IT314: Software engineering
Lab 8

202201151
Pandavadara Sahil Rasikbhai

Question 1:

1. Program Specification

Input: A date represented as a triple of (day, month, year)

Input Ranges:

- Month: 1 to 12
- Day: 1 to 31
- Year: 1900 to 2015

Output: The previous date or the message "Invalid date"

2. Test Suite

2.1 Equivalence Partitioning

Valid Partitions:

- **Normal Days:** Regular days that are not at the end of a month or year.
- **Month End:** The last day of any month that is not December 31.
- **Year End:** December 31.
- **Leap Year:** February 29.

Invalid Partitions:

- **Invalid Month:** Values less than 1 or greater than 12.
- **Invalid Day:** Values less than 1 or greater than the maximum days in the respective month.
- **Invalid Year:** Values less than 1900 or greater than 2015.
- **Specific Month Invalid Days:** For instance, February 30 is invalid.

2.2 Boundary Value Analysis

- **First Day of Year:** January 1 of any valid year (YYYY).
- **Last Day of Year:** December 31 of any valid year (YYYY).
- **First Day of Month:** Day 1 for any valid month (MM).
- **Last Day of Month:**
    - Day 30 or 31 for months with those maximum days.
    - Day 28 or 29 for February, depending on whether it's a leap year.
- **Minimum Valid Year:** 1900.
- **Maximum Valid Year:** 2015.

2.3 Test Cases:

Tester Action	Input Data	Expected Outcome	Remarks
a, b, c	An Error message	Invalid input format	
	15, 6, 2000	14, 6, 2000	Normal day
	1, 7, 2010	30, 6, 2010	Month end
	1, 1, 2005	31, 12, 2004	Year end
	1, 3, 2000	29, 2, 2000	Leap year
	1, 3, 2001	28, 2, 2001	Non-leap year
	0, 6, 2000	Invalid date	Invalid day (too low)
	32, 6, 2000	Invalid date	Invalid day (too high)
	15, 0, 2000	Invalid date	Invalid month (too low)
	15, 13, 2000	Invalid date	Invalid month (too high)
	15, 6, 1899	Invalid date	Invalid year (too low)
	15, 6, 2016	Invalid date	Invalid year (too high)
	31, 4, 2000	Invalid date	Invalid day for April

Tester Action	Input Data	Expected Outcome	Remarks
	29, 2, 2001	Invalid date	Invalid day for February in non-leap year
	1, 1, 1900	31, 12, 1899	Boundary: Minimum valid year - 1
	31, 12, 2015	30, 12, 2015	Boundary: Maximum valid year
	1, 1, 2000	31, 12, 1999	Boundary: First day of year
	31, 12, 2000	30, 12, 2000	Boundary: Last day of year
	1, 5, 2000	30, 4, 2000	Boundary: First day of month
	31, 5, 2000	30, 5, 2000	Boundary: Last day of 31-day month
	30, 4, 2000	29, 4, 2000	Boundary: Last day of 30-day month
	29, 2, 2000	28, 2, 2000	Boundary: Last day of February in leap year
	28, 2, 2001	27, 2, 2001	Boundary: Last day of February in non-leap year

C++ code:

```
#include <iostream>

#include <vector>

#include <string>
```

```
using namespace std;

bool isLeapYear(int year) {

    return (year % 4 == 0 && (year % 100 != 0 || year % 400 == 0));

}

int daysInMonth(int month, int year) {

    vector<int> days = {31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};

    if (month == 2 && isLeapYear(year)) {

        return 29;

    }

    return days[month - 1];

}



string previousDate(int day, int month, int year) {

    if (!(1 <= month && month <= 12 && 1900 <= year && year <= 2015)) {

        return "Invalid date";

    }

    int maxDays = daysInMonth(month, year);

    if (!(1 <= day && day <= maxDays)) {

        return "Invalid date";

    }

    if (day > 1) {

        return to_string(day - 1) + ", " + to_string(month) + ", " +
to_string(year);

    } else if (month > 1) {
```

```
        int prevMonth = month - 1;

        return to_string(daysInMonth(prevMonth, year)) + ", " +
to_string(prevMonth) + ", " + to_string(year);

    } else {

        return "31, 12, " + to_string(year - 1);

    }

}

void runTests() {

    vector<pair<vector<int>, string>> testCases = {

        {{15, 6, 2000}, "14, 6, 2000"},

        {{1, 7, 2010}, "30, 6, 2010"},

        {{31, 12, 2004}, "30, 12, 2004"},

        {{1, 3, 2000}, "29, 2, 2000"},

        {{1, 3, 2001}, "28, 2, 2001"},

        {{30, 2, 2000}, "Invalid date"},

        {{32, 6, 2000}, "Invalid date"},

        {{15, 8, 1800}, "Invalid date"},

        {{6, 6, 1899}, "Invalid date"},

        {{14, 5, 2016}, "Invalid date"},

        {{31, 4, 2000}, "Invalid date"},

        {{1, 1, 2000}, "31, 12, 1999"},

        {{31, 12, 2000}, "30, 12, 2000"},

        {{30, 5, 2000}, "29, 5, 2000"},

        {{28, 2, 2000}, "27, 2, 2000"},
```

```
        {{29, 2, 2000}, "28, 2, 2000"},
        {{28, 2, 2001}, "27, 2, 2001"}
    };
    for (int i = 0; i < testCases.size(); i++) {
        vector<int> input = testCases[i].first;
        string expected = testCases[i].second;
        string result = previousDate(input[0], input[1], input[2]);
        cout << "Test " << i + 1 << ": " << ((result == expected) ? "PASS"
: "FAIL") << endl;
        cout << "Input: " << input[0] << ", " << input[1] << ", " <<
input[2] << endl;
        cout << "Expected: " << expected << endl;
        cout << "Actual: " << result << endl;
        cout << endl;
    }
}


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

Question: 2

Problem 1: code

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

Problem 1: Equivalence Partitioning

Input Data	**Expected Outcome**
5, {1, 2, 3}	-1
2, {1, 2, 3}	1
-1, {-1, 0, 1}	0
1, {}	-1
4, {4}	0
1, {1, 2, 3}	0
3, {1, 2, 3}	2
null, {1, 2, 3}	An Error message
{1, 2, 3}, null	An Error message

Problem 1: Boundary Value Analysis

Input Data	**Expected Outcome**
5, {}	-1
-2147483648, {-2147483648, 0, 2147483647}	0

Input Data	**Expected Outcome**
2147483647, {-2147483648, 0, 2147483647}	2
1, {1, 2}	0
2, {1, 2}	1
4, {1, 2, 3}	-1
5, null	An Error message
{1, 2, 3}, {}	An Error message

Modified code: -

```
#include <iostream>

#include <cmath>

using namespace std;




int findIndex(int target, const int arr[], int length) {

    for (int i = 0; i < length; i++) {

        if (arr[i] == target) {

            return i;

        }

    }

    return -1;

}



int main() {

    // Define datasets

    int dataSet1[] = {10, 20, 30, 40, 50};

    int dataSet2[] = {};

    int dataSet3[] = {-10, -20, -30};
```

```
    // Run test cases
    cout << "Test 1: " << findIndex(30, dataSet1, 5) << endl;
    cout << "Test 2: " << findIndex(60, dataSet1, 5) << endl;
    cout << "Test 3: " << findIndex(30, dataSet2, 0) << endl;
    cout << "Test 4: " << findIndex(-20, dataSet3, 3) << endl;
    cout << "Test 5: " << findIndex(10, dataSet1, 1) << endl;
    cout << "Test 6: " << findIndex(10, dataSet1, 5) << endl;
    cout << "Test 7: " << findIndex(50, dataSet1, 5) << endl;
    cout << "Test 8: " << findIndex(20, dataSet2, 0) << endl;
    cout << "Test 9: " << findIndex(60, dataSet1, 5) << endl;


    return 0;
}
```

Problem 2: 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);
}
```

Problem 2: Equivalence Partitioning

Input Data	**Expected Outcome**
5, {1, 2, 3}	0
2, {1, 2, 3}	1
-1, {-1, 0, 1}	1
1, {}	0
4, {4, 4, 4}	3
1, {1, 2, 3, 1, 1}	3
3, {1, 2, 3, 3, 3, 3}	4
null, {1, 2, 3}	An Error message
{1, 2, 3}, null	An Error message

Problem 2: Boundary Value Analysis

Input Data	**Expected Outcome**
5, {}	0
-2147483648, {-2147483648, 0, 2147483647}	1
2147483647, {-2147483648, 0, 2147483647}	1
1, {1, 2}	1
2, {1, 2, 2}	2
4, {1, 2, 3}	0
5, null	An Error message

Input Data	Expected Outcome
{1, 2, 3}, {}	An Error message

Modified code:-

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

int countOccurrences(int target, const int elements[], int length) {
    int count = 0;
    for (int i = 0; i < length; i++) {
        if (elements[i] == target)
            count++;
    }
    return count;
}

int main() {
    int array1[] = {1, 2, 1, 4, 1};
    int array2[] = {};
    int array3[] = {-1, -2, -1};
    int array4[] = {2};
    int array5[] = {1};

    cout << "Test 1: " << countOccurrences(1, array1, 5) << endl;
    cout << "Test 2: " << countOccurrences(6, array1, 5) << endl;
    cout << "Test 3: " << countOccurrences(3, array2, 0) << endl;
    cout << "Test 4: " << countOccurrences(-1, array3, 3) << endl;
    cout << "Test 5: " << countOccurrences(2, array4, 1) << endl;
    cout << "Test 6: " << countOccurrences(2, array5, 1) << endl;
    cout << "Test 7: " << countOccurrences(1, array1, 5) << endl;
    cout << "Test 8: " << countOccurrences(3, array1, 5) << endl;
    cout << "Test 9: " << countOccurrences(2, array2, 0) << endl;
    cout << "Test 10: " << countOccurrences(4, array1, 5) << endl;

    return 0;
```

```
}
```

Problem 3: code

```
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);
}
```

Problem 3: Equivalence Partitioning

Input Data	**Expected Outcome**
5, {1, 2, 3}	-1
2, {1, 2, 3}	1
1, {1, 2, 3}	0
3, {1, 2, 3}	2

Input Data	Expected Outcome
4, {1, 4, 6, 8}	1
0, {0, 1, 2, 3}	0
100, {10, 20, 30, 100}	3
null, {1, 2, 3}	An Error message
{1, 2, 3}, null	An Error message

Problem 3: Boundary Value Analysis

Input Data	Expected Outcome
5, {}	-1
-2147483648, {-2147483648, 0, 2147483647}	0
2147483647, {-2147483648, 0, 2147483647}	2
1, {1, 2}	0
2, {1, 2}	1
4, {1, 2, 3}	-1
5, null	An Error message
{1, 2, 3}, {}	An Error message

Modified code:

```
#include <iostream>

#include <vector>

using namespace std;
```

```
int searchBinary(const vector<int> &arr, int val) {
    int low = 0;
    int high = arr.size() - 1;

    while (low <= high) {
        int middle = low + (high - low) / 2;
        if (arr[middle] == val) {
            return middle;
        } else if (arr[middle] < val) {
            low = middle + 1;
        } else {
            high = middle - 1;
        }
    }
    return -1;
}

int main() {
    vector<int> array1 = {};
    vector<int> array2 = {1, 2, 3, 5, 6};
    vector<int> array3 = {1, 2, 3, 4, 6};
    vector<int> array4 = {5};
    vector<int> array5 = {3};
```

```
    cout << searchBinary(array1, 5) << endl;
    cout << searchBinary(array2, 5) << endl;
    cout << searchBinary(array3, 5) << endl;
    cout << searchBinary(array4, 5) << endl;
    cout << searchBinary(array5, 5) << endl;

    return 0;
}
```

Problem 4: 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);

}
```

Problem 4: Equivalence Partitioning

Input Data	Expected Outcome
3, 3, 3	EQUILATERAL (0)
3, 3, 2	ISOSCELES (1)
3, 4, 5	SCALENE (2)
1, 2, 3	INVALID (3)
1, 1, 2	INVALID (3)
5, 1, 1	INVALID (3)
2, 2, 3	ISOSCELES (1)
0, 1, 1	An Error message
1, 0, 1	An Error message

Problem 4: Boundary Value Analysis

Input Data	Expected Outcome
1, 1, 1	EQUILATERAL (0)
1, 1, 2	INVALID (3)
2, 2, 4	INVALID (3)
2, 3, 5	INVALID (3)
3, 4, 7	INVALID (3)

Input Data	Expected Outcome
1, 2, 2	ISOSCELES (1)
1, 2, 3	INVALID (3)
0, 1, 1	An Error message
1, 1, 0	An Error message

Modified code:

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

const char* getTriangleType(int side1, int side2, int side3) {
    if (side1 <= 0 || side2 <= 0 || side3 <= 0 || side1 + side2 <= side3
|| side1 + side3 <= side2 || side2 + side3 <= side1) {
        return "Invalid";
    }
    if (side1 == side2 && side2 == side3) {
        return "Equilateral";
    }
    if (side1 == side2 || side2 == side3 || side1 == side3) {
        return "Isosceles";
    }
    return "Scalene";
}

int main() {
    cout << getTriangleType(3, 3, 3) << endl;
    cout << getTriangleType(4, 4, 5) << endl;
    cout << getTriangleType(3, 4, 5) << endl;
    cout << getTriangleType(1, 2, 3) << endl;
    cout << getTriangleType(-1, 2, 3) << endl;
    cout << getTriangleType(0, 2, 2) << endl;
    cout << getTriangleType(1, 1, 1) << endl;
    cout << getTriangleType(1, 1, 2) << endl;
    cout << getTriangleType(-1, 1, 1) << endl;
    cout << getTriangleType(0, 1, 1) << endl;
    cout << getTriangleType(2, 2, 2) << endl;
```

```
    return 0;
}
```

Problem 5: 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;
}
```

Problem 5: Equivalence Partitioning

Input Data	**Expected Outcome**
"pre", "prefix"	true
"pre", "postfix"	false
"prefix", "pre"	false
"test", "test"	true
"", "anything"	true
"anything", ""	false
"pre", "preparation"	true
null, "prefix"	An Error message
"prefix", null	An Error message

Modified code:

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

bool isPrefix(string str1, string str2) {
    if (str1.length() > str2.length()) {
        return false;
    }
    for (int i = 0; i < str1.length(); i++) {
        if (str1[i] != str2[i]) {
            return false;
        }
    }
    return true;
}

int main() {
    cout << "TC1: " << (isPrefix("abcdef", "abc") ? "true" : "false") <<
endl;
    cout << "TC2: " << (isPrefix("abc", "abcdef") ? "true" : "false") <<
endl;
    cout << "TC3: " << (isPrefix("xyz", "abcdef") ? "true" : "false") <<
endl;
    cout << "TC4: " << (isPrefix("", "abcdef") ? "true" : "false") <<
endl;
    cout << "TC5: " << (isPrefix("abc", "") ? "true" : "false") << endl;
    cout << "TC6: " << (isPrefix("a", "") ? "true" : "false") << endl;
    cout << "TC7: " << (isPrefix("abcdef", "abcdef") ? "true" : "false")
<< endl;
    cout << "TC8: " << (isPrefix("abc", "abc") ? "true" : "false") <<
endl;
    cout << "TC9: " << (isPrefix("", "") ? "true" : "false") << endl;
    return 0;
}
```

Problem 5: Boundary Value Analysis

Input Data	Expected Outcome
"test", ""	false
"a", "a"	true
"a", "b"	false
"", ""	true
"start", "startmiddle"	true
"longprefix", "short"	false
"short", "longprefix"	true
null, "anything"	An Error message
"anything", null	An Error message

Problem 6:

a) Identify the Equivalence Classes

- Equilateral Triangle: All three sides are equal.
- Isosceles Triangle: Exactly two sides are equal.
- Scalene Triangle: No sides are equal.
- Right-Angled Triangle: Satisfies a2+b2=c2.
- Invalid Triangle: Does not satisfy the triangle inequality a+b>c.
- Non-positive Input: One or more sides are non-positive.

B. Equivalence Partitioning

Input Data	Expected Outcome	Equivalence Class
3.0, 3.0, 3.0	Equilateral	Equilateral Triangle
3.0, 3.0, 2.0	Isosceles	Isosceles Triangle
3.0, 4.0, 5.0	Scalene	Scalene Triangle
3.0, 4.0, 0.0	Invalid	Invalid Triangle
0.0, 0.0, 0.0	Invalid	Non-positive Input
5.0, 1.0, 1.0	Invalid	Invalid Triangle
3.0, 4.0, 6.0	Scalene	Scalene Triangle

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

Input Data	Expected Outcome
2.0, 2.0, 3.99	Scalene
2.0, 2.0, 4.0	Invalid
2.0, 2.0, 4.01	Invalid

D. Boundary Condition: A = C (Isosceles Triangle)

Input Data	Expected Outcome
3.0, 4.0, 3.0	Isosceles
3.0, 3.0, 3.0	Equilateral
3.0, 3.0, 4.0	Isosceles

E. Boundary Condition: A = B = C (Equilateral Triangle)

Input Data	Expected Outcome
3.0, 3.0, 3.0	Equilateral
1.0, 1.0, 1.0	Equilateral
2.5, 2.5, 2.5	Equilateral

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

Input Data	Expected Outcome
3.0, 4.0, 5.0	Right Angled
6.0, 8.0, 10.0	Right Angled
5.0, 12.0, 13.0	Right Angled

G. Non-Triangle Case

Input Data	Expected Outcome
1.0, 2.0, 3.0	Invalid
1.0, 2.0, 4.0	Invalid
1.0, 1.0, 2.0	Invalid

H. Non-Positive Input

Input Data	Expected Outcome
0.0, 1.0, 1.0	Invalid
-1.0, 1.0, 1.0	Invalid
1.0, 0.0, 1.0	Invalid

Modified code:

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

const char *determineTriangleType(float sideA, float sideB, float sideC) {
    if (sideA <= 0 || sideB <= 0 || sideC <= 0 || sideA + sideB <= sideC
|| sideA + sideC <= sideB || sideB + sideC <= sideA) {
        return "Invalid";
    }
    if (fabs(pow(sideA, 2) + pow(sideB, 2) - pow(sideC, 2)) < 1e-6 ||
fabs(pow(sideA, 2) + pow(sideC, 2) - pow(sideB, 2)) < 1e-6 ||
fabs(pow(sideB, 2) + pow(sideC, 2) - pow(sideA, 2)) < 1e-6) {
        return "Right-angled";
    }
    if (sideA == sideB && sideB == sideC) {
        return "Equilateral";
    }
    if (sideA == sideB || sideB == sideC || sideA == sideC) {
        return "Isosceles";
    }
    return "Scalene";
}

int main() {
    cout << "Test 1: " << determineTriangleType(3.0, 3.0, 3.0) << endl;
    cout << "Test 2: " << determineTriangleType(4.0, 4.0, 5.0) << endl;
    cout << "Test 3: " << determineTriangleType(3.0, 4.0, 5.0) << endl;
    cout << "Test 4: " << determineTriangleType(3.0, 4.0, 6.0) << endl;
    cout << "Test 5: " << determineTriangleType(-1.0, 2.0, 3.0) << endl;
    cout << "Test 6: " << determineTriangleType(0.0, 2.0, 2.0) << endl;
    cout << "Test 7: " << determineTriangleType(5.0, 12.0, 13.0) << endl;
    return 0;
}
```