SOFTWARE TESTING

UNIT-II [12 Hours]

Equivalence Class Testing: Equivalence Classes, Weak Normal Vs Strong Normal Equivalence Class Testing, Weak Robust Vs Strong Robust Equivalence Class Testing, Equivalence Class Test Cases for Triangle Problem, Equivalence Class Test cases for NextDate Function and Equivalence Class Test Cases for Commission Problem, Guidelines for Equivalence Class Testing.

Decision Table Based Testing: Decision Tables, Test Cases for the Triangle Problem, Test Cases for the NextDate Function, Test cases for the commission problem, Guidelines and observations.

Data Flow Testing: Definition Use Testing, Example – The Commission Problem, Slice-Based Testing, Guidelines and Observations.

CHAPTER 4

What is Decision Table?

A decision table is a systematic and structured method used in software testing and system analysis to represent complex logical relationships between conditions and actions. It provides a visual and tabular representation of different combinations of input conditions and corresponding outcomes or actions. Decision tables help in designing test cases, analyzing decision-making logic, and ensuring comprehensive coverage of various scenarios within a software system.

What is Decision Table Based Testing?

Decision table based testing is a systematic testing technique that uses decision tables to design and execute test cases for software systems. This approach involves creating decision tables that capture various combinations of input conditions and corresponding actions or outcomes. Testers analyze these decision tables to derive test cases that cover different scenarios based on the defined conditions.

Characteristics of Decision Tables

Decision tables are a powerful tool for organizing and managing complex decision-making processes in software development and other fields that require logical analysis. Some key characteristics of decision tables are:

1. **Conditions:** Decision tables include conditions that represent the input variables or factors that need to be evaluated. These conditions define the criteria that influence the decision-making process within the system.
2. **Actions:** Actions in decision tables specify the outcomes or responses that result from the evaluation of specific conditions. Each combination of conditions leads to a corresponding action or set of actions.

3. Rules: Rules in decision tables represent individual rows that capture a unique combination of conditions and the associated actions. Each rule defines a specific scenario or decision logic within the system.

4. Structured Format: Decision tables are organized into distinct sections, such as the stub portion, entry portion, condition portion, and action portion. This structured format provides a clear framework for organizing and analyzing the decision-making logic.

5. Comprehensive Coverage: Decision tables aim to ensure comprehensive coverage of various scenarios by considering all possible combinations of conditions and actions. This helps in identifying potential gaps in the decision logic and designing test cases that cover all possible paths through the system.

6. Visual Representation: Decision tables offer a visual representation of complex decision logic, making it easier for testers and developers to understand the relationships between conditions and actions. The tabular format simplifies the analysis of decision-making processes.

7. Scalability: Decision tables can scale to accommodate a large number of conditions and actions, making them suitable for analyzing complex systems with multiple decision points. They can handle increasing complexity while maintaining clarity and organization.

8. Flexibility: Decision tables provide flexibility in modifying and updating the decision logic by adjusting conditions, actions, or rules as needed. This adaptability allows for easy maintenance and refinement of the decision-making process over time.

4.2 Importance (or) Benefits of Decision Tables

Decision tables play a crucial role in software testing by enhancing clarity, improving test coverage, identifying dependencies, validating logic, mitigating risks, facilitating communication, and ensuring scalability and flexibility in decision-making processes. Their importance lies in their ability to streamline testing efforts, enhance understanding, and promote effective decision-making within software systems.

1. Clarity and Understanding: Decision tables provide a clear and structured representation of complex decision logic, making it easier for stakeholders, including testers, developers, and business analysts, to understand the relationships between conditions and actions within a system.

2. Comprehensive Coverage: By systematically capturing all possible combinations of conditions and corresponding actions, decision tables help ensure comprehensive test coverage. This reduces the risk of overlooking critical scenarios and improves the overall quality of testing.

3. Effective Test Case Design: Decision tables serve as a valuable tool for designing test cases that cover various scenarios based on different combinations of conditions. Testers can use decision tables to create targeted test cases that validate the behavior of the system under specific conditions.

4. Identification of Dependencies: Decision tables highlight dependencies between input conditions and output actions, enabling testers to identify complex relationships and potential dependencies that may impact the system's behavior. This insight is crucial for thorough testing.

5. Logic Validation: Decision tables facilitate the validation of decision-making logic within a system. By analyzing the rules and conditions in the table, testers can verify that the system behaves as expected under different conditions and that the logic is correctly implemented.

6. Risk Mitigation: Using decision tables helps mitigate the risk of overlooking critical decision paths or scenarios during testing. By systematically documenting all possible conditions and actions, testers can address potential risks and ensure that the system functions correctly in diverse scenarios.

7. Efficient Communication: Decision tables provide a standardized format for communicating decision logic across teams and stakeholders. They enable effective collaboration by presenting complex information in a structured and easily understandable manner.

8. Scalability and Flexibility: Decision tables can scale to accommodate increasing complexity in decision-making processes. They offer flexibility in modifying conditions, actions, and rules, allowing for easy adaptation to changing requirements and evolving system behavior.

Decision Table Design Format (or) Components of Decision Table

Decision tables follow a structured format that includes specific components to represent the conditions, actions, and rules governing the decision-making process within a system. The design format typically consists of the following sections:

1. Condition Stub:

- **Purpose:** Lists all the conditions relevant to the decision process. Conditions are typically questions or statements that can be answered with Yes (Y), No (N), or Don't Care (-).
- **Location:** Left side of the table.

2. Condition Entries:

- **Purpose:** Contains entries (Y, N, -) corresponding to each condition for every rule defined in the decision table.
- **'Don't Care' Entries:** Sometimes a condition in a particular rule might not affect the outcome, indicated by a '-' entry. This means that the action does not depend on this condition's state.
- **Location:** Right of the condition stub, divided by a vertical line.

3. Action Stub:

- **Purpose:** Lists the possible actions that can be taken based on the evaluation of the conditions.

- **Location:** Below the condition stub.

4. Action Entries:

- **Purpose:** Indicates which actions are to be taken for each rule by marking with an X or leave blank based on the condition entries above.
- **Location:** Below the condition entries, divided by a horizontal line.

5. Rules:

- **Purpose:** Represents a unique combination of condition entries and specifies the resulting actions. Each rule is a feasible scenario that might occur, and the table explores the outcomes.
- **Location:** Each column in the entry portion of the table.

Example: Decision Table Design Format

Let us understand the condition stub, the condition entries, the action stub, and the action entries from the below table. A column in the entry portion is a rule. Rules indicate which actions, if any, are taken for the circumstances indicated in the condition portion of the rule. In the decision table in Table 4.1, when conditions c1, c2, and c3 are all true, actions a1 and a2 occur. When c1 and c2 are both true and c3 is false, then actions a1 and a3 occur. The entry for c3 in the rule where c1 is true and c2 is false is called a "don't care" entry. The don't care entry has two major interpretations: the condition is irrelevant, or the condition does not apply. Sometimes people will enter the "n/a" symbol for this latter interpretation.

Stub	**Rule 1**	**Rule 2**	**Rules 3, 4**	**Rule 5**	**Rule 6**	**Rules 7, 8**
c1	T	T	T	F	F	F
c2	T	T	F	T	T	F
c3	T	F	-	T	F	-
a1	X	X		X		
a2	X				X	
a3		X		X		
a4			X			X

When we have binary conditions (true/false, yes/no, 0/1), the condition portion of a decision table is a truth table (from propositional logic) that has been rotated 90°. This structure guarantees that we consider every possible combination of condition values. When we use decision tables for test case identification, this completeness property of a decision table guarantees a form of complete testing.

Decision tables in which all the conditions are binary are called **Limited Entry Decision Tables (LEDTs)**. If conditions are allowed to have several values, the resulting tables are called **Extended Entry Decision Tables (EEDTs)**.

Example 1 Decision Table Imagine a software system that manages subscriptions for an online platform. The system must decide whether to activate a subscription based on user age verification and payment status.

Sample Decision Table:

Condition Stub	**Rule 1**	**Rule 2**	**Rule 3**	**Rule 4**
User is over 18	Y	Y	N	N
Payment received	Y	N	Y	N
Action Stub				
Activate Subscription	X			
Send Reminder		X		
Reject Application			X	X

Detailed Explanation:

1. Rule 1 (Y, Y):

- **Conditions:** User is over 18 (Y) and payment received (Y).
- **Action:** Activate Subscription.
- **Scenario:** The user meets all criteria for subscription activation.

2. Rule 2 (Y, N):

- **Conditions:** User is over 18 (Y) but payment not received (N).
- **Action:** Send Reminder.
- **Scenario:** The user is eligible but needs to complete payment; hence a reminder is sent.

3. Rule 3 (N, Y):

- **Conditions:** User is not over 18 (N) but payment received (Y).
- **Action:** Reject Application.
- **Scenario:** Even though the payment was received, the user does not meet the age requirement, leading to rejection.

4. Rule 4 (N, N):

- **Conditions:** User is not over 18 (N) and payment not received (N).
- **Action:** Reject Application.
- **Scenario:** Neither condition is met; the application is outright rejected.

Decision Table Techniques

Decision tables are a powerful tool for designing test cases especially when handling complex logical conditions and their associated actions. Let us discuss the main techniques used in decision tables.

1. Completeness and Redundancy:

Completeness in decision tables means that the table must account for every possible combination of input conditions. Redundancy refers to the presence of duplicate rules within the decision table that lead to the same actions based on the same or similar conditions.

Benefits of Ensuring Completeness and Minimizing Redundancy:

1. Ensures Comprehensive Coverage: Completeness in decision tables ensures that every possible input combination is considered, which significantly reduces the risk of defects slipping through undetected. This thorough coverage is essential for critical systems where failures can have severe implications.

2. Identifies All Possible Behaviors: By accounting for all potential conditions, completeness helps testers understand every possible behavior of the system. This understanding is crucial for verifying that the system operates correctly across all scenarios, which is especially important in complex systems with many interdependent components.

3. Reduces the Risk of Unexpected Issues: Complete decision tables help in detecting and addressing edge cases and rare scenarios that might not be immediately apparent. This proactive identification helps prevent unexpected issues post-deployment, thereby enhancing the reliability and stability of the system.

4. Streamlines Testing and Maintenance: Minimizing redundancy in decision tables makes the testing process more streamlined and resource-efficient. It eliminates unnecessary testing efforts, saving time and reducing the potential for confusion or error that can arise from handling duplicate test cases.

Example: Let's consider a decision table for an online booking system where users can book rooms depending on membership status and room availability.

1. Conditions:

- Membership Status: Valid or Invalid
- Room Availability: Available or Not Available.

2. Actions:

- Book Room: Proceed with the booking.
- Show Error: Display an error message regarding availability or membership status.

Decision Table:

Condition Stub	**Rule 1**	**Rule 2**	**Rule 3**	**Rule 4**
Membership Status (Valid)	Y	Y	N	N
Room Availability (Available)	Y	N	Y	N
Action Stub				
Book Room	X			
Show Error		X	X	X

Explanation of the Table:

- **Rule 1:** When a user has a valid membership and the room is available, the action is to book the room. This rule is necessary and unique.
- **Rule 2:** For a valid member when the room is not available, the system shows an error about room availability.
- **Rule 3:** If the membership is invalid but the room is available, it displays an error about membership status.
- **Rule 4:** Shows an error when both membership is invalid and the room is not available. This covers the scenario where multiple issues prevent booking.

By ensuring every combination of conditions is addressed (completeness) and avoiding duplication of the same outputs for the same inputs (redundancy), the decision table helps in thorough testing of the booking system. This method ensures that the software's reaction to every possible user interaction is tested and verified, which enhances reliability and user satisfaction.

2. Don't Care Entries:

Don't care entries in decision tables are used when the state of a particular condition does not affect the outcome for specific rules. This approach simplifies the table by focusing only on the conditions that influence the decision, reducing the complexity and size of the table.

Advantages of Using Don't Care Entries:

1. Simplification: Reduces the number of rules in the decision table by not considering irrelevant conditions, making the table easier to read and understand.

2. Focus on Relevant Conditions: Allows stakeholders to focus only on the conditions that impact outcomes, which can speed up decision-making processes.

3. Efficiency: Minimizes the effort needed to test scenarios that are unaffected by certain conditions, enhancing the efficiency of the testing process.

Example: Consider a monitoring system that checks for errors and system load. The system needs to decide whether to send an alert or continue monitoring based on these factors.

1. Conditions:

- **System Load is High:** This condition can impact how the system prioritizes resources but might not directly influence whether an alert should be sent or not.
- **Error Reported:** This condition directly determines whether an alert needs to be sent.

2. Actions:

- **Send Alert:** Triggered when an error is reported, regardless of the system load.
- **Continue Monitoring:** The default action when no errors are reported.

Decision Table:

Condition Stub	**Rule 1**	**Rule 2**
System Load High	Don't Care	Don't Care
Error Reported	Yes	No
Action Stub		
Send Alert	X	
Send Alert		X

Explanation of the Table:

- **Rule 1:** The action to send an alert is triggered if an error is reported, irrespective of whether the system load is high or not. Here, the 'Don't Care' entry for the system load indicates that the presence or absence of high system load does not impact the decision to send an alert.
- **Rule 2:** Continue monitoring is the action when no error is reported. Again, the system load does not influence this action, hence the 'Don't Care' designation.

In this example, using 'Don't Care' entries effectively ignores the system load when deciding to send alerts or continue monitoring, as the action depends solely on whether an error is reported.

This technique is particularly useful in systems where some inputs are significantly more critical than others in determining the output.

3. Impossible Rules:

Impossible rules in decision tables happen when some conditions just don't make sense together, so the actions linked to them wouldn't apply. By spotting and marking these impossible situations, we prevent confusion and make sure the decision table shows the system's rules correctly.

Advantages of Identifying Impossible Rules:

1. Prevent Logical Errors: Helps prevent situations where the system might attempt to execute actions under conditions that logically cannot happen.

2. Clarity in Test Case Design: Clear identification of impossible scenarios helps in designing more effective test cases and avoids wasting resources on testing unrealistic conditions.

3. Enhanced Understanding: Provides all stakeholders with a better understanding of the system's constraints and logical flows, facilitating better system design and troubleshooting.

Example: A payment processing system needs to decide whether to complete or decline a transaction based on the validity of a credit card and whether the payment has been processed.

1. Conditions:

- **Credit Card Valid:** Determines if the credit card being used is valid.
- **Payment Processed:** Indicates whether the payment transaction has successfully been processed.

2. Actions:

- **Complete Transaction:** Execute this action if the transaction can be successfully completed.
- **Decline Transaction:** Execute this action if the transaction must be declined due to various issues.

Decision Table:

Condition Stub	**Rule 1**	**Rule 2**	**Rule 3**
Credit Card Valid	Yes	No	Yes
Payment Processed	Yes	Yes	No
Action Stub			

Complete Transaction	X		X
Decline Transaction		Impossible (X)	

Explanation of the Table:

- **Rule 1:** This rule represents a scenario where both the credit card is valid and the payment has been processed. The logical action is to complete the transaction, so the "Complete Transaction" action is marked.
- **Rule 2:** Here, the credit card is not valid, but the condition states that the payment has been processed. This is an impossible scenario because a payment cannot be processed using an invalid credit card. Thus, this rule is marked as impossible, and the intended action (decline the transaction) is indicated but should not be executed under this rule because the scenario cannot occur.
- **Rule 3:** The credit card is valid, but the payment has not been processed successfully. The logical action is to decline the transaction.

4. Rule Count Adjustment:

Rule Count Adjustment is a technique used in decision tables to ensure that the rule counts accurately reflect the number of unique and meaningful test scenarios, especially when "don't care" entries are used. "Don't care" entries indicate that the outcome does not depend on the state of a particular condition, allowing for multiple possible states without affecting the rule's actions.

Advantages of Rule Count Adjustment:

1. Accuracy in Testing: Adjusting the rule count ensures that the number of test cases reflects all possible scenarios that might need to be tested, enhancing the thoroughness and reliability of testing.

2. Effective Resource Allocation: By understanding the actual number of scenarios each rule represents, resources can be better allocated to test all relevant cases effectively.

3. Clarity in Specification: This adjustment clarifies how "don't care" entries impact the overall decision-making process, helping to prevent misunderstandings and ensuring that all potential scenarios are considered.

Example: A system controls access based on whether a login is required and whether a form is filled out correctly.

1. Conditions:

- **Login Required:** Specifies whether the user needs to log in.
- **Form Filled:** Indicates whether the user has filled out a form correctly.

2. Actions:

- **Allow Access:** The user is allowed access to the system.
- **Deny Access:** The user is denied access.

Decision Table:

Condition Stub	**Rule 1**	**Rule 2**
Login Required	Don't care	Yes
Form Filled	Yes	No
Action Stub		
Allow Access	X	
Deny Access		X

Explanation of the Table:

- **Rule 1:** The condition for "Login Required" is marked as "don't care", which means the action "Allow Access" can occur regardless of whether login is required or not, as long as the form is filled correctly ("Form Filled" = Yes).
- **Rule 2:** Access is denied ("Deny Access") when login is required and the form is not filled correctly.

Rule Count Adjustment:

- **Without Adjustment:** Normally, each rule in a decision table corresponds to a specific scenario. Here, because "Login Required" in Rule 1 is a "don't care", this single rule actually represents two scenarios: one where login is required and the form is filled, and another where login is not required and the form is filled.
- **With Adjustment:** For Rule 1, each "don't care" state effectively doubles the count of scenarios it represents. Thus, Rule 1 actually covers two scenarios:
    - Login Required = Yes, Form Filled = Yes
    - Login Required = No, Form Filled = Yes

The Rule Count Adjustment technique is crucial for ensuring that decision tables not only represent but also accurately count all unique conditions covered by rules, particularly when "don't care" entries expand the implications of a rule beyond a single scenario.

5. Using Equivalence Classes:

Using Equivalence Classes is a technique in decision table design that involves defining conditions based on distinct groups or classes of inputs that are expected to behave similarly. This method helps simplify the testing process by reducing the number of individual cases that need to be tested, assuming that testing one sample from each class is representative of the entire class.

Benefits of Using Equivalence Classes:

1. Simplification of Test Cases: By treating all instances within an equivalence class the same way, fewer tests need to be designed and executed, focusing resources on varied scenarios.

2. Comprehensive Coverage: Ensures that each unique combination of input classes is considered, providing thorough coverage without the redundancy of testing each possible input value.

3. Efficiency in Test Execution: Reduces the number of tests required while maintaining an effective assessment of system behavior across different user categories and ticket types.

Example: A ticketing system issues tickets based on the age group of the customer and the type of ticket they choose (Standard or Premium).

1. Conditions:

- **Age Group:** Defines whether the customer is an Adult, Child, or Senior.
- **Ticket Type:** Specifies whether the ticket is Standard or Premium.

2. Actions:

- **Issue Ticket:** A ticket is issued to the customer.
- **Offer Discount:** A discount is offered, typically associated with specific conditions like age group or ticket type.

Decision Table:

Condition Stub	**Rule 1**	**Rule 2**	**Rule 3**	**Rule 4**
Age Group	Adult	Child	Senior	Adult

Ticket Type	Standard	Standard	Premium	Premium
Action Stub				
Issue Ticket	X	X	X	X
Offer Discount			X	

Explanation of the Table:

- **Rule 1:** Adults buying standard tickets are simply issued a ticket.
- **Rule 2:** Children with standard tickets receive the same action as adults with standard tickets due to the simplicity of the transaction.
- **Rule 3:** Seniors purchasing premium tickets get both an issued ticket and a discount, acknowledging the combination of a higher-priced ticket and a potentially discounted group.
- **Rule 4:** Adults purchasing premium tickets are issued a ticket but do not receive a discount, distinguishing this scenario from seniors due to the absence of age-related incentives.

This technique is particularly useful in systems where the input space is large and can be logically divided into categories or classes that are expected to elicit the same response from the system. By applying this approach, decision tables remain manageable while still offering a structured and systematic way to capture and test complex business rules and their outcomes.

6. Mutually Exclusive Conditions:

Design decision tables need to clearly separate conditions that cannot occur simultaneously to ensure that each scenario is distinct and unambiguous. Mutually exclusive conditions in decision tables help simplify the decision-making process by ensuring that conditions in a given rule cannot overlap. This clarity prevents the creation of impossible or contradictory rules and helps maintain the integrity of logical evaluations.

Benefits of Mutually Exclusive Conditions

1. Simplification of Decision Logic: Mutually exclusive conditions ensure that each condition or scenario is distinct and does not overlap with others. This simplification helps to reduce complexity in understanding and analyzing the decision logic, making it more straightforward to implement and test.

2. Prevention of Rule Overlap: Since mutually exclusive conditions do not overlap, they prevent multiple rules from being triggered simultaneously for a single set of inputs. This clear delineation helps avoid conflicts or contradictions in decision outcomes, ensuring consistent and predictable actions.

3. Efficient Testing: Testing becomes more manageable and less time-consuming because each condition set leads to a unique path or outcome. This efficiency aids in targeted testing strategies and reduces the potential for errors during test case execution, as the conditions define clear boundaries for each test scenario.

4. Enhanced Clarity and Communication: Decision tables with mutually exclusive conditions are easier to read and understand. They provide a clear visual representation of how different scenarios are handled, which can be beneficial for communication among team members and stakeholders, ensuring everyone understands the logic and expected behaviors.

Example

Let us consider a simple banking transactions.

1. Conditions:

- **Account Type:** Checking, Savings (Cannot be both)
- **Transaction Type:** Deposit, Withdrawal (Cannot be both)

2. Actions:

- Process Transaction
- Reject Transaction

Decision Table:

Condition Stub	**Rule 1**	**Rule 2**	**Rule 3**	**Rule 4**
Account Type - Checking	Yes	No	Yes	No
Account Type - Savings	No	Yes	No	Yes
Transaction Type - Deposit	Yes	Yes	No	No
Transaction Type - Withdrawal	No	No	Yes	Yes
Action Stub				
Process Transaction	X	X	X	X
Reject Transaction				

Explanation of the Table:

- **Rule 1:** Rule 1: Transaction is a deposit in a checking account. Process the transaction because conditions are valid and mutually exclusive.
- **Rule 2:** Transaction is a deposit in a savings account. Process the transaction as it meets the mutually exclusive conditions.
- **Rule 3:** Transaction is a withdrawal from a checking account. Again, process it due to valid and exclusive conditions.

- **Rule 4:** Transaction is a withdrawal from a savings account. Process the transaction as all conditions align without conflict.

This table ensures that each combination of account type and transaction type is uniquely handled, avoiding overlaps such as attempting to process a deposit and withdrawal simultaneously.

Test Cases for the Triangle Problem

The decision table for the triangle problem can be constructed by breaking down the conditions into more specific tests that consider all inequalities and equalities among the triangle's sides. The decision table method ensures that all logical scenarios are covered.

Conditions:

- **c1:** a<b+c? Checks if side a is less than the sum of sides b and c, which is a requirement for forming a triangle.
- **c2:** b<a+c? Similar check for side b.
- **c3:** c<a+b? Similar check for side c.
- **c4:** a=b? Checks if sides a and b are equal.
- **c5:** a=c? Checks if sides a and c are equal.
- **c6:** b=c? Checks if sides b and c are equal.

Actions:

- **a1:** Not a triangle - Action if the sides do not satisfy the triangle inequality theorem.
- **a2:** Scalene - Action if no sides are equal and the sides form a triangle.
- **a3:** Isosceles - Action if exactly two sides are equal and the sides form a triangle.
- **a4:** Equilateral - Action if all three sides are equal and the sides form a triangle.
- **a5:** Impossible - Marks configurations that are logically contradictory or impossible.

Decision Table:

	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**
c1: a<b+c?	F	T	T	T	T	T	T	T	T	T	T
c2: b<a+c?	-	F	T	T	T	T	T	T	T	T	T
c3: c<a+b?	-	-	F	T	T	T	T	T	T	T	T
c4: a=b?	-	-	-	T	T	T	T	F	F	F	F
c5: a=c?	-	-	-	T	T	F	F	T	T	F	F
c6: b=c?	-	-	-	T	F	T	F	T	F	T	F

a1: Not a triangle	X	X	X								
a2: Scalene											X
a3: Isosceles							X		X	X	
a4: Equilateral				X							
a5: Impossible					X	X		X			

Analysis:

- **Columns (Rules):** Each column under the conditions represents a rule which is a specific scenario that could occur given the set of conditions. The way the table is set up guarantees that every possible logical scenario is considered.
- **Don't Care Entries ("-"):** These are used where the outcome of a condition does not affect the outcome of the test because other conditions already determine the result. For example, if c1 is false (F), it doesn't matter what the values of c2, c3, c4, c5, or c6 are, the triangle cannot be formed.
- **Rule 1 (F, -, -, -, -, -):** Indicates that if c1 is false, the sides cannot form a triangle, triggering action a1: Not a triangle.
- **Rule 10 (T, T, T, F, F, T):** This rule represents a scenario where all inequalities are satisfied (T for c1, c2, c3), but a=b is false (F for c4), a=c is false (F for c5), and b=c is true (T for c6). This means the triangle is isosceles (a3).
- **Rule 12 (T, T, T, T, T, T):** All conditions are true, indicating an equilateral triangle (a4).
- **Rules with "Impossible" (a5):** These rules indicate scenarios where the set conditions logically contradict the properties of a triangle, such as having one side equal to the sum of the others, which would not form a triangle.

Test Cases Derived from Decision Table:

Case ID	**a**	**b**	**c**	**Expected Output**
Rule 1: DT1	4	1	2	Not a triangle
Rule 2: DT2	1	4	2	Not a triangle
Rule 3: DT3	1	2	4	Not a triangle
Rule 4: DT4	5	5	5	Equilateral
Rule 5: DT5	?	?	?	Impossible
Rule 6: DT6	?	?	?	Impossible
Rule 7: DT7	2	2	3	Isosceles
Rule 8: DT8	?	?	?	Impossible
Rule 9: DT9	2	3	2	Isosceles

Rule 10: DT10	3	2	2	Isosceles
Rule 11: DT11	3	4	5	Scalene

Test Cases for the Next Date Function

The NextDate function is designed to calculate the date of the following day given a specific input date consisting of day, month, and year. This function addresses various complexities associated with the Gregorian calendar, including different month lengths, leap years, and transitions from one month or year to the next.

Conditions

- **Month:** Specifies the current month. This is crucial as months have varying numbers of days.
- **Day:** Specifies the current day within the month.
- **Year:** Determines if the year is a leap year or a non-leap year based on the rules of the Gregorian calendar.

Actions

- **Next Day:** The day number for the following day.
- **Next Month:** The month number for the following day.
- **Next Year:** The year for the following day.

Decision Table for Valid Scenarios:

Condition \ Rule	**Rule 1**	**Rule 2**	**Rule 3**	**Rule 4**	**Rule 5**	**Rule 6**	**Rule 7**	**Rule 8**
Month	Jan-Dec (not last day)	Jan-Nov (last day)	Dec (last day)	Feb (28, non-leap)	Feb (28, leap)	Feb (29, leap)	30-Day Months (Apr, Jun, Sep, Nov)	31-Day Months (Jan, Mar, May, July, Aug, Oct)
Day	Not Last Day	Last Day	Last Day	28	28	29	30	31
Year	Any	Any	Any	Non-Leap Yea	Leap Yea	Leap Year	Any	Any
Actions								
Next Day	+1 to current day	01	01	01	01	01	01	01

Next Month	Same	+1 to current month	Jan	Mar	Same	Mar	+1 to current month	+1 to current month
Next Year	Same	Same	+1 to current year (Next year)	Same	Same	Same	Same	Same

Analysis:

- **Rule 1:** For any regular day that is not the last of the month, simply increment the day (e.g., March 14th to March 15th).
- **Rule 2:** For the last day of any month except December, the date changes to the first day of the next month (e.g., January 31st to February 1st).
- **Rule 3:** For December 31st, the date changes to January 1st of the next year (e.g., December 31st, 2021 to January 1st, 2022).
- **Rule 4:** For February 28th in a non-leap year, the next date is March 1st (e.g., February 28th, 2021 to March 1st, 2021).
- **Rule 5:** For February 28th in a leap year, the next date is February 29th (e.g., February 28th, 2020 to February 29th, 2020).
- **Rule 6:** For February 29th in a leap year, the next date is March 1st (e.g., February 29th, 2020 to March 1st, 2020).
- **Rule 7:** For the last day of months with 30 days, the date changes to the first day of the next month (e.g., April 30th to May 1st).
- **Rule 8:** For the last day of months with 31 days, the date changes to the first of the next month except for December (e.g., July 31st to August 1st).

This decision table effectively maps out the transitions between different dates, ensuring that each scenario is specifically addressed, thereby facilitating comprehensive testing and validation of the NextDate function.

Test Cases Derived from Decision Table:

Case ID	Day	Month	Year	Expected Output
Rule 1: DT1	15	6	2022	16/6/2022
Rule 2: DT2	30	4	2022	1/5/2022
Rule 3: DT3	31	12	2022	1/1/2023
Rule 4: DT4	28	2	2023	1/3/2023
Rule 5: DT5	28	2	2024	29/2/2024
Rule 6: DT6	29	2	2024	1/3/2024
Rule 7: DT7	30	4	2024	1/5/2024
Rule 8: DT8	31	5	2024	1/6/2024

Test Cases for the Commission Problem

The Commission problem revolves around calculating the commission for a salesperson based on the quantity of locks, stocks, and barrels sold within a month. Each product has a fixed selling price and a commission rate that varies based on the total sales amount.

Problem Statement:

The Commission Problem involves a scenario where a salesperson sells rifle components (locks, stocks, and barrels) manufactured by a gunsmith in Missouri. The problem statement includes the following key elements:

Product Costs:

- Locks cost $45 each.
- Stocks cost $30 each.
- Barrels cost $25 each.

Sales Requirements:

- The salesperson must sell at least one lock, one stock, and one barrel each month, but they do not necessarily need to be sold as part of a complete rifle.
- There are maximum sales limits due to production constraints: 70 locks, 80 stocks, and 90 barrels per month.

Sales Reporting:

- After visiting each town, the salesperson sends a telegram to the gunsmith about the number of locks, stocks, and barrels sold.
- At the end of the month, a final telegram with the figures "-1 locks sold" signals the completion of that month's sales, prompting the gunsmith to compute the salesperson's commission.

Commission Calculation: The commission structure is tiered:

- 10% commission on sales up to and including $1000.
- 15% commission on the next $800 of sales.
- 20% commission on any sales beyond $1800.

Conditions for Decision Table:

- **Locks Sold:** Number of locks sold within their allowed range.
- **Stocks Sold:** Number of stocks sold within their allowed range.

- **Barrels Sold:** Number of barrels sold within their allowed range.
- **Total Sales Bracket:** Defined brackets based on the total sales amount for calculating commissions.

Actions Defined for Decision Table:

- **Calculate Commission:** The specific formula used based on total sales.
- **Validate Sales:** Indicates whether the sales are within the permissible range or not.

Decision Table:

Condition \ Rule	**Rule 1**	**Rule 2**	**Rule 3**	**Rule 4**	**Rule 5**	**Rule 6**
Locks Sold	1-70	1-70	1-70	>70	Any	Any
Stocks Sold	1-80	1-80	1-80	Any	>80	Any
Barrels Sold	1-90	1-90	1-90	Any	Any	>90
Total Sales	≤ $1000	$1001-$1800	> $1800	N/A	N/A	N/A
Actions						
Calculate Comm.	10% of sales	$100 + 15% over $1000	$220 + 20% over $1800	0	0	0
Validate Sales	Valid	Valid	Valid	Invalid	Invalid	Invalid

Analysis:

- **Rule 1:** Applies when all items are within limits and total sales are ≤ 1000. Commission is 10% of sales.
- **Rule 2:** Applies when all items are within limits and total sales are between 1001 and 1800. Commission includes a base of 100 plus 15% of the amount over 1000.
- **Rule 3:** Applies when all items are within limits and total sales exceed 1800. Commission includes a base of 220 plus 20% of the amount over 1800.
- **Rule 4, 5, 6:** Handle invalid sales scenarios where the number of locks, stocks, or barrels exceeds their respective maximum limits. In these cases, sales are marked invalid and no commission is awarded.

Test Cases Derived from Decision Table:

Case ID	Locks	Stocks	Barrels	Total Sales	Commission
Rule 1: DT1	5	5	5	$500	$50
Rule 2: DT2	15	15	15	$1500	$175
Rule 3: DT3	20	25	30	$2400	$340
Rule 4: DT4	80	40	40	Invalid	$0
Rule 5: DT5	30	85	30	Invalid	$0
Rule 6: DT6	30	40	95	Invalid	$0

1. **Rule C1 (DT1) - Valid under $1000 range:**
    - Locks = 5, Stocks = 5, Barrels = 5
    - Total Sales = $(5\times45)+(5\times30)+(5\times25)=225+150+125=\$500$
    - Commission = $500\times0.10=\$50$
2. **Rule C2 (DT2) - Valid within the $1001 to $1800 range:**
    - Locks = 15, Stocks = 15, Barrels = 15
    - Total Sales = $(15\times45)+(15\times30)+(15\times25)=675+450+375=\$1500$
    - Commission = $100+(\$1500-\$1000)\times0.15=\$100+\$75=\$175$
3. **Rule C3 (DT3) - Valid above $1800:**
    - Locks = 20, Stocks = 25, Barrels = 30
    - Total Sales = $(20\times45)+(25\times30)+(30\times25)=900+750+750=\$2400$
    - Commission = $220+(\$2400-\$1800)\times0.20=\$220+\$120=\$340$
4. **Rule C4 (DT4) - Invalid due to exceeding lock limits:**
    - Locks = 80, Stocks = 40, Barrels = 40
    - Sales Invalid, Commission = 0
5. **Rule C5 (DT5) - Invalid due to exceeding stock limits:**
    - Locks = 30, Stocks = 85, Barrels = 30
    - Sales Invalid, Commission = 0

6. **Rule C6 (DT6) - Invalid due to exceeding barrel limits:**
    - Locks = 30, Stocks = 40, Barrels = 95
    - Sales Invalid, Commission = 0

Guidelines and Observations of Decision Table Testing

Decision table testing is particularly effective in scenarios where decision-making is complex, involving numerous logical conditions. This testing method is ideal for applications with significant conditional logic, like if-then-else structures, or where inputs and outputs are closely linked through logical operations. Some guidelines and observations of using decision table testing:

1. When to Use Decision Tables:

- Complex Decision Logic: Best for scenarios with if-then-else conditions.
- Interdependent Inputs: Useful when inputs are logically connected and affect each other's outcomes.
- Calculative Operations: Ideal for functions that perform calculations on subsets of input data.
- Input-Output Correlation: Effective in environments where inputs directly dictate outputs.
- High Complexity: Suitable for systems with complex pathways indicating many potential paths through the code.

2. Challenges with Scalability:

- Decision tables can become large as the number of conditions increases, with a binary decision table for 'n' conditions resulting in 2n rules. To manage this complexity:
    - **a) Extended Entry Tables:** Use tables that allow multiple values per condition to reduce the number of rules.
    - **b) Table Simplification:** Algebraically simplify the rules to make the table more manageable.
    - **c) Table Factoring:** Break down large tables into smaller, more manageable pieces.
    - **d) Pattern Recognition:** Identify and leverage repeating patterns to streamline the decision-making process.

3. Iterative Refinement:

- The initial set of conditions and actions may not perfectly capture the necessary logic or may be inefficient. Iteratively refining these elements, based on testing results and deeper understanding of the application, helps in developing more effective and streamlined decision tables.