1. A brief description of notable obstacles you overcame

I had some compile errors on codeboard because I accidentally named one of my headers as FrequentFlyerAcccount.h . After fixing that I had errors about the placement of #endif because I had forgotten to place it.

1. A list of the test data that could be used to thoroughly test your functions, along with the reason for each test. You must note which test cases your program does not handle correctly. (This could happen if you didn't have time to write a complete solution, or if you ran out of time while still debugging a supposedly complete solution.) Notice that most of this portion of your report can be written just after you read the requirements in this specification, before you even start designing your program.

Note: because my constructor for FrequentFlyerAccount and PlaneFlight and my getter and setter funcitons ignore empty strings. However, because of this if strings are not initialized, they will be empty strings. My addFreeflightToAccount will return false if any string is the empty string.

Note: there will be no test cases for getter functions

**CLASS: PlaneFlight**

The following account will be used for all PlaneFlight Class test cases:

FrequentFlyerAccount account("Howard");

**Test Case 1:**

**Reason: a valid cost must be a value of 0 or more.**

PlaneFlight Flight00(“Howard” , “LAX” , “LAS”, 0 , 3000);

assert( Flight00.getCost() == 0);

PlaneFlight Flight01(“Howard” , “LAX” , “LAS”, 2000 , 3000);

assert( Flight01.getCost() == 2000);

**Test Case 2:**

**Reason: A PlaneFlight should store the value -1 to indicate when an invalid flight cost was attempted to be stored.**

PlaneFlight Flight001(“Howard” , “LAX” , “LAS”, -100 , 3000);

assert( Flight001.getCost() != -100);

assert( Flight001.getCost() == -1);

PlaneFlight Flight002(““ , “LAX” , “LAS”, -100 , 3000);

assert( Flight002.getCost() != -100);

assert( Flight002.getCost() == -1);

PlaneFlight Flight003(““ , ““ , ““, -100 , 0);

assert( Flight003.getCost() != -100);

assert( Flight003.getCost() == -1);

**Test Case 3:**

**Reason:** **empty string values are not valid for either a FromCity or a ToCity value.** **The specified FromCity and ToCity must be different. A PlaneFlight should ignore and not accept an invalid FromCity or ToCity value.**

PlaneFlight Flight1(“Howard”, ““ , ““, -100 , 3000);

assert(account.addFlightToAccount(Flight1) == false);

PlaneFlight Flight2(““, “LAX“ , “LAS”, -100 , 3000);

assert(account.addFlightToAccount(Flight2) == false);

assert(Flight2.getToCity() == “LAS”);

PlaneFlight Flight3(“Howard”, “LAS” , “LAS”, -100 , 0);

assert(Flight3.getFromCity() != “LAS”);

assert(Flight3.getToCity() != “LAS”);

PlaneFlight Flight4(““, “LAX” , “LAS”, -100 , 3000);

assert(Flight4.getFromCity() == “LAX”);

assert(Flight4.getToCity() == “LAS”);

assert(account.addFlightToAccount(Flight4) == false);

PlaneFlight Flight5(“Howard”, “LAX#123a” , “LAS”, -100 , 3000);

assert(Flight5.getFromCity() == “LAX#123a”);

assert(Flight5.getToCity() == “LAS”);

assert(account.addFlightToAccount(Flight5) == false);

**Test Case 4:**

**Reason: the empty string is not valid for a passenger's Name. A PlaneFlight should ignore and not accept an invalid Name value.**

PlaneFlight Flight6(““, “LAX” , “LAS”, 100 , 3000);

assert(Flight6.getToCity() == “LAS”);

assert(Flight6.getMileage() == 3000);

assert(Flight6.getCost() == 100);

assert(account.addFlightToAccount(Flight6) == false);

PlaneFlight Flight7(“ASDFGHJK!#123”, “LAX” , “LAS”, -100 , -3000);

assert( Flight7.getName() == “ASDFGHJK!#123”);

**Test Case 5:**

**Reason: a valid mileage must be a value greater than 0. A PlaneFlight should store the value -1 to indicate when an invalid mileage amount was attempted to be stored.**

PlaneFlight Flight8(“mil”, “LAX” , “LAS”, -100 , 3000);

assert( Flight8.getMileage() == 3000);

PlaneFlight Flight9(““, ““ , ““, -100 , 3000);

assert( Flight9.getMileage() == 3000);

PlaneFlight Flight10(““, ““ , ““, -100 , 0);

assert( Flight10.getMileage() != 0);

assert( Flight10.getMileage() == -1);

PlaneFlight Flight11(““, ““ , ““, -100 , -100);

assert( Flight11.getMileage() != -100);

assert( Flight11.getMileage() == -1);

**Test Case 6:**

**Reason: user must not be able to set name to “”**

PlaneFlight Flightget1(“Howard”, “LAX”, “LAS”, 10, 100);

Flightget1.setName(“”);

assert( Flightget1.getName() == “Howard”);

**Test Case 7:**

**Reason: if the user attempts to set mCost to some number <0 then -1 must be stored in mCost**

PlaneFlight Flightget2(“Howard”, “LAX”, “LAS”, 100, 100);

Flightget2.setCost(-100);

assert( Flightget2.getCost() == -1);

**Test Case 8:**

**Reason: if the user attempts to set mMileage to some number <=0 then -1 must be stored in mMileage**

PlaneFlight Flightget3("Howard", "LAX", "LAS", 100, 1000);

Flightget3.setMileage(0);

assert(Flightget3.getMileage() == -1);

Flightget3.setMileage(100);

assert(Flightget3.getMileage() == 100);

Flightget3.setMileage(-100);

assert(Flightget3.getMileage() == -1);

**Test Case 9:**

**Reason: user must not be able to set FromCity or ToCity to the same string or the empty string**

PlaneFlight Flightget4("Howard", "LAX", "LAS", 100, 1000);

Flightget4.setFromCity("LAS");

Flightget4.setToCity("LAX");

assert(Flightget4.getToCity() == "LAS");

assert(Flightget4.getFromCity() == "LAX");

**CLASS: FrequentFlyerAccount**

**Test Case 10:**

**Reason: In the beginning of time, the balance should start at zero**

FrequentFlyerAccount account1("Howard");

assert(account1.getBalance() == 0);

FrequentFlyerAccount account2("");

assert(account2.getBalance() == 0);

**Test Case 11:**

**Reason: For .addFlightToAccount to return true and update the balance, the names must match between the flight and the account, the flight's cost and its mileage must be bigger than zero, the city values must be matching and not empty. In any other situation, return false and don't update the balance.**

PlaneFlight f1("Howard", "LAX", "LAS", 100, 3000);

assert(account3.addFlightToAccount(f1) == true);

PlaneFlight f2("", "LAX", "LAS", 100, 3000);

assert(account3.addFlightToAccount(f2) == false);

PlaneFlight f3("Howarda", "LAX", "LAS", 100, 3000);

assert(account3.addFlightToAccount(f3) == false);

PlaneFlight f4("Howard", "LAS", "LAS", 100, 3000);

assert(account3.addFlightToAccount(f4) == false);

PlaneFlight f5("Howard", "", "LAS", 100, 3000);

assert(account3.addFlightToAccount(f5) == false);

PlaneFlight f6("Howard", "LAX", "", 100, 3000);

assert(account3.addFlightToAccount(f6) == false);

PlaneFlight f66("Howard", "", "", 100, 3000);

assert(account3.addFlightToAccount(f66) == false);

PlaneFlight f7("Howard", "LAX", "LAS", -100, 3000);

assert(account3.addFlightToAccount(f7) == false);

PlaneFlight f8("Howard", "LAX", "LAS", 0, 3000);

assert(account3.addFlightToAccount(f8) == false);

PlaneFlight f9("Howard", "LAX", "LAS", 100, 0);

assert(account3.addFlightToAccount(f9) == false);

PlaneFlight f10("Howard", "LAX", "LAS", 100, -100);

assert(account3.addFlightToAccount(f10) == false);

**Test Case 12:**

**Reason: For .canEarnFreeFlight to return true, the mileage must be bigger than zero and not exceed the account balance. In any other situation, return false.**

FrequentFlyerAccount account4("Howard");

PlaneFlight f11("Howard", "LAX", "LAS", 1000, 499);

account4.addFlightToAccount(f11);

assert(account4.canEarnFreeFlight(500) == false);

assert(account4.canEarnFreeFlight(0) == false);

PlaneFlight f12("Howard", "LAX", "LAS", 10, 1);

account4.addFlightToAccount(f12);

assert(account4.canEarnFreeFlight(500) == true);

account4.addFlightToAccount(f12);

assert(account4.canEarnFreeFlight(500) == true);

**Test Case 13:**

**Reason: For .freeFlight to return true, the city values must not match or be empty and the mileage must be bigger than zero and not exceed the account balance. If all this holds true, update the PlaneFlight parameter and reduce the account balance accordingly. In any other situation, return false and don't update the PlaneFlight parameter at all.**

**Note: since we are creating the desired flight we do not check the passenger name parameter.**

FrequentFlyerAccount account5("Howard");

PlaneFlight F("Howard", "LAX", "LAS", 100, 1000);

PlaneFlight f13("Howard", "LAX", "LAS", 100, 1000);

PlaneFlight f14("", "LAX", "LAS", 100, 1000);

PlaneFlight f15("Howard", "", "LAS", 100, 1000);

PlaneFlight f16("Howard", "LAX", "", 100, 1000);

PlaneFlight f17("Howard", "LAX", "LAS", -100, 1000);

PlaneFlight f18("Howard", "LAX", "LAS", 0, 1000);

PlaneFlight f19("Howard", "LAX", "LAS", 100, 0);

PlaneFlight f20("Howard", "LAX", "LAS", 100, -110);

account5.addFlightToAccount(f13);

assert(account5.canEarnFreeFlight(500) == true);

assert(account5.freeFlight("CANADA", "USA", 500, f13) == true);

assert(account5.freeFlight("CANADA", "USA", 0, f13) == false);

assert(account5.freeFlight("CANADA", "USA", -100, f13) == false);

assert(account5.freeFlight("CANADA", "CANADA", 500, f13) == false);

assert(account5.freeFlight("CANADA", "", 500, f13) == false);

assert(account5.freeFlight("", "CANADA", 500, f13) == false);

assert(account5.freeFlight("CANADA", "USA", 500, f14) == true);

account5.addFlightToAccount(F);

assert(account5.freeFlight("CANADA", "USA", 500, f15) == true);

account5.addFlightToAccount(F);

assert(account5.freeFlight("CANADA", "USA", 500, f16) == true);

account5.addFlightToAccount(f13);

assert(account5.freeFlight("CANADA", "USA", 500, f17) == true);

account5.addFlightToAccount(F);

assert(account5.freeFlight("CANADA", "USA", 500, f18) == true);

account5.addFlightToAccount(F);

assert(account5.freeFlight("CANADA", "USA", 500, f19) == true);

account5.addFlightToAccount(F);

assert(account5.freeFlight("CANADA", "USA", 500, f20) == true);