Lab 9 - Group-27

Q1. Consider a program for determining the previous date. Its input is triple of day, month and year with the following ranges 1 <= month <= 12, 1 <= day <= 31, 1900 <= year <= 2015.The possible output dates would be previous date or invalid date. Design the equivalence class test cases?

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. **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.**

Ans.

Equivalence Partitioning Technique :

- ➔ 1<= Day <= 31 and 1<=Month<=12 and 1900<=year<=2015
  - ◆ Any Day < 1 => Invalid
  - ◆ Day between 1 and 31 (Including both 1 and 31) => Valid
  - ◆ Any Day > 31 => Invalid
  - ◆ Any Month < 1 => Invalid
  - ◆ Month Between 1 and 12 (Including both 1 and 12) => Valid
  - ◆ Any Month > 12 => Invalid
  - ◆ Any Year < 1900 => Invalid
  - ◆ Year Between 1900 and 2015 (Including Both) => Valid
  - ◆ Year > 2015 => Invalid
- ➔ If any one field of this three is Invalid then whole Date is invalid.

➔ Set of Value for Date = {0,10,50}
➔ Set of Value for Month = {0,5,15}
➔ Set of value for Year = {1800,2000,2100}
➔ Total possible combination = 3*3*3 = 27

Test Case and Expected Result:

Test Case Number	Day	Month	Year	Result
1	0	0	1800	Invalid
2	0	0	2000	Invalid
3	0	0	2100	Invalid
4	0	5	1800	Invalid
5	0	5	2000	Invalid
6	0	5	2100	Invalid
7	0	15	1800	Invalid
8	0	15	2000	Invalid

9	0	15	2100	Invalid
10	10	0	1800	Invalid
11	10	0	2000	Invalid
12	10	0	2100	Invalid
13	10	5	1800	Invalid
14	10	5	2000	9-5-2000
15	10	5	2100	Invalid
16	10	15	1800	Invalid
17	10	15	2000	Invalid
18	10	15	2100	Invalid
19	50	0	1800	Invalid
20	50	0	2000	Invalid
21	50	0	2100	Invalid

22	50	5	1800	Invalid
23	50	5	2000	Invalid
24	50	5	2100	Invalid
25	50	15	1800	Invalid
26	50	15	2000	Invalid
27	50	15	2100	Invalid

Considering the Date validation (Further validation):

- ➔ Day :
  - ◆ D1 : Day between 1 and 28 (Including both 1 and 28)
  - ◆ D2 : Day = 29
  - ◆ D3 : Day = 30
  - ◆ D4 : Day = 31
- ➔ Month :
  - ◆ M2 : {1,3,5,7,8,10,12} (Month which has 31 days)
  - ◆ M3 : {4,6,9,11} (Month which has 30 days)
  - ◆ M4 : {2} (Month which has 28/29 days)
- ➔ Year :
  - ◆ Y1 : Given year is leap year
  - ◆ Y2 : Given Year is not leap Year

➔ Set of value for Day ={10,29,30,31}
➔ Set of value for Month = {1,4,2}
➔ Set of value for year = {2000, 2022}

Test Case and Expected Result :

Test Case Number	Day	Month	Year	Expected Result
1	10	7	2000	14-1-2000
2	10	7	2011	14-1-2011
3	10	9	2000	14-4-2000
4	10	9	2011	14-4-2011
5	10	2	2000	14-2-2000
6	10	2	2011	14-2-2011
7	29	7	2000	28-1-2000
8	29	7	2011	28-1-2011

9	29	9	2000	28-4-2011
10	29	9	2011	28-4-2011
11	29	2	2000	28-2-2000
12	29	2	2011	Invalid
13	30	7	2000	29-1-2000
14	30	7	2011	29-1-2011
15	30	9	2000	29-4-2000
16	30	9	2011	29-4-2011
17	30	2	2000	Invalid
18	30	2	2011	Invalid
19	31	7	2000	30-1-2000
20	31	7	2011	30-1-2011
21	31	9	2000	Invalid

22	31	9	2011	Invalid
23	31	2	2000	Invalid
24	31	2	2011	Invalid

Boundary Test Cases:

As per boundary cases format value set ({min-1,min, min+1, max-1, max,max+1}) days, months and years sets are as follow:

- ➔ Day : {0,1,2,15,30,31,32}
- ➔ Month : {0,1,2,6,11,12,13}
- ➔ Year : {1899,1900,1901,2000,2014,2015,2016}

And we add 29 to the day's set for checking leap year constraints.

Test Case ID	Day	Month	Year	Expected Output
1	1	6	2000	31-5-2000
2	2	6	2000	1-6-2000
3	30	6	2000	29-6-2000
4	31	6	2000	Invalid input

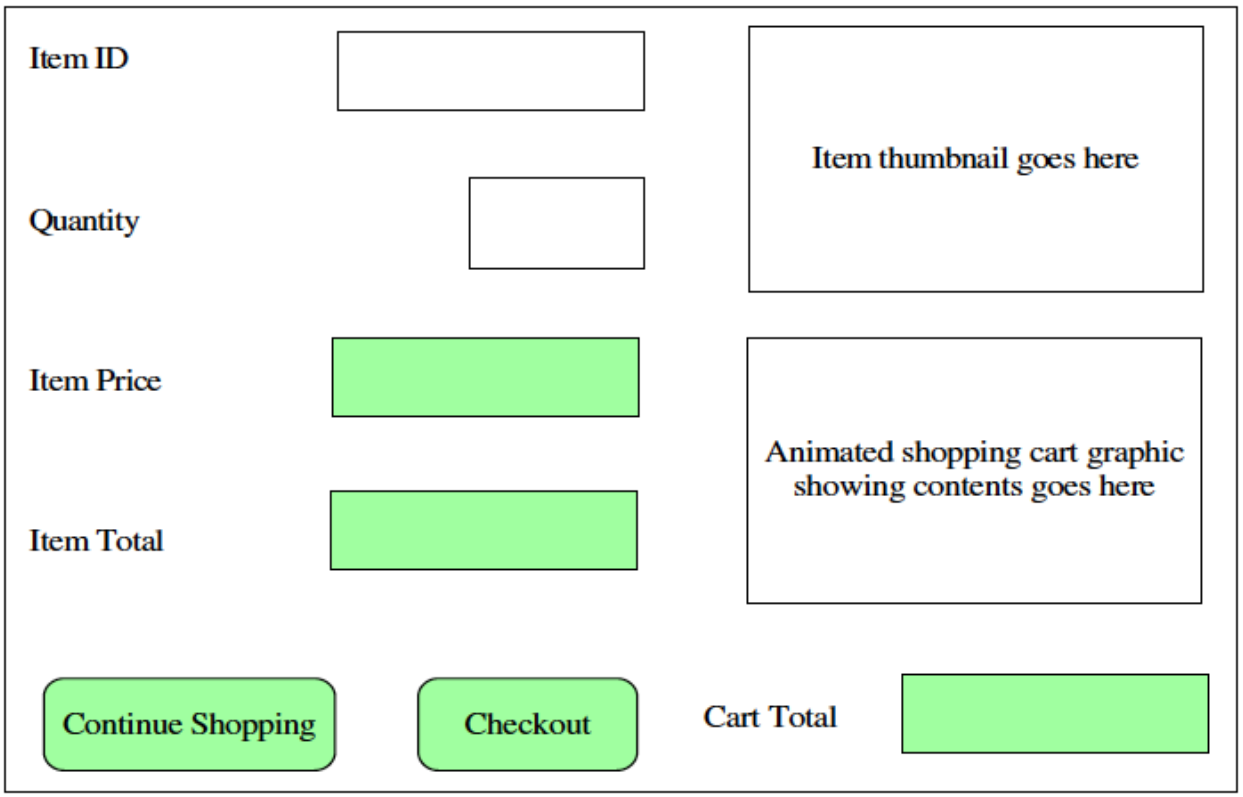
5	15	1	2000	14-1-2000
6	15	2	2000	14-2-2000
7	15	11	2000	14-11-2000
8	15	12	2000	14-12-2000
9	15	6	1900	14-6-1900
10	15	6	1901	14-6-1901
11	15	6	2014	14-6-2014
12	15	6	2015	14-6-2015
13	29	2	2010	Invalid input
14	29	2	2012	28-2-2012

If the value of date is 0 or 32 , or value of month is 0 or 13, or value of year is 1899 or 2016 , then the input is invalid.

```
#include <bits/stdc++.h>
using namespace std;
#define int long long
typedef long long ll;

//Credits  of leap year code gfg
bool isLeapYear(int year)
{
    if (year % 400 == 0)
        return true;
    if (year % 100 == 0)
        return false;
    if (year % 4 == 0)
        return true;
    return false;
}

int main()
{
    int date, month, year;
    cin >> date >> month >> year;

     int days[13] = {0,31,28,31,30,31,30,31,31,30,31,30,31};

    if(isLeapYear(year)) {
          days[2]++;
     }
    if(date < 1 || date > 31) {
        cout << "Invalid Date";
    }
    else if(month < 1 || month > 12) {
        cout << "Invald Month";
    }
    else if(year < 1900 || year > 2015) {
        cout << "Invalid Year";
    }
    else if(date > days[month]) {
        cout << "Invalid date";
    }
    else {
```

```
        date--;
        if(date == 0) {
            month--;
            if(month == 0) {
                month = 12;
                year--;
            }
            date = days[month];
        }
        cout << date << "/" << month << "/" << year << endl;
    }
}
```

Q2. You are testing an e-commerce system that sells products like caps and jackets. The problem is to create functional tests using boundary-value analysis and equivalence class partitioning techniques for the webpage that accepts the orders. A screen prototype for the order-entry web page is shown below.



The system accepts a five-digit numeric item ID number from 00000 to 99999. The system accepts a quantity to be ordered, from 1 to 99. If the user enters a previously ordered item ID and a 0 quantity to be ordered, that item is removed from the shopping cart. Based on these inputs, the system retrieves the item price, calculates the item total (quantity times item price), and adds the item total to the cart total. Due to limits on credit card orders that can be processed, the maximum cart total is $999.99

Ans.

- ➔ Constraints:
  - ◆ ID: 00000 - 99999
  - ◆ Cart total : Maximum $999.99
  - ◆ Quantity : 1 - 99
- ➔ Equivalence class:
  - ◆ ItemID < 00000 (Invalid partition, for any quantity)
  - ◆ ItemID > 99999 (Invalid partition, for any quantity)
  - ◆ Cart Total > $999.99 (Invalid partition, for any ItemID or Quantity)
  - ◆ Quantity < 0 (Invalid partition, for any ItemID)
  - ◆ Quantity > 99 (Invalid partition, for any ItemID)
  - ◆ 00000<= ItemID <= 99999, 0 <= Quantity <= 99, 0 <= Cart total <= $999.99 (valid partition)

Test Case	Inputs	Outputs
1	ItemId: 32014 Quantity: 12	Valid, Cart total will be displayed (here, cart total is less than or equal to $999.99)
2	ItemID: -1	Invalid
3	ItemId: 100000	Invalid
4	ItemID: 12345 (any valid id) Quantity: -50	Invalid

5	ItemID: 12345 (any valid id) Quantity: 120	Invalid
6	ItemID: 12345 (any valid id) Quantity: 90 (Item Price: $200)	Invalid
7	ItemID: 12345 (any valid id) Quantity: 0	Item will be removed if the ItemID was added previously in the list
8	ItemID: 12345 (any valid id) Quantity: 0	Invalid, if the ItemID is not in the list

- ➔ Boundary Value Analysis:
  - ◆ For ID:
    - ID = 00000
    - ID = 00001
    - ID = 32456
    - ID = 99998
    - ID = 99999
  - ◆ For Quantity :
    - Quantity = 1
    - Quantity = 0
    - Quantity = 98
    - Quantity = 99
  - ◆ Max-Cart Total:
    - Cart_total = $0
    - Cart_total = $1.00
    - Cart_total = $998.99
    - Cart_total = $999.99

➔ Test Cases:

Test Case	**Input**	**Output**
ID < 00000	ID = -00001	Error
ID > 99999	ID = 100000	Error
Valid ID	ID = 25534	Add to Cart
Quantity < 0	Quantity = -1	Error
Quantity = 0	ID = 65423 (present in the cart)	Remove item from cart with given ID
Quantity = 0	ID = 32456 (not present in cart)	Error[Item with given ID is not not added previously]
Valid Quantity	Quantity = 56	Add item to cart
Quantity > 99	Quantity = 100	Error
Valid Cart total	ID = 13289 , Quantity = 47	Cart_Total = $ 560
Invalid Cart total	ID = 98546, Quantity = 34	Cart_Total=$3561 (Error because art_total>$999.99)