**National Institute of Technology, Warangal**

**End Lab Exam – MCA 5th Sem**



***Software Testing***

**Name – Divyanshu Shrivastava**

**Roll No. – 187909**

**MCA 3rd Year**

**Reg. No. – MC18111**

**Task 1 – TCASES**

**Definition –**

Tcases is a tool for designing tests. It doesn't matter what kind of system you are testing. Nor does it matter what level of the system you are testing — unit, subsystem, or full system. You can use Tcases to design your tests in any of these situations. With Tcases, you define the input space for your system-under-test and the level of coverage that you want. Then Tcases generates a minimal set of test cases that meets your requirements.

Tcases is primarily a tool for black-box test design. For such tests, the concept of "coverage" is different from structural testing critieria such as line coverage, branch converage, etc. Instead, Tcases is guided by coverage of the input space of your system.

**How Tcases Work?**

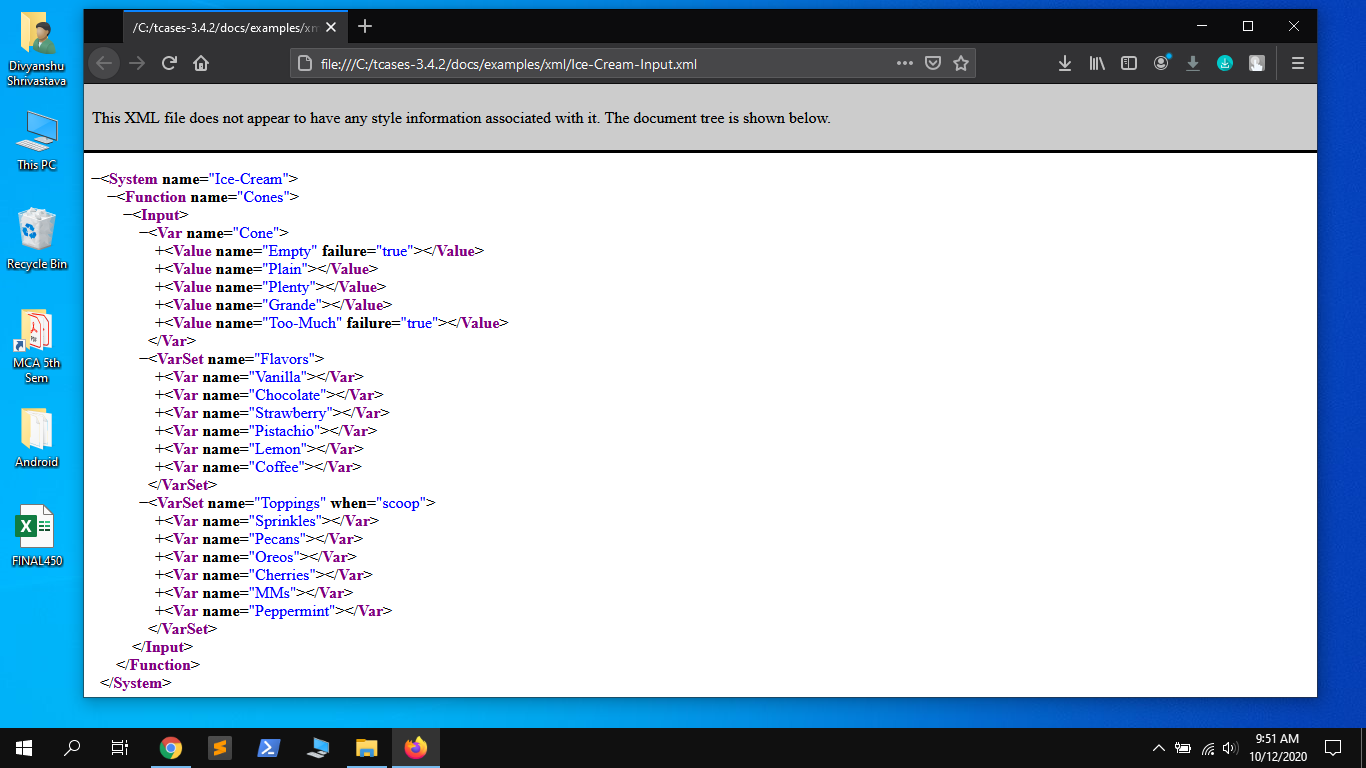
First, you create a system input definition, a document that defines your system as a set of functions. For each system function, the system input definition defines the variables that characterize the function input space.

Then, you can create a generator definition. That's another document that defines the coverage you want for each system function. The generator definition is optional. You can skip this step and still get a basic level of coverage.

Finally, you run Tcases. Tcases is a Java program that you can run from the command line or from your favorite IDE. Tcases comes with built-in support for running using a shell script or an **ant** target. You can also run Tcases with Maven using the Tcases Maven Plugin. Using your input definition and your generator definition, Tcases generates a system test definition. The system test definition is a document that lists, for each system function, a set of test cases that provides the specified level of coverage. Each test case defines a specific value for every function input variable. Tcases generates not only valid input values that define successful test cases but also invalid values for the tests cases that are needed to verify expected error handling.

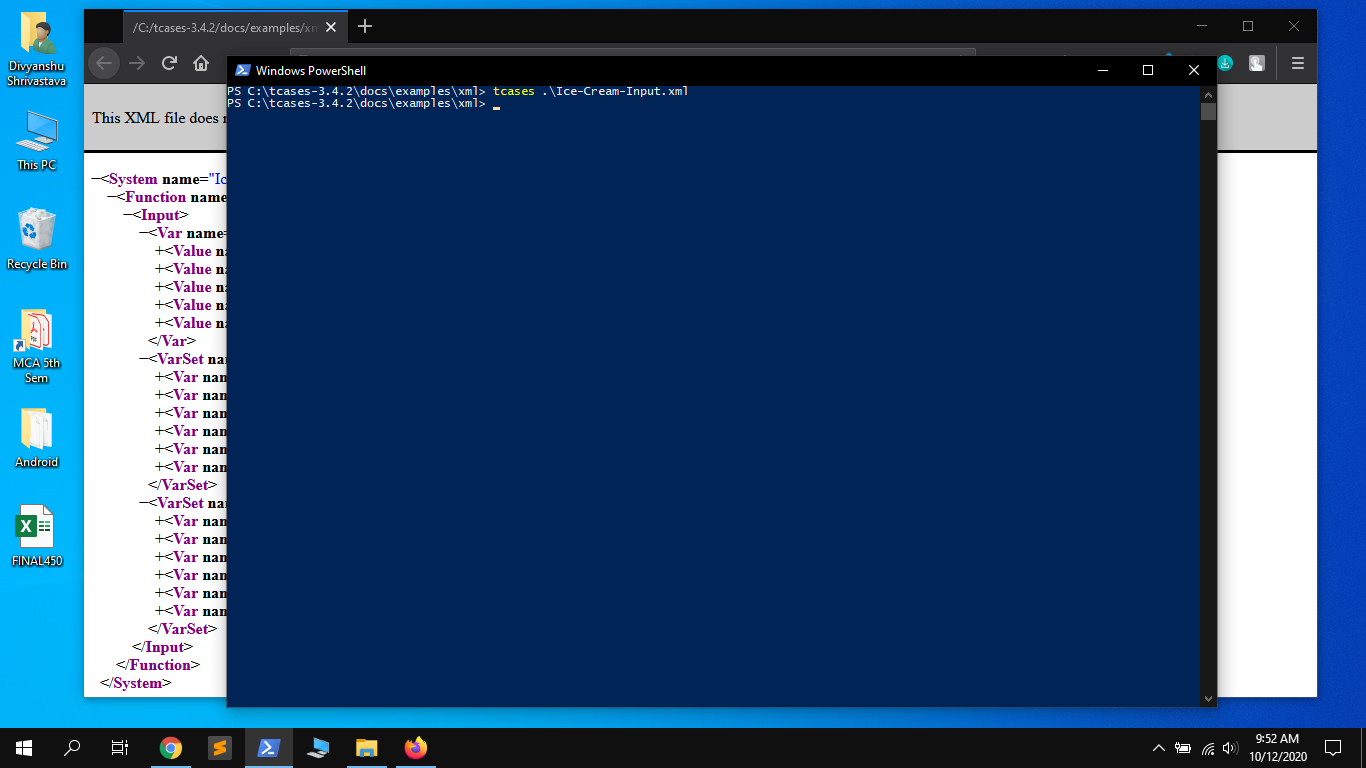
**Running XML File**

1. Input File – **Ice-Cream-Input.xml**



The above image is showing the input file that we are using for Tcases which mainly three types of input Cone, Flavors and Toppings.

1. **Running Input File.**

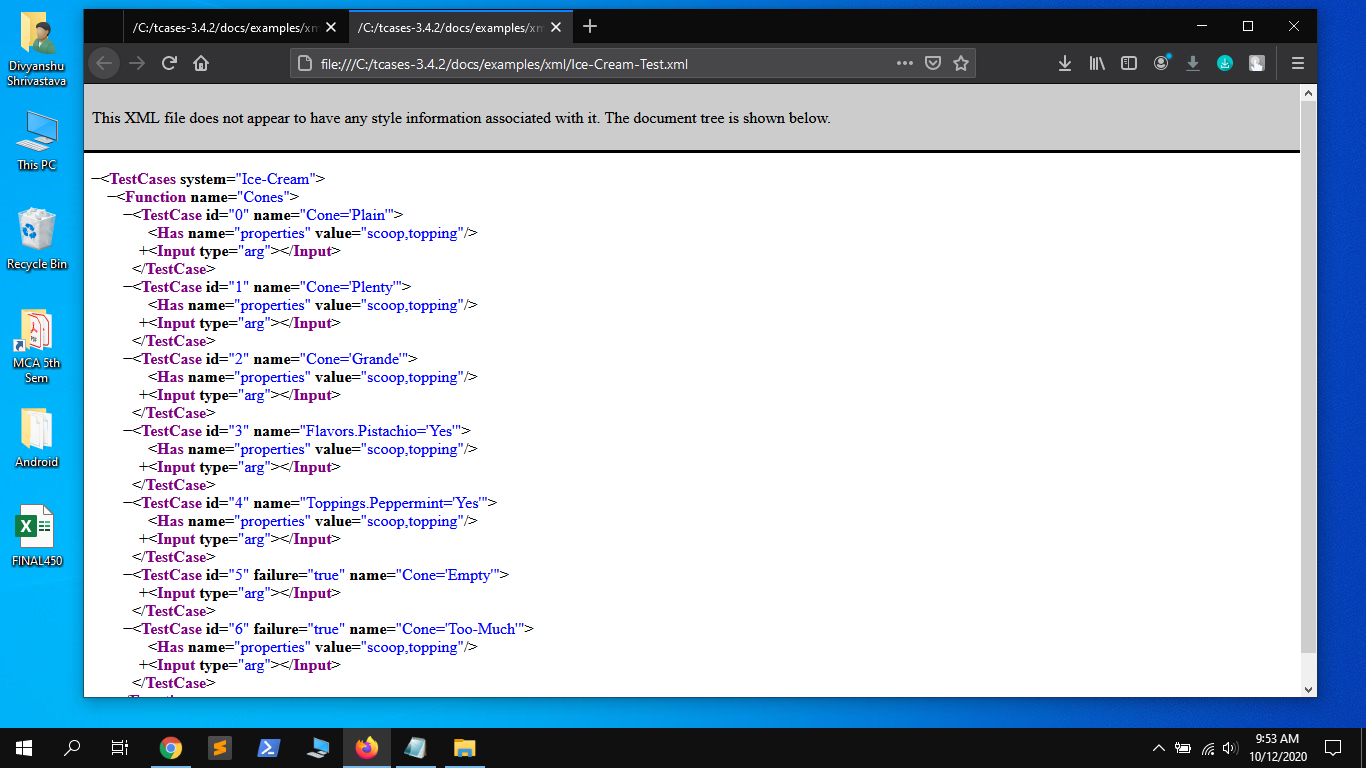


We have executed the input file by using the command –

**tcases Ice-Cream-Input.xml**

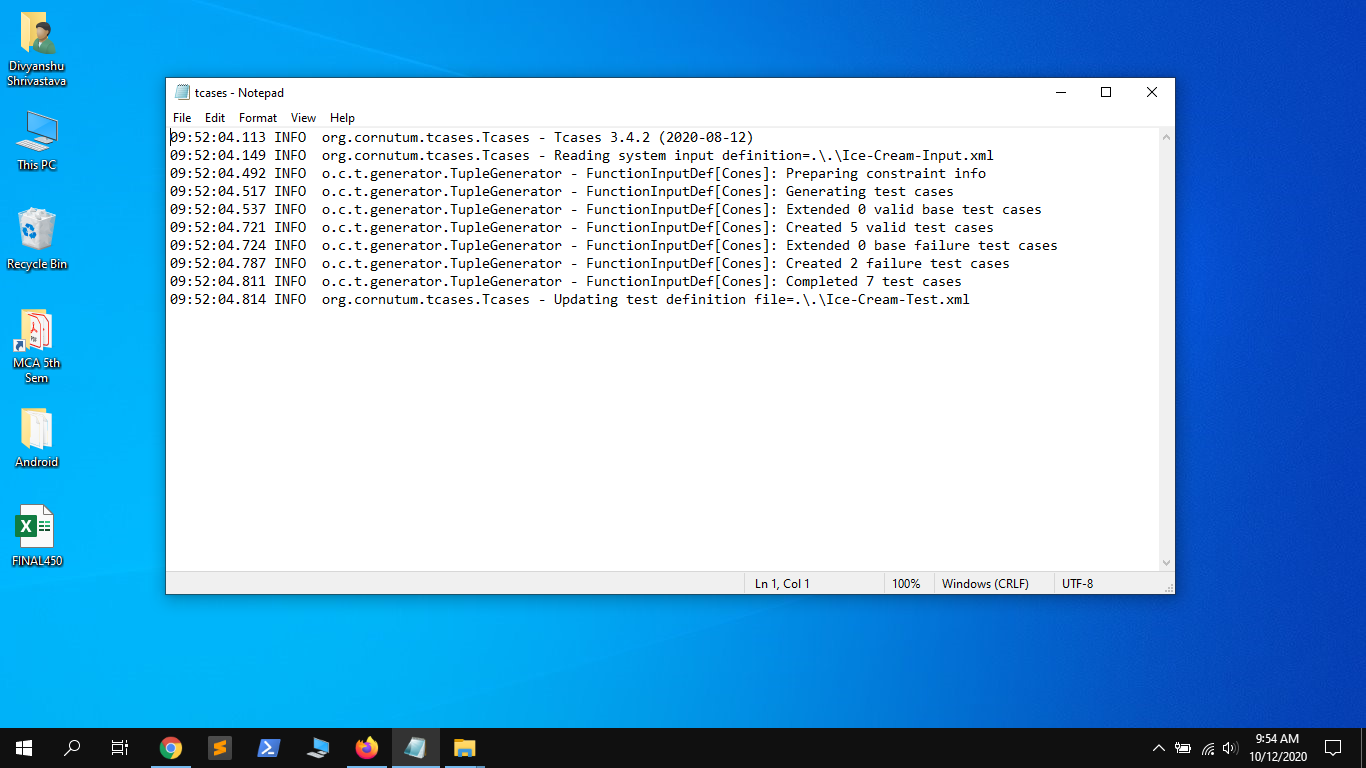
which when completed successfully generates output file Ice-Cream-Test.xml within the same directory.

1. **Output File – Ice-Cream-Test.xml**



From the output file we can observe that it gives 7 different test ids by the combination of values present.

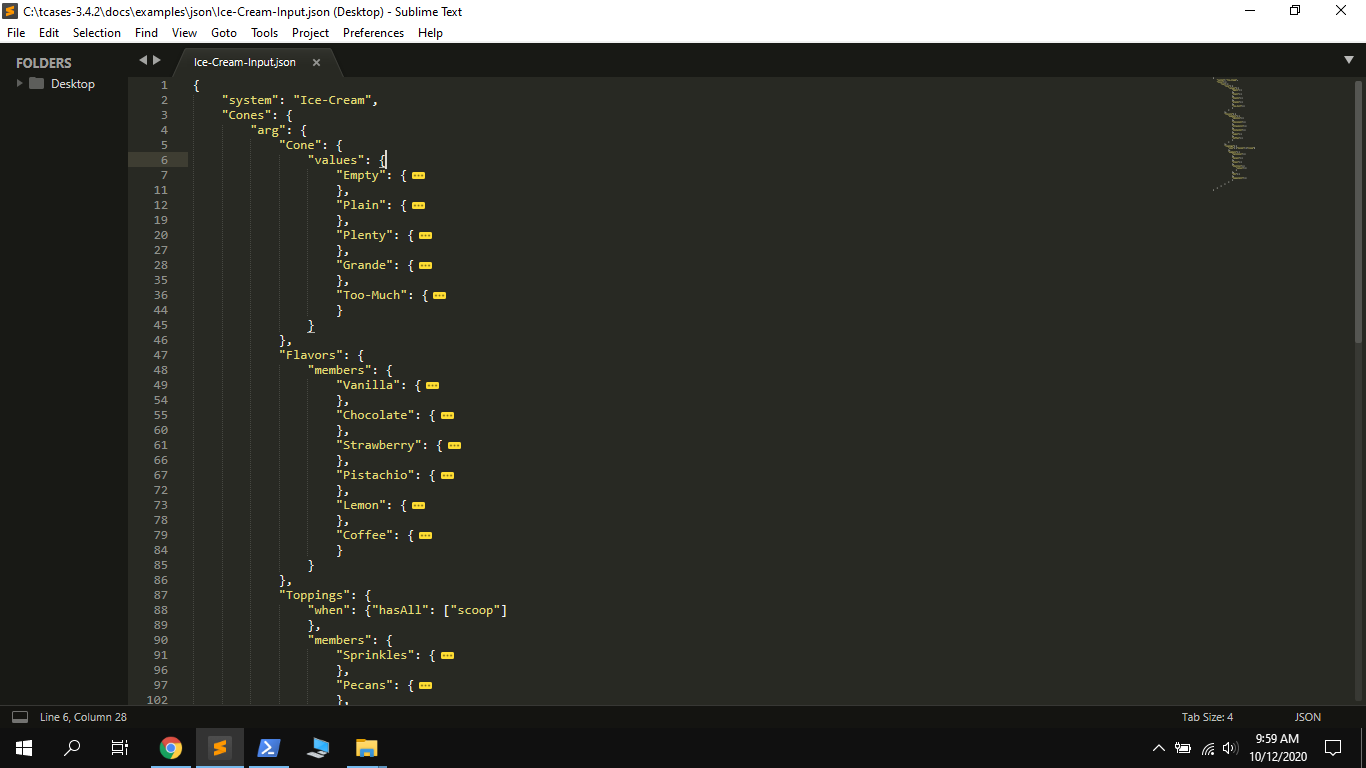
1. **Tcases log file**



The tcases.log file clearly shows that it created 5 valid test cases and 2 failure or invalid test Cases out of total 7 test cases.

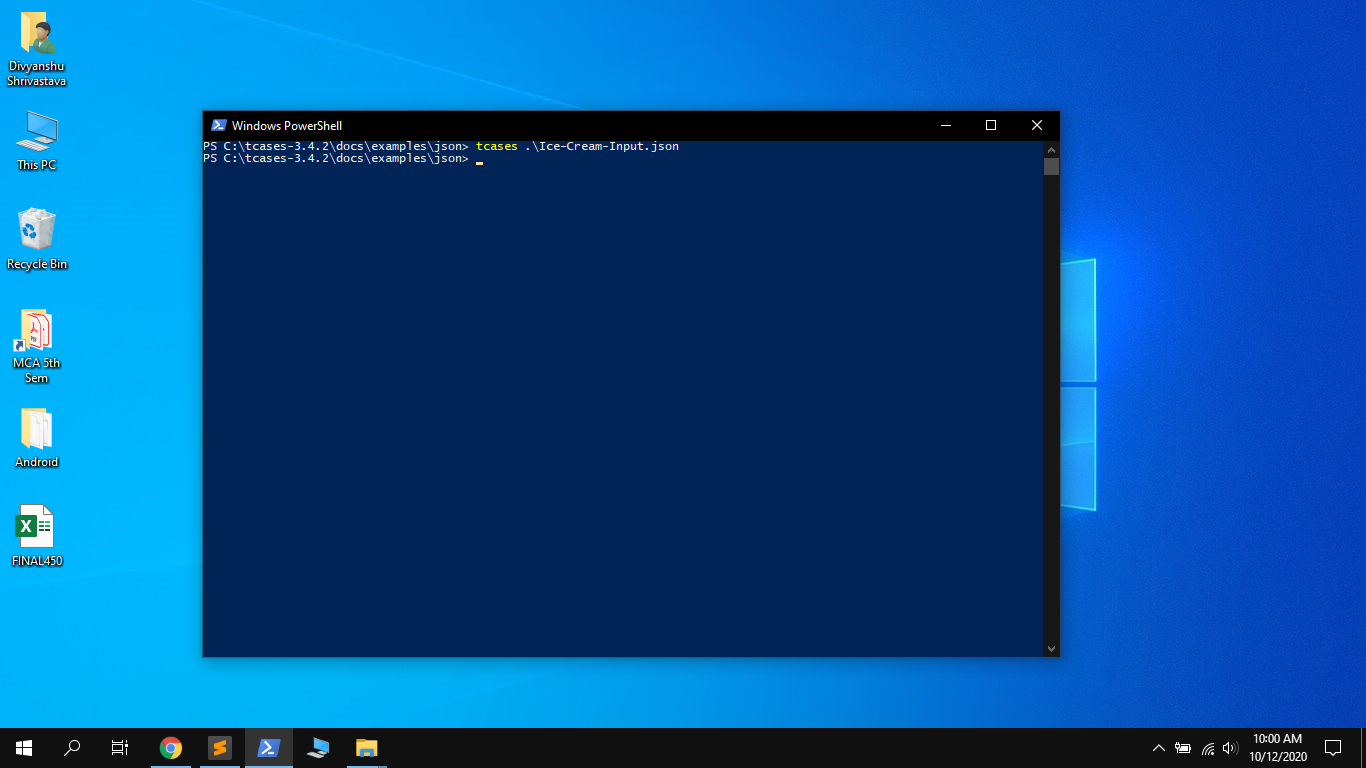
**Running JSON File**

1. Input File – **Ice-Cream-Input.json**



The above image is showing the input file that we are using for Tcases which mainly three types of input Cone, Flavors and Toppings and each of these having subparts.

1. **Running Input File.**

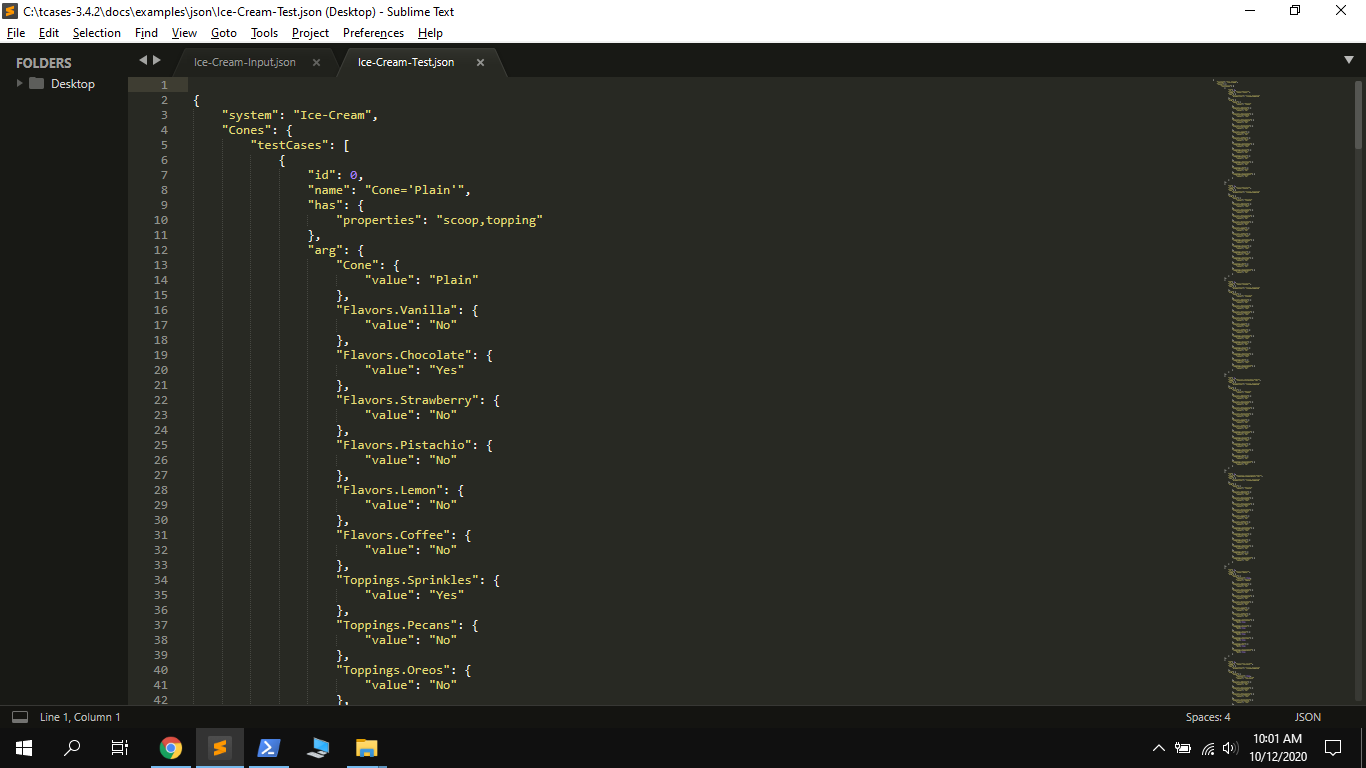


We have executed the input file by using the command –

**tcases Ice-Cream-Input.json**

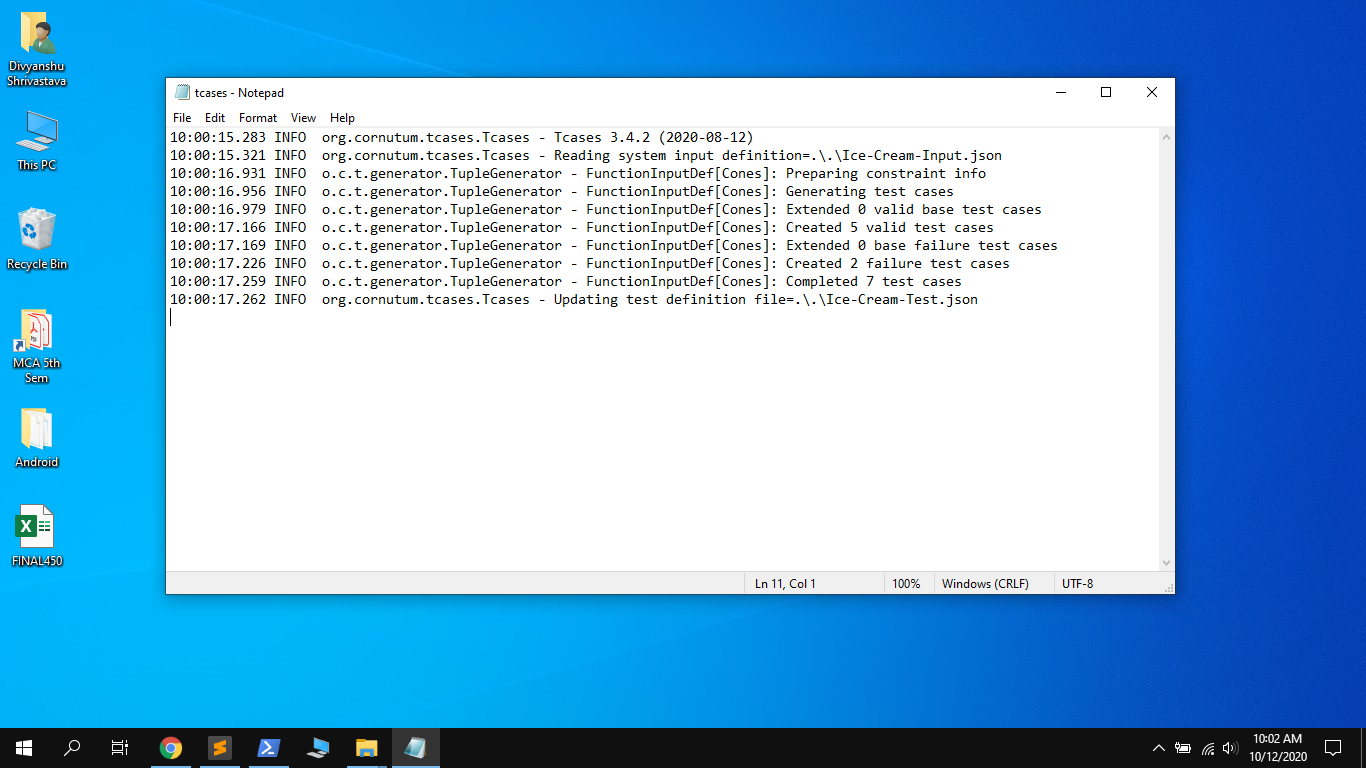
which when completed successfully generates output file Ice-Cream-Test.json within the same directory.

1. **Output File – Ice-Cream-Test.json**



From the output file we can observe that it gives 7 different test ids by the combination of values present.

1. **Tcases log file**



The tcases.log file clearly shows that it created 5 valid test cases and 2 failure or invalid test Cases out of total 7 test cases.