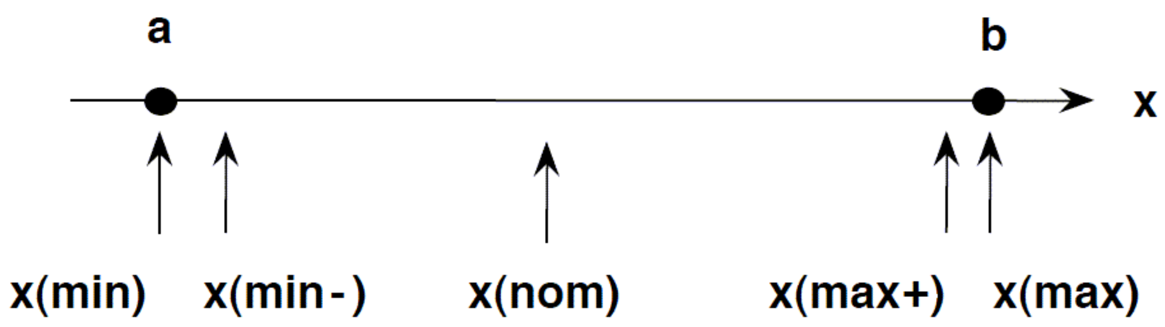
BOUNDARY VALUE TESTING

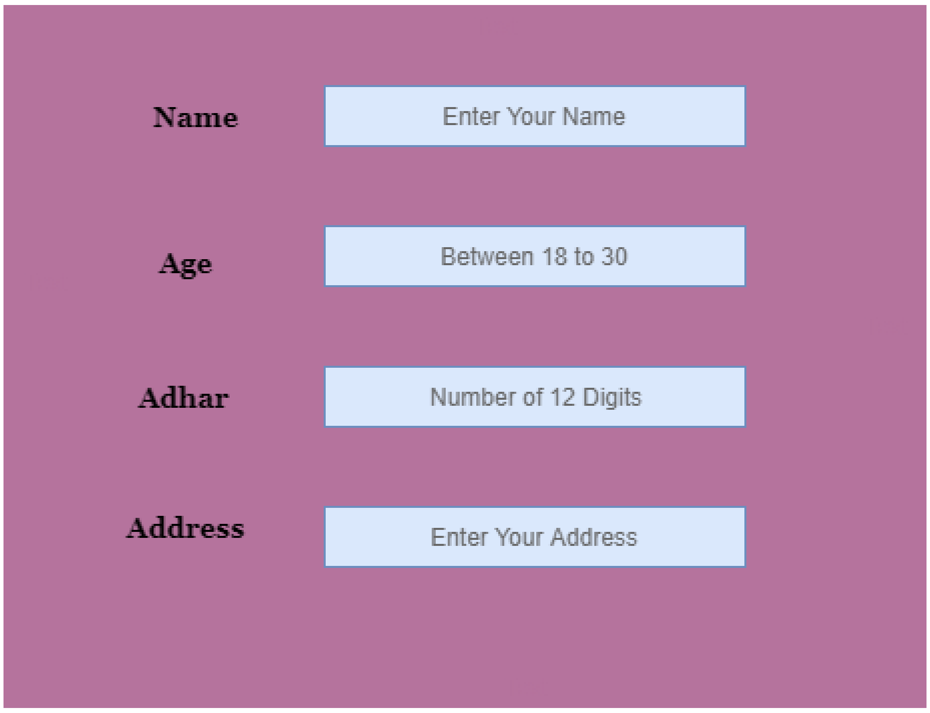
Boundary Value Analysis is a popular technique for black box testing. It is used to identify defects and errors in software by testing input values on the boundaries of the allowable ranges.

The goal of boundary value analysis is to find any issues which may arise due to incorrect assumptions about the system behavior. Testing these boundary values ensures that the software functions correctly.



EXAMPLE:

There is 18 and 30 are the boundary values that's why tester pays more attention to these values, but this doesn't mean that the middle values like 19, 20, 21, 27, 29 are ignored. Test cases are developed for each and every value of the range.



Name	Enter Your Name
Age	Between 18 to 30
Adhar	Number of 12 Digits
Address	Enter Your Address

Testing of boundary values is done by making valid and invalid partitions. Invalid partitions are tested because testing of output in adverse condition is also essential. Tester develops test cases for both valid and invalid partitions to capture the behaviour of the system on different input conditions.

```
12  14   15   16   17  18  20 22  24  25 26  28  30  31  32  34  36  38  40
--------------------------------|---------------------------------|------------------------------
Invalid Partition                     Valid Partition                    Invalid Partition
```

Invalid test cases	Valid test cases	Invalid test cases
11, 13, 14, 15, 16, 17	18, 19, 24, 27, 28, 30	31, 32, 36, 37, 38, 39

Every partition for boundary analysis has its minimum and maximum values. Let's look at the following points to understand BVA in software testing in brief. For every variable, testers check the following.

- Nominal value
- Minimum value
- Above minimum value
- Below the minimum value
- Maximum value
- Boundary value for an invalid partition is called invalid boundary value
- Boundary value for valid partition is called boundary value

Single Fault Assumption: When more than one variable for the same application is checked then one can use a single fault assumption. Holding all but one variable to the extreme value and allowing the remaining variable to take the extreme value.

Example

Input: *Day, Month, Year with valid ranges as-*
1 ≤ Month≤12
1 ≤ Day ≤31
1900 ≤ Year ≤ 2000

Solution: Taking the year as a Single Fault Assumption i.e. year will be having values varying from 1900 to 2000 and others will have nominal values like Day (1 to 31) and Month (1 to 12)

Test Cases	Month	Day	Year	Output
1	6	15	1990	14 June 1990
2	6	15	1901	14 June 1901
3	6	15	1960	14 June 1960
4	6	15	1999	14 June 1999
5	6	15	2000	14 June 2000

.

Equivalence partitioning

Equivalence partitioning is a black-box testing technique that allows testers to group input data into sets or classes, making it possible to reduce the number of test cases while still achieving comprehensive coverage. This technique is particularly useful when dealing with a large range of input values

Advantages of Boundary Value Analysis

1. It is easier and faster to find defects using this technique. This is because the density of defects at boundaries is more.

2. Instead of testing will all sets of test data, we only pick the one at the boundaries. So, the overall test execution time reduces.

Disadvantages of boundary value analysis

1. The success of the testing using this technique depends on the equivalence classes identified, which further depends on the expertise of the tester and his knowledge of the application. Hence, incorrect identification of equivalence classes leads to incorrect boundary value testing.

2. Applications with open boundaries or applications not having one-dimensional boundaries are not suitable for this technique. In those cases, other black-box techniques like “Domain Analysis” are used.

Generalizing boundary value analysis

Generalizing Boundary Value Analysis BVA can be generalized in 2 ways:

- By the number of variables (easy)

• By the kind of ranges (quite difficult)

Generalizing by the number of variables:

- If there are functions with n variables, hold all but one at the nominal value and let the remaining variables assume min, min+, nom, max-, max values, repeating this for each variable.
- Thus, for a function of n variables, BVA yields 4n + 1 unique test cases.

Generalizing by the kinds of range:

- Depends on the nature/ type of variables themselves.
- When no explicit bounds are specified, as in Triangle problems, we usually have to create artificial bounds.
- The lower bound of side of lengths is 1, but what might be the upper bound?
- One possibility is, largest representable integer (MAXINT)
- Or impose an arbitrary upper limit such as 200 or 2000.
- BVA does not make much sense for Boolean variables: because the extreme values are TRUE or FALSE, and there is no clear choice for the remaining values.

Limitations of Boundary Value Analysis :

- It works well when the product is under test.
- It cannot consider the nature of the functional dependencies of variables.
- BVA is quite fundamental testing(incomplete).

Robustness Testing

- Robustness is a measure of how well a software system can cope with invalid inputs or unexpected user interactions.

- A robust system is one that continues to function correctly even in the face of unexpected or invalid inputs.

- A software system that is robust is able to handle errors and unexpected inputs gracefully, without crashing or producing incorrect results.

- It is black-box testing, where QA professionals have no knowledge about the system's internal working and implementation details.
- It involves validating the system with a wide range of invalid, unexpected, or out-of-range inputs.
- QA professionals validate the system's ability to handle unexpected errors and exceptions without crashing or failing.
- Robustness testing is usually done at the later stages of the software development life cycle (SDLC), after verifying that the software works well under normal conditions.
- It also verifies the system's behaviour under stress, such as high traffic volume and sudden increase in user requests. The system should be stable and perform consistently in such scenarios.
- simple example: Consider you develop a small program that takes input ranging from only 1 to 10. When you test the program for robustness, you provide inputs out of range, such as 0, -2, 11, etc. Even if you provide an invalid input, your program must respond as 'Invalid Input' rather than behaving unexpectedly.

Why is Robustness Testing Important?

- **Identify Unexpected Errors:** It will help you to identify the potential issues that can lead the system to fail or crash unexpectedly.
- **Stability:** It ensures that the system can withstand invalid and unexpected inputs.
- **Customer Satisfaction:** Customers are more likely to satisfy with their experience using your system if they see that your product can handle extreme cases. This can lead to increased customer loyalty and trustworthy relationship.

- **Reduced Cost:** Robustness testing uncovers potential bugs before releasing the software to real users. The cost of fixing bugs after production is high. This also comes with reduced customer satisfaction and trust.

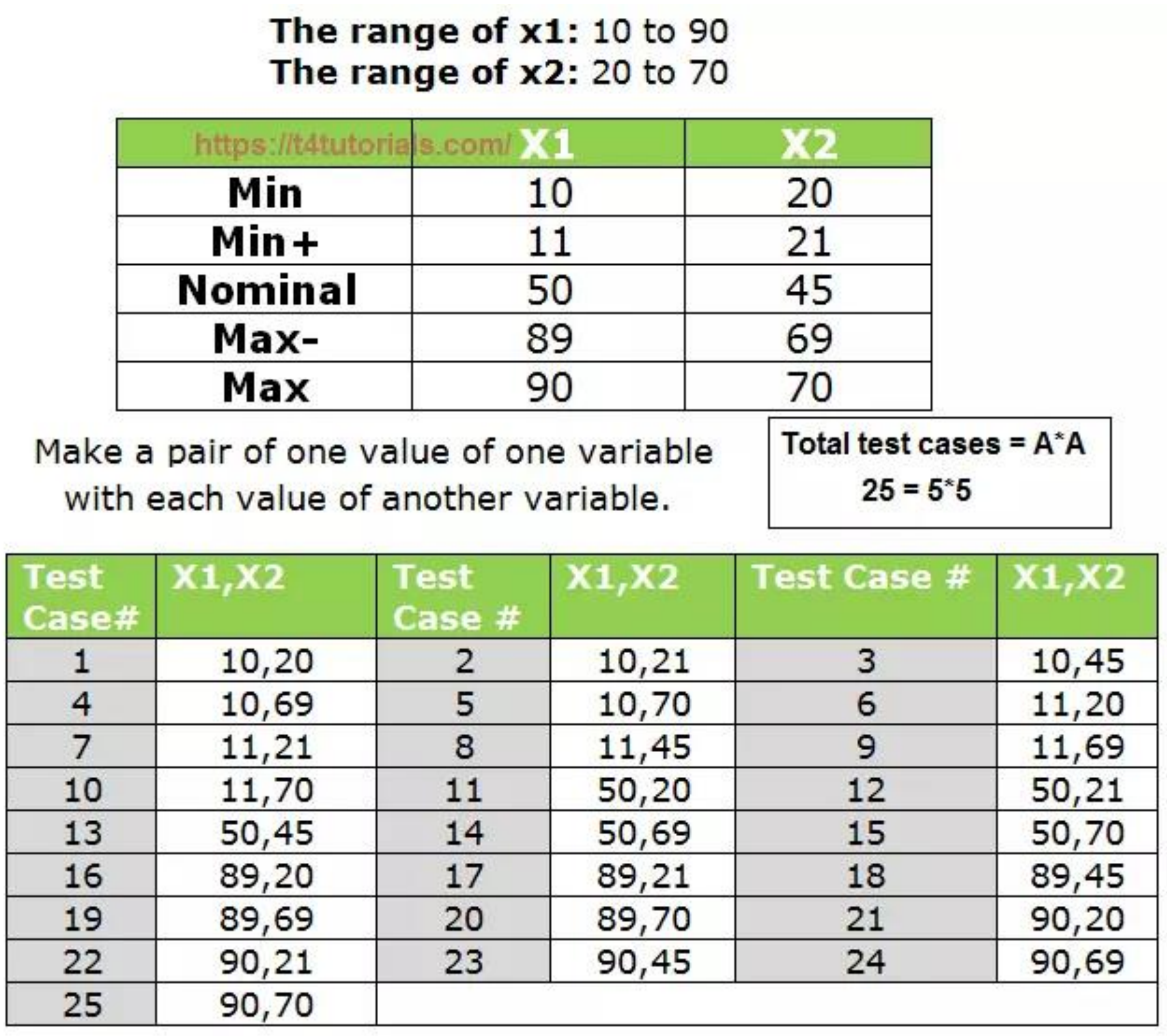
Worst Case Testing

- In Worst Case Boundary Value testing more than one variable has an extreme value. Furthermore, it follows a generalisation pattern and is more thorough than boundary value analysis

- Worst-Case boundary value analysis is a Black Box software testing technique.
- Here, single value fault of Boundary value analysis is rejected.
- Worst case testing is interested in “What happens when more than one variable has an extreme value”.
- Start with the five-elements set that contains min, min+, nom, max-, max. Then take the cartesian product of these sets to generate test cases.
- Boundary value analysis test cases are a proper subset of worst-case test cases.
- Worst-case testing is more thorough form of testing. Also requires more effort.
- Worst-case testing for a function on n variables generates **5n** test cases. [Min, Min+, Nominal, Max-, Max]

In Worst case boundary value testing, we make all combinations of each value of one variable with each value of another variable.

Worst Boundary value testing on 2 variables



The range of x1: 10 to 90
The range of x2: 20 to 70

	X1	X2
Min	10	20
Min+	11	21
Nominal	50	45
Max-	89	69
Max	90	70

Make a pair of one value of one variable with each value of another variable.

Total test cases = A*A
25 = 5*5

Test Case#	X1,X2	Test Case #	X1,X2	Test Case #	X1,X2
1	10,20	2	10,21	3	10,45
4	10,69	5	10,70	6	11,20
7	11,21	8	11,45	9	11,69
10	11,70	11	50,20	12	50,21
13	50,45	14	50,69	15	50,70
16	89,20	17	89,21	18	89,45
19	89,69	20	89,70	21	90,20
22	90,21	23	90,45	24	90,69
25	90,70				

Special Value Testing

- Special Value testing occurs when a tester uses domain knowledge, experience with similar programs and information about "soft spots" to device test cases.

- No guidelines are used other than to use "best engineering judgement".

- Special value testing is very dependent on the abilities of the tester.

Special Value testing has several reasons which makes it the best option for testing programs, like:

- The testing executed by Special Value Testing technique is based on past experiences, which validates that no bugs or defects are left undetected.
- Moreover, the testers are extremely knowledgeable about the industry and use this information while performing Special Value testing.
- Another benefit of opting Special Value Testing technique is that it is Ad-hoc in nature. There are no guidelines used by the testers other than their “best engineering judgment”.
- The most important aspect of this testing is that, it has had some very valuable inputs and success in finding bugs and errors while testing a software.

Random Testing

Random testing is a type of functional black box testing technique in which test inputs are generated randomly without following any specific test design or test case specification. It is also known as monkey testing.

Here, random inputs are used to test the program. The test results are then compared to the expected outcomes to determine if the test passes or fails.

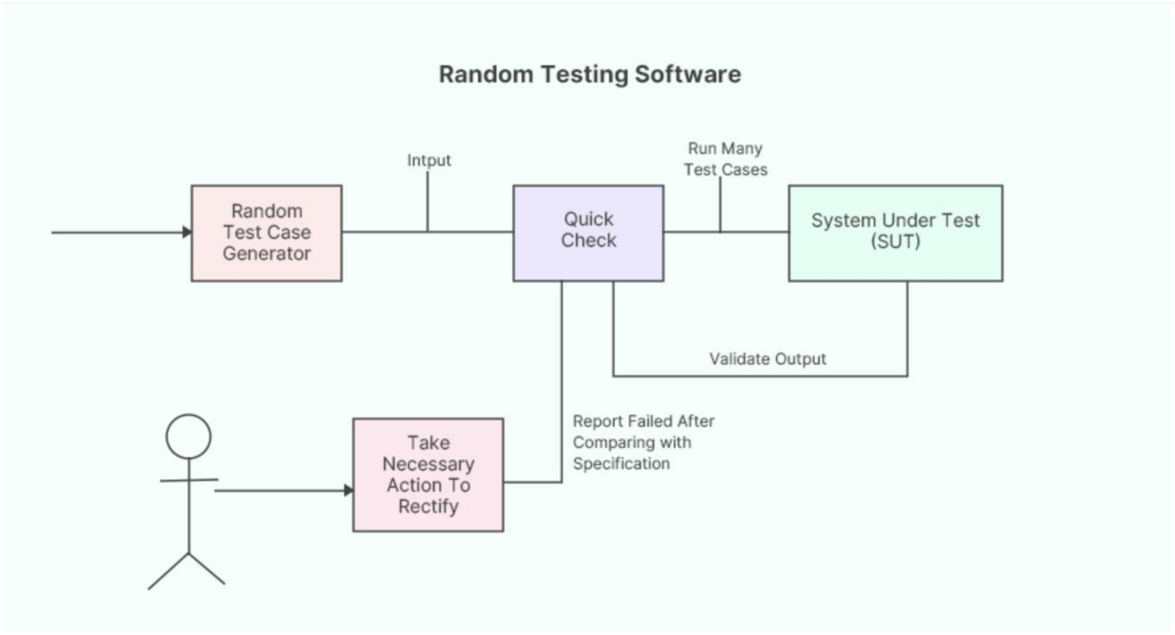
- ➢ Random testing is implemented when the bug in an application is not recognized.
- ➢ It is used to check the system’s execution and dependability.
- ➢ It saves our time and does not need any extra effort.
- ➢ Random testing is less costly, it doesn’t need extra knowledge for testing the program.

Tools used for Random Testing **QuickCheck, Randoop, Gram Test**

How Random Testing Works?

Step 1: Determine the input domain
Step 2: Select the inputs randomly/independently from the input domain
Step 3: Use these inputs to test the system and create a random test set
Step 4: Analyze and compare the test result with the software specification
Step 5: If the test report fails, then take the required action.

The image below depicts how this testing works,


Random Testing Software
Intput
Run Many Test Cases
Random Test Case Generator
Quick Check
System Under Test (SUT)
Validate Output
Report Failed After Comparing with Specification
Take Necessary Action To Rectify

Test Case for the Triangle problem ,NextDate function,Commission Problem

Test Case for the Triangle problem

- The triangle problem is a classic example of using boundary value analysis to test a software program.

- **Test Case Design for BVA:** While designing the test cases for BVA first we determine the number of input variables in the problem. For each input variable, we then determine the range of values it can take. Then we determine the extreme values and nominal value for each input variable.

- Before we generate the test cases, firstly we need to define the problem domain as described below.

- **Problem Domain:** "The triangle program accepts three integers, a, b and c as input. These are taken to be the sides of a triangle. The integers a, b and c must satisfy the following conditions

- The **output of the program** may be either of: Equilateral Triangle (all the sides are equal), Isosceles Triangle (one pair of sides is equal), Scalene (all the sides are different) or "Not a Traingle".

1. Verify that the figure is closed (polygon).
2. Verify that the figure is two-dimensional and formed with straight lines only.
3. Verify that the figure has exactly three sides.
4. Verify that the figure has exactly three vertices.
5. Verify that the figure has exactly three angles.
6. Verify that the sum of the angles of the figure is 180 degrees.
7. Verify that no two sides are parallel to each other.
8. Verify that the sum of the length of two sides of the triangle should be greater than the length of the third side.

9. Verify that no two angles of a triangle have 90 degrees and above value.

10. Verify the type of triangle is as per the specification, based on its sides – scalene, isosceles or equilateral.

11. Verify the type of triangle is as per the specification, based on its angles – acute angle, obtuse angle, or right-angled triangle.

12. Verify that the area of a triangle is equal to half of the product of its base and height.

Test Case ID	a	b	c	Expected Output
T1	1	50	50	Isosceles
T2	2	50	50	Isosceles
T3	99	50	50	Isosceles
T4	100	50	50	Not a Triangle
T5	50	50	50	Equilateral
T6	50	1	50	Isosceles
T7	50	2	50	Isosceles
T8	50	99	50	Isosceles
T9	50	100	50	Not a Triangle
T10	50	50	1	Isosceles
T11	50	50	2	Isosceles
T12	50	50	99	Isosceles
T13	50	50	100	Not a Triangle

For a, b, c to form a triangle the following conditions should be satisfied –

a < b+c

b < a+c

c < a+b

If any of these conditions is violated output is Not a Triangle.

Test Case for the NextDate function

To Generate BVA Test Cases-For the Next Date Function

Problem Domain: "Next Date" is a function consisting of three variables like: month, date and year. It returns the date of next day as output. It reads current date as input date.

The conditions are

C1: 1<Day<31
C2: 1<Month<12

C3: 1800 <Year <2048

If any one condition out of C1, C2 or C3 fails, then this function produces an output "value of month not in the range 1...12".
Since many combinations of dates can exist, hence we can simply display one message for this function: "Invalid Input Date".

Boundary Value Analysis:
No. of test Cases (n = no. of variables) = **4n+1** = 4*3 +1 =13

Equivalence Class Testing: Input classes:
Day:

D1: day between 1 to 28

D2: 29

D3: 30

D4: 31

Month:

M1: Month has 30 days

M2: Month has 31 days

M3: Month is February

Year:

Y1: Year is a leap year

Y2: Year is a normal year

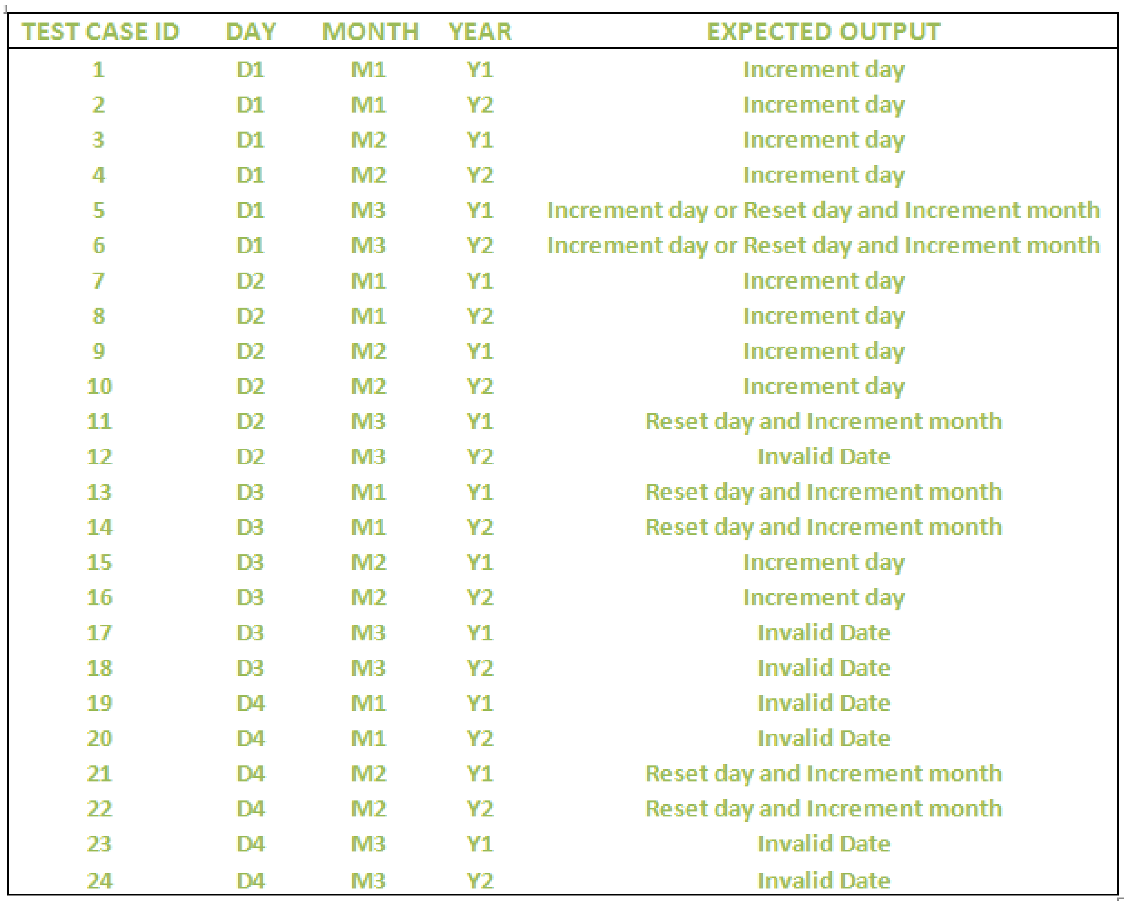
Output Classes:

Increment Day

Reset Day and Increment Month

Increment Year

Invalid Date



TEST CASE ID	DAY	MONTH	YEAR	EXPECTED OUTPUT
1	D1	M1	Y1	Increment day
2	D1	M1	Y2	Increment day
3	D1	M2	Y1	Increment day
4	D1	M2	Y2	Increment day
5	D1	M3	Y1	Increment day or Reset day and Increment month
6	D1	M3	Y2	Increment day or Reset day and Increment month
7	D2	M1	Y1	Increment day
8	D2	M1	Y2	Increment day
9	D2	M2	Y1	Increment day
10	D2	M2	Y2	Increment day
11	D2	M3	Y1	Reset day and Increment month
12	D2	M3	Y2	Invalid Date
13	D3	M1	Y1	Reset day and Increment month
14	D3	M1	Y2	Reset day and Increment month
15	D3	M2	Y1	Increment day
16	D3	M2	Y2	Increment day
17	D3	M3	Y1	Invalid Date
18	D3	M3	Y2	Invalid Date
19	D4	M1	Y1	Invalid Date
20	D4	M1	Y2	Invalid Date
21	D4	M2	Y1	Reset day and Increment month
22	D4	M2	Y2	Reset day and Increment month
23	D4	M3	Y1	Invalid Date
24	D4	M3	Y2	Invalid Date

Test Cases:

Test Case ID	Day	Month	Year	Expected Output
E1	15	4	2004	16-4-2004
E2	15	4	2003	16-4-2003
E3	15	1	2004	16-1-2004
E4	15	1	2003	16-1-2003
E5	15	2	2004	16-2-2004
E6	15	2	2003	16-2-2003
E7	29	4	2004	30-4-2004
E8	29	4	2003	30-4-2003
E9	29	1	2004	30-1-2004
E10	29	1	2003	30-1-2003
E11	29	2	2004	1-3-2004
E12	29	2	2003	Invalid Date
E13	30	4	2004	1-5-2004
E14	30	4	2003	1-5-2003
E15	30	1	2004	31-1-2004
E16	30	1	2003	31-1-2003
E17	30	2	2004	Invalid Date
E18	30	2	2003	Invalid Date

E19	31	4	2004	Invalid Date
E20	31	4	2003	Invalid Date
E21	31	1	2004	1-2-2004
E22	31	1	2003	1-5-2003
E23	31	2	2004	Invalid Date
E24	31	2	2003	Invalid Date

TEST CASES FOR COMMISSION PROBLEM IN BVA TESTING

To Generate Equivalence Class Test Cases-For the Salesman Commission Calculation Program

"A desert cooler sales person sold cooler fans, pumps and bodies that were made by a cooler maker. Fans cost $45, pumps cost $30 and bodies cost $25.
The salesperson had to sell at least one complete cooler per month, and the production limits were such that the most the sales person could sell in a month was 70 fans, 80 pumps and 90 bodies. The sales person used to send the details of sold items to the cooler maker. The cooler maker then computed the sales person's commission as follows:
1) 10% on sales upto and including $1000.
2) 15% of the next $800.
3) And 20% on any sales in excess of $1800.

The commission program produced a monthly sales report that gave the total number of fans, pumps and bodies sold, the sales person's total dollar sales and finally, the commission."

Here we have three inputs for the program, hence n = 3.
Since BVA yields (4n + 1) test cases according to single fault assumption theory, hence we can say that the total number of test cases will be (4*3+1)=12+1=13.

The boundary value test cases are

We can see that the monthly sales are limited by the following consideration:

$1 \leq$ fans ≤ 70

$1 \leq$ pumps ≤ 80

$1 \leq$ bodies ≤ 90

Test Case ID	Fans	Pumps	Bodies	Expected Output of Sales
1	35	40	1	2800
2	35	40	2	2825
3	35	40	45	3900
4	35	40	89	5000
5	35	40	90	5025
6	35	1	45	2730
7	35	2	45	2760
8	35	40	45	**3900**
9	35	79	45	5070
10	35	80	45	5100
11	1	40	45	2370
12	2	40	45	2415
13	35	40	45	3900
14	69	40	45	5430
15	70	40	45	5475

Here we can see that out of 15 test cases, two are redundent. Hence 13 test cases are sufficient to test this program.

Guidelines for Boundary Value Testing (check)

Guidelines for Boundary Value analysis

- If an input condition is restricted between values x and y, then the test cases should be designed with values x and y as well as values which are above and below x and y.
- If an input condition is a large number of values, the test case should be developed which need to exercise the minimum and maximum numbers. Here, values above and below the minimum and maximum values are also tested.
- Apply guidelines 1 and 2 to output conditions. It gives an output which reflects the minimum and the maximum values expected. It also tests the below or above values.

Example:

Input condition is valid between 1 to 10

Boundary values 0,1,2 and 9,10,11

- Guidelines for BVA are similar in many respects to those provided for equivalence partitioning:
  - If an input condition specifies a range bounded by values a and b, test cases should be designed with values a and b and just above and just below a and b.
  - If an input condition specifies a number of values, test cases should be developed that exercise the minimum and maximum numbers. Values just above and below minimum and maximum are also tested.
  - Apply guidelines 1 and 2 to output conditions. For example, assume that a temperature vs. pressure table is required as output from an engineering analysis program. Test cases should be designed to create an output report that produces the maximum (and minimum) allowable number of table entries.
  - If internal program data structures have prescribed boundaries (e.g., an array has a defined limit of 100 entries), be certain to design a test case to exercise the data structure at its boundary