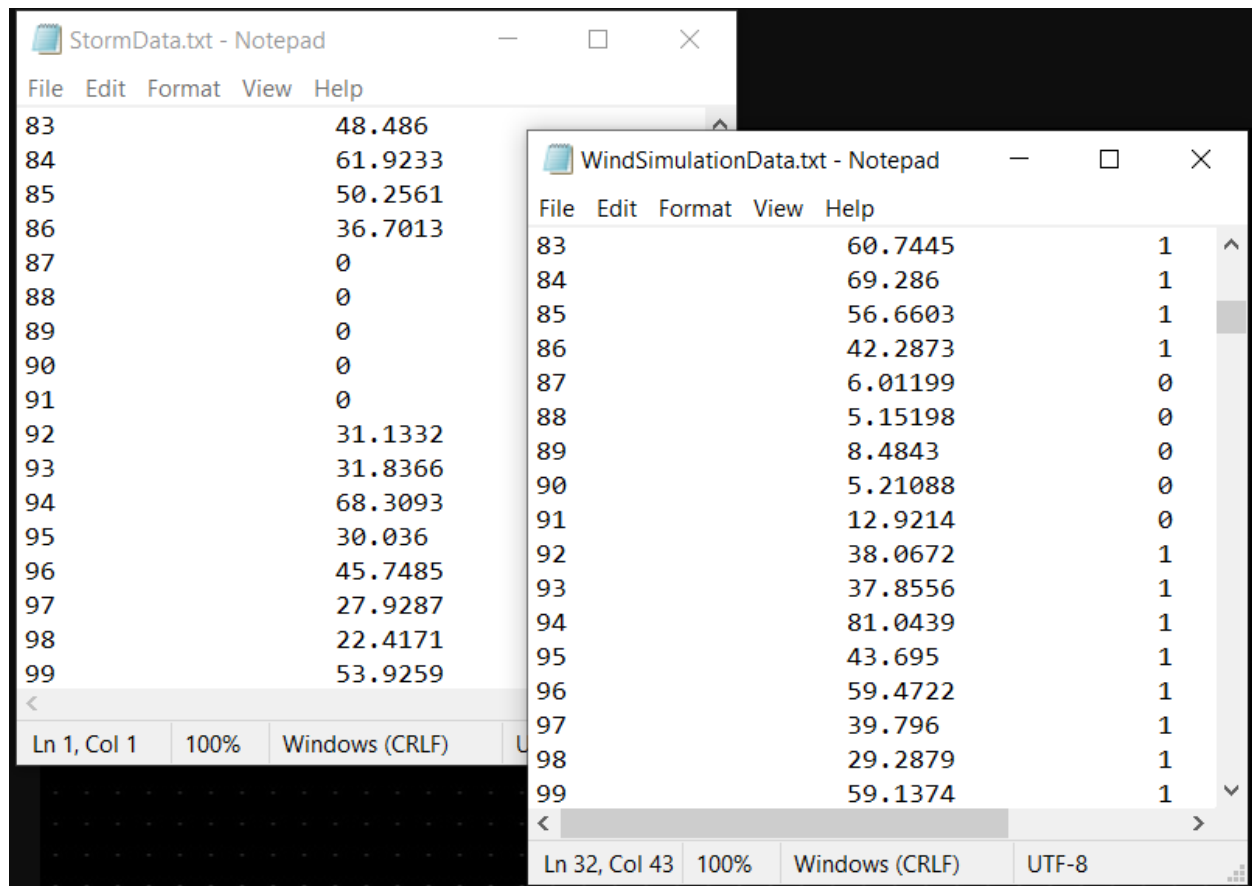
There is always a storm => This case passes

### The Right-Most Column of WindSimulationData.txt:

Below is to show that the binary digit which shows when there is a storm works:



The binary digit becomes 1 when the storm starts



The binary digit becomes 0 when the storm ends and 1 again when another storm starts.

## User Guide:

- Create a file called simulationConfiguration.txt and store it in the same folder as the .cpp file.
- Make sure the file is in the correct format (according to the aforementioned guidelines).
- Compile and run the file called Assignment2.cpp
- If there are no problems with the file, the code will output will write confirmation messages and create 4 files:
  - WindSpeedData.txt
  - StormData.txt
  - BurstData.txt
  - WindSimulationData.txt



## Credits:

Special thanks to the developer(s) of GNUPlotter for their file-to-graph tool.

<http://gnuplot.respawned.com/>

## Bibliography:

- Herzstein, Erin. "How Do Computers Generate Random Numbers?." *Medium*. N.p., 2021. Web. 21 Oct. 2021 . <<https://levelup.gitconnected.com/how-do-computers-generate-random-numbers-a72be65877f6>>.
- Hoffman, Chris. "How Computers Generate Random Numbers." *How-To Geek*. N.p., 2021. Web. 21 Oct. 2021 . <<https://www.howtogeek.com/183051/htg-explains-how-computers-generate-random-numbers/>>.