**Module-2**

**Functional Testing**

**Overview:**

* This module gives knowledge about the different ways of deriving test cases with specifications.
* At the end of this module, one should be able to derive test cases following a strategy for any type of specifications.

**Recall of Functional Testing:**

* Otherwise referred to as Blackbox testing tries to test the functionality of an application.
* The input generally considered for generating test cases in black box testing is specification.
* There are 3 test generation strategies for deriving test cases using functionality testing.
* They are as follows:

1. Boundary value analysis
2. Equivalence Partitioning
3. Decision Tables.

**Why Test generation strategies?**

class Roots {

// Solve ax2 + bx + c = 0

public roots(double a, double b, double c)

{ … }

// Result: values for x

double root\_one, root\_two;

}

Which values for a, b, c should we test? assuming a, b, c, were 32-bit integers, we’d have 1028 legal inputs with 1.000.000.000.000 tests/s, we would still require 2.5 billion years.

For the given problem above, it is nearly impossible for anyone to identify the total number of test cases necessary to test this application. To avoid selection of random testcases and to gain confidentiality if we follow a strategy and test an application, we can avoid 99% of failures in terms of testing functionality.

These strategies help testers to avoid exhaustive testing and check for optimality while deriving test cases.

**Test Generation Strategies:**

If one needs to understand the generation strategies, the first step is understanding the requirement. Let us consider the “Triangle Problem” for understanding all the strategies.

**Triangle Problem:**

**Input:**

The three sides of a triangle as integers.

**Output:**

The type of the triangle Equilateral, Isosceles, Scalene and Not a Triangle.

**Procedure:**

This problem is solved using the triangle property **“The sum of any pair of two sides**

**must be greater than the third side” (i.e) .**

* 1. If any of the above conditions fails, then the triangle is “Not a Triangle”
  2. If all three sides are equal, then the triangle is “Equilateral Triangle”
  3. Exactly one pair of sides is equal, then it forms “Isosceles Triangle”
  4. If no pair of sides is equal, then it forms “Scalene Triangle”.

**Pseudocode: (Structured Implementation):**

Dim a, b, c As Integer

Dim c1, c2, c3, IsATriangle As Boolean

**‘Step 1: Get Input**

Do

Output (“Enter 3 integers which are sides of a triangle”)

Input (a, b, c)

c1 = (1 ≤ a) AND (a ≤ 200)

c2 = (1 ≤ b) AND (b ≤ 200)

c3 = (1 ≤ c) AND (c ≤ 200)

If NOT(c1)

Then Output (“Value of a is not in the range of permitted values”) EndIf

If NOT(c2)

Then Output (“Value of b is not in the range of permitted values”) EndIf

If NOT(c3)

Then Output (“Value of c is not in the range of permitted values”) EndIf

Until c1 AND c2 AND c3

Output (“Side A is”,a)

Output (“Side B is”,b)

Output (“Side C is”,c)

**‘Step 2: Is A Triangle?**

If (a < b + c) AND (b < a + c) AND (c < a + b)

Then IsATriangle = True

Else IsATriangle = False

EndIf

**‘Step 3: Determine Triangle Type**

If IsATriangle

Then If (a = b) AND (b = c)

Then Output (“Equilateral”)

Else If (a ≠b) AND (a ≠c) AND (b ≠c)

Then Output (“Scalene”)

Else Output (“Isosceles”)

EndIf

EndIf

Else Output (“Not a Triangle”)

EndIf

With the triangle problem, let us try to understand all the test generation strategies.

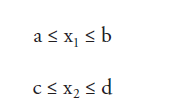
**Boundary Value Analysis:(BVA)**

* While generating test cases for functional testing, the underlying idea to generate test cases is by understanding the functional nature of the program.
* Input domain testing/ Boundary value analysis focuses on the input domain.
* Two considerations apply to input value testing
  + **Consideration of invalid values for a variable provides 4 variants of Boundary value Analysis**

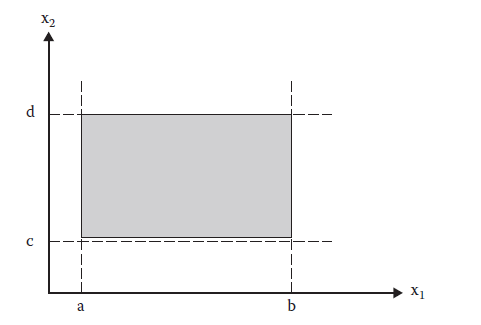
1. **Normal Boundary Value Analysis**
2. **Robust Boundary Value Analysis**
3. **Worst Boundary Value Analysis**
4. **Robust worst boundary value analysis**
   * **Single Fault Assumption: failure is due to invalid values of a single variable.**

**Applicability:**

This boundary value strategy can be used for generating testcases if and only if the variables in the function take up input values in range.

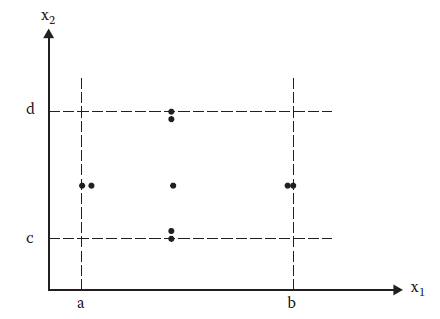
****

Suppose a function takes up two input variable x1 and x2 in the interval [a,b] and [c,d] the input space for the considered function is depicted below:



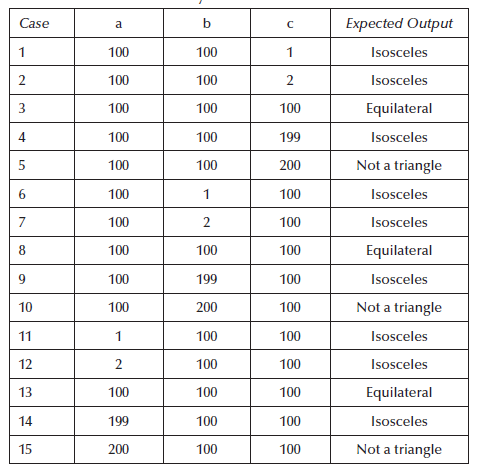
**Boundary Value Analysis logic:**

* All four forms of boundary value testing focus on the boundary of the input space to identify test cases.
* The rationale behind boundary value testing is that errors tend to occur near the extreme values of an input variable.
* Loop conditions, for example, may test for < when they should test for ≤, and counters often are “off by one.”
* The basic idea of boundary value analysis is to use input variable values at their **minimum, just above the minimum, a nominal value, just below their maximum, and at their maximum.**
* So, for a function of n variables, boundary value analysis yields 4n + 1 unique test cases.
* The boundary for the variables is defined explicitly however if there is absence of explicit bounds, artificial bounds must be created as we are going to do for the triangle problem.
* For a function with two variables as input, the test cases are depicted below:

****

**Normal Boundary Value Test cases for the triangle problem:**

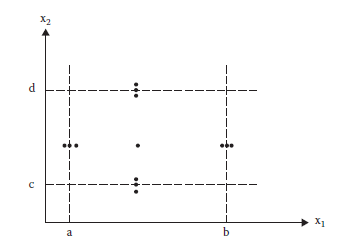
* In the problem statement, no conditions are specified on the triangle sides, other than being integers.
* Obviously, the lower bounds of the ranges are all 1. We arbitrarily take 200 as an upper bound.
* Triangle problem possess 3 variables as input so **4n + 1= 4\*3+1= 13 test cases.**
* Applying BVA for each side, the test values are **{1, 2, 100, 199, 200}.**



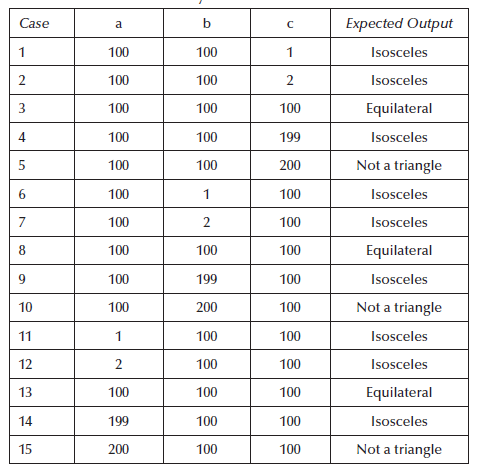
* Arrow mark denotes three test cases that are redundant, so only one must be considered that will yield a total of 13 test cases.

**Robust Boundary Value Test Cases:**

* Robust boundary value testing is a simple extension of normal boundary value testing: in addition to the five boundary value analysis values of a variable, we see what happens when the extrema are exceeded with a value slightly greater than the maximum (max+) and a value slightly less than the minimum (min–).
* Robust boundary value test cases for our continuing example are depicted below:



* The exception handling choice mandates robustness testing as robustness testing tries to identify the variations of output.
* Suppose if the considered function is printing date and if the function is not tested for outlying maximum value, then it might even print May 32 and so on. This mandates robustness testing.
* So, robustness BVA produces **6n + 1= 6\*3+1= 19 test cases for the triangle problem.**

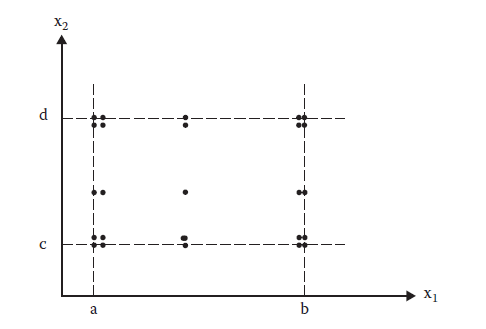


|  |  |  |  |
| --- | --- | --- | --- |
| 16 | 100 | 100 | 201 |
| 17 | 100 | 100 | 0 |
| 18 | 0 | 100 | 100 |
| 19 | 201 | 100 | 100 |
| 20 | 100 | 0 | 100 |
| 21 | 100 | 201 | 100 |

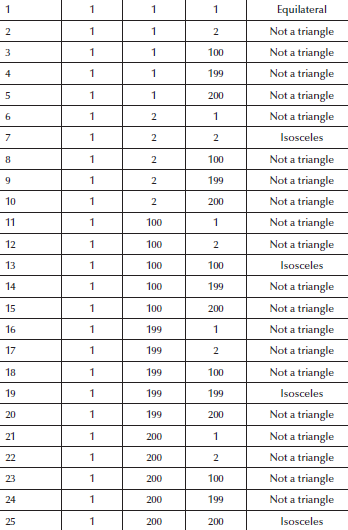
* Elimination of redundancies yields us 19 test cases.

**Worst Case Boundary Value Testing:**

* Worst case testing become unavoidable if the applications are of life critical types.
* One form of variation w.r.to other forms of boundary analysis is the rejection of single fault assumption.
* Rejecting single-fault assumption means that we are interested in what happens when more than one variable has an extreme value.
* worst-case testing for a function of n variables generates test cases.

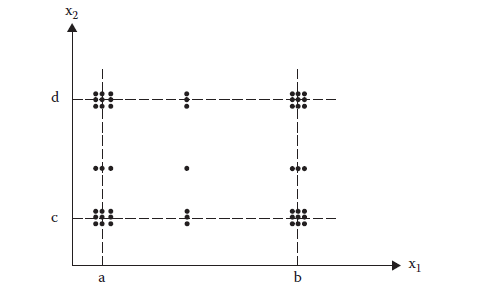


* For the triangle problem, the number of test cases are 125. It is not mandatory to write all the 125 test cases; however the concept is essential and must show failure of single fault assumption with the sample test cases.



**Robust-Worst Case Testing:**

* In robustness testing, we add two more states i.e. just below minimum value (minimum value–)and just above maximum value (maximum value+).
* We also give invalid inputs and observe the behaviour of the program.
* A program should be able to handle invalid input values, otherwise it may fail and give unexpected output values. There are seven states (minimum -, minimum, minimum +, nominal, maximum –, maximum, maximum +) and a total of testcases will be generated.
* This will be the largest set of test cases and requires the maximum effort to generate such test cases.
* For two variables, the robust test cases are depicted below:



* For the triangle problem it is 7^3 yielding 343 test cases. It is not possible to tabulate those test cases, so as we did for robustness testing, make sure you give a sample test cases.

**Applicability of BVA:**

* Boundary value analysis is a simple technique and may prove to be effective when used

correctly. Here, input values should be independent which restricts its applicability in many programs.

* This technique does not make sense for Boolean variables where input values are TRUE and FALSE only, and no choice is available for nominal values, just above boundary values, just below boundary values, etc. This technique can significantly reduce the number of test cases and is suited to programs in which input values are within ranges or within sets.
* This is equally applicable at the unit, integration, system and acceptance test levels. All we want is input values where boundaries can be identified from the requirements.

**Equivalence Partitioning:**

**Why Equivalence Partitioning?**

* One of the major drawbacks noted in boundary value analysis is the redundancy.
* We would like to have a sense of complete testing with no redundancy.

**What is Equivalence Partitioning?**

* Equivalence class testing echoes the two deciding factors of boundary value testing, robustness and the single/multiple fault assumption.
* Equivalence classes is that they form a partition of a set, where partition refers to a collection of mutually disjoint subsets, the union of which is the entire set.
* This has two important implications for testing—the fact that the entire set is represented provides a form of completeness, and the disjointedness ensures a form of non-redundancy.
* Because the subsets are determined by an equivalence relation, the elements of a subset

have something in common. The idea of equivalence class testing is to identify test cases by using one element from each equivalence class. Eg: If we have tested the triangle problem for equilateral triangle it is enough to test with a single set as (5,5,5) and that need not get repeated for other values as (6,6,6). This method allows one to select few test cases and still achieve reasonable coverage.

* The four forms of equivalence class testing all address the problems of gaps and redundancies that are common to the four forms of boundary value testing.

**Basic Idea:**

* Again, this methodology can also be used only if the variables in a function takes up values in range.
* Eg: Consider an application that takes x as input and produces the square of x as output 1≤x≤100
* If we must partition the above example using equivalence partitioning this has only two partitions as valid and invalid partition

Invalid

0,-1,-2…etc

Invalid

101,102,…..etc

Valid

1,2….100

So if we have to derive equivalence class test cases for the above example then it is as follows:

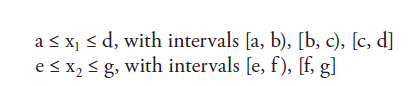
|  |  |  |
| --- | --- | --- |
| **Test Case** | **Input** | **Expected Output** |
| 1 | 0 | Invalid |
| 2 | 50 | Valid |
| 3 | 101 | Invalid |

* For every partition, the application must be tested with random values.
* There are various forms of equivalence partitioning and they are as follows:
  1. Weak Normal Equivalence partitioning
  2. Strong Normal Equivalence partitioning
  3. Weak Robust Equivalence partitioning
  4. Strong Robust Equivalence partitioning

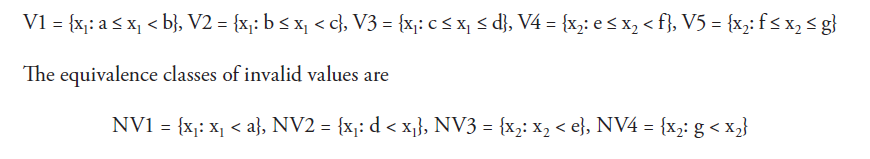
Let us try to understand each of these classes considering the triangle problem

**Weak Normal Equivalence Partitioning:**

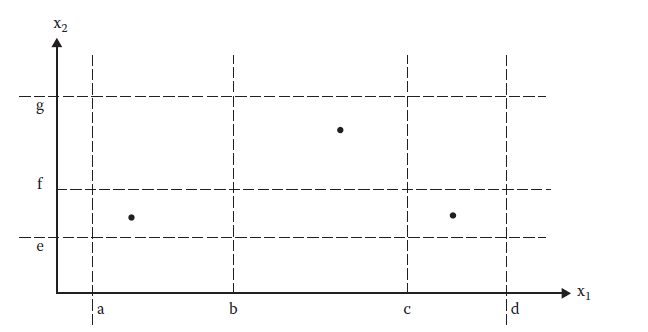
* This partition will consider only the valid values of a variable resulting test cases only with valid values.
* Consider two variables x1 and x2 whose intervals are shown below:



* The valid Equivalence class for the considered example are

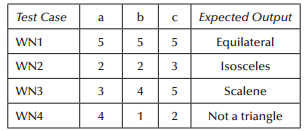
****

* For this example if we plot the weak normal equivalence partitioning it is as follows:

****

* The plot clearly indicates for the variable x1 three valid intervals are there and one test case in every interval.

**Weak Normal for Triangle Problem:**

****

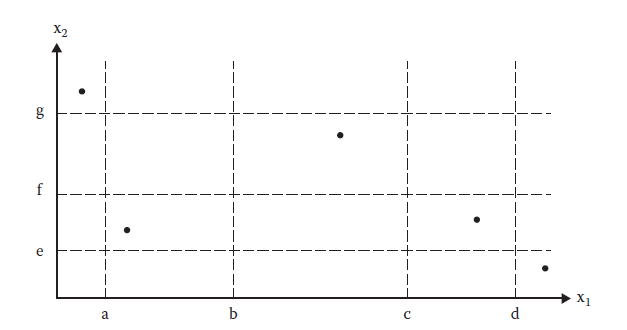
* Clearly note that the weak normal possess only test cases for checking valid values and it follows single fault assumption.
* For the triangle problem we need to test the program only for these four outputs and we can obtain valid outputs if we give only valid inputs, so after partitioning one valid value for each output is considered.

**Weak Robust Equivalence Class:**

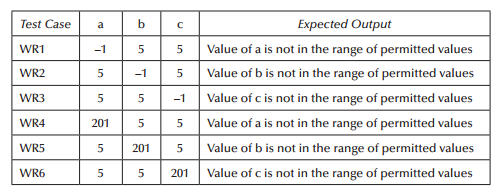
* The robust part comes from consideration of invalid values, and the weak part refers to the single fault assumption.
* The process of weak robust equivalence class testing is a simple extension of that for weak normal equivalence class testing—pick test cases such that each equivalence class is represented.
* The two additional test cases cover all four classes of invalid values. The process is like that for boundary value testing:

1. For valid inputs, use one value from each valid class (as in what we have called weak normal equivalence class testing). (Note that each input in these test cases will be valid.)

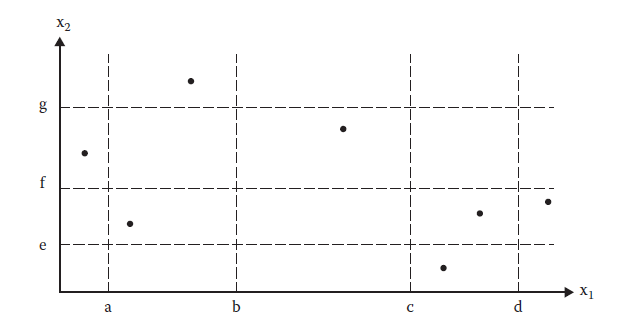
2. For invalid inputs, a test case will have one invalid value and the remaining values will all be valid. (Thus, a “single failure” should cause the test case to fail.)

****

**Weak Robust for Triangle Problem:**

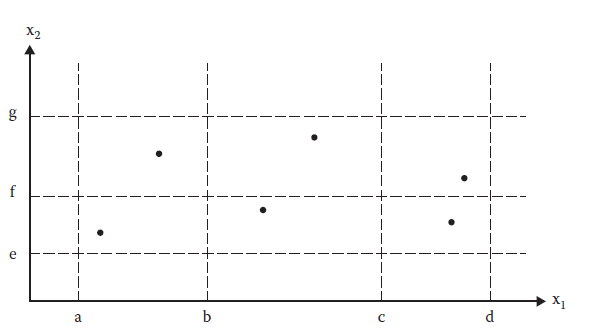
****

* For every variable assuming the valid range to be 1 to 200 yields us invalid partition of {0, -1, -2….} and {201,202,}.
* Following single fault assumption, at a time only one variable can have invalid value.
* With these assumptions the table above shows the test cases for weak robust equivalence classes and this will include the test cases already derived using normal equivalence classes.



**Strong Normal Equivalence class:**

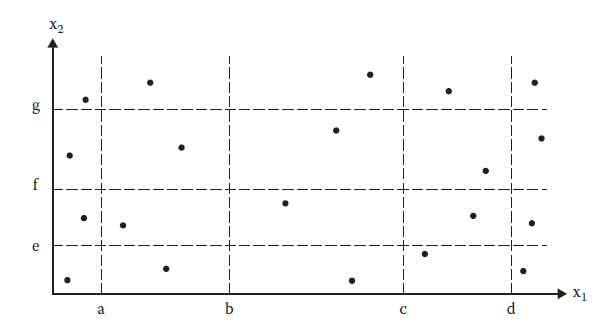
* Normal denotes only valid values and strong denotes multiple fault assumption yielding test cases as depicted below:

****

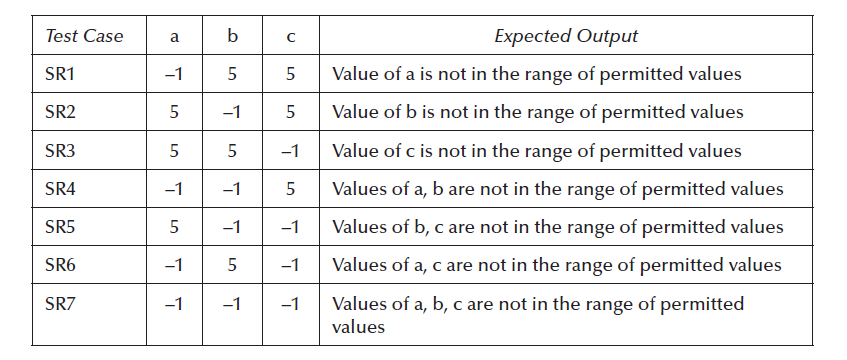
* Since no valid subintervals of variables, a, b and c exist, so the strong normal equivalence class test cases are identical to the weak normal equivalence class test cases.

**Strong Robust Equivalence Classes:**

* The robust part comes from consideration of invalid values, and the strong part refers to the multiple fault assumption.
* We obtain test cases from each element of the Cartesian product of all the equivalence classes, both valid and invalid.



* The strong robust test cases for the triangle problem are as follows:



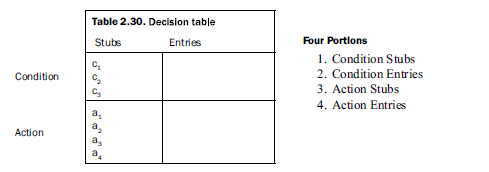
* For the above consideration, we haven’t tested the triangle problem with triangle property, you can derive test cases considering that as well.

**Applicability:**

* It is applicable at unit, integration, system and acceptance test levels.
* The basic requirement is that inputs or outputs must be partitioned based on the requirements and every partition will give a test case.
* The selected test case may test the same thing, as would have been tested by another test case of the same equivalence class, and if one test case catches a bug, the other probably will too.
* If one test case does not find a bug, the other test cases of the same equivalence class may also not find any bug.
* We do not consider dependencies among different variables while designing equivalence classes.
* The design of equivalence classes is subjective, and two testing persons may design two different sets of partitions of input and output domains.
* This is understandable and correct if the partitions are reviewed, and all agree that they acceptably cover the program under test.

**Decision Table Testing:**

* So far, the testing strategies we have seen covered the methodologies to derive test cases if the input for the application/program are of values in ranges.
* Another important functional testing methodology to derive test cases whenever there is logic in involved is decision table testing.
* The most rigorous because of their strong logical basis.
* Decision tables are used in many engineering disciplines to represent complex logical relationships. An output may be dependent on many input conditions and decision tables give a pictorial view of various combinations of input conditions.
* There are four portions of the decision table and are shown in Table below:



**Parts of a Decision Table:**

The four parts of the decision table are given as:

**Condition Stubs:** All the conditions are represented in this upper left section of the decision table. These conditions are used to determine a particular action or set of actions.

**Action Stubs:** All possible actions are listed in this lower left portion of the decision table.

**Condition Entries:** In the condition entries portion of the decision table, we have a number

of columns and each column represents a rule. Values entered in this upper right portion of the table are known as inputs.

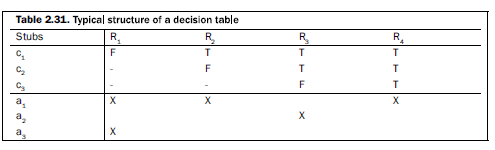
**Action Entries:** Each entry in the action entries portion has some associated action or set of

actions in this lower right portion of the table. These values are known as outputs and are

dependent upon the functionality of the program.

**Limited Entry and Extended Entry Decision Tables:**

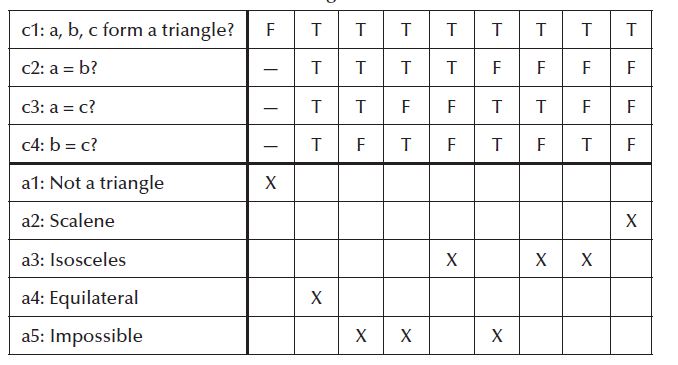
* Decision table testing technique is used to design a complete set of test cases without using the
* internal structure of the program.
* Every column is associated with a rule and generates a test case.
* A typical decision table is given in Table 2.31.



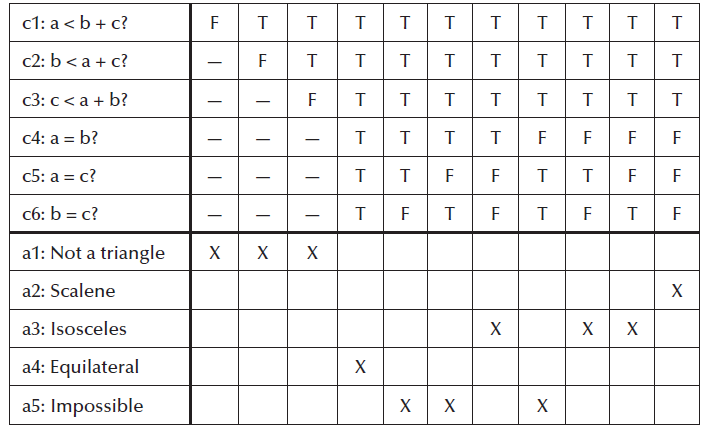
* In Table 2.31, input values are only True (T) or False (F), which are binary conditions.
* The decision tables which use only binary conditions are known as limited entry decision tables.
* The decision tables which use multiple conditions where a condition may have many possibilities instead of only ‘true’ and ‘false’ are known as extended entry decision tables.

**Decision Table For Triangle Problem:**

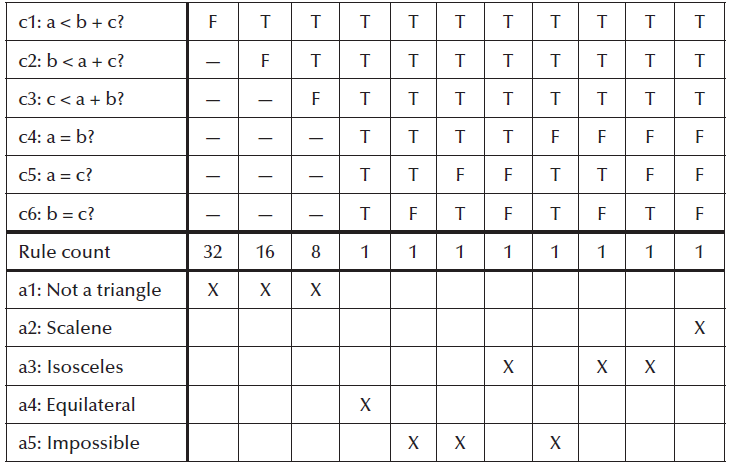
* In a decision table every condition is input, and every rule is test case.
* Because the decision table can mechanically be forced to be complete, we have some assurance that we will have a comprehensive set of test cases. Several techniques that produce decision tables are more useful to testers.



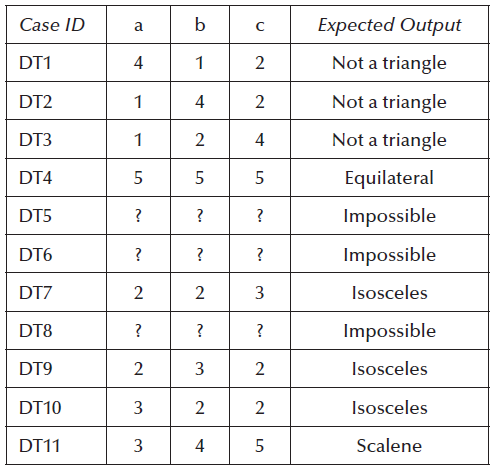
* We see examples of don’t care entries and impossible rule usage. If the integers a, b, and c do not constitute a triangle, we do not even care about possible equalities, as indicated in the first rule.
* In rules 3, 4, and 6, if two pairs of integers are equal, by transitivity, the third pair must be equal; thus, the negative entry makes these rules impossible.
* The choice of conditions can greatly expand the size of a decision table. Here, we have expanded the old condition (c1: a, b, c form a triangle?) to a more detailed view of the three inequalities of the triangle property. If any one of these fails, the three integers do not constitute sides of a triangle. We could expand this still further because there are two ways an inequality could fail: one side could equal the sum of the other two, or it could be strictly greater.



* Use of don’t care entries has a subtle effect on the way in which complete decision tables are recognized. For a limited entry decision table with n conditions, there must be 2 *n* independent rules.
* When don’t care entries really indicate that the condition is irrelevant, we can develop a rule count as follows: rules in which no don’t care entries occur count as one rule, and each don’t care entry in a rule doubles the count of that rule.



* The calculation of rule count holds very important as the number of test cases for a decision table testing relies on the number of rule counts.
* With these observations, the test cases for the triangle problem are as follows:



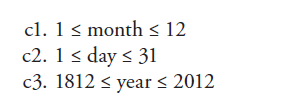
After understanding all these concepts clearly with triangle problem, keep all your notes closed and try to derive test cases for the below mentioned questions:

1. Next Date Problem
2. Commission Problem
3. Finding the average of 3 marks and printing the class each student has secured etc.

**Next Date Problem:**

**Input:**

Month, date and year and it is defined as follows:

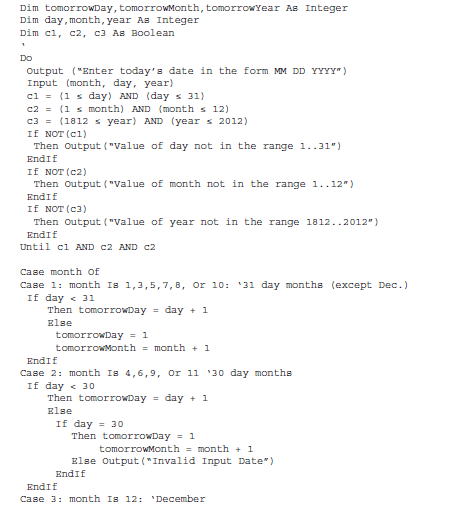


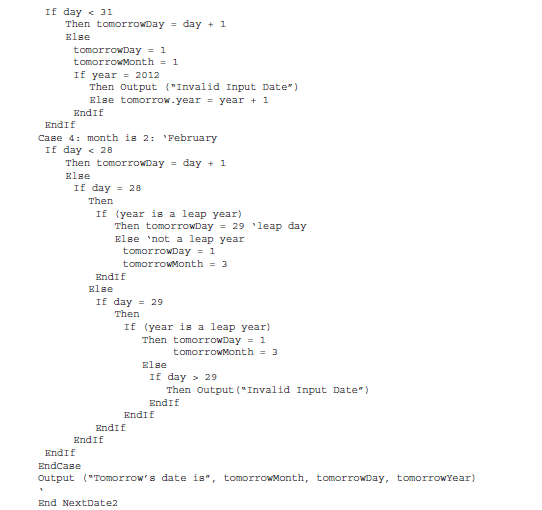
**Output:**

Prints the next date of the given date (input date)

**Procedure /Pseudocode:**

* Pseudocode verifies first for the input values are of range.
* Checks for 31 days,30days, December month and February month.





**Boundary Value Analysis:**

* Three variables month, date and year so

1. **Normal Boundary Value Analysis (4\*3)+1=13 test cases:**

{min,min+,nom,max-,max} {1,2,16,30,31} {1,2,6,11,12}{1812,1813,1912,2011,2012}

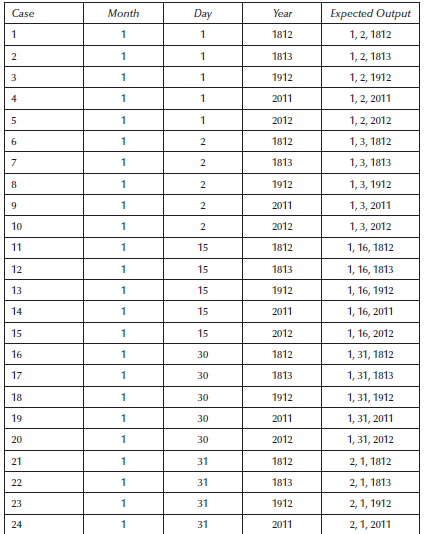
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test case** | **Month** | **Date** | **Year** | **Expected Output** |
| 1 | 6 | 16 | 1812 | 6/17/1812 |
| 2 | 6 | 16 | 1813 | 6/17/1813 |
| 3 | 6 | 16 | 1912 | 6/17/1912 |
| 4 | 6 | 16 | 2011 | 6/17/2011 |
| 5 | 6 | 16 | 2012 | 6/17/2012 |
| 6 | 1 | 16 | 1912 | 1/17/1912 |
| 7 | 2 | 16 | 1912 | 2/17/1912 |
| 8 | 6 | 16 | 1912 |  |
| 9 | 11 | 16 | 1912 |  |
| 10 | 12 | 16 | 1912 |  |
| 11 | 6 | 1 | 1912 |  |
| 12 | 6 | 2 | 1912 |  |
| 13 | 6 | 16 | 1912 |  |
| 14 | 6 | 30 | 1912 |  |
| 15 | 6 | 31 | 1912 |  |

1. **Robust Boundary Value Test cases: (6\*3+1=19 test cases)**

Include the value next to maximum one before minimum

1. **Worst Case Boundary Value Analysis: (5 ^n =125 test cases)**

**Only a snapshot is given**

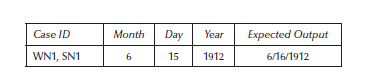
****

1. **Robust worst BVA: (7^3=343 test cases)**

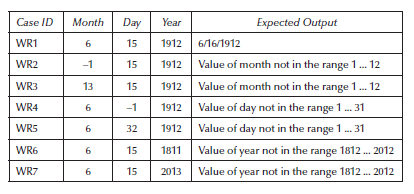
Same snapshot of worst case can be considered.

**Equivalence Partitioning for Next Date Problem:**

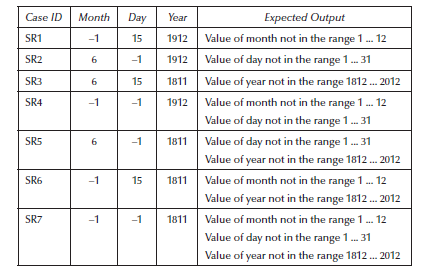
1. **Strong Normal Equivalence Class:**

****

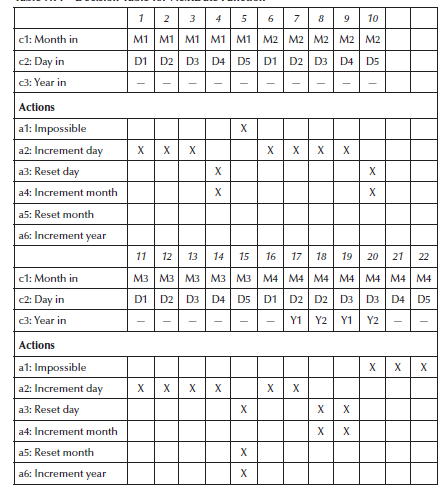
1. **Weak Robust Equivalence Class:**

****

1. **Strong Robust Test cases:**

****

**Decision Table Testing:**

****

**Commission Problem:**

**Input:**

Number of Locks, stocks and Barrels and cost of it as $45, $30 and $25.

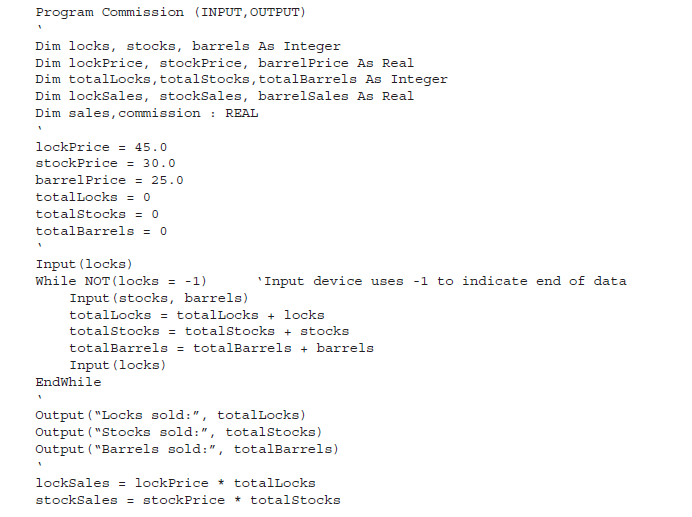
**Output:**

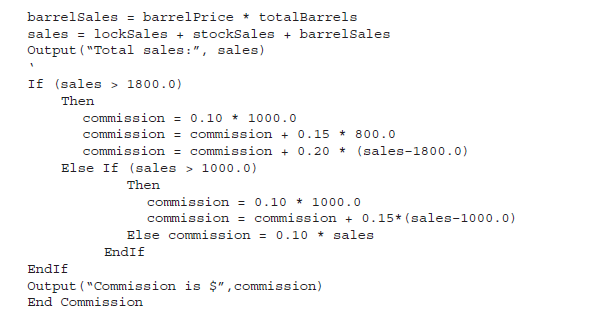
Salesperson Commission value depending on the sales made.

**Procedure:**

Minimum expectation of sales is atleast one loack,one stock and one barrel and the maximum expectation is 70,80 and 90. After every town visit sales person sends the sales data and at the end of the month he sends a message as -1 denoting the end of the sale of that month. Once that is received, company calculates commission as 10% on sales up to (and including) $1000, 15% on the next $800, and 20% on any sales in excess of $1800.

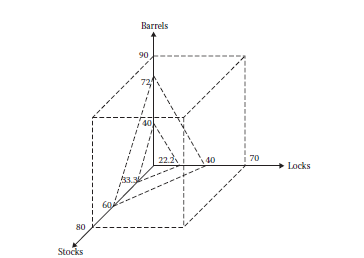
**Pseudocode:**

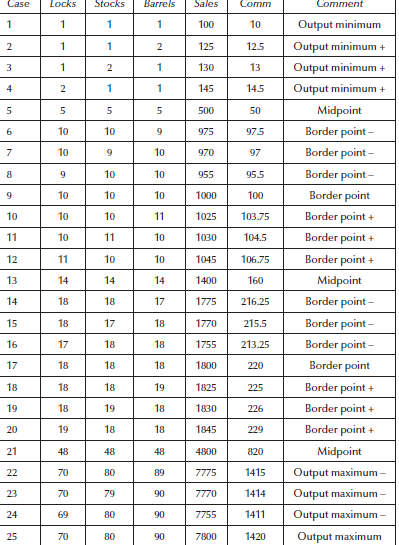




**Boundary Value Analysis:**

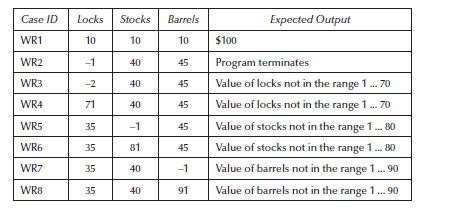
This time, we will look at boundary values derived from the output range, especially near the threshold points of $1000 and $1800 where the commission percentage changes. The output space of the commission is shown in Figure 5.6. The intercepts of these threshold planes with the axes are shown.



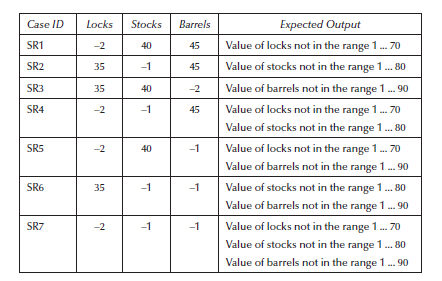


**Equivalence partitioning for the Commission Problem:**

1. **Weak Robust Test cases:**



ii**. Strong Robust Equivalence Classes:**



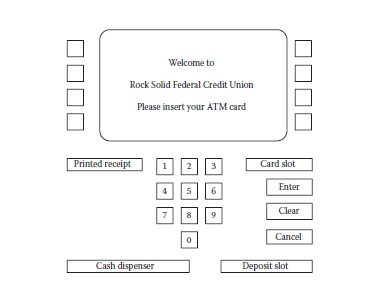
**Decision Table:**

Its not applicable as it doesn’t involve any logical output and the output is varying.

**Note:**

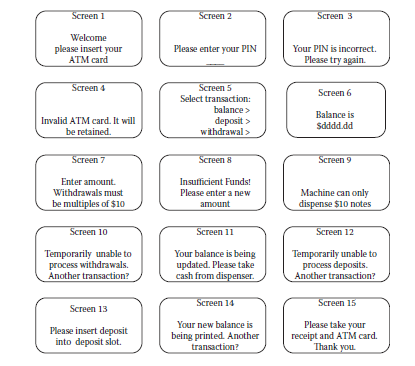
**Clearly understand the concepts using the triangle problem and you should be able to derive test cases using any of the strategy for any problem. So, understand and remember the logic clearly. Also check all the exercises we have done in classroom as well as in ppt.**

**SATM Problem:**



**Problem Statement:**

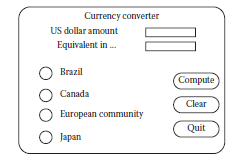
* The SATM system communicates with bank customers via the 15 screens



* SATM customers can select any of three transaction types: deposits, withdrawals, and balance inquiries. For simplicity, these transactions can only be done on a checking account.
* When a bank customer arrives at an SATM station, screen 1 is displayed.
* The bank customer accesses the SATM system with a plastic card encoded with a personal account number (PAN),which is a key to an internal customer account file, containing, among other things, the customer’s name and account information.
* If the customer’s PAN matches the information in the customer account file, the system presents screen 2 to the customer. If the customer’s PAN is not found, screen 4 is displayed, and the card is kept.
* At screen 2, the customer is prompted to enter his or her personal identification number (PIN).
* If the PIN is correct (i.e., matches the information in the customer account file), the system displays screen 5; otherwise, screen 3 is displayed.
* The customer has three chances to get the PIN correct; after three failures, screen 4 is displayed, and the card is kept.
* On entry to screen 5, the customer selects the desired transaction from the options shown on screen.
* If balance is requested, screen 14 is then displayed. If a deposit is requested, the status of the deposit envelope slot is determined from a field in the terminal control file. If no problem is known, the system displays screen 7 to get the transaction amount.
* If a problem occurs with the deposit envelope slot, the system displays screen 12. Once the deposit amount has been entered, the system displays screen 13, accepts the deposit envelope, and processes the deposit. The system then displays screen 14.
* If a withdrawal is requested, the system checks the status (jammed or free) of the withdrawal chute in the terminal control file. If jammed, screen 10 is displayed; otherwise, screen 7 is displayed so the customer can enter the withdrawal amount.
* Once the withdrawal amount is entered, the system checks the terminal status file to see if it has enough currency to dispense. If it does not, screen 9 is displayed; otherwise, the withdrawal is processed.
* The system checks the customer balance (as described in the balance request transaction); if the funds in the account are insufficient, screen 8 is displayed.
* If the account balance is enough, screen 11 is displayed and the money is dispensed.
* The balance is printed on the transaction receipt as it is for a balance request transaction.
* After the cash has been removed, the system displays screen 14.
* When the “No” button is pressed in screens 10, 12, or 14, the system presents screen 15 and returns the customer’s ATM card. Once the card is removed from the card slot, screen 1 is displayed.
* When the “Yes” button is pressed in screens 10, 12, or 14, the system presents screen 5 so the customer can select additional transactions.

**Currency Convertor:**

* The currency conversion program is another event-driven program that emphasizes code associated with a GUI. A sample GUI is shown in Figure 2.5.

****

* The application converts US dollars to any of four currencies: Brazilian reals, Canadian dollars, European Union euros, and Japanese yen.
* Currency selection is governed by the radio buttons (option buttons), which are mutually exclusive.
* When a country is selected, the system responds by completing the label; for example, “Equivalent in …” becomes “Equivalent in Canadian dollars” if the Canada button is clicked. Also, a small Canadian flag appears next to the output position for the equivalent currency amount.
* Either before or after currency selection, the user inputs an amount in US dollars. Once both tasks are accomplished, the user can click on the Compute button, the Clear button, or the Quit button.
* Clicking on the Compute button results in the conversion of the US dollar amount to the equivalent amount in the selected currency.
* Clicking on the Clear button resets the currency selection, the US dollar amount, and the equivalent currency amount and the associated label.
* Clicking on the Quit button ends the application

**Saturn Windshield Wiper Controller**

* The windshield wiper on some Saturn automobiles is controlled by a lever with a dial.
* The lever has four positions: OFF, INT (for intermittent), LOW, and HIGH; and the dial has three positions, numbered simply 1, 2, and 3.
* The dial positions indicate three intermittent speeds, and the dial position is relevant only when the lever is at the INT position.
* The decision table below shows the windshield wiper speeds (in wipes per minute) for the lever and dial positions.

