

IT 314 Software engineering

LAB 7



Name: Vivek Godhasara

Id: 202001451

Date: 18/4/2023

Lab: 7

Section A

Consider a program for determining the previous date. Its input is triple of day, month and year with the following ranges $1 \leq \text{month} \leq 12$, $1 \leq \text{day} \leq 31$, $1900 \leq \text{year} \leq 2015$. The possible output dates would be previous date or invalid date. Design the equivalence class test cases?

- The equivalence class test cases.

E1: month < 1, E2: $1 \leq \text{month} \leq 12$, E3: month > 12 (day and year ignored)

E4: day < 1, E5: $1 \leq \text{day} \leq 31$, E6: day > 31 (month and year ignored)

E7: year < 1900, E8: $1900 \leq \text{year} \leq 2015$, E9: year > 2015 (day and month ignored)

When we take each month, day and year one by one then there will be 3 equivalent classes. Now we will take them in combination so equivalent classes will be:

$$3 * 3 * 3 = 27$$

E1: month < 1 , day < 1 , year < 1900

E2: $1 \leq \text{month} \leq 12$, day < 1 , year < 1900

E3: month > 12 , day < 1 , year < 1900

E4: month < 1 , $1 \leq \text{day} \leq 31$, year < 1900

E5: $1 \leq \text{month} \leq 12$, $1 \leq \text{day} \leq 31$, year < 1900

E6: month > 12 , $1 \leq \text{day} \leq 31$, year < 1900

E7: month < 1 , day > 31 , year < 1900

E8: $1 \leq \text{month} \leq 12$, day > 31 , year < 1900

E9: month > 12 , day > 31 , year < 1900

E10: month < 1 , day < 1 , $1900 \leq \text{year} \leq 2015$

E11: $1 \leq \text{month} \leq 12$, day < 1 , $1900 \leq \text{year} \leq 2015$

E12: month > 12 , day < 1 , $1900 \leq \text{year} \leq 2015$

E13: month < 1 , $1 \leq \text{day} \leq 31$, $1900 \leq \text{year} \leq 2015$

E14: $1 \leq \text{month} \leq 12$, $1 \leq \text{day} \leq 31$, $1900 \leq \text{year} \leq 2015$

E15: month > 12 , $1 \leq \text{day} \leq 31$, $1900 \leq \text{year} \leq 2015$

E16: month < 1 , day > 31 , $1900 \leq \text{year} \leq 2015$

E17: $1 \leq \text{month} \leq 12$, day > 31 , $1900 \leq \text{year} \leq 2015$

E18: month > 12 , day > 31 , $1900 \leq \text{year} \leq 2015$

E19: month < 1 , day < 1 , year > 2015

E20: 1 <= month <= 12 , day < 1 , year > 2015

E21: month > 12 , day < 1 , year > 2015

E22: month < 1 , 1 <= day <= 31 , year > 2015

E23: 1 <= month <= 12 , 1 <= day <= 31 , year > 2015

E24: month > 12 , 1 <= day <= 31 , year > 2015

E25: month < 1 , day > 31 , year > 2015

E26: 1 <= month <= 12 , day > 31 , year > 2015

E27: month > 12 , day > 31 , year > 2015

- **Write a set of test cases (i.e., test suite) – specific set of data – to properly test the programs. Your test suite should include both correct and incorrect inputs.**

1. Enlist which set of test cases have been identified using Equivalence Partitioning and Boundary Value Analysis separately.

2. Modify your programs such that it runs on eclipse IDE, and then execute your test suites on the program. While executing your input data in a program, check whether the identified expected outcome (mentioned by you) is correct or not.

Programs:

- 1. The function linearSearch searches for a value v in an array of integers a. If v appears in the array a, then the function returns the first index i, such that a[i] == v; otherwise, -1 is returned.**

```
int linearSearch(int v, int a[])
{
    int i = 0;
    while (i < a.length)
    {
        if (a[i] == v)
            return(i);
        i++;
    }
    return (-1);
}
```

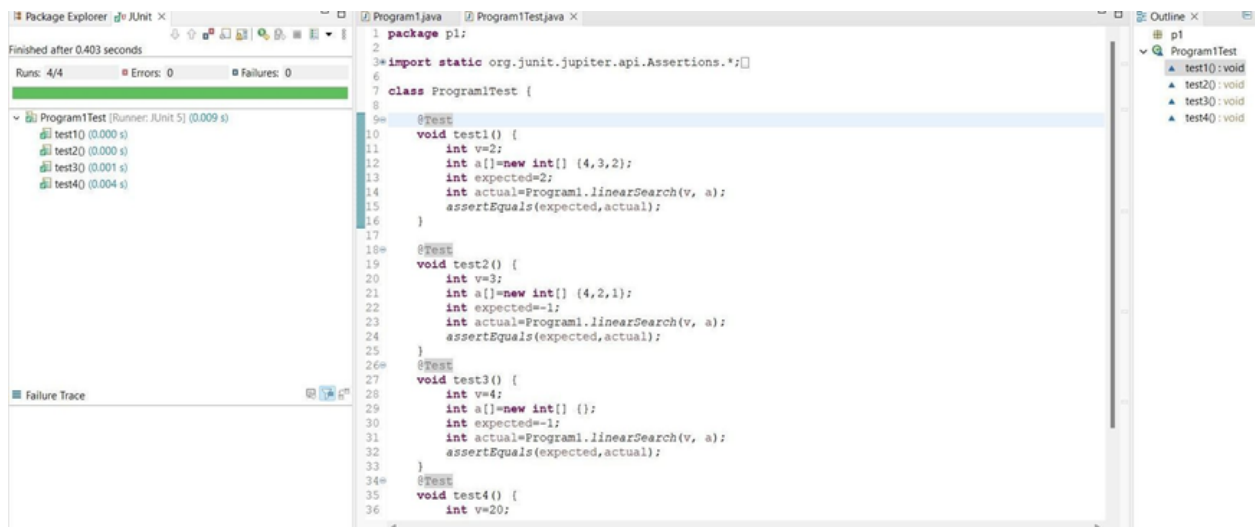
ANSWER:

Tester Action and Input Data	Expected Outcome
v = 3, a = {1, 2, 3, 4, 5}	2
v = 10, a = {1, 2, 3, 4, 5}	-1
v = 5, a = {5}	0
v = 7, a = {}	-1
v = -2, a = {1, 0, -2, 3, 5}	2

Equivalence Partitioning	Expected Outcome
v = 3, a = {1, 2, 3, 4, 5}	2
v = -2, a = {1, 0, -2, 3, 5}	2
v = 5, a = {5}	0

Boundary Value Analysis	Expected Outcome
v = 7, a = {}	-1
v = 1, a = {1, 2, 3, 4, 5}	0
v = 6, a = {1, 5}	-1

Junit Testing:



- The function `countItem` returns the number of times a value `v` appears in an array of integers `a`.

```
int countItem(int v, int a[])
{
    int count = 0;
    for (int i = 0; i < a.length; i++)
    {
        if (a[i] == v)
            count++;
    }
    return (count);
}
```

ANSWER:

Tester Action and Input Data	Expected Outcome
<code>v = 5, a = []</code>	0
<code>v = 2, a = [2]</code>	1
<code>v = 3, a = [2]</code>	0
<code>v = 4, a = [4, 2, 4, 6, 4, 8, 4]</code>	4

v = 7, a = [2, 4, 6, 8, 7, 9]	1
v = 5, a = [1, 2, 3, 4, 6, 7, 8, 9]	0

Equivalence Partitioning	Expected Outcome
v = 5, a = [1, 2, 3, 4, 6, 7, 8, 9]	0
v = 7, a = [2, 4, 6, 8, 7, 9]	1
v = 4, a = [4, 2, 4, 6, 4, 8, 4]	4

Boundary Value Analysis	Expected Outcome
v = 7, a = {}	-1
v = 1, a = {1, 2, 3, 4, 5}	0
v = 6, a = {1, 5}	-1

JUnit Testing:

The screenshot shows an IDE with the following components:

- Package Explorer:** Shows a project named 'p2' with a sub-package 'Program2Test'. It lists four test methods: 'test10: void', 'test20: void', 'test30: void', and 'test40: void'.
- JUnit Runner:** Displays the results of a test run. It shows 'Finished after 0.15 seconds', 'Runs: 4/4', 'Errors: 0', and 'Failures: 0'. A list of tests is shown: 'test10 (0.000 s)', 'test20 (0.000 s)', 'test30 (0.000 s)', and 'test40 (0.000 s)', all with green status icons.
- Java Editor:** Shows the source code for 'Program2Test.java'. The code includes:


```

12  int v=2;
13  int a[]=new int[] {4,2,3,2,1};
14  int expected=2;
15  int actual=Program2.countItem(v, a);
16  assertEquals(expected,actual);
17  }
18  @Test
19  void test2() {
20      int v=3;
21      int a[]=new int[] {4,2,3};
22      int expected=1;
23      int actual=Program2.countItem(v, a);
24      assertEquals(expected,actual);
25  }
26  @Test
27  void test3() {
28      int v=20;
29      int a[]=new int[] {1,2,3};
30      int expected=0;
31      int actual=Program2.countItem(v, a);
32      assertEquals(expected,actual);
33  }
34  @Test
35  void test4() {
36      int v=1;
37      int a[]=new int[] {};
38      int expected=0;
39      int actual=Program2.countItem(v, a);
40      assertEquals(expected,actual);
41  }
42  }
43  }
44  }
45  }
```

3. The function `binarySearch` searches for a value `v` in an ordered array of integers `a`. If `v` appears in the array `a`, then the function returns an index `i`, such that `a[i] == v`; otherwise, `-1` is returned.

Assumption: the elements in the array `a` are sorted in non-decreasing order.

```
int binarySearch(int v, int a[])
{
    int lo, mid, hi;
    lo = 0;
    hi = a.length-1;
    while (lo <= hi)
    {
        mid = (lo+hi)/2;
        if (v == a[mid])
            return (mid);
        else if (v < a[mid])
            hi = mid-1;
        else
            lo = mid+1;
    }
    return(-1);
}
```

ANSWER:

Tester Action and Input Data	Expected Outcome
<code>v = 2, a = {1, 2, 3, 4, 5}</code>	1
<code>v = -4, a = {1, 2, 3, 4, 5}</code>	-1
<code>v = 5, a = {5}</code>	0
<code>v = -2, a = {1, 0, -2, 3, 5}</code>	2

Equivalence Partitioning	Expected Outcome
<code>v = 3, a = {1, 2, 3, 4, 5}</code>	2
<code>v = -2, a = {1, 0, -2, 3, 5}</code>	2

v = 5, a = {5}	0
----------------	---

Boundary Value Analysis	Expected Outcome
v = -4, a = {1, 2, 3, 4, 5}	-1
v = 6, a = {1, 5}	-1

JUnit Testing:

The screenshot shows an IDE with the following components:

- Package Explorer:** Shows a project named 'p3' with a sub-package 'Program3Test' containing three test classes: 'test10', 'test20', and 'test30'.
- JUnit Runner:** Displays the results of a test run. It shows 'Finished after 0.12 seconds', 'Runs: 3/3', 'Errors: 0', and 'Failures: 0'. The tests 'test10', 'test20', and 'test30' all passed with execution times of 0.022s, 0.002s, and 0.003s respectively.
- Source Code:** The main editor shows the source code for 'Program3Test.java'. It contains three test methods:
 - `test1()`: Tests with `v=2` and array `{0,1,2,3,4}`. Expected result is 2.
 - `test2()`: Tests with `v=-4` and array `{1,2,3,4,5}`. Expected result is -1.
 - `test3()`: Tests with `v=5` and array `{2,3,4,5,6}`. Expected result is 3. It includes a comment about two possible correct outputs and uses a try-catch block for `assertEquals`.
- Failure Trace:** The bottom panel is empty, indicating no test failures.

- The following problem has been adapted from The Art of Software Testing, by G. Myers (1979). The function triangle takes three integer parameters that are interpreted as the lengths of the sides of a triangle. It returns whether the triangle is equilateral (three lengths equal), isosceles (two lengths equal), scalene (no lengths equal), or invalid (impossible lengths).

```
final int EQUILATERAL = 0;
final int ISOSCELES = 1;
final int SCALENE = 2;
final int INVALID = 3;
int triangle(int a, int b, int c)
{
    if (a >= b+c || b >= a+c || c >= a+b)
```



```

        return(INVALID);
    if (a == b && b == c)
        return(EQUILATERAL);
    if (a == b || a == c || b == c)
        return(ISOSCELES);
    return(SCALENE);
}

```

ANSWER:

Tester Action and Input Data	Expected Outcome
a=4,b=4,c=4	EQUILATERAL
a=1,b=2,c=3	INVALID
a=-1,b=2,c=3	INVALID
a=3,b=4,c=5	SCALENE
a=5,b=5,c=9	ISOSCELES
a=5, b=5, c=10	INVALID

Equivalence Partitioning	Expected Outcome
a=4,b=4,c=4	EQUILATERAL
a=5,b=5,c=9	ISOSCELES
a=5, b=5, c=10	INVALID
a=3,b=4,c=5	SCALENE

Boundary Value Analysis	Expected Outcome
a=-1,b=2,c=3	INVALID
a=1,b=2,c=3	INVALID

JUnit Testing:



- The function prefix (String s1, String s2) returns whether or not the string s1 is a prefix of string s2 (you may assume that neither s1 nor s2 is null).

```
public static boolean prefix(String s1, String s2)
{
    if (s1.length() > s2.length())
    {
        return false;
    }
    for (int i = 0; i < s1.length(); i++)
    {
        if (s1.charAt(i) != s2.charAt(i))
        {
            return false;
        }
    }
    return true;
}
```

ANSWER:

Tester Action and Input Data	Expected Outcome
s1="soft", s2="software"	True
s1="abd", s2="abc"	False

s1="health" , s2="health"	True
s1="one" , s2="two"	False
s1="",s2="sdf"	True

Equivalence Partitioning	Expected Outcome
s1="soft" , s2="software"	True
s1="abd" , s2="abc"	False
s1="health" , s2="health"	True
s1="one" , s2="two"	True

Boundary Value Analysis	Expected Outcome
s1="",s2="sdf"	True

Junit Testing:

The screenshot displays an IDE with the following components:

- Package Explorer:** Shows the project structure with 'Program5Test' expanded, listing four test methods: test1() (0.021 s), test2() (0.000 s), test3() (0.001 s), and test4() (0.001 s).
- JUnit Console:** Displays the message 'Finished after 0.114 seconds' and 'Runs: 4/4', 'Errors: 0', 'Failures: 0'.
- Source Code (Program5Test.java):**

```

6
7 class Program5Test {
8
9     @Test
10    void test1() {
11        String s1="soft",s2="software";
12        boolean output=Program5.prefix(s1, s2);
13        boolean expected=true;
14        assertEquals(expected,output);
15    }
16
17    @Test
18    void test2() {
19        String s1="abd",s2="abc";
20        boolean output=Program5.prefix(s1, s2);
21        boolean expected=false;
22        assertEquals(expected,output);
23    }
24
25    @Test
26    void test3() {
27        String s1="health",s2="health";
28        boolean output=Program5.prefix(s1, s2);
29        boolean expected=true;
30        assertEquals(expected,output);
31    }
32
33    @Test
34    void test4() {
35        String s1="one",s2="two";
36        boolean output=Program5.prefix(s1, s2);
37        boolean expected=false;
38        assertEquals(expected,output);
39    }
40
41 }
```
- Outline:** Lists the test methods: test1(): void, test2(): void, test3(): void, test4(): void.

6. Consider again the triangle classification program (P4) with a slightly different specification: The program reads floating values from the standard input. The three values A, B, and C are interpreted as representing the lengths of the sides of a triangle. The program then prints a message to the standard output that states whether the triangle, if it can be formed, is scalene, isosceles, equilateral, or right angled.

Determine the following for the above program:

a) Identify the equivalence classes for the system

Invalid case:

E1: $a+b \leq c$

E2: $a+c \leq b$

E3: $b+c \leq a$

Equilateral case:

E4: $a=b, b=c, c=a$

Isosceles case:

E5: $a=b, a \neq c$

E6: $a=c, a \neq b$

E7: $b=c, b \neq a$

Scalene case:

E8: $a \neq b, b \neq c, c \neq a$

Right-angled triangle case:

E9: $a^2 + b^2 = c^2$

E10: $b^2 + c^2 = a^2$

E11: $a^2 + c^2 = b^2$

b) Identify test cases to cover the identified equivalence classes. Also, explicitly mention which test case would cover which equivalence class.

(Hint: you must need to be ensure that the identified set of test cases cover all identified equivalence classes)

Test case	Output	Equivalent class covered
$a=1.5, b=2.6, c=4.1$	Not a triangle	E1

a = -1.6, b=5, c=6	Not a triangle	E2
a=7.1, b=6.1, c=1	Not a triangle	E3
a=5.5, b= 5.5, c=5.5	Equilateral	E4
a=4.5, b=4.5, c=5	isosceles	E5
a=6, b=4, c=6	isosceles	E6
a=8, b=5, c=5	isosceles	E7
a=6,b=7,c=8	scalene	E8
a=3,b=4,c=5	Right-angled triangle	E9
a=0.13,b=0.12,c=0.05	Right-angled triangle	E10
a=7,b=25,c=23	Right-angled triangle	E11

c) For the boundary condition $A + B > C$ case (scalene triangle), identify test cases to verify the boundary.

- Test cases to verify the boundary condition:
 - 1) a=5 b=5 c=9 (a+b=c)
 - 2) a=5.5 b=5.5 c=10.9 (a+b just greater than c)
 - 3) a=5.5 b=5 c=9.6 (a+b just less than c)

d) For the boundary condition $A = C$ case (isosceles triangle), identify test cases to verify the boundary.

- Test cases to verify the boundary condition:
 - 1) a=5 b=5 c=5 (a=c)
 - 2) a=5.5 b=5.5 c=5.6 (a just less than c)
 - 3) a=5.5 b=5 c=5.4 (a just greater than c)

e) For the boundary condition $A = B = C$ case (equilateral triangle), identify test cases to verify the boundary.

- Test cases to verify the boundary condition:
 - 1) a=5 b=5 c=5 (a=b=c)
 - 2) a=10 b=10 c=9 (a= b but a≠c)
 - 3) a=10 b=11 c=10 (a=c but a≠b)

f) For the boundary condition $A^2 + B^2 = C^2$ case (right-angle triangle), identify test cases to verify the boundary.

- Test cases to verify the boundary condition:
 - 1) $a=3, b=4, c=5$ ($a^2+b^2=c^2$)
 - 2) $a=0.12, b=0.5, c=0.14$ (a^2+b^2 just less than c^2)
 - 3) $a=7, b=23, c=24$ (a^2+b^2 just greater than c^2)

g) For the non-triangle case, identify test cases to explore the boundary.

- Test cases to verify the boundary condition:
 - 1) $a=1, b=2, c=3$
 - 2) $a=5, b=5, c=10$
 - 3) $a=0, b=0, c=0$

h) For non-positive input, identify test points.

- Test points for non-positive input:
 - 1) $a=-4.0, b=3.2, c=4.5$
 - 2) $a=5, b=-4.2, c=-3.2$
 - 3) $a=4, b=5, c=-10$

Section B

The code below is part of a method in the ConvexHull class in the VMAP system. The following is a small fragment of a method in the ConvexHull class. For the purposes of this exercise you do not need to know the intended function of the method.

The parameter `p` is a Vector of Point objects, `p.size()` is the size of the vector `p`, `(p.get(i)).x` is the x component of the `i`th point appearing in `p`, similarly for `(p.get(i)).y`. This exercise is concerned with structural testing of code and so the focus is on creating test sets that satisfy some particular coverage criterion.

```
Vector doGraham(Vector p) {
    int i,j,min,M;

    Point t;
    min = 0;

    // search for minimum:
    for(i=1; i < p.size(); ++i) {
        if( ((Point) p.get(i)).y <
            ((Point) p.get(min)).y )
        {
            min = i;
        }
    }

    // continue along the values with same y component
    for(i=0; i < p.size(); ++i) {
        if( ((Point) p.get(i)).y ==
            ((Point) p.get(min)).y ) &&
            (((Point) p.get(i)).x >
              ((Point) p.get(min)).x ))
        {
            min = i;
        }
    }
}
```

For the given code fragment you should carry out the following activities.

2. Construct test sets for your flow graph that are adequate for the following criteria:

- a. Statement Coverage.
- b. Branch Coverage.
- c. Basic Condition Coverage.

Ans:

```
1.  int i,j,min,M;
2.  Point t;
3.  min=0;
4.  for(i=1;i<p.size();++i)
    {
5.      if(((Point)P.get(i)).y<((Point)P.get(min)).y)
6.          min=i;
    }

7.  for(i=0;i<p.size();++i)
    {
8.      if(((Point)P.get(i)).y==((Point)P.get(min)).y
        ((Point)P.get(i)).x>((Point)P.get(min)).x)
9.          min=i;
    }
```

Test cases:

1) p=[(x=2,y=2),(x=2,y=3),(x=1,y=3),(x=1,y=4)]

Statements covered = { 1,2,3,4,5,7,8}

Branches covered = {5,8}

Basic conditions covered = {5-false, 8-false}

2) p=[(x=2,y=3),(x=3,y=4),(x=1,y=2),(x=5,y=6)]

Statements covered = { 1,2,3,4,5,6,7}

Branches covered = {5,8}

Basic conditions covered = {5-false,true, 8-false}

3) p=[(x=1,y=5),(x=2,y=7),(x=3,y=5),(x=4,y=5),(x=5,y=6)]

Statements covered = { 1,2,3,4,5,6,7,8,9}

Branches covered = {5,8}

Basic conditions covered = {5-false,true, 8-false,true}

4) $p=[(x=1,y=2)]$

Statements covered = { 1,2,3,7,8} Branches covered = {8}

Basic conditions covered = {}

5) $p=[]$

Statements covered = { 1,2,3} Branches covered = {}

Basic conditions covered = {}