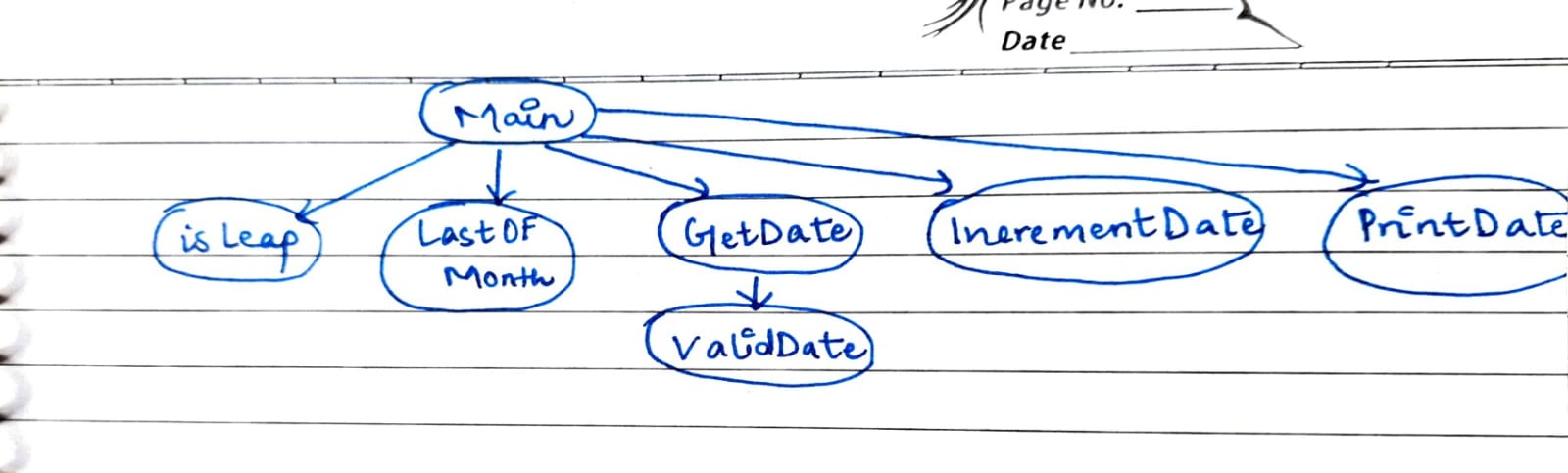
**LAB 9 AND 10**

Write a program that takes a valid date (Day, Month, Year) as input and returns the next valid date according to the Gregorian calendar.

1. #include <stdio.h>
2. #include <stdbool.h>
3. **bool** **isLeap**(int year);
4. int **LastOfMonth**(int month, int year);
5. **bool** **ValidDate**(int d, int m, int y);
6. void **IncrementDate**(int \*d, int \*m, int \*y);
7. void **PrintDate**(int d, int m, int y);
8. void **GetDate**(int \*d, int \*m, int \*y);
9. **bool** **isLeap**(int year) {
10. return (year % 400 == 0) || (year % 4 == 0 && year % 100 != 0);
11. }
12. int **LastOfMonth**(int month, int year) {
13. switch(month) {
14. case 1: return 31;
15. case 2: return (**isLeap**(year) ? 29 : 28);
16. case 3: return 31;
17. case 4: return 30;
18. case 5: return 31;
19. case 6: return 30;
20. case 7: return 31;
21. case 8: return 31;
22. case 9: return 30;
23. case 10: return 31;
24. case 11: return 30;
25. case 12: return 31;
26. default: return -1;
27. }
28. }
29. **bool** **ValidDate**(int d, int m, int y) {
30. if (y < 1900 || y > 2100) return **false**;
31. if (m < 1 || m > 12) return **false**;
32. if (d < 1 || d > **LastOfMonth**(m, y)) return **false**;
33. return **true**;
34. }
35. void **IncrementDate**(int \*d, int \*m, int \*y) {
36. if (\*d < **LastOfMonth**(\*m, \*y)) {
37. (\*d)++;
38. } else {
39. \*d = 1;
40. if (\*m < 12) {
41. (\*m)++;
42. } else {
43. \*m = 1;
44. (\*y)++;
45. }
46. }
47. }
48. void **PrintDate**(int d, int m, int y) {
49. if (**ValidDate**(d, m, y))
50. **printf**("Next Date: %02d/%02d/%d\n", d, m, y);
51. else
52. **printf**("Invalid Date!\n");
53. }
54. void **GetDate**(int \*d, int \*m, int \*y) {
55. **printf**("Enter Date (DD MM YYYY): ");
56. **scanf**("%d %d %d", d, m, y);
57. }
58. int **main**() {
59. int d, m, y;
60. **GetDate**(&d, &m, &y);
61. if (!**ValidDate**(d, m, y)) {
62. **printf**("Invalid Input Date!\n");
63. return 0;
64. }
65. **IncrementDate**(&d, &m, &y);
66. **PrintDate**(d, m, y);
67. return 0;
68. }

**PART A: Call Graph-based Integration Testing**

1. Draw the call graph



1. Plan integration order:

Bottom-up approach

Test isLeap(), LastOfMonth(), then integrate validate(), IncrementDate(), GetDate(), PrintDate(), then Main()

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Test ID | Function | Input | Expected Output | Actual Output | Result |
| CG1 | isLeap() | 2020 | True | True | Pass |
| CG2 | isLeap() | 1900 | False | False | Pass |
| CG3 | isLeap() | 2000 | True | True | Pass |
| CG4 | isLeap() | 2023 | false | False | Pass |
| CG5 | LastOfMonth() | 2, 2020 | 29 | 29 | Pass |
| CG6 | LastOfMonth() | 2, 2000 | 28 | 28 | Pass |
| CG7 | LastOfMonth() | 4, 2000 | 30 | 30 | Pass |
| CG8 | LastOfMonth() | 12, 1950 | 31 | 31 | Pass |
| CG9 | ValidDate() | 31, 4, 2000 | False | False | Pass |
| CG10 | ValidDate() | 29, 2, 2020 | True | True | Pass |
| CG11 | ValidDate() | 29, 2, 2000 | False | False | Pass |
| CG12 | IncrementDate() | 15, 3, 2025 | 16/3/2025 | 16/3/2025 | Pass |
| CG13 | IncrementDate() | 29, 2, 2020 | 1/3/2020 | 1/3/2020 | Pass |

1. Design interface tests to verify correctness at each integration step.

isLeap() interace

1. #include <stdio.h>
2. #include <stdbool.h>
3. **bool** **isLeap**(int year) {
4. return (year % 400 == 0) || (year % 4 == 0 && year % 100 != 0);
5. }
6. int **main**() {
7. **printf**("isLeap(2020): %d \n", **isLeap**(2020));
8. **printf**("isLeap(1900): %d \n", **isLeap**(1900));
9. **printf**("isLeap(2000): %d \n", **isLeap**(2000));
10. **printf**("isLeap(2023): %d \n", **isLeap**(2023));
11. return 0;
12. }

LastOfMonth() interface

1. #include <stdio.h>
2. #include <stdbool.h>
3. **bool** **isLeap**(int year) {
4. return (year % 400 == 0) || (year % 4 == 0 && year % 100 != 0);
5. }
6. int **LastOfMonth**(int month, int year) {
7. switch(month) {
8. case 1: return 31;
9. case 2: return (**isLeap**(year) ? 29 : 28);
10. case 3: return 31;
11. case 4: return 30;
12. case 5: return 31;
13. case 6: return 30;
14. case 7: return 31;
15. case 8: return 31;
16. case 9: return 30;
17. case 10: return 31;
18. case 11: return 30;
19. case 12: return 31;
20. default: return -1;
21. }
22. }
23. int **main**() {
24. **printf**("LastOfMonth(2, 2020): %d \n", **LastOfMonth**(2,2020));
25. **printf**("LastOfMonth(2, 2000): %d \n", **LastOfMonth**(2,2023));
26. **printf**("LastOfMonth(4, 2000): %d \n", **LastOfMonth**(4,2025));
27. **printf**("LastOfMonth(12, 1950): %d \n", **LastOfMonth**(12,2025));
28. return 0;
29. }

ValidDate() interface

1. #include <stdio.h>
2. #include <stdbool.h>
3. **bool** **isLeap**(int year) { return (year % 400 == 0) || (year % 4 == 0 && year % 100 != 0); }
4. int **LastOfMonth**(int m, int y) { return (m==2 ? (**isLeap**(y)?29:28) : (m==4||m==6||m==9||m==11?30:31)); }
5. **bool** **ValidDate**(int d, int m, int y) {
6. if (y < 1900 || y > 2100) return **false**;
7. if (m < 1 || m > 12) return **false**;
8. if (d < 1 || d > **LastOfMonth**(m, y)) return **false**;
9. return **true**;
10. }
11. int **main**() {
12. **printf**("ValidDate(31, 4, 2000): %d \n", **ValidDate**(31,4,2000));
13. **printf**("ValidDate(29, 2, 2020): %d \n", **ValidDate**(29,2,2020));
14. **printf**("ValidDate(29, 2, 2000): %d \n", **ValidDate**(29,2,2000));
15. return 0;
16. }

NextDate() interface

1. #include <stdio.h>
2. #include <stdbool.h>
3. **bool** **isLeap**(int year) { return (year % 400 == 0) || (year % 4 == 0 && year % 100 != 0); }
4. int **LastOfMonth**(int m, int y) { return (m==2 ? (**isLeap**(y)?29:28) : (m==4||m==6||m==9||m==11?30:31)); }
5. void **IncrementDate**(int \*d, int \*m, int \*y) {
6. if (\*d < **LastOfMonth**(\*m, \*y)) (\*d)++;
7. else { \*d = 1; if (\*m < 12) (\*m)++; else { \*m = 1; (\*y)++; } }
8. }
9. int **main**() {
10. int d=15,m=3,y=2025;
11. **IncrementDate**(&d,&m,&y);
12. **printf**("NextDate of 15/3/2025: %d/%d/%d \n", d,m,y);
13. d=29; m=2; y=2020;
14. **IncrementDate**(&d,&m,&y);
15. **printf**("NextDate of 29/2/2020: %d/%d/%d \n", d,m,y);
16. return 0;
17. }

**PART B: Path Based Integration Testing**

1. Identify decision points in IncrementDate() and ValidDate().

ValidDate() decision points:

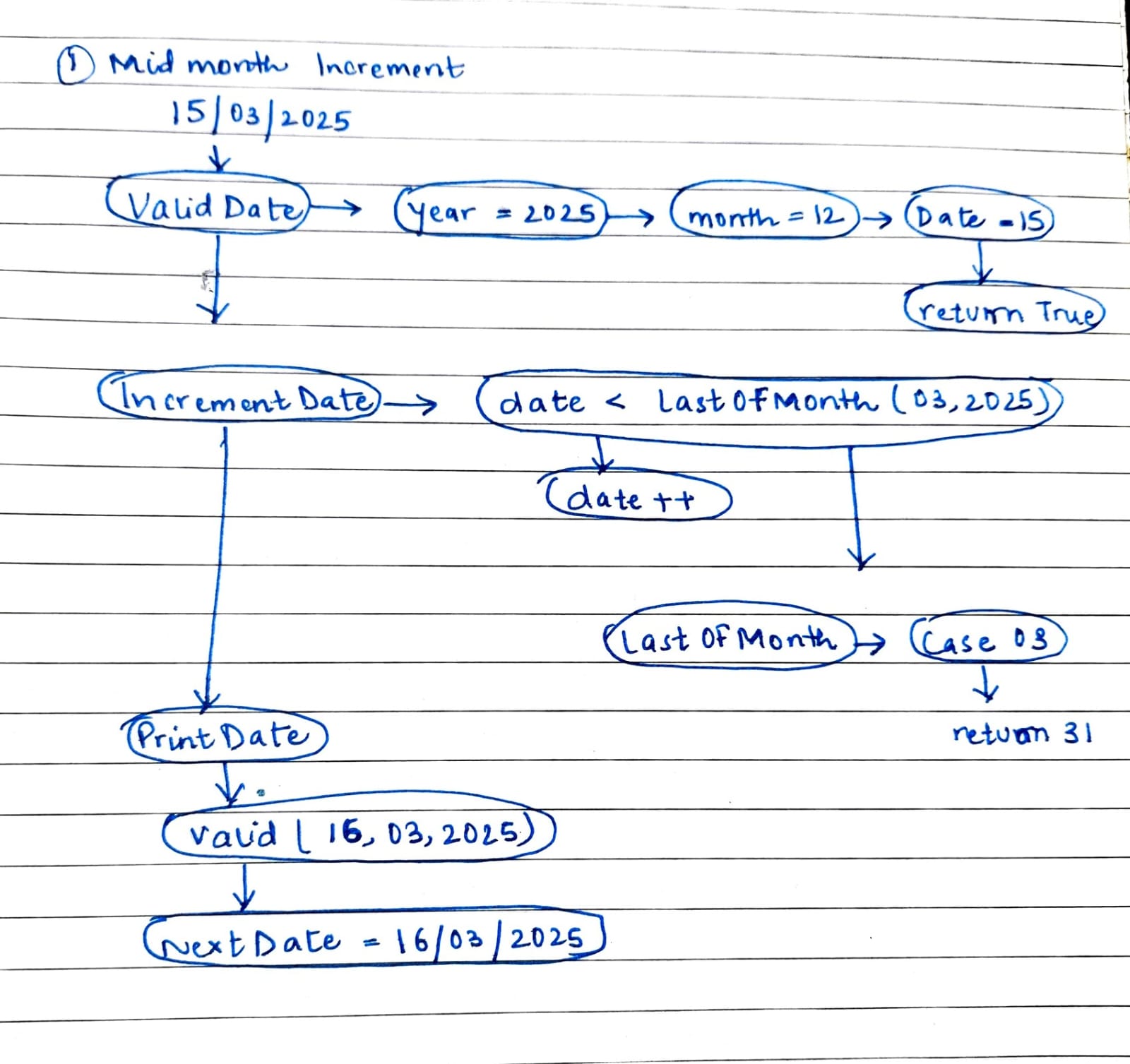
* Year is within range (1900, 2100)
* Month is between 1 and 12
* Day is valid using LastOfMonth(month, year)

IncrementDate() decision points:

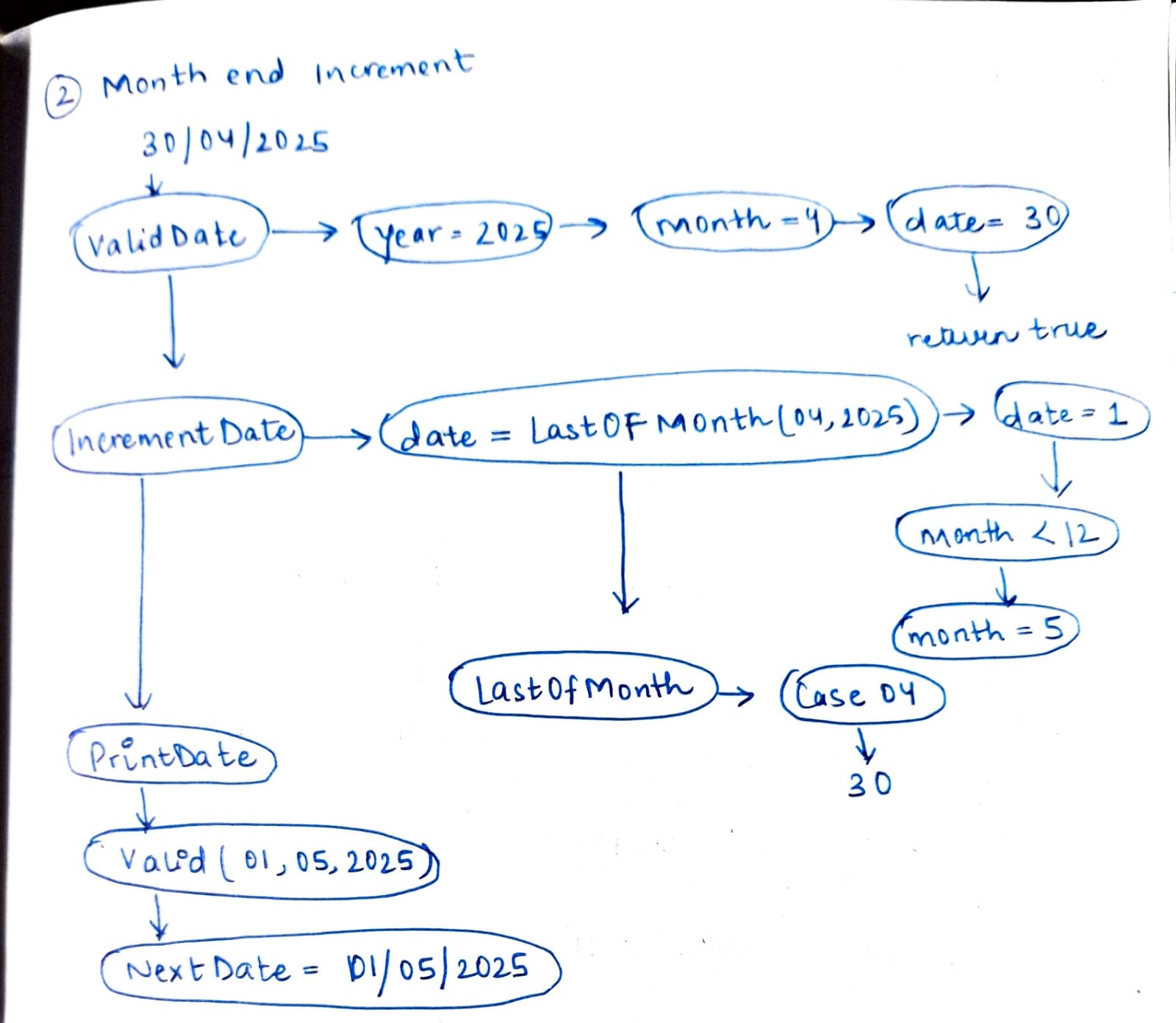
* Day < LastOfMonth(month, year) then increment day
* Else if month < 12, then reset day = 1, increment month
* Else (next year), reset day = 1, month = 1, increment year

1. Enumerate independent paths (basis paths):

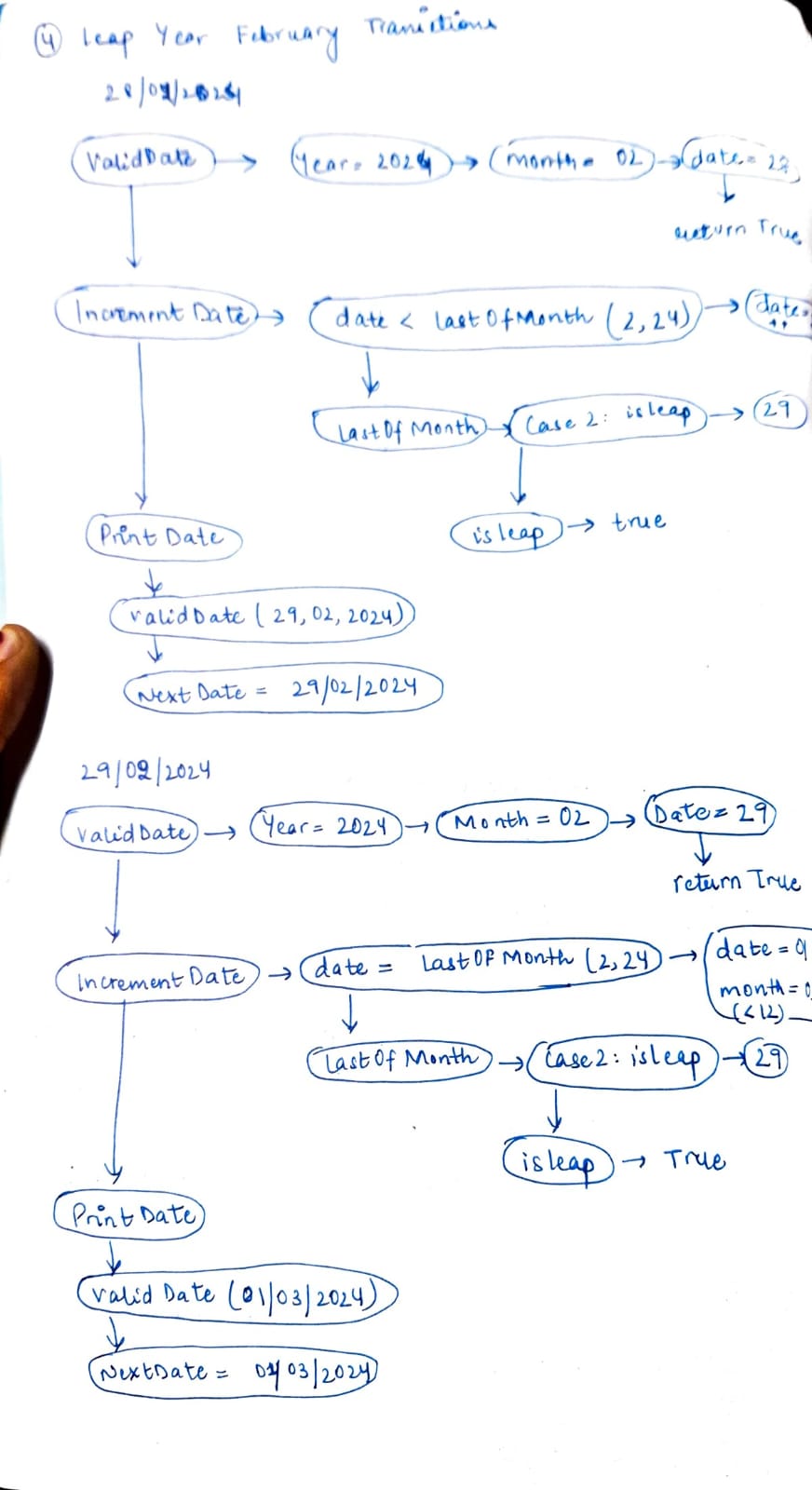
* Mid-month increment (15/03/2025 → 16/03/2025)



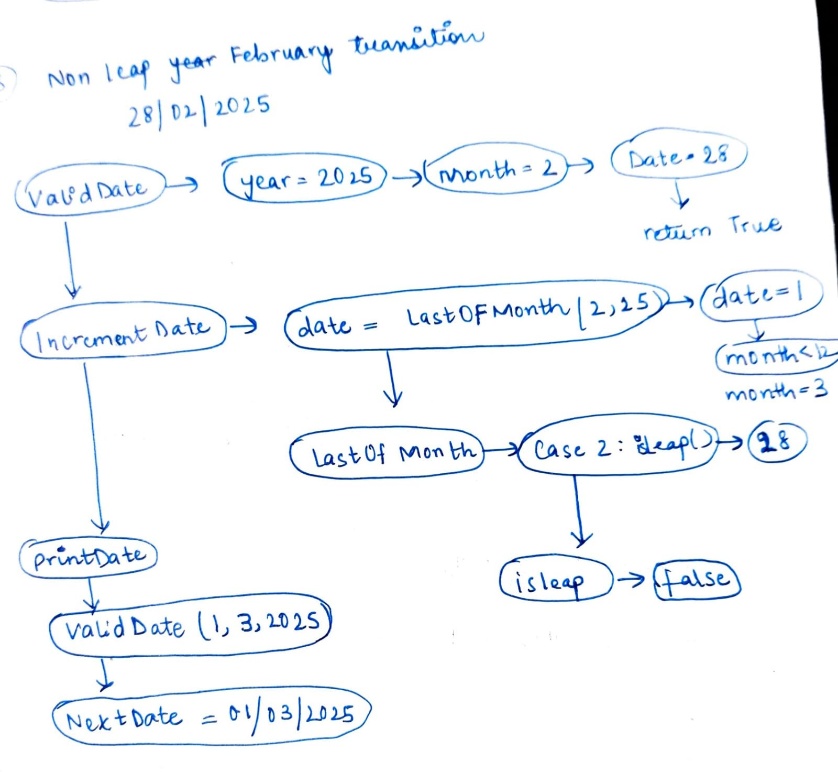
* Month-end increment (30/04/2025 → 01/05/2025)



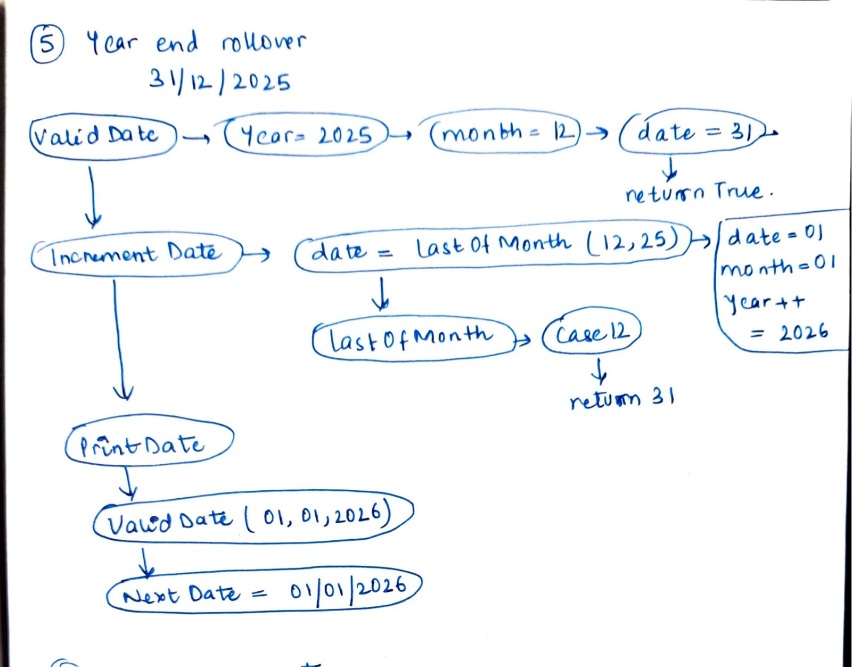
* Leap year February transitions (28/02/2024 → 29/02/2024, 29/02/2024 → 01/03/2024)



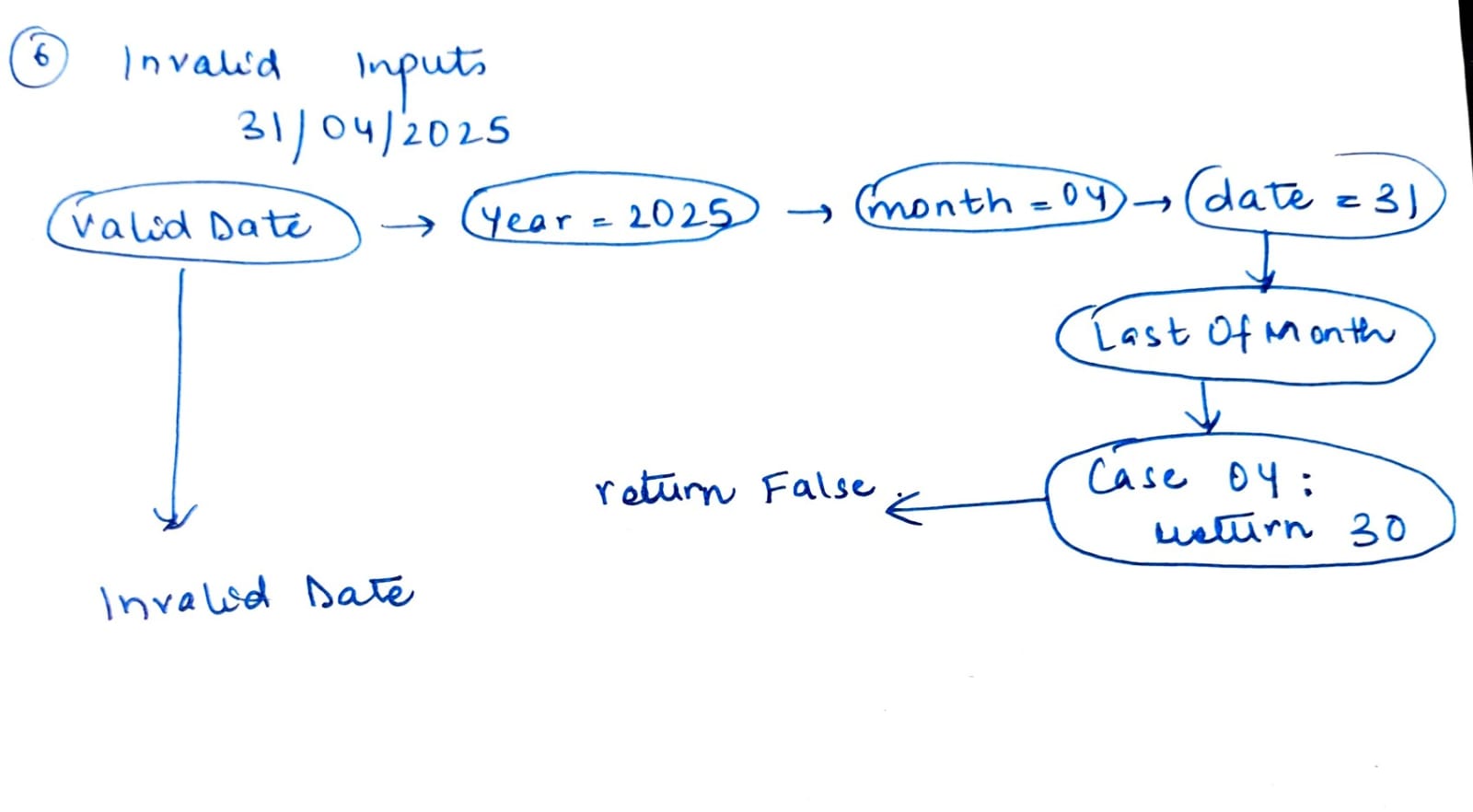
* Non-leap February transition (28/02/2025 → 01/03/2025)



* Year-end rollover (31/12/2025 → 01/01/2026)



* Invalid inputs (31/04/2025 → Invalid)



1. Design test cases to cover all independent paths

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Test Case | Input | Expected Output | Actual output | Result | Path covered |
| 1 | 15/03/2025 | 16/03/2025 | 16/03/2025 | Pass | Mid-month increment |
| 2 | 30/04/2025 | 01/05/2025 | 01/05/2025 | Pass | Month-end increment |
| 3 | 31/01/2025 | 01/02/2025 | 01/02/2025 | Pass | Month-end increment |
| 4 | 28/02/2024 | 29/02/2024 | 29/02/2024 | Pass | Leap year increment |
| 5 | 29/02/2024 | 01/03/2024 | 01/03/2024 | Pass | Leap year increment |
| 6 | 28/02/2025 | 01/03/2025 | 01/03/2025 | Pass | Non-leap year increment |
| 7 | 31/12/2025 | 01/01/2026 | 01/01/2026 | Pass | Year-end rollover |
| 8 | 31/04/2025 | Invalid date | Invalid date | Pass | Invalid input |
| 9 | 29/02/2023 | Invalid date | Invalid date | Pass | Invalid input |
| 10 | 15/05/1800 | Invalid date | Invalid date | Pass | Invalid input |