My Project

Generated by Doxygen 1.8.6

Mon Feb 23 2015 20:54:01

Contents

Chapter 1

Class Index

1.1 Class List

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

Chapter 2

File Index

2.1 File List

Here is a list of all files with brief descriptions:

Chapter 3

Class Documentation

3.1 city Class Reference

`#include <myclass.h>`

Public Member Functions

- city ()
- void setName (const string)
- string getName () const
- void setVisited (const bool)
- bool isVisited () const
- void setFlightNumber (const int)
- int getFlightNumber ()
- void setFlightCost (const int)
- int getFlightCost ()

3.1.1 Detailed Description

City class. Heavily involved in map class operations.

Modified from problem 11/12 implementations to include flight number and flight cost data members, used in additional flight map operations, and set/gets for them.

3.1.2 Constructor & Destructor Documentation

3.1.2.1 city::city ( )

City constructor. Initializes 'visited' value for the city to false and flight number and cost to 0.

Precondition

None.

Postcondition

City object created with visited value equal to false and flight number and flight cost to 0.

Returns

None.

3.1.3 Member Function Documentation

3.1.3.1 int city::getFlightCost ( )

City getFlightCost function.

Precondition

None.

Postcondition

City flight cost returned; data in city object unchanged.

Returns

int - The flight cost of the city object.

3.1.3.2 int city::getFlightNumber ( )

City getFlightNumber function.

Precondition

None.

Postcondition

City flight number returned; data in city object unchanged.

Returns

int - The flight number value of the city object.

3.1.3.3 string city::getName ( ) const

City getName function.

Precondition

None, but caveat emptor that this will return garbage if the city's name was not set elsewhere.

Postcondition

City name returned; data in city object unchanged.

Returns

string - The name of the city object.

3.1.3.4 bool city::isVisited ( ) const

City isVisited function.

Precondition

None.

Postcondition

City visited value returned; data in city object unchanged.

Returns

bool - The visited value of the city object.

3.1.3.5 void city::setFlightCost (const int *value*)

City setFlightCost function.

Parameters

value	The flight cost desired for the city's flight cost value.

Precondition

None.

Postcondition

City flight cost set to the one specified in the value parameter.

Returns

None.

3.1.3.6 void city::setFlightNumber (const int *value*)

City setFlightNumber function.

Parameters

value	The flight number desired for the city's flight number value.

Precondition

None.

Postcondition

City flight number set to the one specified in the value parameter.

Returns

None.

3.1.3.7 void city::setName (const string *value*)

City setName function.

Parameters

value	The string desired for the city's name value.

Precondition

None.

Postcondition

The city's name has been set to that of the value parameter.

Returns

None.

3.1.3.8 void city::setVisited (const bool *value*)

City setVisited function.

Parameters

value	The bool desired for the city's visited value.

Precondition

None.

Postcondition

City visited value set to the one specified in the value parameter.

Returns

None.

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

- myclass.h
- myclass.cpp

3.2 map Class Reference

`#include <myclass.h>`

Public Member Functions

- map ()
- ~map ()
- bool isPath (const city origin, const city destination)
- bool isValidRequest (const int)
- bool isServicedCity (const city) const
- city getRequestOrigin (const int) const
- city getRequestDestination (const int) const
- int getNumRequests () const

3.2.1 Detailed Description

Map class. Conducts operations derived from user datafiles to determine if certain flights between cities are possible. Contains a flight map of linked nodes that represents the valid flights that are possible from a given city, and an array of paired requests for origin and destination cities. Format the datafiles as specified to insure proper operation. MAX_CITIES number of total cities in the flight map and MAX_CITIES number of requests allowed.

Modified from problem 11 to feature an array of try-next pointers designed to make getAdjacentCity operation more efficient in some situations.

Modified from problem 12 to support a flight map that includes flight numbers and flight costs. Has new function, printItinerary, which will put this information in the console should a flight path be found.

3.2.2 Constructor & Destructor Documentation

3.2.2.1 map::map ( )

Map object constructor. Conducts the creation, through file readin, of the node-based flightmap and array of flight requests that will be used in later operations. Also initailizes the tryNext pointers for use in getAdjacentCity to NULL. Will call readin functions, which will prompt user for appropriate filenames. Filenames MUST be formatted correctly to ensure successful operation. Remember, flightFile.txt is formatted differently for this problem!

Precondition

None, but the functions called by this one will prompt user for valid and correctly formatted data files, as per the prompt. MAX_CITIES number of cities supported allowed for flight map; MAX_REQUESTS number of flight requests supported. Remember, flightFile.txt is formatted differently for this problem!

Postcondition

Map object created with flight map and requested flights as specified in the data files. Relevant values in tryNext array set to NULL.

Returns

None.

3.2.2.2 map::~map ( )

Map object destructor. Deallocates the nodes in the flight map so as to avoid memory leak.

Precondition

None.

Postcondition

Map object destroyed; nodes for the flight map, if any, deallocated.

Returns

None.

3.2.3 Member Function Documentation

3.2.3.1 int map::getNumRequests ( ) const

Map getNumRequests function.

Precondition

Correctly read-in data from the constructor is needed.

Postcondition

Value returned; flight map and requests unchanged.

Returns

int - The number of flight requests from read-in.

3.2.3.2 city map::getRequestDestination (const int *index*) const

Map getRequestDestination function. Returns the destination city for a read-in flight request.

Parameters

index	The index location of the request whose destination is to be returned.

Precondition

Index must be less than numFlightRequests. Correctly read-in data from the constructor is needed.

Postcondition

Value returned; flight map and requests unchanged.

Returns

city - The destination city of the specified request.

3.2.3.3 city map::getRequestOrigin (const int *index*) const

Map getRequestOrigin function. Returns the origin city for a read-in flight request.

Parameters

index	The index location of the request whose origin is to be returned.

Precondition

Index must be less than numFlightRequests. Correctly read-in data from the constructor is needed.

Postcondition

Value returned; flight map and requests unchanged.

Returns

city - The origin city of the specified request.

3.2.3.4 bool map::isPath (const city *origin*, const city *destination*)

City isPath function. Determines if it is possible to fly from the specified origin city to the specified destination city. Implemented using stack class.

Modified from Project 11/12 implementations to call a function which will report the itinerary to the console if a path is found. This implementation also uses a pointer to the stack used for flight map operations as a data member, so the itinerary function can access the data in the stack easily.

Parameters

origin	The city one is attempting to depart from.
destination	The city one is attempting to reach.

Precondition

None, but correctly read-in data from the constructor is needed for correct results.

Postcondition

Flight map may have had a number of cities within it marked visited and a number of tryNext pointers may have been created (these are reset to unvisited and NULL respectively if this function is called again).

Returns

bool - True if a path was found, false if no path was found..

3.2.3.5 bool map::isServicedCity (const city *target*) const

Map isServicedCity function. Determines if target city is one within the flight map (ie, if it is serviced by the airline).

Parameters

target	The city that is to be looked for within the flight map.

Precondition

Correctly read-in data from the constructor is needed.

Postcondition

Value returned; flight map and requests unchanged.

Returns

bool - True if city is contained in the flight map, false if not.

3.2.3.6 bool map::isValidRequest (const int *index*)

Map isValidRequest function. Determines if a read-in flight request has an origin and a destination that exist in the flight map.

Parameters

index	The index location of the request that is to be tested to see if it is valid.

Precondition

NIndex must be less than numFlightRequests. Correctly read-in data from the constructor is needed.

Postcondition

Bool returned; flight map and request unchanged.

Returns

bool - True if the request is valid, false if it is not.

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

- myclass.h
- myclass.cpp

3.3 namePair Class Reference

`#include <myclass.h>`

Public Member Functions

- void setOrigin (const city)
- city getOrigin () const
- void setDest (const city)
- city getDest () const

3.3.1 Detailed Description

namePair class. Contains origin and destination city; used in map operations.

3.3.2 Member Function Documentation

3.3.2.1 city namePair::getDest ( ) const

namePair getDest function. Returns the destination city of a namePair object.

Precondition

None.

Postcondition

namePair object's destination returned; object itself unchanged.

Returns

city - The destination city of the pair.

3.3.2.2 city namePair::getOrigin ( ) const

namePair getOrigin function. Returns the origin city of a namePair object.

Precondition

None.

Postcondition

namePair object's origin returned; object itself unchanged.

Returns

city - The origin city of the pair.

3.3.2.3 void namePair::setDest (const city *d*)

namePair setDest function. Sets the destination of a namePair object to specified city.

Parameters

d	The city that will be stored as the pair's destination.

Precondition

None.

Postcondition

namePair object's destination set to specified value.

Returns

None.

3.3.2.4 void namePair::setOrigin (const city *c*)

namePair setOrigin function. Sets the origin of a namePair object to specified city.

Parameters

c	The city that will be stored as the pair's origin.

Precondition

None.

Postcondition

namePair object's origin set to specified value.

Returns

None.

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

- myclass.h
- myclass.cpp

3.4 node Class Reference

`#include <myclass.h>`

Friends

- class map

3.4.1 Detailed Description

Node class. Used in map class flight map operations.

Modified from problem 11/12 implementation to support the new data members of the city class, flight number and flight cost.

3.4.2 Friends And Related Function Documentation

3.4.2.1 friend class map [friend]

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

- myclass.h
- myclass.cpp

3.5 readin Class Reference

`#include <myclass.h>`

Public Member Functions

- string getName (ifstream &)
- namePair getNamePair (ifstream &)
- namePair getNamePairFlightFile (ifstream &)

3.5.1 Detailed Description

Readin class. As specified by the prompt, used to simplify some file I/O processes.

Modified from problem 11/12 implementation to support the new datafile format specified for flightFile.txt and the new data members of the city class, flight number and flight cost.

3.5.2 Member Function Documentation

3.5.2.1 string readin::getName (ifstream & *fin*)

readin getName function. Reads in and returns a string from the datafile being handled by the fstream object fin.

Parameters

fin	The fstream object for the file currently being handled.

Precondition

A datafile must be opened by the fin object. This is designed for use in a correctly formatted "cityFile.txt" as specified, but will take any string in practice.

Postcondition

Object read in as per fstream specifications.

Returns

string - The string read out of the datafile.

3.5.2.2 namePair readin::getNamePair (ifstream & *fin*)

readin getNamePair function. Reads in and returns the data for a namePair object from the datafile being handled by the fstream object fin.

Contra problem 11, this function now only matches the formatting for "requestFile.txt". getNamePairFlightMap function has been added to handle "flightFile.txt".

Parameters

fin	The fstream object for the file currently being handled.

Precondition

A datafile must be opened by the fin object. The file must be formatted as specified in "requestFile.txt" as per the prompt.

Postcondition

Object read in as per fstream specifications.

Returns

namePair - The namePair read out of the datafile.

3.5.2.3 namePair readin::getNamePairFlightFile (ifstream & *fin*)

readin getNamePairFlightMap function. Reads in and returns the data for a namePair object from the datafile being handled by the fstream object fin.

Parameters

fin	The fstream object for the file currently being handled.

Precondition

A datafile must be opened by the fin object. The file must be formatted as specified in "flightFile.txt" as per the prompt.

Postcondition

Object read in as per fstream specifications.

Returns

namePair - The namePair read out of the datafile.

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

- myclass.h
- myclass.cpp

3.6 stack< ItemType > Class Template Reference

`#include <myclass.h>`

Public Member Functions

- stack ()
- bool isEmpty () const
- bool push (const ItemType &newEntry)
- bool pop ()
- ItemType peekAtTop () const
- void initCursor ()
- bool traverseToNext ()
- ItemType peekAtCursor () const

3.6.1 Detailed Description

template<class ItemType>class stack< ItemType >

Templated stack class. Used in implementation of map::isPath function. Adapted from implementation in class textbook, "Data Abstraction & Problem Solving with C++, sixth ed". Supports maximum MAX_STACK number of items.

Modified from problem 11/12 implementations to be traversable from the bottom up, as specified in the prompt. User can traverse from the bottom of the stack up, one item at a time, peeking at the values, using the cursor functions.

3.6.2 Constructor & Destructor Documentation

3.6.2.1 template<class ItemType > stack< ItemType >::stack ( )

Stack constructor.

Precondition

None.

Postcondition

Stack is created and ready for first entry.

Returns

None.

3.6.3 Member Function Documentation

3.6.3.1 template<class ItemType > void stack< ItemType >::initCursor ( )

Stack initCursor function.

Precondition

None.

Postcondition

Cursor is set (or reset) to the bottom of the stack. Caveat emptor: use before any cursor operations, as cursor is initailized at -1, but make sure the stack is not empty first.

Returns

None.

3.6.3.2 template<class ItemType > bool stack< ItemType >::isEmpty ( ) const

Stack isEmpty function.

Precondition

None.

Postcondition

Bool returned; stack data unchanged.

Returns

bool - True if empty, false if not.

3.6.3.3 template<class ItemType > ItemType stack< ItemType >::peekAtCursor ( ) const

Stack peekAtCursor function.

Precondition

None.

Postcondition

Value at cursor returned, if there was one.

Returns

The item located at cursor, if there was one.

3.6.3.4 template<class ItemType > ItemType stack< ItemType >::peekAtTop ( ) const

Stack peekAtTop function.

Precondition

None.

Postcondition

Top value returned, if there was one. Data in the stack unchanged.

Returns

The item at the top of the stack, if there was one.

3.6.3.5 template<class ItemType > bool stack< ItemType >::pop ( )

Stack pop function.

Precondition

None.

Postcondition

Bool returned; if the stack was not empty, the value at the top of the stack has been removed.

Returns

bool - True if pop was successful, false if not.

3.6.3.6 template<class ItemType> bool stack< ItemType >::push (const ItemType & *newValue*)

Stack push function.

Parameters

newValue	The Item to be added to the stack.

Precondition

None.

Postcondition

Bool returned; if the stack was not full, the value has been pushed onto the stack.

Returns

bool - True if the value was added, false if it could nto be added because the stack was full.

3.6.3.7 template<class ItemType > bool stack< ItemType >::traverseToNext ( )

Stack traverseToNext function.

Precondition

None.

Postcondition

Cursor incremented by 1 if not at the top of the stack already.

Returns

bool - True if cursor could be incremented, false if not.

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

- myclass.h
- myclass.cpp

Chapter 4

File Documentation

4.1 main.cpp File Reference

```
#include "myclass.h"
#include <iostream>
```

Functions

- int main ()

4.1.1 Detailed Description

CS 302 Project 3 Question 13 - main program for testing

Author

Patrick Austin

Date

2/21/2015

4.1.2 Function Documentation

4.1.2.1 int main ( )

4.2 myclass.cpp File Reference

```
#include "myclass.h"
#include <cassert>
#include <fstream>
#include <iostream>
```

4.2.1 Detailed Description

CS 302 Project 3 Question 13 - class implementations

Author

Patrick Austin

Date

2/21/2015

4.3 myclass.h File Reference

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

Classes

- class stack< ItemType >
- class city
- class namePair
- class map
- class node
- class readin

Variables

- const int MAX_STACK = 100
- const int MAX_REQUESTS = 20
- const int MAX_CITIES = 20

4.3.1 Detailed Description

CS 302 Project 3 Question 13 - header for classes

Author

Patrick Austin

Date

2/21/2015

4.3.2 Variable Documentation

4.3.2.1 const int MAX_CITIES = 20

4.3.2.2 const int MAX_REQUESTS = 20

4.3.2.3 const int MAX_STACK = 100

Index