22/10/2020 **Mohd. Aftab Alam**
02013302717

EXPERIMENT 7

Aim :

(a) Write a program in C/C++ for classification of Triangle on the basis of angle. Its input is triple of positive integers (say a, b, c) and values may be from the interval [1, 100]. The output may have one of the following: Right Angled, Acute Angled, Obtuse Angled or Not a Triangle.

Perform Slice-based testing and generate test cases.

Algorithm :

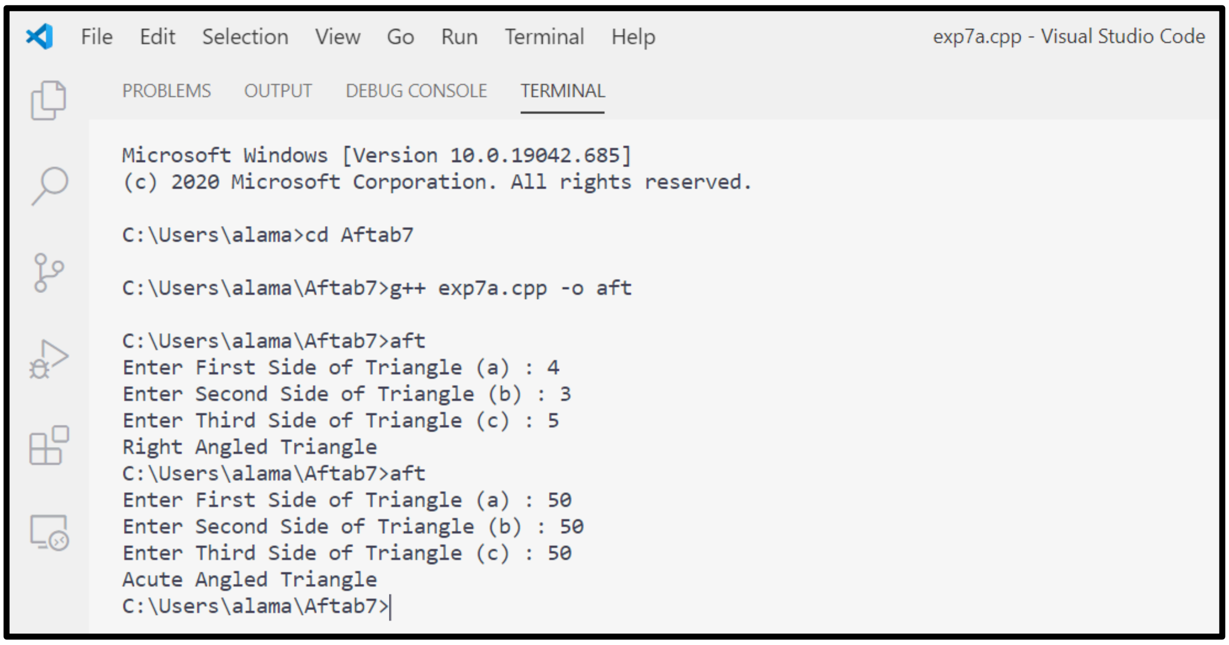
- Take three inputs from the user as the sides of the triangle.
- Check whether they lie in the given interval.
- If the condition is true then check if it is a Triangle or not.
- If the condition is true then put valid = 1, otherwise, put valid = -1.
- If valid = 1 then calculate the value of a1, a2, and a3 as follows:
    - **a1 = (a2 + b2) / c2**
    - **a2 = (b2 + c2) / a2**
    - **a3 = (c2 + a2) / b2**
- Check if a1, a2, or a3 are less than 1.
- If the condition is true then it is an Obtuse Angled Triangle.
- Else if a1, a2 or a3 are equal to 1.
- If the condition is true then it is a Right-Angled Triangle.
- If the condition is false then it is an Acute Angled Triangle.
- Else if valid = -1, then it is an Invalid Triangle.
- Else the input values are Out of Range.

Code :

```cpp
#include <iostream>
using namespace std;

int main()
{
    float a, b, c;
    float a1, a2, a3;
    int valid = 0;
    cout << "Enter First Side of Triangle (a) : ";
    cin >> a;
    cout << "Enter Second Side of Triangle (b) : ";
    cin >> b;
    cout << "Enter Third Side of Triangle (c) : ";
    cin >> c;
    if (a > 0 && a <= 100 && b > 0 && b <= 100 && c > 0 && c <= 100)
    {
        if ((a + b) > c && (b + c) > a && (c + a) > b)
            valid = 1;
        else
            valid = -1;
    }
    if (valid == 1)
    {
        a1 = (a * a + b * b) / (c * c);
        a2 = (b * b + c * c) / (a * a);
        a3 = (c * c + a * a) / (b * b);
        if (a1 < 1 || a2 < 1 || a3 < 1)
            cout << "Obtuse Angled Triangle";
        else if (a1 == 1 || a2 == 1 || a3 == 1)
            cout << "Right Angled Triangle";
        else
            cout << "Acute Angled Triangle";
    }
    else if (valid == -1)
        cout << "Invalid Triangle";
    else
        cout << "Out of Range";
    return 0;
}
```

Output Screenshots :



```
Microsoft Windows [Version 10.0.19042.685]
(c) 2020 Microsoft Corporation. All rights reserved.

C:\Users\alama>cd Aftab7

C:\Users\alama\Aftab7>g++ exp7a.cpp -o aft

C:\Users\alama\Aftab7>aft
Enter First Side of Triangle (a) : 4
Enter Second Side of Triangle (b) : 3
Enter Third Side of Triangle (c) : 5
Right Angled Triangle
C:\Users\alama\Aftab7>aft
Enter First Side of Triangle (a) : 50
Enter Second Side of Triangle (b) : 50
Enter Third Side of Triangle (c) : 50
Acute Angled Triangle
C:\Users\alama\Aftab7>
```

22/10/2020 **Mohd. Aftab Alam**
02013302717

Slice Based Testing :

There is total 7 variables in the program. We can create slices for each of them.

1. S(a,8)/S(a,42) = {1-8,42}
2. S(b,10) / S(b,42) = {1-6,9,10,42}
3. S(c,12) / S(c,42) = {1-6,11,12,42}
4. S(a1,22) / S(a1,42) = {1-16,20-22,34,42}
5. S(a1,26) = {1-16,20-22,25-27,34,41,42}
6. S(a1,29) = {1-16,20-22,25,27-31,33,34,41,42}
7. S(a1,32) = {1-16,20-22,25,27,28,30-34,41,42}
8. S(a2,23) /S(a2,42) = {1-16,20,21,23,34,42}
9. S(a2,26) = {1-16,20,21,23,25-27,34,41,42}
10. S(a2,29) = {1-16,20,21,23,25,27-31,33,34,41,42}
11. S(a2,32) = {1-16,20,21,23,25,27,28,30-34,41,42}
12. S(a3,24) / S(a3,42) = {1-16,20,21,24,34,42}
13. S(a3,26) = {1-16,20,21,24-27,34,41,42}
14. S(a3,29) = {1-16,20,21,24,25,27-31,33,34,41,42}
15. S(a3,32) = {1-16,20,21,24,25,27,28,30-34,41,42}
16. S(valid,5) = {1-5,42}
17. S(valid,15) = {1-16,20,42}
18. S(valid,18) = {1-14,16-20,42}
19. S(valid,36) = {1-14,16-20,21,34-38,40-42}
20. S(valid,39) = {1-13,20,21,34,35,37-42}
21. S(valid,42) = {1-14,16-20,42}

Test Cases :

S. No.	Slice	Lines Covered	Variables			Expected Output
			a	b	c	
1	S(a,8)/S(a,42)	1-8,42	20			No Output
2	S(b,10)/S(b,42)	1-6,9,10,42		20		No Output
3	S(c,12)/S(c,42)	1-6,11,12,42			20	No Output
4	S(a1,22)/S(a1,42)	1-16,20-22,34,42	30	20	40	No Output
5	S(a1,26)	1-16,20-22,25-37,34,41,42	30	20	40	Obtuse Angled
6	S(a1,29)	1-16,20-22,25,27-31, 33,34,41,42	30	40	50	Right Angled
7	S(a1,32)	1-16,20-22,25,27,30-34,41,42	50	60	40	Acute Angled
8	S(a2,23)/S(a2,42)	1-16,20,21,23,34,42	30	20	40	No Output
9	S(a2,26)	1-16,20,21,23,25-27, 34,41,42	40	30	20	Obtuse Angled
10	S(a2,29)	1-16,20,21,23,25,27-31, 33,34,41,42	50	40	30	Right Angled
11	S(a2,32)	1-16,20,21,23, 25,27,28,30-34,41,42	40	50	60	Acute Angled
12	S(a3,24)/S(a3,42)	1-16,20,21,24,34,42	30	20	40	No Output
13	S(a3,26)	1-16,20,21,24-27,34,41,42	20	40	30	Obtuse Angled
14	S(a3,29)	1-16,20,21,24,25,27-31, 33,34,41,42	40	50	30	Right Angled
15	S(a3,32)	1-16,20,21,24,25,27,28, 30-34,41,42	50	40	60	Acute Angled
16	S(valid,5)	1-5,42				No Output
17	S(valid,15)	1-16,20,42	20	40	30	No Output
18	S(valid,18)	1-14,18-20,42	30	10	15	No Output
19	S(valid,36)	1-14,16-20,21,34-38,40-42	30	10	15	Invalid Triangle
20	S(valid,39)	1-13,20,21,34,35,37-42	102	-1	6	Out of Range
21	S(valid,42)	1-14,16-20,42	30	10	15	No Output

22/10/2020 **Mohd. Aftab Alam**
02013302717

Aim :

(b) Write a Program in C/C++ to compute total salary of an employee when his basic salary is given.

(Given: HRA = 3% of basic, DA = 8% of basic, CCA/MA = Rs. 100, Tax = Rs. 300, PF = Rs.780, TA = Rs. 800). Perform Slice based testing for all variables

Total Salary = (Basic + HRA + DA + TA) – (Tax + CM + PF)

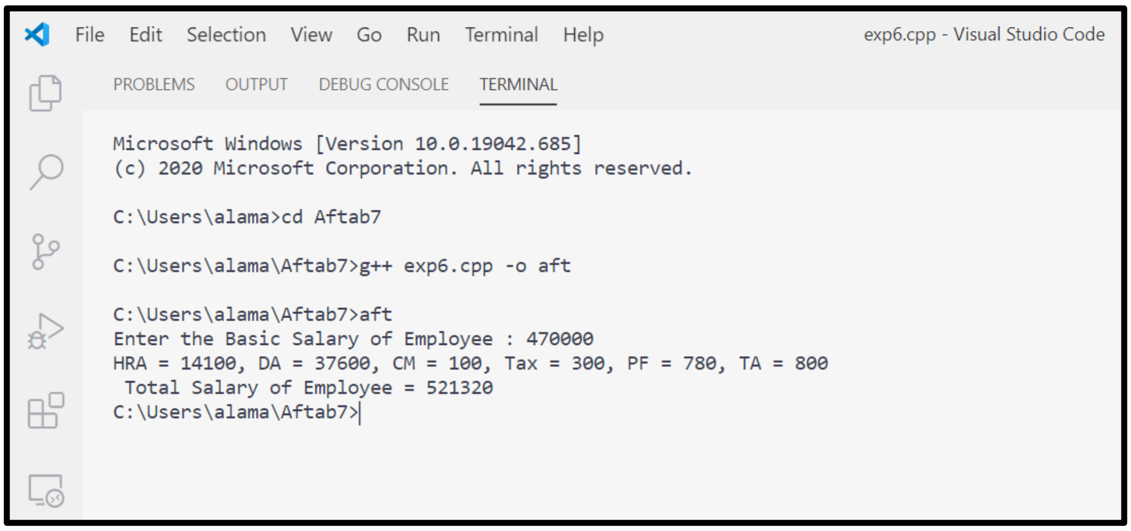
Algorithm :

- Take the Basic Salary of the employee as input from the user.
- Calculate HRA and DA using the basic salary.
- Calculate the Total Salary by combining all the values.
- Print the Total Salary of the employee as calculated on the screen.

Code :

```
#include <iostream>
using namespace std;
1. int main() {
2. float basic, HRA, DA, CM, tax, PF, TA, total_salary;
3. printf("Enter the Basic Salary of Employee : ");
4. cin >> basic;
5. HRA = (basic * 3) / 100;
6. DA = (basic * 8) / 100;
7. CM = 100;
8. tax = 300;
9. PF = 780;
10. TA = 800;
11.cout<<"HRA="<<HRA<<"DA="<<DA<<"CM="<<CM<<"Tax="<<tax<<"PF="<<PF<<"TA="<<TA;
12. total_salary = (basic + HRA + DA + TA) – (tax + CM + PF);
13. cout << "\n Total Salary of Employee = " << total_salary;
14. return 0;
15. }
```

Output Screenshots :


File Edit Selection View Go Run Terminal Help
exp6.cpp - Visual Studio Code
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
Microsoft Windows [Version 10.0.19042.685]
(c) 2020 Microsoft Corporation. All rights reserved.

C:\Users\alama>cd Aftab7

C:\Users\alama\Aftab7>g++ exp6.cpp -o aft

C:\Users\alama\Aftab7>aft
Enter the Basic Salary of Employee : 470000
HRA = 14100, DA = 37600, CM = 100, Tax = 300, PF = 780, TA = 800
 Total Salary of Employee = 521320
C:\Users\alama\Aftab7>
```

22/10/2020 **Mohd. Aftab Alam**
02013302717

Slice Based Testing :

There is total 8 variables in the program. We can create slices for each of them.

- ***Variable: basic***

S(basic,5) / S(basic,15) = {1-5,15}

```
1. int main() {
2. float basic, HRA, DA, CM, tax, PF, TA, total_salary;
3. printf("Enter the Basic Salary of Employee : ");
4. cin >> basic;
15.}
```

- ***Variable: HRA***

S(HRA,6) / S(HRA,15) = {1-6,15}

```
1. int main() {
2. float basic, HRA, DA, CM, tax, PF, TA, total_salary;
3. printf("Enter the Basic Salary of Employee : ");
4. cin >> basic;
5. HRA = (basic * 3) / 100;
15.}
```

- ***Variable: DA***

S(DA,6) / S(DA,15) = {1-6,7,15}

```
1. int main() {
2. float basic, HRA, DA, CM, tax, PF, TA, total_salary;
3. printf("Enter the Basic Salary of Employee : ");
4. cin >> basic;
6. DA = (basic * 8) / 100;
15.}
```

- ***Variable: CM***

S(CM,8) / S(CM,15) = {1-3,8,15}

```
1. int main() {
2. float basic, HRA, DA, CM, tax, PF, TA, total_salary;
7. CM = 100;
15.}
```

- ***Variable: tax***

S(tax,8) / S(tax,15) = {1-3,8,15}

```
1. int main() {
2. float basic, HRA, DA, CM, tax, PF, TA, total_salary;
8. tax = 300;
15.}
```

- ***Variable: PF***

S(PF,10) / S(PF,15) = {1-3,10,15}

```
1. int main() {
2. float basic, HRA, DA, CM, tax, PF, TA, total_salary;
9. PF = 780;
15.}
```

- ***Variable: TA***

S(TA,11) / S(TA,15) = {1-3,11,15}

```
1. int main() {
2. float basic, HRA, DA, CM, tax, PF, TA, total_salary;
10.TA = 800;
15.}
```

- ***Variable: total_salary***

S(Total,12) = {1-12,15}

```
1. int main() {
2. float basic, HRA, DA, CM, tax, PF, TA, total_salary;
3. printf("Enter the Basic Salary of Employee : ");
4. cin >> basic;
5. HRA = (basic * 3) / 100;
6. DA = (basic * 8) / 100;
7. CM = 100;
8. tax = 300;
9. PF = 780;
10.TA = 800;
12.total_salary = (basic + HRA + DA + TA) - (tax + CM + PF);
15.}
```

S(Total,13) / S(Total,15) = {1-13,14,15}

```
1. int main() {
2. float basic, HRA, DA, CM, tax, PF, TA, total_salary;
3. printf(“Enter the Basic Salary of Employee : “);
4. cin >> basic;
5. HRA = (basic * 3) / 100;
6. DA = (basic * 8) / 100;
7. CM = 100;
8. tax = 300;
9. PF = 780;
10. TA = 800;
11.cout<<”HRA=”<<HRA<<”DA=”<<DA<<”CM=”<<CM<<”Tax=”<<tax<<”PF=”<<PF<<”TA=”<<TA;
12. total_salary = (basic + HRA + DA + TA) – (tax + CM + PF);
13. cout << “\n Total Salary of Employee = “ << total_salary;
14. return 0;
15. }
```

22/10/2020 **Mohd. Aftab Alam**
02013302717

Test Cases :

S. No.	Slice	Lines Covered	Variables			Expected Output
			Basic	HRA	DA	
1	S(Basic,5) / S(Basic,15)	1-5,15	1000	30	80	No Output
2	S(HRA,6) / S(HRA,15)	1-6,15	3000	90	240	No Output
3	S(DA,7) / S(DA,15)	1-5,7,15	3000	90	240	No Output
4	S(MA,8) / S(MA,15)	1-3,8,15	3000	90	240	No Output
5	S(ITAX,9) / S(ITAX,15)	1-3,9,15	3000	90	240	No Output
6	S(PF,10) / S(PF,15)	1-3,10,15	3000	90	240	No Output
7	S(TA,11) / S(TA,15)	1-3,11,15	3000	90	240	No Output
8	S(Total,12)	1-12,15	3000	90	240	No Output
9	S(Total,13) / S(Total,15)	1-13,14,15	5000	150	400	7530

Aim :

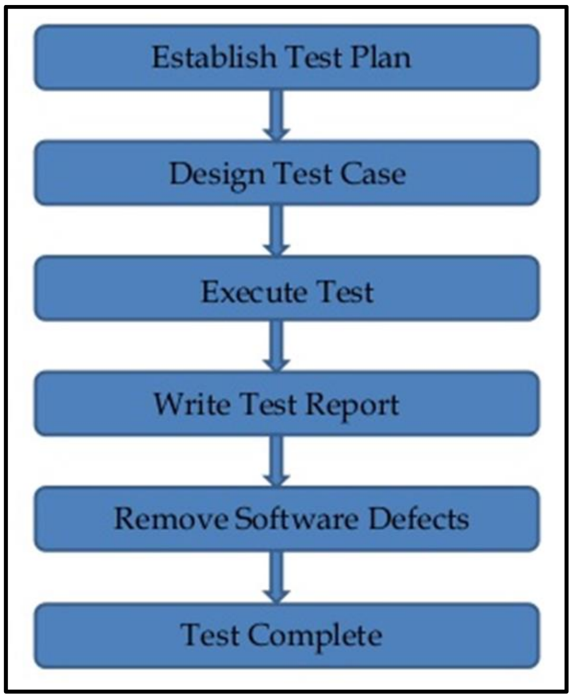
(c) To study a Testing Tool – **WinRunner**

1. Introduction

If you have ever tested software manually, you are aware of its drawbacks. Manual testing is time-consuming and tedious, requiring a heavy investment in human resources. Worst of all, time constraints often make it impossible to manually test every feature thoroughly before the software is released. This leaves you wondering whether serious bugs have gone undetected.

Automated testing with WinRunner addresses these problems by dramatically speeding up the testing process. You can create test scripts that check all aspects of your application, and then run these tests on each new build. As WinRunner runs tests, it simulates a human user by moving the mouse cursor over the application, clicking Graphical User Interface (GUI) objects, and entering keyboard input-but WinRunner does this faster than any human user.

2. Process Flow for WinRunner



3. Features

- Functional Regression Testing Tool
- Windows Platform Dependent
- Only for Graphical User Interface (GUI) based Application
- Based on Object Oriented Technology (OOT) concept
- Only for Static content
- Developed by Mercury Interactive
- Functionality testing tool
- WinRunner run on Windows only
- XRunner run only in UNIX and Linux
- Tool developed in C on VC++ environment
- To automate our manual test Win runner used TSL (Test Script language like C)
- Record/Playback Tool

4. Add-ins

WinRunner includes the following Add-ins:

- Web Test
- Visual Basic
- ActiveX
- Power Builder

5. Running the Test

WinRunner provides three modes for running test:

- Use Verify mode when running a test to check the behaviour of our application and when we want to save the test result.
- Use Debug Mode, when you want to check that the test script runs smoothly without errors in syntax. The debug mode will not give the test result.
- Use Update mode, when you want to create new expected results for a GUI check point or bitmap check point.

6. WinRunner Testing Process

- Create GUI Map File: By creating GUI Map file the WinRunner can identify the GUI objects in the application going to be tested.
- Create Test Scripts: This process involves recording, programming or both. During the process of recording tests, insert checkpoints where the response of the application needs to be tested.
- Debug Test: Run the tests in Debug mode to make sure whether they run smoothly.
- Run Tests: Run tests in Verify mode to test the application.
- View Results: This determines the success or failure of the tests.
- Report Defects: If a particular test run fails due to the defect in the application being tested, defects can be directly reported through the Test Results window.

7. Facts about WinRunner

- WinRunner doesn’t support web application.
- WinRunner only supports IE and Netscape Navigator.
- The default time setting of WinRunner is 10000ms.
- The logical name of true and false in WinRunner is 1 and 0.
- WinRunner doesn’t provide the snapshot of output.
- WinRunner has 3 kinds of checkpoints, 2 kinds of Recordings and 3 kinds of Exception Handling mechanisms.
- In WinRunner GUI, Spy is available.
- WinRunner uses TSL script (Test Script Language).
- WinRunner only works on 32-bit machine.