CSCI 3901 Lab 3: Testing

Members:

Benny Tharigopala [B00899629]

Nadipineni Hemanth Kumar [B00899473]

PART - 1:

1) Ambiguities about the problem statement:

- The number of columns to display "number of games won/lost/tied" is '2' and that for "points scored by/against the team" is '4'. Also, it was mentioned that one space between each column. When the first three columns take just single digits, there will be two spaces, which is contradictory to the other point.
- What's the maximum number of games that a team can play? What's the stopping condition?
- Can two teams play against each other more than once?

2) Boundary conditions that exist in the problem statement:

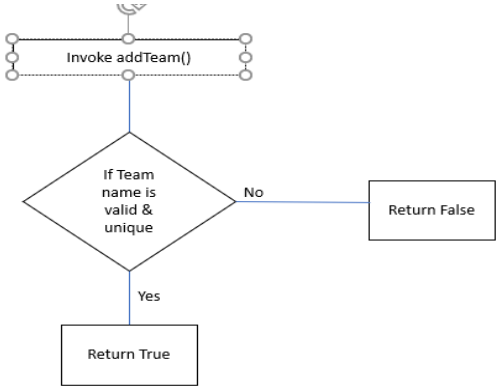
- The number of teams in the league shouldn't exceed 24.
- Team names should not exceed 15 characters, assuming that each column represents a character.
- Maximum number of games Won / Lost is 99, since, values for number of games Won, Lost and Tied, each, shouldn't exceed 2 characters.
- Maximum points that can be Won / Lost by a team is 9999, since, values for points scored by and against a team, each, shouldn't exceed 4 characters.

3) The types of control flow needed:

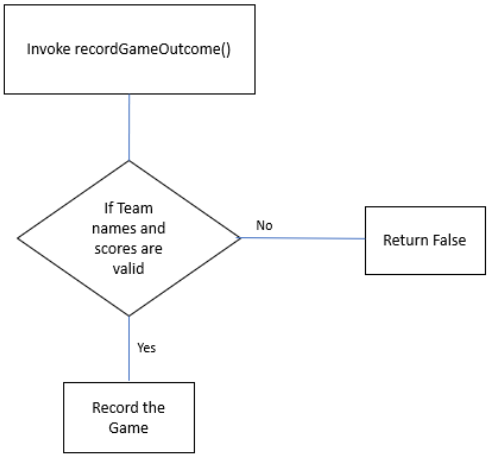
- Branching is necessary to determine the winning team.
- Looping is required to display the scoreboard.
- Condition-controlled loops, count-controlled loops can be used to make sure the league has less than 24 teams as mentioned in the statement.
- The core logic of any implementation would follow the "**Selection**" control-flow. Various values associated with each team (Games Won, Games Lost, Games Drawn) are compared **(with conditional statements)** to determine which team has a higher rank in the leaderboard.

  Control Flow

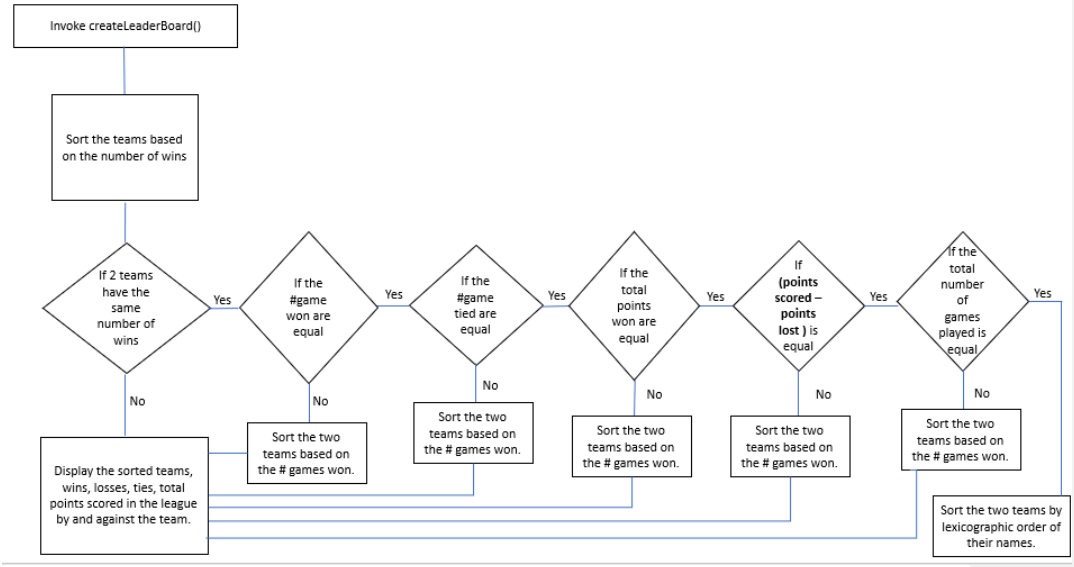
A. Add teams to the league



B. Record a match between two teams



C. Display the Leaderboard



4) Normal Order:

PART - 2:

1) Input validation tests:

1. addTeam()
    - Null value passed as teamName
    - Empty string passed as teamName
2. recordGameOutcome()
    - Null value passed as Team name
    - Empty value passed as Team name
    - team1's name is identical to team2's name
    - A team's score is negative.
    - A team's score is greater than **2,147,483,647**
    - Both team's scores are negative.
    - Both team's score are greater than **2,147,483,647**

2) Boundary tests

1. addTeam()
    - Single character team name
    - 15-characterd team name
    - Add a team when 24 teams exist
2. recordGameOutcome()
    - Team 1's score is 0
    - Team 1's score is 0
    - Team 2's score is 9999
    - Team 2's score is 9999
3. createLeaderBoard()
    - No games were recorded
    - One game was recorded

3)Control flow tests

1. addTeam()
    - Add team when no teams exist
    - Add team when one team exists
    - Add team when many teams exist

2. recordGameOutcome()
    - Record a game when no teams exist

Commented [BT1]: Maximum value that a team can take for all games is 9999. So why bother with int's max value?

Commented [HN2R1]: Any number entered greater than this value will become negative. Just to make sure we reject them at input

Commented [BT3]: Maximum value that a team can take for all games is 9999. So why bother with int's max value?

Commented [HN4R3]: Any number entered greater than this value will become negative. Just to make sure we reject them at input

- Record a game when one game exists
- Record a game when many teams exist
- Record a game between the same teams
- Record a game between two teams having played a game previously
- Record a game with one 0 score
- Record a game with both 0 scores
- Record a game with one 9999 score
- Record a game with both 9999 scores
- scoreTeam1 > scoreTeam2
- scoreTeam1 = scoreTeam2
- scoreTeam1 < scoreTeam2
- Record a game with the same teams and the same score
- Record a game with two teams having played previously and having the same score
- Record a game with one invalid team name and one valid team name
- Record a game with two invalid team names
- Record a game with one invalid team name and one invalid score
- Record a game with two invalid team names and two invalid scores

3. createLeaderBoard()
    - No teams were defined
    - 24 teams were defined
    - 0 number of games won for a team
    - 0 number of games won for all teams
    - 0 number of games lost for a team
    - 0 number of games lost for all teams
    - 0 number of games tied for a team
    - 0 number of games tied for all teams
    - 24 games won for a team
    - 24 games lost for a team
    - 24 games tied for all teams (All ties)
    - 0 points scored by a team
    - 0 points scored by all teams
    - 9999 points scored by a team (cumulative)
    - 9999 points scored by all teams (cumulative)
    - 9999 points were scored against a team (cumulative)
    - 9999 points were scored against a team (cumulative)

4)Data flow tests

1. createLeaderBoard()
    - Two teams have the same number of matches won in the league.
    - Two teams have the same number of matches won and matches tied in the league.
    - Two teams have the same number of matches won, matches tied and cumulative points in the league.
    - Two teams have the same number of matches won, matches tied, cumulative points and highest difference of points scored to points lost in the league
    - Two teams have the same number of matches won, matches tied, cumulative points, highest difference of points scored to points lost and number of games played in the league
    - Two teams have the same name, number of matches won, matches tied, cumulative points, highest difference of points scored to points lost and number of games played in the league

Completeness of the tests:

Commented [BT5]: Please feel free to add content to this section.

Commented [BT6R5]:

*To be completed at the end

The aforementioned tests address most, if not all, possible variants of the input data. The tests primarily cover the following scenarios:

1. Incorrect input is passed to methods
2. Appropriate input is passed to methods
3. Input which results in an ambiguous endpoint

Answering questions:

1)user input can be validated while typing in the user interface using JS/jQuery event handlers. But parameters are to be passed after typing the entire input.

When inputs are passed as parameters to methods the validation is handled in the method whereas if input is obtained from an user interface the validation can be handled immediately after scanning the input. In the first approach, error handling is part of the function block and in the latter error handling is separated from business logic. Therefore, input validation and feedback on input is faster when inputs are obtained through an user interface.

2)Boundary cases still exist even after checking the incoming input values. For example,

Boundary test cases do not exist beyond checking the incoming input value. The whole purpose of boundary cases is to reduce the number of scenarios to a finite number of testable cases.

Values at the extreme ends of input domain can be first validated to ensure that scripts work as expected before moving on to values at the center of the input range.

3)In case of white box testing, it is more focused on conditions, branches and structure. Therefore, control flow goes towards looping and branching, whereas black-box testing is more focused on the behavior and functionality of the application.

Black box testing analyzes whether an application meets the requirements or external expectations. The control flow relevant to logic of the application isn't validated. Rather, the functionality of the application is validated by comparison of the expected output with the actual output.

Whereas in the case of White Box testing, the level of granularity is higher. The control flow of an application is validated by testing the path, conditions along with a data domain.

4)In a way, it does result in data flow tests. But permutations create more redundant test cases where one would be testing with the same type of tests.

All permutations of methods should result data flow tests. The objective of Data Flow tests is to analyze all paths is a program to identify situations where the data flow of the program is inappropriate. If only a subset of all possible permutations of methods is used in data flow tests, inadvertent situations, relevant to permutations which weren't tested, might be missed. This eventually affects the functionality of the program. Therefore all possible permutations of methods must be addressed through data flow tests.

References:

1. Clean Code - Robert C. Martin
2. What is Data Flow Testing?