Name: Varnika Chhawcharia

Student id: 202201269

Lab 8: Functional testing(black box)

Question 1:

Equivalence Partitioning:

Input Month	Input Day Input Year Expected outcome
1	32 2010 error
1	0 2010 error
13	15 2010 error
0	12 2010 error
6	15 1899 error
6	15 2016 error

10 1 2004 9-1-2004 Boundary value analysis:

Input month	Input day Input year Expected outcome
1	31 2010 30-1-2010
1	1 2010 31-12-2009
3	1 2000 29-2-2000
3	1 2009 29-2-2009
5	1 2010 30-4-2010
2	29 2000 28-1-2000
4	30 2010 24-4-2010

Executable code for the above is:

```
#include <iostream>
using namespace std;
bool isLeapYear(int year) {
    if ((year % 400 == 0) || (year % 100 != 0 && year % 4 == 0)) {
        return true;
    }
    return false;
}
string previousDate(int day, int month, int year) {
    int daysInMonth[] = {31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};
    if (isLeapYear(year)) {
        daysInMonth[1] = 29;
    }
    if (year < 1900 || year > 2015 || month < 1 || month > 12 || day < 1 || day
> daysInMonth[month - 1]) {
        return "Invalid Date";
    }
    if (day == 1) {
        if (month == 1) {
            year--;
            month = 12;
            day = 31;
        } else {
            month--;
            day = daysInMonth[month - 1];
        }
    } else {
        day--;
    }
    return "Previous date is " + to_string(day) + "/" + to_string(month) + "/" +
to_string(year);
```

```
}
int main() {
    cout << previousDate(32, 1, 2010) << endl;
    cout << previousDate(0, 1, 2010) << endl;
    cout << previousDate(1, 1, 1900) << endl;
    cout << previousDate(15, 6, 2010) << endl;
    cout << previousDate(1, 3, 2010) << endl;
    cout << previousDate(1, 3, 2000) << endl;
    cout << previousDate(1, 3, 1900) << endl;
    cout << previousDate(29, 2, 2000) << endl;
    cout << previousDate(30, 4, 2010) << endl;
    return 0;
}
```

Question 2:

P1:

Equivalence partitioning:

Input v	Input a[] Expected outcome
3	{1,2,3,4} 2
6	{1,2,3,4,5} -1
1	{ } -1
4	{1,2,3,4,5,6} 3
8	{1,2} -1

Boundary Value Analysis:

Input v	Input a[]	Expected outcome
1	{1}	0

2	{1}	-1
1	{1,2,3,4,5}	0

5	{1,2,3,4,5}	4
1000	{1,2,3,....,1000}	999
1001	{1,2,3,4,.....,1000}	-1
-5	{-10,-5,0,5}	1

P2

Equivalence Partitioning:

Input v	Input a[] Expected outcome
3	{1,2,3,4,3,5} 2
2	{1,2,3,4,5} 1
4	{1,2,3,5} 0
3	{ } 0
-2	{-2,-1,0,1,2} 1

Boundary Value analysis:

Input v	Input a[] Expected outcome
1	{1} 1
2	{1} 0
1	{1,2,3,4,5} 1
1000	{1,2,3,4…..,1000} 1

1001	{1,2,3,4,.....,1000} 0

-5 {-5,-4,-5,10,0} 2

P3

Equivalence partitioning:

Input v	Input a[] Expected outcome
3	{1,2,3,4} 2
6	{1,2,3,4,5} -1
1	{ } -1
4	{1,2,3,4,5,6} 3
8	{1,2} -1

Boundary Value Analysis:

Input v	Input a[]	Expected outcome
1	{1}	0
2	{1}	-1
1	{1,2,3,4,5}	0
5	{1,2,3,4,5}	4
1000	{1,2,3,....,1000}	999
1001	{1,2,3,4,.....,1000}	-1
-5	{-10,-5,0,5}	1

P4

Equivalence Partitioning:

a	b	c Expected outcome

3	3	3 EQUILATERAL
3	3	4 ISOSCELES
2	3	4 SCALENE

1	2	3 INVALID
0	2	3 INVALID
-1	2	3 INVALID

Boundary Value Analysis:

a	b	c Expected Outcome
1	1	1 EQUILATERAL
1	2	2 ISOSCELES
3	4	5 SCALENE
1	2	3 INVALID
1	2	4 INVALID
0	1	2 INVALID
-1	2	3 INVALID

P5:

Equivalence Partitioning:

S1	S2 Expected outcome
abc	abcdef true
abc	abc true

abcd	abc false
abd	abc false
abd	abcde false

Boundary Value Analysis:

S1	S2 Expected outcome
“ “	abc true
abc	“ “ false
a	abc true
abc	a false
a	a true
abc	abx false

P6:

Equivalence partitioning:

a	b	c	Expected outcome
3	3	3	Equilateral
4	4	5	Isosceles
3	4	5	Scalene
5	12	13	Right angle
1	2	3	Invalid

0 5 5 Invalid

Boundary Value Analysis:

a) Boundary condition for Scalene

a	b	c Expected Outcome
1	1	2 invalid

1.1	1	2 Scalene

b) Boundary condition for Isosceles:

a	b	c Expected Outcome
4	4	5 Isosceles
3	3	6 invalid

c) Boundary Condition for Equilateral triangle:

a	b	c Expected Outcome
5	5	5 Equilateral
5	5	5.1 invalid

d) Boundary Condition for Right angle triangle:

a	b	c Expected Outcome
5	12	13 Right angled
2	2	2.68 Right angled

e) Boundary value for non triangle:

a	b	c Expected Outcome

1	2	3 invalid
0	1	2 invalid