# PA03 - Flight Path

Generated by Doxygen 1.8.11

# Contents



# 1 Class Index

## 1.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

# 2 File Index

## 2.1 File List

Here is a list of all documented files with brief descriptions:



# 3 Class Documentation

## 3.1 City Class Reference

Collaboration diagram for City:

### Public Attributes

- string **cityName**
- bool **visited**
- City * **next**
- City * **prev**
- City * **adjData**

### Friends

- class **Map**

### 3.1.1 Detailed Description

City class that stores inputted cities

### Postcondition

The ADT is created with default values

The documentation for this class was generated from the following file:

- FlightMap.v1.cpp

## 3.2 Map Class Reference

Collaboration diagram for Map:

### Public Member Functions

- void readFlightMap (string inputCity, string inputFlight)
- void displayFlightMap ()
- void displayAllCities ()
- void displayAdjacentCities (City aCity)
- void markVisited (City &aCity)
- void unvisitAll ()
- bool isVisited (City aCity)
- void insertAdjacent (City &aCity, City adjCity)
- void getNextCity (string aCityName)
- bool isPath (City &originCity, City &destinationCity)
- bool pop ()
- void push (City &aCity)
- void test (string str1, string str2)

### Public Attributes

- City * **cityData** = NULL
- City * **cdMove** = NULL
- City * **cdSet** = NULL
- City * **fMove** = NULL
- City * **adMove** = NULL
- Pair **ezReading**

### 3.2.1 Member Function Documentation

#### 3.2.1.1 void Map::displayAdjacentCities ( City *aCity* ) [inline]

Displays a list of all the cities adjacent to the city

**Postcondition**

A list of all the cities that are adjacent to the current city

#### 3.2.1.2 void Map::displayAllCities ( ) [inline]

Displays a list of all the cities HPAir is affiliated with

**Postcondition**

A list of all the cities HPAir is affiliated with is outputted to the terminal

#### 3.2.1.3 void Map::displayFlightMap ( ) [inline]

Display all the flights

**Postcondition**

A list of all the flights is outputted to terminal

#### 3.2.1.4 void Map::getNextCity ( string *aCityName* ) [inline]

Finds the next city to fly to

**Postcondition**

Finds the next city in the flight, returning if no flights work, or the destination has been reached.

**Parameters**

| | |
|---|---|
| *aCityName* | The city the find the next flight from |

#### 3.2.1.5 void Map::insertAdjacent ( City & *aCity,* City *adjCity* ) [inline]

Inserts a city adjacent to another city

**Postcondition**

A city is created in the aCity's stack adjData

**Parameters**

| | |
|---|---|
| *aCity* | The city to have an adjacent city placed next to it |
| *adjCity* | The city adjacent to aCity |

#### 3.2.1.6 bool Map::isPath ( City & *originCity,* City & *destinationCity* ) [inline]

Calls on other functions to create a flight path

**Postcondition**

Outputs whether HPAir flies from one city to another

**Parameters**

| | |
|---|---|
| *originCity* | The original city to fly from |
| *destinationCity* | The destination to fly to |

**Returns**

A bool signalling if a flight path was successfully generated

#### 3.2.1.7 bool Map::isVisited ( City *aCity* ) [inline]

Check if a city has been visited

**Postcondition**

A city is known to have/have not been visited

**Returns**

A bool determining if the city has been visited

#### 3.2.1.8 void Map::markVisited ( City & *aCity* ) [inline]

Marks a city as visited

**Postcondition**

A city is marked as visited

#### 3.2.1.9 bool Map::pop ( ) [inline]

Removes the first node in the stack

**Postcondition**

A node is removed if the stack is not empty

**Returns**

A bool signalling if the stack is empty or not

#### 3.2.1.10 void Map::push ( City & *aCity* ) [inline]

Pushes a city into the stack

**Postcondition**

The stack has a new node. First node is created as necessary

#### 3.2.1.11 void Map::readFlightMap ( string *inputCity,* string *inputFlight* ) [inline]

Read in the data from the files

**Precondition**

Three files with data inside them

**Postcondition**

The data from the three files are read into stacks

**Parameters**

| | |
|---|---|
| *inputCity* | String of the name of the first input file |
| *inputFLight* | String of the name of the second input file |

**Note**

The third read in file has not been implemented yet.

#### 3.2.1.12 void Map::test ( string *str1,* string *str2* ) [inline]

Function to test flight between two cities without reading in data from 3rd file

**Postcondition**

Flight path is generated if possible.

#### 3.2.1.13 void Map::unvisitAll ( ) [inline]

Marks all cities as unvisited

**Postcondition**

All cities are now unvisited.

The documentation for this class was generated from the following file:

- FlightMap.v1.cpp

## 3.3 Pair Class Reference

**Private Member Functions**

- void getNamePair (string &first, string &second, ifstream &readIn)
- void getName (string &cityName, ifstream &readIn)

**Friends**

- class **Map**

### 3.3.1 Member Function Documentation

#### 3.3.1.1 void Pair::getName ( string & *cityName,* ifstream & *readIn* ) [inline], [private]

Reads in a single name on a line from a file

**Precondition**

A line contains a single city name

**Postcondition**

The name is read into a string from the file

**Parameters**

| | |
|---|---|
| *cityName* | The string to store the city name |
| *readIn* | The ifstream to read from the file |

#### 3.3.1.2 void Pair::getNamePair ( string & *first,* string & *second,* ifstream & *readIn* ) [inline],[private]

Read in a pair of city names

**Precondition**

A line of text is stored in the file

**Postcondition**

Only the names of the first two city are taken in as strings.

**Parameters**

| | |
|---|---|
| *first* | The string to store the first city name |
| *second* | The string to store the second city name |
| *readIn* | The ifstream to read from the file |

The documentation for this class was generated from the following file:

- FlightMap.v1.cpp

# 4 File Documentation

## 4.1 FlightMap.v1.cpp File Reference

Find flight path for cities.

```
#include <iostream>
#include <fstream>
```

Include dependency graph for FlightMap.v1.cpp:

## 4.2 FlightMap.v2.cpp File Reference

Find flight path for cities.

This graph shows which files directly or indirectly include this file:

#### 4.2.1 Detailed Description

Find flight path for cities.

Three files are read in, and stored into a node list. A flight path from one city to another is then found using the stack.

**Version**

1.00 Wei Tong (18 October 2016) Turned in version and initial development

**Note**

Adapted from Frank M. Carrano and Timothy M. Henry Copyright (c) 2012 Pearson Education, Hoboken, New Jersey.

### 4.3 PA03.cpp File Reference

Put the FlightMap implementation to use.

```
#include "FlightMap.v1.cpp"
#include "FlightMap.v2.cpp"
```

Include dependency graph for PA03.cpp:

**Functions**

- int **main** ()

#### 4.3.1 Detailed Description

Put the FlightMap implementation to use.

Main file for project. Will call upon functions to read in data and generate a flight path.

**Version**

1.00 Wei Tong (18 October 2016) Turned in version and initial development

**Note**

Adapted from Frank M. Carrano and Timothy M. Henry Copyright (c) 2012 Pearson Education, Hoboken, New Jersey.

# Index