**Report: Programming Project 5**

1. Test Cases
   1. **Test case 1:**

cout.setf(ios::boolalpha);

Gym Bfit( "Bfit", "Los Angeles", REGULAR );

Bfit.setFeatures( true, true, true, true );

GymMember executive( "Preyasi", "223344", EXECUTIVE );

executive.startNewMonth();

cout << Bfit.checkin( executive, true, false, false, false );

cout << Bfit.checkin( executive, true, false, false, false );

cout << executive.workoutsThisMonth();

executive.startNewMonth();

cout << executive.workoutsThisMonth();

**Expected Output: true true 2 0**

**Actual Output: true true 2 0**

Reasoning: To test whether the function startNewMonth() works, and resets the workout count to zero.

* 1. **Test Case 2**  
     cout.setf(ios::boolalpha);

Gym Bfit( "Bfit", "Los Angeles", REGULAR );

Bfit.setFeatures( true, true, false, true );

GymMember executive( "Preyasi", "223344", EXECUTIVE );

executive.startNewMonth();

cout << Bfit.checkin( executive, true, false, true, false );

cout << Bfit.checkin( executive, true, false, false, false );

cout << executive.workoutsThisMonth();

**Expected Output: false true 1**

**Actual Output: false true 1**

Reasoning behind the test: To check whether the checkin increments only when the features are available, and to test the functionality of checkin.

* 1. **Test Case 3:**

cout.setf(ios::boolalpha);

Gym Bfit( "Bfit", "Los Angeles", REGULAR );

Bfit.setFeatures( true, true, false, true );

GymMember executive( "Preyasi", "223344", EXECUTIVE );

cout << executive.getName ();

cout << executive.getAccountNumber ();

**Expected Output: Preyasi223344**

**Actual Output: : Preyasi223344**

Reasoning behind the test: To check the functions string getName() and the functions getAccountNumber(), and see whether they return the correct value.

* 1. **Test Case 4:**

cout.setf(ios::boolalpha);

Gym Bfit( "Bfit", "Los Angeles", REGULAR );

Bfit.setFeatures( true, true, false, true );

GymMember executive( "Preyasi", "223344", EXECUTIVE );

GymMember Aashu( "Aashu", "001", REGULAR );

cout << executive.getName ();

cout << executive.getAccountNumber ();

cout << Aashu.getName ();

cout <<Aashu.getAccountNumber ();

**Expected Output: Preyasi223344Aashu001**

**Actual Output: : Preyasi223344Aashu001**

Reasoning behind the test: To check the functions string getName() and the functions getAccountNumber() when there are different members, and to check any overlap problems.

* 1. **Test Case 5:**

cout.setf(ios::boolalpha);

Gym Bfit( "Bfit", "Los Angeles", REGULAR );

Bfit.setFeatures( true, false, false, true );

GymMember executive( "Preyasi", "223344", EXECUTIVE );

cout <<Bfit.hasCardioFeatures();

cout << Bfit.hasWeightFeatures();

**Expected Output: truefalse**

**Actual Output: : truefalse**

Reasoning behind the test: To check the functions hasCardioFeatures and hasWeightsfeatures in the class Gym, and output them as a bool.

* 1. **Test Case 6:**

cout.setf(ios::boolalpha);

Gym Bfit( "Bfit", "Los Angeles", REGULAR );

Bfit.setFeatures( true, false, false, true );

GymMember executive( "Preyasi", "223344", EXECUTIVE );

cout <<Bfit.hasPoolFeatures();

cout << Bfit.hasTrackFeatures();

**Expected Output: falsetrue**

**Actual Output: : falsetrue**

Reasoning behind the test: To check the functions hasPoolFeatures and hasTrackfeatures in the class Gym, and output them as a bool.

* 1. **Test Case 7:**

cout.setf(ios::boolalpha);

Gym Bfit( "Bfit", "Los Angeles", REGULAR );

Bfit.setFeatures( true, false, false, true );

GymMember executive( "Preyasi", "223344", PREMIER );

cout << executive.getKindAsString ();

**Expected Output: “PREMIER”**

**Actual Output: : “PREMIER”**

Reasoning behind the test: To check the getKindAsString function, and to check if it is able to print the GymMember Kind correctly as a string.

* 1. **Test Case 8:**

cout.setf(ios::boolalpha);

Gym Bfit("Bfit", "Los Angeles", REGULAR );

Bfit.setFeatures( true, true, false, true );

GymMember executive( "Preyasi", "223344", EXECUTIVE );

cout << Bfit.canWorkoutHere(executive);

cout << executive.workoutsThisMonth();

**Expected Output: true 0**

**Actual Output: true 0**

Reasoning behind the test: To check the working of the canWorkoutHere function, and to test whether a member checking the features of the Gym increments the workouts that month or not.

* 1. **Test case 9:**

Input:

Gym exec( "ExecGym", "Area 51", EXECUTIVE );  
GymMember executive( "James Bond", "007", EXECUTIVE );  
GymMember premier( "Sidhaant", "-infinity", OTHER );  
GymMember exectives( "Terence Tao", "005", REGULAR );  
assert( exec.canWorkoutHere( executive ) == false );  
assert( exec.canWorkoutHere( premier ) == false );  
assert( exec.canWorkoutHere( executive ) == true );

Output: **Assertion failed: (exec.canWorkoutHere( executive ) == false)**

Reason: Checks if an executive member can check in at an executive and returns the correct bool value.

* 1. **Test case 10:**

Input:

Gym exec( "ExecGym", "Los Angeles", EXECUTIVE );  
GymMember executive( "You", "-500", EXECUTIVE );  
GymMember premier( "Me", "Yayyy", PREMIER );  
GymMember regular( "And her", "This is NOT A NUMBER!!", REGULAR );  
assert( exec.canWorkoutHere( executive ) == true );  
assert( exec.canWorkoutHere( premier ) == false );  
assert( exec.canWorkoutHere( regular ) == false );  
Reason: To check whether the account number was taken in as a string or a number. If it were taking number values for the account number, it would have shown a compile error for “Yayyy” and “This is NOT A NUMBER!!”

* 1. **Test case 2:**

Input:

cout.setf( ios::boolalpha );

    Gym exec( "ExecGym", "Los Angeles", EXECUTIVE );

    Gym regular( "ExecGym", "Los Angeles", EXECUTIVE );

    GymMember executive( "You", "67890", EXECUTIVE );

    GymMember premier( "Me", "11111", PREMIER );

    GymMember regulars( "Someone Else", "22222", REGULAR );

    cout << exec.canWorkoutHere( premier );

    cout << regular.canWorkoutHere( regulars );

    cout <<  exec.canWorkoutHere(executive) ;

    cout << executive.workoutsThisMonth();

    executive.startNewMonth ();

    cout << executive.workoutsThisMonth();

Output :  **00100**

Reason: To check whether the canWorkout function works and workoutsThisMonth. It also checks the if the workouts count resets after we call the function startNewMonth.