Software Engineering

Assignment 5

Name : ABID ALI

Roll : 2019380141

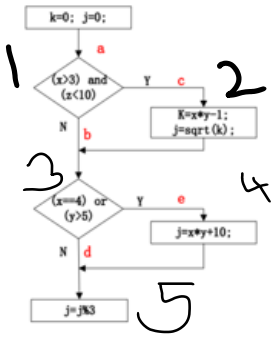
Send to : lining@nwpu.edu.cn , 480755534@qq.com

Assignment 5

Deadline: April 17

Q1: White-box testing

```
1  void DoWork (int x, int y, int z)
2  {
3      int k=0,j=0;
4
5      if((x>3) && (z<10))
6      {
7          k=x*y-1;
8          j=sqrt(k);      //block 1
9      }
10
11     if((x==4) || (y>5))
12     {
13         j=x*y+10;       //block 2
14     }
15
16     j=j%3;              //block 3
17 }
```



Design testcases with the following techniques:

1) statement coverage
2) branch coverage
3) decision coverage
4) decision/branch coverage
5) decision combination coverage
6) basic path coverage

program: input: x, y, z ; output: k,j

Design six test cases tables for each coverage technique.

78

Program:

Input: x = 6, y = 4, z = 9 (ace)

Output:

Block 1:

k = 23 , j = 4

Block 2:

j = 34

Block 3:

j = 1

Input: x = 3, y = 5, z = 9 (abd)

Output:

Block 1:

Condition not fulfilled. So, the block 1 is skipped.

Block 2:

j = 37

Block 3:

j = 1

(1) statement Coverage:

In this the source code is executed at least once. It's a white box testing technique. that is used for the calculation of the no q statements in the source code that have been. executed. The main purpose of this is to cover all the possible paths, lines & statements in code.

Statement coverage: every segment is executed

1 and 3 is conditional statements.

2,4 and 5 assignment statements.

Input: x = 4 z = 9 y = 6 (ace)

Output: k = 23 , j = 1

It has covered 3 stages out of 6. It has coverage of 60%.

(2) Branch Coverage:

It is used to ensure that each decision condition from every at least once. branch is executed

It helps to measure fractions of independent code segments & to find out sections that have no branches.

It is a white box testing method in which every from a code is tested.

1 and 3 conditional statements.

Cover all branch(Y/N):

acd,abc

Input : x = 6, x = 6,z = 8 (acd)

Output : k = 35, j = 1

Input : x = 2, x = 6,z = 11 (acd)

Output : k = 0,j = 1

(3) Decision Coverage:

It is used to covert and validate all the accesside sourive lope by checking & confirming that each branch of every possible decision point is executed at least once. It is a white box testing technique that reports the true or false outcomes. Of each boolean expression of these code.

Decision coverage : Every decision is executed :

x> 3, z < 10, x == 4, y > 5

False => Input : x = 2 , x =4 , z =11 (abd)

Output : k = 0 , j = 0

True => Input : x = 5 z = 7 y = 8 (ace)

Output : k = 34 , i = 0 (abd)

4) Decision/ branch :

x> 3, z < 10, x == 4, y > 5

False => Input : x = 2 , x =4 , z =11 (abd)

Output : k = 0 , j = 0

True => Input : x = 5 z = 7 y = 8 (ace)

Output : k = 34 , i = 0 (abd)

5) Decision combination

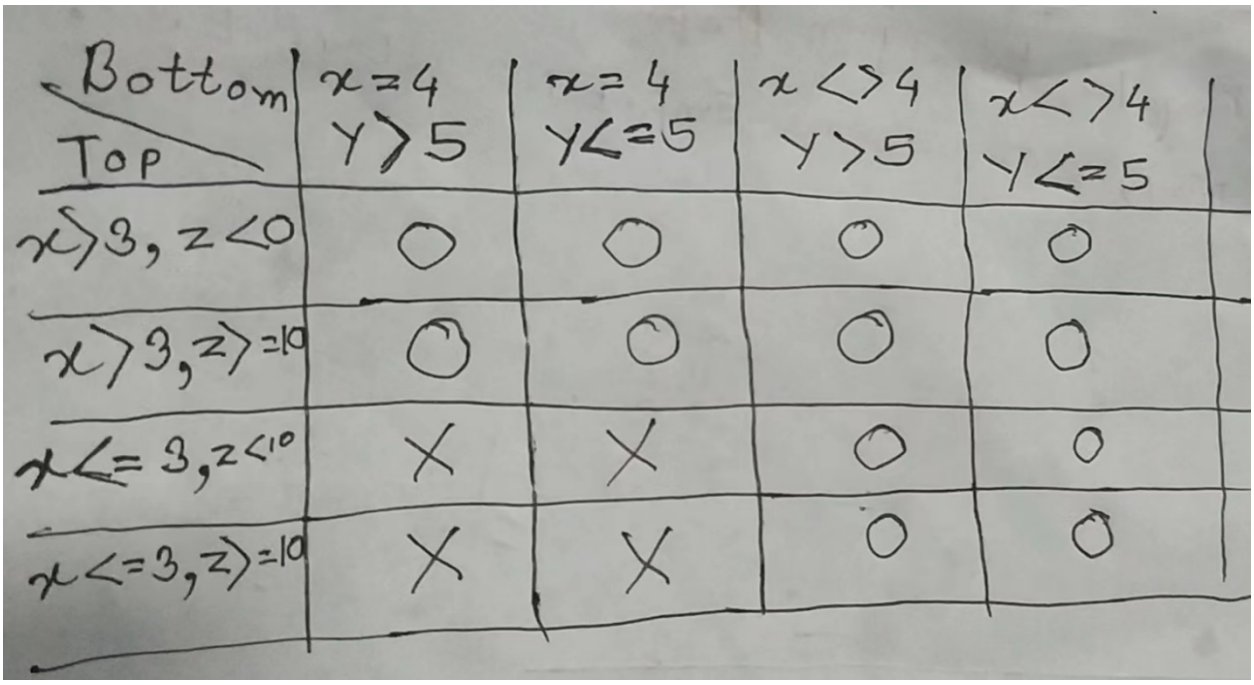
Top :

1) T and T x > 3, z < 10
2) T and F x > 3, z > = 10

3) F and T x <= 3, z < 10
4) F and F x <= 3, z >= 10

Bottom :

1) T and T x = 4, y > 5
2) T and F x = 4, y < = 5
3) F and T x <> 4, y > 5
4) F and F x <> 4, y <= 5



(6)Basic Path Coverage:

TT : ace

TF : acd

FT : ade

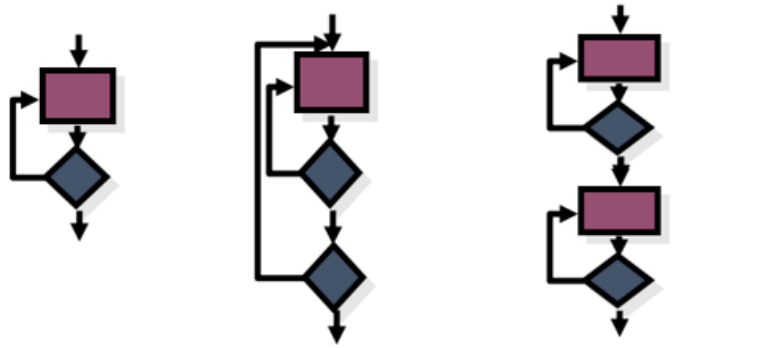
FF : abd

Path	x	y	z	k	J
ace	4	6	9	23	1
acd	4	5	9	19	0
abe	5	6	10	0	1
abd	5	5	10	0	0

Assignment 5

Q2: White-box testing

Please write three pieces of code (simple loop, nested loops, concatenated loops), then design test case for them.



simple loop nested loops concatenated loops

Code has been completed in python language

Simple loop:

CASE 1) When value1 = 1 and loop is passed.

value1 = 1

while(value1 == 1):

print("while loop has passed")

exit(0)

print("while loop has failed")

```
value1 = 1
while(value1 == 1):
    print("while loop has passed")
    exit(0)

print("while loop has failed")
```

```
while loop has passed
repl process died unexpectedly:
>
```

CASE 2) When value1= anything except 1 then loop should be failed.

```
value1 = 2
while(value1 == 1):
    print("while loop has passed")
    exit(0)

print("while loop has failed")
```

```
while loop has failed
>
```

Nested Loop:

Case 1) When both while loop conditions are met.

```
value1 = 2

value2 = 2

while(value1 !=1):

   while(value2 == 2):

      print("first WHILE and second WHILE passed")

      exit(0)


   print("first WHILE passed")

   exit(0)

print("both values didnt pass through WHILE loops")
```

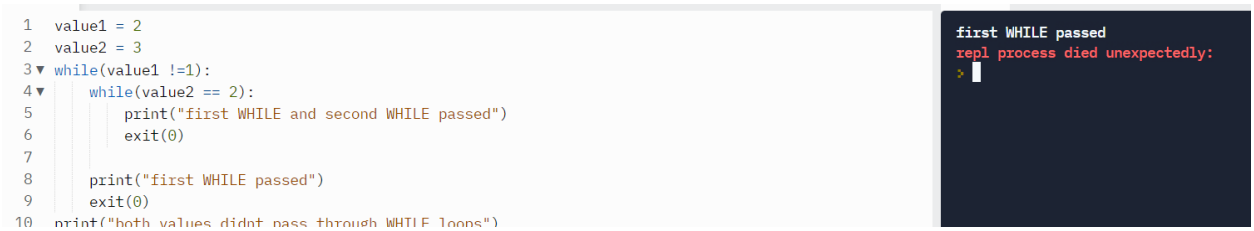
```
value1 = 2
value2 = 2
while(value1 !=1):
    while(value2 == 2):
        print("first WHILE and second WHILE passed")
        exit(0)

    print("first WHILE passed")
    exit(0)
print("both values didnt pass through WHILE loops")
```

```
first WHILE and second WHILE passed
repl process died unexpectedly:
>
```

CASE 2) When the first WHILE is passed but the second WHILE Loop condition fails.

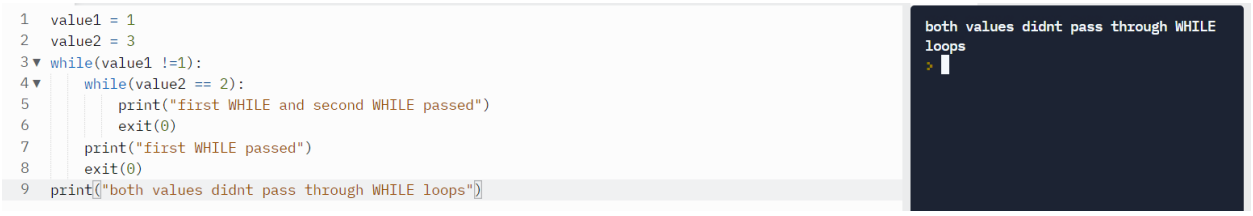
```
value1 = 2

value2 = 3

while(value1 !=1):

   while(value2 == 2):

      print("first WHILE and second WHILE passed")

      exit(0)
```

```
    print("first WHILE passed")
    exit(0)
print("both values didnt pass through WHILE loops")
```



CASE 3) When both WHILE loops aren't passed

```
value1 = 1
value2 = 3
while(value1 !=1):
    while(value2 == 2):
        print("first WHILE and second WHILE passed")
        exit(0)
    print("first WHILE passed")
    exit(0)
print("both values didnt pass through WHILE loops")
```



Concatenated Loops:

CASE 1) When all conditions are met

```
value1 = 1
value2 = 2
while(value1 == 1):
    print("first while loop passed")
    while(value2 == 2):
```

```
        print("second while loop also passed")
        exit(0)
```

```
value1 = 1
value2 = 2
while(value1 == 1):
    print("first while loop passed")
    while(value2 == 2):
        print("second while loop also passed")
        exit(0)
```

```
both values didnt pass through WHILE
loops
```

CASE 2) When the first condition is not met

```
value1 = 2
value2 = 2
while(value1 == 1):
    print("first while loop passed")
    while(value2 == 2):
        print("second while loop also passed")
        exit(0)
    print("Only first while loop passed")
    exit(0)
print("all while loops failed")
```

```
value1 = 2
value2 = 2
while(value1 == 1):
    print("first while loop passed")
    while(value2 == 2):
        print("second while loop also passed")
        exit(0)
    print("Only first while loop passed")
    exit(0)
print("all while loops failed")
```

```
all while loops failed
```

CASE 3) When only the first condition is met

```
value1 = 1
value2 = 3
while(value1 == 1):
    print("first while loop passed")
    while(value2 == 2):
        print("second while loop also passed")
        exit(0)
```

```
    print("Only first while loop passed")
    exit(0)
print("all while loops failed")
```

```
value1 = 1
value2 = 3
while(value1 == 1):
    print("first while loop passed")
    while(value2 == 2):
        print("second while loop also passed")
        exit(0)
    print("Only first while loop passed")
    exit(0)
print("all while loops failed")
```

```
first while loop passed
Only first while loop passed
repl process died unexpectedly:
>
```

CASE 4) When both conditions are not met

```
value1 = 3
value2 = 3
while(value1 == 1):
    print("first while loop passed")
    while(value2 == 2):
        print("second while loop also passed")
        exit(0)
    print("Only first while loop passed")
    exit(0)
print("all while loops failed")
```

```
value1 = 3
value2 = 3
while(value1 == 1):
    print("first while loop passed")
    while(value2 == 2):
        print("second while loop also passed")
        exit(0)
    print("Only first while loop passed")
    exit(0)
print("all while loops failed")
```

```
all while loops failed
>
```

Assignment 5

Q3: Black-box testing

A program accepts as input three integers which it interprets as the lengths of sides of a triangle. It reports whether the triangle is equilateral, isosceles, or scalene (neither equilateral nor isosceles).

Design test cases with the following techniques:

1) Equivalence Class Partitioning :

List valid and invalid equivalence classes you designed

2) Boundary Value Analysis

List all boundary conditions what you can consider

/* Black-box testing

A program accepts as input three integers which it interprets as the lengths of sides of a triangle.

It reports whether the triangle is equilateral, isosceles, or scalene (neither equilateral nor isosceles).

Design test cases with the following techniques:

1) Equivalence Class Partitioning:

List valid and invalid equivalence classes you designed.

2) Boundary Value Analysis

List all boundary conditions what you can consider.

Solution:

The code has been coded in C++ language

```
#include<iostream>
using namespace std;
main()
{
   int a,b,c,choice;
   cout<<"Enter First Side";
   cin>>a;
   cout<<"Enter Second Side";
   cin>>b;
   cout<<"Enter third Side";
   cin>>c;
   cout<<"\n1. Equivalence Class Partitioning:";
   cout<<"\n2.  Boundary Value Analysis";
   cin>>choice;
   switch(choice)
   {
/*Equivalence Class Partitioning
```

valid case 1 = f(a,b, c) a > 0 and b > 0 and c > 0. for scalene Triangle

valid case 2=f(a,b,c) a=b and a=c and b=c for Equilateral triangle

valid case 3=f(a,b,c) a=b or a=c or b=c for isosceles Triangle

For the invalid classes, we need to consider the case where each of the three variables in turn can be negative and so we have the following equivalence classes:

Invalid case 1 = f(a, b, c) a < 0 and b > 0 and c > 0

Invalid case 2 = f(a, b, c) a > 0 and b <0 and c > 0

Invalid case 3 = f(a, b, c) a > 0 and b > 0 and c < 0

Equivalence class Test Inputs Expected Outputs

scalene f(3, 5, 7) ====== “Scalene”

isosceles f(2, 3, 3)===== “Isosceles”

equilateral f(7, 7, 7) ======“Equilateral”

invalid1 f(-1, 2, 3)==== “Error Value”

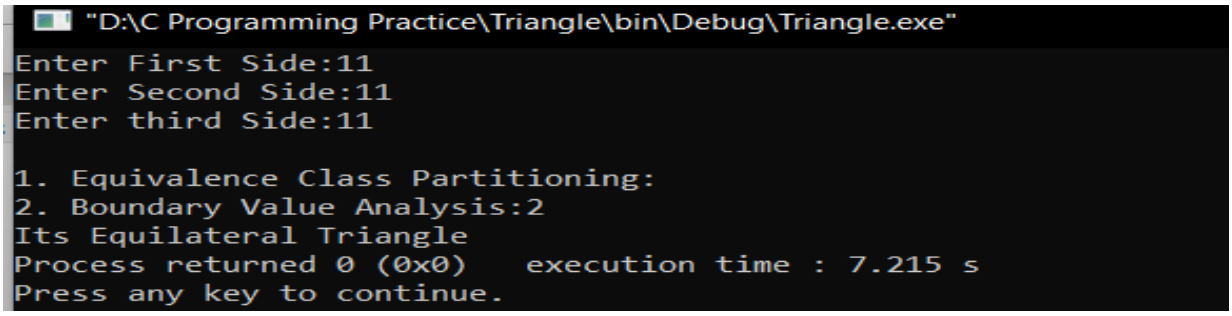
inavlid2 f(1, -2, 3)==== “Error Value”

inavlid3 f(1, 2, -3)==== “Error Value”*/

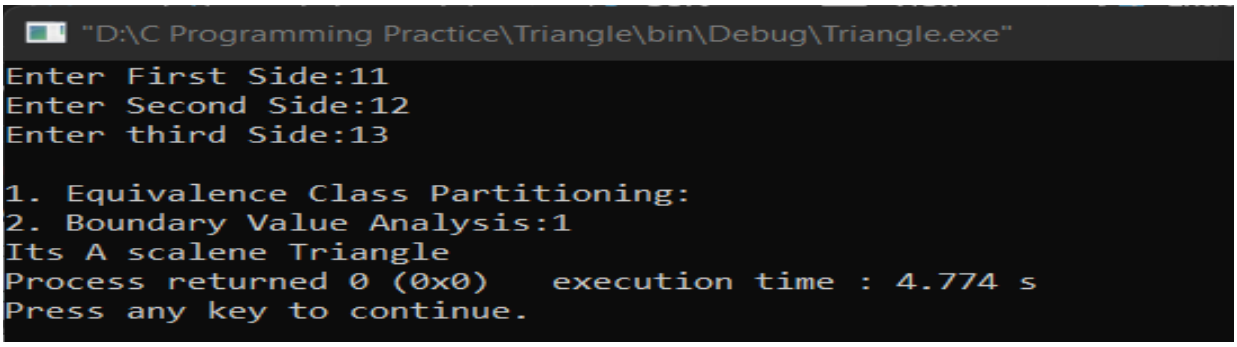
```
        case 1:
           if(a<0 || b< 0 || c<0)
              cout<<"invalid ";
           else
           if (a < 0  && b > 0 && c > 0)
                 cout<<"Wrong input Invalid case 1";
              else
              if (a > 0  && b < 0 && c > 0)
                 cout<<"Wrong input Invalid case 2";
              else
              if (a > 0  && b > 0 && c < 0)
                 cout<<"Wrong input Invalid case 3";
                 else
           if(a>0 && b>0 && c>0 && a!=b && a!=c && b!=c)
              cout<<"Its A scalene Triangle";

           else if(a==b && b==c)
           cout<<"Its Equilateral Triangle";
           else if(a==b ||a==c||b==c)
           cout<<"Its Isosceles Triangle";
           else
```

```
            cout<<"Not A Triangle";
   break;
   case 2:
/*Boundary Value Analysis
Following possible boundary conditions are formed:
1. Given sides (A; B; C) for a scalene triangle, the sum of any two sides is greater than the third and so, we have boundary conditions A + B > C, B + C > A and A + C > B.
2. Given sides (A; B; C) for an isosceles triangle two sides must be equal and so we have boundary conditions A = B, B = C or A = C.
3. Continuing in the same way for an equilateral triangle the sides must all be of equal length and we have only one boundary where A = B = C
*/
      if(a+b>c && b+c > a && a+c>b && a!=b && b!=c)
         cout<<"It Is Scalene Triangle";
             else if(a==b && b==c)
         cout<<"Its Equilateral Triangle";
         else if(a==b ||a==c||b==c)
         cout<<"Its Isosceles Triangle";
         else    cout<<"Not A Triangle";
      break;
   default :
      cout<<"wrong choice";
   }
}
```

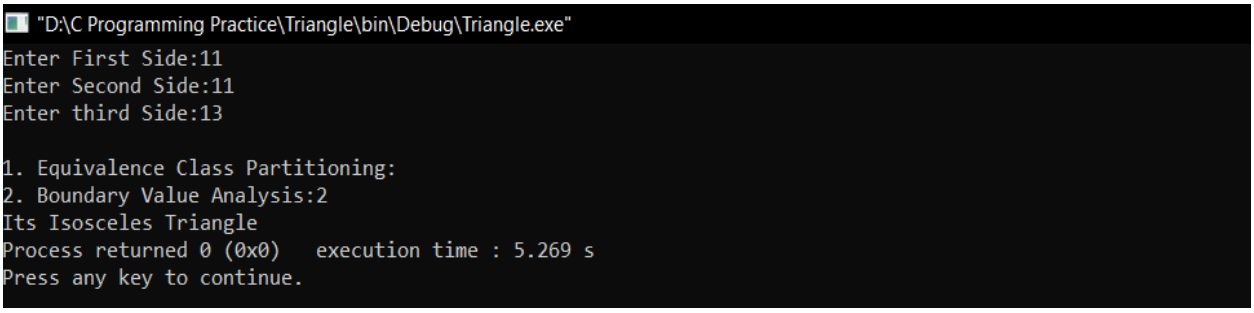
Equilateral Triangle:



Scalene Triangle:



Isosceles Triangle:



Not Triangle:

The code for this program(Question-3) has been provided in a zip file.