**HMR**

**Institute of Technology & Management**



Laboratory Manual

**SOFTWARE TESTING**

**AND**

**QUALITY ASSURANCE**

For

Forth Year Students CSE

Dept: Computer Science & Engineering

**INDEX**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **S.No.** | **Experiment** | **Performed On** | **Checked On** | **Signature** |
| 1 | Introduction to software testing |  |  |  |
| 2 | Write a program to find the roots of the quadratic equation, its input is triple positive integers (say a, b, c) and the values may be from the interval [1,100] and perform |  |  |  |
|  | 1. Boundary Value Analysis (BVA). |  |  |  |
|  | 1. Robust Testing |  |  |  |
| 3 | Write a program to find the largest among three numbers, its input is triple positive integers (say a, b, c) and the values may be from the interval [1,100] and perform |  |  |  |
|  | 1. BVA. |  |  |  |
|  | 1. Robust Worst-Case Testing |  |  |  |
| 4 | Write a program to determine the type of triangle, its input is triple positive integers (say a, b, c) and the values may be from the interval [1,100]. The program output may be one of the following [Scalene, Isosceles, Equilateral, Not a triangle]. |  |  |  |
|  | 1. Perform BVA. |  |  |  |
|  | 1. Perform Worst-Case Testing |  |  |  |
| 5 | WAP in C/C++ to find the area of a circle, Triangle, Square and Rectangle and perform equivalence class testing**.** |  |  |  |
| 6 | WAP in C/C++ to compute 3 sides of a triangle and to determine whether they form scalene, isosceles, or equilateral triangle and perform Cause effect testing, Decision table-based testing and Equivalence Class testing. |  |  |  |
| 7 | WAP in C/C++ to compute ab and perform its decision table-based testing. |  |  |  |
| 8 | WAP in C/C++ to compute previous date, given the present date as input and perform decision table-based testing |  |  |  |
| 9 | Create a test plan document for any application (e.g. Library Management System). |  |  |  |
| 10 | Study of Any Testing Tool (Win Runner). |  |  |  |
| 11 | Study of Any Test Management Tool (QA Complete). |  |  |  |
| 12 | Automate the Test cases using Test Automation tool  (using QA Complete). |  |  |  |
| 13 | Learn how to raise and report Bugs using Bug tracking tool  (Bugzilla, Jira using QA Complete). |  |  |  |
| 14 | Study of any open source testing tool (Web Performance Analyzer/O STA). |  |  |  |
| 15 | 1. Write a program for the determination of day of the given Date and perform slice-based testing for all variables. |  |  |  |
|  | 1. To Perform slice-based testing for all variables for program to compute total salary of an employee, given his basic salary |  |  |  |
| 16 | Write a program in C/C++ to find whether a triangle is right, acute and obtuse. Its input is a triplet of 3 positive integers (say a, b, c) from interval (1 to 100). Perform slice-based testing and generate test cases for the given program. |  |  |  |
| 17 | Write a program to find the largest of three 3 numbers. The test suite selected by a testing technique- |  |  |  |

**EXPERIMENT 1**

**AIM:** Introduction to software testing

* SOFTWARE ENGINEERING:

Software engineering is a field of engineering for designing and writing programs for computers or other electronic devices. It can also be called as the systematic application of scientific and technological knowledge, methods, and experience to the design, implementation, testing and documentation of software.

* SOFTWARE TESTING (with process):

Software testing is the process of evaluating a system or its component with the intent to find whether it satisfies the specified requirements or not. In simple words, testing is executing a system in order to identify any gaps, errors, or missing requirements in contrary to the actual requirements.

* PROCESS OF SOFTWARE TESTING:

1. PLANNING AND CONTROL:

Test planning has the major tasks:

- To determine the scope and risks and identify the major tasks of testing.

- To determine the test approach.

- To implement the test strategy.

Test control has the major tasks:

* To measure and analyze the results of reviews and testing.
* To monitor and document progress, test coverage and exit criteria.
* To provide information on testing.
* To initiate corrective actions.
* To make decisions.

1. ANALYSIS AND DESIGN:

It has the following major tasks:

* To review the test basis.
* To identify test conditions.
* To design the tests.
* To evaluate requirements of testability of the requirements and the system.
* To design the test environment setup and identify required infrastructure and tools.

1. IMPLEMENTATION AND EXECUTION:

Test implementation has major tasks:

* To develop and prioritize test cases by using techniques and create test data for those tests.
* To create test suites from test cases for efficient execution.
* To verify and implement test environment.

Test execution has the following tasks:

* To execute test suites and test cases following the test procedures.
* To re-execute the tests that previously failed in order to confirm a fix. This is known as re-testing.
* To log the outcome of the test execution and record the identities and version of the software under tests.
* To compare actual results and expected results.
* To report the discrepancies as incidents if there is difference between actual and expected results.

1. EVALUATING EXIT CRITERIA AND REPORTING:

* To check the test logs against the exit criteria specified in test planning.
* To assess if more test is needed or if the specified exit criteria need to be changed.
* To write a summary report for stakeholders.

1. TEST CLOSURE:

Test closure has the major following tasks:

* To check whether all the deliverables have been delivered and to ensure that all incident reports have been resolved.
* To finalize and archive test ware such as scripts, test environments etc., for later use.
* To handover the test ware to the maintenance organization.
* To evaluate how testing went and learn lessons for future releases.
* WHY SHOULD WE TEST?

Software testing is done to reduce costly errors. The cost of errors in software can vary from nothing at all to large amounts of money. There are many reasons why systems fail but the issue that stands out the most is the lack of adequate testing. A sound testing process identifies the defects and reports them in a structured way.

* WHO SHOULD WE DO TESTING?

Testing a software system may not be responsibility of a single person but team work and size of the team is dependent on the complexity, criticality and functionality of the software under test. Testing persons executes the test and document the results. The testing persons must be cautious, curious, critical and non-judgmental and good communicators.

* WHAT SHOULD WE TEST?

We may like to test a program for all possible valid and invalid inputs and furthermore, we may like to execute all possible paths; but practically, it is quite difficult. Every exit condition of a branch statement is similarly difficult to test due to a large number of such conditions. We require effective planning, strategies and sufficient resources to target the minimum possible bottom line. We should also check the program for very large numbers, very small numbers, numbers that are close to each other, negative numbers, some extreme cases, characters, and some strange cases.

* PROGRAM AND SOFTWARE:

The SOFTWARE is the superset of the programs. It consists of one or many programs, documentation manuals and operating procedure manuals.

The PROGRAM is a combination of source code and object code. Every phase of the software development life cycle requires preparation of a few documentation manuals.

Software=Program+ Documentation manual + Operation procedure manual

* VERIFICATION AND VALIDATION:

VERIFICATION is the process of evaluating the system or component to determine whether the products of a given development phase satisfy the conditions imposed at the start of the phase.

VALIDATION is the process of evaluating a system or component during or at the end of development process to determine whether it satisfies the specified requirements.

Testing= Verification+ Validation

* FAULT, ERROR, BUG AND FAILURE:

When we make error during coding, we call this a BUG. Hence, ERROR/MISTAKE/DEFECT in coding is called a BUG.

A FAULT is the representation of an error where representation is the mode of expression such as ER diagrams, source code etc.

When the expected output does not match with the observed output, we experience a FAILURE.

* DELIVERABLES AND MILESTONES:

Different DELIVERABLES are generated during various phases of the software development. The examples are: source code, SRS etc.

The MILESTONES are the events that are used to ascertain the status of the project. For e.g. finalization of a project is a milestone.

* ALPHA, BETA AND ACCEPTANCE TESTING:

ACCEPTANCE TESTING term is used when the software is developed for specific customer. The customer is involved during acceptance testing. The discovered errors are fixed and modified and then the software is delivered to the customer. The ALPHA TESTS are conducted at the developer’s site by the anonymous customers whereas BETA TESTING is conducted by potential customers at their site. Unlike, alpha testing, the developer is not present here.

* QUALITY AND RELIABILITY:

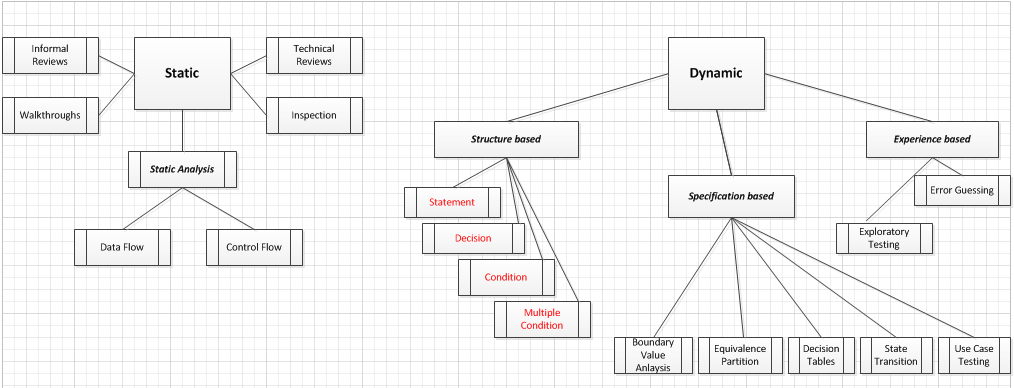
SOFTWARE RELIABILITY is defined as the probability of failure free operation for a specified time in a specified environment whereas SOFTWARE QUALITY determines how well the software is designed (quality of design) and how well the software confirms the design (quality of conformance).

* STATIC AND DYNAMIC TESTING:

STATIC TESTING refers to testing activities without executing the source code. All verification activities like inspection, walkthrough etc., come

under this category. DYNAMIC TESTING refers to executing the source code and seeing how it performs with specific inputs. All validation activities come in this category where execution of the program is essential.

DIAGRAM OF STATIC AND DYNAMIC TESTING:



* WHITE BOX AND BLACK BOX TESTING:

BLACK BOX TESTING is a software testing method in which the internal structure/design/implementation of the item being tested is not known to the tester. A software testing method in which the internal structure/design/implementation of the item is known to the tester is WHITE BOX TESTING.

* V SHAPED SOFTWARE LIFE CYCLE MODEL:

V model means Validation and Verification model just like Waterfall model., the V- shaped life cycle is a sequential path of execution of processes. Each phase must be completed before the next phase begins. Testing of the product is planned in parallel with a corresponding phase of development in V model. This model should be used for small to medium sized projects where requirements are clearly defined and fixed.

DIAGRAM OF V SHAPED MODEL:



* LIMITATIONS OF TESTING:
* Errors in the software requirement and specification document
* Logical bugs
* Difficult to measure the progress of testing
* Exhaustive (total) testing is not possible in present scenario
* Every path, valid inputs(all) and invalid input (all) cannot be tested at once
* QUALITY ASSURANCE AND QUALITY FACTOR:

**Software Quality Assurance** − Software Quality Assurance (SQA) is a set of activities to ensure the quality in software engineering processes that ultimately result in quality software products. The activities establish and evaluate the processes that produce products. It involves process-focused action.

**Software Quality Factor-** The various factors, which influence the software, are termed as software Quality factors.

They can be broadly divided into two categories.

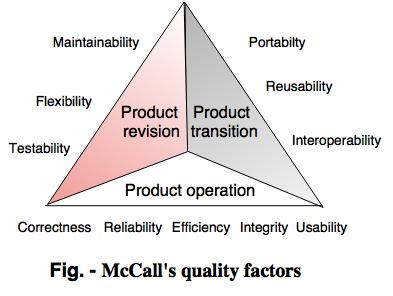
1. The first category of the factors is of those that can be measured directly such as the number of logical errors, and
2. The second category clubs those factors which can be measured only indirectly.

The various quality factors are:-

* correctness
* efficiency
* flexibility
* integrity
* interoperability
* maintainability
* portability
* reliability
* reusability
* testability
* usability
* MODELS OF QUALITY MANAGEMENT AND ITS CORE COMPONENT:

1. **Mc Call Software Quality Model** - This model classifies all software requirements into 11 software quality factors. The 11 factors are grouped into three categories – product operation, product revision, and product transition factors.

* **Product operation factors** − Correctness, Reliability, Efficiency, Integrity, Usability.
* **Product revision factors** − Maintainability, Flexibility, Testability.
* **Product transition factors** − Portability, Reusability, Interoperability.



Product Revision : This identifies quality factors that influence the ability to change the software product.  
(1) Maintainability : Effort required to locate and fix a fault in the program within its operating environment.  
(2) Flexibility : The ease of making changes required as dictated by business by changes in the operating environment.  
(3) Testability : The ease of testing program to ensure that it is error-free and meets its specification, i.e, validating the software requirements.  
Product Transition : This identifies quality factors that influence the ability to adapt the software to new environments.  
(1) Portability : The effort required to transfer a program from one environment to another.  
(2) Re-usability : The ease of reusing software in a different context.  
(3) Interoperability: The effort required to couple the system to another system.

Product Operations : This identifies quality factors that influence the extent to which the software fulfills its specification.  
(1) Correctness : The extent to which a functionality matches its specification.  
(2) Reliability : The system’s ability not to fail/ the extent to which the system fails.  
(3) Efficiency : Further categorized into execution efficiency and storage efficiency and generally means the usage of system resources, example: processor time, memory.  
(4) Integrity : The protection of program from unauthorized access.  
(5) Usability : The ease of using software.

1. **Boehm Software Quality Model** - Boehm’s model is similar to the McCall Quality Model in that it also presents a hierarchical quality model structured around **high-level characteristics, intermediate level characteristics, primitive characteristics** – each of which contributes to the overall quality level.

The **high-level characteristics** represent basic high-level requirements of actual use to which evaluation of software quality could be put – the general utility of software. The high-level characteristics address three main questions that a buyer of software has:  
• As-is utility : How well (easily, reliably, efficiently) can I use it as-is?  
• Maintainability: How easy is it to understand, modify and retest?  
• Portability : Can I still use it if I change my environment?

The **intermediate level characteristic** represents Boehm’s 7 quality factors that together represent the qualities expected from a software system:

• Portability

• Reliability

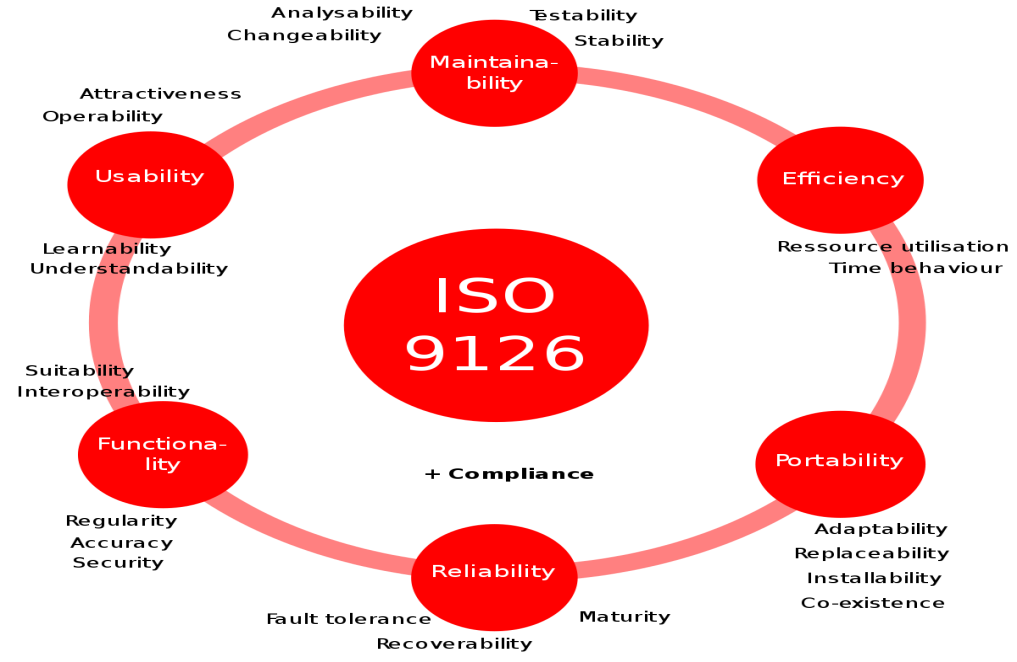
• Efficiency

• Usability

• Testability

• Understandability and Flexibility

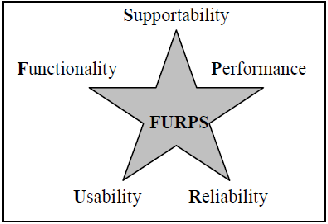
1. **ISO 9126** -ISO 9126 is an international standard for the evaluation of software.

****

The ISO 9126-1 software quality model identifies **6 main quality characteristics**, namely:

* Functionality
* Reliability
* Usability
* Efficiency
* Maintainability
* Portability

1. **Furps -** FURPS stands for **Functionality, Usability, Reliability, portability and Supportability.**

****

* TOTAL QUALITY MANAGEMENT:

**TQM** is a management approach to improve the quality. Key elements and the objectives of TQM system are:-

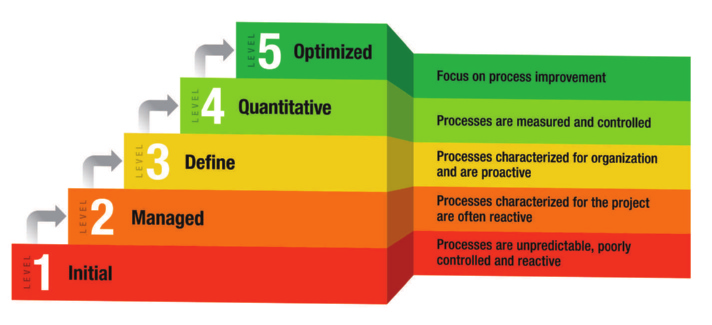
* 1. Customer focus – to achieve total customer satisfaction.
  2. Process improvement – to reduce process variations.
  3. Human side of quality – to create a companywide quality culture
  4. Metric, models, measurement, and analysis – to drive continuous improvement in all quality parameters
* ZERO DEFECT MOVEMENT:

Zero Defects is a management tool aimed at the reduction of defects through prevention. Zero Defects seeks to directly reverse the attitude that the number of mistakes a worker makes doesn't matter since inspectors will catch them before they reach the customer. Zero defects theory ensures that there is no waste existing in a project.

* CMM AND CMMI:

**CMM**

The Capability Maturity Model (CMM) is a methodology used to develop and refine an organization's software development process. The model describes a five-level. The higher the level, the better the software development process, hence reaching each level is an expensive and time-consuming process.



* 1. **Level One: Initial** - The software process is characterized as inconsistent, and occasionally even chaotic. Defined processes and standard practices that exist are abandoned during a crisis. Success of the organization majorly depends on an individual effort, talent, and heroics. The heroes eventually move on to other organizations taking their wealth of knowledge or lessons learnt with them.
  2. **Level Two: Repeatable** - This level of Software Development Organization has a basic and consistent project management processes to track cost, schedule, and functionality.
  3. **Level Three: Defined** - The software process for both management and engineering activities are documented, standardized, and integrated into a standard software process for the entire organization.
  4. **Level Four: Managed -** At this level, organization set a quantitative quality goal for both software process and software maintenance. At this maturity level, the performance of processes is controlled using statistical and other quantitative techniques, and is quantitatively predictable.
  5. **Level Five: Optimizing** - The Key characteristic of this level is focusing on continually improving process performance through both incremental and innovative technological improvements. Processes are constantly being improved through monitoring feedback from current processes and introducing innovative processes to better serve the organization's particular needs.

**CMMI**

The Capability Maturity Model Integration (CMMI) principal is that “the quality of a system or product is highly influenced by the process used to develop and maintain it”. It has 5 maturity levels:-

## Maturity Level 1 – Initial - At maturity level 1, processes are usually ad hoc and chaotic. The organization usually does not provide a stable environment.

## Maturity Level 2 – Managed - the projects of the organization have ensured that requirements are managed and that processes are planned, performed, measured, and controlled.

## Maturity Level 3 – Defined - processes are well characterized and understood, and are described in standards, procedures, tools, and methods.

## Maturity Level 4 - Quantitatively Managed - Sub processes are selected that significantly contribute to overall process performance. These selected sub processes are controlled using statistical and other quantitative techniques.

## Maturity Level 5 – Optimizing - Processes are continually improved based on a quantitative understanding of the common causes of variation inherent in processes. Maturity level 5 focuses on continually improving process performance through both incremental and innovative technological improvements.

* SIX SIGMA CONCEPTS:

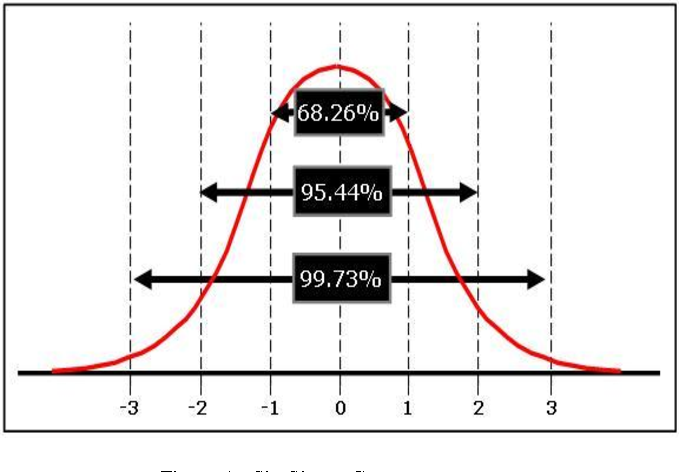
Six Sigma has been extensively used for achieving total customer satisfaction with innovative products at competitive price. Its objectives are to deliver products when promised, without delivered defects, early life failures, or failures during use.

Six Sigma uses the following three principles:

1. Focus on customers

2. Process orientation

3. Leadership based on metrics



Applying Six Sigma to software development makes software projects transparent to both management and customers.

The Six Sigma approach for Software is:

* Set the goal – Define
* Define the metrics – Measure
* Measure where you go – Analyse
* Improve your processes while you go – Improve
* Act immediately if going the wrong path – Control

Six Sigma offers six major **benefits** that attract companies −

* Generates sustained success
* Sets a performance goal for everyone
* Enhances value to customers
* Accelerates the rate of improvement
* Promotes learning and cross-pollination
* Executes strategic change
* QUALITY PLAN OBJECTIVE:

1. Scheduling development activities lead to successful and timely completion of project.
2. It helps in estimating the required manpower, resources and budget.
3. It helps in recruiting the team members and allocating resources to them.
4. Resolving development risks
5. Implementing required Software Quality Assurance activities.
6. Having requirements noted in the project quality plan helps the quality assurance team to correctly validate them.
7. The project quality plan has guidelines and instructions on how to manage the defects.
8. The project quality plan indicates the training requirements and necessary steps to get the staff trained.

**EXPERIMENT 2(a)**

**AIM:** Write a program to find the roots of the quadratic equation, its input is triple positive integers (say a, b, c) and the values may be from the interval [1,100] and perform Boundary Value Analysis (BVA).

**Program**

#include<iostream.h>

#include<conio.h>

#include<math.h>

void main()

{

clrscr();

int a,b,c,d;

float r1,r2;

cout<<"enter co-efficient of x^2 : ";

cin>>a;

cout<<"enter co-efficient of x : ";

cin>>b;

cout<<"enter the constant : ";

cin>>c;

if(a>100||b>100||c>100||a<1||b<1||c<1)

cout<<"out of range";

d=pow(b,2)-(4\*a\*c);

r1=(-b+sqrt(d))/(2\*a);

r2=(-b-sqrt(d))/(2\*a);

cout<<"root 1 = "<<r1;

cout<<"\nroot 2 = "<<r2;

getch();

}

**OUTPUT**



**BVA**

Range – R [1,100]

Domain – 1. Minimum value = 1

2. Above minimum value = 2

3. Nominal value = 50

4. Below maximum value = 99

5. Maximum value = 100

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **S.no.** | **Input-a** | **Input-b** | **Input-c** | **Expected output** | **Actual output** |
| 1 | 1 | 50 | 50 | -1.02084, -48.9792 | -1.02084, -48.9792 |
| 2 | 2 | 50 | 50 | -1.04356, -23.9564 | -1.04356, -23.9564 |
| **3** | **50** | **50** | **50** | **-0.5+0.866i, -0.5-0.866i** | **-1.#IND, -1.#IND** |
| 4 | 99 | 50 | 50 | -0.25+0.66i, -0.25-0.66i | -1.#IND, -1.#IND |
| 5 | 100 | 50 | 50 | -0.25+0.66i, -0.25-0.66i | -1.#IND, -1.#IND |
| 6 | 50 | 1 | 50 | -0.01+0.99i, -0.01-0.99i | -1.#IND, -1.#IND |
| 7 | 50 | 2 | 50 | -0.02+0.99i, -0.02-0.99i | -1.#IND, -1.#IND |
| **8** | **50** | **50** | **50** | **-0.5+0.866i, -0.5-0.866i** | **-1.#IND, -1.#IND** |
| 9 | 50 | 99 | 50 | -0.99+0.14i, -0.99-0.14i | -1.#IND, -1.#IND |
| 10 | 50 | 100 | 50 | -1, -1 | -1, -1 |
| 11 | 50 | 50 | 1 | -0.0204168, -0.979583 | -0.0204168, -0.979583 |
| 12 | 50 | 50 | 2 | -0.0417424, -0.958258 | -0.0417424, -0.958258 |
| **13** | **50** | **50** | **50** | **-0.5+0.866i, -0.5-0.866i** | **-1.#IND, -1.#IND** |
| 14 | 50 | 50 | 99 | -0.5+1.31i, -0.5-1.31i | -1.#IND, -1.#IND |
| 15 | 50 | 50 | 100 | -0.5+1.32i, -0.5-1.32i | -1.#IND, -1.#IND |

Test cases no. 3, 8 & 13 are redundant.

Therefore, total no. of test cases = 15-2 = 13 (4n-1 = 4\*3+1 = 13)

**EXPERIMENT 2(b)**

**AIM:** Write a program to find the roots of the quadratic equation, its input is triple positive integers (say a, b, c) and the values may be from the interval [1,100] and perform Robust Testing.

**Program**

#include<iostream.h>

#include<conio.h>

#include<math.h>

void main()

{

clrscr();

int a,b,c,d;

float r1,r2;

cout<<"enter co-efficient of x^2 : ";

cin>>a;

cout<<"enter co-efficient of x : ";

cin>>b;

cout<<"enter the constant : ";

cin>>c;

if(a>100||b>100||c>100||a<1||b<1||c<1)

cout<<"Invalid Input";

d=pow(b,2)-(4\*a\*c);

r1=(-b+sqrt(d))/(2\*a);

r2=(-b-sqrt(d))/(2\*a);

cout<<"root 1 = "<<r1;

cout<<"\nroot 2 = "<<r2;

getch();

}

**OUTPUT**



**Robust Testing**

Range – R [1,100]

Domain

1. Below Minimum value = 0
2. Minimum value = 1
3. Above minimum value = 2
4. Nominal value = 50
5. Below maximum value = 99
6. Maximum value = 100
7. Above Maximum = 101

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **S.no.** | **Input - a** | **Input - b** | **Input - c** | **Expected output** | **Actual output** |
| 1 | 0 | 50 | 50 | Invalid Input | Invalid Input |
| 2 | 1 | 50 | 50 | -1.02084, -48.9792 | -1.02084, -48.9792 |
| 3 | 2 | 50 | 50 | -1.04356, -23.9564 | -1.04356, -23.9564 |
| **4** | **50** | **50** | **50** | **-0.5+0.866i, -0.5-0.866i** | **-1.#IND, -1.#IND** |
| 5 | 99 | 50 | 50 | -0.25+0.66i, -0.25-0.66i | -1.#IND, -1.#IND |
| 6 | 100 | 50 | 50 | -0.25+0.66i, -0.25-0.66i | -1.#IND, -1.#IND |
| 7 | 101 | 50 | 50 | Invalid Input | Invalid Input |
| 8 | 50 | 0 | 50 | Invalid Input | Invalid Input |
| 9 | 50 | 1 | 50 | -0.01+0.99i, -0.01-0.99i | -1.#IND, -1.#IND |
| 10 | 50 | 2 | 50 | -0.02+0.99i, -0.02-0.99i | -1.#IND, -1.#IND |
| **11** | **50** | **50** | **50** | **-0.5+0.866i, -0.5-0.866i** | **-1.#IND, -1.#IND** |
| 12 | 50 | 99 | 50 | -0.99+0.14i, -0.99-0.14i | -1.#IND, -1.#IND |
| 13 | 50 | 100 | 50 | -1, -1 | -1, -1 |
| 14 | 50 | 101 | 50 | Invalid Input | Invalid Input |
| 15 | 50 | 0 | 0 | Invalid Input | Invalid Input |
| 16 | 50 | 50 | 1 | -0.0204168, -0.979583 | -0.0204168, -0.979583 |
| 17 | 50 | 50 | 2 | -0.0417424, -0.958258 | -0.0417424, -0.958258 |
| **18** | **50** | **50** | **50** | **-0.5+0.866i, -0.5-0.866i** | **-1.#IND, -1.#IND** |
| 19 | 50 | 50 | 99 | -0.5+1.31i, -0.5-1.31i | -1.#IND, -1.#IND |
| 20 | 50 | 50 | 100 | -0.5+1.32i, -0.5-1.32i | -1.#IND, -1.#IND |
| 21 | 50 | 50 | 101 | Invalid Input | Invalid Input |

Test cases no. 4, 11 & 18 are redundant.

Therefore, total no. of test cases = 21-2 = 13 (6n+1 = 6\*3+1 = 19

**EXPERIMENT 3(a)**

**AIM:** Write a program to find the largest among three numbers, its input is triple positive integers (say a, b, c) and the values may be from the interval [1,100] and perform BVA.

**Program**

#include <stdio.h>

#include <stdlib.h>

int main()

{

int a,b,c;

printf("\nEnter 3 No. A B C\n");

scanf("%d",&a);

scanf("%d",&b);

scanf("%d",&c);

if(a>100||b>100||c>100||a<1||b<1||c<1){

printf("Invalid value");

}else

if(a>b){

if(a>c){

printf("a=%d is largest",a);

}else

if(a=c){

printf("a=c=%d is largest",a);

}else

printf("c=%d is largest",c);

}else

if(a==b){

if(a==c){

printf("All are equal");

}else

if(a>c){

printf("a=b=%d is largest",a);

}else

printf("c=%d is largest",c);

}else

if(b>c){

printf("b=%d is largest",b);

}else

if(b=c){

printf("b=c=%d is largest",a);

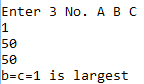
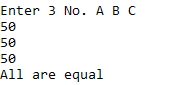
}else

printf("c=%d is largest",c);

return 0;

}

**OUTPUT**

**BVA**

Range – R [1,100]

Domain – 1. Minimum value = 1

2. Above minimum value = 2

3. Nominal value = 50

4. Below maximum value = 99

5. Maximum value = 100

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **S.no.** | **Input-a** | **Input-b** | **Input-c** | **Expected output** | **Actual output** |
| 1 | 1 | 50 | 50 | 50, b & c equal | 50, b & c equal |
| 2 | 2 | 50 | 50 | 50, b & c equal | 50, b & c equal |
| **3** | **50** | **50** | **50** | **All equal** | **All equal** |
| 4 | 99 | 50 | 50 | 99 | 99 |
| 5 | 100 | 50 | 50 | 100 | 100 |
| 6 | 50 | 1 | 50 | 50, a & c equal | 50, a & c equal |
| 7 | 50 | 2 | 50 | 50, a & c equal | 50, a & c equal |
| **8** | **50** | **50** | **50** | **All equal** | **All equal** |
| 9 | 50 | 99 | 50 | 99 | 99 |
| 10 | 50 | 100 | 50 | 100 | 100 |
| 11 | 50 | 50 | 1 | 50, a & b equal | 50, a & b equal |
| 12 | 50 | 50 | 2 | 50, a & b equal | 50, a & b equal |
| **13** | **50** | **50** | **50** | **All equal** | **All equal** |
| 14 | 50 | 50 | 99 | 99 | 99 |
| 15 | 50 | 50 | 100 | 100 | 100 |

Test cases no. 3, 8 & 13 are redundant.

Therefore, total no. of test cases = 15-2 = 13 (4n-1 = 4\*3+1 = 13)

**EXPERIMENT 3(b)**

**AIM:** Write a program to find the largest among three numbers, its input is triple positive integers (say a, b, c) and the values may be from the interval [1,100] and perform Robust Worst-Case Testing.

**Program**

#include <stdio.h>

#include <stdlib.h>

int main()

{

int a,b,c;

printf("\nEnter 3 No. A B C\n");

scanf("%d",&a);

scanf("%d",&b);

scanf("%d",&c);

if(a>100||b>100||c>100||a<1||b<1||c<1){

printf("Invalid value");

}else

if(a>b){

if(a>c){

printf("a=%d is largest",a);

}else

if(a=c){

printf("a=c=%d is largest",a);

}else

printf("c=%d is largest",c);

}else

if(a==b){

if(a==c){

printf("All are equal");

}else

if(a>c){

printf("a=b=%d is largest",a);

}else

printf("c=%d is largest",c);

}else

if(b>c){

printf("b=%d is largest",b);

}else

if(b=c){

printf("b=c=%d is largest",a);

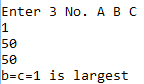
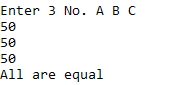
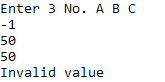
}else

printf("c=%d is largest",c);

return 0;

}

**OUTPUT**

**Robust Worst-Case testing**

Range – R [1,100]

Domain – 1. Below minimum value = 0

2. Minimum value = 1

3. Above minimum value = 2

4. Nominal value = 50

5. Below maximum value = 99

6. Maximum value = 100

7. Above maximum value = 101

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **S.no.** | **Input-a** | **Input-b** | **Input-c** | **Expected output** | **Actual output** |
| 1 | 0 | 0 | 0 | Invalid input | Invalid input |
| 2 | 0 | 0 | 1 | Invalid input | Invalid input |
| 3 | 0 | 0 | 2 | Invalid input | Invalid input |
| 4 | 0 | 0 | 50 | Invalid input | Invalid input |
| 5 | 0 | 0 | 99 | Invalid input | Invalid input |
| 6 | 0 | 0 | 100 | Invalid input | Invalid input |
| 7 | 0 | 0 | 101 | Invalid input | Invalid input |
| 8 | 0 | 1 | 0 | Invalid input | Invalid input |
| 9 | 0 | 1 | 1 | Invalid input | Invalid input |
| 10 | 0 | 1 | 2 | Invalid input | Invalid input |
| 11 | 0 | 1 | 50 | Invalid input | Invalid input |
| 12 | 0 | 1 | 99 | Invalid input | Invalid input |
| 13 | 0 | 1 | 100 | Invalid input | Invalid input |
| 14 | 0 | 1 | 101 | Invalid input | Invalid input |
| 15 | 0 | 2 | 0 | Invalid input | Invalid input |
| 16 | 0 | 2 | 1 | Invalid input | Invalid input |
| 17 | 0 | 2 | 2 | Invalid input | Invalid input |
| 18 | 0 | 2 | 50 | Invalid input | Invalid input |
| 19 | 0 | 2 | 99 | Invalid input | Invalid input |
| 20 | 0 | 2 | 100 | Invalid input | Invalid input |
| 21 | 0 | 2 | 101 | Invalid input | Invalid input |
| 22 | 0 | 50 | 0 | Invalid input | Invalid input |
| 23 | 0 | 50 | 1 | Invalid input | Invalid input |
| 24 | 0 | 50 | 2 | Invalid input | Invalid input |
| 25 | 0 | 50 | 50 | Invalid input | Invalid input |
| 26 | 0 | 50 | 99 | Invalid input | Invalid input |
| 27 | 0 | 50 | 100 | Invalid input | Invalid input |
| 28 | 0 | 50 | 101 | Invalid input | Invalid input |
| 29 | 0 | 99 | 0 | Invalid input | Invalid input |
| 30 | 0 | 99 | 1 | Invalid input | Invalid input |
| 31 | 0 | 99 | 2 | Invalid input | Invalid input |
| 32 | 0 | 99 | 50 | Invalid input | Invalid input |
| 33 | 0 | 99 | 99 | Invalid input | Invalid input |
| 34 | 0 | 99 | 100 | Invalid input | Invalid input |
| 35 | 0 | 99 | 101 | Invalid input | Invalid input |
| 36 | 0 | 100 | 0 | Invalid input | Invalid input |
| 37 | 0 | 100 | 1 | Invalid input | Invalid input |
| 38 | 0 | 100 | 2 | Invalid input | Invalid input |
| 39 | 0 | 100 | 50 | Invalid input | Invalid input |
| 40 | 0 | 100 | 99 | Invalid input | Invalid input |
| 41 | 0 | 100 | 100 | Invalid input | Invalid input |
| 42 | 0 | 100 | 101 | Invalid input | Invalid input |
| 43 | 0 | 101 | 0 | Invalid input | Invalid input |
| 44 | 0 | 101 | 1 | Invalid input | Invalid input |
| 45 | 0 | 101 | 2 | Invalid input | Invalid input |
| 46 | 0 | 101 | 50 | Invalid input | Invalid input |
| 47 | 0 | 101 | 99 | Invalid input | Invalid input |
| 48 | 0 | 101 | 100 | Invalid input | Invalid input |
| 49 | 0 | 101 | 101 | Invalid input | Invalid input |
| 50 | 1 | 0 | 0 | Invalid input | Invalid input |
| 51 | 1 | 0 | 1 | Invalid input | Invalid input |
| 52 | 1 | 0 | 2 | Invalid input | Invalid input |
| 53 | 1 | 0 | 50 | Invalid input | Invalid input |
| 54 | 1 | 0 | 99 | Invalid input | Invalid input |
| 55 | 1 | 0 | 100 | Invalid input | Invalid input |
| 56 | 1 | 0 | 101 | Invalid input | Invalid input |
| 57 | 1 | 1 | 0 | Invalid input | Invalid input |
| 58 | 1 | 1 | 1 | All equal | All equal |
| 59 | 1 | 1 | 2 | 2 | 2 |
| 60 | 1 | 1 | 50 | 50 | 50 |
| 61 | 1 | 1 | 99 | 99 | 99 |
| 62 | 1 | 1 | 100 | 100 | 100 |
| 63 | 1 | 1 | 101 | Invalid input | Invalid input |
| 64 | 1 | 2 | 0 | Invalid input | Invalid input |
| 65 | 1 | 2 | 1 | 2 | 2 |
| 66 | 1 | 2 | 2 | 2, b & c equal | 2, b & c equal |
| 67 | 1 | 2 | 50 | 50 | 50 |
| 68 | 1 | 2 | 99 | 99 | 99 |
| 69 | 1 | 2 | 100 | 100 | 100 |
| 70 | 1 | 2 | 101 | Invalid input | Invalid input |
| 71 | 1 | 50 | 0 | Invalid input | Invalid input |
| 72 | 1 | 50 | 1 | 50 | 50 |
| 73 | 1 | 50 | 2 | 50 | 50 |
| 74 | 1 | 50 | 50 | 50, b & c equal | 50, b & c equal |
| 75 | 1 | 50 | 99 | 99 | 99 |
| 76 | 1 | 50 | 100 | 100 | 100 |
| 77 | 1 | 50 | 101 | Invalid input | Invalid input |
| 78 | 1 | 99 | 0 | Invalid input | Invalid input |
| 79 | 1 | 99 | 1 | 99 | 99 |
| 80 | 1 | 99 | 2 | 99 | 99 |
| 81 | 1 | 99 | 50 | 99 | 99 |
| 82 | 1 | 99 | 99 | 99, b & c equal | 99, b & c equal |
| 83 | 1 | 99 | 100 | 100 | 100 |
| 84 | 1 | 99 | 101 | Invalid input | Invalid input |
| 85 | 1 | 100 | 0 | Invalid input | Invalid input |
| 86 | 1 | 100 | 1 | 100 | 100 |
| 87 | 1 | 100 | 2 | 100 | 100 |
| 88 | 1 | 100 | 50 | 100 | 100 |
| 89 | 1 | 100 | 99 | 100 | 100 |
| 90 | 1 | 100 | 100 | 100, b & c equal | 100, b & c equal |
| 91 | 1 | 100 | 101 | Invalid input | Invalid input |
| 92 | 1 | 101 | 0 | Invalid input | Invalid input |
| 93 | 1 | 101 | 1 | Invalid input | Invalid input |
| 94 | 1 | 101 | 2 | Invalid input | Invalid input |
| 95 | 1 | 101 | 50 | Invalid input | Invalid input |
| 96 | 1 | 101 | 99 | Invalid input | Invalid input |
| 97 | 1 | 101 | 100 | Invalid input | Invalid input |
| 98 | 1 | 101 | 101 | Invalid input | Invalid input |
| 99 | 2 | 0 | 0 | Invalid input | Invalid input |
| 100 | 2 | 0 | 1 | Invalid input | Invalid input |
| 101 | 2 | 0 | 2 | Invalid input | Invalid input |
| 102 | 2 | 0 | 50 | Invalid input | Invalid input |
| 103 | 2 | 0 | 99 | Invalid input | Invalid input |
| 104 | 2 | 0 | 100 | Invalid input | Invalid input |
| 105 | 2 | 0 | 101 | Invalid input | Invalid input |
| 106 | 2 | 1 | 0 | Invalid input | Invalid input |
| 107 | 2 | 1 | 1 | 2 | 2 |
| 108 | 2 | 1 | 2 | 2, a & c equal | 2, a & c equal |
| 109 | 2 | 1 | 50 | 50 | 50 |
| 110 | 2 | 1 | 99 | 99 | 99 |
| 111 | 2 | 1 | 100 | 100 | 100 |
| 112 | 2 | 1 | 101 | Invalid input | Invalid input |
| 113 | 2 | 2 | 0 | Invalid input | Invalid input |
| 114 | 2 | 2 | 1 | 2, a & b equal | 2, a & b equal |
| 115 | 2 | 2 | 2 | All equal | All equal |
| 116 | 2 | 2 | 50 | 50 | 50 |
| 117 | 2 | 2 | 99 | 99 | 99 |
| 118 | 2 | 2 | 100 | 100 | 100 |
| 119 | 2 | 2 | 101 | Invalid input | Invalid input |
| 120 | 2 | 50 | 0 | Invalid input | Invalid input |
| 121 | 2 | 50 | 1 | 50 | 50 |
| 122 | 2 | 50 | 2 | 50 | 50 |
| 123 | 2 | 50 | 50 | 50, b & c equal | 50, b & c equal |
| 124 | 2 | 50 | 99 | 99 | 99 |
| 125 | 2 | 50 | 100 | 100 | 100 |
| 126 | 2 | 50 | 101 | Invalid input | Invalid input |
| 127 | 2 | 99 | 0 | Invalid input | Invalid input |
| 128 | 2 | 99 | 1 | 99 | 99 |
| 129 | 2 | 99 | 2 | 99 | 99 |
| 130 | 2 | 99 | 50 | 99 | 99 |
| 131 | 2 | 99 | 99 | 99, b & c equal | 99, b & c equal |
| 132 | 2 | 99 | 100 | 100 | 100 |
| 133 | 2 | 99 | 101 | Invalid input | Invalid input |
| 134 | 2 | 100 | 0 | Invalid input | Invalid input |
| 135 | 2 | 100 | 1 | 100 | 100 |
| 136 | 2 | 100 | 2 | 100 | 100 |
| 137 | 2 | 100 | 50 | 100 | 100 |
| 138 | 2 | 100 | 99 | 100 | 100 |
| 139 | 2 | 100 | 100 | 100, b & c equal | 100, b & c equal |
| 140 | 2 | 100 | 101 | Invalid input | Invalid input |
| 141 | 2 | 101 | 0 | Invalid input | Invalid input |
| 142 | 2 | 101 | 1 | Invalid input | Invalid input |
| 143 | 2 | 101 | 2 | Invalid input | Invalid input |
| 144 | 2 | 101 | 50 | Invalid input | Invalid input |
| 145 | 2 | 101 | 99 | Invalid input | Invalid input |
| 146 | 2 | 101 | 100 | Invalid input | Invalid input |
| 147 | 2 | 101 | 101 | Invalid input | Invalid input |
| 148 | 50 | 0 | 0 | Invalid input | Invalid input |
| 149 | 50 | 0 | 1 | Invalid input | Invalid input |
| 150 | 50 | 0 | 2 | Invalid input | Invalid input |
| 151 | 50 | 0 | 50 | Invalid input | Invalid input |
| 152 | 50 | 0 | 99 | Invalid input | Invalid input |
| 153 | 50 | 0 | 100 | Invalid input | Invalid input |
| 154 | 50 | 0 | 101 | Invalid input | Invalid input |
| 155 | 50 | 1 | 0 | Invalid input | Invalid input |
| 156 | 50 | 1 | 1 | 50 | 50 |
| 157 | 50 | 1 | 2 | 50 | 50 |
| 158 | 50 | 1 | 50 | 50, a & c equal | 50, a & c equal |
| 159 | 50 | 1 | 99 | 99 | 99 |
| 160 | 50 | 1 | 100 | 100 | 100 |
| 161 | 50 | 1 | 101 | Invalid input | Invalid input |
| 162 | 50 | 2 | 0 | Invalid input | Invalid input |
| 163 | 50 | 2 | 1 | 50 | 50 |
| 164 | 50 | 2 | 2 | 50 | 50 |
| 165 | 50 | 2 | 50 | 50, a & c equal | 50, a & c equal |
| 166 | 50 | 2 | 99 | 99 | 99 |
| 167 | 50 | 2 | 100 | 100 | 100 |
| 168 | 50 | 2 | 101 | Invalid input | Invalid input |
| 169 | 50 | 50 | 0 | Invalid input | Invalid input |
| 170 | 50 | 50 | 1 | 50, a & b equal | 50, a & b equal |
| 171 | 50 | 50 | 2 | 50, a & b equal | 50, a & b equal |
| 172 | 50 | 50 | 50 | All equal | All equal |
| 173 | 50 | 50 | 99 | 99 | 99 |
| 174 | 50 | 50 | 100 | 100 | 100 |
| 175 | 50 | 50 | 101 | Invalid input | Invalid input |
| 176 | 50 | 99 | 0 | Invalid input | Invalid input |
| 177 | 50 | 99 | 1 | 99 | 99 |
| 178 | 50 | 99 | 2 | 99 | 99 |
| 179 | 50 | 99 | 50 | 99 | 99 |
| 180 | 50 | 99 | 99 | 99, b & c equal | 99, b & c equal |
| 181 | 50 | 99 | 100 | 100 | 100 |
| 182 | 50 | 99 | 101 | Invalid input | Invalid input |
| 183 | 50 | 100 | 0 | Invalid input | Invalid input |
| 184 | 50 | 100 | 1 | 100 | 100 |
| 185 | 50 | 100 | 2 | 100 | 100 |
| 186 | 50 | 100 | 50 | 100 | 100 |
| 187 | 50 | 100 | 99 | 100 | 100 |
| 188 | 50 | 100 | 100 | 100, b & c equal | 100, b & c equal |
| 189 | 50 | 100 | 101 | Invalid input | Invalid input |
| 190 | 50 | 101 | 0 | Invalid input | Invalid input |
| 191 | 50 | 101 | 1 | Invalid input | Invalid input |
| 192 | 50 | 101 | 2 | Invalid input | Invalid input |
| 193 | 50 | 101 | 50 | Invalid input | Invalid input |
| 194 | 50 | 101 | 99 | Invalid input | Invalid input |
| 195 | 50 | 101 | 100 | Invalid input | Invalid input |
| 196 | 50 | 101 | 101 | Invalid input | Invalid input |
| 197 | 99 | 0 | 0 | Invalid input | Invalid input |
| 198 | 99 | 0 | 1 | Invalid input | Invalid input |
| 199 | 99 | 0 | 2 | Invalid input | Invalid input |
| 200 | 99 | 0 | 50 | Invalid input | Invalid input |
| 201 | 99 | 0 | 99 | Invalid input | Invalid input |
| 202 | 99 | 0 | 100 | Invalid input | Invalid input |
| 203 | 99 | 0 | 101 | Invalid input | Invalid input |
| 204 | 99 | 1 | 0 | Invalid input | Invalid input |
| 205 | 99 | 1 | 1 | 99 | 99 |
| 206 | 99 | 1 | 2 | 99 | 99 |
| 207 | 99 | 1 | 50 | 99 | 99 |
| 208 | 99 | 1 | 99 | 99, a & c equal | 99, a & c equal |
| 209 | 99 | 1 | 100 | 100 | 100 |
| 210 | 99 | 1 | 101 | Invalid input | Invalid input |
| 211 | 99 | 2 | 0 | Invalid input | Invalid input |
| 212 | 99 | 2 | 1 | 99 | 99 |
| 213 | 99 | 2 | 2 | 99 | 99 |
| 214 | 99 | 2 | 50 | 99 | 99 |
| 215 | 99 | 2 | 99 | 99, a & c equal | 99, a & c equal |
| 216 | 99 | 2 | 100 | 100 | 100 |
| 217 | 99 | 2 | 101 | Invalid input | Invalid input |
| 218 | 99 | 50 | 0 | Invalid input | Invalid input |
| 219 | 99 | 50 | 1 | 99 | 99 |
| 220 | 99 | 50 | 2 | 99 | 99 |
| 221 | 99 | 50 | 50 | 99 | 99 |
| 222 | 99 | 50 | 99 | 99, a & c equal | 99, a & c equal |
| 223 | 99 | 50 | 100 | 100 | 100 |
| 224 | 99 | 50 | 101 | Invalid input | Invalid input |
| 225 | 99 | 99 | 0 | Invalid input | Invalid input |
| 226 | 99 | 99 | 1 | 99, a & b equal | 99, a & b equal |
| 227 | 99 | 99 | 2 | 99, a & b equal | 99, a & b equal |
| 228 | 99 | 99 | 50 | 99, a & b equal | 99, a & b equal |
| 229 | 99 | 99 | 99 | All equal | All equal |
| 230 | 99 | 99 | 100 | 100 | 100 |
| 231 | 99 | 99 | 101 | Invalid input | Invalid input |
| 232 | 99 | 100 | 0 | Invalid input | Invalid input |
| 233 | 99 | 100 | 1 | 100 | 100 |
| 234 | 99 | 100 | 2 | 100 | 100 |
| 235 | 99 | 100 | 50 | 100 | 100 |
| 236 | 99 | 100 | 99 | 100 | 100 |
| 237 | 99 | 100 | 100 | 100, b & c equal | 100, b & c equal |
| 238 | 99 | 100 | 101 | Invalid input | Invalid input |
| 239 | 99 | 101 | 0 | Invalid input | Invalid input |
| 240 | 99 | 101 | 1 | Invalid input | Invalid input |
| 241 | 99 | 101 | 2 | Invalid input | Invalid input |
| 242 | 99 | 101 | 50 | Invalid input | Invalid input |
| 243 | 99 | 101 | 99 | Invalid input | Invalid input |
| 244 | 99 | 101 | 100 | Invalid input | Invalid input |
| 245 | 99 | 101 | 101 | Invalid input | Invalid input |
| 246 | 100 | 0 | 0 | Invalid input | Invalid input |
| 247 | 100 | 0 | 1 | Invalid input | Invalid input |
| 248 | 100 | 0 | 2 | Invalid input | Invalid input |
| 249 | 100 | 0 | 50 | Invalid input | Invalid input |
| 250 | 100 | 0 | 99 | Invalid input | Invalid input |
| 251 | 100 | 0 | 100 | Invalid input | Invalid input |
| 252 | 100 | 0 | 101 | Invalid input | Invalid input |
| 253 | 100 | 1 | 0 | Invalid input | Invalid input |
| 254 | 100 | 1 | 1 | 100 | 100 |
| 255 | 100 | 1 | 2 | 100 | 100 |
| 256 | 100 | 1 | 50 | 100 | 100 |
| 257 | 100 | 1 | 99 | 100 | 100 |
| 258 | 100 | 1 | 100 | 100, a & c equal | 100, a & c equal |
| 259 | 100 | 1 | 101 | Invalid input | Invalid input |
| 260 | 100 | 2 | 0 | Invalid input | Invalid input |
| 261 | 100 | 2 | 1 | 100 | 100 |
| 262 | 100 | 2 | 2 | 100 | 100 |
| 263 | 100 | 2 | 50 | 100 | 100 |
| 264 | 100 | 2 | 99 | 100 | 100 |
| 265 | 100 | 2 | 100 | 100, a & c equal | 100, a & c equal |
| 266 | 100 | 2 | 101 | Invalid input | Invalid input |
| 267 | 100 | 50 | 0 | Invalid input | Invalid input |
| 268 | 100 | 50 | 1 | 100 | 100 |
| 269 | 100 | 50 | 2 | 100 | 100 |
| 270 | 100 | 50 | 50 | 100 | 100 |
| 271 | 100 | 50 | 99 | 100 | 100 |
| 272 | 100 | 50 | 100 | 100, a & c equal | 100, a & c equal |
| 273 | 100 | 50 | 101 | Invalid input | Invalid input |
| 274 | 100 | 99 | 0 | Invalid input | Invalid input |
| 275 | 100 | 99 | 1 | 100 | 100 |
| 276 | 100 | 99 | 2 | 100 | 100 |
| 277 | 100 | 99 | 50 | 100 | 100 |
| 278 | 100 | 99 | 99 | 100 | 100 |
| 279 | 100 | 99 | 100 | 100, a & c equal | 100, a & c equal |
| 280 | 100 | 99 | 101 | Invalid input | Invalid input |
| 281 | 100 | 100 | 0 | Invalid input | Invalid input |
| 282 | 100 | 100 | 1 | Equal | - |
| 283 | 100 | 100 | 2 | Equal | - |
| 284 | 100 | 100 | 50 | Equal | - |
| 285 | 100 | 100 | 99 | Equal | - |
| 286 | 100 | 100 | 100 | All equal | All equal |
| 287 | 100 | 100 | 101 | Invalid input | Invalid input |
| 288 | 100 | 101 | 0 | Invalid input | Invalid input |
| 289 | 100 | 101 | 1 | Invalid input | Invalid input |
| 290 | 100 | 101 | 2 | Invalid input | Invalid input |
| 291 | 100 | 101 | 50 | Invalid input | Invalid input |
| 292 | 100 | 101 | 99 | Invalid input | Invalid input |
| 293 | 100 | 101 | 100 | Invalid input | Invalid input |
| 294 | 100 | 101 | 101 | Invalid input | Invalid input |
| 295 | 101 | 0 | 0 | Invalid input | Invalid input |
| 296 | 101 | 0 | 1 | Invalid input | Invalid input |
| 297 | 101 | 0 | 2 | Invalid input | Invalid input |
| 298 | 101 | 0 | 50 | Invalid input | Invalid input |
| 299 | 101 | 0 | 99 | Invalid input | Invalid input |
| 300 | 101 | 0 | 100 | Invalid input | Invalid input |
| 301 | 101 | 0 | 101 | Invalid input | Invalid input |
| 302 | 101 | 1 | 0 | Invalid input | Invalid input |
| 303 | 101 | 1 | 1 | Invalid input | Invalid input |
| 304 | 101 | 1 | 2 | Invalid input | Invalid input |
| 305 | 101 | 1 | 50 | Invalid input | Invalid input |
| 306 | 101 | 1 | 99 | Invalid input | Invalid input |
| 307 | 101 | 1 | 100 | Invalid input | Invalid input |
| 308 | 101 | 1 | 101 | Invalid input | Invalid input |
| 309 | 101 | 2 | 0 | Invalid input | Invalid input |
| 310 | 101 | 2 | 1 | Invalid input | Invalid input |
| 311 | 101 | 2 | 2 | Invalid input | Invalid input |
| 312 | 101 | 2 | 50 | Invalid input | Invalid input |
| 313 | 101 | 2 | 99 | Invalid input | Invalid input |
| 314 | 101 | 2 | 100 | Invalid input | Invalid input |
| 315 | 101 | 2 | 101 | Invalid input | Invalid input |
| 316 | 101 | 50 | 0 | Invalid input | Invalid input |
| 317 | 101 | 50 | 1 | Invalid input | Invalid input |
| 318 | 101 | 50 | 2 | Invalid input | Invalid input |
| 319 | 101 | 50 | 50 | Invalid input | Invalid input |
| 320 | 101 | 50 | 99 | Invalid input | Invalid input |
| 321 | 101 | 50 | 100 | Invalid input | Invalid input |
| 322 | 101 | 50 | 101 | Invalid input | Invalid input |
| 323 | 101 | 99 | 0 | Invalid input | Invalid input |
| 324 | 101 | 99 | 1 | Invalid input | Invalid input |
| 325 | 101 | 99 | 2 | Invalid input | Invalid input |
| 326 | 101 | 99 | 50 | Invalid input | Invalid input |
| 327 | 101 | 99 | 99 | Invalid input | Invalid input |
| 328 | 101 | 99 | 100 | Invalid input | Invalid input |
| 329 | 101 | 99 | 101 | Invalid input | Invalid input |
| 330 | 101 | 100 | 0 | Invalid input | Invalid input |
| 331 | 101 | 100 | 1 | Invalid input | Invalid input |
| 332 | 101 | 100 | 2 | Invalid input | Invalid input |
| 333 | 101 | 100 | 50 | Invalid input | Invalid input |
| 334 | 101 | 100 | 99 | Invalid input | Invalid input |
| 335 | 101 | 100 | 100 | Invalid input | Invalid input |
| 336 | 101 | 100 | 101 | Invalid input | Invalid input |
| 337 | 101 | 101 | 0 | Invalid input | Invalid input |
| 338 | 101 | 101 | 1 | Invalid input | Invalid input |
| 339 | 101 | 101 | 2 | Invalid input | Invalid input |
| 340 | 101 | 101 | 50 | Invalid input | Invalid input |
| 341 | 101 | 101 | 99 | Invalid input | Invalid input |
| 342 | 101 | 101 | 100 | Invalid input | Invalid input |
| 343 | 101 | 101 | 101 | Invalid input | Invalid input |

Total no. of test cases = 343 (7^3 = 343)

**EXPERIMENT 4(a)**

**AIM:** Write a program to determine the type of triangle, its input is triple positive integers (say a, b, c) and the values may be from the interval [1,100]. The program output may be one of the following [Scalene, Isosceles, Equilateral, Not a triangle]. Perform BVA.

**Program**

#include <stdio.h>

void main(){

int a,b,c;

printf("\nEnter three sides of triangle A,B,C\n");

scanf("%d",&a);

scanf("%d",&b);

scanf("%d",&c);

if(a>100||b>100||c>100||a<1||b<1||c<1){

printf("Invalid Input\n");

}else

if((a+b)>c || (b+c)>a || (a+c)>b ){

if(a==b && b==c){

printf("Equilateral\n");

}else

if(a==b||b==c||a==c){

printf("Isoceles\n");

}else

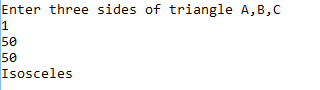
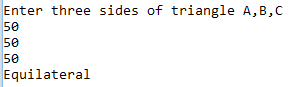
printf("Scaline\n");

}else

printf("No Triangle can be formed\n");

}

**OUTPUT**

**BVA**

Range – R [1,100]

Domain – 1. Minimum value = 1

2. Above minimum value = 2

3. Nominal value = 50

4. Below maximum value = 99

5. Maximum value = 100

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **S.no.** | **Input-a** | **Input-b** | **Input-c** | **Expected output** | **Actual output** |
| 1 | 1 | 50 | 50 | Isosceles | Isosceles |
| 2 | 2 | 50 | 50 | Isosceles | Isosceles |
| **3** | **50** | **50** | **50** | **Equilateral** | **Equilateral** |
| 4 | 99 | 50 | 50 | Isosceles | Isosceles |
| 5 | 100 | 50 | 50 | Isosceles | Isosceles |
| 6 | 50 | 1 | 50 | Isosceles | Isosceles |
| 7 | 50 | 2 | 50 | Isosceles | Isosceles |
| **8** | **50** | **50** | **50** | **Equilateral** | **Equilateral** |
| 9 | 50 | 99 | 50 | Isosceles | Isosceles |
| 10 | 50 | 100 | 50 | Isosceles | Isosceles |
| 11 | 50 | 50 | 1 | Isosceles | Isosceles |
| 12 | 50 | 50 | 2 | Isosceles | Isosceles |
| **13** | **50** | **50** | **50** | **Equilateral** | **Equilateral** |
| 14 | 50 | 50 | 99 | Isosceles | Isosceles |
| 15 | 50 | 50 | 100 | Isosceles | Isosceles |

Test cases no. 3, 8 & 13 are redundant.

Therefore, total no. of test cases = 15-2 = 13 (4n-1 = 4\*3+1 = 13)

**EXPERIMENT 4(b)**

**AIM:** Write a program to determine the type of triangle, its input is triple positive integers (say a, b, c) and the values may be from the interval [1,100]. The program output may be one of the following [Scalene, Isosceles, Equilateral, Not a triangle]. Perform Worst-Case Testing.

**Program**

#include <stdio.h>

void main(){

int a,b,c;

printf("\nEnter three sides of triangle A,B,C\n");

scanf("%d",&a);

scanf("%d",&b);

scanf("%d",&c);

if(a>100||b>100||c>100||a<1||b<1||c<1){

printf("Invalid Input\n");

}else

if((a+b)>c || (b+c)>a || (a+c)>b ){

if(a==b && b==c){

printf("Equilateral\n");

}else

if(a==b||b==c||a==c){

printf("Isosceles\n");

}else

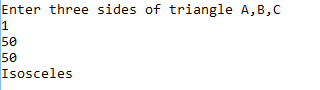
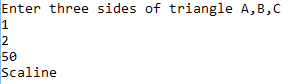
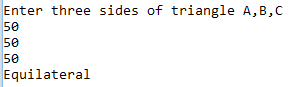
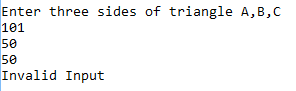
printf("Scalene\n");

}else

printf("No Triangle can be formed\n");

}

**OUTPUT**

**Robust Worst-Case testing**

Range – R [1,100]

Domain – 1. Below minimum value = 0

2. Minimum value = 1

3. Above minimum value = 2

4. Nominal value = 50

5. Below maximum value = 99

6. Maximum value = 100

7. Above maximum value = 101

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **S.no.** | **Input-a** | **Input-b** | **Input-c** | **Expected output** | **Actual output** |
| 1 | 0 | 0 | 0 | Invalid Input | Invalid Input |
| 2 | 0 | 0 | 1 | Invalid Input | Invalid Input |
| 3 | 0 | 0 | 2 | Invalid Input | Invalid Input |
| 4 | 0 | 0 | 50 | Invalid Input | Invalid Input |
| 5 | 0 | 0 | 99 | Invalid Input | Invalid Input |
| 6 | 0 | 0 | 100 | Invalid Input | Invalid Input |
| 7 | 0 | 0 | 101 | Invalid Input | Invalid Input |
| 8 | 0 | 1 | 0 | Invalid Input | Invalid Input |
| 9 | 0 | 1 | 1 | Invalid Input | Invalid Input |
| 10 | 0 | 1 | 2 | Invalid Input | Invalid Input |
| 11 | 0 | 1 | 50 | Invalid Input | Invalid Input |
| 12 | 0 | 1 | 99 | Invalid Input | Invalid Input |
| 13 | 0 | 1 | 100 | Invalid Input | Invalid Input |
| 14 | 0 | 1 | 101 | Invalid Input | Invalid Input |
| 15 | 0 | 2 | 0 | Invalid Input | Invalid Input |
| 16 | 0 | 2 | 1 | Invalid Input | Invalid Input |
| 17 | 0 | 2 | 2 | Invalid Input | Invalid Input |
| 18 | 0 | 2 | 50 | Invalid Input | Invalid Input |
| 19 | 0 | 2 | 99 | Invalid Input | Invalid Input |
| 20 | 0 | 2 | 100 | Invalid Input | Invalid Input |
| 21 | 0 | 2 | 101 | Invalid Input | Invalid Input |
| 22 | 0 | 50 | 0 | Invalid Input | Invalid Input |
| 23 | 0 | 50 | 1 | Invalid Input | Invalid Input |
| 24 | 0 | 50 | 2 | Invalid Input | Invalid Input |
| 25 | 0 | 50 | 50 | Invalid Input | Invalid Input |
| 26 | 0 | 50 | 99 | Invalid Input | Invalid Input |
| 27 | 0 | 50 | 100 | Invalid Input | Invalid Input |
| 28 | 0 | 50 | 101 | Invalid Input | Invalid Input |
| 29 | 0 | 99 | 0 | Invalid Input | Invalid Input |
| 30 | 0 | 99 | 1 | Invalid Input | Invalid Input |
| 31 | 0 | 99 | 2 | Invalid Input | Invalid Input |
| 32 | 0 | 99 | 50 | Invalid Input | Invalid Input |
| 33 | 0 | 99 | 99 | Invalid Input | Invalid Input |
| 34 | 0 | 99 | 100 | Invalid Input | Invalid Input |
| 35 | 0 | 99 | 101 | Invalid Input | Invalid Input |
| 36 | 0 | 100 | 0 | Invalid Input | Invalid Input |
| 37 | 0 | 100 | 1 | Invalid Input | Invalid Input |
| 38 | 0 | 100 | 2 | Invalid Input | Invalid Input |
| 39 | 0 | 100 | 50 | Invalid Input | Invalid Input |
| 40 | 0 | 100 | 99 | Invalid Input | Invalid Input |
| 41 | 0 | 100 | 100 | Invalid Input | Invalid Input |
| 42 | 0 | 100 | 101 | Invalid Input | Invalid Input |
| 43 | 0 | 101 | 0 | Invalid Input | Invalid Input |
| 44 | 0 | 101 | 1 | Invalid Input | Invalid Input |
| 45 | 0 | 101 | 2 | Invalid Input | Invalid Input |
| 46 | 0 | 101 | 50 | Invalid Input | Invalid Input |
| 47 | 0 | 101 | 99 | Invalid Input | Invalid Input |
| 48 | 0 | 101 | 100 | Invalid Input | Invalid Input |
| 49 | 0 | 101 | 101 | Invalid Input | Invalid Input |
| 50 | 1 | 0 | 0 | Invalid Input | Invalid Input |
| 51 | 1 | 0 | 1 | Invalid Input | Invalid Input |
| 52 | 1 | 0 | 2 | Invalid Input | Invalid Input |
| 53 | 1 | 0 | 50 | Invalid Input | Invalid Input |
| 54 | 1 | 0 | 99 | Invalid Input | Invalid Input |
| 55 | 1 | 0 | 100 | Invalid Input | Invalid Input |
| 56 | 1 | 0 | 101 | Invalid Input | Invalid Input |
| 57 | 1 | 1 | 0 | Invalid Input | Invalid Input |
| 58 | 1 | 1 | 1 | Equilateral | Equilateral |
| 59 | 1 | 1 | 2 | Isosceles | Isosceles |
| 60 | 1 | 1 | 50 | Isosceles | Isosceles |
| 61 | 1 | 1 | 99 | Isosceles | Isosceles |
| 62 | 1 | 1 | 100 | Isosceles | Isosceles |
| 63 | 1 | 1 | 101 | Isosceles | Isosceles |
| 64 | 1 | 2 | 0 | Invalid Input | Invalid Input |
| 65 | 1 | 2 | 1 | Isosceles | Isosceles |
| 66 | 1 | 2 | 2 | Isosceles | Isosceles |
| 67 | 1 | 2 | 50 | Scalene | Scalene |
| 68 | 1 | 2 | 99 | Scalene | Scalene |
| 69 | 1 | 2 | 100 | Scalene | Scalene |
| 70 | 1 | 2 | 101 | Invalid Input | Invalid Input |
| 71 | 1 | 50 | 0 | Invalid Input | Invalid Input |
| 72 | 1 | 50 | 1 | Isosceles | Isosceles |
| 73 | 1 | 50 | 2 | Scalene | Scalene |
| 74 | 1 | 50 | 50 | Isosceles | Isosceles |
| 75 | 1 | 50 | 99 | Scalene | Scalene |
| 76 | 1 | 50 | 100 | Scalene | Scalene |
| 77 | 1 | 50 | 101 | Invalid Input | Invalid Input |
| 78 | 1 | 99 | 0 | Invalid Input | Invalid Input |
| 79 | 1 | 99 | 1 | Isosceles | Isosceles |
| 80 | 1 | 99 | 2 | Scalene | Scalene |
| 81 | 1 | 99 | 50 | Scalene | Scalene |
| 82 | 1 | 99 | 99 | Isosceles | Isosceles |
| 83 | 1 | 99 | 100 | Scalene | Scalene |
| 84 | 1 | 99 | 101 | Invalid Input | Invalid Input |
| 85 | 1 | 100 | 0 | Invalid Input | Invalid Input |
| 86 | 1 | 100 | 1 | Isosceles | Isosceles |
| 87 | 1 | 100 | 2 | Scalene | Scalene |
| 88 | 1 | 100 | 50 | Scalene | Scalene |
| 89 | 1 | 100 | 99 | Scalene | Scalene |
| 90 | 1 | 100 | 100 | Isosceles | Isosceles |
| 91 | 1 | 100 | 101 | Invalid Input | Invalid Input |
| 92 | 1 | 101 | 0 | Invalid Input | Invalid Input |
| 93 | 1 | 101 | 1 | Invalid Input | Invalid Input |
| 94 | 1 | 101 | 2 | Invalid Input | Invalid Input |
| 95 | 1 | 101 | 50 | Invalid Input | Invalid Input |
| 96 | 1 | 101 | 99 | Invalid Input | Invalid Input |
| 97 | 1 | 101 | 100 | Invalid Input | Invalid Input |
| 98 | 1 | 101 | 101 | Invalid Input | Invalid Input |
| 99 | 2 | 0 | 0 | Invalid Input | Invalid Input |
| 100 | 2 | 0 | 1 | Invalid Input | Invalid Input |
| 101 | 2 | 0 | 2 | Invalid Input | Invalid Input |
| 102 | 2 | 0 | 50 | Invalid Input | Invalid Input |
| 103 | 2 | 0 | 99 | Invalid Input | Invalid Input |
| 104 | 2 | 0 | 100 | Invalid Input | Invalid Input |
| 105 | 2 | 0 | 101 | Invalid Input | Invalid Input |
| 106 | 2 | 1 | 0 | Invalid Input | Invalid Input |
| 107 | 2 | 1 | 1 | Isosceles | Isosceles |
| 108 | 2 | 1 | 2 | Isosceles | Isosceles |
| 109 | 2 | 1 | 50 | Scalene | Scalene |
| 110 | 2 | 1 | 99 | Scalene | Scalene |
| 111 | 2 | 1 | 100 | Scalene | Scalene |
| 112 | 2 | 1 | 101 | Invalid Input | Invalid Input |
| 113 | 2 | 2 | 0 | Invalid Input | Invalid Input |
| 114 | 2 | 2 | 1 | Isosceles | Isosceles |
| 115 | 2 | 2 | 2 | Equilateral | Equilateral |
| 116 | 2 | 2 | 50 | Isosceles | Isosceles |
| 117 | 2 | 2 | 99 | Isosceles | Isosceles |
| 118 | 2 | 2 | 100 | Isosceles | Isosceles |
| 119 | 2 | 2 | 101 | Invalid Input | Invalid Input |
| 120 | 2 | 50 | 0 | Invalid Input | Invalid Input |
| 121 | 2 | 50 | 1 | Scalene | Scalene |
| 122 | 2 | 50 | 2 | Isosceles | Isosceles |
| 123 | 2 | 50 | 50 | Isosceles | Isosceles |
| 124 | 2 | 50 | 99 | Scalene | Scalene |
| 125 | 2 | 50 | 100 | Scalene | Scalene |
| 126 | 2 | 50 | 101 | Invalid Input | Invalid Input |
| 127 | 2 | 99 | 0 | Invalid Input | Invalid Input |
| 128 | 2 | 99 | 1 | Scalene | Scalene |
| 129 | 2 | 99 | 2 | Isosceles | Isosceles |
| 130 | 2 | 99 | 50 | Scalene | Scalene |
| 131 | 2 | 99 | 99 | Isosceles | Isosceles |
| 132 | 2 | 99 | 100 | Scalene | Scalene |
| 133 | 2 | 99 | 101 | Invalid Input | Invalid Input |
| 134 | 2 | 100 | 0 | Invalid Input | Invalid Input |
| 135 | 2 | 100 | 1 | Scalene | Scalene |
| 136 | 2 | 100 | 2 | Isosceles | Isosceles |
| 137 | 2 | 100 | 50 | Scalene | Scalene |
| 138 | 2 | 100 | 99 | Scalene | Scalene |
| 139 | 2 | 100 | 100 | Isosceles | Isosceles |
| 140 | 2 | 100 | 101 | Invalid Input | Invalid Input |
| 141 | 2 | 101 | 0 | Invalid Input | Invalid Input |
| 142 | 2 | 101 | 1 | Invalid Input | Invalid Input |
| 143 | 2 | 101 | 2 | Invalid Input | Invalid Input |
| 144 | 2 | 101 | 50 | Invalid Input | Invalid Input |
| 145 | 2 | 101 | 99 | Invalid Input | Invalid Input |
| 146 | 2 | 101 | 100 | Invalid Input | Invalid Input |
| 147 | 2 | 101 | 101 | Invalid Input | Invalid Input |
| 148 | 50 | 0 | 0 | Invalid Input | Invalid Input |
| 149 | 50 | 0 | 1 | Invalid Input | Invalid Input |
| 150 | 50 | 0 | 2 | Invalid Input | Invalid Input |
| 151 | 50 | 0 | 50 | Invalid Input | Invalid Input |
| 152 | 50 | 0 | 99 | Invalid Input | Invalid Input |
| 153 | 50 | 0 | 100 | Invalid Input | Invalid Input |
| 154 | 50 | 0 | 101 | Invalid Input | Invalid Input |
| 155 | 50 | 1 | 0 | Invalid Input | Invalid Input |
| 156 | 50 | 1 | 1 | Isosceles | Isosceles |
| 157 | 50 | 1 | 2 | Scalene | Scalene |
| 158 | 50 | 1 | 50 | Scalene | Scalene |
| 159 | 50 | 1 | 99 | Scalene | Scalene |
| 160 | 50 | 1 | 100 | Scalene | Scalene |
| 161 | 50 | 1 | 101 | Invalid Input | Invalid Input |
| 162 | 50 | 2 | 0 | Invalid Input | Invalid Input |
| 163 | 50 | 2 | 1 | Scalene | Scalene |
| 164 | 50 | 2 | 2 | Isosceles | Isosceles |
| 165 | 50 | 2 | 50 | Scalene | Scalene |
| 166 | 50 | 2 | 99 | Scalene | Scalene |
| 167 | 50 | 2 | 100 | Scalene | Scalene |
| 168 | 50 | 2 | 101 | Invalid Input | Invalid Input |
| 169 | 50 | 50 | 0 | Invalid Input | Invalid Input |
| 170 | 50 | 50 | 1 | Isosceles | Isosceles |
| 171 | 50 | 50 | 2 | Isosceles | Isosceles |
| 172 | 50 | 50 | 50 | Equilateral | Equilateral |
| 173 | 50 | 50 | 99 | Isosceles | Isosceles |
| 174 | 50 | 50 | 100 | Isosceles | Isosceles |
| 175 | 50 | 50 | 101 | Invalid Input | Invalid Input |
| 176 | 50 | 99 | 0 | Invalid Input | Invalid Input |
| 177 | 50 | 99 | 1 | Scalene | Scalene |
| 178 | 50 | 99 | 2 | Scalene | Scalene |
| 179 | 50 | 99 | 50 | Isosceles | Isosceles |
| 180 | 50 | 99 | 99 | Isosceles | Isosceles |
| 181 | 50 | 99 | 100 | Scalene | Scalene |
| 182 | 50 | 99 | 101 | Invalid Input | Invalid Input |
| 183 | 50 | 100 | 0 | Invalid Input | Invalid Input |
| 184 | 50 | 100 | 1 | Scalene | Scalene |
| 185 | 50 | 100 | 2 | Scalene | Scalene |
| 186 | 50 | 100 | 50 | Isosceles | Isosceles |
| 187 | 50 | 100 | 99 | Scalene | Scalene |
| 188 | 50 | 100 | 100 | Isosceles | Isosceles |
| 189 | 50 | 100 | 101 | Invalid Input | Invalid Input |
| 190 | 50 | 101 | 0 | Invalid Input | Invalid Input |
| 191 | 50 | 101 | 1 | Invalid Input | Invalid Input |
| 192 | 50 | 101 | 2 | Invalid Input | Invalid Input |
| 193 | 50 | 101 | 50 | Invalid Input | Invalid Input |
| 194 | 50 | 101 | 99 | Invalid Input | Invalid Input |
| 195 | 50 | 101 | 100 | Invalid Input | Invalid Input |
| 196 | 50 | 101 | 101 | Invalid Input | Invalid Input |
| 197 | 99 | 0 | 0 | Invalid Input | Invalid Input |
| 198 | 99 | 0 | 1 | Invalid Input | Invalid Input |
| 199 | 99 | 0 | 2 | Invalid Input | Invalid Input |
| 200 | 99 | 0 | 50 | Invalid Input | Invalid Input |
| 201 | 99 | 0 | 99 | Invalid Input | Invalid Input |
| 202 | 99 | 0 | 100 | Invalid Input | Invalid Input |
| 203 | 99 | 0 | 101 | Invalid Input | Invalid Input |
| 204 | 99 | 1 | 0 | Invalid Input | Invalid Input |
| 205 | 99 | 1 | 1 | Isosceles | Isosceles |
| 206 | 99 | 1 | 2 | Scalene | Scalene |
| 207 | 99 | 1 | 50 | Scalene | Scalene |
| 208 | 99 | 1 | 99 | Isosceles | Isosceles |
| 209 | 99 | 1 | 100 | Scalene | Scalene |
| 210 | 99 | 1 | 101 | Invalid Input | Invalid Input |
| 211 | 99 | 2 | 0 | Invalid Input | Invalid Input |
| 212 | 99 | 2 | 1 | Scalene | Scalene |
| 213 | 99 | 2 | 2 | Isosceles | Isosceles |
| 214 | 99 | 2 | 50 | Scalene | Scalene |
| 215 | 99 | 2 | 99 | Isosceles | Isosceles |
| 216 | 99 | 2 | 100 | Scalene | Scalene |
| 217 | 99 | 2 | 101 | Invalid Input | Invalid Input |
| 218 | 99 | 50 | 0 | Invalid Input | Invalid Input |
| 219 | 99 | 50 | 1 | Scalene | Scalene |
| 220 | 99 | 50 | 2 | Scalene | Scalene |
| 221 | 99 | 50 | 50 | Isosceles | Isosceles |
| 222 | 99 | 50 | 99 | Isosceles | Isosceles |
| 223 | 99 | 50 | 100 | Scalene | Scalene |
| 224 | 99 | 50 | 101 | Invalid Input | Invalid Input |
| 225 | 99 | 99 | 0 | Invalid Input | Invalid Input |
| 226 | 99 | 99 | 1 | Isosceles | Isosceles |
| 227 | 99 | 99 | 2 | Isosceles | Isosceles |
| 228 | 99 | 99 | 50 | Isosceles | Isosceles |
| 229 | 99 | 99 | 99 | Equilateral | Equilateral |
| 230 | 99 | 99 | 100 | Isosceles | Isosceles |
| 231 | 99 | 99 | 101 | Invalid Input | Invalid Input |
| 232 | 99 | 100 | 0 | Invalid Input | Invalid Input |
| 233 | 99 | 100 | 1 | Scalene | Scalene |
| 234 | 99 | 100 | 2 | Scalene | Scalene |
| 235 | 99 | 100 | 50 | Scalene | Scalene |
| 236 | 99 | 100 | 99 | Isosceles | Isosceles |
| 237 | 99 | 100 | 100 | Isosceles | Isosceles |
| 238 | 99 | 100 | 101 | Invalid Input | Invalid Input |
| 239 | 99 | 101 | 0 | Invalid Input | Invalid Input |
| 240 | 99 | 101 | 1 | Invalid Input | Invalid Input |
| 241 | 99 | 101 | 2 | Invalid Input | Invalid Input |
| 242 | 99 | 101 | 50 | Invalid Input | Invalid Input |
| 243 | 99 | 101 | 99 | Invalid Input | Invalid Input |
| 244 | 99 | 101 | 100 | Invalid Input | Invalid Input |
| 245 | 99 | 101 | 101 | Invalid Input | Invalid Input |
| 246 | 100 | 0 | 0 | Invalid Input | Invalid Input |
| 247 | 100 | 0 | 1 | Invalid Input | Invalid Input |
| 248 | 100 | 0 | 2 | Invalid Input | Invalid Input |
| 249 | 100 | 0 | 50 | Invalid Input | Invalid Input |
| 250 | 100 | 0 | 99 | Invalid Input | Invalid Input |
| 251 | 100 | 0 | 100 | Invalid Input | Invalid Input |
| 252 | 100 | 0 | 101 | Invalid Input | Invalid Input |
| 253 | 100 | 1 | 0 | Invalid Input | Invalid Input |
| 254 | 100 | 1 | 1 | Isosceles | Isosceles |
| 255 | 100 | 1 | 2 | Scalene | Scalene |
| 256 | 100 | 1 | 50 | Scalene | Scalene |
| 257 | 100 | 1 | 99 | Scalene | Scalene |
| 258 | 100 | 1 | 100 | Isosceles | Isosceles |
| 259 | 100 | 1 | 101 | Invalid Input | Invalid Input |
| 260 | 100 | 2 | 0 | Invalid Input | Invalid Input |
| 261 | 100 | 2 | 1 | Scalene | Scalene |
| 262 | 100 | 2 | 2 | Isosceles | Isosceles |
| 263 | 100 | 2 | 50 | Scalene | Scalene |
| 264 | 100 | 2 | 99 | Scalene | Scalene |
| 265 | 100 | 2 | 100 | Isosceles | Isosceles |
| 266 | 100 | 2 | 101 | Invalid Input | Invalid Input |
| 267 | 100 | 50 | 0 | Invalid Input | Invalid Input |
| 268 | 100 | 50 | 1 | Scalene | Scalene |
| 269 | 100 | 50 | 2 | Scalene | Scalene |
| 270 | 100 | 50 | 50 | Isosceles | Isosceles |
| 271 | 100 | 50 | 99 | Scalene | Scalene |
| 272 | 100 | 50 | 100 | Isosceles | Isosceles |
| 273 | 100 | 50 | 101 | Invalid Input | Invalid Input |
| 274 | 100 | 99 | 0 | Invalid Input | Invalid Input |
| 275 | 100 | 99 | 1 | Scalene | Scalene |
| 276 | 100 | 99 | 2 | Scalene | Scalene |
| 277 | 100 | 99 | 50 | Scalene | Scalene |
| 278 | 100 | 99 | 99 | Isosceles | Isosceles |
| 279 | 100 | 99 | 100 | Isosceles | Isosceles |
| 280 | 100 | 99 | 101 | Invalid Input | Invalid Input |
| 281 | 100 | 100 | 0 | Invalid Input | Invalid Input |
| 282 | 100 | 100 | 1 | Isosceles | Isosceles |
| 283 | 100 | 100 | 2 | Isosceles | Isosceles |
| 284 | 100 | 100 | 50 | Isosceles | Isosceles |
| 285 | 100 | 100 | 99 | Isosceles | Isosceles |
| 286 | 100 | 100 | 100 | Equilateral | Equilateral |
| 287 | 100 | 100 | 101 | Invalid Input | Invalid Input |
| 288 | 100 | 101 | 0 | Invalid Input | Invalid Input |
| 289 | 100 | 101 | 1 | Invalid Input | Invalid Input |
| 290 | 100 | 101 | 2 | Invalid Input | Invalid Input |
| 291 | 100 | 101 | 50 | Invalid Input | Invalid Input |
| 292 | 100 | 101 | 99 | Invalid Input | Invalid Input |
| 293 | 100 | 101 | 100 | Invalid Input | Invalid Input |
| 294 | 100 | 101 | 101 | Invalid Input | Invalid Input |
| 295 | 101 | 0 | 0 | Invalid Input | Invalid Input |
| 296 | 101 | 0 | 1 | Invalid Input | Invalid Input |
| 297 | 101 | 0 | 2 | Invalid Input | Invalid Input |
| 298 | 101 | 0 | 50 | Invalid Input | Invalid Input |
| 299 | 101 | 0 | 99 | Invalid Input | Invalid Input |
| 300 | 101 | 0 | 100 | Invalid Input | Invalid Input |
| 301 | 101 | 0 | 101 | Invalid Input | Invalid Input |
| 302 | 101 | 1 | 0 | Invalid Input | Invalid Input |
| 303 | 101 | 1 | 1 | Invalid Input | Invalid Input |
| 304 | 101 | 1 | 2 | Invalid Input | Invalid Input |
| 305 | 101 | 1 | 50 | Invalid Input | Invalid Input |
| 306 | 101 | 1 | 99 | Invalid Input | Invalid Input |
| 307 | 101 | 1 | 100 | Invalid Input | Invalid Input |
| 308 | 101 | 1 | 101 | Invalid Input | Invalid Input |
| 309 | 101 | 2 | 0 | Invalid Input | Invalid Input |
| 310 | 101 | 2 | 1 | Invalid Input | Invalid Input |
| 311 | 101 | 2 | 2 | Invalid Input | Invalid Input |
| 312 | 101 | 2 | 50 | Invalid Input | Invalid Input |
| 313 | 101 | 2 | 99 | Invalid Input | Invalid Input |
| 314 | 101 | 2 | 100 | Invalid Input | Invalid Input |
| 315 | 101 | 2 | 101 | Invalid Input | Invalid Input |
| 316 | 101 | 50 | 0 | Invalid Input | Invalid Input |
| 317 | 101 | 50 | 1 | Invalid Input | Invalid Input |
| 318 | 101 | 50 | 2 | Invalid Input | Invalid Input |
| 319 | 101 | 50 | 50 | Invalid Input | Invalid Input |
| 320 | 101 | 50 | 99 | Invalid Input | Invalid Input |
| 321 | 101 | 50 | 100 | Invalid Input | Invalid Input |
| 322 | 101 | 50 | 101 | Invalid Input | Invalid Input |
| 323 | 101 | 99 | 0 | Invalid Input | Invalid Input |
| 324 | 101 | 99 | 1 | Invalid Input | Invalid Input |
| 325 | 101 | 99 | 2 | Invalid Input | Invalid Input |
| 326 | 101 | 99 | 50 | Invalid Input | Invalid Input |
| 327 | 101 | 99 | 99 | Invalid Input | Invalid Input |
| 328 | 101 | 99 | 100 | Invalid Input | Invalid Input |
| 329 | 101 | 99 | 101 | Invalid Input | Invalid Input |
| 330 | 101 | 100 | 0 | Invalid Input | Invalid Input |
| 331 | 101 | 100 | 1 | Invalid Input | Invalid Input |
| 332 | 101 | 100 | 2 | Invalid Input | Invalid Input |
| 333 | 101 | 100 | 50 | Invalid Input | Invalid Input |
| 334 | 101 | 100 | 99 | Invalid Input | Invalid Input |
| 335 | 101 | 100 | 100 | Invalid Input | Invalid Input |
| 336 | 101 | 100 | 101 | Invalid Input | Invalid Input |
| 337 | 101 | 101 | 0 | Invalid Input | Invalid Input |
| 338 | 101 | 101 | 1 | Invalid Input | Invalid Input |
| 339 | 101 | 101 | 2 | Invalid Input | Invalid Input |
| 340 | 101 | 101 | 50 | Invalid Input | Invalid Input |
| 341 | 101 | 101 | 99 | Invalid Input | Invalid Input |
| 342 | 101 | 101 | 100 | Invalid Input | Invalid Input |
| 343 | 101 | 101 | 101 | Invalid Input | Invalid Input |

Total no. of test cases = 343 (7^3 = 343)

**EXPERIMENT 5**

**Aim:** WAP in C/C++ to find the area of a circle, Triangle, Square and Rectangle and perform equivalence class testing with range of input is 1-100**.**

PROGRAM:

#include<stdio.h>

#include<conio.h>

void main()

{

clrscr();

int ch;

char c;

float, b, h, a;

1: cout<<“Enter your choice”;

cout<<“n1.Triangle”;

cout<<“n2.Square”;

cout<<“n3.Rectangle”;

cout<<“n4.Circle”;

cout<<“n5.Exitn”;

cin>>ch;

switch(ch)

{ case 1 : b: cout<<“nEnter the base of the triangle (1-200)”;

cin>>b;

if ((b<=0)||(b>200))

{ cout<<“nInvalid entry for base n”;

goto b;

}

h: cout<<“nEnter the height of the triangle (1-200)”;

cin>>h;

if ((h<=0)||(h>200))

{ cout<<“nInvalid height nEnter the height (1-200)”;

goto h;

}

a= 0.5\*b\*h;

cout<<“nThe area is “<<a; <=”” font=””></a;>

cout<<“nWant to enter more?(y/n) “;

cin>>c;

if((c==’y’)||(c==’Y’))

goto 1;

break

case 2 : s: cout<<“nEnter the side of the square (1-200)”;

cin>>b;

if ((b<=0)||(b>200))

{ cout<<“nInvalid entry for base n”;

goto s;

}

a= b\*b;

cout<<“nThe area is “<<a; <=”” font=””></a;>

cout<<“nWant to enter more?(y/n) “;

cin>>c;

if((c==’y’)||(c==’Y’))

goto 1;

break;

case 3: d: cout<<“nEnter the base of the triangle (1-200)” ;

cin>>b;

if((b<=0)||(b>200))

( cout<<“nInvalid entry for base n”;

goto d;

}

p: cout<<“nEnter the height of the triangle (1-200) “;

cin>>h;

if ((h<=0)||(h>200))

{ cout<<“nInvalid height nEnter the height(1-200)”;

goto p;

}

a=b\*h;

cout<<“nThe area is “<<a; <=”” font=””></a;>

cout<<“nWant to enter more?(y/n) “;

cin>>c;

if((c==’y’)||(c==’Y’))

goto 1;

break;

case 4: t: cout<<“nEnter the radius of the circle “;

cin>>b;

if ((b<=0)||(b>200))

{ cout<<“nInvalid entry for base n”;

goto t;

}

a= 3.14\*b\*b;

cout<<“nThe area is “<<a; <=”” font=””></a;>

cout<<“nWant to enter more?(y/n)”;

cin>>c;

if ((c==’y’)||(c==’Y’))

goto 1;

break;

case 5: exit(0);

break;

default : cout<<“n WRONG CHOICE”;

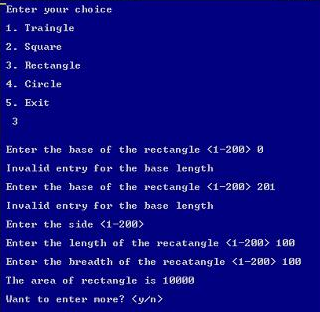
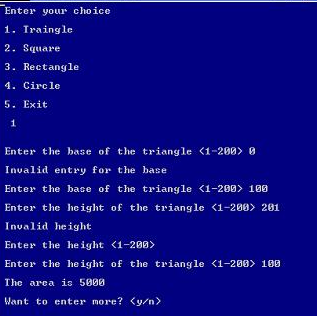
goto 1;

}

getch();

}

**OUTPUT**



**Equivalence Class Test Cases**  
In Equivalence Class Testing, we find two types of equivalence classes 1) Input Domain and 2) Output Domain.

Input Domain is formed from one valid sequence and two invalid sequences. The Output Domain is obtained from different types of outputs of the problem.

**Case 1: Triangle  
  
Input Domain:**

I1 = {h : h<=0}

I2 = {h : h>200}

I3 = {h : 1<=h<=200}

I4 = {b : b<=0}

I5 = {b : b>200}

I6 = {b : 1<=b<=2001}

Test Cases for the Case of Triangle are as under:

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case ID** | **h** | **b** | **Expected Output** |
| 1 | 0 | 100 | Invalid Input |
| 2 | 100 | 100 | 5000 |
| 3 | 201 | 100 | Invalid Input |
| 4 | 100 | 0 | Invalid Input |
| 5 | 100 | 100 | 5000 |
| 6 | 100 | 201 | Invalid Input |

**Output Domain:**

O1 = {: Triangle if h > 0, b > 0}

O2 = {: Not a triangle if h <= 0, b <= 0}

**Case 2: Square**

**Input Domain:**

I1 = {s : s<=0}

I2 = {s : s>200}

I3 = {s : 1<=s<=200}

Test Cases for the Case of Square are as under:

|  |  |  |
| --- | --- | --- |
| **Test Case ID** | **s** | **Expected Output** |
| 1 | 0 | Invalid Input |
| 2 | 100 | 10000 |
| 3 | 201 | Invalid Input |

**Output Domain:**

O1 = {: Square if s>}

O2 = {: Not a square if s <= 0}

**Case 3: Rectangle**

**Input Domain:**

I1 = {l : l<=0}

I2 = {l : l>200}

I3 = {l : 1<=l<=200}

I4 = {b : b<=0}

I5 = {b : b>200}

I6 = {b : 1<=b<=200}

Test Cases for the Case of Rectangle are as under:

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case ID** | **l** | **b** | **Expected Output** |
| 1 | 0 | 100 | Invalid Input |
| 2 | 100 | 100 | 10000 |
| 3 | 201 | 100 | Invalid Input |
| 4 | 100 | 0 | Invalid Input |
| 5 | 100 | 100 | 10000 |
| 6 | 100 | 201 | Invalid Input |

**Case 4: Circle**

**Input Domain:**

I1 = {r : r<=0}

I2 = {r : r>200}

I3 = {r : 1<=r<=200}

Test Cases for the Case of Circle are as under:

|  |  |  |
| --- | --- | --- |
| **Test Case ID** | **r** | **Expected Output** |
| 1 | 0 | Invalid Input |
| 2 | 100 | 31400 |
| 3 | 201 | Invalid Input |

**Output Domain:**

O1 = {: Circle if 1<=r<=200}

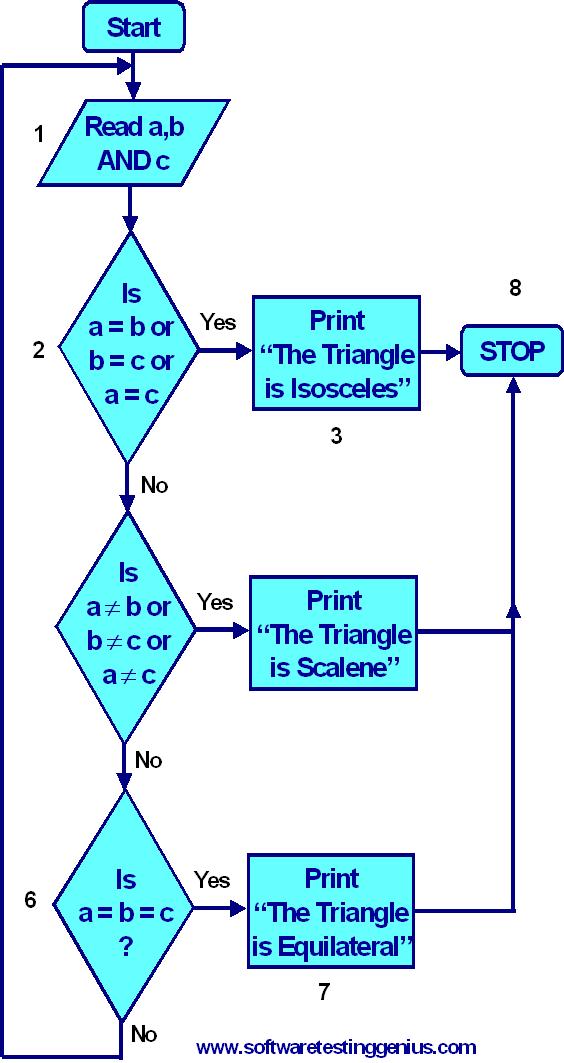
O2 = {: Not a Circle if r <= 0}

**EXPERIMENT 6**

**Aim:** WAP in C/C++ to compute 3 sides of a triangle and to determine whether they form scalene, isosceles, or equilateral triangle and perform Equivalence Class testing with range of input is 1-100.

**PROGRAM**: **FLOWCHART:**

#include<stdio.h>

#include<math.h>

#include<conio.h>

void main()

{

int x,y,z;

clrscr();

printf("Enter the value of the three sides : \n");

printf("X : ");

scanf("%d",&x);

printf("\nY : ");

scanf("%d",&y);

printf("\nZ : ");

scanf("%d",&z);

if ((x<1 || x >100) || (y<1 || y>100) || (z<1 || z>100))

printf("\nValues outside range");

else

if ((x+y > z) && (y+z > x) && (z+x > y))

{

if (x==y && y==z)

printf("\nThis ia an Equilateral triangle");

else if (x==y || y==z || z==y)

printf("\nThis is an Isoceles triangle");

else

printf("\nScalene Triangle");

if (x\*x+y\*y==z\*z || y\*y+z\*z==x\*x || z\*z+x\*x==y\*y)

printf("\nRight Angled Triangle");

}

else

printf("\nNot a triangle");

getch();

}

Input Domain Based Classes are:

|  |  |
| --- | --- |
| I1 = {x: x<1}  I2 = {x: x>100}  I3 = {x: 1≤x≤100}  I4 = {y: y<1}  I5 = {y: y>100}  I6 = {y: 1≤y≤100}  I7 = {z: z<1}  I8 = {z: z>100}  I9 = {z: 1≤z≤100}  I10 = {<x, y, z>:x=y=z} | I11 = {<x, y, z>: x = y, x ≠ z}  I12 = {<x, y, z>: x = z, x ≠ y}  I13 = {<x, y, z>: y = z, x ≠ y}  I14 = {<x, y, z>: x ≠ y, x ≠ z, y ≠ z}  I15 = {<x, y, z>: x = y + z}  I16 = {<x, y, z>: x > y + z}  I17 = {<x, y, z>: y = x +z}  I18 = {<x, y, z>: y > x +z}  I19 = {<x, y, z>: z = x +y}  I20 = {<x, y, z>: z > x + y} |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test case | X | Y | Z | Expected Output |
| 1 | 0 | 50 | 50 | Invalid Input |
| 2 | 101 | 50 | 50 | Invalid Input |
| 3 | 50 | 50 | 50 | Equilateral |
| 4 | 50 | 0 | 50 | Invalid Input |
| 5 | 50 | 101 | 50 | Invalid Input |
| 6 | 50 | 50 | 50 | Equilateral |
| 7 | 50 | 50 | 0 | Invalid Input |
| 8 | 50 | 50 | 101 | Invalid Input |
| 9 | 50 | 50 | 50 | Equilateral |
| 10 | 60 | 60 | 60 | Equilateral |
| 11 | 50 | 50 | 60 | Isosceles |
| 12 | 50 | 60 | 50 | Isosceles |
| 13 | 60 | 50 | 50 | Isosceles |
| 14 | 100 | 99 | 50 | Scalene |
| 15 | 100 | 50 | 50 | Not a Triangle |
| 16 | 100 | 50 | 25 | Not a Triangle |
| 17 | 50 | 100 | 50 | Not a Triangle |
| 18 | 50 | 100 | 25 | Not a Triangle |
| 19 | 50 | 50 | 100 | Not a Triangle |
| 20 | 25 | 50 | 100 | Not a Triangle |

Output Domain Equivalence Classes are:

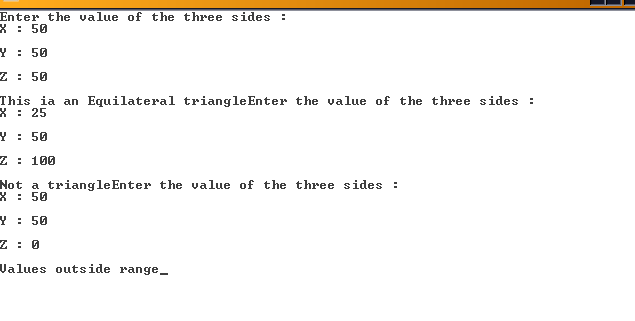
O1 = {<x, y, z>: Equilateral Triangle with sides x, y, z}

O2 = {<x, y, z>: Isosceles Triangle with sides x, y, z}

O3 = {<x, y, z>: Scalene Triangle with sides x, y, z}

O4 = {<x, y, z>: Not a Triangle with sides x, y, z}

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test Case | X | Y | Z | Expected Output |
| 1 | 50 | 50 | 50 | Equilateral |
| 2 | 50 | 50 | 99 | Isosceles |
| 3 | 100 | 99 | 50 | Scalene |
| 4 | 50 | 100 | 50 | Not a Triangle |



**EXPERIMENT 7**

**Aim:** WAP in C/C++ to compute ab and perform its decision-based testing.

**PROGRAM:**

#include<stdio.h>

void main()

c1rscr();

int a,b;

float c;

char ch;

I : cout<<TO P < b?n?; power the to `a CALCULATE>

cin>>;

c=pow(a,b);

cout<<endl<<; <=”” p=””></endl<<;>

cout<<“nWant to enter again?(y/n)

cin>>ch;

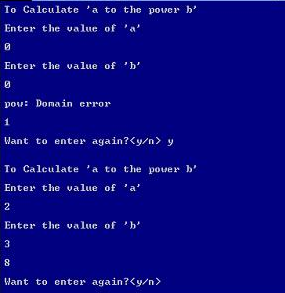
if((ch==’y’)||(ch==’Y’))

goto I;

getch();

}

**OUTPUT**



**Decision Table Based Test Cases**

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case ID** | **a** | **b** | **Expected Output** |
| 1 | 2 | 3 | +ve Result |
| 2 | -1 | 3 | -ve Result |
| 3 | -2 | -4 | +ve Result |
| 4 | 0 | 1 | Result is 0 |
| 5 | 0 | 0 | Domain Error |
| 6 | -1 | -0.6 | Result is 1 |

**Decision Table is as under**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Conditions** |  |  |  |  |  |  |  |  |
| C1:a = 0,b = 0 | T |  |  |  |  |  |  |  |
| C2:a = -ve, b = +ve |  | T |  |  |  |  |  |  |
| C3:a = +ve, b = -ve |  |  |  | T |  |  |  |  |
| C4:a = -ve, b = -ve |  |  |  |  | T |  |  |  |
| C5:a = +ve, b = +ve |  |  |  |  |  | T |  |  |
| C6:a = 0, b = integer |  |  |  |  |  |  | T |  |
| C7:b = 0, a = integer |  |  |  |  |  |  |  | T |
| C8:a = -ve, b = -ve odd |  |  | T |  |  |  |  |  |
| **Actions** |  |  |  |  |  |  |  |  |
| A1: Domain Error | X |  |  |  |  |  |  |  |
| A2: Negative Output |  |  | X |  |  |  |  |  |
| A3: Output = 1 |  |  |  |  |  |  |  | X |
| A4: Positive Output |  | X |  | X |  | X |  |  |
| A5: Output = 0 |  |  |  |  |  |  | X |  |

**EXPERIMENT 8**

**Aim-5 (A):** WAP in C/C++ to compute previous date, given the present date as input and perform decision table-based testing between year 1901-2050

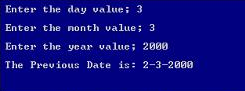
**PROGRAM**

#include<stdio.h>

#include<conio.h>

int main()  
{  
int day, month, year, flag=0;  
printf(” enter the day value;”);  
scanf(“%d”,&day) ;  
printf(“enter the month value;”);  
scanf(“%d”,&month);  
printf(“enter the year value;”);  
scanf(“%d”,&year) ;  
if(year> = 1900 && year< = 2025)  
{  
if(month==l||month==3||month==5||month==7||month==8||month==10||month==12)  
{  
if(day> = 1&&day<=31)  
{  
flag = 1;  
}  
else  
{  
flag = 0;  
}  
}  
elseif (month==2)  
{  
int rval=0;  
if(year%4==0)  
{  
rval=l;  
if((year%100)==0 && (year%400)! = 0  
{rval = 0;  
}  
}  
if(rva1==&&(day>=1 &&day <=29))  
{ flag = 1;  
}  
elseif (day>=1 &&day <=28)  
{ flag = 1;  
else  
}  
flag = 0;  
}  
}  
if (flag)  
{  
if(day==1)  
{  
if(month==1)  
{  
year–;  
day = 31;  
month = 12;  
}  
else if(month==3)  
{  
int rval = 0;  
if(year%4==0)  
{  
rval = 1;  
if ((year%100) == 0 && (year%400): = 0)  
{ravl = 0;  
}  
}  
if (ravl==1)  
{  
day = 29;  
month–;  
}  
else  
{  
day = 28;  
month–;  
}  
}  
else if (month==2|| month==4||month==6||month==9||month==)  
{day = 31;  
month–;  
}  
else  
{  
day = 30;  
month–;  
}  
}  
else  
{  
day–;  
}  
printf (“the next data is:%d-%d-%d”, day, month, year) ;  
}  
else  
{  
printf (“the entered data is invalid: %d-%d-%d, day, month, year)  
{  
getche();  
return 1;  
}

**OUTPUT**



**Decision Table Based Test Cases**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test Case ID** | **Month** | **Day** | **Year** | **Expected Output** |
| 1 | June | 1 | 1964 | 31 May 1964 |
| 2 | June | 31 | 1984 | Impossible |
| 3 | May | 1 | 1945 | 30 April 1945 |
| 4 | March | 31 | 2007 | 30 March 2007 |
| 5 | August | 29 | 2007 | 28 August 2007 |
| 6 | February | 29 | 1962 | Impossible |

**Decision Table is as under**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Conditions** |  | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| C1:  y<1901 or y>2050 | T | | | | | | | | | | | | | | | | | | F | | | | | | | | | | | | | | | |
| C2:  d, m <1 or d>31, m>12 | T | | | | | | | | | | F | | | | | | | | T | | | | | | | | F | | | | | | | |
| C3:  leap year m=2 and d>29 | T | | | | | | F | | | | T | | | | F | | | | T | | | | F | | | | T | | | | F | | | |
| C4:  m=2 and d>28 | T | | | | F | | T | | F | | T | | F | | T | | F | | T | | F | | T | | F | | T | | F | | T | | F | |
| C5: m=4,6,9,11 and d>30 | T | | F | | T | F | T | F | T | F | T | F | T | F | T | F | T | F | T | F | T | F | T | F | T | F | T | F | T | F | T | F | T | F |
| **Actions** |  | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| A1: Impossible |  |  | |  | |  |  |  |  |  |  |  |  |  |  |  |  |  | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X |  |
| A2:  Previous date |  |  | |  | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | X |
| A3:  Input out of range | X | X | | X | | X | X | X | X | X | X | X | X | X | X | X | X | X |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

**EXPERIMENT 9**

**Aim:** Create a test plan document for any application (**Library Management System**).

**1 BACKGROUND**

The Library Management System is an online application for assisting a librarian in managing a book library in a University. The system would provide basic set of features to add/update clients, add/update books, search for books, and manage check-in / checkout processes. Our test group tested the system based on the requirement specification.

**2 INTRODUCTION**

This test report is the result for testing in the LMS. It mainly focuses on two problems: what we will test and how we will test.

**3 Result**

**3.1 GUI test**

**Pass criteria**: librarians could use this GUI to interface with the backend library database without any difficulties

**Result**: pass

**3.2 Database test**

**Pass criteria:** Results of all basic and advanced operations are normal (refer to section 4)

**Result:** pass

**3.3 Basic function test**

**3.3.1 Add a student**

**Pass criteria:**

• Each customer/student should have following attributes: Student ID/SSN (unique), Name, Address and

Phone number.

**Result:** pass

• The retrieved customer information by viewing customer detail should contain the four attributes.

**Result:** pass

**3.3.2 Update/delete student**

**Pass criteria:**

• The record would be selected using the student ID

**Result:** pass

• Updates can be made on full. Items only: Name, Address, Phone number

**Result:** pass

• The record can be deleted if there are no books issued by user.

**Result:** Partially pass. When no books issued by user, he can be deleted. But when there are books

Issued by this user, he was also deleted. It is wrong.

• The updated values would be reflected if the same customer's ID/SSN is called for.

**Result:** pass

• If customer were deleted, it would not appear in further search queries.

**Result:** pass

**3.3.3 Add a book**

**Pass criteria:**

• Each book shall have following attributes: Call Number, ISBN, Title, Author name.

**Result:** pass

• The retrieved book information should contain the four attributes.

**Result:** pass

**3.3.4 Update/delete book**

**Pass criteria:**

• The book item can be retrieved using the call number

**Result:** did not pass. Cannot retrieve using the call number

• The data items which can be updated are: ISBN, Title, Author name

**Result:** pass

• The book can be deleted only if no user has issued it.

**Result:** partially pass. When no user has issued it, pass. When there are user having issued it, did not

pass

• The updated values would be reflected if the same call number is called for

**Result:** pass

• If book were deleted, it would not appear in further search queries.

**Result:** pass

**3.3.5 Search for book**

**Pass criteria:**

• The product shall let Librarian query books’ detail information by their ISBN number or Author or Title.

**Result:** pass

• The search results would produce a list of books, which match the search parameters with following

**Details**: Call number, ISBN number, Title, Author

**Result:** pass

• The display would also provide the number of copies which is available for issue

**Result:** pass

• The display shall provide a means to select one or more rows to a user-list

**Result:** pass

• A detailed view of each book should provide information about check-in/check out status, with the b

borrower’s information.

**Result:** pass

• The search display will be restricted to 20 results per page and there would be means to navigate from

sets of search results.

**Result:** pass

• The user can perform multiple searches before finally selecting a set of books for check in or checkout.

These should be stored across searches.

**Result:** pass

• A book may have more than one copy. But every copy with the same ISBN number should have same

detail information.

**Result:** pass

• The borrower’s list should agree with the data in students’ account

**Result:** pass

**3.3.6 Check-in book**

**Pass criteria:**

• Librarians can check in a book using its call number

**Result:** pass

• The check-in can be initiated from a previous search operation where user has selected a set of books.

**Result:** pass

• The return date would automatically reflect the current system date.

**Result:** did not pass.

• Any late fees would be computed as difference between due date and return date at rate of 10 cents a

day.

**Result:** did not pass

• A book, which has been checked in once, should not be checked in again

**Result:** pass

**3.3.7 Check-out book**

**Pass criteria:**

• Librarians can check out a book using its call number

**Result:** pass

• The checkout can be initiated from a previous search operation where user has selected a set of

books.

**Result:** pass

• The student ID who is issuing the book would be entered

**Result:** pass

• The issue date would automatically reflect the current system date.

**Result:** did not pass

• The due date would automatically be stamped as 5 days from current date.

**Result:** did not pass

• A book, which has been checked out once, should not be checked out again

**Result:** pass

• A student who has books due should not be allowed to check out any books

**Result:** did not pass

• The max. No of books that can be issued to a customer would be 10. The system should not allow

checkout of books beyond this limit.

**Result:** pass

**3.3.8 View book detail**

**Pass criteria:**

• This view would display details about a selected book from search operation

**Result:** pass

• The details to be displayed are: Call number, IBN, Title, Author, Issue status (In library or checked

out), If book is checked out it would display, User ID & Name, Checkout date, Due date

**Result:** for checkout date and due date, did not pass

• Books checked in should not display user summary

**Result:** pass

• Books checked out should display correct user details.

**Result:** pass

**3.3.9 View student detail**

**Pass criteria:**

• Librarians can select a user record for detailed view

**Result:** pass

• The detail view should show:

1. User name, ID, Address & Phone number

**Result:** pass

**b**. The books issued by user with issue date, due date, call number, title

**Result:** did not pass

1. Late fees & Fines summary and total

**Result:** did not pass

• The display should match existing user profile

**Result:** pass

• The books checked out should have their statuses marked

**Result**: pass

• The book search query should show the user id correctly.

**Result:** pass

**3.4 Network test**

**Pass criteria:** Results of operations (ping, ftp and ODBC connectivity check) are normal

**Result:** did not test this item, because no enough machines and no available environment.

**4 ENVIRONMENTAL used**

**4.1 Hardware**: Core2Duo

**4.2 Software**: Microsoft Windows XP

**5 RESOURCES**

**5.1** Developers of the system are involved in testing process (debugging, unit testing, even integrity testing).

**5.2** Users of the system are involved in testing process (integrity testing).

**EXPERIMENT 10**

**Aim:** Study of Any Testing Tool (Win Runner).

**HP Win Runner** software was an automated functional GUI testing tool that allowed a user to record and play back user interface (UI) interactions as test scripts.

As a functional test suite, it worked with [HP Quick Test Professional](https://en.wikipedia.org/wiki/HP_QuickTest_Professional) and supported enterprise [quality assurance](https://en.wikipedia.org/wiki/Quality_assurance). It captured, verified and replayed user interactions automatically, in order to identify defects and determine whether business processes worked as designed. The software implemented a proprietary Test Script Language (TSL) that allowed customization and parameterization of user input.

HP Win Runner was originally written by **Mercury Interactive**. Mercury Interactive was subsequently acquired by Hewlett Packard (HP) in 2006. On February 15, 2008, [HP Software Division](https://en.wikipedia.org/wiki/HP_Software_Division) announced the end of support for HP Win Runner versions 7.5, 7.6, 8.0, 8.2, 9.2—suggesting migration to [HP Functional Testing](https://en.wikipedia.org/wiki/HP_Application_Lifecycle_Management#HP_Functional_Testing) software as a replacement.

Main Features of **Win Runner** are

* Developed by Mercury Interactive
* Functionality testing tool
* Supports C/s and web technologies such as (VB, VC++, D2K, Java, HTML, Power Builder, Delphe, Cibell (ERP))
* To Support .net, xml, SAP, PeopleSoft, Oracle applications, Multimedia we can use QTP.
* Win runner run on Windows only.
* Xrunner run only UNIX and Linux.
* Tool developed in C on VC++ environment.
* To automate our manual test win runner used TSL (Test Script language like c)

The main **testing Process** in Win Runner is

**1. Learning** **🡪** Recognition of objects and windows in our application by win runner is called learning. Win runner 7.0 follows Auto learning.

**2. Recording 🡪** Win runner records over manual business operation in TSL

**3. Edit Script** **🡪** Depends on corresponding manual test, test engineer inserts check points in to that record script.

**4. Run Script 🡪** During test script execution, win runner compare tester given expected values and application actual values and returns results.

**5. Analyse Results** **🡪** Tester analyzes the tool given results to concentrate on defect tracking if required.

**EXPERIMENT 11**

**Aim:** Study of Any Test Management Tool (QA Complete).



QA Complete is a powerful, flexible test management tool that helps users easily manage requirements, tests and defects all in one place. The tool is easy to use, and provides a central hub to manage and report on all of your tests – manual, Selenium, Test Complete, SoapUI.

It is customizable enough to fit into any development process, from Waterfall to Agile, and integrates tightly with the project management and workflow tools you already use, such as Jira, Bugzilla, Visual Studio and more.

**Comprehensive Software Test Management with Centralized Reporting Across Manual, Selenium, API, and Automated Tests**

QA Complete provides you with an ability to manage, plan, organize, and execute all test cases and defects associated with your release or user story in one single place

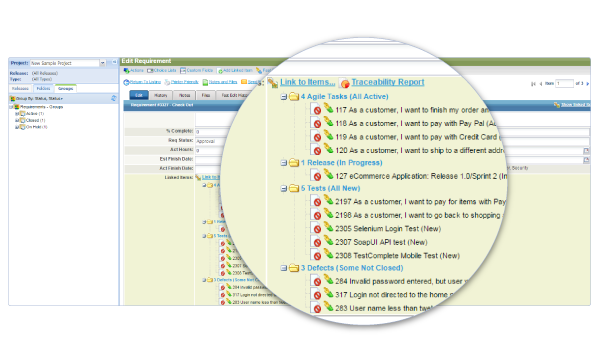
* Repurpose and reuse assets across your testing cycles.
* 0Customize the tool for your development methodology: Agile, Waterfall, KanBan, Hybrid and More.
* Determine test coverage and ensure tests exists for all requirements.
* Establish clear traceability between requirements, test cases, and defects.
* Integrate with Jira and various other project management and workflow tools.

[**CENTRALIZED TEST CASE MANAGEMENT**](https://smartbear.com/product/qacomplete/features/test-case-management)

**Organize Your Test Cases**

Everyone cares about software quality. And everyone knows that comprehensive testing improves software quality. Now you can get complete visibility into manual and automated testing assets, and drill into supporting detail quickly. QA Complete helps testing teams communicate more effectively, see all aspects of their testing efforts across the team, and stay organized across releases.

Make the most of your team's time by keeping your test management simple and centralized.



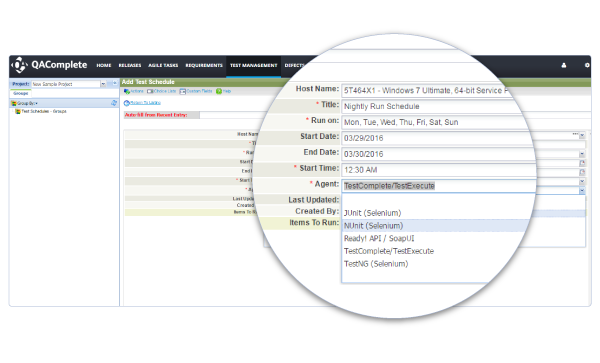
**QA Complete's centralized test management functionality:**

* link test cases to requirements
* use a centralized test library to store and reuse tests easily
* Control your test assets and actions at the test step level
* Create test suites that include both manual and automated tests

[**MANAGE TEST SCHEDULES AND EXECUTION**](https://smartbear.com/product/qacomplete/features/manage-test-hosts)

**Schedule Your Test Hosts and Cases**

With QA Complete, you can set up all of the hosts you normally use for testing and enter specific configuration information for each of them. Hosts are those systems that run our Test Execute agents and can be used to kick off automated tests. There's no need to update that configuration information each time you update the OS or deactivate an agent on that host. Our intelligent host management system can detect changes in OS levels and whether a host is available for test runs. Once you have your hosts configured, you can then link them to test schedules. Let your testing team concentrate on other activities while QAComplete handles your test schedules, running as many tests as you want based on the schedules you define. Each host can be used to run multiple tests on a variety of schedules.

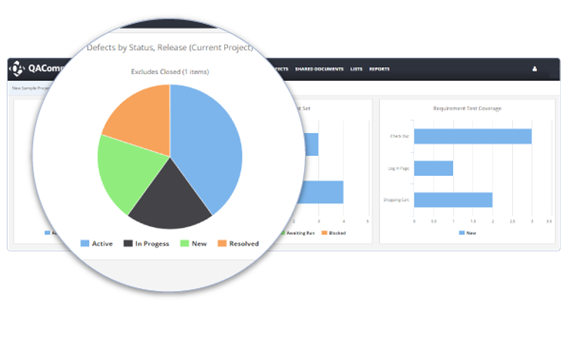


[**CREATE, MANAGE AND LOG DEFECTS**](https://smartbear.com/product/qacomplete/features/software-defect-management)

**Seamlessly Create, Track and Log Test Defects**

Save time by automatically creating and tracking defects as you test. Failed tests can automatically create defects and QA Complete's defect tracking will keep you updated on the defect resolution progress.

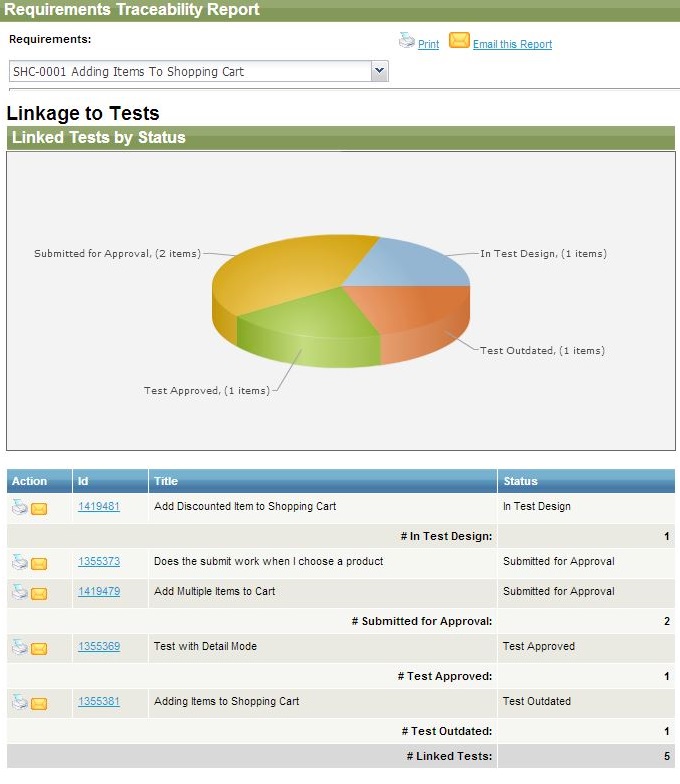
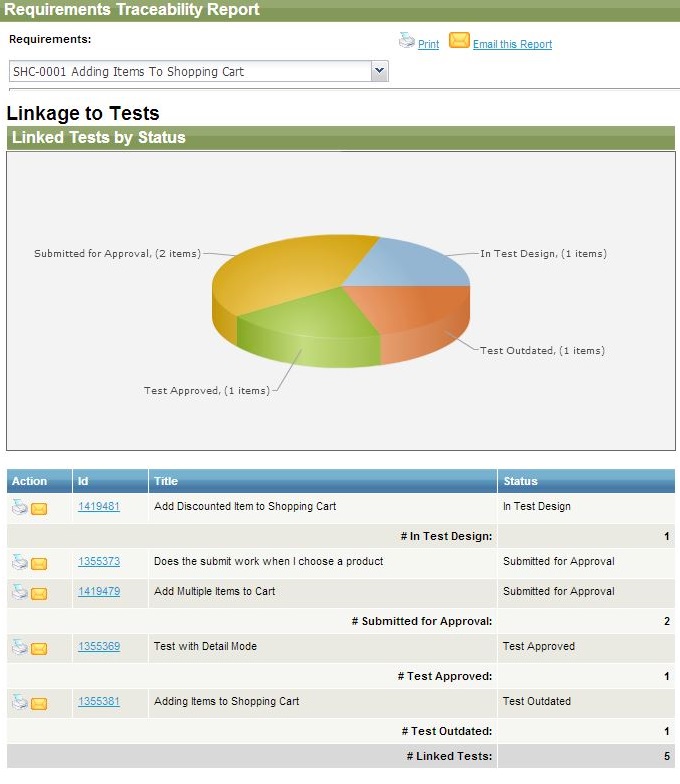
Invest more of your time running tests and let QA Complete take care of the work associated with logging defects and linking them to failed tests. QA Complete generates a defect identifier on failed test cases and allows you to track status and resolution progress for each issue.



* Integrate with external bug tracking or source code control systems for even more robust tracking and reporting of each defect
* Eliminate redundant data entry by generating defects from failed test cases using our one-click cloning capability. Cloning passes along all relevant test steps to reproduce the problem, along with expected and actual results... all without having to enter anything.

[**TRACE TESTS TO USER STORIES**](https://smartbear.com/product/qacomplete/features/requirements-traceability)

**Link**

With QAComplete's centralized test management, you can not only keep track of all of your tests across environments and test types, but you can also easily and quickly link them to user stories so you know you've covered the most important aspects of the product you are building. 

**Using QAComplete's traceability function:**

* view all relevant information in one place, including audit histories of defects, artifacts and threaded team discussion notes.
* accurately estimate the time, budget and employees needed for test.
* review information about defects, tests and requirements by severity so you can make wise decisions about where to focus your team's attention.

[**REPORT ON TEST EXECUTION**](https://smartbear.com/product/qacomplete/features/test-result-management-and-reporting-tool)

**Communicate**

Use dashboards and drill-down reporting to stay on task and ensure adequate test coverage. QA Complete gives you a host of reports in a variety of formats to meet all your needs. Our test management reporting includes status screens, dashboards, and reports that provide insight into the state of your release quality. Select from a variety of reports to make your life easier.

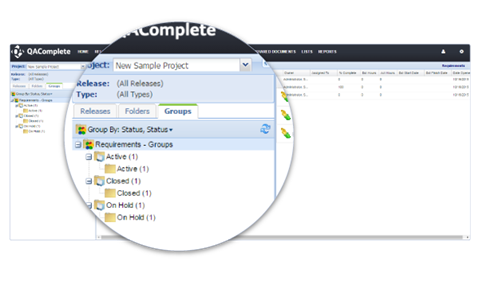


* See the results of your automated test runs together with your manual test results
* Make smarter decisions about the readiness of each release
* Get easy visibility into how many of your automated tests have run or awaiting execution

[**GET FULL VISIBILITY INTO TESTING PROCESS**](https://smartbear.com/product/qacomplete/features/have-full-visibility-of-your-testing-process)

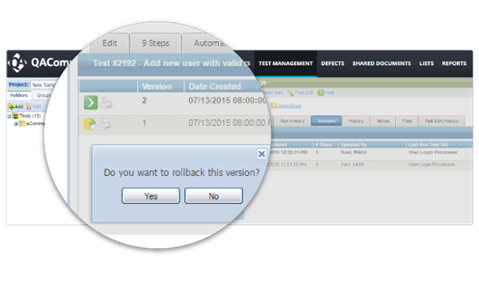
**Extensible platform for requirement management**

* Easily define and track requirements and user story for different releases within QAComplete
* Link existing Jira and Bugzilla issues to requirements and test cases
* Decide which manual test cases for requirements can be automated to reduce costs



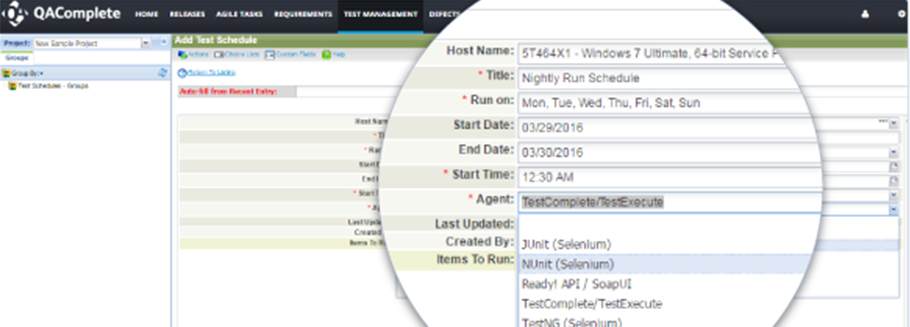
**Archive & Audit changes**

* Create baseline test versions and access past results at any time for performing comparison analysis within QA Complete
* Or integrate with source control systems like Git, Subversion, and Mercurial for test case versioning
* Maintain a log of changes made to test cases for regulatory purposes
* Archive your test results to keep a track of different versions and ensure transparency



### Schedule test runs

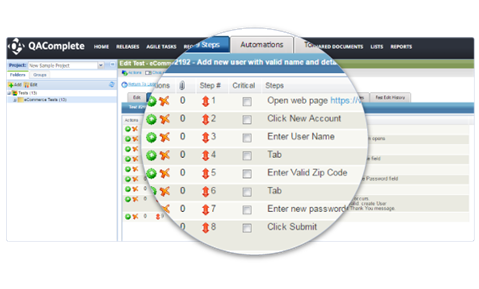
* Use configurations that let you verify tests against any platform or environment
* Organize your test runs in plans to track results for sprints and iterations



[**SHIP SOFTWARE WITH CONFIDENCE**](https://smartbear.com/product/qacomplete/features/ship-software-with-confidence)

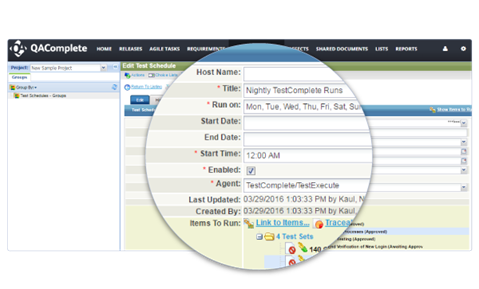
﻿**Reduce Manual Testing Efforts**

* Leverage detailed reports to decide which manual test cases can be automated to reduce costs.
* Enable multiple teams members to work in parallel on test design.
* Track changes and maintain history for versioning.
* Import already existing manual tests into QAComplete.
* Link manual tests to requirements, test cases, and reported defects.



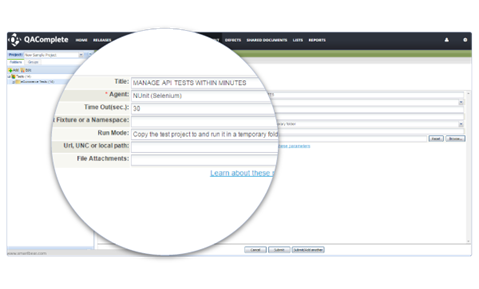
**Get more from your Existing Test Complete Tests**

* Combine automated functional TestComplete tests with manual, exploratory, and API tests.
* Get a single report across different tests, multiple test runs, and system configurations.
* Create test schedules to kick off automated tests without manual intervention.
* Set-up test suites combining manual and automated TestComplete tests.



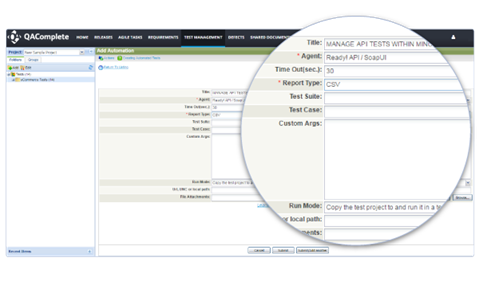
**Manage open Source Tests within Minutes**

* Zero configuration required for managing open source tests such as Selenium and SoapUI.
* Avoid tedious manual steps by reusing tests across releases and user stories.
* Integrate with requirement and defect management systems Jira, Bugzilla.
* Lower response times by setting up alerts and notifications.



**Get accurate insights into your API Tests**

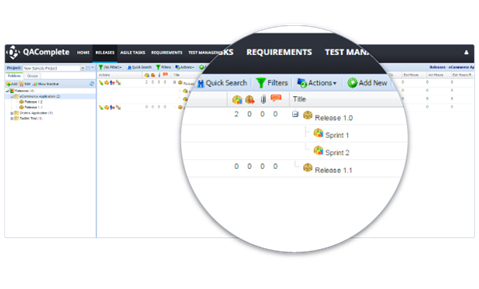
* Establish clear traceability between requirements, API test cases, and defects.
* Ensure proper coverage exists for internal and third-party APIs.
* Combine front-end tests with back-end API tests together into one test suite.
* Set up baseline and maintain full history of changes made to API tests.



[**FITS SEAMLESSLY INTO YOUR EXISTING PROCESSES**](https://smartbear.com/product/qacomplete/features/fits-seamlessly-into-your-existing-processes)

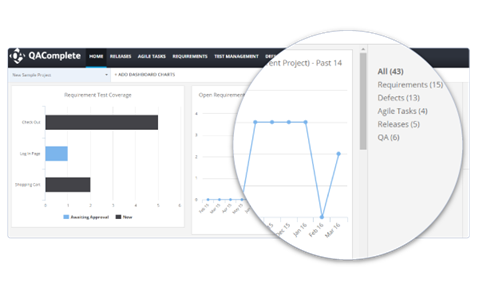
**Scale Agile fast , Easily & Smartly**

* Plan, track, and report user stories, test cases, and defects across iterations.
* Leverage TestComplete integration for implementing test-driven development (TDD).
* Establish traceability between user story and other testing artifacts.
* Reuse user stories, test cases, and defects across iterations.
* Track team capacity to adjust workload and assignments.



**Complete visibility into Waterfall processes**

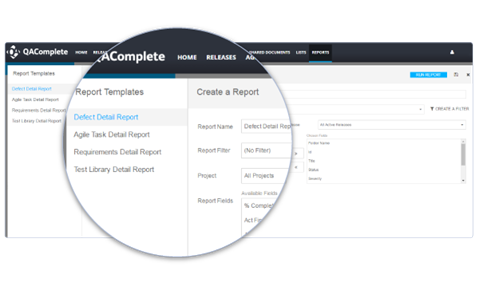
* Get detailed information on all the projects, milestones, releases, and test runs.
* Integrated with defect and requirement management systems Jira, Bugzilla, and TFS.
* Set-up customizable workflows to automate approval process for test and requirements.
* Maintain full test case history to avoid tedious manual steps.



[**MAKE INFORMED DECISIONS**](https://smartbear.com/product/qacomplete/features/make-informed-decisions)

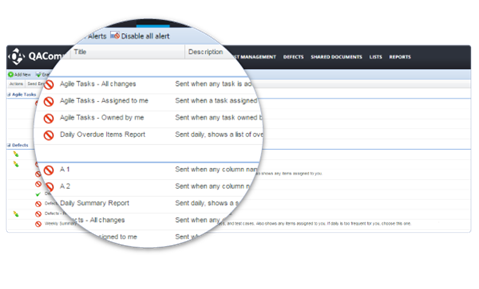
**Get deep visibility into Testing processes**

* Track the coverage of your tests using comprehensive reports
* Get a single report for different tests run against any platform or environment
* Leverage end-to-end traceability reports for requirements, tests, and defects
* Quickly identify which tests can be automated to reduce cost and time.



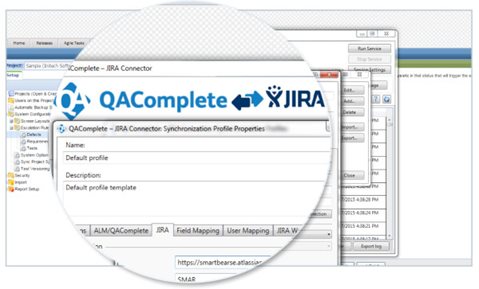
**Define Escalation Rules to Quickly Resolve Bottlenecks**

* Reduce false alerts by setting up notifications
* Revert back to previous versions when needed
* Drill down into individual defects to trace back to requirements and user stories.



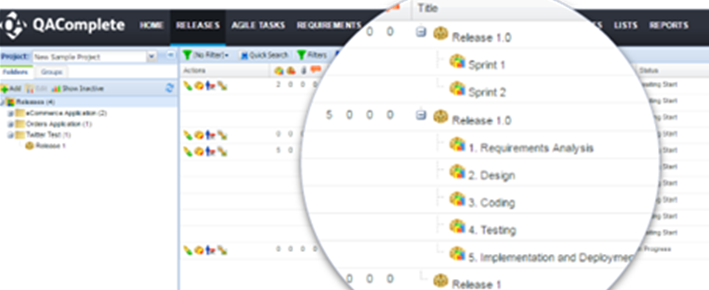
B**ring in data from existing Tools to get End-to-End Analytics**

* Stop building integration by out of box plugins
* Full two-way integration with Jira and other tools
* API access available for bringing in third-party data



**Manage both Agile & Waterfall in one Interface**

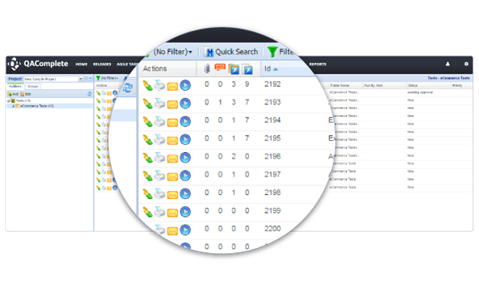
* Reuse test assets across agile and waterfall methodology.
* Define custom fields or change other aspects of the application as per your development methodology.
* Avoid tedious manual steps by reusing test assets across different methodologies.



[**SAVE TIME AND INCREASE PRODUCTIVITY**](https://smartbear.com/product/qacomplete/features/save-time-and-increase-productivity)

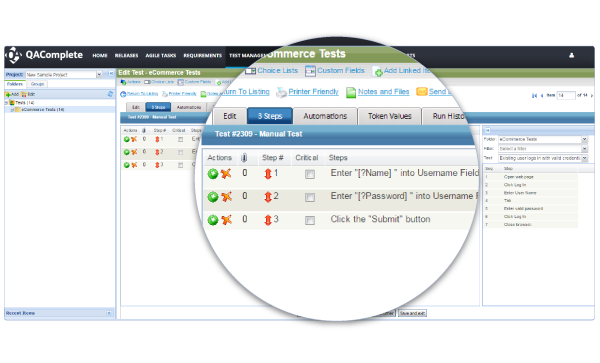
**Increase Productivity through Resuse**

* Reuse Selenium, SoapUI, TestComplete and other automated and/or manual tests
* Repurpose tests, data, and other assets across releases and sprints
* Define and use custom fields and roles across projects to reduce setup time
* Organize automated and/or manual tests in suites for reuse across projects.



**Save Time by Reducing Manual steps**

* Pull in data from external sources such as CSV to increase test coverage.
* Set job schedules and kick-off automated tests without manual intervention
* File defects in external tools such as Jira right from QAComplete
* Use out of the box plugins to integrate with third party tools like Jira.



**EXPERIMENT 12**

**Aim:** Automate the Test cases using Test Automation tool (using QA Complete).

QA Complete gives the test team a single application for managing test cases, test environments, automated tests, defects and testing project tasks. QA Complete is the all-encompassing solution, providing visibility of your test management process and ensuring the delivery of high quality software. QA Complete allows you to manage your manual test cases and control your automated tests, giving you one place from which to coordinate all of your testing.

**Key Features:  
 1. Test Case Management 🡪** The ability to organize, plan, and analyze your testing efforts across the lifecycle is critical to your success or failure whether you use manual or automated test cases today. As projects cope with fewer development resources, higher quality expectations, and shorter development timelines, any serious development effort needs better test case management. QA Complete delivers: -

* Manage manual test cases and link them back to the original requirements, thereby ensuring a requirement has been met. Evaluate the test run history of those automated tests right from QA Complete.
* Employ re-usable manual test libraries to quickly create new test scenarios.
* Add, print, edit, or email test cases with a single click.

**2. Test Automation Tool Integrations 🡪** QA Complete supports many automated testing tools, including Automated QA Test Complete and HP Quick Test Pro. Integration with test automation tools allows you review the run history of any automated test on any machine, so if you have a test lab with multiple machines running automated tests, you can compare machine run history. Since you can co-ordinate both manual and automated tests, you have better test information to make release decisions. By integrating automated testing into QA Complete, you can:

* Launch the tests from within your automated tool and automatically report the run information to QA Complete for analysis of runs over time.
* Trend results using graphical dashboards and schedule tests to run unattended.

**3. Bi-Directional Traceability 🡪** The goal of traceability is to ensure “adequate” test coverage for each software requirement. It is important to maintain traceability both forwards and backwards, from requirement to test case and from test case to requirement. This ensures that design specifications are appropriately verified and that requirements are appropriately validated, ultimately reducing software defects.

* Link together requirements, test cases, and reported defects.
* Drag and drop functionality to link test cases or defects to a requirement.
* With one click, see a traceability report showing all linkages to a particular requirement.

**4. Requirements Management 🡪** QA Complete helps you manage requirements regardless of your team’s development methodology. It lets you define requirements for each release and track the release for which each requirement is scheduled. Workflow configuration allows users to define their own requirement statuses and add custom workflow rules (like automated assignment and state transition rules). Integrated linking with test cases delivers clear requirements test coverage reports.

**5. Defect and Issue Tracking 🡪** QA Complete allows you to track status and resolution progress of defects and issues for each release. Instead of spending your time entering data, the software automatically generates a defect identifier on failed test cases, so testers can invest their time running the tests, not administration. Integration with Atlassian JIRA, Bugzilla, and other web-based defect tracking tools allow you to blend QA Complete features with the defect tracking tools your organization already uses.

* Coordinate QA and development teams to coordinate activities as bugs are found. QA Complete has a full featured defect tracking component.
* If your team already owns a bug tracking system (like JIRA, Bugzilla, Microsoft TFS, etc.), you can create defects inside of QA Complete and have those automatically synchronized with your bug tracking system.
* Defect reports and dashboards show defects by severity, priority, or other criteria.

**6. Seamless Integration with other ALM tools 🡪** QA Complete can seamlessly integrate with many defect and ALM tools including Microsoft Team Foundation Server (TFS), HP Quality Center, IBM Rational Doors, IBM Rational Requisite Pro, Rational Team Concert, Atlassian JIRA, Rally, Version One, Bugzilla, and Accept 360. If you are using any of these tools for defect or requirements management, you can enter defects or requirements in QA Complete and automatically sync them to the other system. Likewise, if you add or update items in the other system, it can automatically sync those changes to QA Complete.

**7. Seamless Integration with Source Control Systems 🡪** Associating defects and requirements with source code provides great traceability, allowing you to quickly discover troublesome code and requirements that required the most re-work. Using the QA Complete SCM integrators, you can associate defects or requirements when checking in source code. Example of SCM systems: Subversion, Perforce, CVS, and Microsoft Team Foundation Server (TFS).

**8. Extensive Dashboards and Reporting for QA Activities 🡪** QA Complete provides an array of analysis tools. Status screens, dashboards, and reports help you stay on track, better plan your next release, and answer the most pressing questions about your software development projects.

**9. Easy Data Entry 🡪** QA Complete makes it easy for you to automatically generate a defect from a failed test case. With one-click cloning of records, you can pass along to your development team all the relevant steps to reproduce the problem, along with expected and actual results, without having to re-enter anything.

* Automatically create a link between the defect and the test case.
* Existing test cases linked to a requirement automatically link the defect to the requirement as well for full traceability.

**10. Web-Based User Interface 🡪** QA Complete has a Web interface. Nothing needs to be installed on your hard drive.

* Users with an Internet connection may access QA Complete from home or anywhere in the world.
* Distributed and offshore teams can easily share QA artifacts with one another and local teams.
* Supports all major browsers, including Internet Explorer, Fire Fox, Safari, and Google Chrome.

**EXPERIMENT 13**

**Aim**: Learn how to raise and report Bugs using Bug tracking tool

(Bugzilla, Jira using QA Complete).

**Bugzilla** is an open-source issue/bug tracking system that allows developers effectively to keep track of outstanding problems with their product. It is written in Perl and uses MYSQL database.

Bugzilla is a defect tracking tool, however it can be used as a test management tool as such it can be easily linked with other test case management tools like Quality Center, Testlink etc.

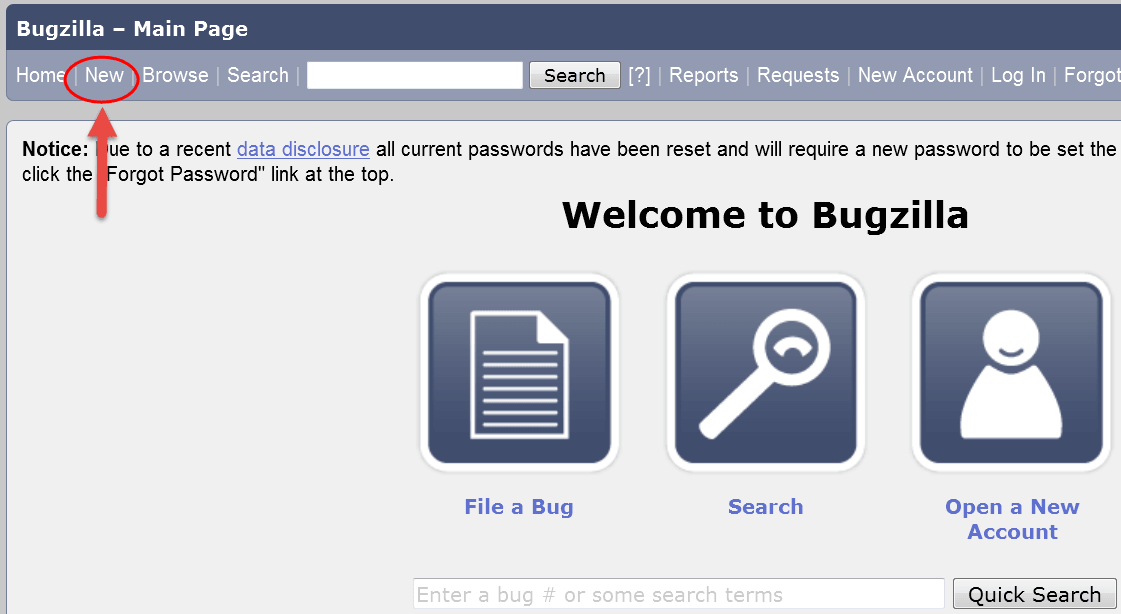
This open bug-tracker enables users to stay connected with their clients or employees, to communicate about problems effectively throughout the data-management chain.

**Key features** of Bugzilla include: -

* Advanced search capabilities
* E-mail Notifications
* Modify/file Bugs by e-mail
* Time tracking
* Strong security
* Customization
* Localization

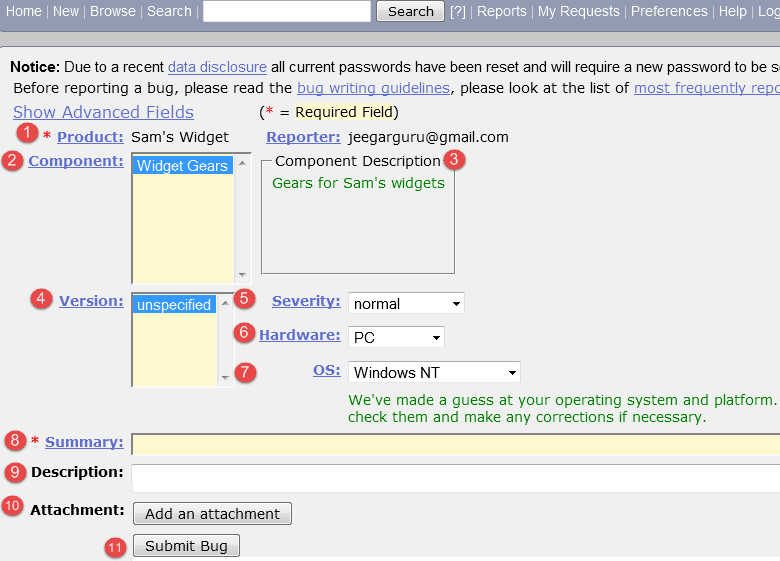
**Creating a bug report in BUGZILLA: -**

**Step 1:** To create a new bug in Bugzilla, visit the home-page of Bugzilla and click on **NEW** tab from the main menu

[](http://cdn.guru99.com/images/1-2015/011015_0940_BugzillaTut4.png)

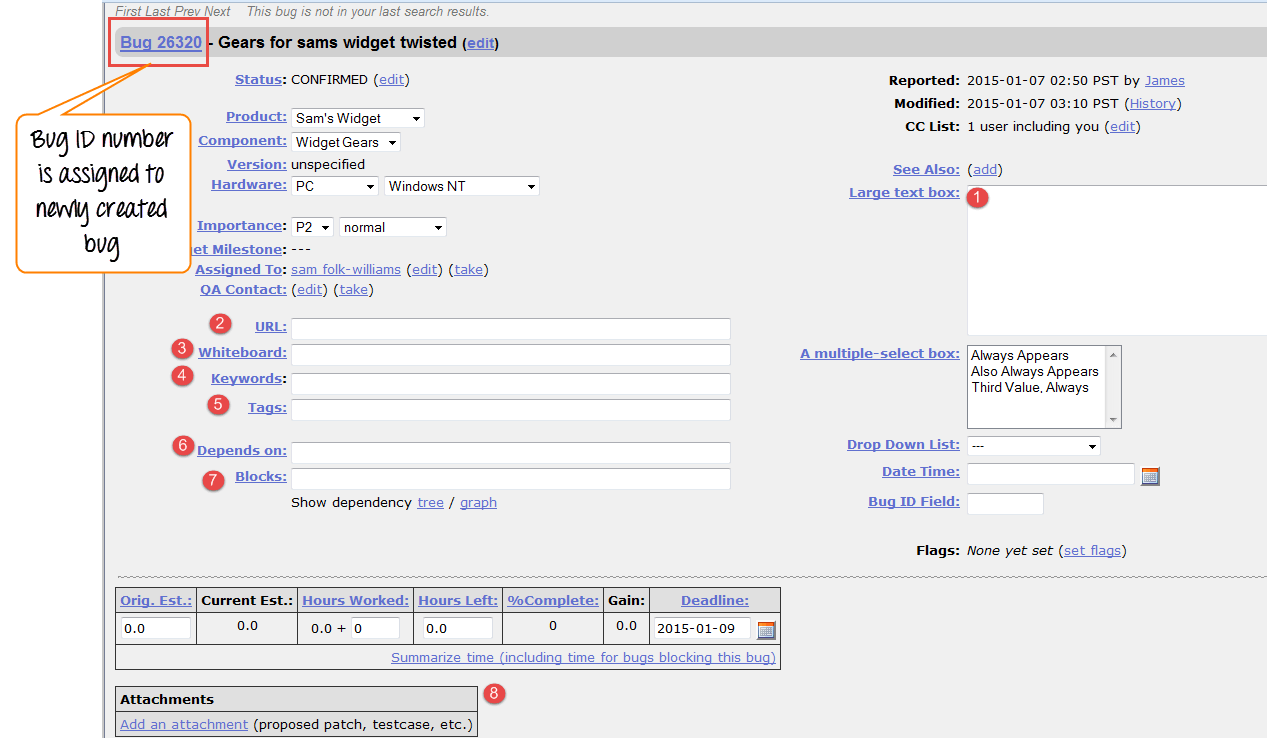
**Step 2:** In the next window

1. Enter Product
2. Enter Component
3. Give Component description
4. Select version,
5. Select severity
6. Select Hardware
7. Select OS
8. Enter Summary
9. Enter Description
10. Attach Attachment
11. Submit

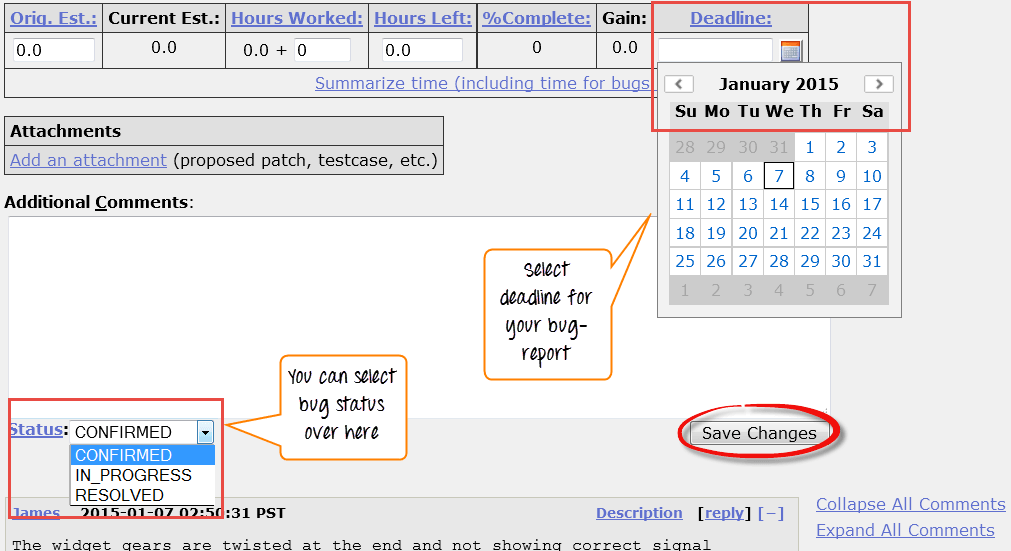
[](http://cdn.guru99.com/images/1-2015/011015_0940_BugzillaTut5.png)

**Step 3: Bug is created** ID# 26320 is assigned to our Bug. You can also add additional information to the assigned bug like URL, keywords, whiteboard, tags, etc. This extra-information is helpful to give more detail about the Bug you have created.

1. Large text box
2. URL
3. Whiteboard
4. Keywords
5. Tags
6. Depends on
7. Blocks
8. Attachments

[](http://cdn.guru99.com/images/1-2015/011015_0940_BugzillaTut7.png)

**Step 4:** In the same window if you scroll down further. You can select deadline date and also status of the bug. **Deadline in Bugzilla usually gives the time-limit to resolve the bug in given time frame.** 

[](http://cdn.guru99.com/images/1-2015/011015_0940_BugzillaTut8.png)

**EXPERIMENT 14**

**Aim:** Study of any open source testing tool (Web Performance Analyzer/O STA).

[OpenSTA](http://cdn.softwaretestinghelp.com/wp-content/qa/uploads/2012/09/OpenSTA.jpg)

O STA stands for Open System Testing Architecture. This is a GUI based performance tool used by application developers for load testing and analyzing. This is believed to be a complex tool among the all other performance testing tools. It has proven capability in the past and the current toolset is capable of performing the heavy load test and analyses for the scripted HTTP and HTTPS. Here, the testing is carried out by using the recordings and simple scripts. To carry out the test successfully, results and other statistics are taken through various test runs. These data and results can be later exported to software for creating reports. This is a free testing tool and it is distributed under GNU GPL and it will remain free forever. This tool was originally developed by Cyrano, which was later taken over by Quotium.

**Open STA** **System Requirement:**  Open STA runs only on Windows operating system.Open STA is a distributed software testing architecture designed around CORBA, it was originally developed to be commercial software by [CYRANO](http://portal.opensta.org/faq.php?topic=CYRANO). The current toolset has the capability of performing scripted HTTP and HTTPS heavy load tests with performance measurements from Win32 platforms. However, the architectural design means it could be capable of much more.

**Web Load Testing** (HTTP Stress & Performance Tests):

|  |
| --- |
|  |

The applications that make up the current OpenSTA toolset were designed to be used by performance testing consultants or other technically proficient individuals. This means testing is performed using the record and replay metaphor common in most other similar commercially available toolsets. Recordings are made in the tester's own browser producing simple scripts that can be edited and controlled with a special high-level scripting language. These scripted sessions can then be played back to simulate many users by a high-performance load generation engine. Using this methodology, a user can generate realistic heavy loads simulating the activity of hundreds to thousands of virtual users.

**Data Collection** (Timers, Windows Performance & SNMP Statistics):

Results and statistics are collected during test runs by a variety of automatic and user-controlled mechanisms. These can include scripted timers, SNMP data, Windows Performance Monitor stats and HTTP results & timings. Much of the data logged can be monitored live during the test runs; once test runs are complete, logs can be viewed, graphed, filtered and exported for use by more sophisticated report generation software.

**Completely Free & Open Source:**

**[Thanks to SourceForge.net!](http://opensta.sf.net/)**

|  |
| --- |
|  |

The Open STA toolset is Open Source software licensed under the GNU GPL (General Public License), this means it is free and will always remain free. If you wish to build your own customized version of Open STA or take part in its development then the complete toolset source code, buildable in Microsoft Visual Studio 6, and all related information is available from [OpenSTA.SourceForge.net](http://opensta.sf.net/), the developer home site.

**EXPERIMENT 15(a)**

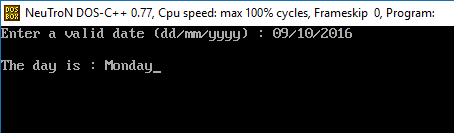
**Aim:** Write a program for the determination of day of the given Date and perform slice-based testing for all variables.

# include <stdio.h>

# include <conio.h>

1. void main()
2. {
3. int month[12] = {31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};
4. char week[7][10] ;
5. int date, mon, year, i, s = 0 ;
6. clrscr();
7. strcpy(week[0], "Sunday") ;
8. strcpy(week[1], "Monday") ;
9. strcpy(week[2], "Tuesday") ;
10. strcpy(week[3], "Wednesday") ;
11. strcpy(week[4], "Thursday") ;
12. strcpy(week[5], "Friday") ;
13. strcpy(week[6], "Saturday") ;
14. printf("Enter a valid date (dd/mm/yyyy) : ") ;
15. scanf("%d / %d / %d", &date, &mon, &year) ;
16. if( (year % 400 == 0) || ((year % 4 == 0) && (year % 100 != 0)) )
17. month[1] = 29 ;
18. for(i = 0 ; i < mon - 1 ; i++)
19. s = s + month[i] ;
20. s = s + (date + year + (year / 4) - 2) ;
21. s = s % 7 ;
22. printf("\nThe day is : %s", week[s]) ;
23. getch() ;
24. }

**OUTPUT:**



**SLICE BASED TEST CASES:**

There are seven variables – month [], week [], date, mon, year, i, s. The slices of these variables are:

1. S (month [], 24) = (1, 2, 3, 17, 19, 23, 24)
2. S (week [], 24) = (1,2, 4, 7-13, 22-24)
3. S (date, 24) = (1-2, 5, 14-15, 20, 23-24)
4. S (mon, 24) = (1, 2, 5, 14-15, 18, 23, 24)
5. S (year, 24) = (1, 2, 5, 14-15, 16, 20, 23-24)
6. S (i, 24) = (1, 2, 5, 18-19, 23-24)
7. S (s, 24) = (1-5, 7-24)

**SLICE BASED TEST CASES TABLE:**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **S. No.** | **Slices** | **Lines covered** | date | **mon** | **year** | **Actual Output** |
| 1 | S (month [], 24) | 1, 2, 3, 17, 19, 23, 24 | - | - | - | No output |
| 2 | S (week [], 24) | 1,2, 4, 7-13, 22-24 | - | - | - | Garbage value |
| 3 | S (date, 24) | 1-2, 5, 14-15, 20, 23-24 | 11 | 06 | 16 | No output |
| 4 | S (mon, 24) | 1, 2, 5, 14-15, 18, 23, 24 | 20 | 09 | 15 | No output |
| 5 | S (year, 24) | 1, 2, 5, 14-15, 16, 20, 23-24 | 01 | 12 | 12 | No output |
| 6 | S (i, 24) | 1, 2, 5, 18-19, 23-24 | - | - | - | No output |
| 7 | S (s, 24) | 1-5, 7-24 | 09 | 10 | 16 | Monday |

**EXPERIMENT 15(b)**

**AIM:** To Perform slice-based testing for all variables for program to compute total salary of an employee, given his basic salary. (Given: HRA=3% of basic, DA=8% of basic, CCA/MA=Rs 100/-, I. Tax=300/-, PF=780, TA=Rs 800/-).

#include<stdio.h>

#include<conio.h>

1. void main () {

2. int total\_salary, basic, da, hra;

3. int ma=100, ta=800, i\_tax=300, pf=780;

4. printf ("Enter basic salary: ");

5. scanf ("%d", &basic);

6. da = (8 \* basic) / 100;

7. hra = (3 \* basic) / 100;

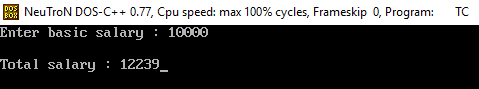
8. total\_salary = basic + da + hra + ma + ta + i\_tax + pf;

9. printf ("\nTotal salary: %d", total\_salary);

10. getch ();

11. }

**OUTPUT:**



**SLICE BASED TEST CASES:**

There are eight variables – total\_salary, basic, da, hra, ma, ta, i\_tax, pf. The slices of these variables are:

1. S (total\_salary,11) = (1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11)
2. S (basic, 11) = (1-2, 4-8, 10, 11)
3. S (da, 11) = (1-2, 4-6, 8, 10, 11)
4. S (hra, 11) = (1, 2, 4, 5, 7, 8, 10, 11)
5. S (ma, 11) = (1, 3, 8)
6. S (ta, 11) = (1, 3, 8)
7. S (i-tax, 11) = (1, 3, 8)
8. S (pf, 11) = (1, 3, 8)

**SLICE BASED TEST CASES TABLE:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **S. No.** | **Slices** | **Lines covered** | basic | **Expected Output** | **Actual Output** |
| 1 | S (total\_salary, 11) | 1-11 | 10000 | 12,239 | 12,239 |
| 2 | S (basic, 11) | 1-2, 4-8, 10-11 | 10000 | 11,100 | 11,100 |
| 3 | S (da, 11) | 1-2, 4-6, 8, 10-11 | 10000 | 10,800 | 10,800 |
| 4 | S (hra, 11) | 1-2, 4-5, 7-8, 10, 11 | 10000 | 10,300 | 10,300 |
| 5 | S (ma, 11) | 1, 3, 8 | - | No output | No output |
| 6 | S (ta, 11) | 1, 3, 8 | - | No output | No output |
| 7 | S (i\_tax, 11) | 1, 3, 8 | - | No output | No output |
| 8 | S (pf, 11) | 1, 3, 8 | - | No output | No output |

**EXPERIMENT 16**

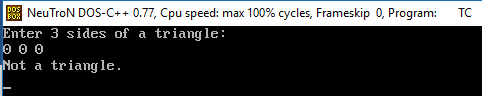
**Aim:** Write a program in C/C++ to find whether a triangle is right, acute and obtuse. Its input is a triplet of 3 positive integers (say a, b, c) from interval (1 to 100). Perform slice-based testing and generate test cases for the given program.

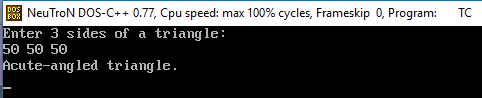
#include<iostream.h>

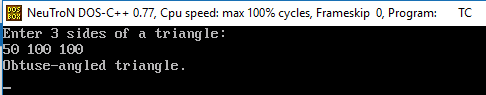
#include<conio.h>

1. void main()
2. {
3. int x,y,z;
4. clrscr();
5. cout<<"Enter 3 sides of a triangle:\n";
6. cin>>x>>y>>z;
7. if((x<=0) || (y<=0) || (z<=0))
8. { cout<<"Not a triangle.\n";
9. }
10. else
11. {
12. if((x + y <= z) || (x + z <= y) || (y + z <= x))
13. {cout<<"Not a triangle.\n";
14. }
15. else
16. {
17. if (((x \* x) + (y \* y) == (z \* z)) || ((x \* x) + (z \* z) == (y \* y)) || ((z \* z) + (y \* y) == (x \* x)))
18. cout<<"Right-angled triangle.\n";
19. else if ((((x \* x) + (y \* y) < (z \* z)) || ((x \* x) + (z \* z) < (y \* y)) || ((z \* z) + (y \* y) < (x \* x))) || ((x<=z && y<=z) || (x<=y && z<=y) || (y<=x && z<=x)))
20. { cout<<"Acute-angled triangle.\n";
21. }
22. else if((((x \* x) + (y \* y) > (z \* z)) || ((x \* x) + (z \* z) > (y \* y)) || ((z \* z) + (y \* y) > (x \* x))) || ((x>z && y>z) || (x>y && z>y) || (y>x && z>x)))
23. { cout<<"Obtuse-angled triangle.\n";
24. }
25. else
26. { cout<<"Not a triangle\n";
27. }
28. }
29. }
30. getch();
31. }

**OUTPUT:**







**SLICE BASED TESTING TEST CASES:**

There are 3 variables – a, b, c. The slices of these variables are:

1. S (x, 9) = (1-3, 5-9, 30, 31)
2. S (x, 14) = (1-3, 5-7, 10-14, 29, 30, 31)
3. S (x, 21) = (1-3, 5-7, 10-12, 15-21, 28, 29, 30, 31)
4. S (x, 31) = (1-3, 5-7, 10-12, 15-17, 19, 22, 25, 27-31)
5. S (y, 9) = (1-3, 5-9, 30, 31)
6. S (y, 17) = (1-3, 5-7, 10-12, 15-17, 28, 29, 30, 31)
7. S (y, 22) = (1-3, 5-7, 10-12, 15-17, 19-22, 28, 29, 30, 31)
8. S (y, 31) = (1-3, 5-7, 10-31)
9. S (z, 9) = (1-3, 5-9, 30, 31)
10. S (z, 14) = (1-3, 5-7, 10-12, 14, 30, 31)
11. S (z, 24) = (1-3, 5-7, 10-24, 28, 29, 30, 31)

**SLICE BASED TEST CASES TABLE:**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **S. No.** | **Slices** | **Lines covered** | x | y | z | **Actual Output** |
| 1 | S (x, 9) | 1-3, 5-9, 30-31 | 0 | 50 | 50 | Not a triangle. |
| 2 | S (x, 14) | 1-3, 5-7, 10-14, 29-31 | 50 | 50 | 100 | Not a triangle. |
| 3 | S (x, 21) | 1-3, 5-7, 10-12, 15-21, 28-31 | 3 | 4 | 5 | Right-angled triangle. |
| 4 | S (x, 31) | 1-3, 5-7, 10-12, 15-17, 19, 22, 25, 27-31 | 50 | 99 | 100 | No output |
| 5 | S (y, 9) | 1-3, 5-9, 30-31 | 50 | 0 | 50 | Not a triangle. |
| 6 | S (y, 17) | 1-3, 5-7, 10-12, 15-17, 28-31 | 99 | 50 | 100 | No output |
| 7 | S (y, 22) | 1-3, 5-7, 10-12, 15-17, 19-22, 28-31 | 50 | 50 | 50 | Acute-angled triangle. |
| 8 | S (y, 31) | 1-3, 5-7, 10-31 | 50 | 100 | 100 | Obtuse-angled triangle. |
| 9 | S (z, 9) | 1-3, 5-9, 30-31 | 50 | 50 | 0 | Not a triangle. |
| 10 | S (z, 14) | 1-3, 5-7, 10-12, 14, 30-31 | 100 | 99 | 50 | No output |
| 11 | S (z, 24) | 1-3, 5-7, 10-24, 28-31 | 0 | 0 | 0 | Not a triangle. |

**EXPERIMENT 17**

**Aim:** Write a program to find the largest of three 3 numbers. The test suite selected by a testing technique-

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| S. No. | A | B | C | Expected O/P |
| 1 | 6 | 10 | 2 | 10 |
| 2 | 10 | 6 | 2 | 10 |
| 3 | 6 | 2 | 10 | 10 |
| 4 | 6 | 10 | 20 | 20 |

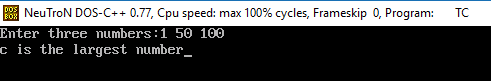
Create five mutants (M1 to M5) and calculate mutation score of this test suite.

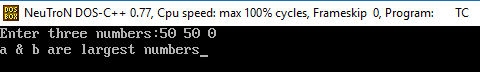
#include<iostream.h>

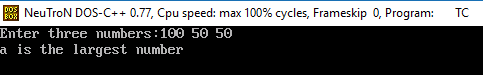
#include<conio.h>

1. void main()
2. { int a,b,c;
3. clrscr();
4. cout<<"Enter three numbers:";
5. cin>>a>>b>>c;
6. if(a>b)
7. { if(a>c)
8. { cout<<a;
9. }
10. else
11. { cout<<c;
12. }
13. }
14. else
15. { if(b>c)
16. { cout<<b;
17. }
18. else
19. { cout<<c;
20. }
21. }
22. getch();
23. }

**OUTPUT:**







**MUTATION TESTING TEST CASES:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Mutant No.** | **Line No.** | **Original Line** | **Modified Line** |
| M1 | 6 | if (a>b) | if (a!=b) |
| M2 | 8 | cout<<a | cout<<c |
| M3 | 7 | if (a>c) | if (a==c) |
| M4 | 15 | If (b>c) | If (c>b) |
| M5 | 19 | cout<<c | cout<<a |

**ACTUAL OUTPUT OF MUTANT NO. M1:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Test Cases** | a | b | c | **Expected Output** | **Actual Output** |
| 1 | 6 | 10 | 2 | 10 | 6 |
| 2 | 10 | 6 | 2 | 10 | 10 |
| 3 | 6 | 2 | 10 | 10 | 10 |
| 4 | 6 | 10 | 20 | 20 | 20 |

**ACTUAL OUTPUT OF MUTANT NO. M2:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Test Cases** | a | b | c | **Expected Output** | **Actual Output** |
| 1 | 6 | 10 | 2 | 10 | 10 |
| 2 | 10 | 6 | 2 | 10 | 2 |
| 3 | 6 | 2 | 10 | 10 | 10 |
| 4 | 6 | 10 | 20 | 20 | 20 |

**ACTUAL OUTPUT OF MUTANT NO. M3:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Test Cases** | a | b | c | **Expected Output** | **Actual Output** |
| 1 | 6 | 10 | 2 | 10 | 10 |
| 2 | 10 | 6 | 2 | 10 | 2 |
| 3 | 6 | 2 | 10 | 10 | 10 |
| 4 | 6 | 10 | 20 | 20 | 20 |

**ACTUAL OUTPUT OF MUTANT NO. M4:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Test Cases** | a | b | c | **Expected Output** | **Actual Output** |
| 1 | 6 | 10 | 2 | 10 | 2 |
| 2 | 10 | 6 | 2 | 10 | 10 |
| 3 | 6 | 2 | 10 | 10 | 10 |
| 4 | 6 | 10 | 20 | 20 | 10 |

**ACTUAL OUTPUT OF MUTANT NO. M5:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Test Cases** | a | b | c | **Expected Output** | **Actual Output** |
| 1 | 6 | 10 | 2 | 10 | 10 |
| 2 | 10 | 6 | 2 | 10 | 10 |
| 3 | 6 | 2 | 10 | 10 | 10 |
| 4 | 6 | 10 | 20 | 20 | 6 |