# Take Home C Exam

## Black Box Testing

**Specification.**

**Input Data.**

feelings[i] for i = 1, …, n, offset

**Preconditions.**

feelings[i] in {-1, 0, 1}

offset in [0, …, n - 1]

**Output Results.**

position

**Postconditions.**

position in [-1, 0, …, n - 1]

**Equivalence Classes**

| EC Number | Condition | Valid EC | Invalid EC |
|---|---|---|---|
| 1 | feelings[i] in {-1, 0, 1} | feelings[i] = -1 | |
| 2 | | feelings[i] = 0 | |
| 3 | | feelings[i] = 1 | |
| 4 | | | feelings[i] not in {-1, 0, 1} |
| 5 | 0 <= offset <= n - 1 | 0 <= offset < n - 1 | |
| 6 | | | offset < 0 |
| 7 | | | offset > n - 1 |
| 8 | -1 <= position <= n - 1 | -1 <= position <= n - 1 | |
| 9 | | | position < -1 |
| 10 | | | position > n - 1 |

## Test Cases

| TC Number | ECs | Input | | Output | |
|---|---|---|---|---|---|
| | | feelings[i] | offset | expected position | actual position |
| 1 | 1, 2, 3, 5, 8 | [0, 1, -1] | 0 | 2 | 2 |
| 2 | 4 | [0, 2, 1] | 1 | error message | error message |
| 3 | 6 | [-1, 0, 1] | -5 | error message | error message |
| 4 | 7 | [-1, 0, 1] | 3 | error message | error message |
| 5 | 9, 10 | - | - | not possible | - |

## Boundary Value Analysis

| BVC Number | BV TC Number | BV Condition | BVA Test Case |
|---|---|---|---|
| 1 | 1 | feelings[i] in {-1, 0, 1} | feelings[i] = -1 |
| | 2 | | feelings[i] = 1 |
| 2 | 3 | 0 <= offset <= n - 1 | offset = -1 |
| | 4 | | offset = 0 |
| | 5 | | offset = 1 |
| | 6 | | offset = n - 2 |
| | 7 | | offset = n - 1 |
| | 8 | | offset = n |
| 3 | 9 | -1 <= position <= n - 1 | position = -2 |
| | 10 | | position = -1 |
| | 11 | | position = 0 |
| | 12 | | position = n - 2 |
| | 13 | | position = n - 1 |
| | 14 | | position = n |

**Test Case Design**

| TC Number | BVC TCs | Input | | Output | |
|---|---|---|---|---|---|
| | | **feelings** | **offset** | **expected position** | **actual position** |
| 1 | 1, 2, 5, 6, 13 | [0, 1, -1] | 1 | 2 | 2 |
| 2 | 3 | [-1, 0, 1] | -1 | error message | error message |
| 3 | 4, 11, 12 | [-1, -1] | 0 | 0 | 0 |
| 4 | 8 | [-1, 0, 1] | 3 | error message | error message |
| 5 | 7, 10 | [1, 1, 1, 1] | 3 | -1 | -1 |
| 6 | 9, 14 | - | - | not possible | - |

# White Box Testing

**Specification.**

**Input Data.**

feelings[i] for i = 1, …, n, position

**Preconditions.**

feelings[i] in {-1, 0, 1}

position in [0, …, n - 1]

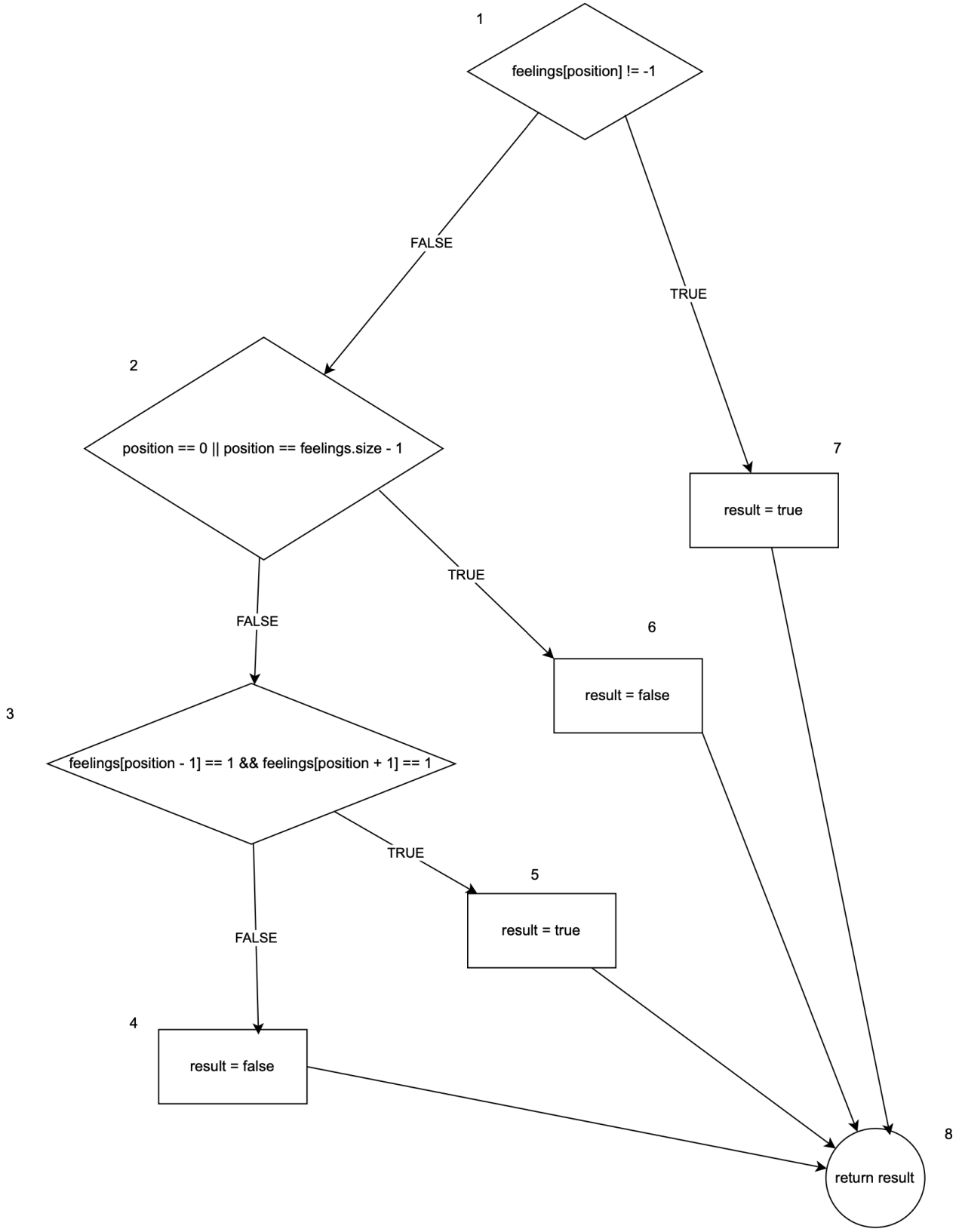
**Output Results.**

result

**Postconditions.**

result in {true, false}

# Control Flow Graph



**Cyclomatic Complexity**

CC = E - N  + 2 = 10 - 8 + 2 = 4, where E = # edges, N = # nodes
CC = P + 1 = 4, where P = # predicates
CC = R = 4, where R = # regions

**Paths**

1: 1 - 2 - 3 - 4 - 8
2: 1 - 2 - 3 - 5 - 8
3: 1 - 2 - 6 - 8
4: 1 - 7 - 8

**Test Case Design**

| TC Number | Input | | Output | Statements | CDC | | | |
|---|---|---|---|---|---|---|---|---|
| | feelings | position | result | | S1 | S2 | S2 C1 | S2 C2 |
| 1 | [0, -1, 1] | 0 | TRUE | 1, 7, 8 | x (true) | | | |
| 2 | [0, -1, 0] | 1 | FALSE | 2, 3, 4 | x (false) | x (false) | FALSE | TRUE |
| 3 | [-1] | 0 | FALSE | 6 | x (false) | x (true) | TRUE | TRUE |
| 4 | [1, -1, 1] | 1 | TRUE | 5 | x (false) | x (false) | FALSE | FALSE |

| TC Number | CDC | | | Path | | | |
|---|---|---|---|---|---|---|---|
| | S3 | S3 C1 | S3 C2 | P1 | P2 | P3 | P4 |
| 1 | | | | | | | x |
| 2 | x (false) | FALSE | FALSE | x | | | |
| 3 | | | | | | x | |
| 4 | x (true) | TRUE | TRUE | | x | | |

# Integration Testing

**Specification.**

**Input Data.**

feelings[i] for i = 1, …, n

**Preconditions.**

feelings[i] in {-1, 0, 1}

**Output Results.**

result[j]

**Postconditions.**

result[j] in {-1, 0, 1}

We used the Big-Bang Integration strategy. The base idea of this strategy is to integrate the tested methods (or components), all at once. For our system we tested each sub-algorithm individually first, and for the integration method, we've simply called the "BeHappy" method, which already integrated in its source code all the other sub-algorithms (FindSadFeeling, CheckNeighbours, InsertHappyFeelings). Beside the unit tests, which tested each component individually, we only had one single integration test (as described above).

**Test Case Design**

Input: [-1,-1, 0, 0, 1, 1, -1, 1, 0, -1, 1, 0, 1, 1, -1, 0, 1, 1]

Output: [1, -1, 1, 1, -1, 1, 0, 0, 1, 1, -1, 1, 0, 1, -1, 1, 1, 0, 1, 1, 1, -1, 1, 0, 1, 1]