PA03 - HPAir

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

Public Member Functions

- City ()

    The default constructor of a City object.
- City (string name)

    The parameterized constructor of a City object.
- ∼City ()

    The destructor of a City object.
- void SetVisitedState (bool state)

    Marks if a city has been visited.
- void SetCityName (string name)

    Changes a city's name.
- void AddDestination (string name, City ∗destinationPtr, int flightNumber, int cost)

    Adds a new detination for the city to go to.
- bool CheckIfDestination (string name)

    Checks if a city is a destination from here.
- bool CheckIfVisited ()

    Checks if this city has been visited.
- City ∗ GetDestination (string name)

    Gives the pointer to the destination sent.
- City ∗ GetUnvisitedDestination ()

    Returns an unvisted destination.
- string GetCityName ()

    Gets a city's name.
- int GetDestinationCount ()

    Gets a city's destination count.
- vector< City ∗ > GetDestinationPointers ()

    Gets a city's destination pointer array.
- void PrintCity ()

    Prints a city and its destinations.
- int PrintFlight (City ∗destinationPtr)

    Prints the flight information.
- void PrintCityToLog (ofstream ∗logFile)

    Prints a city and its destinations to the log file.

Private Attributes

- string **thisCityName**
- bool **beenVisited**
- vector< string > **destinationNames**
- vector< City ∗ > **destinationPointers**
- vector< int > **flightNumbers**
- vector< int > **flightCosts**
- int **destinations**

3.1.1 Constructor & Destructor Documentation

3.1.1.1 City::City ( )

The default constructor of a City object.

This constructor initializes values of a City object to default values

Algorithm None.

Parameters

in	*None.*	
out	*None.*	

Returns

None.

Note

None.

3.1.1.2 City::City (string *name*)

The parameterized constructor of a City object.

This constructor initializes values of a City object to default values except for the name

Algorithm None.

Parameters

in	*name*	The name to give to the new City object.
out	*None.*	

Returns

None.

Note

None.

3.1.1.3 City::∼City ( )

The destructor of a City object.

This is meant to remove any presense of a City object on memory.

Algorithm None.

Parameters

in	*None.*	
out	*None.*	

Returns

None.

Note

None.

3.1.2 Member Function Documentation

3.1.2.1 void City::AddDestination (string *name,* City ∗ *destinationPtr,* int *flightNumber,* int *cost*)

Adds a new detination for the city to go to.

Adds the various sent values to their respective arrays to keep track of adjacent cities

Algorithm Checks if the destination already exists, if it doesn't then it is created.

Parameters

in	*name*	The name of the destination
in	*destinationPtr*	The pointer to the destination City object
in	*flightNumber*	The flight number of the flight between this city and the destination
in	*cost*	The price of the flight from here to there
out	*None.*	

Returns

None.

Note

None.

3.1.2.2 bool City::CheckIfDestination (string *name*)

Checks if a city is a destination from here.

Checks if any of its destinations match the name of the one sent

Algorithm None.

Parameters

`in`	*name*	The name of the city to look for
`out`	*None.*	

Returns

None.

Note

None.

3.1.2.3 bool City::CheckIfVisited ( )

Checks if this city has been visited.

Returns the value of beenVisited

Algorithm None.

Parameters

`in`	*None.*	
`out`	*None.*	

Returns

Returns the value of beenVisited

Note

None.

3.1.2.4 string City::GetCityName ( )

Gets a city's name.

Returns the value at thisCityName

Algorithm None.

Parameters

in	*None.*	
out	*None.*	

Returns

The string of the city's name

Note

None.

3.1.2.5 City ∗ City::GetDestination (string *name*)

Gives the pointer to the destination sent.

If the destination exists from this city it will return a pointer to its object

Algorithm None.

Parameters

in	*name*	The name of the city to search for
out	*None.*	

Returns

Returns a pointer to the city that was searched for.

Note

Assumes that the city does exist as a destination

3.1.2.6 int City::GetDestinationCount ( )

Gets a city's destination count.

Returns the value at destinations

Algorithm None.

Parameters

`in`	*None.*	
`out`	*None.*	

Returns

The number of destinations this city has

Note

None.

3.1.2.7 vector< City * > City::GetDestinationPointers ( )

Gets a city's destination pointer array.

Returns a copy of the destination pointer vector

Algorithm None.

Parameters

`in`	*None.*	
`out`	*None.*	

Returns

A copy of the vector of destinations that this city has

Note

None.

3.1.2.8 City * City::GetUnvisitedDestination ( )

Returns an unvisted destination.

Searches through its destinations to find one that has not been visited yet and returns it.

Algorithm None.

Parameters

`in`	*None.*	
`out`	*None.*	

Returns

A pointer to the first found unvisited city, NULL if none.

Note

None.

3.1.2.9 void City::PrintCity ( )

Prints a city and its destinations.

Prints the city, then loops through its string vector printing its destinations

Algorithm None.

Parameters

`in`	*None.*	
`out`	*None.*	

Returns

None.

Note

None.

3.1.2.10 void City::PrintCityToLog (ofstream ∗ *logFile*)

Prints a city and its destinations to the log file.

Prints the city, then loops through its string vector printing its destinations to the log

Algorithm None.

Parameters

`in`	*logFile*	A pointer to the log file to be output to.
`out`	*None.*	

Returns

None.

Note

Assumes that the logFile has already been opened and will be closed somewhere else.

3.1.2.11 int City::PrintFlight (City $*$ *destinationPtr*)

Prints the flight information.

Prints the flight information from this city to the destination city

Algorithm None.

Parameters

`in`	*destinationPtr*	A pointer to the destination city object.
`out`	*None.*	

Returns

Returns the cost of the flight from here to there.

Note

None.

3.1.2.12 void City::SetCityName (string *name*)

Changes a city's name.

Sets the value of thisCityName to the sent value

Algorithm None.

Parameters

`in`	*name*	The new name to assign to the city.
`out`	*None.*	

Returns

None.

Note

None.

3.1.2.13 void City::SetVisitedState (bool *state*)

Marks if a city has been visited.

Sets the value of beenVisited to the sent value

Algorithm None.

Parameters

in	*state*	The state (true/false) to set beenVisited
out	*None.*	

Returns

None.

Note

None.

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

- PA03/City.h
- PA03/City.cpp

3.2 CityDataInput Class Reference

Public Member Functions

- string **getName** ()
- void **getNamePair** ()
- int **GetDataAmount** (ifstream ∗sentFile)

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

- PA03/CityDataInput.h

3.3 FlightMapV1 Class Reference

Public Member Functions

- FlightMapV1 (City ∗sentCityHead, int sentNumCities)

  The parameter constructor for a FlightMapV2 object.
- ∼FlightMapV1 ()

  The destructor for a FlightMapV2 object.
- void MarkVisited (City ∗sentCity)

  Marks the sent city as visited.
- void UnvisitAll ()

  Marks all cities as unvisited.
- City ∗ GetNextCity (City ∗fromCity)

  Gets the next unvisited city.
- bool IsPath (City ∗originCity, City ∗destinationCity, ofstream ∗logFile)

  Checks if there is a valid flight path between the two cities.

Private Attributes

- City ∗ **cityHead**
- int **numCities**

3.3.1 Constructor & Destructor Documentation

3.3.1.1 FlightMapV1::FlightMapV1 (City ∗ *sentCityHead,* int *sentNumCities*)

The parameter constructor for a FlightMapV2 object.

The parameter constructor for a FlightMapV2 object

Algorithm None.

Parameters

in	*sentCityHead*	Pointer to the start of the city array
in	*sentNumCities*	the number of cities in the city array
out	*None.*	

Returns

None.

Note

None.

3.3.1.2 FlightMapV1::∼FlightMapV1 ( )

The destructor for a FlightMapV2 object.

The destructor for a FlightMapV2 object

Algorithm None.

Parameters

in	*None.*	
out	*None.*	

Returns

None.

Note

None.

3.3.2 Member Function Documentation

3.3.2.1 City ∗ FlightMapV1::GetNextCity (City ∗ *fromCity*)

Gets the next unvisited city.

Gets the first unvisited city from the sent city's destinations

Algorithm None.

Parameters

`in`	*fromCity*	Pointer to the city which will have its destinations checked
`out`	*A*	pointer to a city.

Returns

Returns a pointer to the first unvisited city found, NULL if none found.

Note

None.

3.3.2.2 bool FlightMapV1::IsPath (City ∗ *originCity,* City ∗ *destinationCity,* ofstream ∗ *logFile*)

Checks if there is a valid flight path between the two cities.

Moves through the cities searching for a path from originCity to destinationCity

Algorithm Moves through destinations until it finds a valid path to the destination or it is out of possible routes. Marks cities as visited to avoid re-visitir

Parameters

`in`	*originCity*	Pointer to the city which the search starts from
`in`	*destinationCity*	Pointer to the destination city
`out`	*A*	bool whether or not a path exists

Returns

Returns true if there is a valid path, false otherwise.

Note

None.

3.3.2.3 void FlightMapV1::MarkVisited (City ∗ *sentCity*)

Marks the sent city as visited.

Calls the sent city object's SetVisitedState function with the argument true

Algorithm None.

Parameters

`in`	*sentCity*	Pointer to the city to mark as visited
`out`	*None.*	

Returns

None.

Note

None.

3.3.2.4 void FlightMapV1::UnvisitAll ( )

Marks all cities as unvisited.

Starts at the beginning of the city array and mark each city as unvisited

Algorithm None.

Parameters

`in`	*None.*	
`out`	*All*	cities are now reset to be unvisited.

Returns

None.

Note

None.

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

- PA03/FlightMapV1.h
- PA03/FlightMapV1.cpp

3.4 FlightMapV2 Class Reference

Public Member Functions

- FlightMapV2 (City ∗sentCityHead, int sentNumCities)

  The parameter constructor for a FlightMapV2 object.
- ∼FlightMapV2 ()

  The destructor for a FlightMapV2 object.
- void MarkVisited (City ∗sentCity)

  Marks the sent city as visited.
- void UnvisitAll ()

  Marks all cities as unvisited.
- void UnvisitAll (vector< City ∗ > ∗sentVector)

  Marks all cities as unvisited inside of a vector of cities.
- City ∗ GetNextCity (City ∗fromCity)

  Gets the next unvisited city.
- bool IsPath (City ∗originCity, City ∗destinationCity, ofstream ∗logFile)

  Checks if there is a valid flight path between the two cities.

Private Attributes

- City ∗ **cityHead**
- int **numCities**

3.4.1 Constructor & Destructor Documentation

3.4.1.1 FlightMapV2::FlightMapV2 (City ∗ *sentCityHead,* int *sentNumCities*)

The parameter constructor for a FlightMapV2 object.

The parameter constructor for a FlightMapV2 object

Algorithm None.

Parameters

in	*sentCityHead*	Pointer to the start of the city array
in	*sentNumCities*	the number of cities in the city array
out	*None.*	

Returns

None.

Note

None.

3.4.1.2 FlightMapV2::∼FlightMapV2 ( )

The destructor for a FlightMapV2 object.

The destructor for a FlightMapV2 object

Algorithm None.

Parameters

`in`	*None.*	
`out`	*None.*	

Returns

None.

Note

None.

3.4.2 Member Function Documentation

3.4.2.1 City ∗ FlightMapV2::GetNextCity (City ∗ *fromCity*)

Gets the next unvisited city.

Gets the first unvisited city from the sent city's destinations

Algorithm None.

Parameters

`in`	*fromCity*	Pointer to the city which will have its destinations checked
`out`	*A*	pointer to a city.

Returns

Returns a pointer to the first unvisited city found, NULL if none found.

Note

None.

3.4.2.2 bool FlightMapV2::IsPath (City ∗ *originCity,* City ∗ *destinationCity,* ofstream ∗ *logFile*)

Checks if there is a valid flight path between the two cities.

Moves through the cities searching for a path from originCity to destinationCity

Algorithm Moves through destinations until it finds a valid path to the destination or it is out of possible routes. Marks cities as visited to avoid re-visitir

Parameters

`in`	*originCity*	Pointer to the city which the search starts from
`in`	*destinationCity*	Pointer to the destination city
`out`	*A*	bool whether or not a path exists

Returns

Returns true if there is a valid path, false otherwise.

Note

None.

3.4.2.3 void FlightMapV2::MarkVisited (City ∗ *sentCity*)

Marks the sent city as visited.

Calls the sent city object's SetVisitedState function with the argument true

Algorithm None.

Parameters

`in`	*sentCity*	Pointer to the city to mark as visited
`out`	*None.*	

Returns

None.

Note

None.

3.4.2.4 void FlightMapV2::UnvisitAll ( )

Marks all cities as unvisited.

Starts at the beginning of the city array and mark each city as unvisited

Algorithm None.

Parameters

`in`	*None.*	
`out`	*All*	cities are now reset to be unvisited.

Returns

None.

Note

None.

3.4.2.5 void FlightMapV2::UnvisitAll (vector< City * > * *sentVector*)

Marks all cities as unvisited inside of a vector of cities.

Starts at the beginning of the city vector and mark each city as unvisited

Algorithm None.

Parameters

`in`	*None.*	
`out`	*All*	cities in the vector are now reset to be unvisited.

Returns

None.

Note

None.

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

- PA03/FlightMapV2.h
- PA03/FlightMapV2.cpp

3.5 Stack< itemType > Class Template Reference

Public Member Functions

- **Stack** (int size=10)
- **Stack** (const Stack< itemType > &)
- bool push (itemType)

  This function pushes a value onto the stack if there is room.
- itemType pop ()

  This function is used to remove the value at the top of the stack.
- bool isEmpty () const

  This function checks if the stack has any values stored.
- bool isFull () const

  This function checks if the stack is full.
- bool clear ()

  This function deletes the data stored in the stack.
- itemType peek ()

  This function returns the value at the top of the stack.
- void print ()

  This function prints all values stored in the stack.
- void printFlights ()

  This function prints all values stored in the stack in a flight route format.
- void **printFlightsToLog** (ofstream ∗logFile)

Private Attributes

- int **max**
- int **top**
- itemType ∗ **data**

3.5.1 Member Function Documentation

3.5.1.1 template<class itemType > bool Stack< itemType >::clear ( )

This function deletes the data stored in the stack.

This function deletes the data stored in the stack, and allocates a new set of memory to make a clean stack

Algorithm None.

Parameters

`in`	*None.*	
`out`	*None.*	

Returns

Should always return true.

Note

Originally the function should return false if the stack is already empty, this behaviour was removed and it just wipes the stack no matter what

3.5.1.2 template< class itemType > bool Stack< itemType >::isEmpty ( ) const

This function checks if the stack has any values stored.

This function checks the value of top, which should be >=0 if there is any data inside.

Algorithm None.

Parameters

`in`	*None.*	
`out`	*None.*	

Returns

Returns true if the stack is empty, false otherwise.

Note

None.

3.5.1.3 template< class itemType > bool Stack< itemType >::isFull ( ) const

This function checks if the stack is full.

This function checks the value of top, which should be equal to max - 1 if it is full

Algorithm None.

Parameters

`in`	*None.*	
`out`	*None.*	

Returns

Returns true if the stack is full, false otherwise.

Note

None.

3.5.1.4 template<class itemType > itemType Stack< itemType >::peek ( )

This function returns the value at the top of the stack.

This function goes to the top of the stack and returns the value

Algorithm None.

Parameters

`in`	*None.*	
`out`	*None.*	

Returns

Returns the value at top if there is one

Note

None.

3.5.1.5 template<class itemType > itemType Stack< itemType >::pop ( )

This function is used to remove the value at the top of the stack.

This function removes the value from the top of the stack (if there is one) and returns it.

Algorithm Moves the pointer dataTrav to the top of the stack, removes it, and returns it.

Parameters

`in`	*None.*	
`out`	*None.*	

Returns

Returns the value that was at the top of the stack. NULL if the stack is empty.

Note

None.

3.5.1.6 template< class itemType > void Stack< itemType >::print ( )

This function prints all values stored in the stack.

This function runs through the stack and prints all values inside. Top is marked.

Algorithm None.

Parameters

`in`	*None.*	
`out`	*Outputs*	the contents of the stack to the console

Returns

None.

Note

None.

3.5.1.7 template< class itemType > void Stack< itemType >::printFlights ( )

This function prints all values stored in the stack in a flight route format.

This function runs through the stack and prints all values inside, formatted as a route.

Algorithm None.

Parameters

`in`	*None.*	
`out`	*Outputs*	the contents of the stack to the console

Returns

None.

Note

None.

3.5.1.8 template< class itemType > bool Stack< itemType >::push (itemType *value*)

This function pushes a value onto the stack if there is room.

This function pushes a value on the stack if there is room for the value

Algorithm If there is room, the traverser pointer is moved to the top of the stack and the value is inserted

Parameters

`in`	*value*	The value to be inserted, must be the same type as the stack (i.e. a string can't be pushed into an int stack)
`out`	*None.*	

Returns

Returns true if a value was able to be pushed, returns false if the stack was full

Note

None.

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

- PA03/Stack.cpp

4 File Documentation

4.1 PA03/City.cpp File Reference

This file contains the implementation of the City class.

```
#include "City.h"
```

4.1.1 Detailed Description

This file contains the implementation of the City class.

Author

Bryce Monaco

This file contains the implementations for the various members of the City class. Some functions are not used.

Version

1.0

Note

None.

4.2 PA03/City.h File Reference

This file contains the header of the City class.

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

Classes

- class City

4.2.1 Detailed Description

This file contains the header of the City class.

Author

Bryce Monaco

This file contains the header for the various members of the City class. Some functions are not used.

Version

1.0

Note

None.

4.3 PA03/FlightMapV1.cpp File Reference

This file contains the implementation of the FLightMapV1 class.

```
#include "FlightMapV1.h"
```

4.3.1 Detailed Description

This file contains the implementation of the FLightMapV1 class.

Author

Bryce Monaco

This file contains the implementation for the various members of the FlightMapV1 class.

Version

1.0

Note

None.

4.4 PA03/FlightMapV1.h File Reference

This file contains the header of the FLightMapV1 class.

```
#include <string>
#include "Stack.cpp"
#include "City.h"
```

Classes

- class FlightMapV1

4.4.1 Detailed Description

This file contains the header of the FLightMapV1 class.

Author

Bryce Monaco

This file contains the header for the various members of the FlightMapV1 class.

Version

1.0

Note

None.

4.5 PA03/FlightMapV2.cpp File Reference

This file contains the implementation of the FLightMapV2 class.

```
#include "FlightMapV2.h"
```

4.5.1 Detailed Description

This file contains the implementation of the FLightMapV2 class.

Author

Bryce Monaco

This file contains the implementation for the various members of the FlightMapV2 class.

Version

1.0

Note

None.

4.6 PA03/FlightMapV2.h File Reference

This file contains the header of the FLightMapV2 class.

```
#include <string>
#include "Stack.cpp"
#include "City.h"
```

Classes

- class FlightMapV2

4.6.1 Detailed Description

This file contains the header of the FLightMapV2 class.

Author

Bryce Monaco

This file contains the header for the various members of the FlightMapV2 class.

Version

1.0

Note

None.

4.7 PA03/PA03.cpp File Reference

This is PA03's main driver file.

```
#include <iostream>
#include <fstream>
#include <string>
#include <vector>
#include <math.h>
#include "FlightMapV1.h"
#include "FlightMapV2.h"
```

Functions

- int GetDataAmount (ifstream ∗sentFile)

  This function is used to scan through the data file to see the amount of values within the file.
- void DecodeLine (string sentLine, string ∗leftCity, string ∗rightCity)

  Finds the two city names within the sent string.
- void DecodeLine (string sentLine, string ∗leftCity, string ∗rightCity, int ∗flightNum, int ∗cost)

  Finds the two city names within the sent string, also decodes the flight number and cost.
- int FindCityIndex (string name, City ∗cityArray, int cityCount)

  Finds the index of the city with the given name in the city array.
- bool CheckIfValidCity (string name, City ∗cities, int numCities)

  Checks if the sent city is in the city array.
- int **main** ()

Variables

- ofstream **logFile**

4.7.1 Detailed Description

This is PA03's main driver file.

Author

Bryce Monaco

This is the main driver file for the project, it hands input mostly.

Version

1.0

Note

!!!!!!!!!!!!SEE LINE 142/143 TO SWITCH BETWEEN V1 AND V2!!!!!!!!!!!!!!

4.7.2 Function Documentation

4.7.2.1 bool CheckIfValidCity (string *name,* City ∗ *cities,* int *numCities*)

Checks if the sent city is in the city array.

Uses the sent name to find a city with the same name in the city array

Algorithm None.

Parameters

`in`	*name*	The string of the name to search for
`in`	*cities*	Pointer to the start of the city array
`in`	*numCities*	An int used to make sure that the loop does not go out of bounds in the city array
`out`	*None.*	

Returns

Returns true if the city is in the array, false if it is not

Note

None.

4.7.2.2 void DecodeLine (string *sentLine,* string ∗ *leftCity,* string ∗ *rightCity*)

Finds the two city names within the sent string.

Moves through the sent string, checking each character and storing it if it is part of the name

Algorithm Goes through the string, character by character, and stores each found char into the left city name, once it hits the comma it skips any white

Parameters

`in`	*sentLine*	The string read in from a file with getline()
`in`	*leftCity*	Pointer to the string where the left city name is stored
`in`	*rightCity*	Pointer to the string where the right city name is stored
`out`	*None.*	

Returns

Returns None.

Note

None.

4.7.2.3 void DecodeLine (string *sentLine,* string ∗ *leftCity,* string ∗ *rightCity,* int ∗ *flightNum,* int ∗ *cost*)

Finds the two city names within the sent string, also decodes the flight number and cost.

Moves through the sent string, checking each character and storing it if it is part of the name

Algorithm Goes through the string, character by character, and stores each found char into the left city name, once it hits the comma it skips any white

Parameters

in	*sentLine*	The string read in from a file with getline()
in	*leftCity*	Pointer to the string where the left city name is stored
in	*rightCity*	Pointer to the string where the right city name is stored
in	*flightNum*	Pointer the the int where the flight number is stored
in	*cost*	Pointer to the int where the cost is stored
out	*None.*	

Returns

Returns None.

Note

None.

4.7.2.4 int FindCityIndex (string *name,* City ∗ *cityArray,* int *cityCount*)

Finds the index of the city with the given name in the city array.

Uses the sent name to find a city with the same name in the city array

Algorithm None.

Parameters

in	*name*	The string of the name to search for
in	*cityArray*	Pointer to the start of the city array
in	*cityCount*	An int used to make sure that the loop does not go out of bounds in the city array
out	*None.*	

Returns

Returns the index of the city, 999999 if the city does not exist.

Note

While theoretically there is a possiblility for error with returning 999999, if there is an array of $>$999999 cities then there will likely be a memory capacity issue before this function causes an issue

4.7.2.5 int GetDataAmount (ifstream ∗ *sentFile*)

This function is used to scan through the data file to see the amount of values within the file.

The function continuously reads through the data file to count the number of values within it. The number found is used to dynamically size the values array in main().

Algorithm Continuously reads through the file checking if it has reached the end of the file. It increments an integer to keep count of the values and retu

Parameters

`in`	*sentFile*	The data file opened in main(), contains the data to be read in.
`out`	*count*	The number of values within the data file.

Returns

Returns the amount of valid values within the data file. Plus one because eof isn't marked until after with getline

Note

None.

4.8 PA03/Stack.cpp File Reference

This file contains both the header and the implementation for the Stack class used in PA03.

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

Classes

- class Stack< itemType >

4.8.1 Detailed Description

This file contains both the header and the implementation for the Stack class used in PA03.

Author

Bryce Monaco (Originally for CS202)

This file contains the code for the Stack class used in PA03. This is a modified Stack class. It is based off of a Stack class that I had written for an assignment in CS202 during Spring 2016 (Project 8). It is missing a few functions which I deemed nonessential for this project.

Version

1.0

Note

This code was put into one file instead of a .h and .cpp because of issues with templating. The header section of this code is the only part taken from the 202 code, I set it up to be templated (originally it only worked with chars) and completely rewrote the implementation.

Index