Project 5 Report

* Notable obstacles
  + When the free flight’s to/from are the previous’ flight’s from/to (reverse order)

Originally, this failed because the program does not allow for the to and from cities to equal at a given time. To fix this, I would use placeholders to reset the to and from cities set functions. The placeholders are not equal to each other to work.

**if** (from != to) { //check if different

string p1 = "p1"; //placeholder 1

string p2 = "p2"; //placeholder 2

flight.setToCity(p1); //reset to city

flight.setFromCity(p2); //reset from city

**if** (mBalance >= mileage) { //enough mileage to create a free flight

b = **true**;

mBalance = mBalance - mileage; //subtract flight from balance

flight.setCost(0); //0 for cost bc free

flight.setToCity(to); //to city of free flight

flight.setFromCity(from); //from city of free flight

* Test data

//my test code

PlaneFlight p("Sally", "LAX", "PH", 50.00, 100); //used to test planeFlight

PlaneFlight n("Sally", "LAX", "PH", 50.00, 100); //used to test FrequentFlyerAccount

PlaneFlight wrong("Sally", "LAX", "LAX", 0.00, 10); //to=from is wrong, cost 0 to check add flight

FrequentFlyerAccount acc("Sally");

FrequentFlyerAccount notacc("notSally");

//planeFlight

//set and get cost

assert(std::to\_string(p.getCost()) == "50.000000"); //get initial value

p.setCost(120.00); //new value

assert(std::to\_string(p.getCost()) == "120.000000"); //get new value

p.setCost(-100.00); //invalid negative value

assert(std::to\_string(p.getCost()) == "-1.000000"); //-1 for invalid values

p.setCost(0.00); //can be 0

assert(std::to\_string(p.getCost()) == "0.000000");

//set and get mileage

assert(std::to\_string(p.getMileage()) == "100.000000"); //get initial value

p.setMileage(120.00); //new value

assert(std::to\_string(p.getMileage()) == "120.000000"); //get new value

p.setMileage(-100.00); //invalid negative value

assert(std::to\_string(p.getMileage()) == "-1.000000"); //-1 for invalid values

p.setMileage(0.00); //cannot be 0

assert(std::to\_string(p.getMileage()) == "-1.000000"); //-1 for invalid values

//set and get name

assert(p.getName() == "Sally"); //get initial name

p.setName("Cass"); //new name

assert(p.getName() == "Cass"); //get new name

p.setName(""); //empty string

assert(p.getName() == "Cass"); //should ignore empty string and not change

//set and get from city

assert(p.getFromCity() == "LAX"); //get initial from city

p.setFromCity("SF"); //new city

assert(p.getFromCity() == "SF"); //get new city

p.setFromCity(""); //empty string

assert(p.getFromCity() == "SF"); //should ignore empty string and not change

p.setFromCity("PH"); //same as to city

assert(p.getFromCity() == "SF"); //should ignore if same and not change

assert(wrong.getFromCity() == "?from?"); //same to and from cities, bad string

//set and get to city

assert(p.getToCity() == "PH"); //get initial to city

p.setToCity("SD"); //new city

assert(p.getToCity() == "SD"); //get new city

p.setToCity(""); //empty string

assert(p.getToCity() == "SD"); //should ignore empty string and not change

p.setToCity("SF"); //same as from city (from city was set to SF earlier)

assert(p.getToCity() == "SD"); //should ignore if same and not change

assert(wrong.getToCity() == "?to?"); //same to and from cities, bad string

//Frequent flyer account

//initial account info

assert(acc.getName() == "Sally"); //get name

assert(std::to\_string(acc.getBalance() ) == "0.000000"); //initial balance is 0

assert(acc.canEarnFreeFlight( 100.00 ) == **false** ); //false since initial balance is 0

//add flight to account

assert(acc.addFlightToAccount(n) == **true** ); //name match

assert(std::to\_string(acc.getBalance( ) ) == "100.000000" ); //updated balance

assert(notacc.addFlightToAccount(n) == **false** ); //mismatch name

assert(acc.addFlightToAccount(wrong) == **false** ); //don't add since cost is 0

//can earn free flight

assert(acc.canEarnFreeFlight(50) == **true**); //balance exceeds threshold

assert(acc.canEarnFreeFlight(100) == **true**); //balance equals threshold

assert(acc.canEarnFreeFlight(1000) == **false**); //balance below threshold

//free flight

assert(acc.freeFlight("A", "B", 50, n) == **true** ); //name match and balance can cover mileage

assert(std::to\_string(acc.getBalance() ) == "50.000000"); //subtract mileage used from balance

assert(std::to\_string( n.getCost( ) ) == "0.000000" ); //0 cost for free flight

assert(n.getFromCity() == "A"); //from city for free flight

assert(n.getToCity() == "B"); //to city for free flight

assert(std::to\_string(n.getMileage()) == "50.000000"); //mileage of free flight

n.setCost(50.00); //reset for next test

assert(acc.freeFlight("C", "D", 1000, n) == **false**); //name match but balance can't cover mileage

assert(std::to\_string(acc.getBalance() ) == "50.000000"); //does not change (became 50 earlier)

assert(std::to\_string( n.getCost( ) ) == "50.000000" ); //doesnt change

assert(n.getFromCity() == "p2"); //placeholder only bc didnt get free flight

assert(n.getToCity() == "p1"); //placeholder only bc didnt get free flight

assert(std::to\_string(n.getMileage()) == "50.000000"); //doesnt change bc didnt get free flight

assert(notacc.freeFlight("A", "B", 100, n) == **false** ); //mismatch name

assert(acc.freeFlight("A", "A", 100, n) == **false** ); //to and from same