Mutation Function Implementation:

First we generate a random number between 1 to 1000 and if number is even then we call even_mutation function otherwise we call odd_mutation function

even_mutation :

first we flip a random bit of each input value then generate a random number between 1 to 1000 and if number is even then we flip the sign of each input otherwise we multiply each input by 2

odd_mutation :

first take two random numbers between -500 and 500 and take bitwise XOR and get a value mask then take bitwise XOR of each input value and mask then if number is even then we flip the sign of each input otherwise we multiply each input by 2

updateTotalCoverage:

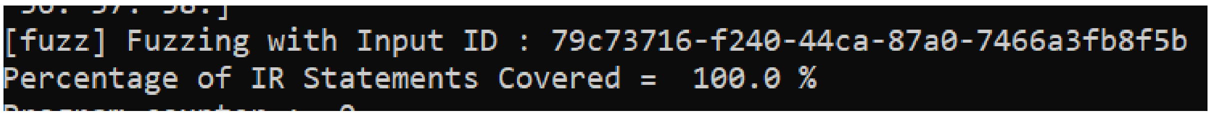
Inside updateTotalCoverage function we update the total metric with value that has been taking by union of both current metric and total metric.

compareCoverage:

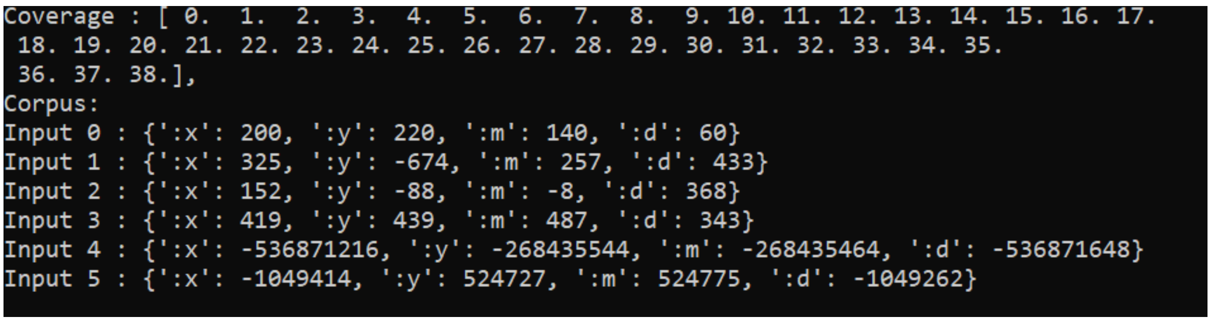
Inside compareCoverage function we check that our current metric has some extra IR statements that has not been covered so far if yes then we return true else we return false.Basically we check that we get improvements in our inputs.

Five Interesting Test Cases:

1. task2.tl

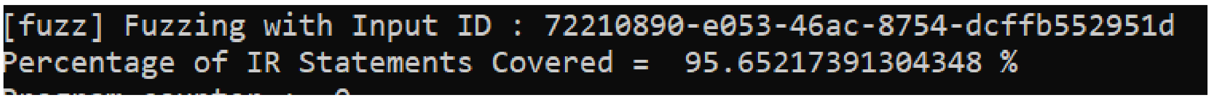


```
[fuzz] Fuzzing with Input ID : 79c73716-f240-44ca-87a0-7466a3fb8f5b
Percentage of IR Statements Covered =  100.0 %
```

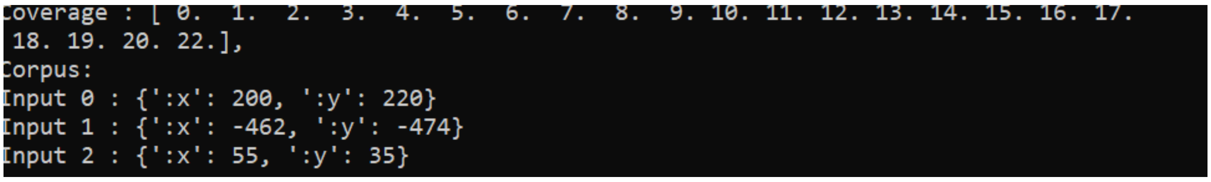


```
Coverage : [ 0.  1.  2.  3.  4.  5.  6.  7.  8.  9. 10. 11. 12. 13. 14. 15. 16. 17.
 18. 19. 20. 21. 22. 23. 24. 25. 26. 27. 28. 29. 30. 31. 32. 33. 34. 35.
 36. 37. 38.],
Corpus:
Input 0 : {':x': 200, ':y': 220, ':m': 140, ':d': 60}
Input 1 : {':x': 325, ':y': -674, ':m': 257, ':d': 433}
Input 2 : {':x': 152, ':y': -88, ':m': -8, ':d': 368}
Input 3 : {':x': 419, ':y': 439, ':m': 487, ':d': 343}
Input 4 : {':x': -536871216, ':y': -268435544, ':m': -268435464, ':d': -536871648}
Input 5 : {':x': -1049414, ':y': 524727, ':m': 524775, ':d': -1049262}
```

2. task4.tl

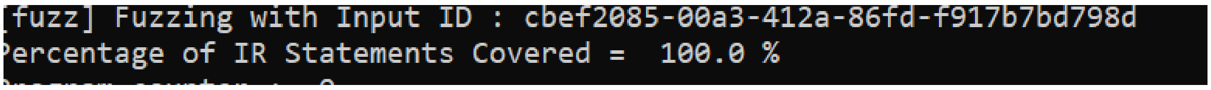


```
[fuzz] Fuzzing with Input ID : 72210890-e053-46ac-8754-dcffb552951d
Percentage of IR Statements Covered =  95.65217391304348 %
```

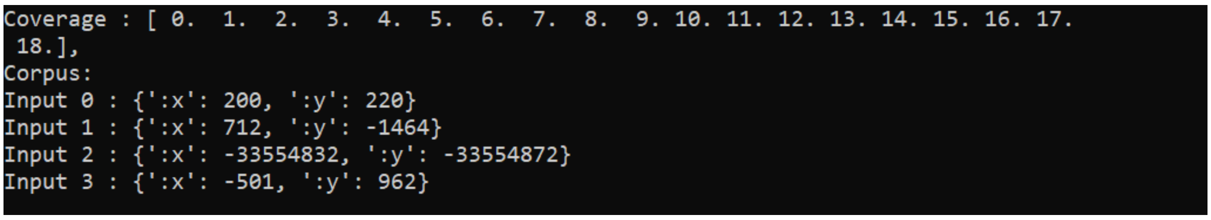


```
Coverage : [ 0.  1.  2.  3.  4.  5.  6.  7.  8.  9. 10. 11. 12. 13. 14. 15. 16. 17.
 18. 19. 20. 22.],
Corpus:
Input 0 : {':x': 200, ':y': 220}
Input 1 : {':x': -462, ':y': -474}
Input 2 : {':x': 55, ':y': 35}
```

3. task5.tl

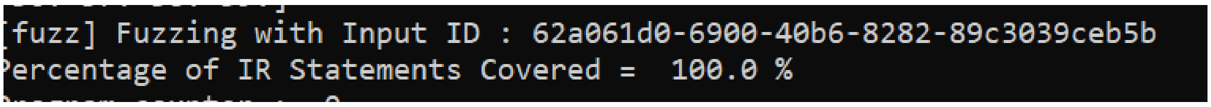


```
[fuzz] Fuzzing with Input ID : cbef2085-00a3-412a-86fd-f917b7bd798d
Percentage of IR Statements Covered =  100.0 %
```

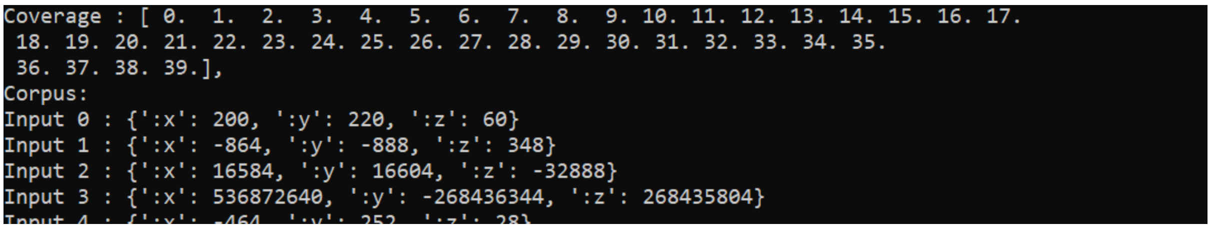


```
Coverage : [ 0.  1.  2.  3.  4.  5.  6.  7.  8.  9. 10. 11. 12. 13. 14. 15. 16. 17.
 18.],
Corpus:
Input 0 : {':x': 200, ':y': 220}
Input 1 : {':x': 712, ':y': -1464}
Input 2 : {':x': -33554832, ':y': -33554872}
Input 3 : {':x': -501, ':y': 962}
```

4. task6.tl

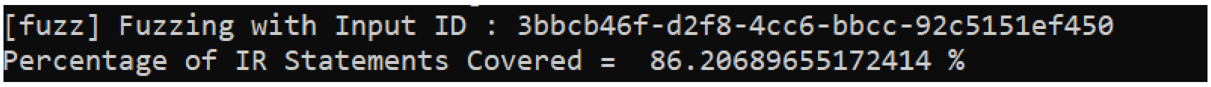


```
[fuzz] Fuzzing with Input ID : 62a061d0-6900-40b6-8282-89c3039ceb5b
Percentage of IR Statements Covered =  100.0 %
```

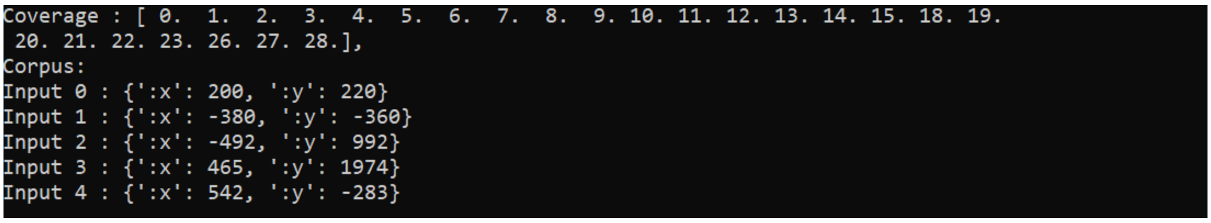


```
Coverage : [ 0.  1.  2.  3.  4.  5.  6.  7.  8.  9. 10. 11. 12. 13. 14. 15. 16. 17.
 18. 19. 20. 21. 22. 23. 24. 25. 26. 27. 28. 29. 30. 31. 32. 33. 34. 35.
 36. 37. 38. 39.],
Corpus:
Input 0 : {':x': 200, ':y': 220, ':z': 60}
Input 1 : {':x': -864, ':y': -888, ':z': 348}
Input 2 : {':x': 16584, ':y': 16604, ':z': -32888}
Input 3 : {':x': 536872640, ':y': -268436344, ':z': 268435804}
```

5. task7.tl



```
[fuzz] Fuzzing with Input ID : 3bbcb46f-d2f8-4cc6-bbcc-92c5151ef450
Percentage of IR Statements Covered =  86.20689655172414 %
```



```
Coverage : [ 0.  1.  2.  3.  4.  5.  6.  7.  8.  9. 10. 11. 12. 13. 14. 15. 18. 19.
 20. 21. 22. 23. 26. 27. 28.],
Corpus:
Input 0 : {':x': 200, ':y': 220}
Input 1 : {':x': -380, ':y': -360}
Input 2 : {':x': -492, ':y': 992}
Input 3 : {':x': 465, ':y': 1974}
Input 4 : {':x': 542, ':y': -283}
```

Limitations:

1. **Randomness and Reproducibility**: The code relies heavily on random number generation (np.random.randint). This randomness may lead to unpredictable behavior, making it challenging to debug or reproduce specific results.
2. **Potential Data Loss**: The code applies XOR and arithmetic operations on data values, which can lead to data loss or distortion. Depending on the application, this may or may not be desired behavior.
3. **Loss of Original Data**: The code alters the data values significantly. If you need to retain the original data or want to track the transformations applied, this code doesn't provide any mechanism for that.

Assumption:

1. **Data Format**: The code assumes that the data in input_data is numerical and can be subjected to bitwise and arithmetic operations.
2. **Bitwise Operations**: The code uses bitwise XOR (^) to mask the data and may be assuming that this operation is a suitable way to obfuscate or modify the data.
3. **Data Structure**: The code assumes that input_data is a custom data structure with a data attribute, and it operates on the values within this structure.
4. **Conditional Transformations**: The code introduces conditional transformations based on the result of (test % 2 == 0). It assumes that applying different transformations (negation or doubling) depending on the random test result is a meaningful operation.