SE-305 Software Qualtiy Engineering Contents

Note: The following software are used: Turbo C, Microsoft Office, HP Unified Functional Testing (Trial Version)

Theory

Introduction to Software Testing

Software: Software is a set of instructions to perform some task. Software is used in many applications of the real world. Some of the examples are

Application software, such as word processors

Firmware in a embedded system

Middleware, which controls and co-ordinates distributed systems

System software such as operating systems

Video Games

Websites

All of these applications need to run without any error and provide a quality service to the user of the application. In this regard the software has to be tested for its accurate and correct working.

Software Testing:

Testing can be defined in simple words as "Performing Verification and Validation of the Software Product" for its correctness and accuracy of working.

Functional Vs non-functional testing

Functional testing refers to tests that verify a specific action or function of the code. These are usually found in the code requirements documentation, although some development methodologies work from use cases or user stories. Functional tests tend to answer the question of "can the user do this" or "does this particular feature work".

Non-functional testing refers to aspects of the software that may not be related to a specific function or user action, such as scalability or security. Non-functional testing tends to answer such questions as "how many people can log in at once", or "how easy is it to hack this software".

Error, Fault and Failure:

Humans make errors in their thoughts, actions, and in the products that might result from their actions. Errors occur in the process of writing a program. A programmer makes an error (mistake), which results in a defect (fault, bug) in the software source code. If this defect is executed, in certain

situations the system will produce wrong results, causing a failure. Not all defects will necessarily result in failures. For example, defects in dead code will never result in failures. A defect can turn into a failure when the environment is changed. Examples of these changes in environment include the software being run on a new hardware platform, alterations in source data or interacting with different software. A single defect may result in a wide range of failure symptoms.

Not all software defects are caused by coding errors. One common source of expensive defects is caused by requirement gaps, e.g., unrecognized requirements that result in errors of omission by the program designer. A common source of requirements gaps is non-functional requirements such as testability, scalability, maintainability, usability, performance, and security.

Errors – Examples

Incorrect usage of software by users

Bad architecture and design by architects and designers

Bad programming by developers

Inadequate testing by testers

Wrong build using incorrect configuration items by Build Team Member

Fault - Examples

A fault is the manifestation of one or more errors

An incorrect statement

Wrong data type

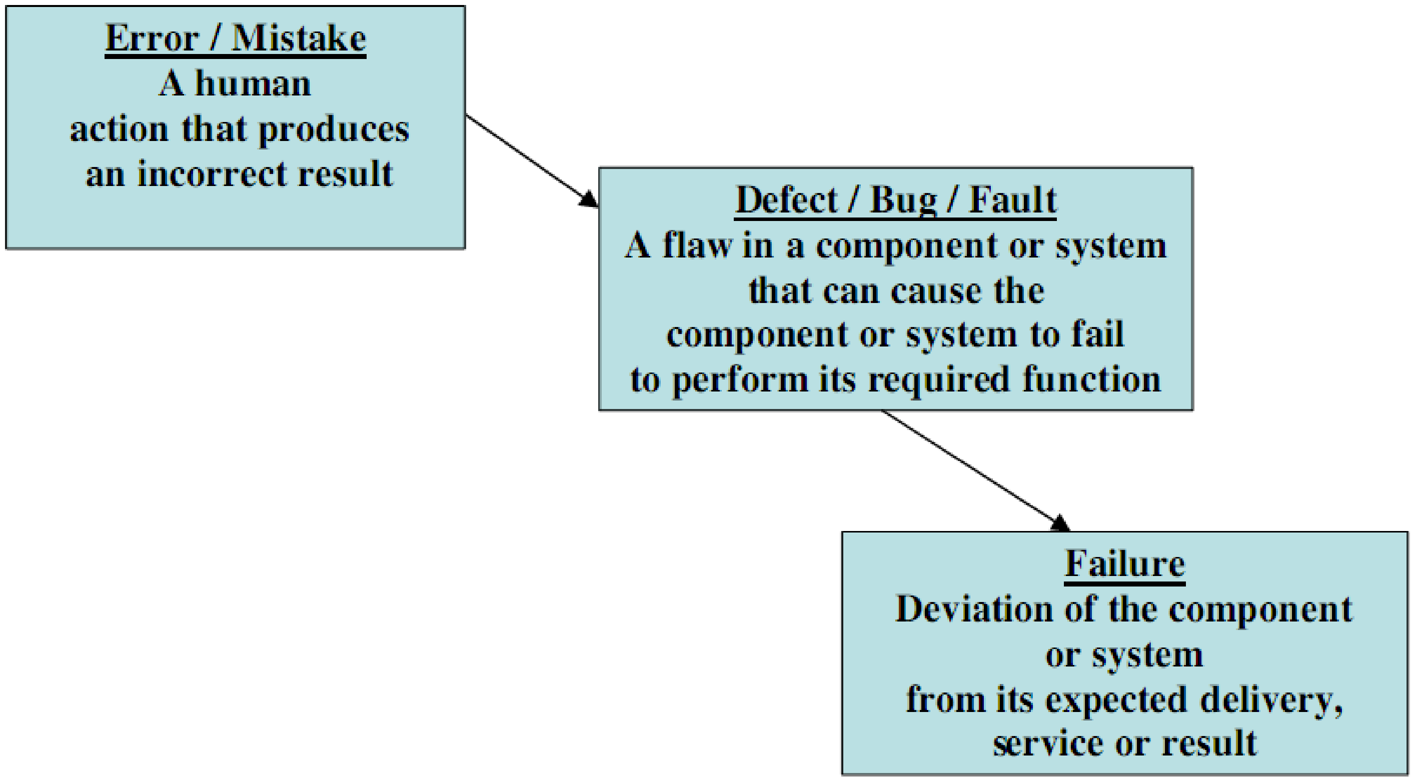
Wrong mathematical formula in design document

Missing functionality in the system

Failure

A failure occurs when a faulty piece of code is executed leading to incorrect state that propagates to the program's output. The following figure tells us how Error made by human will result in failure of the software.

Figure to understand Error, Fault and Failure


Error / Mistake
A human action that produces an incorrect result
Defect / Bug / Fault
A flaw in a component or system that can cause the component or system to fail to perform its required function
Failure
Deviation of the component or system from its expected delivery, service or result

Software Testing Objectives:

Testing is done to fulfill certain objectives

To discuss the distinctions between validation testing and defect testing

To describe the principles of system and component testing

To describe strategies for generating system test cases

To understand the essential characteristics of tool used for test automation

To find or prevent defects

To determine that software products satisfy specified requirements

Ensuring that a system is ready for use

Gaining confidence that it works

Providing information about the level of quality

Determining user acceptability

Software quality measures how well software is designed (quality of design), and how well the software conforms to that design (quality of conformance).

Software quality:

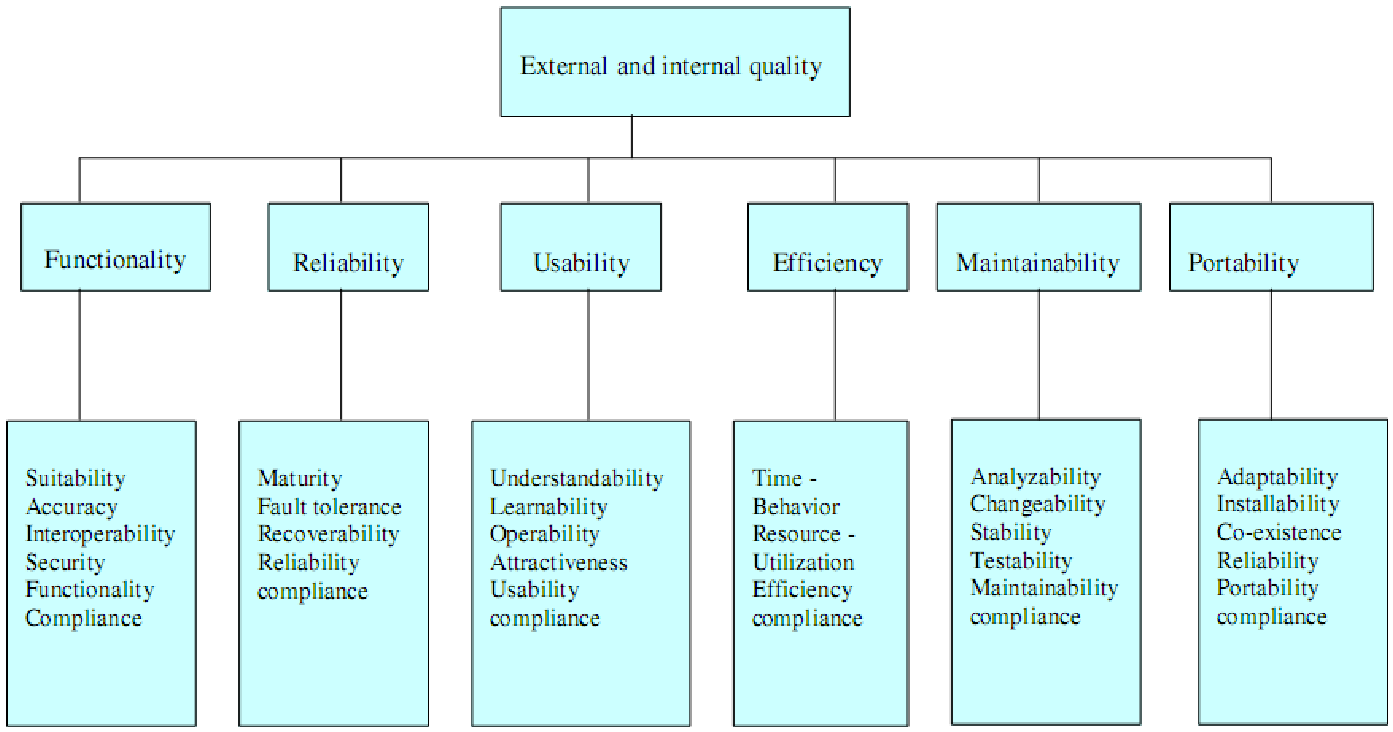
1) Conformance to specification: Quality that is defined as a matter of products and services whose measurable characteristics satisfy a fixed specification – that is, conformance to an in beforehand defined specification.

2) Meeting customer needs: Quality that is identified independent of any measurable characteristics. That is, quality is defined as the products or services capability to meet customer expectations – explicit or not.

Software quality is a multidimensional quantity and is measurable. To do this, we need to divide and measure software quality in terms of quality attributes:

Static Quality Attributes

Dynamic Quality Attributes


External and internal quality
Functionality
Reliability
Usability
Efficiency
Maintainability
Portability
Suitability Accuracy Interoperability Security Functionality Compliance
Maturity Fault tolerance Recoverability Reliability compliance
Understandability Learnability Operability Attractiveness Usability compliance
Time - Behavior Resource - Utilization Efficiency compliance
Analyzability Changeability Stability Testability Maintainability compliance
Adaptability Installability Co-existence Reliability Portability compliance

Software Quality Attributes:

- Static Attributes:

1. Maintainability and its Sub-characteristics:

In software engineering, the ease with which a software product can be modified in order to:

correct defects

meet new requirements

make future maintenance easier, or

cope with a changed environment

These activities are known as software maintenance.

A set of attributes that bear on the effort needed to make specified modifications are:

1.1 Analyzability: Attributes of software that bear on the effort needed for diagnosis of deficiencies or causes of failures, or for identification of parts to be modified

1.2 Changeability: Attributes of software that bear on the effort needed for modification, fault removal or for environmental change

1.3 Stability: Attributes of software that bear on the risk of unexpected effect of modifications

1.4 Testability: Attributes of software that bear on the effort needed for validating the modified software. The degree to which a system or component facilitates the establishment of test criteria and the performance of tests to determine whether those criteria are met

Static Testability Ex: Software Complexity

Dynamic Testability Ex: Test Coverage Criteria

Software Quality Attributes:

- Dynamic Attributes:

1. Completeness: The availability of all the features listed in the requirements or in the user manual.

2. Consistency: adherence to a common set of conventions and assumptions.

2.1 Compliance: Attributes of software that make the software adhere to application related standards or conventions or regulations in laws and similar prescriptions

2.2 Conformance : Attributes of software that make the software adhere to standards or conventions relating to portability.

3. Usability: The ease with which an application can be used. Usability testing also refers to testing of a product by its potential users

3.1 Understandability: Attributes of software that bear on the users' effort for recognizing the logical concept and its applicability

3.2 Learnability: Attributes of software that bear on the users' effort for learning its application

3.3 Operability : Attributes of software that bear on the users' effort for operation and operation control

4. Performance: The time the application takes to perform a requested task. Performance is considered as a nonfunctional requirement.

4.1 Time behavior: Attributes of software that bear on response processing times and on throughput rates in performances its function

4.2 Resource behavior: Attributes of software that bear on the amount of resource used and the duration of such use in performing its function

5. Reliability: Software Reliability is the probability of failure-free operation of software over a given time interval and under given conditions

5.1 Maturity: Attributes of software that bear on the frequency of failure by faults in the software.

5.2 Fault tolerance: Attributes of software that bear on its ability to maintain a specified level of performance in case of software faults or of infringement of its specified interface.

5.3 Recoverability: Attributes of software that bear on the capability to re-establish its level of performance and recover the data directly affected in case of a failure and on the time and effort needed for it.

For example, the Transmission Control Protocol (TCP) is designed to allow reliable two way communication in a packet-switched network, even in the presence of communications links which are imperfect or overloaded. It does this by requiring the endpoints of the communication to expect packet loss, duplication, reordering and corruption, so that these conditions do not damage data integrity, and only reduce throughput by a proportional amount.

Data formats may also be designed to degrade gracefully. HTML for example, is designed to be forward compatible, allowing new HTML entities to be ignored by Web browsers which do not understand them without causing the document to be unusable.

Recovery from errors in fault-tolerant systems can be characterized as either roll-forward or roll-back. When the system detects that it has made an error, roll-forward recovery takes the system state at that time and corrects it, to be able to move forward. Roll-back recovery reverts the system state back to some earlier, correct version, for example using check pointing, and moves forward from there.

6. Correctness: The correct operation of an application

6.1 Accurateness: Attributes of software that bear on the provision of right or agreed results or effects

6.2 Suitability: Attributes of software that bear on the presence and appropriateness of a set of functions for specified tasks

7. Correctness: It attempts to establish that the program is error- free, testing attempts to find if there are any errors in it.

Software Testing Life Cycle:

Software testing life cycle identifies what test activities to carry out and when (what is the best time) to accomplish those test activities. Even though testing differs between organizations, there is a testing life cycle.

Software Testing Life Cycle consists of six (generic) phases:

Test Planning,

Test Analysis,

Test Design,

Construction and verification,

Testing Cycles,

Final Testing and Implementation and

Post Implementation.

Software testing has its own life cycle that intersects with every stage of the SDLC. The basic requirements in software testing life cycle is to control/deal with software testing – Manual, Automated and Performance.

Test Planning

This is the phase where Project Manager has to decide what things need to be tested, do I have the appropriate budget etc. Naturally proper planning at this stage would greatly reduce the risk of low quality software. This planning will be an ongoing process with no end point.

Activities at this stage would include preparation of high level test plan-(according to IEEE test plan template The Software Test Plan (STP) is designed to prescribe the scope, approach, resources, and schedule of all testing activities. The plan must identify the items to be tested, the features to be tested, the types of testing to be performed, the personnel responsible for testing, the resources and schedule required to complete testing, and the risks associated with the plan.). Almost all of the activities done during this stage are included in this software test plan and revolve around a test plan.

Test Analysis

Once test plan is made and decided upon, next step is to delve little more into the project and decide what types of testing should be carried out at different stages of SDLC, do we need or plan to automate, if yes then when the appropriate time to automate is, what type of specific documentation I need for testing.

Proper and regular meetings should be held between testing teams, project managers, and development teams, Business Analysts to check the progress of things which will give a fair idea of the movement of the project and ensure the completeness of the test plan created in the planning phase, which will further help in enhancing the right testing strategy created earlier. We will start creating test case formats and test cases itself. In this stage we need to develop Functional validation matrix based on Business Requirements to ensure that all system requirements are covered by one or more test cases, identify which test cases to automate, begin review of documentation, i.e. Functional Design, Business Requirements, Product Specifications, Product Externals etc. We also have to define areas for Stress and Performance testing.

Test Design

Test plans and cases which were developed in the analysis phase are revised. Functional validation matrix is also revised and finalized. In this stage risk assessment criteria is developed. If you have thought of automation then you have to select which test cases to automate and begin writing scripts for them. Test data is prepared. Standards for unit testing and pass / fail criteria are defined here. Schedule for testing is revised (if necessary) & finalized and test environment is prepared.

Construction and verification

In this phase we have to complete all the test plans, test cases, complete the scripting of the automated test cases, Stress and Performance testing plans needs to be completed. We have to support the development team in their unit testing phase. And obviously bug reporting would be done as when the bugs are found. Integration tests are performed and errors (if any) are reported.

Testing Cycles

In this phase we have to complete testing cycles until test cases are executed without errors or a predefined condition is reached. Run test cases --> Report Bugs --> revise test cases (if needed) --> add new test cases (if needed) --> bug fixing --> retesting (test cycle 2, test cycle 3….).

Final Testing and Implementation In this we have to execute remaining stress and performance test cases, documentation for testing is completed / updated, provide and complete different matrices for testing. Acceptance, load and recovery testing will also be conducted and the application needs to be verified under production conditions.

Post Implementation

In this phase, the testing process is evaluated and lessons learnt from that testing process are documented. Line of attack to prevent similar problems in future projects is identified. Create plans to improve the processes. The recording of new errors and enhancements is an ongoing process. Cleaning up of test environment is done and test machines are restored to base lines in this stage.

Test case/data

A set of test inputs, execution conditions, and expected results developed for a particular objective, such as to exercise a particular program path or to verify compliance with a specific requirement.

A test case is a pair consisting of test data to be input to the program and the expected output. The test data is a set of values, one for each input variable.

A test set is a collection of zero or more test cases.

A test case in software engineering is a set of conditions or variables under which a tester will determine whether an application or software system is working correctly or not. The mechanism for determining whether a software program or system has passed or failed such a test is known as a test oracle. In some settings, an oracle could be a requirement or use case, while in others it could be a heuristic. It may take many test cases to determine that a software program or system is functioning correctly. Test cases are often referred to as test scripts, particularly when written. Written test cases are usually collected into test suites.

Typical written test case format

A test case is usually a single step, or occasionally a sequence of steps, to test the correct behavior/ functionalities, features of an application. An expected result or expected outcome is usually given. Additional information that may be included:

test case ID

test case description

test step or order of execution number

related requirement(s)

depth

test category

author

Additional fields that may be included and completed when the tests are executed:

pass/fail

remarks/comments

These steps can be stored in a word processor document, spreadsheet, database or other common repository.

In a database system, you may also be able to see past test results and who generated the results and the system configuration used to generate those results. These past results would usually be stored in a separate table.

Test suites may also contain

Test summary

Configuration

Besides a description of the functionality to be tested, and the preparation required to ensure that the test can be conducted, the most time consuming part in the test case is creating the tests and modifying them when the system changes.

Under special circumstances, there could be a need to run the test, produce results, and then a team of experts would evaluate if the results can be considered as a pass. This happens often on new products' performance number determination. The first test is taken as the base line for subsequent test / product release cycles.

Acceptance tests, which use a variation of a written test case, are commonly performed by a group of end-users or clients of the system to ensure the developed system meets the requirements specified or the contract. User acceptance tests are differentiated by the inclusion of happy path or positive test cases to the almost complete exclusion of negative test cases. The Sample test cases are discussed below:

Sample test case for sort:

Test data: <"A'' 12 -29 32 > Expected output: -29 12 32

Test Case 2

Test data: <'‘D'' 12 -29 32. > Expected output: 32 12 -29

Test Case 3

Test data: <"A'' .> Expected output: No input to be sorted in ascending order

Test Case 4

Test data: <'‘D''. > Expected output: No input to be sorted in descending order

Test Case 5

Test data: <'‘R'' 12 -29 32. > Expected output: Invalid request character; valid characters ‘A’ and ‘D’

Test Case 6

Test data: <'‘D'' c,17,2 . > Expected output: Invalid number

Lab No. 1: Making a test case for Google login screen: Verify login with valid username and password

Project Name:

Test Case Template example

Test Case ID: Fun_10

Test Priority (Low/Medium/High): Med

Module Name: Google login screen

Test Title: Verify login with valid username and password

Description: Test the Google login page

Test Designed by: <Name>

Test Designed date: <Date>

Test Executed by: <Name>

Test Execution date: <Date>

Pre-conditions: User has valid username and password
Dependencies:

Step	Test Steps	Test Data	Expected Result	Actual Result	Status (Pass/Fail)
1	Navigate to login page	User= example@gmail.com	User should be able to login	User is navigated to	Pass
2	Provide valid username	Password: 1234		dashboard with successful	
3	Provide valid password			login	
4	Click on Login button				

Post-conditions:
User is validated with database and successfully login to account. The account session details are logged in database.

Test case ID: Unique ID for each test case. Follow some convention to indicate types of test. E.g. 'TC_UI_1′ indicating 'user interface test case #1′.

Test priority (Low/Medium/High): This is useful while test execution. Test priority for business rules and functional test cases can be medium or higher whereas minor user interface cases can be low priority. Test priority should be set by reviewer.

Module Name – Mention name of main module or sub module.

Test Designed By: Name of tester

Test Designed Date: Date when wrote

Test Executed By: Name of tester who executed this test. To be filled after test execution.

Test Execution Date: Date when test executed.

Test Title/Name: Test case title. E.g. verify login page with valid username and password.

Test Summary/Description: Describe test objective in brief.

Pre-condition: Any prerequisite that must be fulfilled before execution of this test case. List all pre-conditions in order to successfully execute this test case.

Dependencies: Mention any dependencies on other test cases or test requirement.

Test Steps: List all test execution steps in detail. Write test steps in the order in which these should be executed. Make sure to provide as much details as you can. Tip – to efficiently manage test case with lesser number of fields use this field to describe test conditions, test data and user roles for running test.

Test Data: Use of test data as an input for this test case. You can provide different data sets with exact values to be used as an input.

Expected Result: What should be the system output after test execution? Describe the expected result in detail including message/error that should be displayed on screen.

Post-condition: What should be the state of the system after executing this test case?

Actual result: Actual test result should be filled after test execution. Describe system behavior after test execution.

Status (Pass/Fail): If actual result is not as per the expected result mark this test as failed. Otherwise update as passed.

Notes/Comments/Questions: To support above fields if there are some special conditions which can't be described in any of the above fields or there are questions related to expected or actual results mention those here.

Add following fields if necessary:

Defect ID/Link: If test status is fail, then include the link to defect log or mention the defect number.

Test Type/Keywords: This field can be used to classify tests based on test types. E.g. functional, usability, business rules etc.

Requirements: Requirements for which this test case is being written. Preferably the exact section number of the requirement doc.

Attachments/References: This field is useful for complex test scenarios. To explain test steps or expected result using a visio diagram as a reference. Provide the link or location to the actual path of the diagram or document.

Automation? (Yes/No): Whether this test case is automated or not. Useful to track automation status when test cases are automated.

Lab No. 2a: Write a C program to perform addition, subtraction, multiplication and division. Test it using a template.(test for division by zero)

C program to perform addition, subtraction, multiplication and division

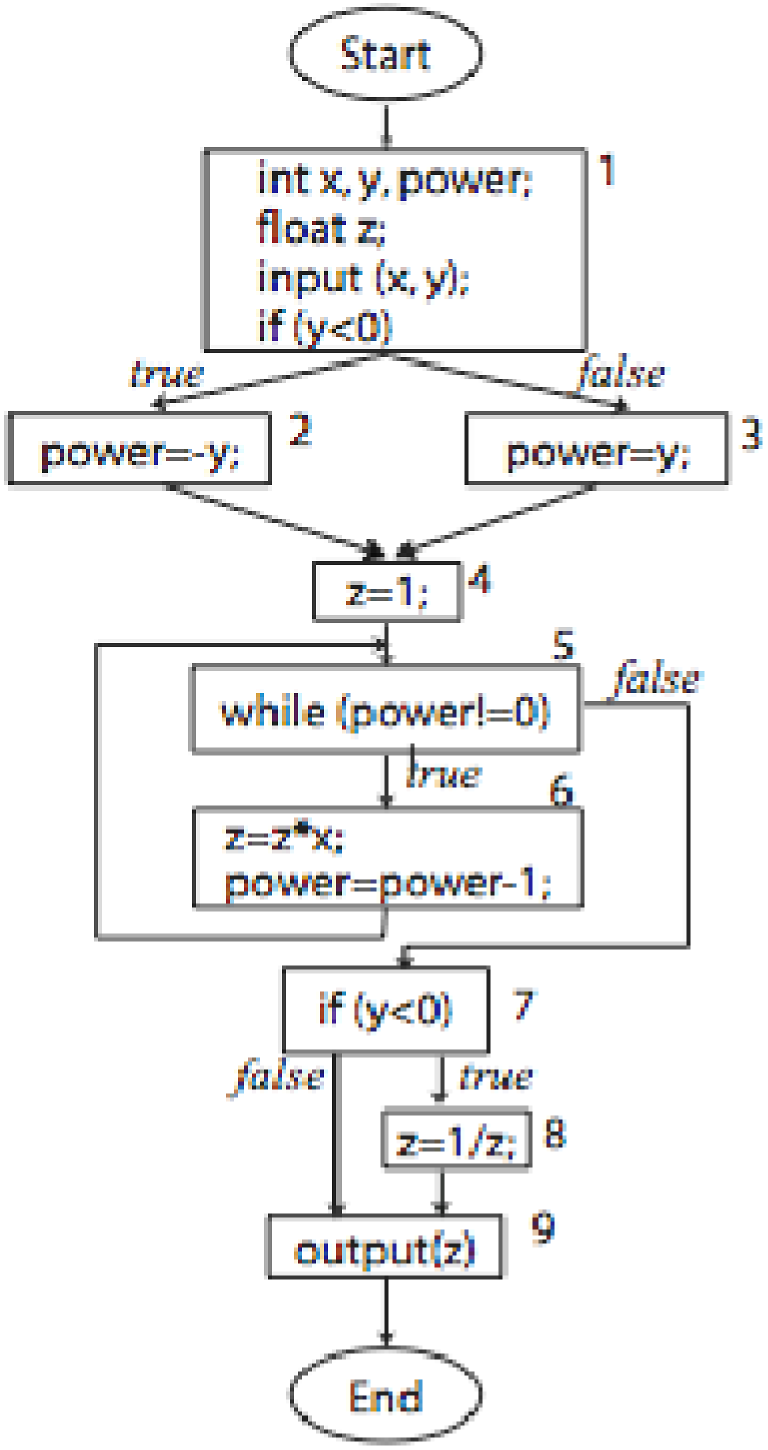
```c
#include <stdio.h>

int main()
{
   int first, second, add, subtract, multiply;
   float divide;

   printf("Enter two integers\n");
   scanf("%d%d", &first, &second);

   add        = first + second;
   subtract = first - second;
   multiply = first * second;
   divide     = first / (float)second;   //typecasting

   printf("Sum = %d\n",add);
   printf("Difference = %d\n",subtract);
   printf("Multiplication = %d\n",multiply);
   printf("Division = %.2f\n",divide);

   return 0;
}
```

Lab No. 2b. Write a C program to find odd or even. Test it using a template.

C program to check odd or even using modulus operator

```
#include<stdio.h>

main()
{
   int n;

   printf("Enter an integer\n");
   scanf("%d",&n);

   if ( n%2 == 0 )
      printf("Even\n");
   else
      printf("Odd\n");

   return 0;
}
```

Lab No. 3. Write a C program for the flow chart given below. Test it using a template.



Lab No. 4: Decision Table Approach for Solving Triangle Problem. Write a C program and test cases

Theory : Cause –Effect Graphs

In CEG analysis, first, identify ***causes***1, ***effects***2 and ***constraints***3 in the (natural language) specification. Second, construct a CEG as a combinational logic network which consists of nodes, called *causes* and *effects, arcs* with Boolean operators (*and, or,not*) between causes and effects, and *constraints*. Finally, trace this graph to build a decision table which may be subsequently converted into use cases and, eventually, test cases.

Focuses on modeling dependency relationships among

• Program input conditions (causes), and

• Output conditions (effects)

Allows selecting only few relevant test cases and thereby helping us to overcome the problem of too many test cases. Cause-Effect Graphing (CEG) is a model used to help identify productive test cases by using a simplified digital-logic circuit (combinatorial logic network) graph. It's origin is in hardware engineering but it has been adapted for use in software engineering. The CEG technique is a Black-Box method, (i.e. it considers the external behavior of a system with respect to how that system has been specified). It takes into consideration the combinations of causes that result in effecting the system's behavior.

Cause-Effect Graphing (CEG) is used to derive test cases from a given natural language specification to validate its corresponding implementation. The CEG technique is a blackbox method, i.e, it considers only the desired external behavior of a system Cause-Effect Graphing technique derives the minimum number of test cases to cover 100% of the functional requirements to improve the quality of test coverage.

A cause-effect graph is "a graphical representation of inputs or stimuli (causes) with their associated outputs (effects), which can be used to design test cases".

Furthermore, cause-effect graphs contain directed arcs that represent logical relationships
between causes and effects.

Each arc can be influenced by Boolean operators.

Such graphs can be used to design test cases, which can directly be derived from the graph , or to visualize and measure the completeness and the clearness of a test model for the tester.

Cause-Effect Graphing is very similar to Decision Table-Based Testing, where logical relationships of the inputs produce outputs; this is shown in the form of a graph.

The CEG technique is a Black-Box method, (i.e. it considers the external behavior of a system with respect to how that system has been specified).

The cause-effect graph is then converted into a decision table or "truth table" representing the logical relationships between the causes and effects.

Each column of the decision table is a test case. Each test case corresponds to a unique possible combination of inputs that are either in a true state, a false state, or a masked state.

Since there are 2 inputs to this example, there are 2 ** 2 = 4 combinations of inputs from which test cases can be selected.

Explores combinations of input conditions

Consists of 2 parts: Condition section and Action section

- Condition Section - Lists conditions and their combinations
- Action Section - Lists responses to be produced

Exposes errors in specification

Columns in decision table are converted to test cases

Similar to Condition Coverage used in White Box Testing

The Decision Table

A decision table is **a structured exercise to formulate requirements when dealing with complex business rules**. Decision tables are used to model complicated logic. They can make it easy to see that all possible combinations of conditions have been considered and when conditions are missed, it is easy to see this.

Let's take an example scenario for an ATM where a decision table would be of use.

A customer requests a cash withdrawal. One of the business rules for the ATM is that the ATM machine pays out the amount if the customer has sufficient funds in their account or if the customer has the credit granted. Already, this simple example of a business rule is quite complicated to describe in text. A decision table makes the same requirements clearer to understand:

Conditions	**R1**	**R2**	**R3**
Withdrawal Amount <= Balance	T	F	F
Credit granted	-	T	F
Actions			
Withdrawal granted	T	T	F

In a decision table, **conditions are usually expressed as true (T) or false (F)**. Each column in the table corresponds to a rule in the business logic that describes the unique combination of circumstances that will result in the actions. The table above contains three different business rules, and one of them is the "withdrawal is granted if the requested amount is covered by the balance." It is normal to create at least one test case per column, which results in full coverage of all business rules.

One advantage of using decision tables is that they **make it possible to detect combinations of conditions that would otherwise not have been found** and therefore not tested or developed. The requirements become much clearer and you often realize that some requirements are illogical, something that is hard to see when the requirements are only expressed in text.

A disadvantage of the technique is that **a decision table is not equivalent to complete test cases** containing step-by-step instructions of what to do in what order. When this level of detail is required, the decision table has to be further detailed into test cases.

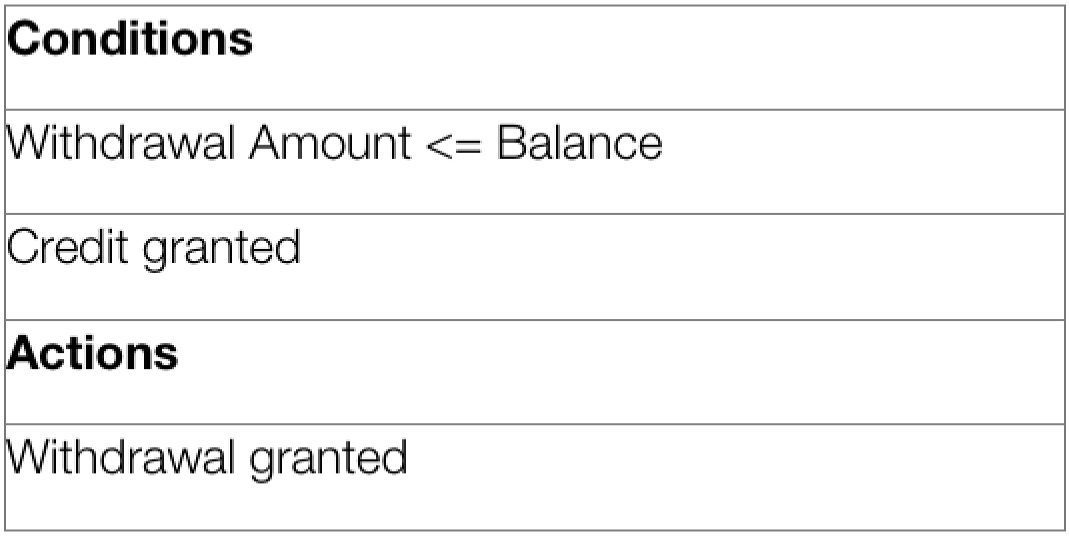
Decision tables can be used in all situations where the outcome depends on the combinations of different choices, and that is usually very often. In many systems there are tons of business rules where decision tables add a lot of value.

Decision tables should best be constructed during system design, since they become useful to both developers and testers. The requirements specialist also becomes more confident that everything that is important is actually documented. If there are no decision tables, testers can create them during test design to be able to write better test cases.

Step 1 – Analyze the requirement and create the first column

Requirement: "Withdrawal is granted if requested amount is covered by the balance or if the customer is granted credit to cover the withdrawal amount".

Express conditions and resulting actions in a list so that they are either TRUE or FALSE. In this case there are two conditions, “withdrawal amount ≤ balance” and “credit granted”. There is one result, the withdrawal is granted.



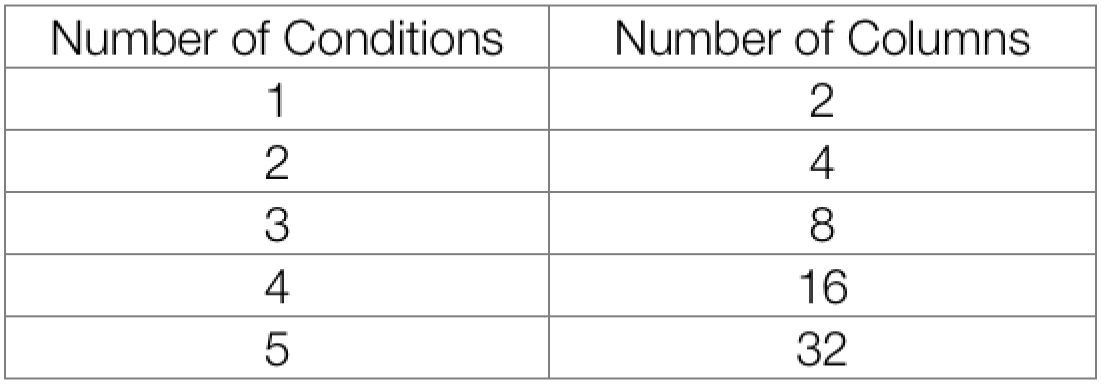
Conditions
Withdrawal Amount <= Balance
Credit granted
Actions
Withdrawal granted

Step 2: Add Columns

Calculate how many columns are needed in the table. The number of columns depends on the number of conditions and the number of alternatives for each condition. If there are two conditions and each condition can be either true or false, you need 4 columns. If there are three conditions there will be 8 columns and so on.

Mathematically, the number of columns is $2^{\text{conditions}}$. In this case $2^2 = 4$ columns.

Number of columns that is needed:



Number of Conditions	Number of Columns
1	2
2	4
3	8
4	16
5	32

The bottom line is that you should **create more smaller decision tables instead of fewer larger ones**, otherwise you run the risk of the decision tables being so large as to be unmanageable. Test the technique by picking areas with fewer business rules.

Now is the time to fill in the T (TRUE) and F (FALSE) for the conditions. How do you do that? The simplest is to say that it should look like this:

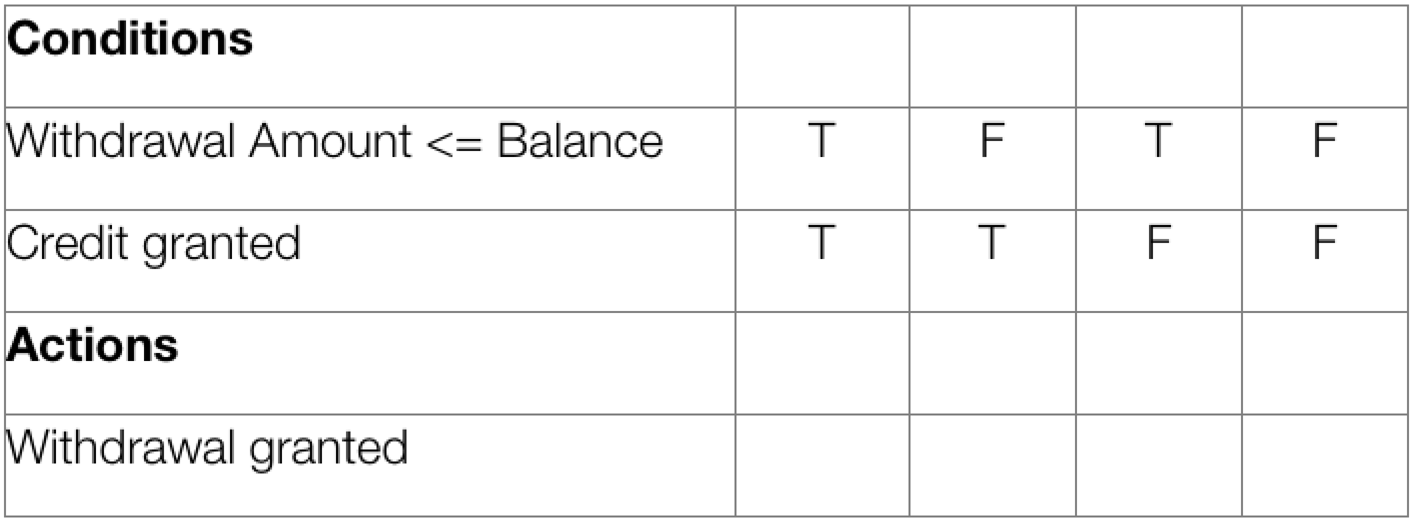
Row 1: TF

Row 2: TTFF

Row 3: TTTTFFFF

For each row, there is twice as many T and F as the previous line.

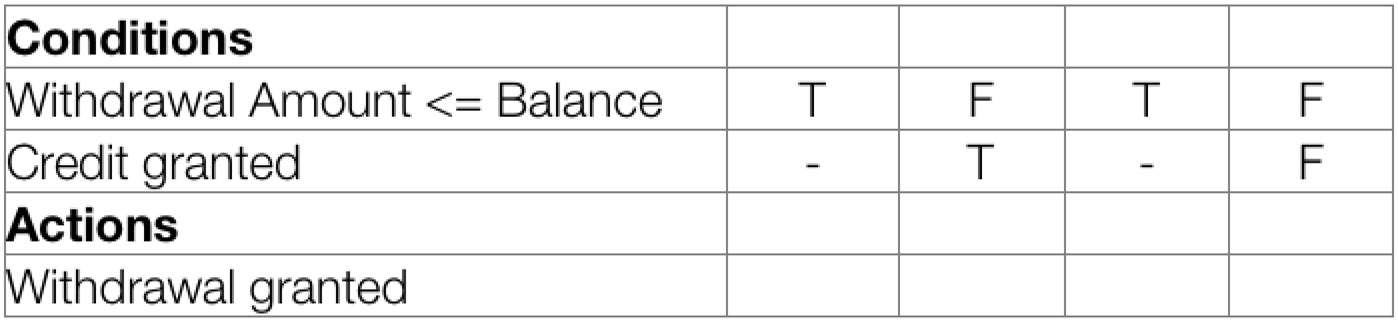
Repeat the pattern above from left to right for the entire row. In other words, for a table with 8 columns, the first row will read TFTFTFTF, the second row will read TTFFTTFF and the third row will read TTTTFFFF.



Conditions				
Withdrawal Amount <= Balance	T	F	T	F
Credit granted	T	T	F	F
Actions				
Withdrawal granted				

Step 3: Reduce the table

Mark insignificant values with “-”. If the requested amount is less than or equal to the account balance it does not matter if credit is granted. In the next step, you can delete the columns that have become identical.



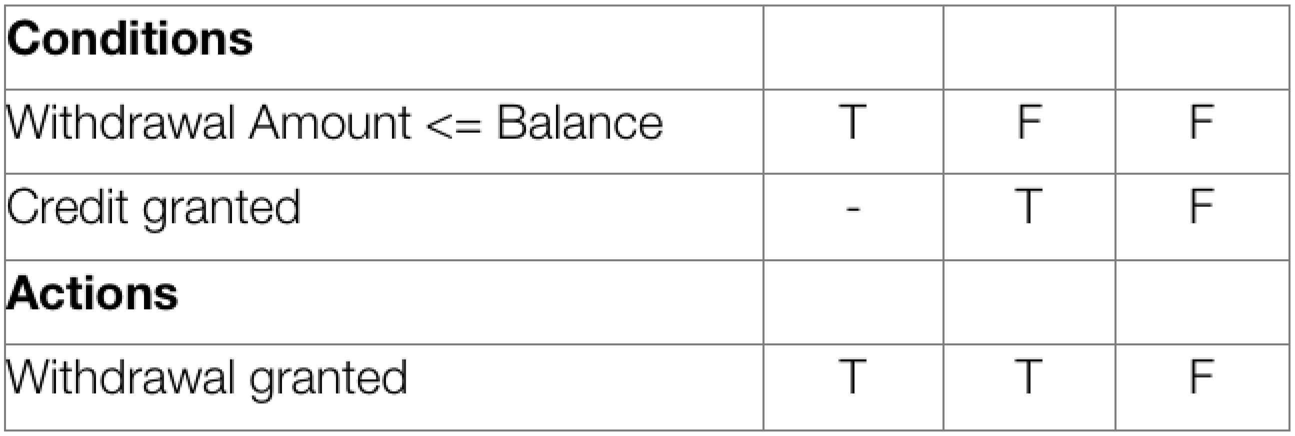
Conditions				
Withdrawal Amount <= Balance	T	F	T	F
Credit granted	-	T	-	F
Actions				
Withdrawal granted				

Check for invalid combinations. Invalid combinations are those that cannot happen, for example, that someone is both an infant and senior. Mark them somehow, e.g. with “X”. In this example, there are no invalid combinations.

Finish by removing duplicate columns. In this case, the first and third column are equal, therefore one of them is removed.

Step 4: Determine actions

Enter actions for each column in the table. You will be able to find this information in the requirement. Name the columns (the rules). They may be named R1/Rule 1, R2/Rule 2 and so on, but you can also give them more descriptive names.



Conditions			
Withdrawal Amount <= Balance	T	F	F
Credit granted	-	T	F
Actions			
Withdrawal granted	T	T	F

Step 5: Write test cases

Write test cases based on the table. At least one test case per column gives full coverage of all business rules.

- Test case for R1: balance = 200, requested withdrawal = 200. Expected result: withdrawal granted.
- Test case for R2: balance = 100, requested withdrawal = 200, credit granted. Expected result: withdrawal granted.
- Test case for R3: balance = 100, requested withdrawal = 200, no credit. Expected Result: withdrawal denied.

Lab No. 4: Decision Table Approach for Solving Triangle Problem

```
/* Design and develop a program in a language of your choice to solve the triangle
problem defined as  follows  : Accept  three  integers which are supposed  to be  the  three
sides of triangle and determine if the three values represent an equilateral triangle,
isosceles triangle, scalene triangle, or they do not form a triangle at all. Derive test cases
for your program based on decision-table approach, execute  the  test cases and discuss
the results */
#include<stdio.h>
int main()
{
  int a,b,c;
  char istriangle;
  printf("enter 3 integers which are sides of triangle\n");
  scanf("%d%d%d",&a,&b,&c);
  printf("a=%d\t,b=%d\t,c=%d",a,b,c);

  // to check is it a triangle or not

  if( a<b+c && b<a+c && c<a+b )
    istriangle='y';
  else
    istriangle ='n';
  ;
  if (istriangle=='y')
    if ((a==b) && (b==c))
      printf("equilateral triangle\n");
    else if ((a!=b) && (a!=c) && (b!=c))
      printf("scalene triangle\n");
       else
      printf("isosceles triangle\n");
  else
    printf("Not a triangle\n");
  return 0;
}
```

Test Case Name :Decision table for triangle problem
Lab No : 4
Test Data : Enter the 3 Integer Value(a , b And c)
Pre-condition : a < b + c , b < a + c and c < a + b
Brief Description : Check whether given value for a equilateral, isosceles , Scalene triangle or can't form a triangle

Input data decision Table

RULES		R1	R2	R3	R4	R5	R6	R7	R8	R9	R10	R11
Conditions	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
Actions	a1 : Not a triangle	X	X	X								
	a2 : Scalene triangle											X
	a3 : Isosceles triangle							X		X	X	
	a4 : Equilateral triangle				X							
	a5 : Impossible					X	X		X			

Triangle Problem -Decision Table Test cases for input data

Case Id	Description	Input Data			Expected Output	Actual Output	Status	Comments
		a	b	c				
1	Enter the value of a, b and c Such that a is not less than sum of two sides	20	5	5	Message should be displayed can't form a triangle			
2	Enter the value of a, b and c Such that b is not less than sum of two sides and a is less than sum of other two sides	3	15	11	Message should be displayed can't form a triangle			
3	Enter the value of a, b and c Such that c is not less than sum of two sides and a and b is less than sum of other two sides	4	5	20	Message should be displayed can't form a triangle			
4	Enter the value a, b and c satisfying precondition and a=b, b=c and c=a	5	5	5	Should display the message Equilateral triangle			
5	Enter the value a ,b and c satisfying precondition and a=b and b ≠ c	10	10	9	Should display the message Isosceles triangle			
6	Enter the value a, b and c satisfying precondition and a ≠b , b ≠ c and c ≠ a	5	6	7	Should display the message Scalene triangle			

Lab No. 5: (Boundary Value Analysis) Using Boundary value Analysis write test cases.

Theory: Boundary Value Analysis

Boundary value analysis is software testing design technique in which tests are designed to include representatives of boundary values. Values on the edge of an equivalence partition or at the smallest value on either side of an edge are taken for testing. The values could be either input or output ranges of a software component. Since these boundaries are common locations for errors that result in software faults they are frequently exercised in test cases.

A Black Box Testing Method, complements to Equivalence partition and BVA leads to a selection of test cases that exercise bounding values.

The expected input and output values should be extracted from the component specification. The input and output values to the software component are then grouped into sets with identifiable boundaries. Each set, or partition, contains values that are expected to be processed by the component in the same way.

For an example, if the input values were months of the year expressed as integers, the input parameter 'month' might have the following partitions:

```
       ... -2 -1  0  1 ..................12 13  14  15 .....
     ---------------|-----------------|---------------------
 invalid partition 1   valid partition     invalid partition 2
```

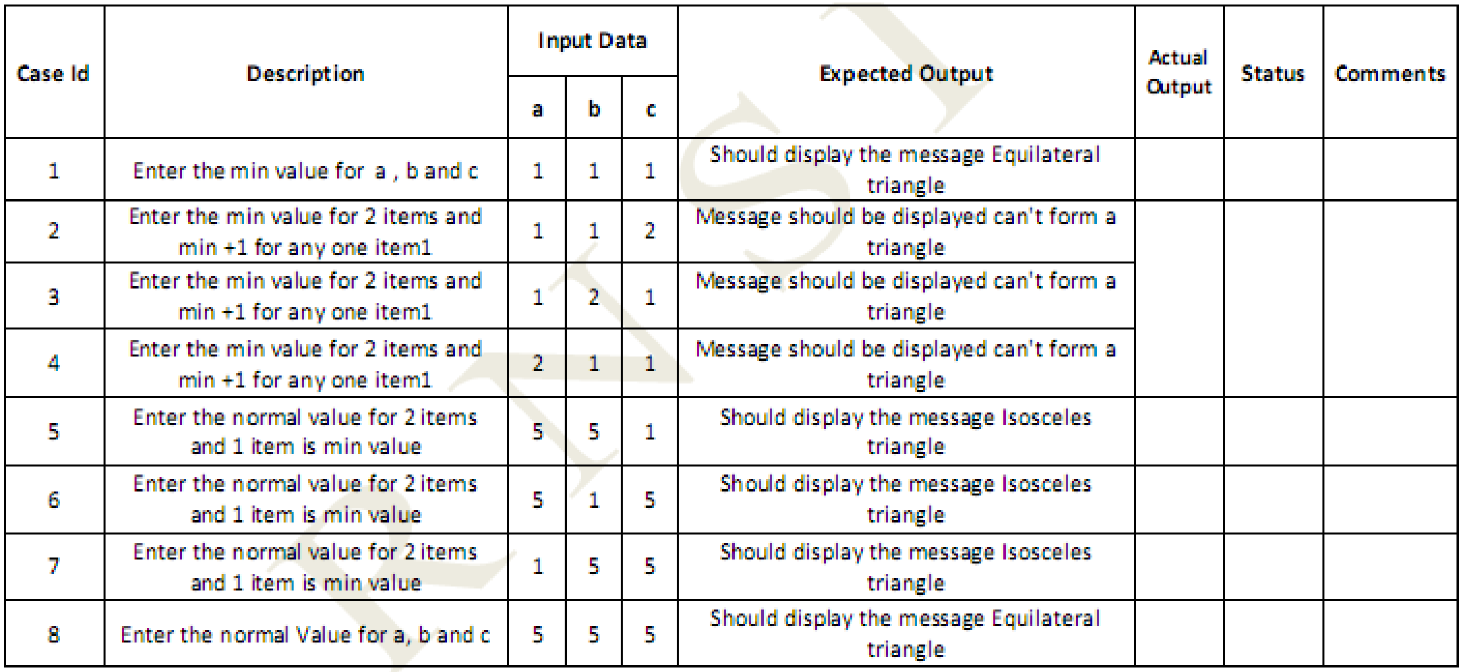
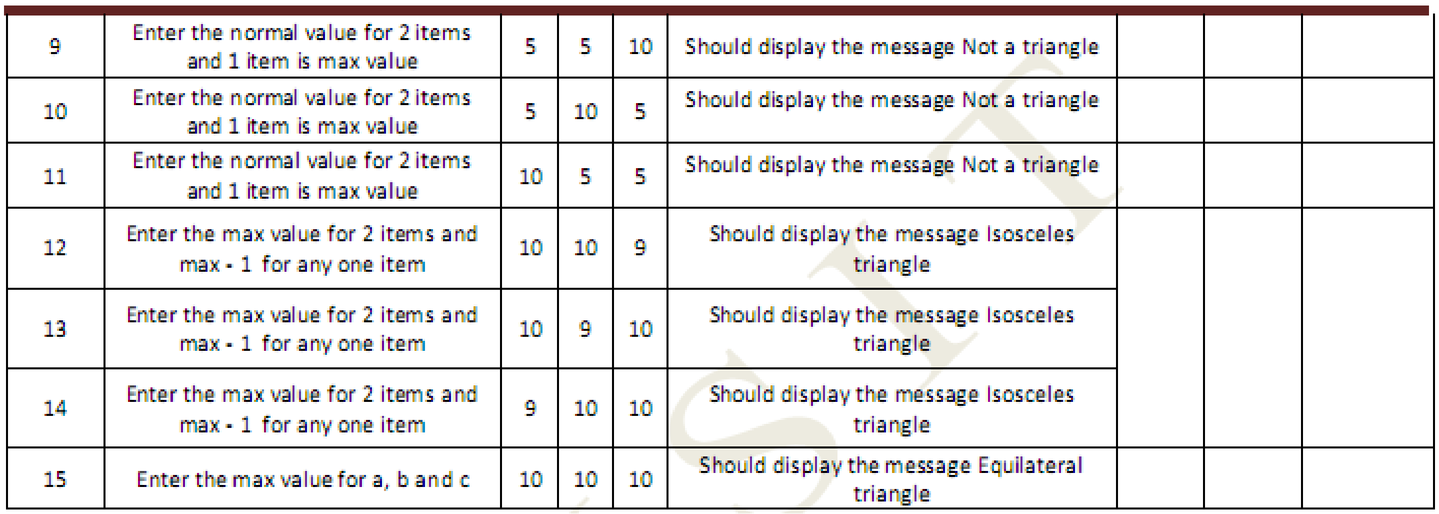
The boundaries are the values on and around the beginning and end of a partition. If possible test cases should be created to generate inputs or outputs that will fall on and to either side of each boundary. This would result in three cases per boundary. The test cases on each side of a boundary should be in the smallest increment possible for the component under test. In the example above there are boundary values at 0,1,2 and 11,12,13.

Lab No. 5: (Boundary Value Analysis Program)

```
/* Design and develop a program in a language of your choice to solve the triangle problem
defined as follows : Accept three integers which are supposed to be the three sides of
triangle and determine if the three values represent an equilateral triangle, isosceles
triangle, scalene triangle, or they do not form a triangle at all. Derive test cases for your
program based on boundary value analysis, execute the test cases and discuss the results */

#include<stdio.h>
int main()
{
  int a,b,c,c1,c2,c3;
  char istriangle;
  do
  {
      printf("\nenter 3 integers which are sides of triangle\n");
       scanf("%d%d%d",&a,&b,&c);
       printf("\na=%d\tb=%d\tc=%d",a,b,c);
      c1 = a>=1 &&  a<=10;
              c2= b>=1 &&  b<=10;
       c3= c>=1 &&  c<=10;
      if (!c1)
                  printf("\nthe value of a=%d is not the range of permitted value",a);
       if (!c2)
            printf("\nthe value of b=%d is not the range of permitted value",b);
       if (!c3)
            printf("\nthe value of c=%d is not the range of permitted value",c);
   } while(!(c1 && c2 && c3));

  // to check is it a triangle or not

  if( a<b+c && b<a+c && c<a+b )
    istriangle='y';
  else
    istriangle ='n';
  if (istriangle=='y')
    if ((a==b) && (b==c))
      printf("equilateral triangle\n");
    else if ((a!=b) && (a!=c) && (b!=c))
      printf("scalene triangle\n");
       else
      printf("isosceles triangle\n");
  else
    printf("Not a triangle\n");
  return 0;
}
```

Test Case Name :Boundary Value Analysis for triangle problem
Lab No. 5
Test Data : Enter the 3 Integer Value(a , b And c)
Pre-condition : $1 \leq a \leq 10$, $1 \leq b \leq 10$ and $1 \leq c \leq 10$ and $a < b + c$, $b < a + c$ and $c < a + b$
Brief Description : Check whether given value for a Equilateral, Isosceles , Scalene triangle or can't form a triangle
Triangle Problem -Boundary value Test cases for input data

Triangle Problem -Boundary value Test cases for input data

Case Id	Description	Input Data			Expected Output	Actual Output	Status	Comments
		a	b	c				
1	Enter the min value for a , b and c	1	1	1	Should display the message Equilateral triangle			
2	Enter the min value for 2 items and min +1 for any one item1	1	1	2	Message should be displayed can't form a triangle			
3	Enter the min value for 2 items and min +1 for any one item1	1	2	1	Message should be displayed can't form a triangle			
4	Enter the min value for 2 items and min +1 for any one item1	2	1	1	Message should be displayed can't form a triangle			
5	Enter the normal value for 2 items and 1 item is min value	5	5	1	Should display the message Isosceles triangle			
6	Enter the normal value for 2 items and 1 item is min value	5	1	5	Should display the message Isosceles triangle			
7	Enter the normal value for 2 items and 1 item is min value	1	5	5	Should display the message Isosceles triangle			
8	Enter the normal Value for a, b and c	5	5	5	Should display the message Equilateral triangle			
9	Enter the normal value for 2 items and 1 item is max value	5	5	10	Should display the message Not a triangle			
10	Enter the normal value for 2 items and 1 item is max value	5	10	5	Should display the message Not a triangle			
11	Enter the normal value for 2 items and 1 item is max value	10	5	5	Should display the message Not a triangle			
12	Enter the max value for 2 items and max - 1 for any one item	10	10	9	Should display the message Isosceles triangle			
13	Enter the max value for 2 items and max - 1 for any one item	10	9	10	Should display the message Isosceles triangle			
14	Enter the max value for 2 items and max - 1 for any one item	9	10	10	Should display the message Isosceles triangle			
15	Enter the max value for a, b and c	10	10	10	Should display the message Equilateral triangle			

Lab No. 6: (Equivalence Class Partitioning Analysis) Using Equivalence Partitioning Analysis write test cases.

Theory: Equivalence Partitioning:

Equivalence partitioning is a software testing technique that divides the input data of a software unit into partition of data from which test cases can be derived. In principle, test cases are designed to cover each partition at least once. This technique tries to define test case that uncovers classes of errors, thereby reducing the total number of test cases that must be developed.

In rare cases equivalence partitioning is also applied to outputs of a software component, typically it is applied to the inputs of a tested component. The equivalence partitions are usually derived from the requirements specification for input attributes that influence the processing of the test object. An input has certain ranges which are valid and other ranges which are invalid. Invalid data here does not mean that the data is incorrect; it means that this data lies outside of specific partition. This may be best explained by the example of a function which takes a parameter "month". The valid range for the month is 1 to 12, representing January to December. This valid range is called a partition. In this example there are two further partitions of invalid ranges. The first invalid partition would be <= 0 and the second invalid partition would be >= 13.

Equivalence Partitioning

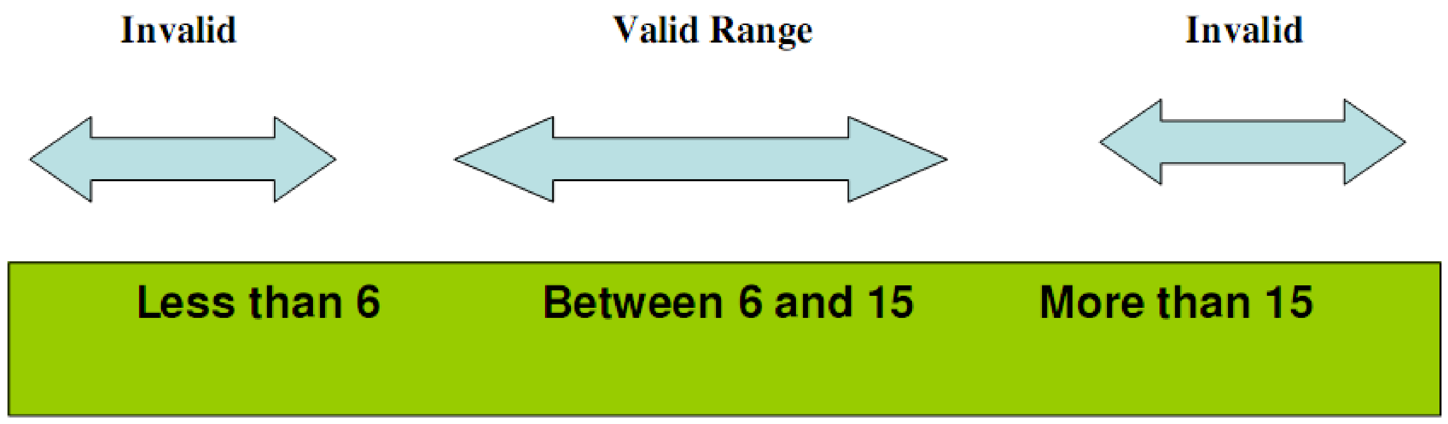
Divide the input domain into classes of data for which test cases can be generated.

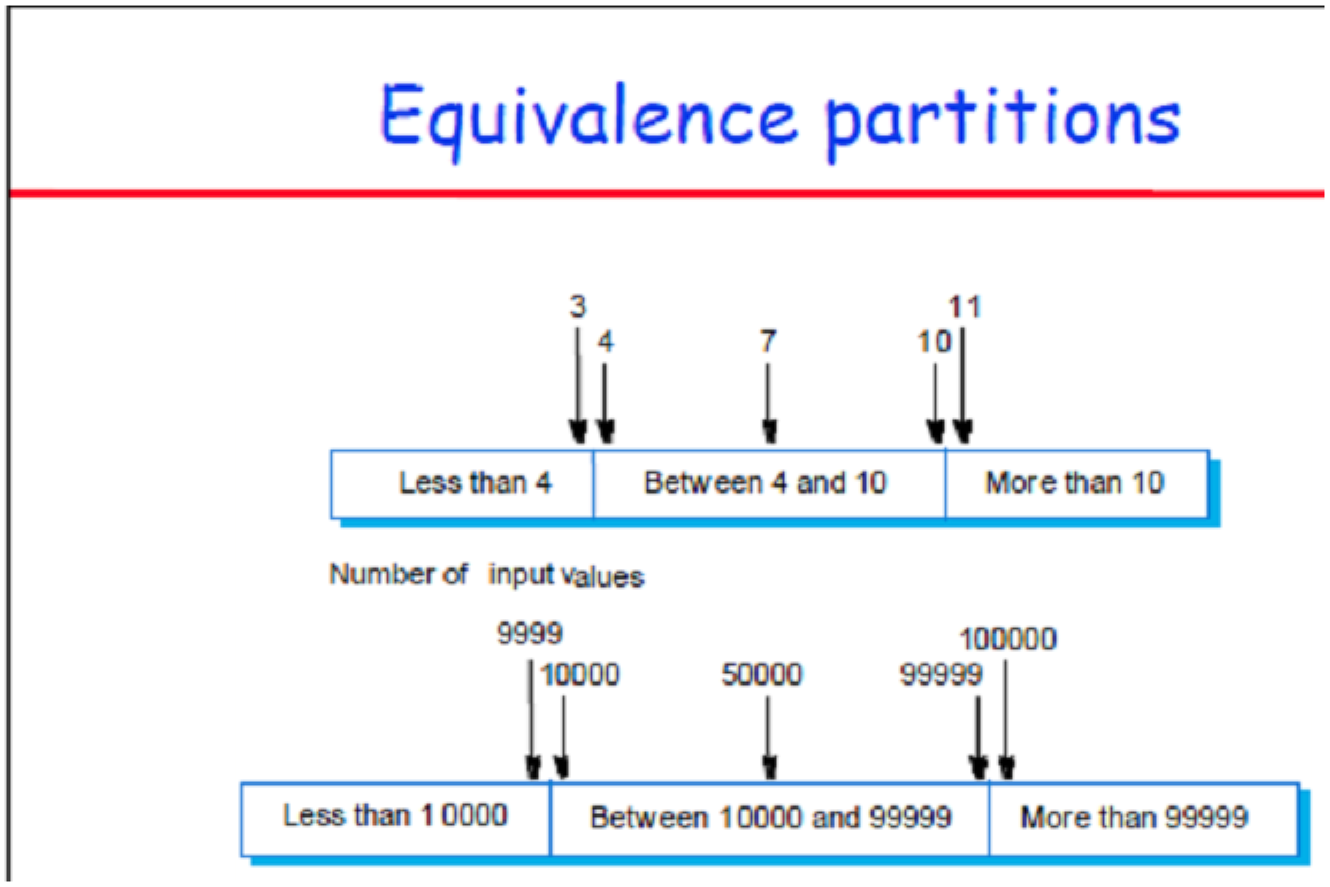
Attempting to uncover classes of errors.

Derives test cases based on these partitions

An equivalence class is a set of valid or invalid states of input

Test case design is based on equivalence classes for an input domain.





Test Case Name :Equivalence class Analysis for triangle problem
Lab No. 6
Test Data : Enter the 3 Integer Value(a , b And c)
Pre-condition : $1 \leq a \leq 10$, $1 \leq b \leq 10$ and $1 \leq c \leq 10$ and $a < b + c$, $b < a + c$ and $c < a + b$
Brief Description : Check whether given value for a Equilateral, Isosceles , Scalene triangle or can't form a triangle

Triangle Problem - Equivalence Class Test cases for input data

Case Id	Description	Input Data			Expected Output	Actual Output	Status	Comments
		a	b	C				
1	Enter the min value for a , b and c	5	5	5	Should display the message Equilateral triangle			
2	Enter the min value for a , b and c	2	2	3	Should display the message Isosceles triangle			
3	Enter the min value for a , b and c	3	4	5	Should display the message Scalene triangle			
4	Enter the min value for a , b and c	4	1	2	Message should be displayed can't form a triangle			

5	Enter one invalid input and two valid value for a , b and c	-1	5	5	Should display value of a is not in the range of permitted values			
6	Enter one invalid input and two valid value for a , b and c	5	-1	5	Should display value of a is not in the range of permitted values			
7	Enter one invalid input and two valid value for a , b and c	5	5	-1	Should display value of a is not in the range of permitted values			
8	Enter one invalid input and two valid value for a , b and c	11	5	5	Should display value of a is not in the range of permitted values			
9	Enter one invalid input and two valid value for a , b and c	5	11	5	Should display value of a is not in the range of permitted values			
10	Enter one invalid input and two valid value for a , b and c	5	5	11	Should display value of a is not in the range of permitted values			
11	Enter one invalid input and two valid value for a , b and c	-1	5	5	Should display value of a is not in the range of permitted values			
12	Enter one invalid input and two valid value for a , b and c	5	-1	5	Should display value of a is not in the range of permitted values			
13	Enter one invalid input and two valid value for a , b and c	5	5	-1	Should display value of a is not in the range of permitted values			
14	Enter two invalid input and two valid value for a , b and c	-1	-1	5	Should display value of a is not in the range of permitted values			
					Should display value of b is not in the range of permitted values			
14	Enter two invalid input and two valid value for a , b and c	5	-1	-1	Should display value of b is not in the range of permitted values			
					Should display value of c is not in the range of permitted values			
14	Enter two invalid input and two valid value for a , b and c	-1	5	-1	Should display value of a is not in the range of permitted values			
					Should display value of c is not in the range of permitted values			
15	Enter all invalid inputs	-1	-1	-1	Should display value of a is not in the range of permitted values			
					Should display value of b is not in the range of permitted values			
					Should display value of c is not in the range of permitted values			

Lab No. 7 (Boundary Value Analysis Test Case for Commission Problem)

- Rifle sales company produces
  - locks Rs. 45
  - stocks Rs. 30
  - barrels Rs. 25
- Salesmen send sales reports via telegraph; commission is 10% on sales up to Rs.1000, 15% on the next Rs. 800, 20% on everything above
- Program produces monthly sales reports and commission to be paid

/* Design, develop, code and run the program in any suitable language to solve the commission problem. Analyze it from the perspective of boundary value, derive test cases, execute these test cases and discuss the test results */

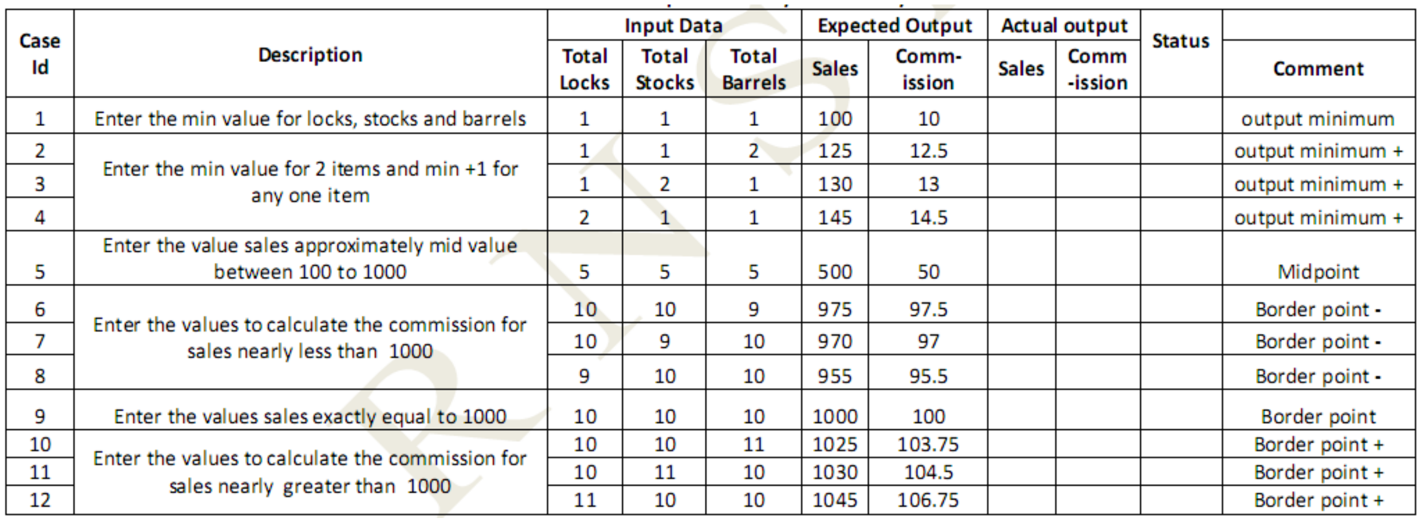
/* Assumption price for lock=45.0, stock=30.0 and barrels=25.0 production limit could sell in a month 70 locks,80 stocks and 90 barrels commission on sales = 10 % <= 1000 and 15 % on 1000 to 1800 and 20 % on above 1800*/

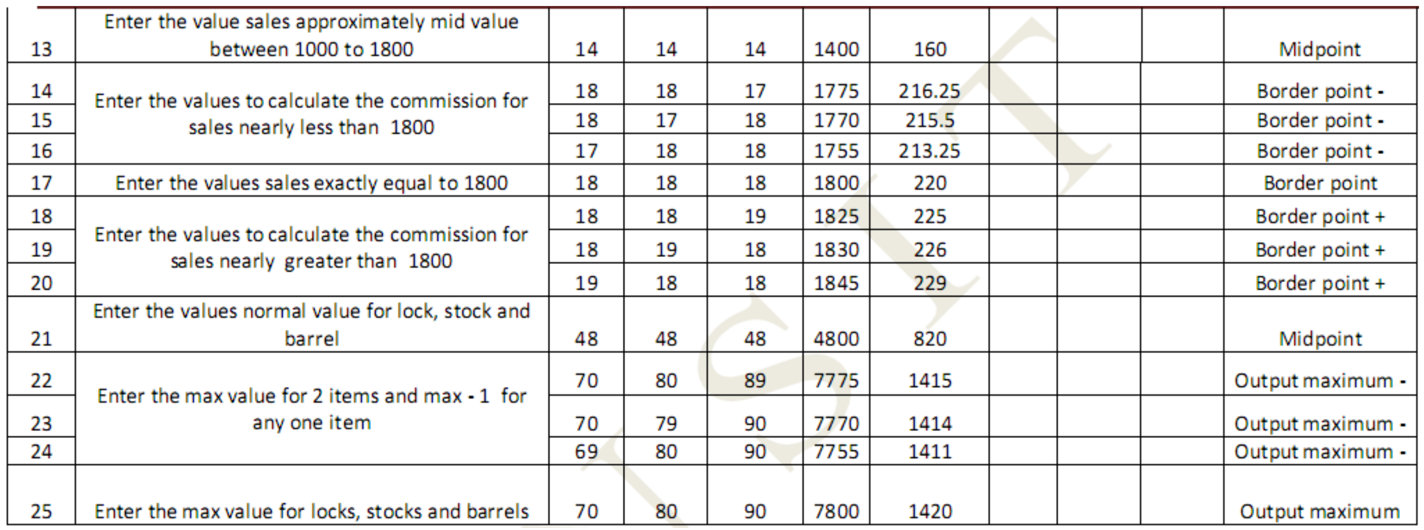
```
#include<stdio.h>
int main()
{
  int locks, stocks, barrels, tlocks, tstocks, tbarrels;
  float lprice, sprice, bprice, sales, comm;
int c1,c2,c3,temp;
  lprice=45.0;
  sprice=30.0;
  bprice=25.0;
  tlocks=0;
  tstocks=0;
  tbarrels=0;
  printf("\nenter the number of locks and to exit the loop enter -1 for locks\n");
  scanf("%d",&locks);
  while(locks!=-1)
  {
    c1=(locks<=0||locks>70);
    printf("enter the number of stocks and barrels\n");
    scanf("%d%d",&stocks,&barrels);
    c2=(stocks<=0||stocks>80);
    c3=(barrels<=0||barrels>90);
    if(c1)
      printf("value of locks not in the range 1..70  ");
    else
      {
      temp=tlocks+locks;
      if(temp>70)
```

```
        printf("new total locks =%d not in the range 1..70 so old ",temp);
      else
tlocks=temp;
        }
    printf("total locks = %d\n",tlocks);

if(c2)
      printf("value of stocks not in the range 1..80  ");
    else
      {

      temp=tstocks+stocks;
      if(temp>80)
      printf("new total stocks =%d  not in the range 1..80 so old   ",temp);
      else
      tstocks=temp;
      }
    printf("total stocks=%d\n",tstocks);

    if(c3)
      printf("value of barrels not in the range 1..90  ");
    else
      {
      temp=tbarrels+barrels;
      if(temp>90)
      printf("new total barrels =%d not in the range 1..90 so old ",temp);
      else
      tbarrels=temp;
       }
    printf("total barrel=%d",tbarrels);
    printf("\nenter the number of locks and to exit the loop enter -1 for locks\n");
    scanf("%d",&locks);
   }
  printf("\ntotal locks = %d\ntotal stocks =%d\ntotal barrels =%d\n",tlocks,tstocks,tbarrels);
  sales = lprice*tlocks+sprice*tstocks+bprice*tbarrels;
printf("\nthe total sales=%f\n",sales);

  if(sales > 0)
  {
    if(sales > 1800.0)
    {
      comm=0.10*1000.0;
      comm=comm+0.15*800;
      comm=comm+0.20*(sales-1800.0);
    }
    else if(sales > 1000)
    {
      comm =0.10*1000;
      comm=comm+0.15*(sales-1000);
    }
    else
      comm=0.10*sales;

    printf("the commission is=%f\n",comm);
  }
  else
    printf("there is no sales\n");
  return 0;
}
```

Test Case Name : Boundary Value for Commission Problem
Lab Number : 7
Test data : price Rs for lock - 45.0 , stock - 30.0 and barrel - 25.0
sales = total lock * lock price + total stock * stock price + total barrel * barrel price
commission : 10% up to sales Rs 1000 , 15 % of the next Rs 800 and 20 % on any sales in excess of 1800
Pre-condition : lock = -1 to exit and 1< =lock < = 70 , 1<=stock <=80 and 1<=barrel<=90
Brief Description : The salesperson had to sell at least one complete rifle per month.

CHECKING BOUNDARY VALUE FOR LOCKS, STOCKS AND BARRELS AND COMMISSION
Commission Problem Output Boundary Value Analysis Cases





Case Id	Description	Input Data			Expected Output		Actual output		Status	Comment
		Total Locks	Total Stocks	Total Barrels	Sales	Comm-ission	Sales	Comm -ission		
1	Enter the min value for locks, stocks and barrels	1	1	1	100	10				output minimum
2	Enter the min value for 2 items and min +1 for any one item	1	1	2	125	12.5				output minimum +
3		1	2	1	130	13				output minimum +
4		2	1	1	145	14.5				output minimum +
5	Enter the value sales approximately mid value between 100 to 1000	5	5	5	500	50				Midpoint
6	Enter the values to calculate the commission for sales nearly less than 1000	10	10	9	975	97.5				Border point -
7		10	9	10	970	97				Border point -
8		9	10	10	955	95.5				Border point -
9	Enter the values sales exactly equal to 1000	10	10	10	1000	100				Border point
10	Enter the values to calculate the commission for sales nearly greater than 1000	10	10	11	1025	103.75				Border point +
11		10	11	10	1030	104.5				Border point +
12		11	10	10	1045	106.75				Border point +
13	Enter the value sales approximately mid value between 1000 to 1800	14	14	14	1400	160				Midpoint
14	Enter the values to calculate the commission for sales nearly less than 1800	18	18	17	1775	216.25				Border point -
15		18	17	18	1770	215.5				Border point -
16		17	18	18	1755	213.25				Border point -
17	Enter the values sales exactly equal to 1800	18	18	18	1800	220				Border point
18	Enter the values to calculate the commission for sales nearly greater than 1800	18	18	19	1825	225				Border point +
19		18	19	18	1830	226				Border point +
20		19	18	18	1845	229				Border point +
21	Enter the values normal value for lock, stock and barrel	48	48	48	4800	820				Midpoint
22	Enter the max value for 2 items and max - 1 for any one item	70	80	89	7775	1415				Output maximum -
23		70	79	90	7770	1414				Output maximum -
24		69	80	90	7755	1411				Output maximum -
25	Enter the max value for locks, stocks and barrels	70	80	90	7800	1420				Output maximum

Output Special Value Test Cases

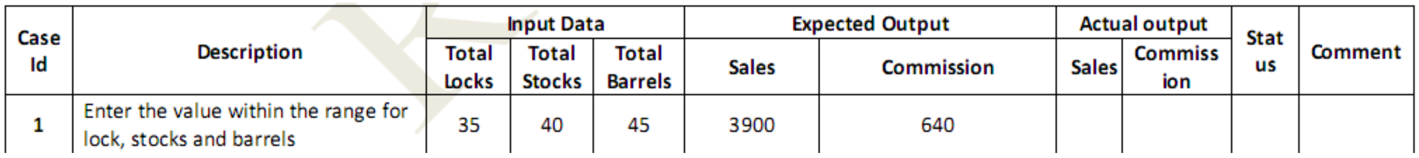
Case Id	Description	Input Data			Expected Output		Actual output		Status	Comment
		Total Locks	Total Stocks	Total Barrels	Sales	Commission	Sales	Commission		
1	Enter the random values such that to calculate commission for sales nearly less than 1000	11	10	8	995	99.5				Border point -
2	Enter the random values such that to calculate commission for sales nearly greater than 1000	10	11	9	1005	100.75				Border point +
3	Enter the random values such that to calculate commission for sales nearly less than 1800	18	17	19	1795	219.25				Border point -
4	Enter the random values such that to calculate commission for sales nearly greater than 1800	18	19	17	1805	221				Border point +

Lab No. 8 (Equivalence Class Partioning Analysis Test Case for Commission Problem)

Test Case Name :Equivalence Class for Commission Problem
Experiment Number : 8
Test data : price Rs for lock - 45.0 , stock - 30.0 and barrel - 25.0
sales = total lock * lock price + total stock * stock price + total barrel * barrel price
commission : 10% up to sales Rs 1000 , 15 % of the next Rs 800 and 20 % on any sales in excess of 1800
Pre-condition : lock = -1 to exit and 1< =lock < = 70 , 1<=stock <=80 and 1<=barrel<=90
Brief Description : The salesperson had to sell at least one complete rifle per month.
Checking boundary value for locks, stocks and barrels and commission

Valid Classes
L1 ={LOCKS :1 <=LOCKS<=70}
L2 ={Locks=-1}(occurs if locks=-1 is used to control input iteration)
L3 ={stocks : 1<=stocks<=80}
L4= {barrels :1<=barrels<=90}
Invalid Classes
L3 ={locks: locks=0 OR locks<-1}
L4 ={locks: locks> 70}
S2 ={stocks : stocks<1}
S3 ={stocks : stocks >80}
B2 ={barrels : barrels <1}
B3 =barrels : barrels >90}

Commission Problem Output Equivalence Class Testing
(Weak & Strong Normal Equivalence Class)



Case Id	Description	Input Data			Expected Output		Actual output		Status	Comment
		Total Locks	Total Stocks	Total Barrels	Sales	Commission	Sales	Commission		
1	Enter the value within the range for lock, stocks and barrels	35	40	45	3900	640				

Weak Robustness Equivalence Class

Case Id	Description	Input Data			Expected Output	Actual output	Status	Comment
		Locks	Stocks	Barrels				
WR1	Enter the value locks = -1	-1	40	45	Terminates the input loop and proceed to calculate sales and commission (if Sales > 0)			
WR2	Enter the value less than -1 or equal to **zero** for locks and other valid inputs	0	40	45	Value of Locks not in the range 1..70			
WR3	Enter the value greater than 70 for locks and other valid inputs	71	40	45	Value of Locks not in the range 1..70			
WR4	Enter the value less than or equal than 0 for stocks and other valid inputs	35	0	45	Value of stocks not in the range 1..80			
WR5	Enter the value greater than 80 for stocks and other valid inputs	35	81	45	Value of stocks not in the range 1..80			
WR6	Enter the value less than or equal 0 for barrels and other valid inputs	35	40	0	Value of Barrels not in the range 1..90			
WR7	Enter the value greater than 90 for barrels and other valid inputs	35	40	91	Value of Barrels not in the range 1..90			

Strong Robustness Equivalence Class

Case Id	Description	Input Data			Expected Output	Actual output	Status	Comment
		Locks	Stocks	Barrels				
SR1	Enter the value less than -1 for locks and other valid inputs	-2	40	45	Value of Locks not in the range 1..70			
SR2	Enter the value less than or equal than 0 for stocks and other valid inputs	35	-1	45	Value of stocks not in the range 1..80			
SR3	Enter the value less than or equal 0 for barrels and other valid inputs	35	40	-2	Value of Barrels not in the range 1..90			
SR4	Enter the locks and stocks less than or equal to 0 and other valid inputs	-2	-1	45	Value of Locks not in the range 1..70 Value of stocks not in the range 1..80			
SR5	Enter the locks and barrel less than or equal to 0 and other valid inputs	-2	40	-1	Value of Locks not in the range 1..70 Value of Barrels not in the range 1..90			
SR6	Enter the stocks and barrel less than or equal to 0 and other valid inputs	35	-1	-1	Value of stocks not in the range 1..80 Value of Barrels not in the range 1..90			
SR7	Enter the stocks and barrel less than or equal to 0 and other valid inputs	-2	-2	-2	Value of Locks not in the range 1..70 Value of stocks not in the range 1..80 Value of Barrels not in the range 1..90			

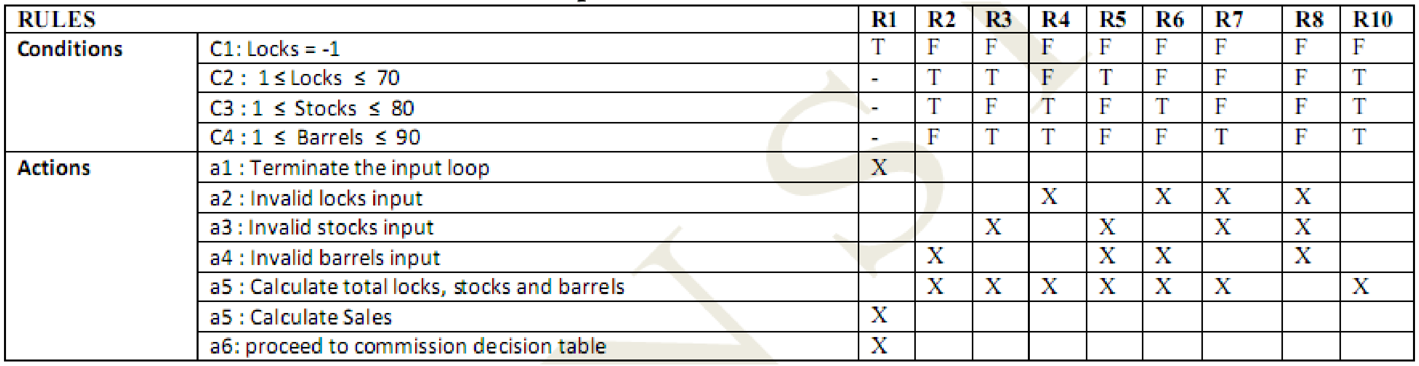
Some addition equivalence Boundary checking

Case Id	Description	Input Data			Expected Output		Actual output			
		Total Locks	Total Stocks	Total Barrels	Sales	Commission	Sales	Commission	Status	Comment
OR1	Enter the value for lock, stocks and barrels where 0 < Sales < 1000	5	5	5	500	50				
OR2	Enter the value for lock, stocks and barrels where 1000 < Sales < 1800	15	15	15	1500	175				
OR3	Enter the value for lock, stocks and barrels where Sales < 1800	25	25	25	2500	360				

Lab No. 9 (Decision Table Analysis Test Case for Commission Problem)

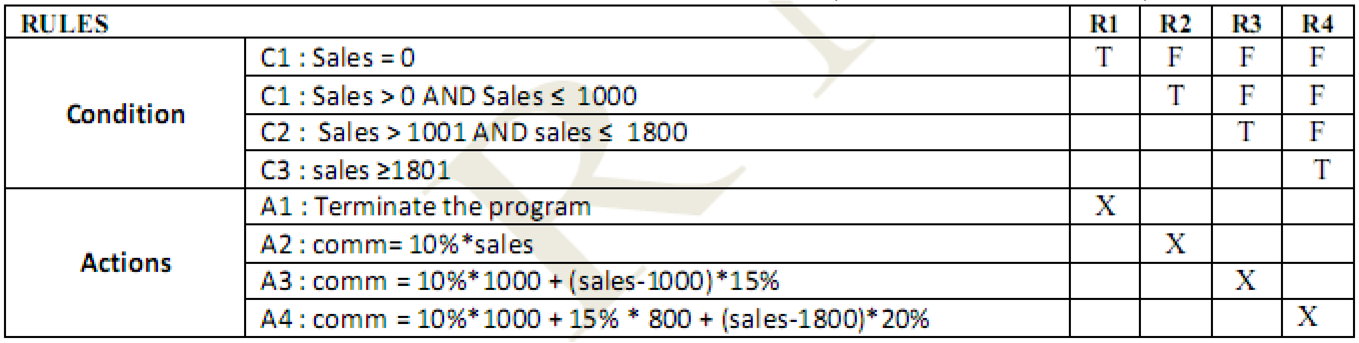
Experiment Number : 9
Test data : price Rs for lock - 45.0 , stock - 30.0 and barrel - 25.0
sales = total lock * lock price + total stock * stock price + total barrel * barrel price
commission : 10% up to sales Rs 1000 , 15 % of the next Rs 800 and 20 % on any sales in excess of 1800
Pre-condition : lock = -1 to exit and 1< =lock < = 70 , 1<=stock <=80 and 1<=barrel<=90
Brief Description : The salesperson had to sell at least one complete rifle per month.

Input data decision Table



RULES		R1	R2	R3	R4	R5	R6	R7	R8	R10
Conditions	C1: Locks = -1	T	F	F	F	F	F	F	F	F
	C2 : 1 ≤ Locks ≤ 70	-	T	T	F	T	F	F	F	T
	C3 : 1 ≤ Stocks ≤ 80	-	T	F	T	F	T	F	F	T
	C4 : 1 ≤ Barrels ≤ 90	-	F	T	T	F	F	T	F	T
Actions	a1 : Terminate the input loop	X								
	a2 : Invalid locks input				X		X	X	X	
	a3 : Invalid stocks input			X		X		X	X	
	a4 : Invalid barrels input		X			X	X		X	
	a5 : Calculate total locks, stocks and barrels		X	X	X	X	X	X		X
	a5 : Calculate Sales	X								
	a6: proceed to commission decision table	X								

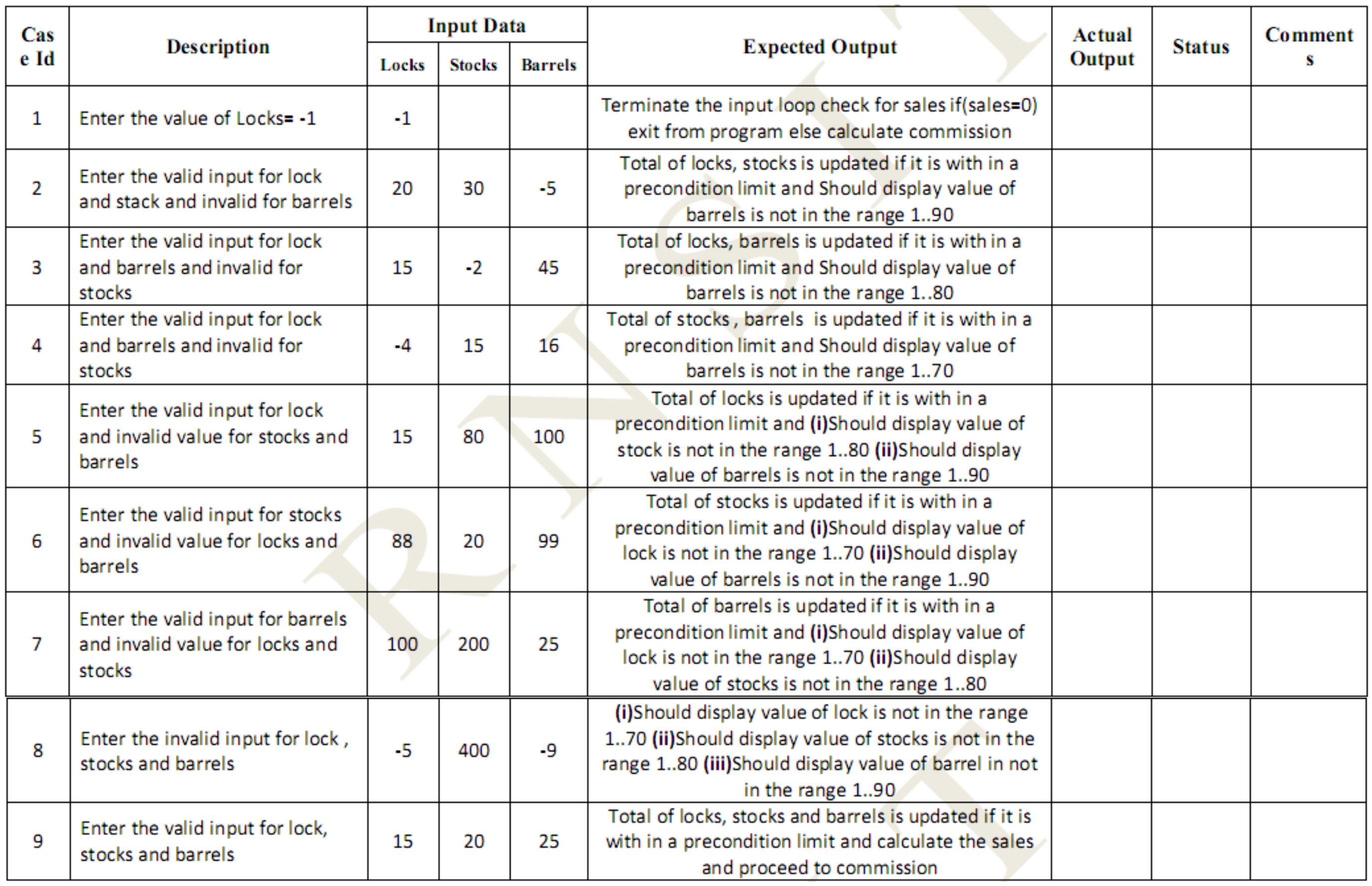
Commission calculation Decision Table (Precondition : lock = -1)



RULES		R1	R2	R3	R4
Condition	C1 : Sales = 0	T	F	F	F
	C1 : Sales > 0 AND Sales ≤ 1000		T	F	F
	C2 : Sales > 1001 AND sales ≤ 1800			T	F
	C3 : sales ≥1801				T
Actions	A1 : Terminate the program	X			
	A2 : comm= 10%*sales		X		
	A3 : comm = 10%*1000 + (sales-1000)*15%			X	
	A4 : comm = 10%*1000 + 15% * 800 + (sales-1800)*20%				X

Precondition : Initial Value Total Locks= 0 , Total Stocks=0 and Total Barrels=0
Precondition Limit :Total locks, stocks and barrels should not exceed the limit 70,80 and 90 respectively

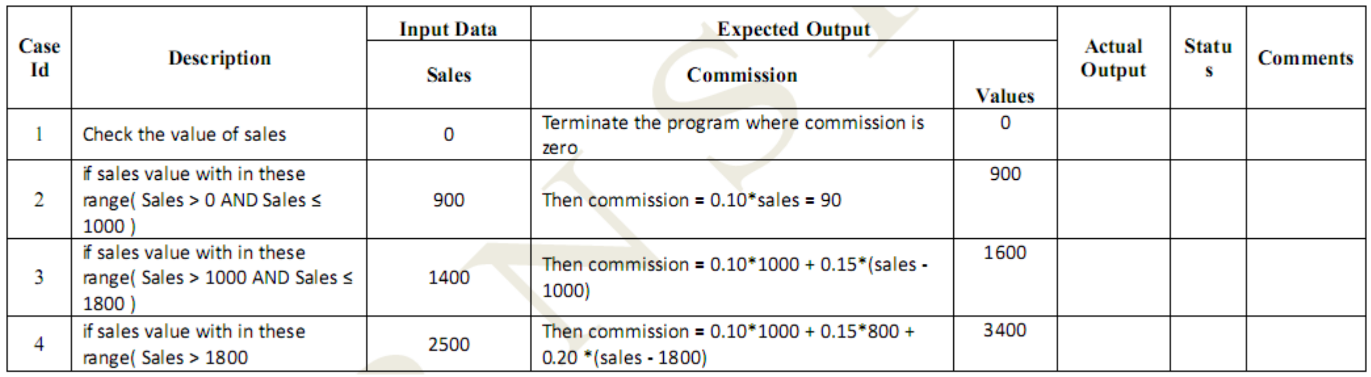
Commission Problem -Decision Table Test cases for input data



Case Id	Description	Input Data			Expected Output	Actual Output	Status	Comments
		Locks	Stocks	Barrels				
1	Enter the value of Locks= -1	-1			Terminate the input loop check for sales if(sales=0) exit from program else calculate commission			
2	Enter the valid input for lock and stack and invalid for barrels	20	30	-5	Total of locks, stocks is updated if it is with in a precondition limit and Should display value of barrels is not in the range 1..90			
3	Enter the valid input for lock and barrels and invalid for stocks	15	-2	45	Total of locks, barrels is updated if it is with in a precondition limit and Should display value of barrels is not in the range 1..80			
4	Enter the valid input for lock and barrels and invalid for stocks	-4	15	16	Total of stocks , barrels is updated if it is with in a precondition limit and Should display value of barrels is not in the range 1..70			
5	Enter the valid input for lock and invalid value for stocks and barrels	15	80	100	Total of locks is updated if it is with in a precondition limit and **(i)**Should display value of stock is not in the range 1..80 **(ii)**Should display value of barrels is not in the range 1..90			
6	Enter the valid input for stocks and invalid value for locks and barrels	88	20	99	Total of stocks is updated if it is with in a precondition limit and **(i)**Should display value of lock is not in the range 1..70 **(ii)**Should display value of barrels is not in the range 1..90			
7	Enter the valid input for barrels and invalid value for locks and stocks	100	200	25	Total of barrels is updated if it is with in a precondition limit and **(i)**Should display value of lock is not in the range 1..70 **(ii)**Should display value of stocks is not in the range 1..80			
8	Enter the invalid input for lock , stocks and barrels	-5	400	-9	**(i)**Should display value of lock is not in the range 1..70 **(ii)**Should display value of stocks is not in the range 1..80 **(iii)**Should display value of barrel in not in the range 1..90			
9	Enter the valid input for lock, stocks and barrels	15	20	25	Total of locks, stocks and barrels is updated if it is with in a precondition limit and calculate the sales and proceed to commission			

Commission Problem -Decision Table Test cases for commission calculation

Precondition : Locks = -1



Case Id	Description	Input Data	Expected Output		Actual Output	Status	Comments
		Sales	Commission	Values			
1	Check the value of sales	0	Terminate the program where commission is zero	0			
2	if sales value with in these range(Sales > 0 AND Sales ≤ 1000)	900	Then commission = 0.10*sales = 90	900			
3	if sales value with in these range(Sales > 1000 AND Sales ≤ 1800)	1400	Then commission = 0.10*1000 + 0.15*(sales - 1000)	1600			
4	if sales value with in these range(Sales > 1800	2500	Then commission = 0.10*1000 + 0.15*800 + 0.20 *(sales - 1800)	3400			

Lab No. 10 (Decision Table Analysis Test Case for Grade Problem)

A student may receive a final course grade of A, B, C, D, or F. In deriving the student's final course grade, the instructor first determines an initial or tentative grade for the student, which is determined in the following manner:

- **A student who has received a total of no lower than 90 percent on the first three assignments and exams and received a score no lower than 70 percent on the fourth assignment will receive an initial grade of A for the course. A student who has scored a total lower than 90 percent but no lower than 80 percent on the first three assignments and exams and received a score no lower 70 percent on the fourth assignment will receive an initial grade of B for the course. A student who has received a total lower than 80 percent but no lower than 70 percent on the first three assignments and exams and received a score no lower than 70 percent on the fourth assignment will receive an initial grade of C for the course. A student who has scored a total lower than 70 percent but no lower than 60 percent on the first three assignments and exams and received a score no lower 70 percent on the fourth assignment will receive an initial grade of D for the course. A student who has scored a total lower than 60 percent on the first three assignments and exams, or received a score lower than 70 percent on the fourth assignment, will receive an initial grade of F for the course. Once the instructor has determined the initial course grade for the student, the final course grade will be determined. The student's final course grade will be the same as his or her initial course grade if no more than three class periods during the semester were missed. Otherwise, the student's final course grade will be one letter grade lower than his or her initial course grade (for example, an A will become a B).**

Lab No. 11 (White Box Path testing for Binary Search problem)

/* Design, develop a code and run the program in any suitable language to implement the binary search algorithm. Determine the basis paths and using them derive different test cases execute these test cases and discuss the test results */

A binary search or half-interval search algorithm finds the position of a specified input value (the search "key") within an array sorted by key value. In each step, the algorithm compares the search key value with the key value of the middle element of the array. If the keys match, then a matching element has been found and its index, or position, is returned. Otherwise, if the search key is less than the middle element's key, then the algorithm repeats its action on the sub-array to the left of the middle element or, if the search key is greater, on the sub-array to the right. If the remaining array to be searched is empty, then the key cannot be found in the array and a special "not found" indication is returned.

Example: L = 1 3 4 6 8 9 11. X = 4.

```
Compare X to 6. It's smaller. Repeat with L = 1 3 4.
Compare X to 3. It's bigger. Repeat with L = 4.
Compare X to 4. It's equal. We're done, we found X.
```

This is called Binary Search: each iteration of (1)-(4) the length of the list we are looking in gets cut in half.

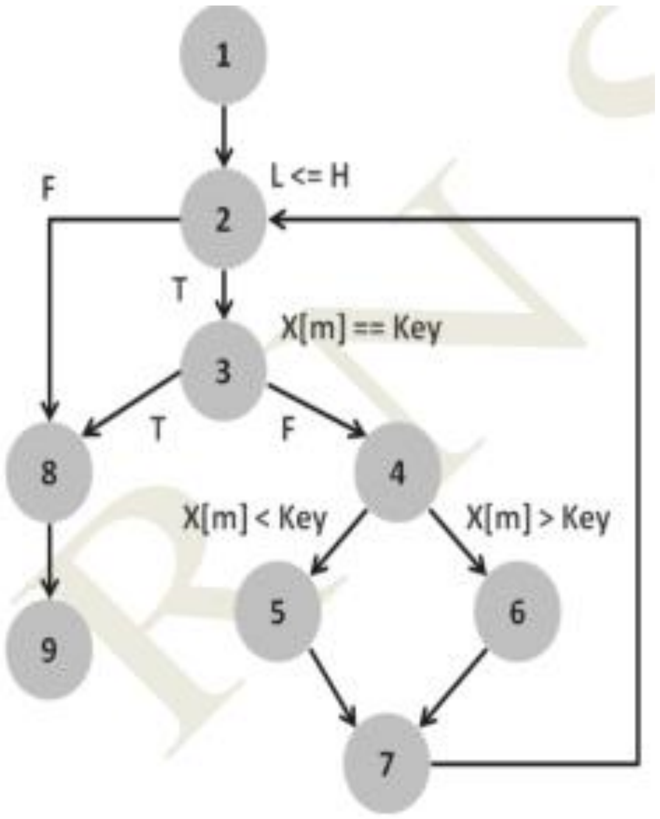
```
#include<stdio.h>
int binsrc(int x[],int low,int high,int key)
{
   int mid;
  while(low<=high)
  {
    mid=(low+high)/2;
    if(x[mid]==key)
      return mid;
    if(x[mid]<key)
      low=mid+1;
    else
      high=mid-1;
  }
  return -1;
}

int main()
{
  int a[20],key,i,n,succ;
  printf("Enter the n value");
```

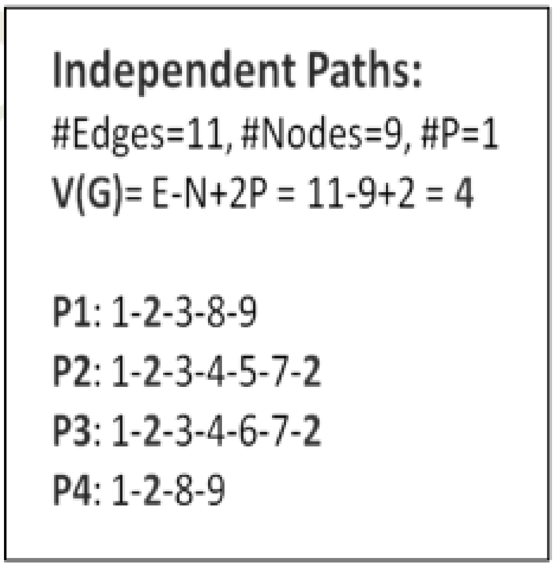
```
 scanf("%d",&n);
 if(n>0)
 {
        printf("enter the elements in ascending order\n");
   for(i=0;i<n;i++)
   scanf("%d",&a[i]);

        printf("enter the key element to be searched\n");
        scanf("%d",&key);
   succ=binsrc(a,0,n-1,key);
   if(succ>=0)
     printf("Element found in position = %d\n",succ+1);
   else
      printf("Element not found \n");
 }
 else
   printf("Number of element should be greater than zero\n");
    return 0;
}
```

Binary Search function with line number

```
int binsrc(int x[],int low, int high, int key)
{
   int mid;        1
  while(low<=high)      2
   {
   mid=(low+high)/2;
   if(x[mid]==key)    3
       return mid;    8
   if(x[mid]<key)    4
     low=mid+1;    5
     else
     high=mid-1;    6
  }         7
  return -1;        8
}          9
```

Program Graph – for Binary Search





Independent Paths:

#Edges=11, #Nodes=9, #P=1

V(G)= E-N+2P = 11-9+2 = 4

P1: 1-2-3-8-9

P2: 1-2-3-4-5-7-**2**

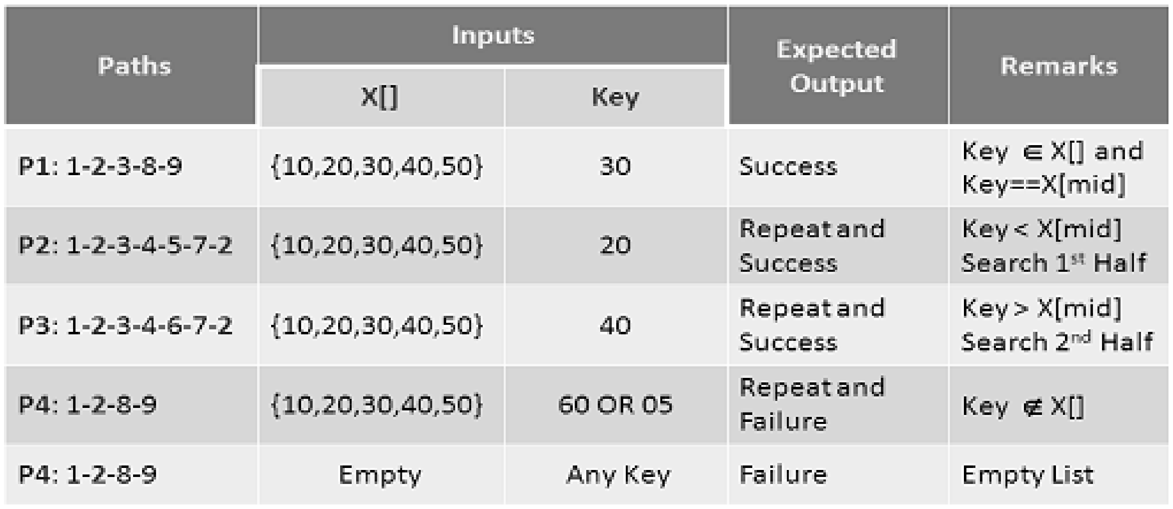
P3: 1-2-3-4-6-7-**2**

P4: 1-2-8-9

Pre-Condition

Array has Elements in Ascending order
Key element is in the Array

Test Cases – Binary Search



Paths	Inputs		Expected Output	Remarks
	X[]	Key		
P1: 1-2-3-8-9	{10,20,30,40,50}	30	Success	Key $\in$ X[] and Key==X[mid]
P2: 1-2-3-4-5-7-2	{10,20,30,40,50}	20	Repeat and Success	Key < X[mid] Search 1st Half
P3: 1-2-3-4-6-7-2	{10,20,30,40,50}	40	Repeat and Success	Key > X[mid] Search 2nd Half
P4: 1-2-8-9	{10,20,30,40,50}	60 OR 05	Repeat and Failure	Key $\notin$ X[]
P4: 1-2-8-9	Empty	Any Key	Failure	Empty List

Lab No. 12 (White Box Path testing for Quick Sort)

/*Design, develop, code and run the program in any suitable language to implement the Quick-Sort Algorithm. Determine the basis paths and using them derive different test cases, execute these test cases and discuss the test results.*/

Quicksort is a divide and conquer algorithm. Quicksort first divides a large list into two smaller sub-lists: the low elements and the high elements. Quicksort can then recursively sort the sub-lists.

The steps are:

1. Pick an element, called a **pivot**, from the list.
2. Reorder the list so that all elements with values less than the pivot come before the pivot, while all elements with values greater than the pivot come after it (equal values can go either way). After this partitioning, the pivot is in its final position. This is called the **partition** operation.
3. Recursively apply the above steps to the sub-list of elements with smaller values and separately the sub-list of elements with greater values.

```
#include<stdio.h>
void quicksort(int x[10],int first,int last)
{
    int temp,pivot,i,j;
  if(first<last)
  {
             pivot=first;
              i=first;
             j=last;
         while(i<j)
        {
              while(x[i]<=x[pivot] && i<last)
            i++;
                 while(x[j]>x[pivot])
        j--;
                if(i<j)
      {
                     temp=x[i];
                      x[i]=x[j];
                      x[j]=temp;
      }
               }
            temp=x[pivot];
        x[pivot]=x[j];
          x[j]=temp;
```

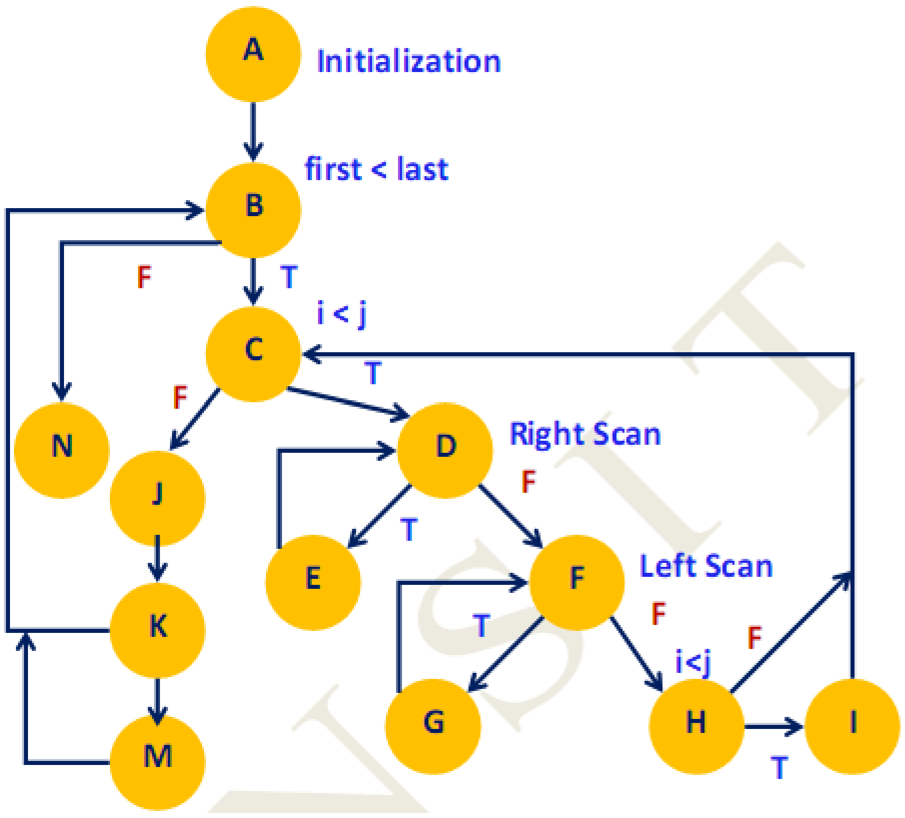
```
              quicksort(x,first,j-1);
               quicksort(x,j+1,last);
      }
}


// main program

int main()
{
    int a[20],i,key,n;
    printf("enter the size of the array");
  scanf("%d",&n);
  if(n>0)
  {
    printf("enter the elements of the array");
    for(i=0;i<n;i++)
      scanf("%d",&a[i]);

            quicksort(a,0,n-1);
    printf("the elements in the sorted array is:\n");
        for(i=0;i<n;i++)
            printf("%d\t",a[i]);
  }
  else
  {
    printf("size of array is invalid\n");
}
```

Quick sort function with line number

```
void quicksort(int x[10],int first,int last)
{
     int temp,pivot,i,j;            1
  if(first<last)              2
  {
               pivot=first;           3
             i=first;                4
               j=last;             5
         while(i<j)           6
        {
              while(x[i]<=x[pivot] && i<last)      7
              i++;          8
                       while(x[j]>x[pivot])       9
        j--;          10
```

```
                if(i<j)        11
      {
                    temp=x[i];       12
                     x[i]=x[j];       13
                     x[j]=temp;       14
      }
           }
            temp=x[pivot];       15
             x[pivot]=x[j];        16
           x[j]=temp;          17
            quicksort(x,first,j-1);         18
             quicksort(x,j+1,last);         19
      }
}               20
```

Program Graph – Quick Sort



Recursive Calls

Independent Paths– Quick Sort
P1: A-B-N
P2: A-B-C-J-K-B
P3: A-B-C-J-K-M-B

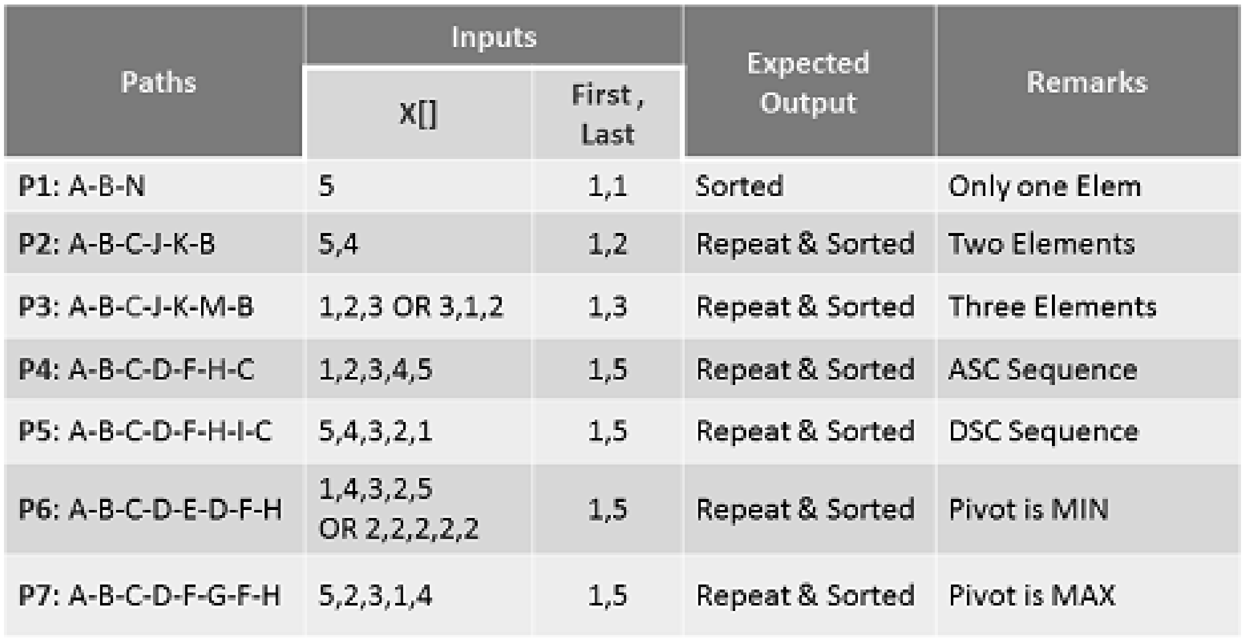
P4: A-B-C-D-F-H-C
P5: A-B-C-D-F-H-I-C
P6: A-B-C-D-E-D-F-H
P7: A-B-C-D-F-G-F-H

Independent Paths:
#Edges=18, #Nodes=13, #P=1
V(G)= E-N+2P = 18-13+2 = 7

Pre-Conditions/Issues:

Array has only one Element, Two Elements, Three Elements (6 Possibilities)
Array has Elements in ASC/DSC/Arbitrary(Any of the Permutations)
EX: 3 elements: 123, 132, 213, 231, 312, 321, 222,111,333

Test Cases – Quick Sort



Paths	Inputs		Expected Output	Remarks
	X[]	First , Last		
P1: A-B-N	5	1,1	Sorted	Only one Elem
P2: A-B-C-J-K-B	5,4	1,2	Repeat & Sorted	Two Elements
P3: A-B-C-J-K-M-B	1,2,3 OR 3,1,2	1,3	Repeat & Sorted	Three Elements
P4: A-B-C-D-F-H-C	1,2,3,4,5	1,5	Repeat & Sorted	ASC Sequence
P5: A-B-C-D-F-H-I-C	5,4,3,2,1	1,5	Repeat & Sorted	DSC Sequence
P6: A-B-C-D-E-D-F-H	1,4,3,2,5 OR 2,2,2,2,2	1,5	Repeat & Sorted	Pivot is MIN
P7: A-B-C-D-F-G-F-H	5,2,3,1,4	1,5	Repeat & Sorted	Pivot is MAX

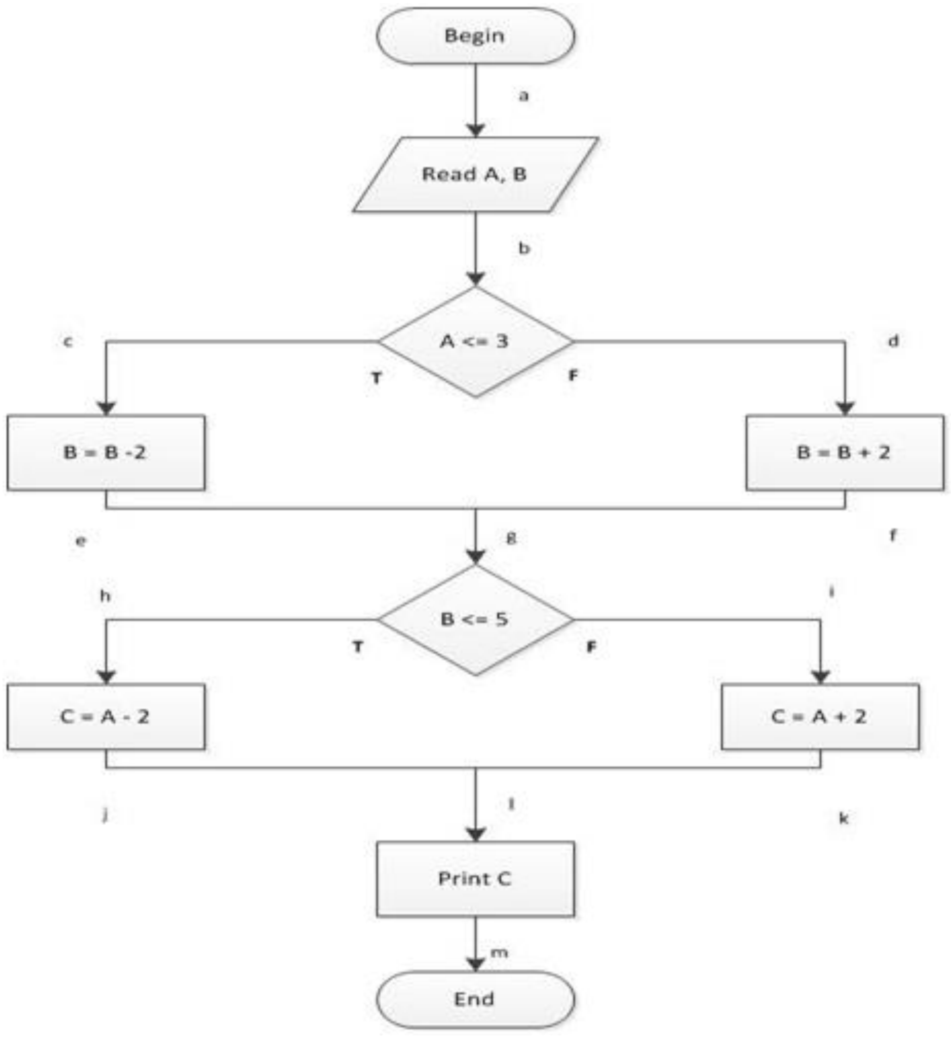
Lab No. 13 (White Box Path testing for Odd/Even)

/*Design, develop, code and run the program in any suitable language to implement odd or even. Determine the basis paths and using them derive different test cases, execute these test cases and discuss the test results. Note: Several mistakes have been inserted intentionally. Please remove them*/

```
public void howComplex() {
        int i=20;

        while (i<10) {
            System.out.printf("i is %d", i);
            if (i%2 == 0) {
                System.out.println("even");
            } else {
                System.out.println("odd");
            }
        }
    }
```

Lab No. 14 (White Box Path testing for flow chart given)

/*Design, develop, code and run the program in any suitable language to implement the flow chart given below. Determine the basis paths and using them derive different test cases, execute these test cases and discuss the test results.*/



Lab No. 15 (Understand The Automation Testing Approach (Theory Concept)

Automation

Automation is making a process automatic eliminating the need for human intervention. It is a self-controlling or self-moving process. Automation Software offers automation wizards and commands of its own in addition to providing a task recording and re-play capabilities. Using these programs you can record an IT or business task.

Benefits of Automation

Fast

Reliable

Repeatable

Programmable

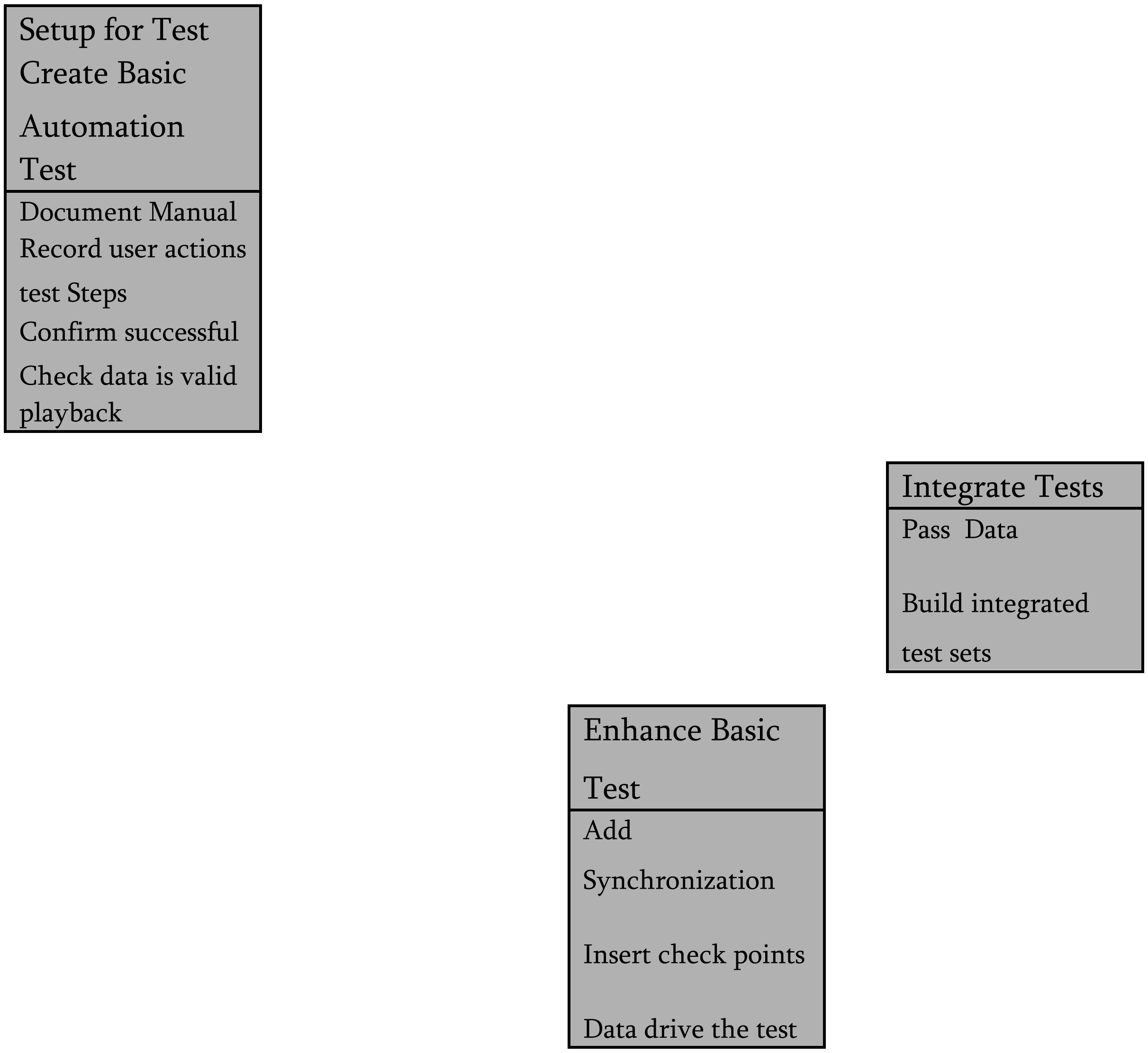
Reusable

Makes Regression testing easy

Enables 24*78 Testing

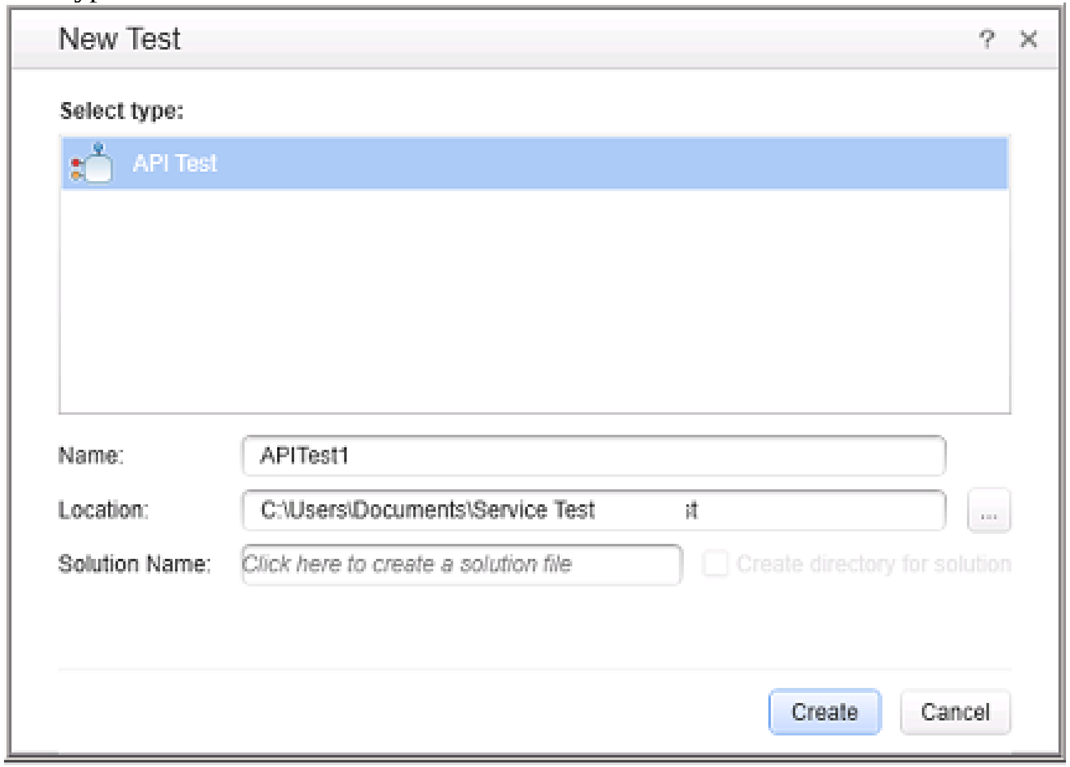
Robust verification.

Automation Test Workflow


Setup for Test
Create Basic
Automation
Test
Document Manual
Record user actions
test Steps
Confirm successful
Check data is valid
playback
Integrate Tests
Pass Data
Build integrated
test sets
Enhance Basic
Test
Add
Synchronization
Insert check points
Data drive the test

To create a new test

The canvas is the central console in which you build and run your test.
1 Open HP Service Test.
Choose Start > (All) Programs > HP Software > HP Service Test > HP Service Test. The Start Page opens.
2 Create a new test.
Click File > New > Test. The New Test dialog box opens. Select the API Test type. On machines with an installation of HP LoadRunner, the API Load Test type is also visible.



Select type:
API Test
Name: APITest1
Location: C:\Users\Documents\Service Test
Solution Name: Click here to create a solution file

3 Generate the new solution.
In the Name box, replace the default name with BasicTest, and click Create. An empty test opens, with a canvas showing the Start, Test Flow, and End sections.
The Test Flow is the section of the test containing the activities whose functionality you want to test. The Start section is ideal for defining items that you want to initialize before the test, such as test variables.

The Service Test panes?

Most of the panes in the Service Test interface, are floating, dockable windows. To show the default panes in their original positions, select View > Reset Window Layout.
The primary panes are:

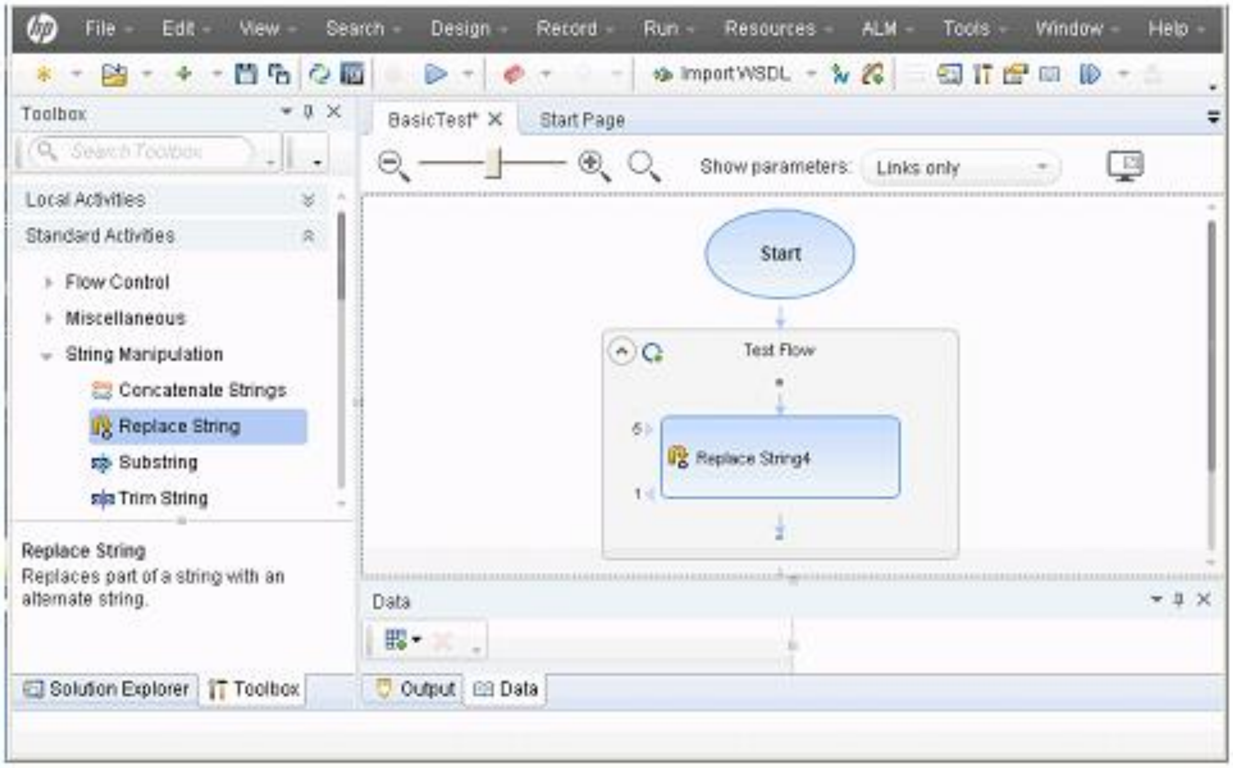
➤ Solution Explorer pane. (left) A tree hierarchy of all tests and actions in the current solution, with their references, flow, and events.

➤ Toolbox pane. (left) A collection of built-in and imported activities that can

be added as test steps. From this pane, you drag activities into the canvas.
➤ Canvas. (middle) The work area in which you organize the test steps.
➤ Data Pane. (bottom) A tree hierarchy of data sources that can be used with the test—imported Excel and XML files or database tables, or a manually defined table.
➤ Output Pane. (bottom) An informational area providing information about the test run and status.

Create a test step

You create test steps by dragging activities from the Toolbox pane into the canvas.
Create a simple test step to illustrate the use of the Toolbox and the various panes.
Create a sample Replace String test step:
1 Locate the Replace String activity.
In the left pane, click the Toolbox tab to show the Toolbox pane. Expand the String Manipulation category and select Replace String.Lesson 2 • Build a Simple Test
18
2 Create a step.
Drag the Replace String activity onto the canvas and drop it in the Test Flow. This activity searches for text within a specific string, and replaces it with new text. Alternatively, double-click on the activity in the Toolbox to add it to the canvas.

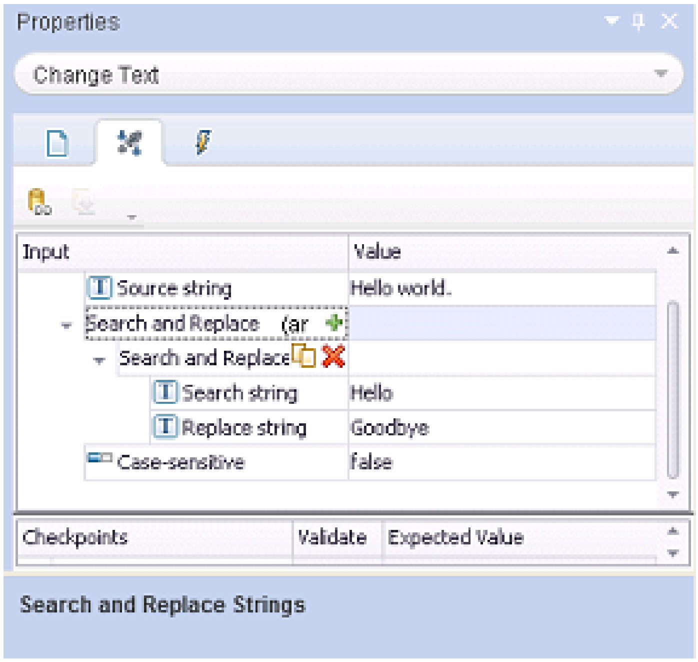


Replace String
Replaces part of a string with an alternate string.

3 Change the step's display name in the General view.
Select View > Properties. Select the Replace String step in the canvas, and in the Properties pane, click the General tab. In the Name row, type Change Text and press ENTER. This changes the step name in the canvas.

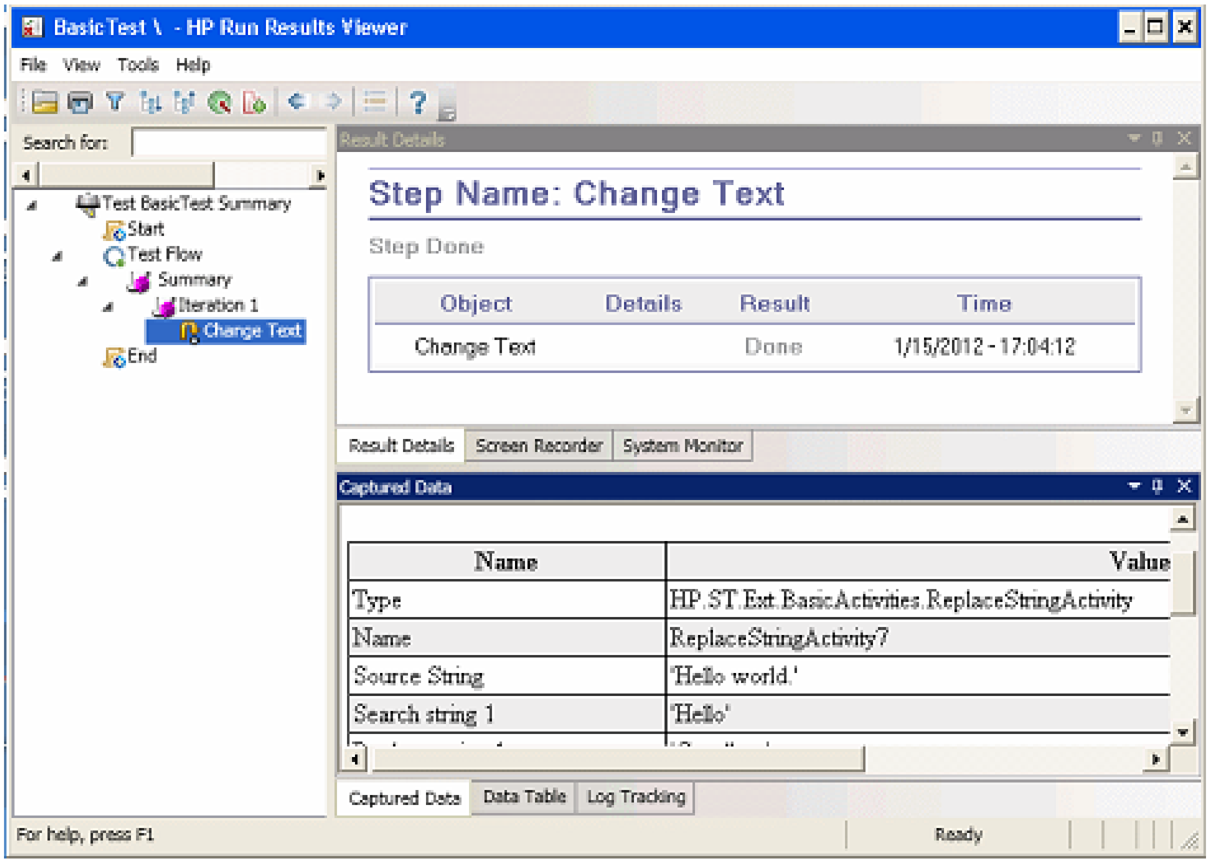
4 Set the input properties.
In the Properties pane, select the Input/Checkpoints tab. Enter the following values:

- ➤ Source string: Hello world.
- ➤ Search string: Hello
- ➤ Replacement string: Goodbye
- ➤ Case-sensitive: false



Input	Value
Source string	Hello world.
Search and Replace (ar	
Search and Replace	
Search string	Hello
Replace string	Goodbye
Case-sensitive	false

5 Run the test.
Click the Run button or press F5 to open the Run Test dialog box. Click Options to expand the dialog box. Select the Temporary run results folder option. Click Run to compile and run the test.
6 View the results.
The Run Results Viewer opens.
Select View > Expand All or click the Expand All toolbar button. Click the Change Text node. View the source and replacement strings and note the result string, Goodbye world. This is in fact the expected string—the test passed.
When you are finished reviewing the results, close the Run Results Viewer.



Step Name: Change Text

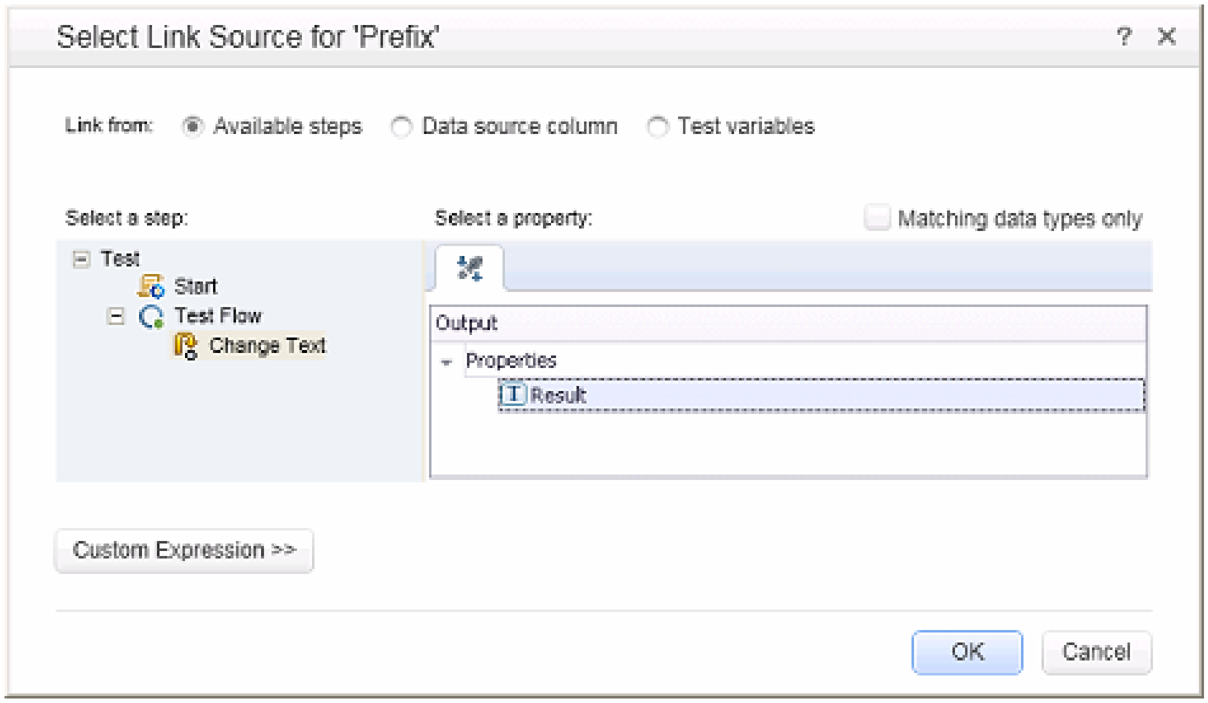
Step Done

Object	Details	Result	Time
Change Text		Done	1/15/2012 - 17:04:12

Name	Value
Type	HP.ST.Ext.BasicActivities.ReplaceStringActivity
Name	ReplaceStringActivity7
Source String	'Hello world.'
Search string 1	'Hello'

7 Set a checkpoint.
In the previous step, you manually viewed the output to check if the result matched the expected value. Checkpoints allow you to see whether the action was successful without having to manually check the result. Checkpoints are the means to validate the test—a success or failure is determined by its checkpoints.
Return to the Properties pane (right pane) and ensure that the Input/Checkpoints tab is displayed. Click in the lower part of the pane, the Checkpoints section, and select the check box in the Results row to enable the checkpoint. In the Expected value column, type the expected string, Goodbye world.
Run the test again. In the Run Results Viewer, expand the nodes, and note the checkmark. This indicates that the checkpoint passed since the result matched the expected value.
When you are finished reviewing the results, close the Run Results Viewer.
Connect test steps
1 Add a Concatenate String step.
In the Toolbox pane, select Concatenate String from the String Manipulation category. Drag the activity into the canvas and drop it below the Change Text step in the Test Flow. This activity concatenates two strings.
2 Set the prefix.
In the canvas, select the Concatenate String(x) step. In the Properties pane, click the Input/Checkpoints tab. In the upper Input section, move the

mouse into the Value cell of the Prefix row. Click the Link to a data source button. The Select Link Source dialog box opens.


Select Link Source for 'Prefix'

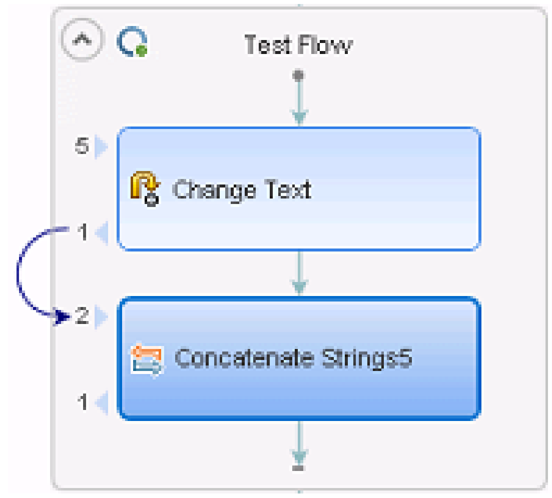
Link from: [✓ Available steps] Data source column Test variables

Select a step: Test > Start, Test Flow > Change Text

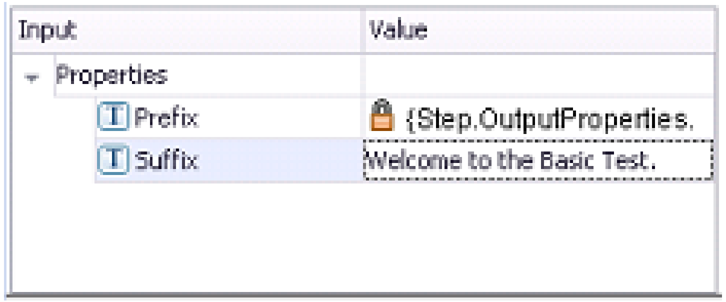
Select a property: Output > Properties > Result

Matching data types only

3 Link the steps.
In the Select Link Source dialog box, select the Available steps option. Select the Test Flow > ChangeText node. In the right pane, double-click the Results node. The canvas now reflects that data is moving from Change Text to ConcatenateString.

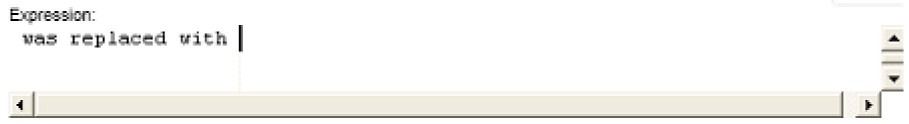


4 Configure the suffix.
In the Properties pane, type the text Welcome to the Basic Test. into the Suffix property’s Value field.



Input	Value
Properties	
Prefix	{Step.OutputProperties.
Suffix	Welcome to the Basic Test.

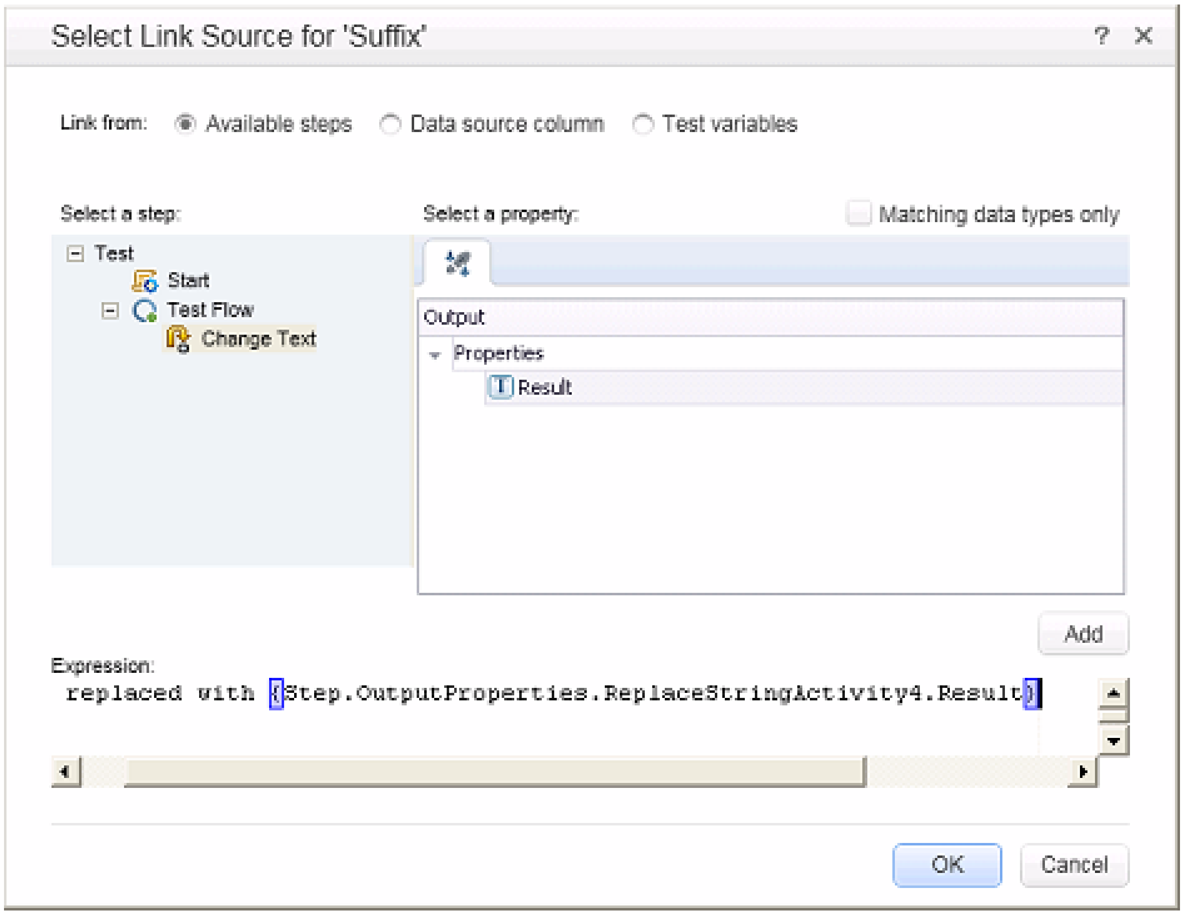
5 Run the test.
Click the Run button or press F5 to run the test.
6 View the report.
Expand the Run Results tree and select the ConcatenateStringsActivity node. The report shows the result of the concatenated strings: Goodbye World.Welcome to the Basic Test.
When you are finished reviewing the results, close the Run Results Viewer.
Map data from multiple sources?
Using the Select Link Source dialog box, you can link to one or more of the following data sources to provide input values: Available steps, Data source column, and Test variables. In the above section, you used the Available steps source for one value, and manually typed in the data for another value.
You can create a custom expression to use multiple data sources as a property value. In this section, you will use the Select Link Source dialog box to create an expression for the Suffix property that uses both manual entry and automatic values from the Available steps option.
1 Set the prefix.
In the canvas, select the ConcatenateString step. Open the Input/Checkpoints view in the Properties pane. Click in the Value cell of the Prefix row and click the X to clear the contents. Type a new prefix Hello world.
2 Open the Select Link Source dialog box.
Click in the Value cell of the Suffix row and click X to clear its contents. Click the Link data to source button. The Select Link Source dialog box opens.
3 Edit the suffix.
In the Select Link Source dialog box, click the Custom Expression button to expand it. In the Expression box, type the following:" was replaced with " (adding a space before and after the phrase to improve the readability).



Expression:

```
 was replaced with 
```

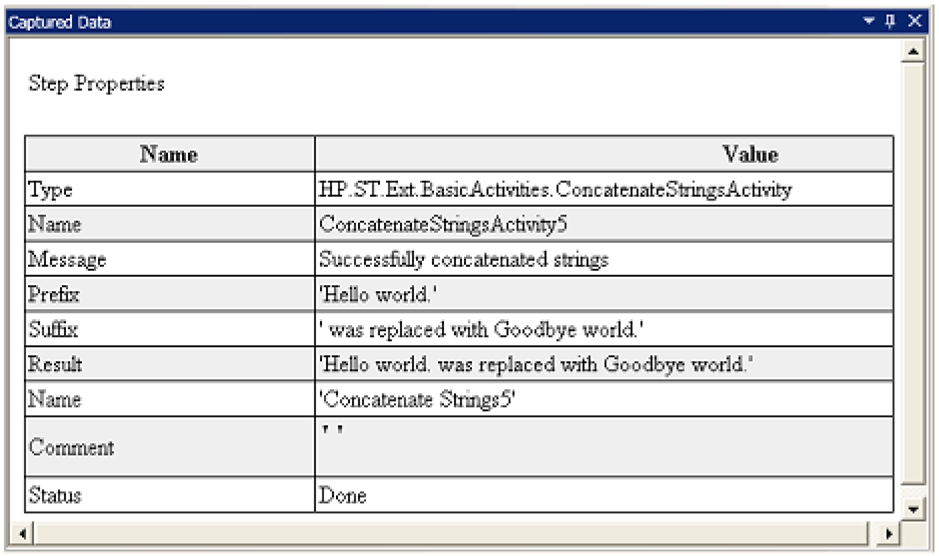
4 Add another source.
Select the Available steps option and select the Change Text node in the left pane. Select the Result node in the right pane, and click Add. The Expression box shows both sources.



Expression:

```
replaced with {Step.OutputProperties.ReplaceStringActivity4.Result}
```

5 Run the test and view the report.
Click the Run button to run the test. Expand the results and select the ConcatenateString node. The report shows the result of the concatenated strings.

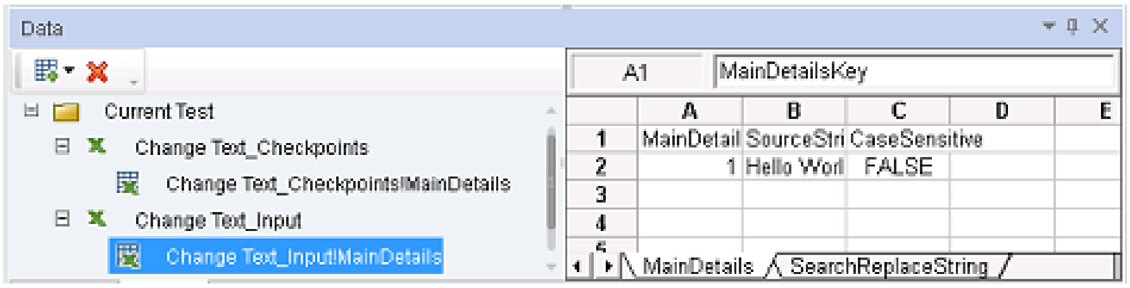


Step Properties

Name	Value
Type	HP.ST.Ext.BasicActivities.ConcatenateStringsActivity
Name	ConcatenateStringsActivity5
Message	Successfully concatenated strings
Prefix	'Hello world.'
Suffix	' was replaced with Goodbye world.'
Result	'Hello world. was replaced with Goodbye world.'
Name	'Concatenate Strings5'
Comment	' '
Status	Done

6 Close the Run Results Viewer window.
How do I data drive the step?

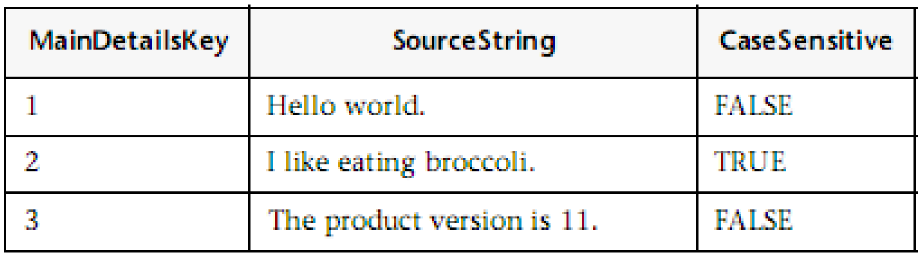
Data driving is the assigning of data to test steps from a data source, such as an Excel or XML file, database, or local table. The goal of data driving is to run the same business process with different values. It allows you to check your application in different scenarios, by modifying only the data tables.
To data drive test steps
1 Data drive the input arguments.
Select the Change Text step in the canvas. Open the Input/Checkpoint view in the Properties pane, and click the Data Drive button. The Data Driving dialog box opens.
2 Specify a data provider.
In the Data Driving dialog box:
a Set the Data Provider type to Excel.
b Enable data driving for Both Input and Checkpoints.
c Clear the Configure 'Test Flow' as a ForEach loop using the new data source option, which repeats the Test Flow according to the number of data rows. You will manually set the number of iterations in a later step.
d Click OK to close the Data Driving dialog box.
e Accept the popup message. The data driving mechanism replaces the constant values with the new expressions, {DataSource.Change Text_Input!MainDetails.SourceString}.
3 View the Data pane.
Make sure the Data pane is visible. If not, choose View > Data. Expand the Change Text_Input node and select the Change Text_Input!MainDetails node. The data pane shows a data table with a column for each input property, and one row of values corresponding to the property, Hello World. and FALSE (or empty check box for installations without Excel) that you entered earlier.



A1 MainDetailsKey

	A	B	C	D	E
1	MainDetail	SourceStri	CaseSensitive		
2	1	Hello Worl	FALSE		
3					
4					

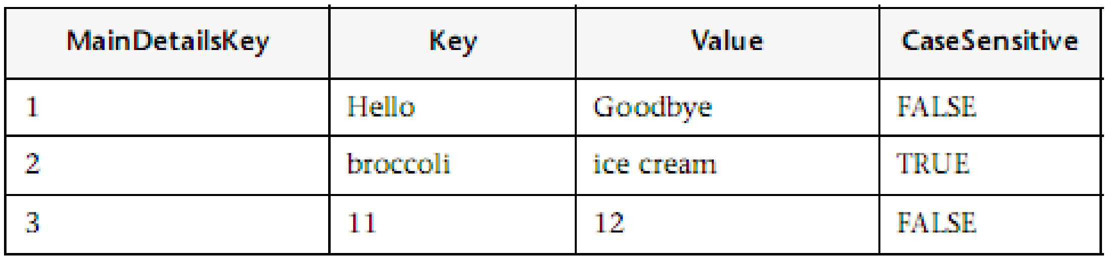
4 Add new data.
Add two additional rows to the Change Text_Input!MainDetails table. Make sure to copy the text exactly, including punctuation where included.



MainDetailsKey	SourceString	CaseSensitive
1	Hello world.	FALSE
2	I like eating broccoli.	TRUE
3	The product version is 11.	FALSE

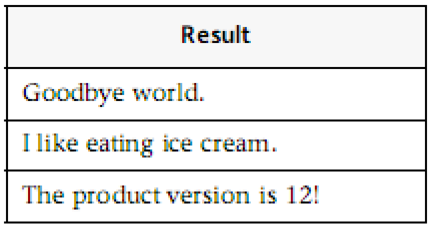
5 Add new search and replace data.
Click the Text_Input!SearchReplaceString node and add two additional rows

to the table. Make sure to copy the text exactly, including punctuation
where included.



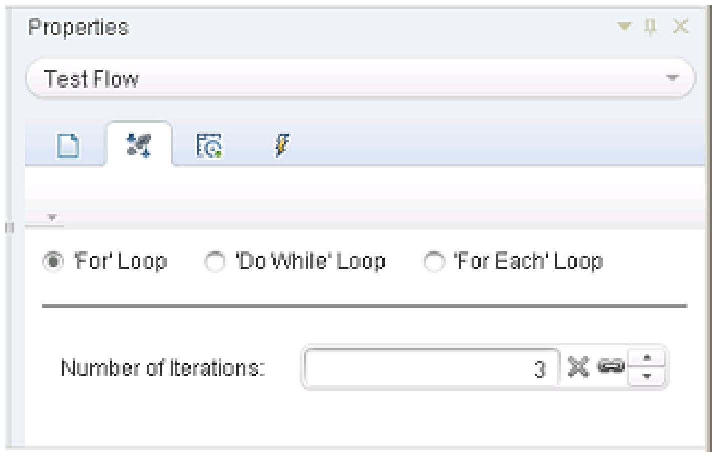
MainDetailsKey	Key	Value	CaseSensitive
1	Hello	Goodbye	FALSE
2	broccoli	ice cream	TRUE
3	11	12	FALSE

6 Add checkpoint values.
Expand the Change Text_Checkpoints node and select the Change
Text_Checkpoints!MainDetails node. Add values to this column as shown
below.



Result
Goodbye world.
I like eating ice cream.
The product version is 12!

7 Set the number of iterations.
The number of iterations is the number of times to repeat the step. We will
set it to 3, corresponding to the number of rows of data in our table.
Return to the canvas and click inside the Test Flow frame—but not within a
test step. Open the Input/Checkpoint view in the Properties pane. Select
’For’ Loop and set the Number of Iterations to 3.



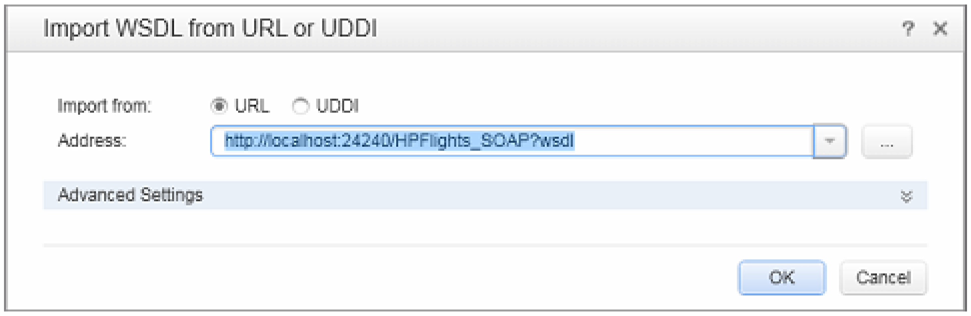
[✓ 'For' Loop] 'Do While' Loop 'For Each' Loop

Number of Iterations: 3

8 Run the test and view the report.
Click the Run button or press F5 to compile and run the test. The test runs
three times, using the three lines of data in the table.

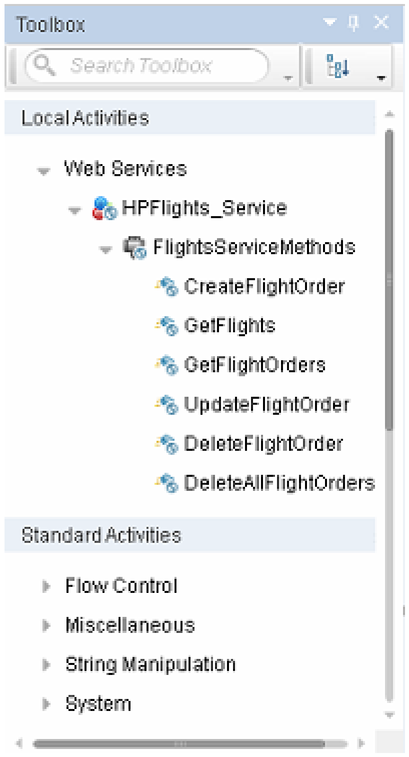
Lab No. 16 (Conduct a test suite for flight reservation system.)

Import a Web service
A WSDL file defines the operations in a Web service. To use the WSDL file,
we can import it into our test. First we will import the sample application's WSDL file.
1 Start the Sample Flight application from start-program-hpsoftware.
2 Create a new solution.
Select File > New > Test and specify the name se-your roll no for a new API
Test. Click Create.
3 Open the Import Service dialog box.
Select Import WSDL > Import WSDL from URL or UDDI on the toolbar.
4 Specify an import source.
Select the URL option and specify the location
http://localhost:24240/HPFlights_SOAP?wsdl. Click OK.

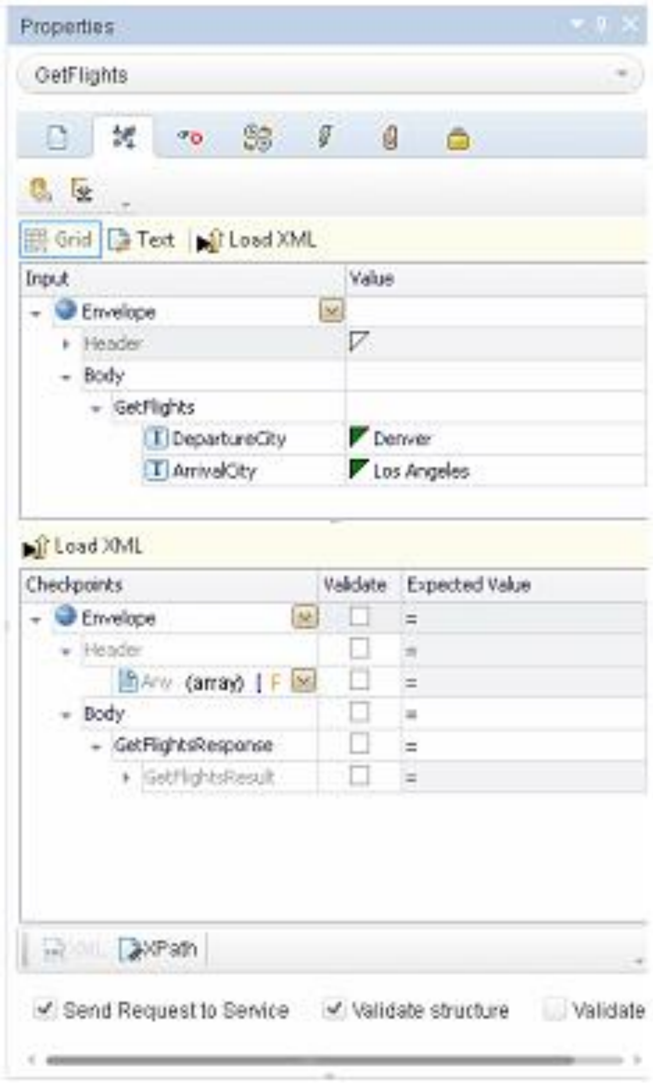


Import from: [✓ URL] UDDI
Address: http://localhost:24240/HPFlights_SOAP?wsdl

5 View the service's operations.
The import created a new branch of Web service operations in the Toolbox,
under the Web Services category. Expand the node to view the operations.



Build a Web service test?
Create a new flight order using the HPFlights Web service.
In order to create a flight order, you must first know the available flights.
First you will run the GetFlights step that retrieves all of the flights to your destination. In the next step, you will use the first flight number returned, as input for the CreateFlightOrder step.
1 Create a GetFlights step.
Expand the Web services > HPFlights_Service node and drag the GetFlights activity into the Test Flow.
2 Assign values for DepartureCity and ArrivalCity.
Open the Input/Checkpoints view and expand the Body > GetFlights node.
To select a city, click the arrow in the row to open a drop down list. Choose Denver as the DepartureCity and Los Angeles for the ArrivalCity.



Input	Value
Envelope	
Header	
Body	
GetFlights	
DepartureCity	Denver
ArrivalCity	Los Angeles

Checkpoints	Validate	Expected Value
Envelope	☐	=
Header	☐	=
Arr (array)	☐	=
Body	☐	=
GetFlightsResponse	☐	=
GetFlightsResult	☐	=

✓ Send Request to Service ✓ Validate structure ☐ Validate

3 Create a CreateFlightOrder step.
Drag the CreateFlightOrder activity from the toolbox into the Test Flow, beneath the GetFlights step.
4 Set the values for the CreateFlightOrder step.

In the Input/Checkpoints view, expand the Body > CreateFlightOrder > FlightOrder node, and set the values for creating a flight order:

➤ Class—Select a class, such as Business from the dropdown list.

➤ CustomerName—any value

➤ DepartureDate—use the dropdown to open a calendar and select a date at least two days in the future.

➤ FlightNumber—leave blank for now. We will set it in the following steps.

➤ NumberofTickets—use the scroller to set any value.

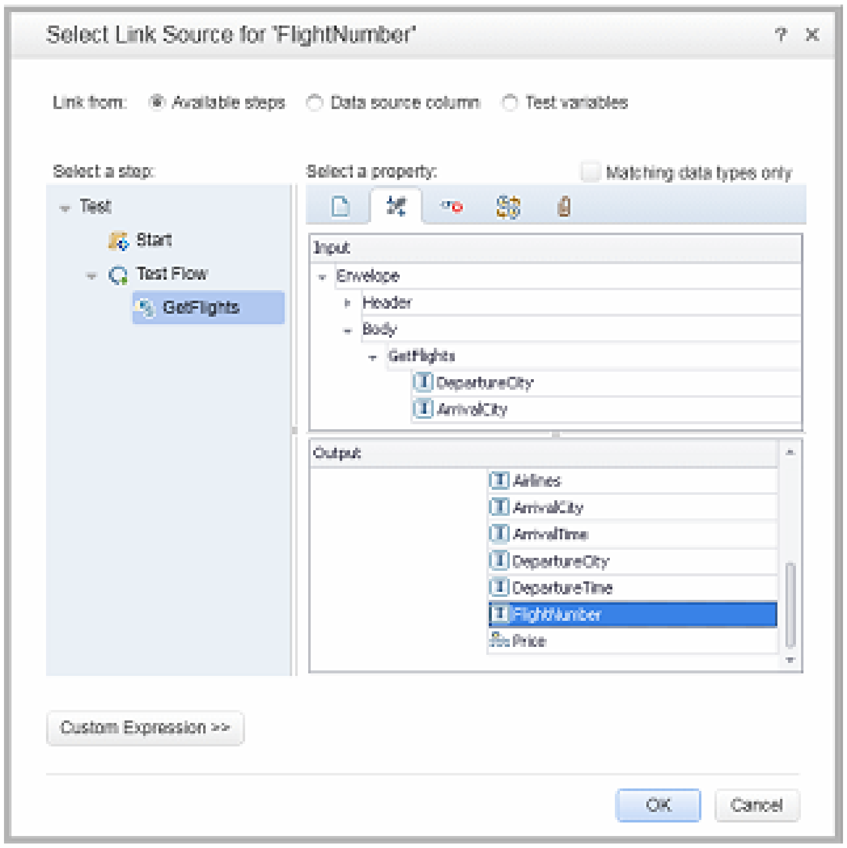
5 Link the output of GetFlights to the CreateFlightOrder step.

a Click the Link to a data source icon in the right corner of the FlightNumber row. The Select Link Source dialog box opens.

b Select Available steps and select the GetFlights node.

c In the right pane, select the Input/Checkpoints button. In the Output section, expand all nodes under the Body node. Click the Add button in the Flight (array) node row to create the Flight[1] array.

d Expand the Flight[1] array, select the FlightNumber element, and click OK. The application asks if you want to enclose the target step in a loop. Select Yes.


Select Link Source for 'FlightNumber'

Link from: (●) Available steps ( ) Data source column ( ) Test variables

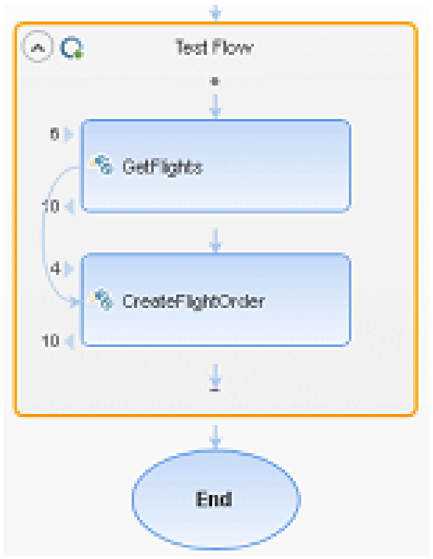
Select a step:
- Test
  - Start
  - Test Flow
    - GetFlights

Select a property: [ ] Matching data types only

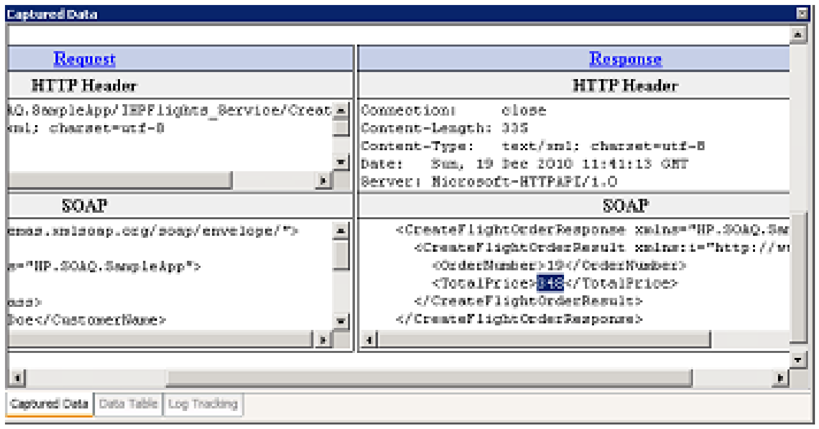
Input
- Envelope
  - Header
  - Body
    - GetFlights
      - DepartureCity
      - ArrivalCity

Output
- Airlines
- ArrivalCity
- ArrivalTime
- DepartureCity
- DepartureTime
- FlightNumber
- Price

The canvas indicates a connection between the two steps.



6 Reset the number of iterations.
The number of iterations is the number of times to repeat the step. Return to the canvas and click inside the Test Flow frame—not within the test step. Open the Properties pane's Input view. Select For Loop and set the Number of Iterations to 1.
7 Run the test.
Click the Run button. Observe the log in the Output tab. The Run Results Viewer opens automatically.
8 Check the results.
In the left pane, click the parent node and select Expand All from the right-click menu. Click the CreateFlightOrder node. In the Captured Data pane, scroll down to the Web service Call HTTP Snapshot section and look at the Response pane. Note the output of the request—OrderNumber and TotalPrice. Copy the TotalPrice value to the clipboard for use in the next step.



```
AQ.SampleApp/IHPFlights_Service/Creat
xml; charset=utf-8
```

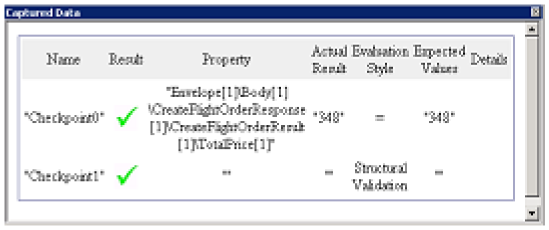
```
Connection:      close
Content-Length:  335
Content-Type:    text/xml; charset=utf-8
Date:    Sun, 19 Dec 2010 11:41:13 GMT
Server:  Microsoft-HTTPAPI/1.0
```

```
emas.xmlsoap.org/soap/envelope/">

s="HP.SOAQ.SampleApp">

ass>
Doe</CustomerName>
```

```
<CreateFlightOrderResponse xmlns="HP.SOAQ.Sa
  <CreateFlightOrderResult xmlns:i="http://w
    <OrderNumber>19</OrderNumber>
    <TotalPrice>848</TotalPrice>
  </CreateFlightOrderResult>
</CreateFlightOrderResponse>
```

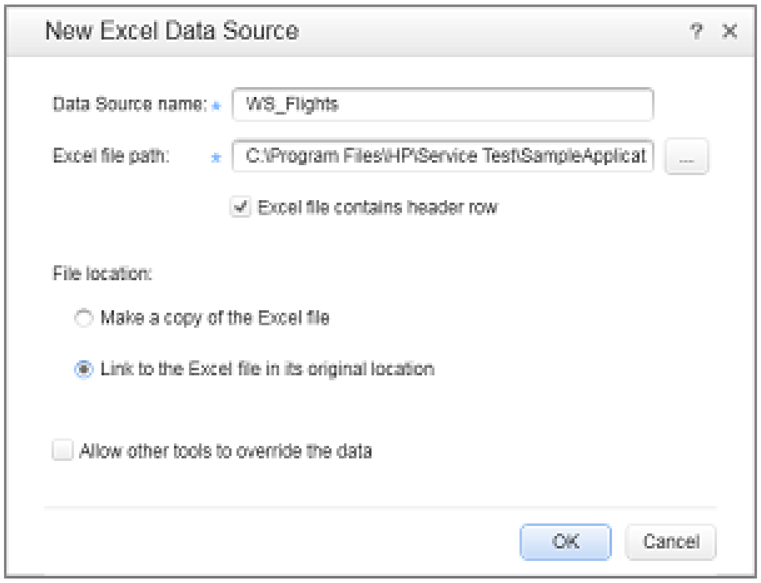
Tip: Click the Request or Response links to open the SOAP in a separate browser.
When you are finished viewing the results, close the Run Results Viewer.

9 Set a checkpoint.
Select the CreatFlightOrder step in the canvas. Open the Properties pane's Input/Checkpoints view, click in the Checkpoints grid, and expand the CreateFlightOrderResponse node. Paste the clipboard contents from the previous step into the TotalPrice field. Select the check box in the TotalPrice row, to include it as a checkpoint.
10 Run the test and view the checkpoint results.
Run the test again and expand the results tree. Select the Checkpoints node for CreateFlightOrder. The report shows a checkmark and indicates the expected and actual values. If the expected value was not returned by the server, the report indicates a failure.



Name	Result	Property	Actual Result	Evaluation Style	Expected Values	Details
"Checkpoint0"	✓	"Envelope[1]\Body[1]\CreateFlightOrderResponse[1]\CreateFlightOrderResult[1]\TotalPrice[1]"	"348"	=	"348"	
"Checkpoint1"	✓	""	—	Structural Validation	—	

When you are finished viewing the results, close the Run Results Viewer.
How do I integrate data into a test?
In this section you will learn how to integrate data from an existing source, and how to data drive the test. When you data drive a test, the Data pane automatically creates a data table whose values you can edit.
1 Import Sample Data
In the Data pane, in the bottom of the Service Testwindow, select New > Excel. The Add New Excel Data Source dialog box opens.
a Browse for the sample application's Excel file, SampleAppData.xlsx, in the <installation directory>/SampleApplication folder. By default, this folder is C:\Program Files\HP\HP Service Test\SampleApplication.
b Enable the Excel file contains header row option, since the sample file contains a header row.
c Enter WS_Flights as a Data source name.
d Select Referenced Data Source as the mode of import. This links to the Excel file at its original location, so that if data changes, your data source will be current.
e Click OK.


New Excel Data Source

Data Source name: * WS_Flights

Excel file path: * C:\Program Files\HP\Service Test\SampleApplicat

[✓] Excel file contains header row

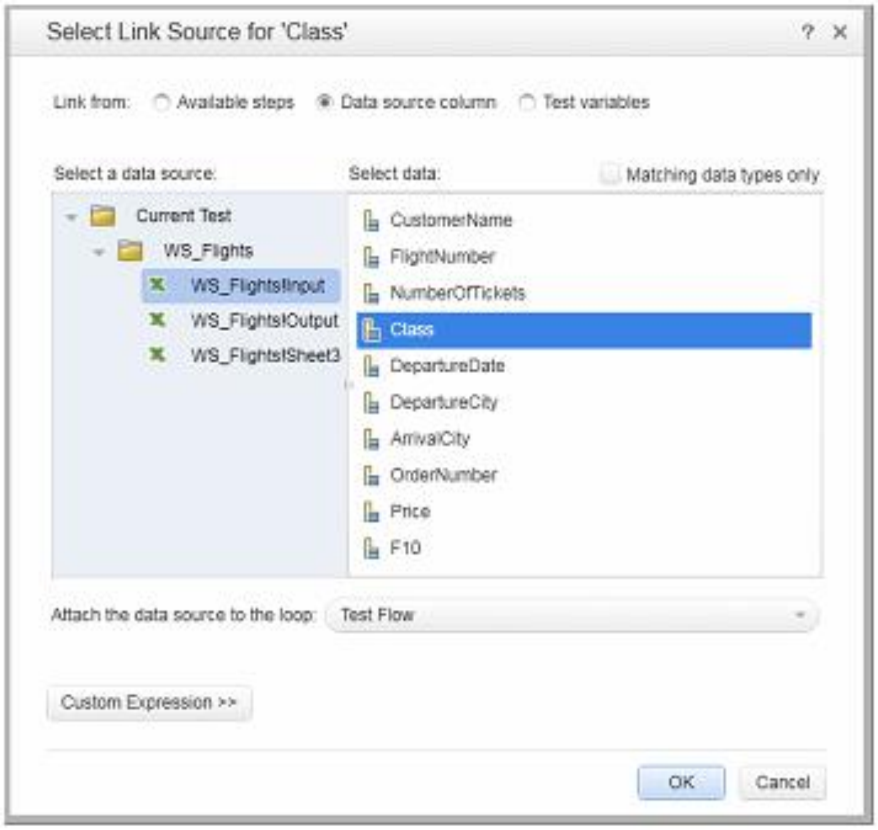
File location:

( ) Make a copy of the Excel file

(●) Link to the Excel file in its original location

[ ] Allow other tools to override the data

2 Open the Select Link Data dialog box.
Select the CreateFlightOrder step in the canvas and open the Input/Checkpoints view. In the Input section, expand all the nodes and select the Class row. Click the Link to a data source icon. The Select Link Source dialog box opens.
3 Select a value from the data source.
Select the Data source column option.


Select Link Source for 'Class'

Link from: ( ) Available steps (●) Data source column ( ) Test variables

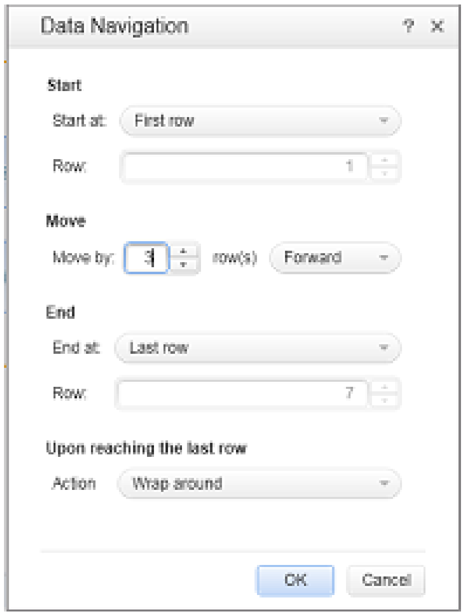
Select a data source:
- Current Test
  - WS_Flights
    - WS_Flights!Input
    - WS_Flights!Output
    - WS_Flights!Sheet3

Select data: [ ] Matching data types only
- CustomerName
- FlightNumber
- NumberOfTickets
- Class
- DepartureDate
- DepartureCity
- ArrivalCity
- OrderNumber
- Price
- F10

Attach the data source to the loop: Test Flow

4 Use the sample Excel data.

Select the WS_Flights!Input node, and select Class in the right pane. Click OK. This instructs the test to refer to this column in the sample data during the test run.
Repeat this for the other input parameters: CustomerName, DepartureDate, FlightNumber, and NumberofTickets.
5 Disable the checkpoint.
In the Input/Checkpoint view, click in the Checkpoints grid. Clear the check box for the TotalPrice property, to exclude it as a checkpoint.
6 Set the navigation settings.
The navigation settings let you indicate how to use the data in your data source. You can specify from which row to begin, how many rows to advance, and in what direction to move for the next set of values. You can also specify what to do when reaching the end of the data table—wrap around or continue using the last line.
a In the canvas, click in the Test Flow but not within a step.
b In the Properties pane, click the Data Sources view button.
c Select the WS_Flights!Input node and click Edit to open the Data Navigation dialog box.
d Specify the data navigation details: Start at: First row, Move: Move by 3 rows Forward, End at: Last row, and Upon reaching the last row: Wrap around.


Data Navigation

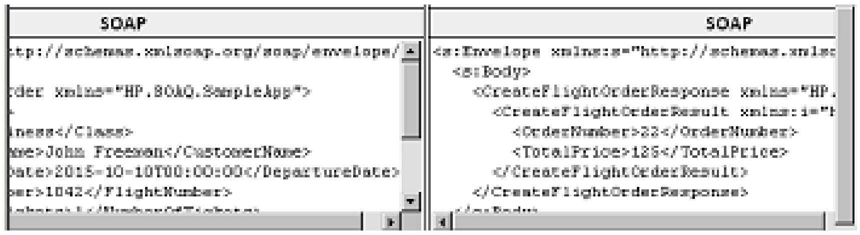
Start
Start at: First row
Row: 1

Move
Move by: 3 row(s) Forward

End
End at: Last row
Row: 7

Upon reaching the last row
Action: Wrap around

e Click OK.
7 Run the test and view the results.
Click the Run button and observe the results in the Output window. The Run Results Viewer opens automatically. Expand the result tree and select the CreateFlightOrder step. Scroll down within the Captured Data tab an note the data from the Excel file in the SOAP request (left pane), and the result in the SOAP response (right pane).



```
tp://schemas.xmlsoap.org/soap/envelope/

rder xmlns="HP.SOAQ.SampleApp">

iness</Class>
me>John Freeman</CustomerName>
Date>2015-10-10T00:00:00</DepartureDate>
ber>1042</FlightNumber>
```

```
<s:Envelope xmlns:s="http://schemas.xmlsc
  <s:Body>
    <CreateFlightOrderResponse xmlns="HP.
      <CreateFlightOrderResult xmlns:i="h
        <OrderNumber>22</OrderNumber>
        <TotalPrice>125</TotalPrice>
      </CreateFlightOrderResult>
    </CreateFlightOrderResponse>
```

When you are finished viewing the results, close the Run Results Viewer.
Using multiple data sources and custom code?
This section describes how to define data using multiple data sources and sending information to the report through a custom code step.
1 Create a new test.
Create a new test called WebServicesCustom and import the HP Flights Services WSDL as described in “How do I import a Web service?” on page 32.
2 Create test steps.
Drag the activities into the canvas in the following order: From the Web services folder: GetFlights and CreateFlightOrder. From the Miscellaneous folder, drag in Custom Code.
3 Add a data source.
In the Data pane, select New > Excel. In the Add New Excel Data Source dialog box:
a Browse for the sample application Excel file. By default, this folder is C:\Program Files\HP\HP Service Test\SampleApplication.
b Select the Excel file contains header row check box.
c Enter WS_Flights as a Data source name.
d Select the Referenced data source mode.
4 Assign values for GetFlights.
Select the GetFlights step in the canvas and open the Input/Checkpoints view in the Properties pane. In the Input section, select DepartureCity= Denver, and ArrivalCity=Los Angeles.
5 Assign values for the CreateFlightOrder.
Select the CreateFlightOrder activity in the canvas and open the Input/Checkpoints view. Expand the Body > FlightOrder node and set the input properties as follows:

➤ Class. Economy

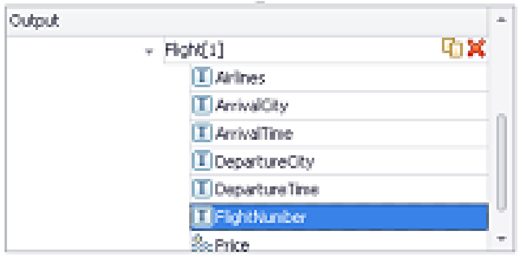
➤ CustomerName. Click the Link to a data source button in the right corner of the CustomerName row. The Select Link Source dialog box opens. Select Data source column, and expand the tree to show the WS_Flights!Input node. In the right pane, select the CustomerName parameter. Click OK.

➤ DepartureDate. A date in the following format YYYY-MM-DDTHH:MM:SS For example, 2015-02-18T00:00:00. Use the drop down arrow to open the calendar. The date must be at least two days ahead of the current date.

➤ NumberofTickets. 3

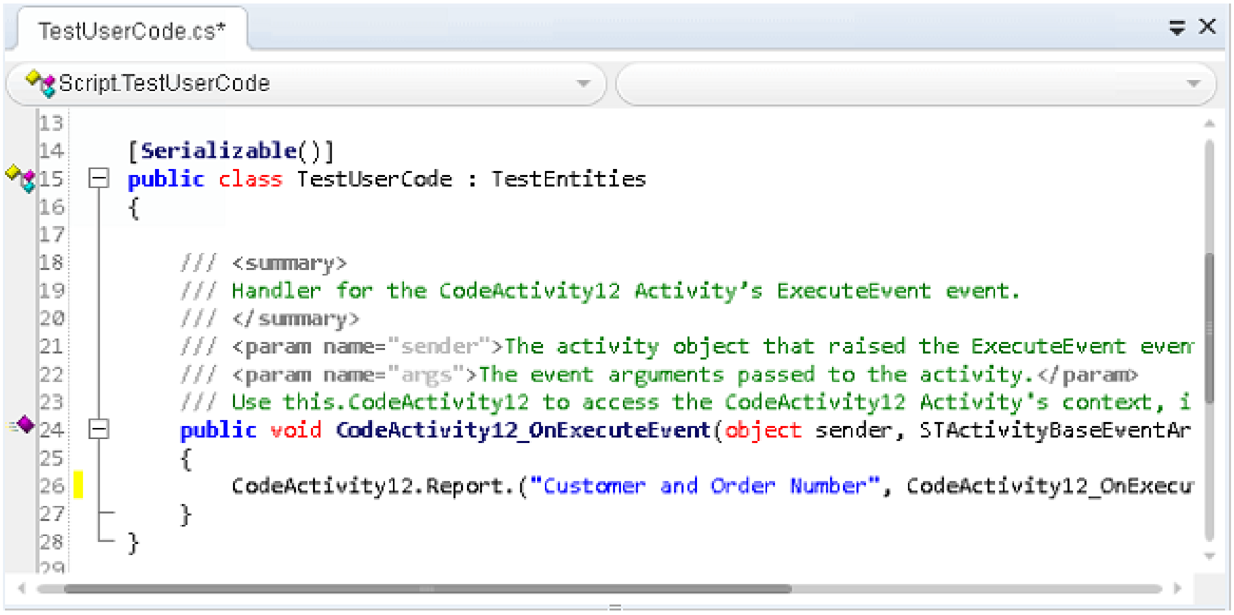
➤ FlightNumber. Link from the previous step:

a Click the Link to a data source button in the right corner of the FlightNumber row.
b In the Link to Source dialog box, select Available steps, expand the Test Flow branch, and click GetFlights.
c In the right pane, select the Input/Checkpoints button.
d In the Output section, click the Add button in the Flight (array) node row to create the Flight[1] array. Expand the array, select FlightNumber, and click OK.



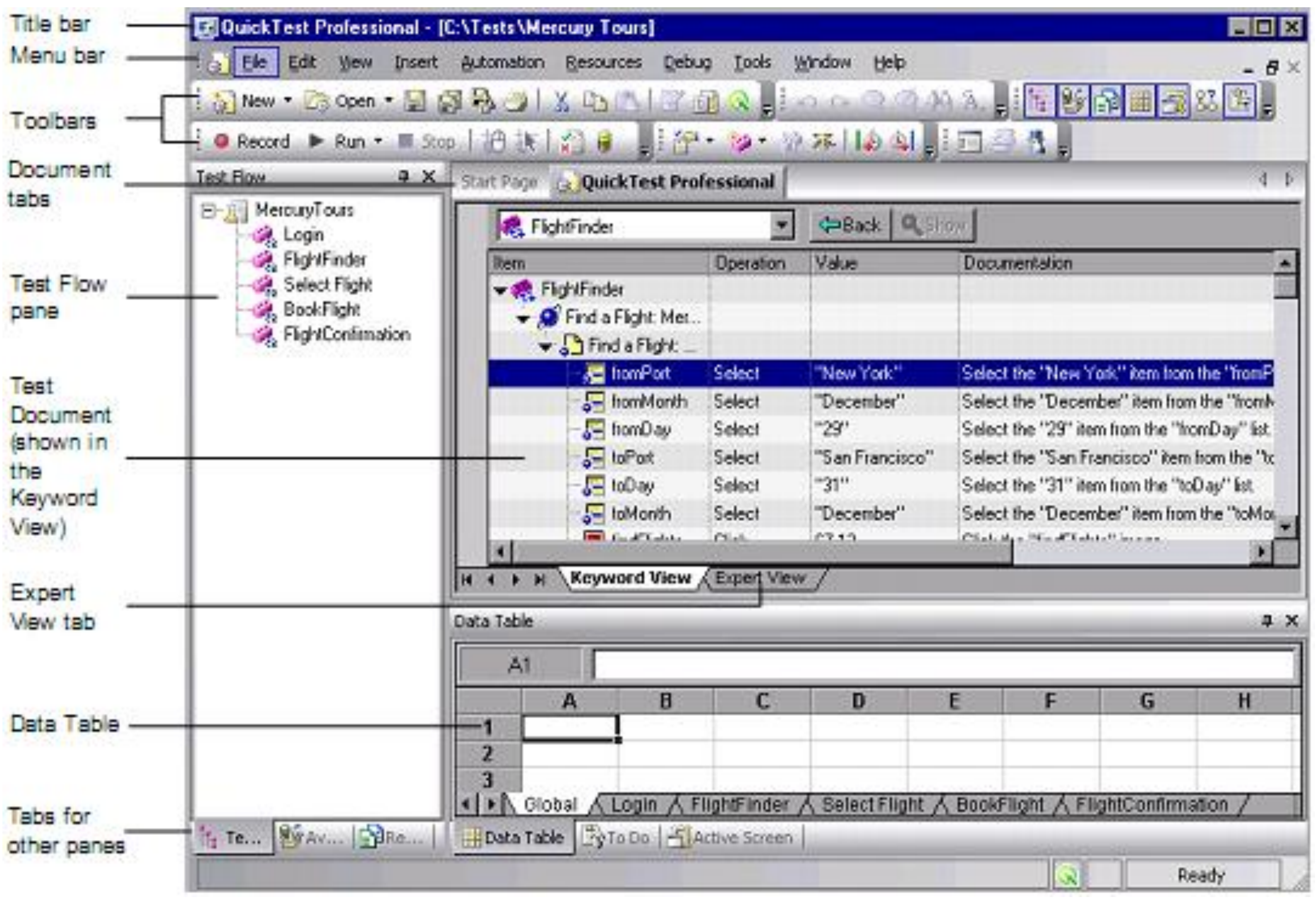
6 Create a property for the custom code step.
Select the Custom Code activity in the canvas and open the Input/Checkpoint view in the Properties pane. Expand the Add Property toolbar button and select Add Input Property. Create a new String type property called FlightInfo.
7 Define values for the custom code step.
In this step you will define a value using multiple sources. In this example, you will set a value which is a combination of the CustomerName, a constant string, and the OrderNumber.
a Click the Link to a data source button in the right corner of the FlightInfo row. The Select Link Source dialog box opens.
b Click Custom Expression to show the Expression area.
c Select the Data source column option. In the tree's WS_Flights!Input node, select CustomerName. Click Add.
d In the Expression area, type _OrderNumber_ (with the underscores) after the existing expression.
e Select Available steps and expand the Test Flow branch. Select the CreateFlightOrder node. In the right pane, select the Input/Checkpoints button. In the lower pane, expand the Output Body node, expand the tree, select the OrderNumber element, and click Add.
The CustomCode input property, FlightInfo, has the following value:
{DataSource.WS_Flights!Input.CustomerName}_OrderNumber_{Step.OutputProperties.StServiceCallActivity(x).Body.CreateFlightOrderResponse.CreateFlightOrderResult.OrderNumber}
Click OK to close the dialog box.
8 Create an event.
In this step you will create an event handler in order to use the Application Program Interface (API). You can add C# code to this section. Defining events let you adapt your test to your custom requirements, and perform actions that are not built-in to Service Test. In this example, you will add code that sends a custom string to the report.
Select the Custom Code step in the canvas. In the Properties pane, click the

Events button. In the ExecuteEvent row, click the drop down arrow and select Create a default handler. Service Test creates an event called CodeActivity(x)_OnExecuteEvent and opens a new tab TestUserCode.cs.
9 Edit the "Todo" section, using the Index assigned to the activity. Replace the commented text in the Todo section with the following: CodeActivity(x).Report("Customer and Order Number",CodeActivity(x).Input.FlightInfo);
In the following example, the index assigned to the event was 12, so the string is CodeActivity12.Report("Customer and Order Number",CodeActivity12.Input.FlightInfo);



```
    [Serializable()]
    public class TestUserCode : TestEntities
    {

        /// <summary>
        /// Handler for the CodeActivity12 Activity’s ExecuteEvent event.
        /// </summary>
        /// <param name="sender">The activity object that raised the ExecuteEvent even
        /// <param name="args">The event arguments passed to the activity.</param>
        /// Use this.CodeActivity12 to access the CodeActivity12 Activity's context, i
        public void CodeActivity12_OnExecuteEvent(object sender, STActivityBaseEventAr
        {
            CodeActivity12.Report.("Customer and Order Number", CodeActivity12_OnExecu
        }
    }
```

Run test and check the results.
Drill down in the results to the Custom Code step. Note the new entry in the Captured Data pane: Customer and Order Number.
Tip: You can also use the Report Message activity under the Miscellaneous folder, to send text and property values to the report.



Item	Operation	Value	Documentation
FlightFinder			
Find a Flight: Mer...			
Find a Flight: ...			
fromPort	Select	"New York"	Select the "New York" item from the "fromP
fromMonth	Select	"December"	Select the "December" item from the "fromM
fromDay	Select	"29"	Select the "29" item from the "fromDay" list.
toPort	Select	"San Francisco"	Select the "San Francisco" item from the "to
toDay	Select	"31"	Select the "31" item from the "toDay" list.
toMonth	Select	"December"	Select the "December" item from the "toMo

Exercise: Evaluating a Test Case
In this exercise you will test your understanding of a useful test case. First, read Test Case 1, then answer the questions in Test Case 1 Review.
Test Case 1
Test Name: InsertOrder
Test Objective: Create a new order with the Flight Reservation application.
Description: Test the functionality of the process to create a new flight reservation.
Application: \QTP\samples\flight\app\flight4a.exe
Windows Versions: Windows 2000 or Windows NT
Test Requirements:
1. Verify that "Insert Done..." appears in the status bar of the Flight Reservation window.
2. Verify that an order number is displayed in the Order No box of the Flight Reservation window.
3. Verify that the application accepts different combinations of from and to cities.
4. Verify the state of the check boxes in the Open Order window.
■ Initial conditions: all enabled.
■ When Customer Name or Flight Date is checked: Order No. is disabled; the other 2 are enabled.
■ When Order No. is checked: Order No. is enable; the other 2 are disabled.
5. Verify that "10" is the maximum number of tickets accepted for a flight reservation.
6. After clearing the window of all data, verify that the order just created can be opened and

displayed in the Flight Reservation window.