



IT-314_SOFTWAREENGINEERING
LAB-8 Functional Testing (BlackBox)

ID:202201084

NAME: Sujal Manavadariya

Q1: Determining the Previous Date

Problem Overview:

You are tasked with designing a test suite for a program that calculates the previous date given a day, month, and year. Inputs should fall within the following ranges:

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

The program will output either a valid previous date or an error indicating an invalid date.

- **Equivalence Class Partitioning:**

We divide the input space into valid and invalid partitions to reduce the number of test cases while maintaining good coverage.

- ❖ **Equivalence Classes:**

- ValidDay: $1 \leq \text{day} \leq 31$
- Invalid Day: $\text{day} < 1$ or $\text{day} > 31$
- ValidMonth: $1 \leq \text{month} \leq 12$
- Invalid Month: $\text{month} < 1$ or $\text{month} > 12$
- ValidYear: $1900 \leq \text{year} \leq 2015$
- InvalidYear: $\text{year} < 1900$ or $\text{year} > 2015$

TestCasesforEquivalencePartitioning:

TestCase	Day	Month	Year	Expected Output	Equivalence Class
TC1	15	6	2010	Previous Date	AllValidInputs
TC2	32	6	2010	Invalid Date	Invalid Day
TC3	0	6	2010	Invalid Date	Invalid Day
TC4	15	13	2010	Invalid Date	Invalid Month
TC5	15	0	2010	Invalid Date	Invalid Month
TC6	15	6	1899	Invalid Date	InvalidYear
TC7	15	6	2016	Invalid Date	InvalidYear

BoundaryTestCases:

Test Case	Day	Month	Year	Expected Output	Boundary Condition
TC8	1	1	1900	Previous Date	LowerBoundofDay,Month,Year
TC9	31	12	2015	Previous Date	UpperBoundofDay,Month,Year
TC10	2	1	1900	Previous Date	Just Above Lower Bound ofDay
TC11	30	12	2015	Previous Date	Just Below Upper Bound ofDay

Program :

```
#include
<iostream>#include
<string>
```

```

bool isLeapYear(int year) {

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

}

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

    if (year < 1900 || year > 2015 || month < 1 || month > 12 || day < 1
|| day > 31) {

        return "Error: Invalid date";

    }

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

    if (isLeapYear(year))        daysInMonth[1] = 29;

    if (day > daysInMonth[month - 1])        return "Error: Invalid date";if (day > 1)

        day--;

    else {

        month--;

        if (month < 1) {month

            = 12;year--;

        }

        day = daysInMonth[month - 1];

    }

    return to_string(day) + "/" + to_string(month) + "/" +to_string(year);

}

int main() {

```

```

    cout << previousDate(15, 7, 2000) << endl;           // Expected: 14/7/2000

    cout << previousDate(1, 1, 1900) << endl;           // Expected: 31/12/1899

    cout << previousDate(32, 7, 2000) << endl;         // Expected: Error:
Invalid date

    return 0;
}

```

Q2:ProgramsTesting

P1: linearSearch(int v, int a[])

- **Functionality:** Searches for the first occurrence of value *v* in array *a*. Returns the index if found, otherwise returns -1.

Equivalence Classes:

1. Value is present: *v* is found in *a*.
2. Value is not present: *v* is not found in *a*.
3. Empty array: *a* has no elements.
4. Single-element array: *a* contains exactly one element.
5. Duplicate values: *a* contains multiple occurrences of *v*.

Boundary Conditions:

1. First element: *v* is the first element of *a*.
2. Last element: *v* is the last element of *a*.
3. Array length 1: *a* contains only one element.

TestCases:

Test Case	v	a	Expected Output	EquivalenceClass	BoundaryCondition
TC1	5	[1,2,3,4,5]	4	ValuePresent	Lastelement
TC2	7	[1,2,3,4,5]	-1	ValueNotPresent	-
TC3	3	[]	-1	EmptyArray	-
TC4	1	[1]	0	SingleElementArray,V aluePresent	First element and Singleelement
TC5	2	[1,2,2,3,4,5]	1	DuplicateValuesPresent	Firstoccurrence

P2: countItem(int v, int a[])

- Functionality: Counts how many times value v appears in array a[].

Equivalence Classes:

1. Multiple occurrences: v appears multiple times in a[].
2. Singleoccurrence: v appears once in a[].
3. Nooccurrences: v does not appear in a[].
4. Empty array: a[] has no elements.
5. Single-element array: a[] contains exactly one element.

Boundary Conditions:

1. Array length 1: `a[]` contains only one element.
2. Array with duplicates: `a[]` contains repeated values of `v`.

TestCases:

TestCase	v	a	Expected Output	EquivalenceClass	Boundary Condition
TC1	5	[1,2,5,5,5]	3	MultipleOccurrences	-
TC2	5	[1,2,3,4,5]	1	SingleOccurrence	-
TC3	7	[1,2,3,4,5]	0	NoOccurrences	-
TC4	5	[]	0	EmptyArray	-
TC5	5	[5]	1	SingleOccurrenceinSingleElementArray	Singleelement

P3: `binarySearch(int v, int a[])`

- **Functionality:** Searches for value `v` in a sorted array `a[]`. Returns the index if found, otherwise returns -1.

Equivalence Classes:

1. Value is present: `v` exists in `a[]`.
2. Value is not present: `v` does not exist in `a[]`.
3. Empty array: `a[]` has no elements.
4. Single-element array: `a[]` contains exactly one element.
5. Value is smaller than all elements: `v` is less than all elements in `a[]`.
6. Value is larger than all elements: `v` is greater than all elements in `a[]`.

Boundary Conditions:

1. First element: vis the first element of a[].
2. Middle element: vis the middle element of a[].
3. Last element: vis the last element of a[].
4. Single element array: a[]contains one element.

TestCases:

Test Case	v	a	Expected Output	EquivalenceClass	Boundary Condition
TC1	10	[1,5,10,15,20]	2	ValuePresent	Middleelement
TC2	1	[1,5,10,15,20]	0	ValuePresent	Firstelement
TC3	20	[1,5,10,15,20]	4	ValuePresent	Lastelement
TC4	7	[1,5,10,15,20]	-1	ValueNotPresent	-
TC5	3	[]	-1	EmptyArray	-
TC6	2	[1]	-1	SingleElementArray,ValueNotPresent	Single elementarray

P4: triangle(int a, int b, int c)

- **Functionality:** Classifies a triangle based on the side lengths a , b , and c . It returns `equilateral`, `isosceles`, `scalene`, or `invalid`.

Equivalence Classes:

1. `EquilateralTriangle`: All sides are equal ($a=b=c$).
2. `IsoscelesTriangle`: Two sides are equal ($a=b, a \neq c$).
3. `ScaleneTriangle`: No sides are equal ($a \neq b \neq c$).
4. `InvalidTriangle`: Triangle inequality is violated ($a+b \leq c$ or similar).

Boundary Conditions:

1. Invalid sides: Check boundary conditions where sides sum exactly to or just greater than the third side.
2. Edge cases for equilateral and isosceles: $a = b = c$, $a = b$.

Test Cases:

TestCase	a	b	c	ExpectedOutput	EquivalenceClass	BoundaryCondition
TC1	3	3	3	Equilateral	EquilateralTriangle	$a=b=c$
TC2	3	3	5	Isosceles	IsoscelesTriangle	$a=b$
TC3	3	4	5	Scalene	ScaleneTriangle	$a \neq b \neq c$
TC4	1	2	3	Invalid	InvalidTriangle($a+b=c$)	$a+b=c$

P5: `prefix(String s1, String s2)`

- **Functionality:** Checks if $s1$ is a prefix of $s2$. Returns `true` if $s1$ is a prefix of $s2$, otherwise returns `false`.

Equivalence Classes:

1. s1 is a prefix of s2: s1 appears at the start of s2.
2. s1 is not a prefix of s2: s1 does not appear at the start of s2.
3. s1 is longer than s2: $s1.length() > s2.length()$.
4. s1 and s2 are equal: Both strings are exactly the same.

Boundary Conditions:

1. Empty strings: One or both strings are empty.
2. Single-character strings: Check with strings of length 1.

Test Cases:

TestCase	s1	s2	Expected Output	EquivalenceClass	BoundaryCondition
TC1	"abc"	"abcdef"	true	s1 is a prefix of s2	-
TC2	"xyz"	"abcdef"	false	s1 is not a prefix of s2	-
TC3	"abcdefg"	"abc"	false	s1 is longer than s2	-
TC4	""	"abcdef"	true	s1 is empty	Empty s1
TC5	"a"	"a"	true	s1 and s2 are equal	Single character strings equal

P6: Floating-point Triangle Classification

- **Functionality:** This program classifies a triangle with floating-point side lengths and also determines if it is a right-angled triangle (based on the Pythagorean theorem).

Equivalence Classes:

1. EquilateralTriangle: $A=B=C$.
2. IsoscelesTriangle: $A=B \neq C$.
3. ScaleneTriangle: $A \neq B \neq C$.
4. Right-angledTriangle: $A^2 + B^2 = C^2$ (or similar).
5. InvalidTriangle: Triangle inequality is violated ($A+B \leq C$).

Boundary Conditions:

1. Check boundaries for triangle inequality: $A + B = C$.
2. Right-angle triangle condition: Tested edge cases where $A^2 + B^2 = C^2$.

Test Cases:

TestCase	A	B	C	ExpectedOutput	EquivalenceClass	BoundaryCondition
TC1	3.0	3.0	3.0	Equilateral	EquilateralTriangle	-
TC2	3.0	3.0	4.0	Isosceles	IsoscelesTriangle	-
TC3	3.0	4.0	5.0	Right-angled	Right-angledTriangle	$A^2 + B^2 = C^2$
TC4	2.0	3.0	4.0	Scalene	ScaleneTriangle	-
TC5	1.0	2.0	3.0	Invalid	InvalidTriangle($A+B=C$)	$A+B=C$