**Lab#7 – White-box testing**

# วัตถุประสงค์การเรียนรู้

1. ผู้เรียนสามารถออกแบบการทดสอบแบบ White-box testing ได้
2. ผู้เรียนสามารถวิเคราะห์ปัญหาด้วย Control flow graph ได้
3. ผู้เรียนสามารถออกแบบกรณีทดสอบโดยคำนึงถึง Line coverage ได้
4. ผู้เรียนสามารถออกแบบกรณีทดสอบโดยคำนึงถึง Block coverage ได้
5. ผู้เรียนสามารถออกแบบกรณีทดสอบโดยคำนึงถึง Branch coverage ได้
6. ผู้เรียนสามารถออกแบบกรณีทดสอบโดยคำนึงถึง Condition coverage ได้
7. ผู้เรียนสามารถออกแบบกรณีทดสอบโดยคำนึงถึง Branch and Condition coverage ได้

# โจทย์: Clump counts

Clump counts (<https://codingbat.com/prob/p193817>) เป็นโปรแกรมที่ใช้ในการนับการเกาะกลุ่มกันของข้อมูลภายใน Array โดยการเกาะกลุ่มกันจะนับสมาชิกใน Array ที่อยู่ติดกันและมีค่าเดียวกันตั้งแต่สองตัวขึ้นไปเป็นหนึ่งกลุ่ม เช่น

[1, 2, 2, 3, 4, 4] → 2

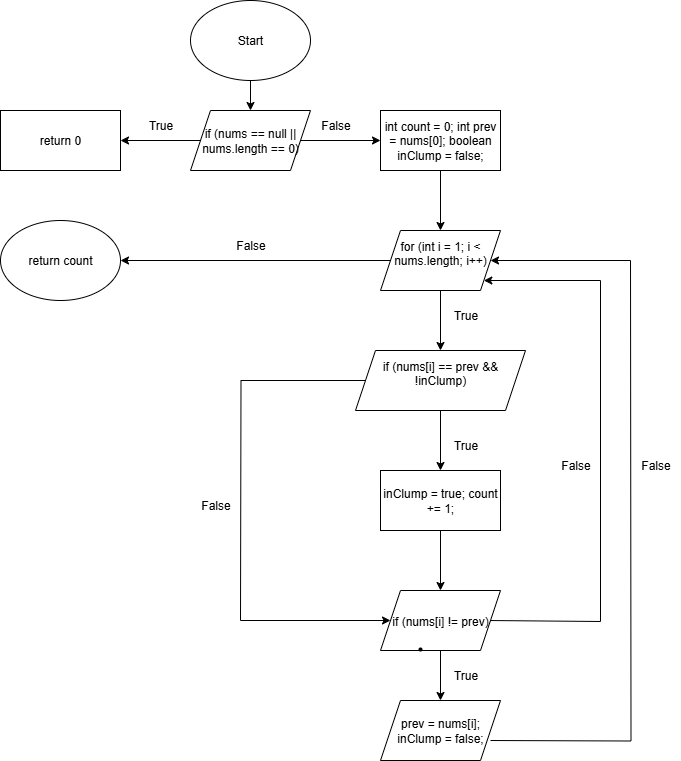
[1, 1, 2, 1, 1] → 2

[1, 1, 1, 1, 1] → 1

ซอร์สโค้ดที่เขียนขึ้นเพื่อนับจำนวนกลุ่มของข้อมูลที่เกาะอยู่ด้วยกันอยู่ที่ <https://github.com/ChitsuthaCSKKU/SQA/tree/2025/Assignment/Lab7> โดยที่ nums เป็น Array ที่ใช้ในการสนับสนุนการนับกลุ่มของข้อมูล (Clump) ทำให้ nums เป็น Array ที่จะต้องไม่มีค่าเป็น Null และมีความยาวมากกว่า 0 เสมอ หาก nums ไม่เป็นไปตามเงื่อนไขที่กำหนดนี้ โปรแกรมจะ return ค่า 0 แทนการ return จำนวนกลุ่มของข้อมูล

# แบบฝึกปฏิบัติที่ 7.1 Control flow graph

จากโจทย์และ Source code ที่กำหนดให้ (CountWordClumps.java) ให้เขียน Control Flow Graph (CFG) ของเมธอด countClumps() จากนั้นให้ระบุ Branch และ Condition ทั้งหมดที่พบใน CFG ให้ครบถ้วน



Branch

**Branch 1 (B1)**: เงื่อนไขการตรวจสอบอาร์เรย์

* **Source**: if (nums == null || nums.length == 0)
* **True Path**: ไปที่คำสั่ง return 0
* **False Path**: ไปที่คำสั่งเริ่มต้นของเมธอด

**Branch 2 (B2)**: เงื่อนไขการทำงานของลูป

* **Source**: for (int i = 1; i < nums.length; i++)
* **True Path**: เข้าไปทำงานในลูป for
* **False Path**: ออกจากลูปและไปที่คำสั่ง return count

**Branch 3 (B3)**: เงื่อนไขการเริ่มต้นนับ Clump

* **Source**: if (nums[i] == prev && !inClump)
* **True Path**: ไปที่คำสั่ง inClump = true; count += 1;
* **False Path**: ไปที่เงื่อนไขถัดไป

**Branch 4 (B4)**: เงื่อนไขการอัปเดตค่า prev

* **Source**: if (nums[i] != prev)
* **True Path**: ไปที่คำสั่ง prev = nums[i]; inClump = false;
* **False Path**: ไปที่การทำงานในลูปซ้ำ

Condition

**Condition 1**: if (nums == null || nums.length == 0)

* **ถ้าเป็นจริง**: Flow จะไปที่ return 0
* **ถ้าเป็นเท็จ**: Flow จะไปที่ int count = 0; int prev = nums[0]; boolean inClump = false;

**Condition 2**: i < nums.length

* **ถ้าเป็นจริง**: Flow จะไปที่ if (nums[i] == prev && !inClump)
* **ถ้าเป็นเท็จ**: Flow จะไปที่ return count

**Condition 3**: if (nums[i] == prev && !inClump)

* **ถ้าเป็นจริง**: Flow จะไปที่ inClump = true; count += 1;
* **ถ้าเป็นเท็จ**: Flow จะไปที่ if (nums[i] != prev)

**Condition 4**: if (nums[i] != prev)

* **ถ้าเป็นจริง**: Flow จะไปที่ prev = nums[i]; inClump = false;
* **ถ้าเป็นเท็จ**: Flow จะกลับไปที่ Condition 2 (i < nums.length) เพื่อทำลูปซ้ำ

# แบบฝึกปฏิบัติที่ 7.2 Line Coverage

1. จาก Control Flow Graph (CFG) ของเมธอด countClumps() ในข้อที่ 1 ให้ออกแบบกรณีทดสอบเพื่อให้ได้ Line coverage = 100%
2. เขียนกรณีทดสอบที่ได้ พร้อมระบุบรรทัดที่ถูกตรวจสอบทั้งหมด
3. แสดงวิธีการคำนวณค่า Line coverage

| Test Case No. | Input(s) | Expected Result(s) | Path and Branch |
| --- | --- | --- | --- |
| TC01 | NULL | 0 | Line No.:2,3 |
| TC02 | {1, 2, 2, 3, 3, 3} | 2 | Line No.:2, 5, 6, 7, 8, 9 (T), 10, 11, 13 (F), 8, 9 (F), 13 (T), 14, 15, 8, 9 (T), 10, 11, 13 (F), 8, 9 (F), 13 (T), 14, 15, 8, 9 (F), 13 (F), 8, 18 |
| TC03 | {1, 2, 3} | 0 | Line No.:2, 5, 6, 7, 8, 9 (F), 13 (T), 14, 15, 8, 9 (F), 13 (T), 14, 15, 8, 18 |
| TC04 | {} | 0 | Line No.:2,3 |

Line coverage = 13/13×100 = 100%

TC01 = 2/13 × 100 ≈ 15.38%

TC02 = 12/13​ × 100 ≈ 92.31%

TC03 = 10/13 × 100 ≈ 76.92%

TC04 = 2/13 ​× 100 ≈ 15.38%

# แบบฝึกปฏิบัติที่ 7.3 Block Coverage

1. จาก Control Flow Graph (CFG) ของเมธอด countClumps() ในข้อที่ 1 ให้ออกแบบกรณีทดสอบเพื่อให้ได้ Block coverage = 100%
2. เขียนกรณีทดสอบที่ได้ พร้อมระบุ Block ที่ถูกตรวจสอบทั้งหมด
3. แสดงวิธีการคำนวณค่า Block coverage

ตอบ กำหนดให้  
**Block A**: if (nums == null || nums.length == 0) (บรรทัด 2)

**Block B**: return 0; (บรรทัด 3)

**Block C**: int count = 0; int prev = nums[0]; boolean inClump = false; (บรรทัด 5, 6, 7)

**Block D**: for (int i = 1; i < nums.length; i++) (บรรทัด 8)

**Block E**: if (nums[i] == prev && !inClump) (บรรทัด 9)

**Block F**: inClump = true; count += 1; (บรรทัด 10, 11)

**Block G**: if (nums[i] != prev) (บรรทัด 13)

**Block H**: prev = nums[i]; inClump = false; (บรรทัด 14, 15)

**Block I**: return count; (บรรทัด 18)

| Test Case No. | Input(s) | Expected Result(s) | Path and Branch |
| --- | --- | --- | --- |
| TC01 | NULL | 0 | Block: A(True)->B |
| TC02 | {1, 2, 2, 3} | 1 | Block:Block: A(False)->C->D->(E(False)->G(True)->H->D)->(E(True)->F->G(False)->D)->(E(False)->G(True)->H->D)->I |

Block coverage = 9/9​×100=100%

# แบบฝึกปฏิบัติที่ 7.3 Branch Coverage

1. จาก Control Flow Graph (CFG) ของเมธอด countClumps() ในข้อที่ 1 ให้ออกแบบกรณีทดสอบเพื่อให้ได้ Branch coverage = 100%
2. เขียนกรณีทดสอบที่ได้ พร้อมระบุ Path และ Branch ที่ถูกตรวจสอบทั้งหมด
3. แสดงวิธีการคำนวณค่า Branch coverage

ตอบ กำหนดให้

**Branch 1**: เงื่อนไข if (nums == null || nums.length == 0)

**Branch 2**: เงื่อนไข for (int i = 1; i < nums.length; i++)

**Branch 3**: เงื่อนไข if (nums[i] == prev && !inClump)

**Branch 4**: เงื่อนไข if (nums[i] != prev)

| Test Case No. | Input(s) | Expected Result(s) | Path and Branch |
| --- | --- | --- | --- |
| TC01 | NULL | 0 | Path: Start -> if (T) -> Exit  Branch: B1(True) |
| TC02 | {1, 2} | 0 | Path: Start -> if (F) -> for (T) -> if-clump (F) -> if-prev (T) -> Loop -> for (F) -> return count  Branch: B1(False), B2(True, False), B3(False), B4(True) |
| TC03 | {1, 1} | 1 | Path: Start -> if (F) -> for (T) -> if-clump (T) -> if-prev (F) -> Loop -> for (F) -> return count  Branch: B1(False), B2(True, False), B3(True), B4(False) |
| TC04 | {1} | 0 | Path: Start -> if (F) -> for (F) -> return count  Branch: B1(False), B2(False) |

Branch coverage = 8/8​×100=100%

# แบบฝึกปฏิบัติที่ 7.4 Condition Coverage

1. จาก Control Flow Graph (CFG) ของเมธอด countClumps() ในข้อที่ 1 ให้ออกแบบกรณีทดสอบเพื่อให้ได้ Condition coverage = 100%
2. เขียนกรณีทดสอบที่ได้ พร้อมระบุ Path และ Condition ที่ถูกตรวจสอบทั้งหมด เช่น Condition A = T และ Condition B = F
3. แสดงวิธีการคำนวณค่า Condition coverage

ตอบ

| Sub-condition | Goal (True/False) |
| --- | --- |
| A: nums == null | TRUE |
| B: nums.length == 0 | TRUE |
| A: nums == null | FALSE |
| B: nums.length == 0 | FALSE |
| C: i < nums.length | TRUE |
| C: i < nums.length | FALSE |
| D: nums[i] == prev | TRUE |
| D: nums[i] == prev | FALSE |
| E: !inClump | TRUE |
| E: !inClump | FALSE |

| Test Case No. | Input(s) | Expected Result(s) | Path and Condition |
| --- | --- | --- | --- |
| TC01 | NULL | 0 | Condition 1: `(A=T |
| TC02 | {} | 0 | Condition 1: `(A=F |
| TC03 | {1, 2} | 0 | Condition 1: `(A=F |
| TC04 | {1, 1, 1} | 1 | Condition 1: `(A=F |

Condition coverage = 10/5×2​×100 = 10/10​×100 = 100%

# แบบฝึกปฏิบัติที่ 7.5 Branch and Condition Coverage (C/DC coverage)

1. จาก Control Flow Graph (CFG) ของเมธอด countClumps() ในข้อที่ 1 ให้ออกแบบกรณีทดสอบให้ได้ C/DC coverage = 100%
2. เขียนกรณีทดสอบที่ได้ พร้อมระบุ Path, Branch, และ Condition ที่ถูกตรวจสอบทั้งหมด
3. แสดงวิธีการคำนวณค่า C/DC coverage
4. เขียนโค้ดสำหรับทดสอบตามกรณีทดสอบที่ออกแบบไว้ด้วย JUnit และบันทึกผลการทดสอบ

ตอบ

| Test Case No. | Input(s) | Expected Result(s) | Actual Result(s) | Path, Branch, and Condition |
| --- | --- | --- | --- | --- |
| TC01 | NULL | 0 | Pass/Fail: | Path: if (T) -> return 0  Branch: B1(T)  Condition: `(A=T |
| TC02 | {} | 0 | Pass/Fail: | Path: if (T) -> return 0  Branch: B1(T)  Condition: `(A=F |
| TC03 | {1} | 0 | Pass/Fail: | Path: if (F) -> for (F) -> return count  Branch: B1(F), B2(F)  Condition: `(A=F |
| TC04 | {1,1} | 1 | Pass/Fail: | Path: if(F) -> for(T) -> if-clump(T) -> if-prev(F) -> Loop -> for(F) -> return count  Branch: B1(F), B2(T), B2(F), B3(T), B4(F)  Condition: (D=T && E=T) -> Decision=T  (F=F) -> Decision=F |
| TC05 | {1,1,1} | 1 | Pass/Fail: | Path: if(F) -> for(T) -> if-clump(T) -> if-clump(F) -> if-prev(F) -> Loop -> for(F) -> return count  Branch: B1(F), B2(T), B2(F), B3(T), B3(F), B4(F)  Condition: (D=T && E=T) -> Decision=T  (D=T && E=F) -> Decision=F  (F=F) -> Decision=F |

C/DC coverage = 20/20​×100 = 100%