Functionality Outline

Airport Terminal Simulation
Programming Assignment 1

28 February 2021

Prepared By
Nolan Anderson
npa0002@uah.edu

Prepared for
Dr. Jacob Hauenstein
CS 307, Object Oriented Programming
Computer Science Department
University of Alabama in Huntsville

1.0 System Overview

To solve the problem of parsing XML files comprised of a large number of different flights (consists of departure time and city, destination city, aircraft type etc.) requires the creation of multiple classes. This software will largely be ran by the class outlined in section 3.0, with the following classes (section 4.0-6.0) parsing the data. In other words, sections 4.0-6.0 will provide section 3.0 with the data it needs to do its calculations and output. Mostly, sections 4.0-6.0 call the parsing functions and return data and they are much simpler than section 3.0. The output of the software is simple. It outputs all of the data of each flight every 5 seconds. This data includes the flight number, aircraft name and type, departure time and city, destination city among many other different data points. In summary, the main functionality of the program is to send out flight data. If you were the flight controller for every flight in the United States, this would provide you with every flight that is going out or currently in transit.

2.0 **Relevant Terms and Acronyms**

Class – This is a user-defined data type that we can use to hold member variables and functions.
Member Variables – The pieces that make up a class.
Member Functions – The "doers" of the class, as in they perform the calculations, output etc.
InFile – A standard name used for input file names.
N/a – This thing or value does not exist and is not needed in this function.
Void – A function type that does not need to return a value. Usually a function that creates, or performs output.

3.0 Flight Simulator Executor

3.1 FlightSim.cpp

3.1.1 - FlightSim::FlightSim();

Initializes and starts the flight simulation.
Displays a message that the simulation is starting.
Call FlightSim::SetMultiplier
Call FlightSim::SetInFile
Call FlightSim::Start

3.1.2 - ~FlightSim::FlightSim();

Clears all array's and values created by running the simulation.

3.1.3- int SetMultiplier();

Prompts the user to select a time multiplier for the program.
"Please enter your desired time multipler: 1, 2, or 3."

3.1.4 - string SetInFile();

Prompts the user to input a file name. This will be stored as a string and used later in the objects, classes, and their functions that follow.
"Please enter a file name:"

3.1.5 - void Start(int Multiplier, string InFile);

This is the main function that actually starts the program.
Calls CityData::CityData // Initialize the data, store into class struct.

Calls FlightData::FlightData
Calls AircraftData::AircraftData
Populate a list of flight numbers and their departure times.
Call FlightData::ReturnFlightNum
Start the global timer

3.1.6 - int CurrentLocation(int FlightNum, int CurrentHr, int CurrentMin);

Using flight num to call 6.16 and 6.17 for speed.
CurrentHr and CurrentMin tells us the current time of the simulator.
Use the speed and current hour to calculate the x and y positon.

3.1.7 - int FlightTime(int FlightNum);

Call 5.16, 5.17 to get 2 cities.
Call 4.16, 4.17 to get city locations
Call 4.18 to get the distance between them
Call 6.16 and 6.17 to calculate speed.
Use the speed and distance to calculate the flight time.

3.1.8 - void OutNewFlight(int FlightNum);

Searches for the correct flight number in the list of data
If it is found:
Output new flight information:
All of this information will be in a struct for the FlightSim
Call 3.17 to get the total flight time
Add this to the current scenario time to get ETA.
scenario clock time, the airline, flight number, type of aircraft, the departure city and state, the arrival city and state, and the estimated time of arrival

3.1.9 - void OutInterval(int FlightNum);

Searches for the correct flight number in the list
Outputs the data corresponding to that flight number
Call CurrentLocation to calculate location of flight
Calculate the distance from the current location to the destination city
Calculate the time based on the distance (flight speed from FlightNum)
current scenario clock time and information on all flights currently in route. Flight information shall include the airline, flight number, type of aircraft, the departure city symbol, latitude and longitude, the time of departure, the arrival city symbol, latitude and longitude, the estimated time of arrival, the current location of the aircraft in latitude and longitude, the number of miles from the departure city, the number of miles to the destination city, the cruise speed of the aircraft, and the current altitude.

There will be get and set functions for each private member variable of the class.

4.0 Populating City Data from XML Files.

4.1 CityData.cpp

4.1.1 - CityData::CityData(string InFile);

Creates a new instance of the Aircraft data.
Call 4.1.3 to Set the data for the class.

4.1.2 - ~CityData::CityData();

Deconstructor for the aircraft data, will clear the data parsed and stored in the class.

4.1.3 - void SetData(string InFile);

Creates struct of data for all of the City Data information
Call InitCityData(InFile)
Assign i to getCityCount();
Loop through i
    Assign array of City Symbols to the array of city symbols
    Loop through i again
        Call getDistTable and assign array of distances accordingly
    End
End
Now that we have a list of city symbols
Loop through i again
    Assign the rest of the structs data with i.
    It will look similar to Struct.City[i] = "City"
    Struct.State[i] = "State"
    Etc.. for all values in the struct.

Now we have all information on our city data.

4.1.4 - string ReturnName(int FlightNum);

If flightnum is found in list of City Names
    Return the City Name that corresponds to this flightnum

4.1.5 - string ReturnState(string CityName);

If the CityName is found in list of State names
    Return the State name that corresponds to the CityName

4.1.6 - float ReturnLatitude(string CityName);

If the CityName is found in List of Latitudes
    Return the Latitude of the corresponding city

4.1.7 - float ReturnLongitude(string CityName);

If the CityName is found in list of Longitudes
    Return the Longitude of the corresponding city

4.1.8 - float ReturnDistance(int FlightNum);

If the FlightNum is found in list of Distances
    Return the flight distance corresponding to the Flight Number.

There will be get and set functions for each private member variable of the class.

5.0 Populating Flight Data from XML files.

5.1 FlightData.cpp

5.1.1 - FlightData::FlightData(string InFile);

Creates a new instance of the Aircraft data.
Call 5.1.3 to Set the data for the class.

5.1.2 - ~FlightData::FlightData();

Deconstructor for the aircraft data, will clear the data parsed and stored in the class.

5.1.3 - void SetData(string InFile);

Creates struct of data for all of the Flight Data information
Call InitFlightData(InFile)
Assign i to getFlightCount();
Assign f to getAircraftCount();
Loop through i
    Assign array of flights
    Call getFlightData for each iteration of i
    Assign the data found from getFlightData to the newly created array of structs.
End
Loop through f
    Assign array of aircraft data
    Call getAircraftData for each iteration of f.
    Assign the data found from getAircraftData to the newly created array of structs.
End

Now we have all information on our city data.

5.1.4 - string ReturnAirline(int FlightNum);

If flightnum is found in list of Airlines
    Return the airlinecity that corresponds to this flightnum

5.1.5 - string ReturnType(int FlightNum);

If flightnum is found in list of Airline types
    Return the airline type that corresponds to this flightnum

5.1.6 - string ReturnDepCity(int FlightNum);

If flightnum is found in list of departure cities
    Return the departure city that corresponds to this flightnum

5.1.7 - string ReturnDestCity(int FlightNum);

If flightnum is found in list of Destination cities
    Return the destination city that corresponds to this flightnum

5.1.8 - int ReturnTime(int FlightNum);

If flightnum is found in list of flight times
    Return the flight time that corresponds to this flightnum

5.1.9 - int ReturnFlightNum(int i);

Searches through the list of flight number and returns the corresponding flight number.

There will be get and set functions for each private member variable of the class.

6.0 Parsing Aircraft Data from XML files.

6.1 AircraftData,cpp

6.1.2 Member Functions

6.1.1 - AircraftData::AircraftData(string InFile);

Creates a new instance of the Aircraft data.
Call 6.1.3 to Set the data for the class.

6.1.2 ~AircraftData::AircraftData();

Deconstructor for the aircraft data, will clear the data parsed and stored in the class.

6.1.3 - void SetData();

Create new struct for the aircraft data.
We will loop through the number of different types of aircraft found in flightdata.
I = number of aircraft found in flight data
Loop through i
    Set each struct variable
    Struct.Airline[i] = FlightDataStruct.Airline[i];
    Etc. for all values in the struct.
Now we have a list of structs with all of the different types of aircraft that will be going out.

6.1.4 - string ReturnMake(int FlightNum);

If flightnum is found in list of Makes
    Return the make that corresponds to this flightnum

6.1.5 - string ReturnModel(int FlightNum);

If flightnum is found in list of Model types
    Return the model type that corresponds to this flightnum

6.1.6 - int ReturnSpeed(string model);

If model is found in list of Model speeds
    Return the flight speed that corresponds to this model.

6.1.7 - int ReturnClimb(string model);

If model is found in list of Model Climb Speeds
    Return the climb speed that corresponds to this model.

6.1.8 - int ReturnWing(string model);

If model is found in list of Model Wing spans
    Return the wingspan that corresponds to this model.

6.1.9 - int ReturnFuselage(string model);

If model is found in list of Model Fuselage lengths
    Return the fuselage length that corresponds to this model.

There will be get and set functions for each private member variable of the class.

7.0 Flight Data Parser

For this section I will not go into the details of what each of these functions perform. They are already defined in the .cpp files provided. What is important is that the previous sections call these parser functions to set the data they need. After the data is set there is really no need for the parsers anymore, all calculations will be done by 3.0-7.0

7.1 FlightDataParser.cpp

7.1.1 FlightDataParser();

7.1.2 - void InitFlightData(const char *dataFile);

7.1.3 - void getStartTime(int *hr, int *min);

7.1.4 - int getAircraftCount();

7.1.5 - int getFlightCount();

7.1.6 - bool getAircraftData(char *make, char *desc,
double *roc, double *wngs, double *len,
double *cs, double *ca);

7.1.7 - bool getFlightData(char *airline, char *plane,
int *flNum, char *departCity, int *depHr,
int *depMin, char *destCity); // Get all data on a flight

7.1.8 - bool getNextLine(char *buffer, int n);

7.1.9 - ~FlightDataParser();

8.0 City Data Parser

For this section I will not go into the details of what each of these functions perform. They are already defined in the .cpp files provided. What is important is that the previous sections call these parser functions to set the data they need. After the data is set there is really no need for the parsers anymore, all calculations will be done by 3.0-6.0

8.1 CityDataParser.cpp

8.1.1 CityDataParser();

8.1.2 - void InitCityData(const char *dataFile);

8.1.3 - void getCityTime()

8.1.4 – bool getCityData(char *name, char *state, char *symbol, double *lat, double *lon);

8.1.5 – void getCitySymbolsArray(char ***array);

8.1.6 - bool getNextLine(char *buffer, int n);

8.1.7 - ~CityDataParser();