Data Flow Testing

Table of Contents

Project Overview

Data flow testing is a white box software testing technique that focuses on the flow of data within a program to ensure that data usage is correct and logical. In order to test the software we examine the code and create paths for each variable. We thoroughly examined each of the seven variables in the program, assigning each of them as a def or use nodes and further sorting the use nodes into p-use and c-use nodes. We also created paths for each variable, starting with the definition of each use, going through every usage of the variables. Finally we created paths within each of the three functions, examining each of the conditions in the functions and each of the different paths the conditionals lead to.

Definition-Coverage Path

1. advancePlayerA Function:

- **Condition 1: if(obstacalProb >= 1 && obstacalProb <= 4)**
  - **Path 1: True (obstacalProb is between 1 and 4)**
  - **Path 2: False (obstacalProb is not between 1 and 4)**
- **Condition 2: else if(obstacalProb >= 5 && obstacalProb <= 6)**
  - **Path 3: True (obstacalProb is 5 or 6)**
  - **Path 4: False (obstacalProb is not 5 or 6)**
- **Condition 3: else if(obstacalProb == 7)**
  - **Path 5: True (obstacalProb is 7)**
  - **Path 6: False (obstacalProb is not 7)**
- **Condition 4: else if(obstacalProb == 8)**
  - **Path 7: True (obstacalProb is 8)**
  - **Path 8: False (obstacalProb is not 8)**
- **Condition 5: else if(obstacalProb == 9)**
  - **Path 9: True (obstacalProb is 9)**
  - **Path 10: False (obstacalProb is not 9)**
- **Boundary Check 1: if(*ptrplayerA < 1)**
  - **Path 11: True (*ptrplayerA is less than 1)**
  - **Path 12: False (*ptrplayerA is not less than 1)**
- **Boundary Check 2: else if(*ptrplayerA > GAME_LENGTH)**
  - **Path 13: True (*ptrplayerA is greater than GAME_LENGTH)**
  - **Path 14: False (*ptrplayerA is not greater than GAME_LENGTH)**

2. advancePlayerB Function:

- **Condition 1: if(obstacalProb >= 1 && obstacalProb <= 3)**
  - **Path 1: True (obstacalProb is between 1 and 3)**
  - **Path 2: False (obstacalProb is not between 1 and 3)**
- **Condition 2: else if(obstacalProb == 4)**
  - **Path 3: True (obstacalProb is 4)**
  - **Path 4: False (obstacalProb is not 4)**
- **Condition 3: else if(obstacalProb >= 5 && obstacalProb <= 6)**
  - **Path 5: True (obstacalProb is 5 or 6)**
  - **Path 6: False (obstacalProb is not 5 or 6)**
- **Condition 4: else if(obstacalProb == 7)**
  - **Path 7: True (obstacalProb is 7)**
  - **Path 8: False (obstacalProb is not 7)**
- **Condition 5: else if(obstacalProb >= 8 && obstacalProb <= 9)**
  - **Path 9: True (obstacalProb is 8 or 9)**
  - **Path 10: False (obstacalProb is not 8 or 9)**

- **Boundary Check 1: if(*ptrplayerB < 1)**
    - **Path 11: True (*ptrplayerB is less than 1)**
    - **Path 12: False (*ptrplayerB is not less than 1)**
- **Boundary Check 2: else if(*ptrplayerB > GAME_LENGTH)**
    - **Path 13: True (*ptrplayerB is greater than GAME_LENGTH)**
    - **Path 14: False (*ptrplayerB is not greater than GAME_LENGTH)**

3. printPosition Function:

- **Condition 1: if(*ptrplayerA == *ptrplayerB)**
    - **Path 1: True (*ptrplayerA is equal to *ptrplayerB)**
    - **Path 2: False (*ptrplayerA is not equal to *ptrplayerB)**
- **Condition 2: if(*ptrplayerA > 1)**
    - **Path 3: True (*ptrplayerA is greater than 1)**
    - **Path 4: False (*ptrplayerA is not greater than 1)**
- **Condition 3: else if(*ptrplayerA < *ptrplayerB)**
    - **Path 5: True (*ptrplayerA is less than *ptrplayerB)**
    - **Path 6: False (*ptrplayerA is not less than *ptrplayerB)**
- **Condition 4: else**
    - **Path 7: True (*ptrplayerA is greater than *ptrplayerB)**
    - **Path 8: False (*ptrplayerA is not greater than *ptrplayerB)**

Definition-Use Path

1. playerA:

- Definition: int playerA = 1; in main()
- Uses:
    - Passed to advancePlayerA(&playerA) → used in advancePlayerA where it is modified.
    - Passed to printPosition(&playerA, &playerB) → used in printPosition.
- DU Paths:
    - Path 1: main() -> advancePlayerA -> main()
    - Path 2: main() -> printPosition -> main()

2. playerB:

- Definition: int playerB = 1; in main()
- Uses:
    - Passed to advancePlayerB(&playerB) → used in advancePlayerB where it is modified.
    - Passed to printPosition(&playerA, &playerB) → used in printPosition.
- DU Paths:
    - Path 1: main() -> advancePlayerB -> main()
    - Path 2: main() -> printPosition -> main()

3. obstacalProb (in advancePlayerA):

- Definition: int obstacalProb = 0; initialized in advancePlayerA.
- Use:
    - Used in multiple if-else conditions for player movement logic.
- DU Paths:
    - Path 1: advancePlayerA() -> if(obstacalProb condition)
    - Path 2: advancePlayerA() -> else if (obstacalProb condition)
    - Each conditional block that checks obstacalProb forms a unique DU path from its definition to its use.

4. obstacalProb (in advancePlayerB):

- Definition: int obstacalProb = 0; initialized in advancePlayerB.
- Use:
    - Used in multiple if-else conditions for player movement logic.
- DU Paths:
    - Path 1: advancePlayerB() -> if(obstacalProb condition)
    - Path 2: advancePlayerB() -> else if (obstacalProb condition)
    - Each conditional block that checks obstacalProb forms a unique DU path from its definition to its use.

5. GAME_LENGTH:

- Definition: #define GAME_LENGTH 50 (constant, used as an upper boundary).
- Uses:
    - Used in boundary checks in advancePlayerA, advancePlayerB, and printPosition.
- DU Paths:
    - Path 1: advancePlayerA() -> GAME_LENGTH in boundary condition
    - Path 2: advancePlayerB() -> GAME_LENGTH in boundary condition
    - Path 3: printPosition() -> GAME_LENGTH used for formatting

6. ptrplayerA:

- Definition: Passed as a parameter to advancePlayerA and printPosition.
- Uses:
    - Used in player movement and boundary conditions within advancePlayerA.
    - Used in position printing logic in printPosition.
- DU Paths:
    - Path 1: main() -> advancePlayerA() -> use within conditions
    - Path 2: main() -> printPosition() -> use in formatting

7. ptrplayerB:

- Definition: Passed as a parameter to advancePlayerB and printPosition.
- Uses:
    - Used in player movement and boundary conditions within advancePlayerB.
    - Used in position printing logic in printPosition.
- DU Paths:
    - Path 1: main() -> advancePlayerB() -> use within conditions
    - Path 2: main() -> printPosition() -> use in formatting

Defining Nodes

Variables: playerA, playerB, srand, obstaclProb, GAME_LENGTH, ptrplayerA, ptrplayerB

Line 17: #define GAME_LENGTH 50
Line 27: int playerA = 1, playerB = 1;
Line 31: srand(time(0));
Line 64: int obstacalProb = 0;
Line 67: obstacalProb = 1 + rand() % 10;
Line 72: *ptrplayerA += 1;
Line 77: *ptrplayerA += 2;
Line 82: *ptrplayerA += 3;
Line 87: *ptrplayerA += 5;
Line 92: *ptrplayerA -= 3;
Line 97: *ptrplayerA -= 5;
Line 103: *ptrplayerA = 1;
Line 108: *ptrplayerA = GAME_LENGTH;
Line 115: int obstacalProb = 0;
Line 118: obstacalProb = 1 + rand() % 10;
Line 123: *ptrplayerB += 1;
Line 128: *ptrplayerB += 2;
Line 133: *ptrplayerB += 3;
Line 138: *ptrplayerB += 5;
Line 143: *ptrplayerB -= 3;
Line 148: *ptrplayerB -= 5;
Line 155: *ptrplayerB = 1;
Line 160: *ptrplayerB = GAME_LENGTH;
Line 173: *ptrplayerA -= 1;

Use Nodes

Variables: playerA, playerB, srand, obstaclProb, GAME_LENGTH, ptrplayerA, ptrplayerB

Lines 36,38,40,42: while (playerA != GAME_LENGTH && playerB != GAME_LENGTH)

```
{
    advancePlayerA(&playerA);

    advancePlayerB(&playerB);

    printPosition(&playerA, &playerB);
}
```

- P use
    - This is a P use because here we are looking anc comparing playerA/B and comparing these to GAMELENGTH

Line 45: if (playerA == GAME_LENGTH)

- P use node,
    - This is another P use of the playerA variable and the GAMELENGTH variable comparing the two

Line 69: if(obstacalProb >= 1 && obstacalProb <= 4)

- P use
    - Another P use node for obstaclProb comparing against one and four

Line 74: else if(obstacalProb >= 5 && obstacalProb <= 6)

- P use
    - Compares obstaclProb with 65 and 6 in P node usage

Line 79: else if(obstacalProb == 7)

- P use
    - P use because obsaclProb is being compared to 7

Line 84: else if(obstacalProb == 8)

- P use
    - obsaclProb is being compared to 8 in a P usage

Line 89: else if(obstacalProb == 9)

- P use
    - obsaclProb is being compared to 9 in a P usage

Line 101: if(*ptrplayerA < 1)

- P use
    - The pointer to prtPlayerA is being compared to 1 in another P usage

Line 105: else if(*ptrplayerA > GAME_LENGTH)

- P use
    - The pointer to prtPlayerA is being compared to GAMELENGTH both in another P usage

Line 120: if(obstacalProb >= 1 && obstacalProb <= 3)

- P use
    - obstcaclProb is being compared twice in two P usages to 1 and 3

Line 125: else if(obstacalProb == 4)

- P use
    - obstaclProb is being compared to 4 in a P usage

Line 130: else if(obstacalProb >= 5 && obstacalProb <= 6)

- P use
    - obstaclProb is in two P uses being compared to 5 and 6

Line 135: else if(obstacalProb == 7)

- P use
    - obstaclProb is being P used against 7

Line 140: else if(obstacalProb >= 8 && obstacalProb <= 9)

- P use
    - obstaclProb is being compared to 8 and 9 twice in a P usage

Line 152: if(*ptrplayerB < 1)

- P use
    - The pointer to prtplayerB is being compared to 1

Line 157: else if(*ptrplayerB > GAME_LENGTH)

- P use
    - The pointer to prtplayerB is being compared to GAME_LENGTH

Line 167: if(*ptrplayerA == *ptrplayerB)

- P use
    - The pointers to prtplayerA/B are being compared to eachother in a P usage

Line 170: if(*ptrplayerA > 1)

- P use
    - The pointer to prtplayerA is being P used to compare against 1

Lines 176-178: std::cout << std::setw(*ptrplayerA) << "A"
<< std::setw(*ptrplayerB - *ptrplayerA) << "B"
<< std::setw(GAME_LENGTH+1 - *ptrplayerB) << "|";

- C use
    - Here we have a C usage of a few variables where we are adding and subtracting different variables and then std::setw'ing them

Line 186: else if(*ptrplayerA < *ptrplayerB)

- P use
    - Here we have another p use comparing the pointers to prtplayerA/B

Lines 189-191: std::cout << std::setw(*ptrplayerA) << "A"
<< std::setw(*ptrplayerB - *ptrplayerA) << "B"
<< std::setw(GAME_LENGTH+1 - *ptrplayerB) << "|";

- C use
    - Here is another C use which takes the pointer to prtplayerA/B as well as GAME_LENGTH and doing math to then std::setw them again

Lines 196-198: std::cout << std::setw(*ptrplayerB) << "B"
<< std::setw(*ptrplayerA - *ptrplayerB) << "A"
<< std::setw(GAME_LENGTH+1 - *ptrplayerA) <<

- C use
    - Here is another C use which takes the pointer to prtplayerA/B as well as GAME_LENGTH and doing math to then std::setw them again again

Reflections:

Gavin: I was in charge of determining the DEF and USE nodes. In this program there are 7 variables: playerA, playerB, srand, obstaclProb, GAME_LENGTH, ptrplayerA, and ptrplayerB. The DEF nodes were determined by examining when contents of the memory location for the variables were changed. The USE nodes were determined by examining when these variables were used in the program for things such as loop controls, conditional statements, and output statements. There are two pointers used: ptrplayerA and ptrplayerB. However, we can treat these as normal variables when we are determining DEF and USE nodes because any change to the pointer is reflected in the original variable.

Sean: I was in charge of the Definition-Coverage Path and Definition-Use Path. For Definition-Coverage Path I broke it up into the three functions advancePlayerA, advancePlayerB, and printFunction. Going through each function I analyzed each condition and the different paths each condition led to. For the Definition-Use Path I analyzed the variables playerA, playerB, obstaclProb (in advncePlayerA), obstaclProb (in advancePlayerB) GAME_LENGTH, ptrplayerA, and ptrplayerB. Then I checked where each variable was defined and all the places it was used in the code, determining the paths for each use.

Aman: I was in charge mainly of the usage nodes including the C use and the P use nodes. Here we had to look through the code to see each time that a variable was accessed and enter it into the file. Once we had all of the variables into the file and when they were being used, we could look at the type of usage being described to see whether or not it is a P usage or a C usage. Here we found that many of the variables being used were P usages and we labeled those accordingly.

Once both the C usage nodes and the P usage nodes were correctly labeled, I went through each one and wrote a short description on exactly how it was actually being used and whether or not the variable was being accessed or modified.