Software Lab 8

StudentId: 202201478

Shrey Bavishi

Q1

1. Equivalence Partitioning Test Cases

Valid Partitions (EP-1)

1)Day: 1 to 31 (varies based on month)
2)Month: 1 to 12
3)Year: 1900 to 2015

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 January, day > 30 in April)
2)Month: Less than 1 or greater than 12
3) Year: Less than 1900 or greater than 2015

Equivalence Partitioning: Test Cases

EP1-1	(15, 8, 2000)	valid date
EP1-2	(1, 1, 2000)	valid date
EP1-3	(31, 12, 2015)	valid date
EP1-4	(29, 2, 2004)	valid date
EP2-1	(32, 8, 2000)	Invalid date
EP2-2	(0, 12, 2005)	Invalid date
EP2-3	(15, 13, 1998)	Invalid month
EP2-4	(25, 6, 1885)	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 April, 31 for January, etc.)
2)Month: 1 and 12
3)Year: 1900 and 2015

Test Cases

BVA1-1	(2, 1, 1900)	valid date
BVA1-2	(1, 1, 1900)	valid date
BVA1-3	(31, 12, 2015)	valid date
BVA1-4	(30, 4, 2000)	valid date
BVA2-1	(32, 1, 2005)	Invalid date
BVA2-2	(0, 12, 2005)	Invalid date
BVA2-3	(29, 2, 1900) (non-leap year)	Invalid date
BVA2-4	(28, 2, 1900) (non-leap year)	valid date

```
#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 < 1900 || year > 2015)
        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; // tm_mon is 0-11
    date.tm_year = year - 1900; // tm_year is years since 1900

    time_t time = mktime(&date) - 86400; // Subtract one day (86400 seconds)
    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 cases
        {15, 8, 2000, "14-08-2000"},
        {1, 1, 2000, "31-12-1999"},
        {31, 12, 2015, "30-12-2015"},
        {29, 2, 2004, "28-02-2004"},

        //: Invalid cases
        {32, 8, 2000, "Error: Invalid date"},
        {0, 12, 2005, "Error: Invalid date"},
        {15, 13, 1998, "Error: Invalid date"},
        {25, 6, 1885, "Error: Invalid date"},

        // Boundary Value Analysis: Valid cases
        {2, 1, 1900, "01-01-1900"},
        {1, 1, 1900, "31-12-1899"},
        {31, 12, 2015, "30-12-2015"},
        {30, 4, 2000, "29-04-2000"},

        // Boundary Value Analysis: Invalid cases
        {32, 1, 2005, "Error: Invalid date"},
        {0, 12, 2005, "Error: Invalid date"},
        {29, 2, 1900, "Error: Invalid date"},
        {28, 2, 1900, "27-02-1900"},
    };
    for (const auto& testCase : testCases) {
         if (result == testCase.expected)
            cout << "Valid Date";
        else
            cout << " Invalid Date"<<endl;;
    }
}

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

Q2

Test Cases

Equivalence Partitioning **Of format linearSearch (V , Arr[ ])**	**Outcome**
linearSearch(5, {1, 2, 5,6, 9})	2
linearSearch(7, {1, 2, 5, 6, 9})	-1(invalid)
linearSearch(1, {})	-1 (invalid)
Boundary Value Analysis	
linearSearch(5, {5})	0
linearSearch(1, {1, 2, 5, 6, 9})	0
linearSearch(9, {1, 2, 5, 6, 9})	4
linearSearch(3, {})	-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[] = {1, 2, 5, 6, 9};
    int testCase2[] = {};
    int testCase3[] = {5};

    // Equivalence Partitioning Tests
    cout << "Test 1 (v = 5, a = {1,2,5,6,9}): " << linearSearch(5, testCase1, 5) << endl;
    cout << "Test 2 (v = 7, a = {1,2,5,6,9}): " << linearSearch(7, testCase1, 5) << endl;
    cout << "Test 3 (v = 1, a = {}): " << linearSearch(1, testCase2, 0) << endl;

    // Boundary Value Analysis Tests
    cout << "Test 4 (v = 5, a = {5}): " << linearSearch(5, testCase3, 1) << endl;
    cout << "Test 5 (v = 1, a = {1,2,5,6,9}): " << linearSearch(1, testCase1, 5) << endl;
    cout << "Test 6 (v = 9, a = {1,2,5,6,9}): " << linearSearch(9, testCase1, 5) << endl;
    cout << "Test 7 (v = 3, a = {}): " << linearSearch(3, testCase2, 0) << endl;
}

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

Outcome

```
Test 1 (v = 5, a = {1,2,5,6,9}): 2
Test 2 (v = 7, a = {1,2,5,6,9}): -1
Test 3 (v = 1, a = {}): -1
Test 4 (v = 5, a = {5}): 0
Test 5 (v = 1, a = {1,2,5,6,9}): 0
Test 6 (v = 9, a = {1,2,5,6,9}): 4
Test 7 (v = 3, a = {}): -1
```

Q3

Test Case

Tester Action and Input Data	Expected Outcome
Valid Inputs	

`countItem(5, {1, 5, 5, 3, 5})`	`3`
`countItem(2, {1, 2, 3, 4, 5})`	`1`
`countItem(7, {1, 2, 3, 4, 5})`	`0`
`countItem(1, {})`	`0`
`countItem(9, {9, 9, 9, 9})`	`4`
`countItem(5, {5})`	`1`
`countItem(2, {5})`	`0`
Invalid Inputs	
`countItem(-5, {1, 2, 3, 4, 5})`	`0`
`countItem('a', {1, 2, 3, 4, 5})`	Error: Non-integer input
`countItem(5, {1, null, 5, null})`	Error: Null values in array
`countItem(5, {1, "two", 5, "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[] = {1, 5, 5, 3, 5};
   int testCase2[] = {1, 2, 3, 4, 5};
   int testCase3[] = {};
   int testCase4[] = {5};
   int testCase5[] = {9, 9, 9, 9};

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

   cout << "Test 6 (v = -5, a = {1,2,3,4,5}): " << countItem(-5, testCase2, 5) << endl;


   try {
      cout << "Test 7 (v = 'a', a = {1,2,3,4,5}): " << countItem('a', testCase2, 5) << endl;
   }
   catch (const invalid_argument& e) {
      cout << "Test 7: " << e.what() << endl;
   }
}

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

Outcome

```
Test 1 (v = 5, a = {1,5,5,3,5}): 3
Test 2 (v = 2, a = {1,2,3,4,5}): 1
Test 3 (v = 7, a = {1,2,3,4,5}): 0
Test 4 (v = 1, a = {}): 0
Test 5 (v = 9, a = {9,9,9,9}): 4
Test 6 (v = -5, a = {1,2,3,4,5}): 0
Test 7 (v = 'a', a = {1,2,3,4,5}): 0
```

Q4

Test Cases

Tester Action and Input Data	Expected Outcome
Equivalence Partitioning: Valid Inputs	
`binarySearch(3, {1, 2, 3, 4, 5})`	`2`
`binarySearch(6, {1, 2, 3, 4, 5})`	`-1`
`binarySearch(0, {1, 2, 3, 4, 5})`	`-1`
`binarySearch(10, {1, 2, 3, 4, 5})`	`-1`
`binarySearch(1, {1})`	`0`
`binarySearch(2, {1})`	`-1`
Equivalence Partitioning: Invalid Inputs	
`binarySearch(3, {})`	`-1`
`binarySearch(3, {5, 1, 3, 2, 4})`	Invalid , array not sorted
`binarySearch(3, {1, null, 3, null})`	Error
`binarySearch(5, {1, "two", 3, "four"})`	Error
Boundary Value Analysis: Valid Inputs	
`binarySearch(1, {1, 2, 3, 4, 5})`	`0`
`binarySearch(5, {1, 2, 3, 4, 5})`	`4`
`binarySearch(0, {1, 2, 3, 4, 5})`	`-1`
`binarySearch(6, {1, 2, 3, 4, 5})`	`-1`
Boundary Value Analysis: Invalid Inputs	
`binarySearch(1, {})`	`-1`

binarySearch(2, {1})	-1

```
#include <iostream>
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, 2, 3, 4, 5};
    int testCase2[] = {1};
    int testCase3[] = {};
    int testCase4[] = {5, 1, 3, 2, 4};

    // Equivalence Partitioning: Valid Inputs
    cout << "Test 1 (v = 3, a = {1, 2, 3, 4, 5}): " << binarySearch(3, testCase1, 5) << endl;
    cout << "Test 2 (v = 6, a = {1, 2, 3, 4, 5}): " << binarySearch(6, testCase1, 5) << endl;

    // Boundary Value Analysis: Valid Inputs
    cout << "Test 3 (v = 1, a = {1, 2, 3, 4, 5}): " << binarySearch(1, testCase1, 5) << endl;
    cout << "Test 4 (v = 5, a = {1, 2, 3, 4, 5}): " << binarySearch(5, testCase1, 5) << endl;

    // Equivalence Partitioning: Invalid Inputs
    cout << "Test 5 (v = 3, a = {}): " << binarySearch(3, testCase3, 0) << endl;
}

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

Outcome

```
Test 1 (v = 3, a = {1, 2, 3, 4, 5}): 2
Test 2 (v = 6, a = {1, 2, 3, 4, 5}): -1
Test 3 (v = 1, a = {1, 2, 3, 4, 5}): 0
Test 4 (v = 5, a = {1, 2, 3, 4, 5}): 4
Test 5 (v = 3, a = {}): -1
```

Q5

Test Cases

Input Values (a, b, c)	Expected Output
3, 3, 3	0
3, 3, 2	1
3, 4, 5	2
1, 1, 2	3
5, 5, 5	0
2, 2, 3	1
2, 3, 4	2
1, 2, 3	3
0, 0, 0	3
-1, -1, -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;

    }

    if (a == b && b == c) {

        return EQUILATERAL;

    }

    if (a == b || a == c || b == c) {

        return ISOSCELES;

    }

    return SCALENE;

}

int main() {

    int testCases[][3] = {{3, 3, 3}, {3, 3, 2}, {3, 4, 5}, {1, 1, 2}, {5, 5, 5},{2, 2, 3}, {2, 3, 4}, {1, 2,
3},{0, 0, 0},{-1, -1, -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;}
```

```
Triangle with sides (3, 3, 3): expected 0, got 0
Triangle with sides (3, 3, 2): expected 1, got 1
Triangle with sides (3, 4, 5): expected 2, got 2
Triangle with sides (1, 1, 2): expected 3, got 3
Triangle with sides (5, 5, 5): expected 0, got 0
Triangle with sides (2, 2, 3): expected 1, got 1
Triangle with sides (2, 3, 4): expected 2, got 2
Triangle with sides (1, 2, 3): expected 3, got 3
Triangle with sides (0, 0, 0): expected 3, got 3
Triangle with sides (-1, -1, -1): expected 3, got 3
```

Q6

Test Cases

Input Strings (s1, s2)	**Expected Output**
"pre" , "prefix"	true
"hello" , "hello world"	true
"world" , "hello world"	false
"java" , "javascript"	true
"test" , "testing"	true
"abc" , "ab"	false
"" , "anything"	true

"non" , ""	false
"prefix" , "pre"	false
"test" , "Test"	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] = { {"pre", "prefix"},{"hello", "hello world"},{"world", "hello world"},{"java",
"javascript"},{"test", "testing"},{"abc", "ab"},{"", "anything"},{"non", ""},{"prefix", "pre"},{"test",
"Test"}};
    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 for (pre, prefix): expected 1, got 1
Prefix check for (hello, hello world): expected 1, got 1
Prefix check for (world, hello world): expected 0, got 0
Prefix check for (java, javascript): expected 1, got 1
Prefix check for (test, testing): expected 1, got 1
Prefix check for (abc, ab): expected 0, got 0
Prefix check for (, anything): expected 1, got 1
Prefix check for (non, ): expected 0, got 0
Prefix check for (prefix, pre): expected 0, got 0
Prefix check for (test, Test): expected 0, got 0
```

Q7

a) Equivalence Classes:

1. Valid Triangle (General):
   - Sides form a triangle (the sum of any two sides is greater than the third).
2. Equivalence Class 1 (EC1): Valid triangle where `a == b == c` (Equilateral triangle).

   Equivalence Class 2 (EC2): Valid triangle where `a == b ≠ c` or `a ≠ b == c` or `a == c ≠ b` (Isosceles triangle).

   Equivalence Class 3 (EC3): Valid triangle where `a ≠ b ≠ c` (Scalene triangle).

   Equivalence Class 4 (EC4): Valid right-angle triangle where $A^2 + B^2 = C^2$ (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 <= c` or `a + c <= b` or `b + c <= a`.

Equivalence Class 6 (EC6): Invalid triangle where one or more sides are zero or negative.

b) Test Case for equivalence Class

Test Case No.	Input (a, b, c)	Expected Output	Equivalence Class Covered
TC1	(3, 3, 3)	EQUILATERAL	EC1
TC2	(4, 4, 2)	ISOSCELES	EC2
TC3	(3, 4, 5)	SCALENE	EC3
TC4	(6, 8, 10)	SCALENE (Right-Angle)	EC4
TC5	(1, 2, 3)	INVALID	EC5
TC6	(1, 1, 2)	INVALID	EC5
TC7	(0, 2, 3)	INVALID	EC6
TC8	(-1, 2, 3)	INVALID	EC6
TC9	(3.0, 3.0, 3.0)	EQUILATERAL	EC1
TC10	(5.0, 5.0, 7.0)	ISOSCELES	EC2
TC11	(4.2, 3.0, 5.0)	SCALENE	EC3
TC12	(6.0, 8.0, 10.0)	RIGHT-ANGLE	EC4
TC13	(1.0, 2.0, 3.0)	INVALID	EC5

TC14	`(0.0, 2.0, 2.0)`	`INVALID`	EC6
TC15	`(-3.0, 4.0, 5.0)`	`INVALID`	EC6

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

Test Case No.	Input (`A, B, C`)	Expected Output	Explanation
TC16	`(1.0, 1.0, 2.0)`	`INVALID`	A + B = C (invalid boundary)
TC17	`(2.0, 3.0, 4.9)`	`SCALENE`	A + B > C (valid boundary)
TC18	`(2.0, 3.0, 5.0)`	`INVALID`	A + B = C (invalid boundary)

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

Test Case No.	Input (`A, B, C`)	Expected Output	Explanation
TC19	`(3.0, 4.0, 3.0)`	`ISOSCELES`	A = C (valid boundary)
TC20	`(3.0, 5.0, 3.0)`	`ISOSCELES`	A = C (valid boundary)
TC21	`(3.0, 3.0, 5.0)`	`ISOSCELES`	A = B (valid boundary)

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

Test Case No.	Input (`A, B, C`)	Expected Output	Explanation
TC22	`(5.0, 5.0, 5.0)`	`EQUILATERAL`	A = B = C (valid boundary)

TC23	`(6.0, 6.0, 6.0)`	`EQUILATERAL`	A = B = C (valid boundary)
TC24	`(6.1, 6.1, 6.1)`	`EQUILATERAL`	A = 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	`(3.0, 4.0, 5.0)`	`RIGHT-ANGLE`	$A^2 + B^2 = C^2$ (valid boundary)
TC26	`(5.0, 12.0, 13.0)`	`RIGHT-ANGLE`	$A^2 + B^2 = C^2$ (valid boundary)
TC27	`(8.0, 15.0, 17.0)`	`RIGHT-ANGLE`	$A^2 + B^2 = C^2$ (valid boundary)

g) Non-Triangle Case Boundaries:

Test Case No.	Input `(A, B, C)`	Expected Output	Explanation
TC28	`(1.0, 1.0, 2.0)`	`INVALID`	A + B = C (invalid)
TC29	`(2.0, 3.0, 6.0)`	`INVALID`	A + B < C (invalid)

h) Non-Positive Input Test Cases:

Test Case No.	Input `(A, B, C)`	Expected Output	Explanation
TC30	`(0.0, 5.0, 7.0)`	`INVALID`	Non-positive side length (A=0)
TC31	`(5.0, 0.0, 7.0)`	`INVALID`	Non-positive side length (B=0)

TC32	`(-3.0, 4.0, 5.0)`	`INVALID`	Negative side length (A=-3)