Software Lab-08
Student ID:-202201328
Name :- Vraj Khokhariya

Question 1)

1. **Equivalence Partitioning Test Cases**
   Valid Partitions (EP-1)

1. Day: 1 to 31 (varies based on the month)
2. Month: 1 to 12
3. Year: 1950 to 2020

Invalid Partitions (EP-2)

1. Day: Less than 1 or greater than the number of days in a specific month (e.g., day > 31 in July, day > 30 in November)
2. Month: Less than 1 or greater than 12
3. Year: Less than 1950 or greater than 2020

Equivalence Partitioning: Test Cases

- EP1-1 (10, 7, 1990) valid date
- EP1-2 (2, 3, 1955) valid date
- EP1-3 (31, 12, 2020) valid date
- EP1-4 (28, 2, 2008) valid date

Invalid Test Cases

- EP2-1 (32, 7, 1995) Invalid date
- EP2-2 (0, 11, 2000) Invalid date
- EP2-3 (15, 14, 1975) Invalid month
- EP2-4 (25, 5, 1935) Invalid year

---

2. **Boundary Value Analysis Test Cases**

Boundary Values Partitions:

1. Day: 1 and max number of days per month (e.g., 30 for November, 31 for July)
2. Month: 1 and 12
3. Year: 1950 and 2020

Test Cases

- BVA1-1 (2, 1, 1950) valid date
- BVA1-2 (1, 1, 1950) valid date
- BVA1-3 (31, 12, 2020) valid date
- BVA1-4 (30, 11, 1980) valid date

Invalid Test Cases

- BVA2-1 (32, 7, 2000) Invalid date
- BVA2-2 (0, 12, 2000) Invalid date
- BVA2-3 (29, 2, 1950) (non-leap year) Invalid date
- BVA2-4 (28, 2, 1950) valid date

Code

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

bool isLeapYear(int year) {
   return (year % 400 == 0) || (year % 100 != 0 && year % 4 == 0);
}

bool isValidDate(int day, int month, int year) {
   if (year < 1950 || year > 2020)
      return false;
   if (month < 1 || month > 12)
      return false;

   int daysInMonth[] = { 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31 };
```

```
    if (month == 2 && isLeapYear(year))
        daysInMonth[1] = 29;

    if (day < 1 || day > daysInMonth[month - 1])
        return false;

    return true;
}

string previousDate(int day, int month, int year) {
    if (!isValidDate(day, month, year)) {
        return "Error: Invalid date";
    }

    struct tm date = {0};
    date.tm_mday = day;
    date.tm_mon = month - 1;
    date.tm_year = year - 1900;
    time_t time = mktime(&date) - 86400;
    struct tm *prevDate = localtime(&time);
    char buffer[11];
    strftime(buffer, sizeof(buffer), "%d-%m-%Y", prevDate);
    return string(buffer);
}

void runTestCases() {
    struct TestCase {
        int day, month, year;
        string expected;
```

```
};

TestCase testCases[] = {
    // Valid
    {10, 7, 1990, "09-07-1990"},
    {2, 3, 1955, "01-03-1955"},
    {31, 12, 2020, "30-12-2020"},
    {28, 2, 2008, "27-02-2008"},
    // Invalid
    {32, 7, 1995, "Error: Invalid date"},
    {0, 11, 2000, "Error: Invalid date"},
    {15, 14, 1975, "Error: Invalid date"},
    {25, 5, 1935, "Error: Invalid date"},
    {2, 1, 1950, "01-01-1950"},
    {1, 1, 1950, "31-12-1949"},
    {31, 12, 2020, "30-12-2020"},
    {30, 11, 1980, "29-11-1980"},
    {32, 7, 2000, "Error: Invalid date"},
    {0, 12, 2000, "Error: Invalid date"},
    {29, 2, 1950, "Error: Invalid date"},
    {28, 2, 1950, "27-02-1950"},
};

for (const auto& testCase : testCases) {
    string result = previousDate(testCase.day, testCase.month, testCase.year);
    if (result == testCase.expected)
        cout << "Valid Date" << endl;
    else
        cout << "Invalid Date" << endl;
```

```
    }
}

int main() {
    runTestCases();
    return 0;
}
```

Question 2)

Test Cases

Equivalence Partitioning

Format: linearSearch(value, array[])

Function Call	**Expected Outcome**
linearSearch(4, {2, 4, 8, 10, 12})	1
linearSearch(6, {2, 4, 8, 10, 12})	-1 (invalid)
linearSearch(3, {})	-1 (invalid)

Boundary Value Analysis

Function Call	**Expected Outcome**
linearSearch(4, {4})	0
linearSearch(2, {2, 4, 8, 10, 12})	0
linearSearch(12, {2, 4, 8, 10, 12})	4
linearSearch(7, {})	-1 (invalid)

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

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

void runTests() {
    int testCase1[] = {2, 4, 8, 10, 12};
    int testCase2[] = {};
    int testCase3[] = {4};

    // Equivalence Partitioning Tests
    cout << "Test 1 (v = 4, a = {2,4,8,10,12}): " << linearSearch(4, testCase1, 5) << endl;
    cout << "Test 2 (v = 6, a = {2,4,8,10,12}): " << linearSearch(6, testCase1, 5) << endl;
    cout << "Test 3 (v = 3, a = {}): " << linearSearch(3, testCase2, 0) << endl;

    // Boundary Value Analysis Tests
    cout << "Test 4 (v = 4, a = {4}): " << linearSearch(4, testCase3, 1) << endl;
    cout << "Test 5 (v = 2, a = {2,4,8,10,12}): " << linearSearch(2, testCase1, 5) << endl;
    cout << "Test 6 (v = 12, a = {2,4,8,10,12}): " << linearSearch(12, testCase1, 5) << endl;
    cout << "Test 7 (v = 7, a = {}): " << linearSearch(7, testCase2, 0) << endl;
}

int main() {
    runTests();
```

```
    return 0;
}
```

Outcome:

Test Number	Function Call	Output
Test 1	linearSearch(4, {2,4,8,10,12})	1
Test 2	linearSearch(6, {2,4,8,10,12})	-1
Test 3	linearSearch(3, {})	-1
Test 4	linearSearch(4, {4})	0
Test 5	linearSearch(2, {2,4,8,10,12})	0
Test 6	linearSearch(12, {2,4,8,10,12})	4
Test 7	linearSearch(7, {})	-1

Question 3)

Test Case
Tester Action and Input Data
Format: countItem(value, array[])

Function Call	Expected Outcome
countItem(3, {3, 3, 6, 9, 3})	3
countItem(4, {4, 5, 6, 4, 4})	3
countItem(8, {1, 2, 3, 5})	0
countItem(5, {})	0
countItem(2, {2, 2, 2, 2})	4
countItem(1, {1})	1

Function Call	Expected Outcome
countItem(6, {5})	0

Invalid Inputs

Function Call	Expected Outcome
countItem(-3, {1, 2, 3, 4})	0
countItem('b', {1, 2, 3, 4})	Error: Non-integer input
countItem(3, {1, null, 3, null})	Error: Null values in array
countItem(3, {1, "two", 3, "four"})	Error: Mixed data types in array

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

int countItem(int v, int a[], int size) {
   if (size < 0)
      throw invalid_argument("Array size cannot be negative");
   int count = 0;
   for (int i = 0; i < size; i++) {
      if (!cin.good())
         throw invalid_argument("Invalid input type detected");
      if (a[i] == v)
         count++;
   }
   return count;
}
```

```
void runTests() {
    int testCase1[] = {3, 3, 6, 9, 3};
    int testCase2[] = {4, 5, 6, 4, 4};
    int testCase3[] = {};
    int testCase4[] = {2, 2, 2, 2};
    int testCase5[] = {1};
    int testCase6[] = {5};

    // Valid Inputs
    cout << "Test 1 (v = 3, a = {3,3,6,9,3}): " << countItem(3, testCase1, 5) << endl;
    cout << "Test 2 (v = 4, a = {4,5,6,4,4}): " << countItem(4, testCase2, 5) << endl;
    cout << "Test 3 (v = 8, a = {1,2,3,5}): " << countItem(8, testCase2, 4) << endl;
    cout << "Test 4 (v = 5, a = {}): " << countItem(5, testCase3, 0) << endl;
    cout << "Test 5 (v = 2, a = {2,2,2,2}): " << countItem(2, testCase4, 4) << endl;
    cout << "Test 6 (v = 1, a = {1}): " << countItem(1, testCase5, 1) << endl;
    cout << "Test 7 (v = 6, a = {5}): " << countItem(6, testCase6, 1) << endl;

    // Invalid Inputs
    try {
        cout << "Test 8 (v = 'b', a = {1,2,3,4}): " << countItem('b', testCase2, 4) << endl;
    } catch (const invalid_argument& e) {
        cout << "Test 8: " << e.what() << endl;
    }
}

int main() {
    runTests();
    return 0;
```

}

Outcome:

Test Number	Function Call	Output
Test 1	countItem(3, {3,3,6,9,3})	3
Test 2	countItem(4, {4,5,6,4,4})	3
Test 3	countItem(8, {1,2,3,5})	0
Test 4	countItem(5, {})	0
Test 5	countItem(2, {2,2,2,2})	4
Test 6	countItem(1, {1})	1
Test 7	countItem(6, {5})	0
Test 8	countItem('b', {1,2,3,4})	Error: Non-integer input

Question 4)

Test Case
Tester Action and Input Data
Format: binarySearch(value, array[])

Equivalence Partitioning: Valid Inputs

Function Call	Expected Outcome
binarySearch(4, {1, 3, 4, 7, 9})	2
binarySearch(8, {1, 3, 4, 7, 9})	-1
binarySearch(0, {1, 3, 4, 7, 9})	-1
binarySearch(10, {1, 3, 4, 7, 9})	-1
binarySearch(3, {3})	0
binarySearch(5, {3})	-1

Equivalence Partitioning: Invalid Inputs

Function Call	Expected Outcome
binarySearch(4, {})	-1
binarySearch(4, {7, 3, 5, 1, 4})	Invalid, array not sorted
binarySearch(4, {2, null, 4, null})	Error: Null values in array
binarySearch(4, {1, "three", 4, "nine"})	Error: Mixed data types in array

---

Boundary Value Analysis: Valid Inputs

Function Call	Expected Outcome
binarySearch(1, {1, 3, 4, 7, 9})	0
binarySearch(9, {1, 3, 4, 7, 9})	4
binarySearch(0, {1, 3, 4, 7, 9})	-1
binarySearch(10, {1, 3, 4, 7, 9})	-1

Boundary Value Analysis: Invalid Inputs

Function Call	Expected Outcome
binarySearch(1, {})	-1
binarySearch(4, {5})	-1

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

int binarySearch(int v, int a[], int size) {
```

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

void runTests() {
    int testCase1[] = {1, 3, 4, 7, 9};
    int testCase2[] = {3};
    int testCase3[] = {};
    int testCase4[] = {7, 3, 5, 1, 4};

    // Equivalence Partitioning: Valid Inputs
    cout << "Test 1 (v = 4, a = {1, 3, 4, 7, 9}): " << binarySearch(4, testCase1, 5) << endl;
    cout << "Test 2 (v = 8, a = {1, 3, 4, 7, 9}): " << binarySearch(8, testCase1, 5) << endl;

    // Boundary Value Analysis: Valid Inputs
    cout << "Test 3 (v = 1, a = {1, 3, 4, 7, 9}): " << binarySearch(1, testCase1, 5) << endl;
    cout << "Test 4 (v = 9, a = {1, 3, 4, 7, 9}): " << binarySearch(9, testCase1, 5) << endl;

    // Equivalence Partitioning: Invalid Inputs
    cout << "Test 5 (v = 4, a = {}): " << binarySearch(4, testCase3, 0) << endl;
```

```
    // Invalid sorted array input test (uncomment if sorted check is implemented)
    // cout << "Test 6 (v = 4, a = {7, 3, 5, 1, 4}): Invalid" << endl;
}

int main() {
    runTests();
    return 0;
}
```

Outcome:

Test Number	Function Call	Output
Test 1	binarySearch(4, {1, 3, 4, 7, 9})	2
Test 2	binarySearch(8, {1, 3, 4, 7, 9})	-1
Test 3	binarySearch(1, {1, 3, 4, 7, 9})	0
Test 4	binarySearch(9, {1, 3, 4, 7, 9})	4
Test 5	binarySearch(4, {})	-1

Question 5)

Test Cases
Input Values (sides of a triangle: a, b, c)
Expected Output (Triangle Type)

a	b	c	Expected Output
4	4	4	0
4	4	1	1

a	b	c	Expected Output
5	7	8	2
2	2	5	3
6	6	6	0
3	3	4	1
1	4	5	2
1	1	3	3
-1	-1	-1	3
0	0	1	3

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

const int EQUILATERAL = 0;
const int ISOSCELES = 1;
const int SCALENE = 2;
const int INVALID = 3;

int triangle(int a, int b, int c) {
   if (a >= b + c || b >= a + c || c >= a + b) {
      return INVALID; // Not a triangle
   }
   if (a == b && b == c) {
      return EQUILATERAL; // All sides equal
   }
   if (a == b || a == c || b == c) {
      return ISOSCELES; // Two sides equal
   }
```

```
    return SCALENE; // No sides equal
}

int main() {
    int testCases[][3] = {
        {4, 4, 4}, {4, 4, 1}, {5, 7, 8},
        {2, 2, 5}, {6, 6, 6}, {3, 3, 4},
        {1, 4, 5}, {1, 1, 3}, {-1, -1, -1},
        {0, 0, 1}
    };

    int expectedOutputs[] = {
        EQUILATERAL, ISOSCELES, SCALENE,
        INVALID, EQUILATERAL, ISOSCELES,
        SCALENE, INVALID, INVALID,
        INVALID
    };

    for (int i = 0; i < sizeof(testCases) / sizeof(testCases[0]); i++) {
        int a = testCases[i][0];
        int b = testCases[i][1];
        int c = testCases[i][2];
        int result = triangle(a, b, c);
        cout << "Triangle with sides (" << a << ", " << b << ", " << c << "): expected "
             << expectedOutputs[i] << ", got " << result << endl;
    }

    return 0;
}
```

Outcome:

Triangle Sides (a, b, c)	Expected Output	Actual Output
(4, 4, 4)	0	0
(4, 4, 1)	1	1
(5, 7, 8)	2	2
(2, 2, 5)	3	3
(6, 6, 6)	0	0
(3, 3, 4)	1	1
(1, 4, 5)	2	2
(1, 1, 3)	3	3
(-1, -1, -1)	3	3
(0, 0, 1)	3	3

Question 6)

Test Cases

Input Strings (s1, s2)
Expected Output (true/false)

s1	s2	Expected Output
"code"	"coding"	true
"open"	"open sesame"	true
"example"	"sample"	false
"python"	"pythonista"	true
"test"	"tester"	true
"xyz"	"xy"	false

s1	s2	Expected Output
""	"non-empty"	true
"some"	""	false
"main"	"mains"	false
"check"	"Check"	false

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

bool prefix(const string& s1, const string& s2) {
    if (s1.length() > s2.length()) {
        return false;
    }
    for (size_t i = 0; i < s1.length(); i++) {
        if (s1[i] != s2[i]) {
            return false;
        }
    }
    return true;
}

int main() {
    string testCases[][2] = {
        {"code", "coding"},
        {"open", "open sesame"},
        {"example", "sample"},
        {"python", "pythonista"},
        {"test", "tester"},
```

```
        {"xyz", "xy"},
        {"", "non-empty"},
        {"some", ""},
        {"main", "mains"},
        {"check", "Check"}
    };

    bool expectedOutputs[] = {
        true, true, false, true, true, false, true, false, false, false
    };

    for (size_t i = 0; i < sizeof(testCases) / sizeof(testCases[0]); i++) {
        string s1 = testCases[i][0];
        string s2 = testCases[i][1];
        bool result = prefix(s1, s2);
        cout << "Prefix check for (" << s1 << ", " << s2 << "): expected "
             << expectedOutputs[i] << ", got " << result << endl;
    }

    return 0;
}
```

Outcome:

Prefix Check (s1, s2)	Expected Output	Actual Output
(code, coding)	1	1
(open, open sesame)	1	1
(example, sample)	0	0
(python, pythonista)	1	1

Prefix Check (s1, s2)	Expected Output	Actual Output
(test, tester)	1	1
(xyz, xy)	0	0
(, non-empty)	1	1
(some,)	0	0
(main, mains)	0	0
(check, Check)	0	0

Question 7)

a) Equivalence Classes:

1. **Valid Triangle (General)**:
    - The sides can form a triangle (the sum of any two sides must be greater than the third).
2. **Equivalence Class 1 (EC1)**: Valid triangle where a==b==ca == b == ca==b==c (Equilateral triangle).
   Equivalence Class 2 (EC2): Valid triangle where a==b≠ca == b \neq ca==b□=c or a≠b==ca \neq b == ca□=b==c or a==c≠ba == c \neq ba==c□=b (Isosceles triangle).
   Equivalence Class 3 (EC3): Valid triangle where a≠b≠ca \neq b \neq ca□=b□=c (Scalene triangle).
   Equivalence Class 4 (EC4): Valid right-angled triangle where A2+B2=C2A^2 + B^2 = C^2A2+B2=C2 (Pythagorean theorem).
3. **Invalid Triangle**:
    - The sum of two sides is less than or equal to the third.
4. **Equivalence Class 5 (EC5)**: Invalid triangle where a+b≤ca + b \leq ca+b≤c or a+c≤ba + c \leq ba+c≤b or b+c≤ab + c \leq ab+c≤a.
   Equivalence Class 6 (EC6): Invalid triangle where one or more sides are zero or negative.

b) Test Cases for Equivalence Classes:

Test Case No.	Input (a, b, c)	Expected Output	Equivalence Class Covered
TC1	(4, 4, 4)	EQUILATERAL	EC1
TC2	(5, 5, 3)	ISOSCELES	EC2
TC3	(2, 3, 4)	SCALENE	EC3
TC4	(5, 12, 13)	SCALENE (Right-Angle)	EC4
TC5	(2, 2, 5)	INVALID	EC5
TC6	(1, 1, 3)	INVALID	EC5
TC7	(0, 3, 4)	INVALID	EC6
TC8	(-2, 3, 5)	INVALID	EC6
TC9	(6.0, 6.0, 6.0)	EQUILATERAL	EC1
TC10	(7.0, 7.0, 9.0)	ISOSCELES	EC2
TC11	(8.0, 6.0, 10.0)	SCALENE	EC3
TC12	(9.0, 12.0, 15.0)	RIGHT-ANGLE	EC4
TC13	(2.0, 3.0, 5.0)	INVALID	EC5
TC14	(0.0, 4.0, 4.0)	INVALID	EC6
TC15	(-3.0, 2.0, 4.0)	INVALID	EC6

c) Boundary Condition for A + B > C (Scalene Triangle):

Test Case No.	Input (A, B, C)	Expected Output	Explanation
TC16	(2.0, 2.0, 4.0)	INVALID	A+B=CA + B = CA+B=C (invalid boundary)
TC17	(2.0, 3.0, 5.1)	SCALENE	A+B>CA + B > CA+B>C (valid boundary)
TC18	(3.0, 3.0, 6.0)	INVALID	A+B=CA + B = CA+B=C (invalid boundary)

d) Boundary Condition for A = C (Isosceles Triangle):

Test Case No.	Input (A, B, C)	Expected Output	Explanation
TC19	(5.0, 6.0, 5.0)	ISOSCELES	A=CA = CA=C (valid boundary)
TC20	(7.0, 8.0, 7.0)	ISOSCELES	A=CA = CA=C (valid boundary)
TC21	(4.0, 4.0, 7.0)	ISOSCELES	A=BA = BA=B (valid boundary)

e) Boundary Condition for A = B = C (Equilateral Triangle):

Test Case No.	Input (A, B, C)	Expected Output	Explanation
TC22	(4.0, 4.0, 4.0)	EQUILATERAL	A=B=CA = B = CA=B=C (valid boundary)
TC23	(7.0, 7.0, 7.0)	EQUILATERAL	A=B=CA = B = CA=B=C (valid boundary)
TC24	(8.1, 8.1, 8.1)	EQUILATERAL	A=B=CA = B = CA=B=C (valid boundary)

f) Boundary Condition for $A^2 + B^2 = C^2$ (Right-Angle Triangle):

Test Case No.	Input (A, B, C)	Expected Output	Explanation
TC25	(5.0, 12.0, 13.0)	RIGHT-ANGLE	A2+B2=C2A^2 + B^2 = C^2A2+B2=C2 (valid boundary)
TC26	(9.0, 12.0, 15.0)	RIGHT-ANGLE	A2+B2=C2A^2 + B^2 = C^2A2+B2=C2 (valid boundary)
TC27	(6.0, 8.0, 10.0)	RIGHT-ANGLE	A2+B2=C2A^2 + B^2 = C^2A2+B2=C2 (valid boundary)

g) Non-Triangle Case Boundaries:

Test Case No.	Input (A, B, C)	Expected Output	Explanation
TC28	(2.0, 2.0, 4.0)	INVALID	A+B=CA + B = CA+B=C (invalid)
TC29	(3.0, 4.0, 8.0)	INVALID	A+B<CA + B < CA+B<C (invalid)

h) Non-Positive Input Test Cases:

Test Case No.	Input (A, B, C)	Expected Output	Explanation
TC30	(0.0, 6.0, 8.0)	INVALID	Non-positive side length (A=0)
TC31	(5.0, 0.0, 9.0)	INVALID	Non-positive side length (B=0)
TC32	(-4.0, 5.0, 6.0)	INVALID	Negative side length (A=-4)