

it - 314
software engineering

lab -7
G- 28
id : 202001278
name : shraddha

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?

Solution :

There will be equivalence classes based on the input constraints

- 1) $E1 = 1 \leq \text{date} \leq 31$
- 2) $E2 = \text{date} < 1$
- 3) $E3 = \text{date} > 31$
- 4) $E4 = \text{month} > 12$
- 5) $E5 = \text{month} < 1$
- 6) $E6 = 1 \leq \text{month} \leq 12$
- 7) $E7 = \text{year} > 2015$
- 8) $E8 = \text{year} < 1900$
- 9) $E9 = 1900 \leq \text{year} \leq 2015$

Found nine equivalent classes

e.class	day	Month	year	output
E1	3	12	2014	2/12/2014
E2	0	1	2010	invalid
E3	32	12	1998	invalid
E4	5	13	1994	invalid
E5	21	-1	1976	invalid
E6	5	9	2002	4/9/2002
E7	13	4	2017	invalid
E8	19	7	1899	invalid
E9	1	5	1980	30/4/1980

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:

P1.

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.

Code :

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

Test cases :

Test cases:

1) $v=1$, $a=\{4,3,2,1,-3,-2,0\}$, expected output = 3

Equivalence Partitioning

2) $v=-3$, $a=\{-3,-3,0,6,7\}$ expected output = 0

Boundary Value Analysis

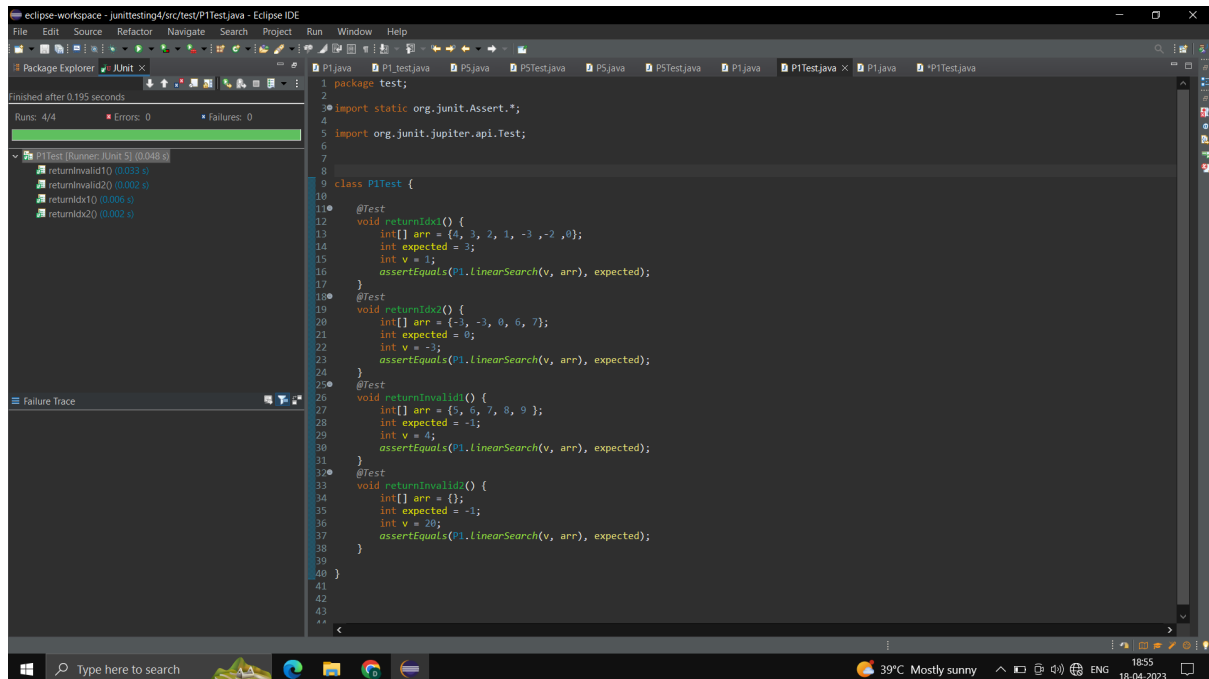
3) $v=4$, $a=\{5,6,7,8,9\}$ expected output = -1

Boundary Value Analysis

4) $v=20$, $a= \{\}$ expected output = -1

Equivalence Partitioning

Junit testing :



P2.

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

Code :

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

Test cases :

1) $v=2$, $a=\{4,2,3,5,2,1,6,7\}$ expected output = 2

Equivalence Partitioning

2) $v=3$, $a=\{6,7,3,6,8,9\}$ expected output =1

Boundary Value Analysis

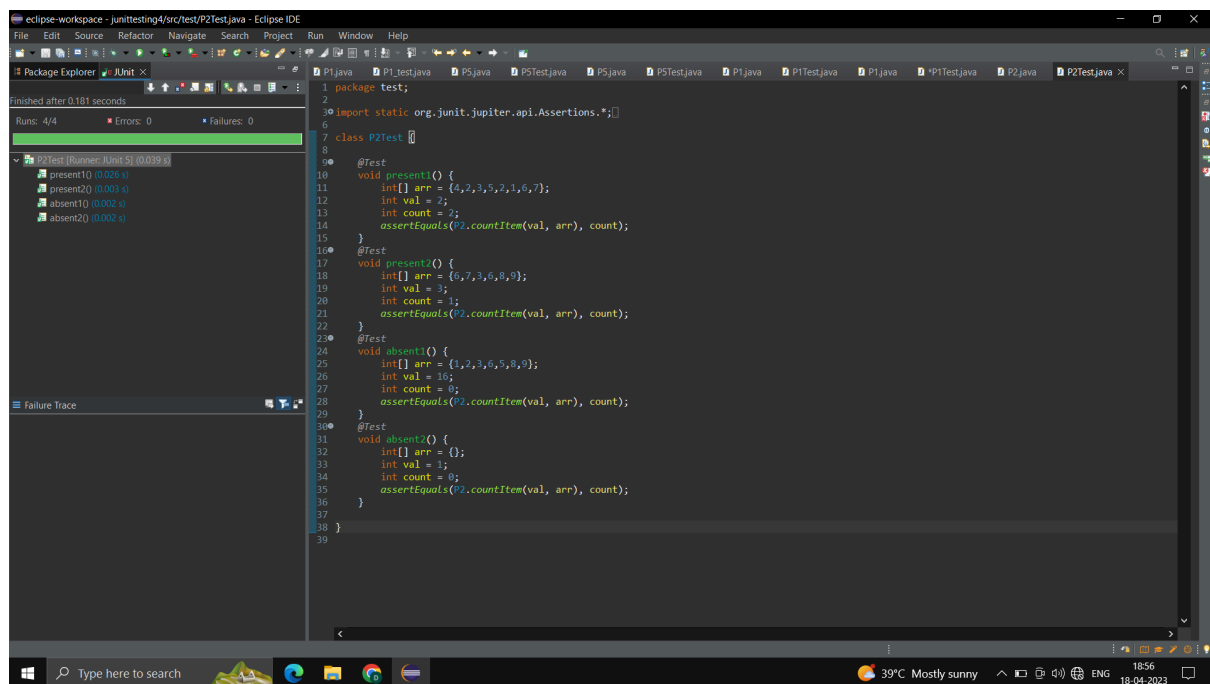
3) $v= 16$, $a= \{1,2,3,6,5,8,9\}$ expected output =0

Boundary Value Analysis

4) $v=1$, $a =\{\}$, expected output = 0

Equivalence Partitioning

Junit testing :



The screenshot shows the Eclipse IDE interface. The top part displays the JUnit test results for the `P2Test` class. The tests passed are:

- `present10` (0.002s)
- `present20` (0.003s)
- `absent10` (0.002s)
- `absent20` (0.002s)

The bottom part shows the source code of `P2Test.java`:

```
1 package test;
2
3 import static org.junit.jupiter.api.Assertions.*;
4
5
6 class P2Test {
7
8     @Test
9     void present1() {
10         int[] arr = {4,2,3,5,2,1,6,7};
11         int val = 2;
12         int count = 2;
13         assertEquals(P2.countItem(val, arr), count);
14     }
15
16     @Test
17     void present2() {
18         int[] arr = {6,7,3,6,8,9};
19         int val = 3;
20         int count = 1;
21         assertEquals(P2.countItem(val, arr), count);
22     }
23
24     @Test
25     void absent1() {
26         int[] arr = {1,2,3,6,5,8,9};
27         int val = 16;
28         int count = 0;
29         assertEquals(P2.countItem(val, arr), count);
30     }
31
32     @Test
33     void absent2() {
34         int[] arr = {};
35         int val = 1;
36         int count = 0;
37         assertEquals(P2.countItem(val, arr), count);
38     }
39 }
```

P3.

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 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);
}
```

Test cases:

1) v=2 , a= { 0, 1,2,3,4} expected output = 2

Equivalence Partitioning

2) v= -4 , a= {1,2,3,4,5 } expected output = -1

Boundary Value Analysis

3) v=5 , a= {2,3,4,5,7,6} expected output = 3

Equivalence Partitioning

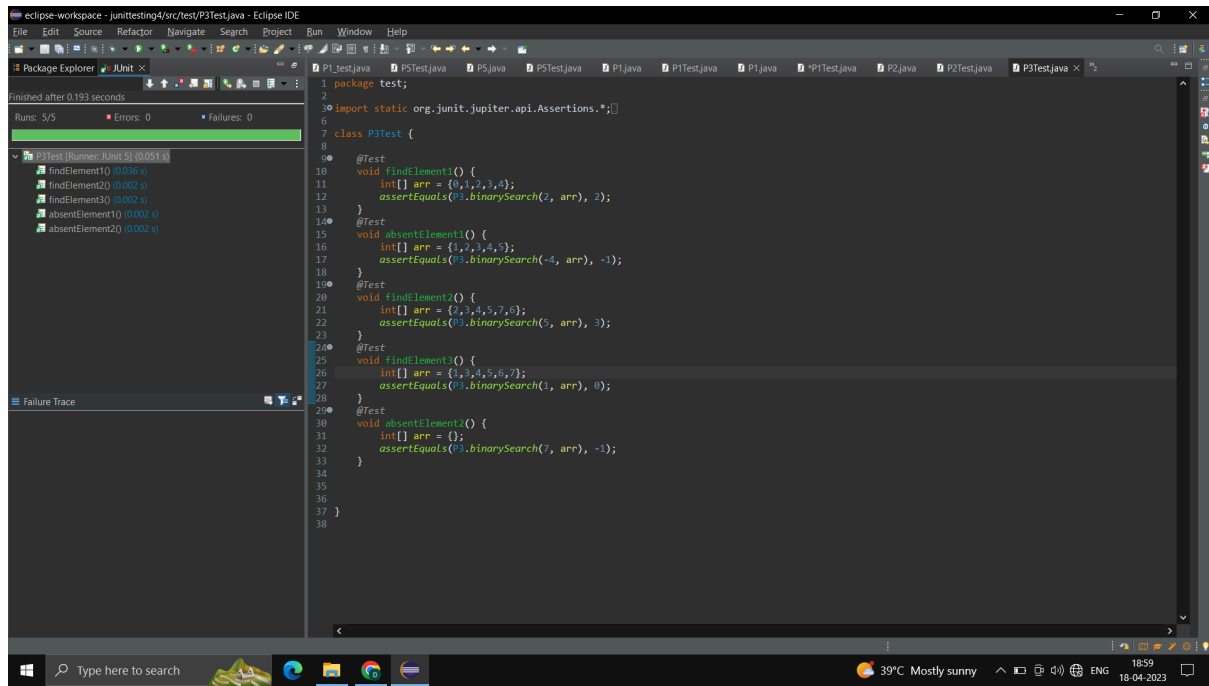
4) v=1 , a= {1,3,4,5,6,7} expected output = 0

Boundary Value Analysis

5) v=7 a = [] expected output = -1

Equivalence Partitioning

Junit testing :



P4.

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).

Code :

```
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);
}

```

Test cases:

1) a=7,b=7,c=7 expected output = EQUILATERAL

Equivalence Partitioning

2) a=1,b=2,c=3 expected output= INVALID

Boundary Value Analysis

3) a=-0,b=0,c=0 expected output = INVALID

Boundary Value Analysis

4) a=3,b=4,c=5 expected output = SCALENE

Boundary Value Analysis

5) a=6,b=6,c=9 expected output = ISOSCELES

Equivalence Partitioning

6) a=7, b=7, c=14 expected output = INVALID

Boundary Value Analysis

7) a=100,b=100,c=100 expected output = EQUILATERAL

Equivalence Partitioning

8) a=9,b=8,c=100 expected output= INVALID

Boundary Value Analysis

9) a=-8,b=-8,c=-8 expected output= INVALID

Boundary Value Analysis

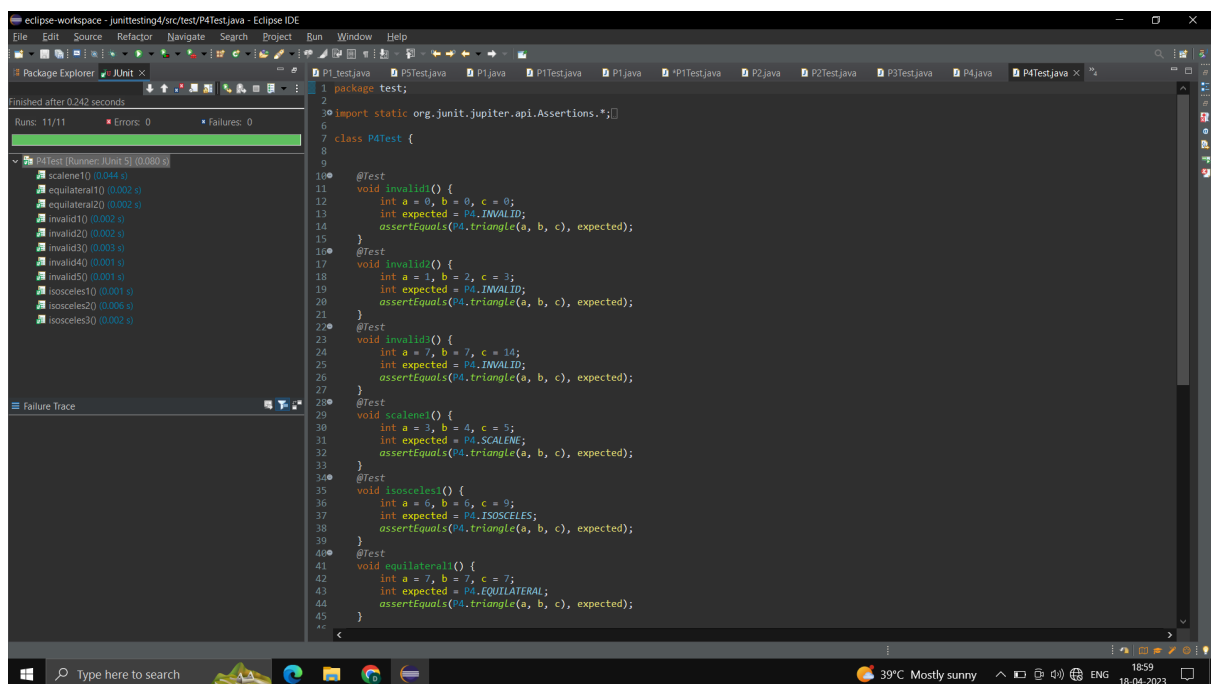
10) a=12, b=10, c=10 expected output = ISOSCELES

Equivalence Partitioning

11) a=150, b=100, c=150 expected output = ISOSCELES

Equivalence Partitioning

Junit testing :



P5.

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).

Code :

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;
}

```

Test cases:

1)s1="oft", s2="often" , expected output =true

Equivalence Partitioning

2)s1="abcd" , s2="abc" , expected output = false

Equivalence Partitioning

3)s1="student" , s2="student", expected output=true

Boundary Value Analysis

4) s1="one" , s2="two", expected output=false

Boundary Value Analysis

5)s1="",s2="sdf" , expected output=true

Boundary Value Analysis

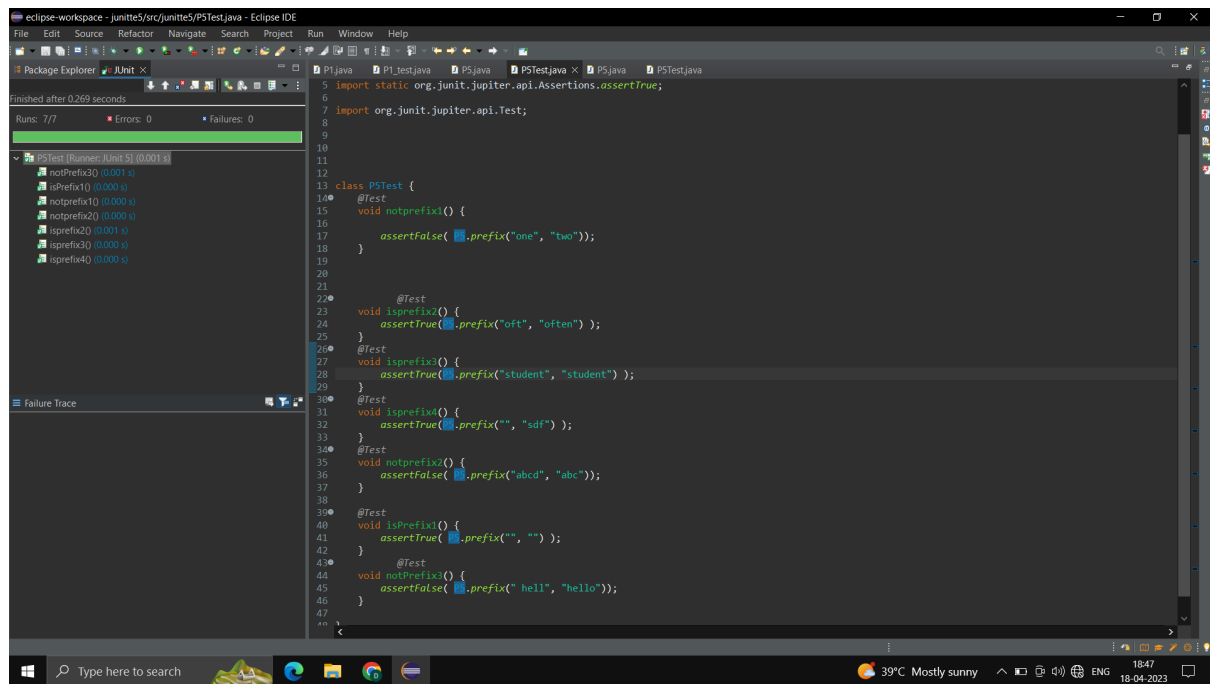
6)s1="",s2="" , expected output=true

Boundary Value Analysis

7)s1=" hell" , s2="hello" , expected output = false

Equivalence Partitioning

Junit testing :



P6: 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

The following are the equivalence classes for different types of triangles

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)

e.class	Test case	output
E1	$a=1.1$ $b=2.1$ $c=3.2$	Invalid /not a triangle
E2	$a=-1$ $b=5$ $c=6$	Invalid /not a triangle
E3	$a=7$ $b=6$ $c=1$	Invalid /not a triangle

E4	a=7 b=7 c=7	Equilateral
E5	a=4 b=4 c=5	isosceles
E6	a=8 b=6 c=8	isosceles
E7	a=9 b=8 c=8	isosceles
E8	a=10 b=11 c=12	scalene
E9	a=3 b=4 c=5	Right angled
E10	a=13 b=12 c=11	Right angled
E11	a=8 b=17 c=15	Right angled

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=7 b=8 c=14

Scalene

2) a=-5 b=-3 c=-9

Scalene

3) a=0.6 b=0.7 c=0.5

Scalene

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) $a=5$ $b=4$ $c=5$

Isosceles

2) $a=5$ $b=11$ $c=5$

invalid

3) $a=0$ $b=4$ $c=0$

invalid

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) $a=0$ $b=0$ $c=0$ ($a=b=c$)

Invalid

2) $a=-10$ $b=-10$ $c=-10$

Invalid

3) $a=0.10$ $b=0.10$ $c=0.10$

equilateral

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) $a=3$, $b=4$, $c=5$

Right angled

2) $a=-5, b=-12, c=13$

Invalid

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

Test cases to verify the boundary condition:

1) $a=1, b=2, c=56$

Invalid

2) $a=-4.5, b=-6.5, c=10$

Invalid

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

Test points for non-positive input:

1) $a=-1, b=2, c=3$

Invalid

2) $a=-4, b=-5, c=-7$

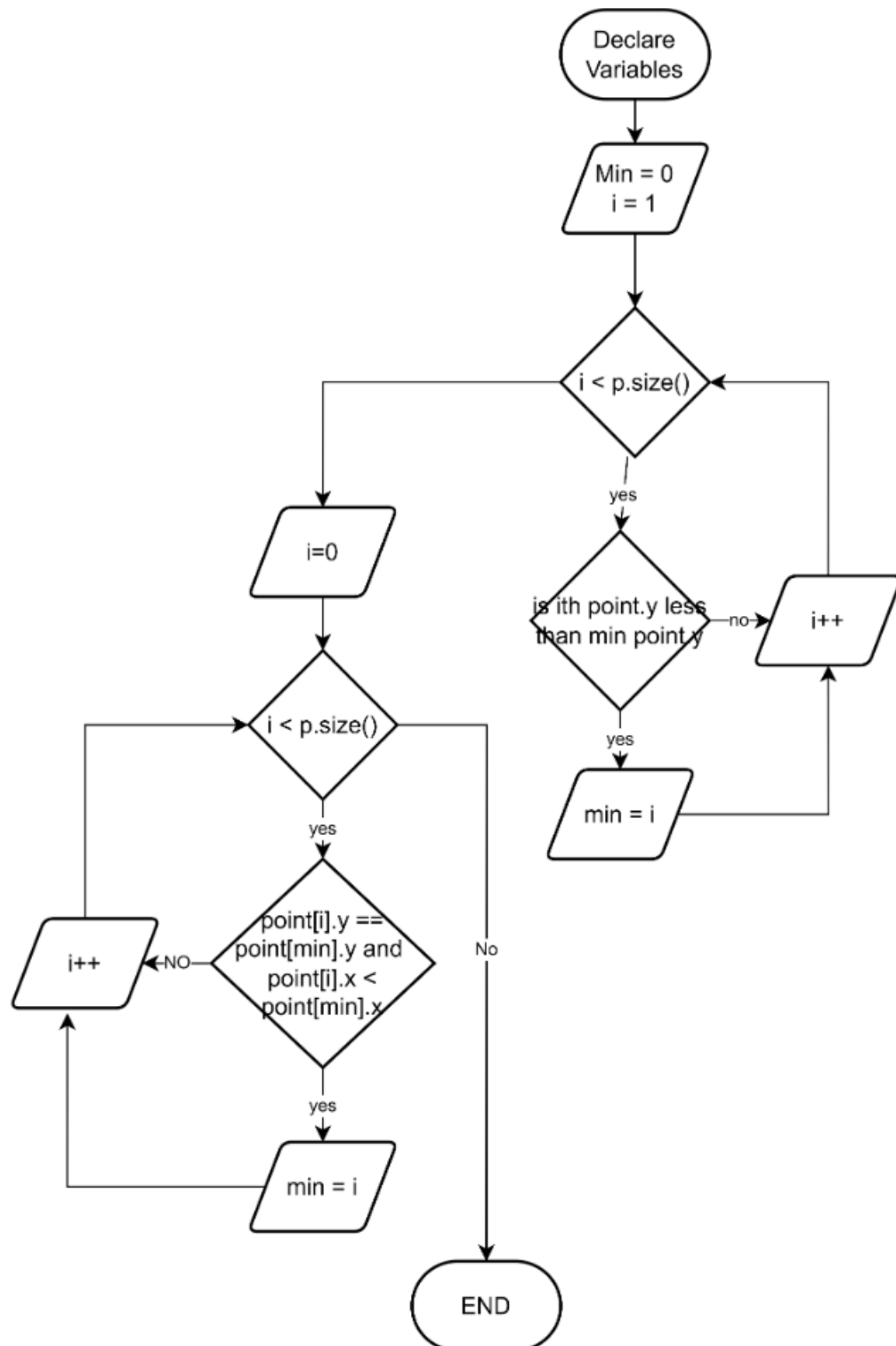
Invalid

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 ρ , 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.

1) Control flow graph :



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.

The following are the test cases and their corresponding coverages of statements

Test cases:

Statement coverage :

testcase	input	output(expected)
1	$\rho = []$	Empty Vector
2	$\rho = [(1,1)]$	Vector - Single Point
3	$p = [(1,1),(2,2),(3,3)]$	Vector - 3 Points

Branch coverage :

testcase	input	output
1	$\rho = []$	Empty Vector
2	$p = [(1,1),(2,2)]$	Vector-two points
3	$p = [(1,1),(2,2),(3,1),(4,3)]$	Vec- 4 points
4	$p = [(1,2),(3,1),(2,1)]$	Vec-3 points in diff order

Basic condition coverage :

testcase	input	output
1	$p=[]$	Empty Vector
2	$p=[(1,1),(2,2)]$	Vector-two points
3	$p=[(1,1),(2,2),(3,1),(4,3)]$	Vec- 4 points
4	$p=[(1,2),(3,1),(2,1)]$	Vec-3 points
5	$p=[(1,1),(1,1),(1,1)]$	Vec - identical point
6	$p=[(1,1),(2,2),(1,1)]$	Vec - 2 identical point
