Throw Catch

1.0

Generated by Doxygen 1.8.11

Contents

Chapter 1

Hierarchical Index

1.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 2

Class Index

2.1 Class List

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

Chapter 3

File Index

3.1 File List

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

Chapter 4

Class Documentation

4.1 Address Struct Reference

The Address struct The struct has four QString to manage the street address, city, state, and zipcode. An easier way to keep track of each segment of the address.

```
#include <stadium.h>
```

Public Attributes

- QString **streetAddress**
- QString **city**
- QString **state**
- QString **zipCode**

4.1.1 Detailed Description

The Address struct The struct has four QString to manage the street address, city, state, and zipcode. An easier way to keep track of each segment of the address.

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

- src/header/stadium.h

4.2 CompleteTree< E > Class Template Reference

A Complete binary tree class This class creates a complete binary tree, or a tree where every level has the maximum number of nodes possible, and the nodes in the last level fill from left to right.

```
#include <CompleteTree.h>
```

Public Types

- typedef std::vector< E >::iterator Position

  Typedef for a position in the tree.

Public Member Functions

- CompleteTree ()

  CompleteTree.
- bool empty () const

  empty
- int size () const

  size
- bool hasLeft (const Position &p) const

  hasLeft
- bool hasRight (const Position &p) const

  hasRight
- bool isRoot (const Position &p) const

  isRoot
- Position root ()

  root
- Position last ()

  last
- Position addLast (const E &e)

  addLast
- void removeLast ()

  Remove an element from the end of the tree.
- void swap (const Position &p, const Position &q)

  swap
- Position left (const Position &p)

  left
- Position right (const Position &p)

  right
- Position parent (const Position &p)

  parent

Protected Member Functions

- Position pos (int i)

  pos
- int idx (const Position &p) const

  idx

4.2.1 Detailed Description

template<typename E>
class CompleteTree< E >

A Complete binary tree class This class creates a complete binary tree, or a tree where every level has the maximum number of nodes possible, and the nodes in the last level fill from left to right.

Author

Ethan Slattery

Date

12APR2016

4.2.2 Constructor & Destructor Documentation

4.2.2.1 template<typename E> CompleteTree< E >::CompleteTree ( ) `[inline]`

CompleteTree.

Constructor: Initialized the vector to a size of one, since we do not use index 0

4.2.3 Member Function Documentation

4.2.3.1 template<typename E> Position CompleteTree< E >::addLast (const E & *e*) `[inline]`

addLast

Parameters

e	

Returns

4.2.3.2 template<typename E> bool CompleteTree< E >::empty ( ) const `[inline]`

empty

Returns

4.2.3.3 template<typename E> bool CompleteTree< E >::hasLeft (const Position & *p*) const `[inline]`

hasLeft

Parameters

p	

Returns

4.2.3.4 template<typename E> bool CompleteTree< E >::hasRight (const Position & *p*) const [inline]

hasRight

Parameters

p	

Returns

4.2.3.5 template<typename E> int CompleteTree< E >::idx (const Position & *p*) const [inline], [protected]

idx

Parameters

p	

Returns

4.2.3.6 template<typename E> bool CompleteTree< E >::isRoot (const Position & *p*) const [inline]

isRoot

Parameters

p	

Returns

4.2.3.7 template<typename E> Position CompleteTree< E >::last ( ) [inline]

last

Returns

4.2.3.8 template<typename E> Position CompleteTree< E >::left (const Position & *p*) [inline]

left

Parameters

p	

Returns

4.2.3.9 template<typename E> Position CompleteTree< E >::parent (const Position & *p*) [inline]

parent

Parameters

p	

Returns

4.2.3.10 template<typename E> Position CompleteTree< E >::pos (int *i*) [inline], [protected]

pos

Parameters

i	

Returns

4.2.3.11 template<typename E> Position CompleteTree< E >::right (const Position & *p*) [inline]

right

Parameters

p	

Returns

4.2.3.12 template<typename E> Position CompleteTree< E >::root ( ) [inline]

root

Returns

4.2.3.13 template<typename E> int CompleteTree< E >::size ( ) const [inline]

size

Returns

4.2.3.14 template<typename E> void CompleteTree< E >::swap (const Position & *p*, const Position & *q*) [inline]

swap

Parameters

p	
q	

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

- src/header/CompleteTree.h

4.3 DBManager Class Reference

Public Member Functions

- DBManager ()

  DBManager.
- ∼DBManager ()

  DBManager::∼DBManager Disconnects the database.
- skiplist< int, Stadium ∗ > getStadiums ()

  getStadiums
- bool AddNewStadium (Stadium ∗s)

  AddNewStadium.
- bool UpdateStadium (Stadium ∗s)

  UpdateStadium.
- bool AddNewSouvenir (int stadiumKey, QString name, double price, int quantity)

  AddNewSouvenir.
- bool RemoveSouvenir (int stadiumKey, QString name)

  RemoveSouvenir.
- bool updateSouvenirName (int stadiumKey, QString oldName, QString newName)

  updateSouvenirName
- bool updateSouvenirPrice (int stadiumKey, QString souvenirName, double newPrice)

  updateSouvenirPrice
- bool updateSouvenirQuantity (int stadiumKey, QString souvenirName, int newQuantity)

  updateSouvenirQuantity
- bool updateTotalRevenue (int stadiumKey, double newRevenue)

  updateTotalRevenue
- int getStadiumID (QString stadiumName)

  getStadiumID
- QVector< int > getAllStadiumsKeys ()

  getAllStadiumsKeys
- bool addEdges (Stadium ∗origin, Stadium ∗destination, int weight)

  addEdges
- Graph< Stadium > ∗ createGraph (skiplist< int, Stadium ∗ > stadiumList)

  createGraph

4.3.1 Constructor & Destructor Documentation

4.3.1.1 DBManager::DBManager ( )

DBManager.

DBManager::DBManager Connects to the database and turns on foreign keys.

4.3.2 Member Function Documentation

4.3.2.1 bool DBManager::addEdges (Stadium ∗ *origin,* Stadium ∗ *destination,* int *weight*)

addEdges

Parameters

origin	
destination	
weight	

Returns

4.3.2.2 bool DBManager::AddNewSouvenir (int *stadiumKey,* QString *name,* double *price,* int *quantity*)

AddNewSouvenir.

DBManager::AddNewSouvenir Add a souvenir item to the souvenir table.

Parameters

stadiumKey	
name	
price	
quantity	

Returns

Parameters

stadiumKey	key links to an stadium key/id.
name	name of the souvenir item.
price	price of the souvenir item.
quantity	quanity of how much is purchased of the specified item.

Returns

true if the new souvenir is successfully added to the souvenir table, otherwise false.

4.3.2.3 bool DBManager::AddNewStadium (Stadium ∗ *s*)

AddNewStadium.

Parameters

s	

Returns

4.3.2.4 Graph$<$ Stadium $>*$ DBManager::createGraph (skiplist$<$ int, Stadium $*>$ *stadiumList*)

createGraph

Parameters

stadiumList	

Returns

4.3.2.5 QVector$<$ int $>$ DBManager::getAllStadiumsKeys ( )

getAllStadiumsKeys

Returns

4.3.2.6 int DBManager::getStadiumID (QString *stadiumName*)

getStadiumID

Parameters

stadiumName	

Returns

4.3.2.7 skiplist$<$ int, Stadium $*>$ DBManager::getStadiums ( )

getStadiums

Returns

A C C E S S O R S

4.3.2.8 bool DBManager::RemoveSouvenir (int *stadiumKey,* QString *name*)

RemoveSouvenir.

DBManager::RemoveSouvenir Removes a souvenir from the souvenir's menu.

Parameters

stadiumKey	
name	

Returns

Parameters

stadiumKey	key links to an stadium key/id.
name	name of the souvenir to removed

Returns

true if the new souvenir is successfully removed to the souvenir table, otherwise false.

4.3.2.9 bool DBManager::updateSouvenirName (int *stadiumKey,* QString *oldName,* QString *newName*)

updateSouvenirName

Parameters

stadiumKey	
oldName	
newName	

Returns

4.3.2.10 bool DBManager::updateSouvenirPrice (int *stadiumKey,* QString *souvenirName,* double *newPrice*)

updateSouvenirPrice

Parameters

stadiumKey	
souvenirName	
newPrice	

Returns

4.3.2.11 bool DBManager::updateSouvenirQuantity (int *stadiumKey,* QString *souvenirName,* int *newQuantity*)

updateSouvenirQuantity

Parameters

stadiumKey	
souvenirName	
newQuantity	

Returns

4.3.2.12 bool DBManager::UpdateStadium (Stadium ∗ *s*)

UpdateStadium.

Parameters

s	

Returns

4.3.2.13 bool DBManager::updateTotalRevenue (int *stadiumKey,* double *newRevenue*)

updateTotalRevenue

Parameters

stadiumKey	
newRevenue	

Returns

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

- src/header/dbmanager.h
- src/source/dbmanager.cpp

4.4 Graph< E >::Edge Class Reference

The Edge class Edge within the graph holds the weight between two incident vertecies and methods to manipulate that data and access the adjacent vertrices.

```
#include <graph.h>
```

Public Member Functions

- **Edge** (const int &weight=0)
- void **setEnd** (VertexItr newEnd)
- void **setStart** (VertexItr newStart)
- void **visit** ()
- void **resetVisited** ()
- Vertex & **start** ()
- Vertex & **end** ()
- int **weight** () const
- VertexItr opposite (Vertex v)

  Graph<E>::Edge::opposite.
- bool isAdjacentTo (Edge f)

  Graph<E>::Edge::isAdjacentTo.
- bool isIncidentOn (Vertex v)

  Test whether this edge is incident on v.
- bool **visited** ()
- QString print ()

  prints the Edge
- int & **operator∗** ()
- bool **operator==** (const Edge &other) const
- bool **operator!=** (const Edge &other) const
- bool **operator>** (const Edge &other) const
- bool **operator<** (const Edge &other) const
- bool **operator>=** (const Edge &other) const
- bool **operator<=** (const Edge &other) const

Friends

- QDebug **operator<<** (QDebug output, const Edge &obj)
- QTextStream & **operator<<** (QTextStream &output, const Edge &obj)

4.4.1 Detailed Description

template<typename E>
class Graph< E >::Edge

The Edge class Edge within the graph holds the weight between two incident vertecies and methods to manipulate that data and access the adjacent vertrices.

4.4.2 Member Function Documentation

4.4.2.1 template<typename E > bool Graph< E >::Edge::isAdjacentTo (Edge *f*)

Graph<E>::Edge::isAdjacentTo.

Parameters

f	

Returns

4.4.2.2 template<typename E > bool Graph< E >::Edge::isIncidentOn (Vertex *v*)

Test whether this edge is incident on v.

Parameters

v	[IN] vertex to test with

Returns

TRUE if this edge is incident on vertex 'v'

4.4.2.3 template<typename E > Graph< E >::VertexItr Graph< E >::Edge::opposite (Vertex *v*)

Graph<E>::Edge::opposite.

Parameters

v	

Returns

4.4.2.4 template<typename E > QString Graph< E >::Edge::print ( )

prints the Edge

Returns

A string representation of the edge

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

- src/header/graph.h

4.5 Entry< K, V > Class Template Reference

This class describes a key-value pair.

```
#include <entry.h>
```

Public Types

- enum **STATE** { **OCCUPIED**, **EMPTY**, **DELETED** }

Public Member Functions

- Entry ()

  Default Constructor sets to empty values.
- Entry (const K &k, const V &v)

  Non-Default consturctor sets key and value.
- const K & key () const

  RETURNS the KEY of the entry.
- K & **key** ()
- void setKey (const K &k)

  SETS the KEY of the value.
- const V & value () const

  RETURNS the VALUE of the entry.
- V & **value** ()
- void setValue (const V &v)

  SETS the VALUE of the entry, and changes the state to OCCUPIED.
- void clear ()

  CLEARS the entry and set to DELETED state.
- STATE state () const

  returns the state of the entry
- bool empty () const

  returns TRUE if the entry is empty
- bool deleted () const

  returns TRUE if the entry was deleted
- bool **operator<** (const Entry &that) const
- bool **operator>** (const Entry &that) const
- bool **operator<=** (const Entry &that) const
- bool **operator>=** (const Entry &that) const
- bool **operator==** (const Entry &that) const
- bool **operator!=** (const Entry &that) const

Friends

- QDebug **operator<<** (QDebug output, const Entry &obj)

4.5.1 Detailed Description

template<typename K, typename V>
class Entry< K, V >

This class describes a key-value pair.

Author

Ethan Slattery

Date

12APR2016

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

- src/header/entry.h

4.6 Graph< E > Class Template Reference

Undirected Graph A graph with built in algorithms and features. Uses a adjacency list structure for implementation with iterators used to pass references to the data around instead of copies of the data.

```
#include <graph.h>
```

Classes

- class Edge

  The Edge class Edge within the graph holds the weight between two incident vertecies and methods to manipulate that data and access the adjacent vertrices.
- class Vertex

  The Vertex class Vertex insisde the graph holds the data methods for workign with vertices.

Public Types

- typedef std::list< Vertex > **VertexList**
- typedef std::list< Edge > **EdgeList**
- typedef VertexList::iterator **VertexItr**
- typedef EdgeList::iterator **EdgeItr**
- typedef std::list< EdgeItr > **EdgeItrList**
- typedef EdgeItrList::iterator **EdgeItrItr**
- typedef HeapPriorityQueue< Edge, std::less< Edge > > **EdgePQueue**
- typedef std::vector< VertexItr > **VertexItrVector**

Public Member Functions

- Graph ()
- void insertVertex (const E &e)

    Adds a vertex to the graph with the data 'x'.
- void insertEdge (const E &v, const E &w, const int &x)

    Inserts a new undirected edge connecting 'v' and 'w' and storing 'x'.
- void eraseVertex (const E &e)

    Erases the given vertex and all edges incident.
- void eraseEdge (const E &v, const E &w, const int &x)

    Erases the edge with the given start, end, and weight.
- VertexList **vertices** ()
- EdgeList **edges** ()
- int **numVertices** ()
- int **numEdges** ()
- void print (std::ofstream &output, std::string title="Graph Output")

    prints the graph to a dot output file with the given title
- VertexList dft (const E &e)
- void Dijkstra (const E &e)

    creates shortest path tree starting at specified vertex
- int **GetDistanceTo** (const E &e)
- int **GetDistance** (const E &start, const E &end)
- VertexList shortestPathTo (const E &end)

    Gets the distance from last dijkstra origin to end.
- EdgeList MSTPrim ()

    Graph< E >::MSTPrim.
- EdgeList PrimJarnek ()

    creates the Minimum Spannin Tree using Prim-Jarnek and PQueue
- VertexItr findVertex (const E &e)

    Finds the vertex with data 'e'.
- EdgeItr findEdge (const E &v, const E &w, const int &x)

    finds the edge matching the given criteria

Protected Member Functions

- void unvisitAll ()

    Graph< E >::unvisitAll.
- void resetDijkstra ()

    Graph::resetDijkstra.
- void dftHelper (Vertex &location, VertexList &outList)

    Depth frist traversal of the graph.

4.6.1 Detailed Description

template<typename E>
class Graph< E >

Undirected Graph A graph with built in algorithms and features. Uses a adjacency list structure for implementation with iterators used to pass references to the data around instead of copies of the data.

Author

Ethan Slattery

Date

12APR2016

4.6.2 Constructor & Destructor Documentation

4.6.2.1 template<typename E> Graph< E >::Graph ( ) [inline]

ACTUAL GRAPH INTERFACE

4.6.3 Member Function Documentation

4.6.3.1 template<typename E> Graph< E >::VertexList Graph< E >::dft (const E & *e*)

performs a depth first traversal starting at vertex E

Parameters

e	[IN] the starting element

4.6.3.2 template<typename E > void Graph< E >::dftHelper (Vertex & *location,* VertexList & *outList*) [protected]

Depth frist traversal of the graph.

Parameters

location	[IN] the location to begin at
outList	[OUT] the list of vertex in depth first order

4.6.3.3 template<typename E> void Graph< E >::Dijkstra (const E & *e*)

creates shortest path tree starting at specified vertex

Parameters

e	[IN] the starting vertex

4.6.3.4 template<typename E> void Graph< E >::eraseEdge (const E & *v,* const E & *w,* const int & *x*)

Erases the edge with the given start, end, and weight.

Parameters

v	[IN] vertex data at 'start' of edge to delete
w	[IN] vertex data at 'end' of edge to delete
x	[IN] weight of the edge as an integer

4.6.3.5 template<typename E> void Graph< E >::eraseVertex (const E & *e*)

Erases the given vertex and all edges incident.

Parameters

e	[IN] The vertex to remove

4.6.3.6 template<typename E> Graph< E >::EdgeItr Graph< E >::findEdge (const E & *v*, const E & *w*, const int & *x*)

finds the edge matching the given criteria

Parameters

v	[IN] vertex data at 'start' of edge
w	[IN] vertex data at 'end' of edge
x	[IN] weight of the edge as an integer

Returns

iterator to the edge or EdgeItrList::end() if not found

4.6.3.7 template<typename E> Graph< E >::VertexItr Graph< E >::findVertex (const E & *e*)

Finds the vertex with data 'e'.

Parameters

e	[IN] the data to find in a given vertex

Returns

iterator to the vertex containing 'e' or VertexList::end() if not found

4.6.3.8 template<typename E> void Graph< E >::insertEdge (const E & *v*, const E & *w*, const int & *x*)

Inserts a new undirected edge connecting 'v' and 'w' and storing 'x'.

Parameters

v	[IN] vertex data at 'start' of edge
w	[IN] vertex data at 'end' of edge
x	[IN] weight of the edge as an integer

4.6.3.9 template<typename E> void Graph< E >::insertVertex (const E & *e*)

Adds a vertex to the graph with the data 'x'.

Parameters

e	[IN] the data to insert into the new vertex

4.6.3.10 template<typename E > Graph< E >::EdgeList Graph< E >::MSTPrim ( )

Graph<E>::MSTPrim.

Returns

A list of edge objects

4.6.3.11 template<typename E > Graph< E >::EdgeList Graph< E >::PrimJarnek ( )

creates the Minimum Spannin Tree using Prim-Jarnek and PQueue

Returns

A list of edge objects that make up the MST

4.6.3.12 template<typename E > void Graph< E >::print (std::ofstream & *output*, std::string *title* = `"Graph< E > Output"`)

prints the graph to a dot output file with the given title

Parameters

output	[IN] reference to an output file stream
title	[IN] Title for the graph

4.6.3.13 template<typename E> Graph< E >::VertexList Graph< E >::shortestPathTo (const E & *end*)

Gets the distance from last dijkstra origin to end.

Parameters

end	[IN] The ending vertex

Returns

ordered vector if vertices represeting path from last dijkstra origin to end

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

- src/header/graph.h

4.7 HeapPriorityQueue< E, C > Class Template Reference

A heap based priority queue This class implements a heap based priority queue, using a vector as the underlying structure. The data to be stored and the comparator is templated. typename E - The data to store in the heap typename C - The comparator to use while sorting the queue.

```
#include <HeapPriorityQueue.h>
```

Public Member Functions

- int size () const

  size
- bool empty () const

  empty
- E & top ()

  top
- void push (const E &e)

  push
- void pop ()

  pop

4.7.1 Detailed Description

template<typename E, typename C>
class HeapPriorityQueue< E, C >

A heap based priority queue This class implements a heap based priority queue, using a vector as the underlying structure. The data to be stored and the comparator is templated. typename E - The data to store in the heap typename C - The comparator to use while sorting the queue.

Author

Ethan Slattery

Date

12APR2016

4.7.2 Member Function Documentation

4.7.2.1 template<typename E , typename C > bool HeapPriorityQueue< E, C >::empty ( ) const [inline]

empty

Returns

4.7.2.2 template<typename E , typename C > void HeapPriorityQueue< E, C >::pop ( )

pop

This method removes the next element from the queue. It then restores heap order by bubbling down

4.7.2.3 template<typename E , typename C > void HeapPriorityQueue< E, C >::push (const E & *e*)

push

Parameters

e	This method adds the element e to the heap. It then performs a bubble up to restore heap order.
e	[IN] The element to add

4.7.2.4 template<typename E , typename C > int HeapPriorityQueue< E, C >::size ( ) const [inline]

size

Returns

4.7.2.5 template<typename E , typename C > E& HeapPriorityQueue< E, C >::top ( ) [inline]

top

Returns

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

- src/header/HeapPriorityQueue.h

4.8 skiplist< K, V >::Iterator Class Reference

Skip List Iterator This class describes a skip list iterator for moving through the skip list.

```
#include <skiplist.h>
```

Public Member Functions

- Iterator (node ∗position)

  Basic Iterator constructor to the given position, always points to bottom level.
- const V & operator∗ () const

  Returns a read only version of value at this location.
- V & operator∗ ()

  Returns a read/write version of the value at this location.
- bool operator== (const Iterator &p) const

  Returns TRUE if iterators point to the same position.
- bool operator!= (const Iterator &p) const

  Returns TRUE if iterators does not point to the same position.
- Iterator operator++ (int)

  Traverse the list in the forward direction - Postfix requires int parameter?
- Iterator & operator++ ()

  Traverse the list in the forward direction - prefix with no int parameter?
- Iterator & operator-- ()

  Traverse the list in the reverse direction.

4.8.1 Detailed Description

template<typename K, typename V>
class skiplist< K, V >::Iterator

Skip List Iterator This class describes a skip list iterator for moving through the skip list.

4.8.2 Constructor & Destructor Documentation

4.8.2.1 template<typename K, typename V> skiplist< K, V >::Iterator::Iterator (node ∗ *position*) [inline]

Basic Iterator constructor to the given position, always points to bottom level.

Iterator

Parameters

position	

4.8.3 Member Function Documentation

4.8.3.1 template<typename K, typename V> bool skiplist< K, V >::Iterator::operator!= (const Iterator & *p*) const [inline]

Returns TRUE if iterators does not point to the same position.

operator !=

Parameters

p	

Returns

4.8.3.2 template<typename K, typename V> const V& skiplist< K, V >::Iterator::operator∗ ( ) const [inline]

Returns a read only version of value at this location.

operator ∗

Returns

4.8.3.3 template<typename K, typename V> V& skiplist< K, V >::Iterator::operator∗ ( ) [inline]

Returns a read/write version of the value at this location.

operator ∗

Returns

4.8.3.4 template<typename K, typename V> Iterator skiplist< K, V >::Iterator::operator++ (int) [inline]

Traverse the list in the forward direction - Postfix requires int parameter?

operator ++

Returns

4.8.3.5 template<typename K, typename V> Iterator& skiplist< K, V >::Iterator::operator++ ( ) [inline]

Traverse the list in the forward direction - prefix with no int parameter?

operator ++

Returns

4.8.3.6 template<typename K, typename V> Iterator& skiplist< K, V >::Iterator::operator-- ( ) [inline]

Traverse the list in the reverse direction.

operator –

Returns

4.8.3.7 template<typename K, typename V> bool skiplist< K, V >::Iterator::operator== (const Iterator & *p*) const [inline]

Returns TRUE if iterators point to the same position.

operator ==

Parameters

p	

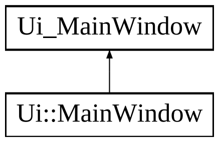
Returns

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

- src/header/skiplist.h

4.9 Ui::MainWindow Class Reference

Inheritance diagram for Ui::MainWindow:



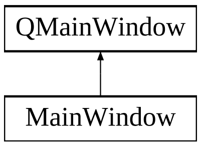
Additional Inherited Members

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

- src/ui_mainwindow.h

4.10 MainWindow Class Reference

Inheritance diagram for MainWindow:



Public Member Functions

- MainWindow (QWidget ∗parent=0)

  MainWindow::MainWindow Displays the window of the program. Retrieves all needed information from the database for this program, such as stadiums and a universal global graph of all the connecting edges and vertex. The constructor also intializes the search bar.
- ∼MainWindow ()

  MainWindow::∼MainWindow Properly closes the window and exits program.
- bool isBlank (QString text)

  MainWindow::isBlank A helper method to check if a QString is blank or not.
- void addToCart (Souvenir ∗s)

  MainWindow::addToCart A helper method to ensure the reqeusted souvenir to be potentially bought is added to the shopping cart.
- void tripProcess (QVector< Stadium ∗ > trip)

  MainWindow::tripProcess A method that helps initialize every time the user clicks the button 'next' on their trip. Such as initializing the current stadium their visiting souvenir's list, along with updating the mileage (how much they have traveled) and how far they are into the trip (ex. 1/5 stadiums have been visisted).
- void tripProcess2 (QVector< VertexItr > trip)

  MainWindow::tripProcess2 This method exactly similar to the original trip process, however their trip process is meant for custom trips only. As it recursively calls the Dijkstra formula to find the next stadium to visit.

4.10.1 Constructor & Destructor Documentation

4.10.1.1 MainWindow::MainWindow (QWidget ∗ *parent* = 0) [explicit]

MainWindow::MainWindow Displays the window of the program. Retrieves all needed information from the database for this program, such as stadiums and a universal global graph of all the connecting edges and vertex. The constructor also intializes the search bar.

Parameters

parent	

4.10.2 Member Function Documentation

4.10.2.1 void MainWindow::addToCart (Souvenir $*$ *s*)

MainWindow::addToCart A helper method to ensure the reqeusted souvenir to be potentially bought is added to the shopping cart.

Adds a desired souvenir to the global shopping cart

Parameters

s	

First checks if the item already exists within the shopping cart, if so it will add to its quantity

Checks if an existing souvenir was found or not

If not found, pushes the new souvenir to the end of the shoppingCart list

4.10.2.2 bool MainWindow::isBlank (QString *text*)

MainWindow::isBlank A helper method to check if a QString is blank or not.

T/F if a string is empty/blank.

Parameters

text	

Returns

T/F if the QString is blank

4.10.2.3 void MainWindow::tripProcess (QVector$<$ Stadium $*$ $>$ *trip*)

MainWindow::tripProcess A method that helps initialize every time the user clicks the button 'next' on their trip. Such as initializing the current stadium their visiting souvenir's list, along with updating the mileage (how much they have traveled) and how far they are into the trip (ex. 1/5 stadiums have been visisted).

Trip process for only quick trip request

Parameters

trip	

Clears the current stadium sovenirs table, to prepare for new stadium's souvenir list

Waits until user clicks 'next' button

4.10.2.4 void MainWindow::tripProcess2 (QVector< VertexItr > *trip*)

MainWindow::tripProcess2 This method exactly similar to the original trip process, however their trip process is meant for custom trips only. As it recursively calls the Dijkstra formula to find the next stadium to visit.

Trip process for custom trip request only

Parameters

trip	

Clears the current stadium sovenirs table, to prepare for new stadium's souvenir list

Waits until user clicks 'next' button

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

- src/header/mainwindow.h
- src/source/mainwindow.cpp

4.11 skiplist< K, V >::node Class Reference

A skip list node.

`#include <skiplist.h>`

Public Types

- enum nodeType { **REGULAR**, **HEAD**, **TAIL** }

  The nodeType enum.

Public Member Functions

- node ()

  Default constructor for the node sets all values to null.
- node (node ∗up, node ∗down, node ∗left, node ∗right, nodeType Type=REGULAR)

  node
- void setUp (node ∗up)

  Sets the node up from this node.
- void setDown (node ∗down)

  Sets the node down from this node.
- void setLeft (node ∗left)

  Sets the node left from this node.
- void setRight (node ∗right)

  Sets the node right from this node.
- node ∗ up () const

  Gets the node up from this node.
- node ∗ down () const

Gets the node down from this node.
- node $*$ left () const
  Gets the node left from this node.
- node $*$ right () const
  Gets the node right from this node.
- nodeType type () const
  Returns the type of node for comparison purposes.
- K key () const
  returns the key of this node
- V & value ()
  Returns the value in this node.
- bool empty () const
  Returns TRUE if the data list for this node is empty.
- void setNodeType (const nodeType &n)
  sets the node status as a head node
- void add (const item &e)
  Adds an element to this node.
- void add (const K &k, const V &v)
  Adds an element to this node by values.
- void clear ()
  Clears the data from this node.
- bool operator< (const node &that) const
  overloads the < operator
- bool operator> (const node &that) const
  overloads the > operator
- bool operator<= (const node &that) const
- bool operator>= (const node &that) const
- bool operator== (const node &that) const
- bool operator!= (const node &that) const

4.11.1 Detailed Description

template<typename K, typename V>
class skiplist< K, V >::node

A skip list node.

4.11.2 Constructor & Destructor Documentation

4.11.2.1 template<typename K, typename V> skiplist< K, V >::node::node ( ) [inline]

Default constructor for the node sets all values to null.

node

4.11.2.2 template<typename K , typename V > skiplist< K, V >::node::node (node $*$ *up*, node $*$ *down*, node $*$ *left*, node $*$ *right*, nodeType *Type* = REGULAR)

node

The non-default constructor of the node class, where you can set the links.

Parameters

up	
down	
left	
right	
Type	
up	
down	
left	
right	
type	

4.11.3 Member Function Documentation

4.11.3.1 template<typename K, typename V> void skiplist< K, V >::node::add (const item & *e*) `[inline]`

Adds an element to this node.

add

Parameters

e	

4.11.3.2 template<typename K, typename V> void skiplist< K, V >::node::add (const K & *k*, const V & *v*) `[inline]`

Adds an element to this node by values.

add

Parameters

k	
v	

4.11.3.3 template<typename K, typename V> void skiplist< K, V >::node::clear ( ) `[inline]`

Clears the data from this node.

clear

4.11.3.4 template<typename K, typename V> node∗ skiplist< K, V >::node::down ( ) const `[inline]`

Gets the node down from this node.

down

Returns

4.11.3.5 template<typename K, typename V> bool skiplist< K, V >::node::empty ( ) const [inline]

Returns TRUE if the data list for this node is empty.

empty

Returns

4.11.3.6 template<typename K, typename V> K skiplist< K, V >::node::key ( ) const [inline]

returns the key of this node

key

Returns

4.11.3.7 template<typename K, typename V> node* skiplist< K, V >::node::left ( ) const [inline]

Gets the node left from this node.

left

Returns

4.11.3.8 template<typename K , typename V > bool skiplist< K, V >::node::operator!= (const node & *that*) const

overloads the != operator

Parameters

that	

4.11.3.9 template<typename K , typename V > bool skiplist< K, V >::node::operator< (const node & *that*) const

overloads the < operator

Parameters

that	

4.11.3.10 template<typename K , typename V > bool skiplist< K, V >::node::operator<= (const node & *that*) const

overloads the <= operator

Parameters

that	

4.11.3.11 template<typename K , typename V > bool skiplist< K, V >::node::operator== (const node & *that*) const

overloads the == operator

Parameters

that	

4.11.3.12 template<typename K , typename V > bool skiplist< K, V >::node::operator> (const node & *that*) const

overloads the > operator

Parameters

that	

4.11.3.13 template<typename K , typename V > bool skiplist< K, V >::node::operator>= (const node & *that*) const

overloads the >= operator

Parameters

that	

4.11.3.14 template<typename K, typename V> node∗ skiplist< K, V >::node::right ( ) const `[inline]`

Gets the node right from this node.

right

Returns

4.11.3.15 template<typename K, typename V> void skiplist< K, V >::node::setDown (node * *down*) [inline]

Sets the node down from this node.

setDown

Parameters

down	

4.11.3.16 template<typename K, typename V> void skiplist< K, V >::node::setLeft (node * *left*) [inline]

Sets the node left from this node.

setLeft

Parameters

left	

4.11.3.17 template<typename K, typename V> void skiplist< K, V >::node::setNodeType (const nodeType & *n*) [inline]

sets the node status as a head node

setNodeType

Parameters

n	

4.11.3.18 template<typename K, typename V> void skiplist< K, V >::node::setRight (node * *right*) [inline]

Sets the node right from this node.

setRight

Parameters

right	

4.11.3.19 template<typename K, typename V> void skiplist< K, V >::node::setUp (node * *up*) [inline]

Sets the node up from this node.

setUp

Parameters

up	

4.11.3.20 template<typename K, typename V> nodeType skiplist< K, V >::node::type ( ) const [inline]

Returns the type of node for comparison purposes.

type

Returns

4.11.3.21 template<typename K, typename V> node* skiplist< K, V >::node::up ( ) const [inline]

Gets the node up from this node.

up

Returns

4.11.3.22 template<typename K, typename V> V& skiplist< K, V >::node::value ( ) [inline]

Returns the value in this node.

value

Returns

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

- src/header/skiplist.h

4.12 PriorityQueue< T > Class Template Reference

Public Member Functions

- void **insert** (T element)
- T **min** ()
- T **removeMin** ()
- int **size** ()
- bool **empty** ()

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

- src/header/priorityqueue.h

4.13 qt_meta_stringdata_MainWindow_t Struct Reference

Public Attributes

- QByteArrayData **data** [46]
- char **stringdata0** [1349]

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

- src/debug/moc_mainwindow.cpp

4.14 qt_meta_stringdata_ShoppingCart_t Struct Reference

Public Attributes

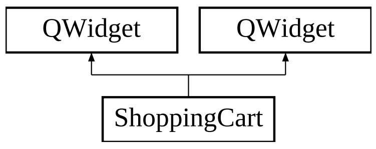
- QByteArrayData **data** [1]
- char **stringdata0** [13]

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

- src/debug/moc_shoppingcart.cpp

4.15 ShoppingCart Class Reference

Inheritance diagram for ShoppingCart:



Public Member Functions

- ShoppingCart (QWidget ∗parent=0)

  ShoppingCart::ShoppingCart Initializes the ui to appear on the screen.
- ∼ShoppingCart ()

  ShoppingCart::∼ShoppingCart Destructor, closes the ui properly.
- **ShoppingCart** (QWidget ∗parent=0)
- void setList (QVector< Souvenir ∗ > shoppingCart, skiplist< int, Stadium ∗ > stadiums)

  ShoppingCart::setList Initializes all needed functionality for ui to display a table (QTreeWidget) on the gui with sub-cost, total cost, total quantity of each item from the customer's shopping experience. Along with an overall grand total.

4.15.1 Constructor & Destructor Documentation

4.15.1.1 ShoppingCart::ShoppingCart (QWidget ∗ *parent* = 0) [explicit]

ShoppingCart::ShoppingCart Initializes the ui to appear on the screen.

Parameters

parent	

4.15.2 Member Function Documentation

4.15.2.1 void ShoppingCart::setList (QVector< Souvenir ∗ > *shoppingCart*, skiplist< int, Stadium ∗ > *stadiums*)

ShoppingCart::setList Initializes all needed functionality for ui to display a table (QTreeWidget) on the gui with sub-cost, total cost, total quantity of each item from the customer's shopping experience. Along with an overall grand total.

ACCESSORS

Parameters

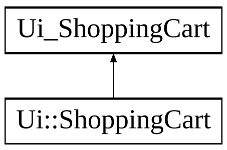
shoppingCart	
stadiums	

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

- src/form/shoppingcart.h
- src/form/shoppingcart.cpp

4.16 Ui::ShoppingCart Class Reference

Inheritance diagram for Ui::ShoppingCart:



Additional Inherited Members

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

- src/ui_shoppingcart.h

4.17 skiplist< K, V > Class Template Reference

A skip list implementation of a map, keys must be unique.

```
#include <skiplist.h>
```

Classes

- class Iterator

    Skip List Iterator This class describes a skip list iterator for moving through the skip list.
- class node

    A skip list node.

Public Member Functions

- skiplist ()

    skiplist
- void insert (const K &k, const V &v)

    insert
- void erase (const K &k)

    erase
- Iterator get (const K &k)

    get
- int size () const

    size
- int height () const

    Returns the height of the list, or the number of levels in the skip list.
- std::string print () const

    print
- std::string printVert () const

    printVert
- Iterator begin ()

    begin
- Iterator end ()

    end

Protected Types

- typedef Entry< K, V > **item**

Protected Member Functions

- void insert (const item &e)

    insert
- bool flipCoin ()

    Gets a random value between 0 and 1.
- int column (node ∗n) const

    column
- node ∗ search (const K &k) const

    search
- void addBlankLevel ()

    addBlankLevel

4.17.1 Detailed Description

template<typename K, typename V>
class skiplist< K, V >

A skip list implementation of a map, keys must be unique.

Author

Ethan Slattery

Date

12APR2016

4.17.2 Constructor & Destructor Documentation

4.17.2.1 template<typename K , typename V > skiplist< K, V >::skiplist ( )

skiplist

The constructor for the skip-list, it makes the basic linkage

4.17.3 Member Function Documentation

4.17.3.1 template<typename K , typename V > void skiplist< K, V >::addBlankLevel ( ) [protected]

addBlankLevel

Adds a new empty level above all the current levels in the list.

4.17.3.2 template<typename K, typename V> Iterator skiplist< K, V >::begin ( ) `[inline]`

begin

Returns

4.17.3.3 template<typename K , typename V > int skiplist< K, V >::column (node * *n*) const `[protected]`

column

Parameters

n	

Returns

the column of the node

Parameters

n	[IN] the node to find the column of

4.17.3.4 template<typename K, typename V> Iterator skiplist< K, V >::end ( ) `[inline]`

end

Returns

4.17.3.5 template<typename K, typename V > void skiplist< K, V >::erase (const K & *k*)

erase

Removes the given key and associated value from the list.

Parameters

k	
k	[IN] The key to erase from the list

4.17.3.6 template<typename K, typename V> bool skiplist< K, V >::flipCoin ( ) [inline], [protected]

Gets a random value between 0 and 1.

flipCoin

Returns

4.17.3.7 template<typename K, typename V> Iterator skiplist< K, V >::get (const K & *k*) [inline]

get

Parameters

k	

Returns

4.17.3.8 template<typename K, typename V> int skiplist< K, V >::height ( ) const [inline]

Returns the height of the list, or the number of levels in the skip list.

height

Returns

4.17.3.9 template<typename K, typename V> void skiplist< K, V >::insert (const K & *k*, const V & *v*) [inline]

insert

Parameters

k	
v	

4.17.3.10 template<typename K, typename V> void skiplist< K, V >::insert (const item & *e*) [protected]

insert

Adds an entry to the skip list.

Parameters

e	
e	[IN] the entry to add to the skip list

4.17.3.11 template<typename K , typename V > std::string skiplist< K, V >::print ( ) const

print

generates ASCII output of the list and returns it through the string

Returns

string containing horizontal ASCII representation of the skip list

4.17.3.12 template<typename K , typename V > std::string skiplist< K, V >::printVert ( ) const

printVert

generates ASCII output of the list and returns it through the string

Returns

string containing vertical ASCII representation of the skip list

4.17.3.13 template<typename K, typename V > skiplist< K, V >::node ∗ skiplist< K, V >::search (const K & *k*) const `[protected]`

search

Parameters

k	

Returns

Finds the key in the list This search will find the node in the base list with the largest key equal to or less than the value of the key being searched for.

Parameters

k	[IN] The key to find

4.17.3.14 template<typename K, typename V> int skiplist< K, V >::size ( ) const [inline]

size

Returns

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

- src/header/skiplist.h

4.18 Souvenir Class Reference

The Souvenir class This class represents a souvenir. A souvenir has a name, price, and qty. A souvenir also has a key to link it to a stadium it belongs to.

`#include <souvenir.h>`

Public Member Functions

- Souvenir ()

  Souvenir constructor.
- Souvenir (unsigned int id, QString name, double price, unsigned int qty)

  Souvenir::Souvenir.
- ~Souvenir ()

  Souvenir destructor.
- unsigned int getStadiumID () const

  gets the stadium ID of the owner stadium
- QString getName () const

  gets the name of the Souvenir
- double getPrice () const

  Souvenir::getPrice.
- unsigned int getQuantity () const

  Souvenir::getQuantity.
- void setName (QString newName)

  Souvenir::setName.
- void setPrice (double newPrice)

  Souvenir::setPrice.
- void setStadiumID (int id)

  Souvenir::setStadiumID.
- void setQuantity (unsigned int newQty)

  Souvenir::setQuantity.
- void addToQuantity (unsigned int addQty)

  Souvenir::addToQuantity.

4.18.1 Detailed Description

The Souvenir class This class represents a souvenir. A souvenir has a name, price, and qty. A souvenir also has a key to link it to a stadium it belongs to.

4.18.2 Constructor & Destructor Documentation

4.18.2.1 Souvenir::Souvenir (unsigned int *id,* QString *name,* double *price,* unsigned int *qty*)

Souvenir::Souvenir.

Parameters

id	[IN] id of the stadium this Souvenir belongs to
name	[IN] name of the Souvenir
price	[IN] price of the Souvenir
qty	[IN] quantity of the Souvenir

4.18.3 Member Function Documentation

4.18.3.1 void Souvenir::addToQuantity (unsigned int *addQty*)

Souvenir::addToQuantity.

Parameters

addQty	[IN] the quantity to add to the current qty

4.18.3.2 QString Souvenir::getName ( ) const

gets the name of the Souvenir

Returns

the Souvenirs name

4.18.3.3 double Souvenir::getPrice ( ) const

Souvenir::getPrice.

Returns

the price of the Souvenir

4.18.3.4 unsigned int Souvenir::getQuantity ( ) const

Souvenir::getQuantity.

Returns

the quantity of the Souvenir

4.18.3.5 unsigned int Souvenir::getStadiumID ( ) const

gets the stadium ID of the owner stadium

Returns

the stadium ID of owning stadium

4.18.3.6 void Souvenir::setName (QString *newName*)

Souvenir::setName.

Parameters

newName	[IN] the new name

4.18.3.7 void Souvenir::setPrice (double *newPrice*)

Souvenir::setPrice.

Parameters

newPrice	[IN] the new price

4.18.3.8 void Souvenir::setQuantity (unsigned int *newQty*)

Souvenir::setQuantity.

Parameters

newQty	[IN] the new quantity

4.18.3.9 void Souvenir::setStadiumID (int *id*)

Souvenir::setStadiumID.

Parameters

id	[IN] the ID of the owning stadium to change to

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

- src/header/souvenir.h
- src/source/souvenir.cpp

4.19 Stadium Class Reference

The Stadium class This class represents a stadium with the attributes of the stadium name, the team name, stadium address, box office number, seating capacity, type of surface, and type of league type (National or American) Stadium also keeps track of its ID, to enable changing in the database. This class also allows souvenir items to be added and removed. If souvenir item's name or price needs to be changed, must first search for souvenir item, then change it's attributes using souvenir's set methods.

```
#include <stadium.h>
```

Public Member Functions

- Stadium ()

  Stadium::Stadium Non-default constructor.
- **Stadium** (int id, QString name, QString team, QString street, QString city, QString state, QString zipCode, QString number, QString date, unsigned int capacity, QString surf, QString league, QString typo, double revenue)
- **Stadium** (int id, QString name)
- **Stadium** (int id)
- ∼Stadium ()

  Stadium::∼Stadium Destructor.
- unsigned int getStadiumID () const

  getStadiumID id is based off it's id in database table.
- QString getStadiumName () const

  Stadium::getStadiumName.
- QString getTeamName () const

  Stadium::getTeamName.
- Address getAddress () const

  Stadium::getAddress returns address in qstring form ex: 12345 streetName, cityName, ST zipCode.
- QString getBoxOfficeNumber () const

  getBoxOfficeNumber returns box office number in qstring form
- QString getDateOpened () const

  Stadium::getDateOpened returns date opened in qstring form.
- unsigned int getSeatingCapacity () const

  Stadium::getSeatingCapacity.
- QString getSurface () const

  Stadium::getSurface.
- QString getLeagueType () const

  Stadium::getLeagueType.
- QString getTypology () const

  Stadium::getTypology.
- QVector< Souvenir > getSouvenirs () const

  Gets all the Souvenirs associated with this stadium.
- double getTotalRevenue () const

  Stadium::getTotalRevenue.
- void setStadiumName (QString newName)

  Stadium::setStadiumName Changes the stadium name to newName.
- void setTeamName (QString newTeam)

  setTeamName Changes the team name to newTeam
- void setAddress (QString streetAddress, QString city, QString state, QString zipCode)

  Stadium::setAddress Changes the address to new address.
- void setAddress (Address newAddress)

  Stadium::setAddress Changes the address to new address.
- void setBoxOfficeNumber (QString newNumber)

  Stadium::setBoxOfficeNumber Changes the box office number to newNumber.
- void setDateOpened (QString newDate)

  Stadium::setDateOpened Changes the date opened to newDate.
- void setSeatingCapacity (unsigned int newCapacity)

  Stadium::setSeatingCapacity Changes seating capacity to newCapacity.
- void setSurface (QString newSurface)

  Stadium::setSurface Changes surface to newSurface.

- void setLeagueType (QString newLeagueType)

  Stadium::setLeagueType Changes league type to newLeagueType.
- void setTypology (QString typo)

  Stadium::setTypology Changes typology to typo.
- void setTotalRevenue (double revenue)

  Stadium::setTotalRevenue Changes totalRevenue to revenue.
- void **addToTotalRevenue** (double addToRevenue)
- Souvenir ∗ findSouvenir (QString name)

  Stadium::findSouvenir.
- QJsonObject toJSON ()

  returns this stadium as a JSON object
- void addSouvenir (Souvenir ∗newSouvenir)

  Stadium::addSouvenir Adds a souvenir to the current stadium's list of souvenirs.
- void removeSouvenir (QString name)

  Stadium::removeSouvenir Removes a souvenir from the current stadium's list of souvenirs.
- bool **operator==** (const Stadium &that) const
- bool **operator!=** (const Stadium &that) const
- bool **operator<** (const Stadium &that) const
- bool **operator<=** (const Stadium &that) const
- bool **operator>** (const Stadium &that) const
- bool **operator>=** (const Stadium &that) const

Friends

- QDebug **operator<<** (QDebug output, const Stadium &obj)
- QTextStream & **operator<<** (QTextStream &output, const Stadium &obj)

4.19.1 Detailed Description

The Stadium class This class represents a stadium with the attributes of the stadium name, the team name, stadium address, box office number, seating capacity, type of surface, and type of league type (National or American) Stadium also keeps track of its ID, to enable changing in the database. This class also allows souvenir items to be added and removed. If souvenir item's name or price needs to be changed, must first search for souvenir item, then change it's attributes using souvenir's set methods.

Author

Sarah Singletary

Date

April-14-2016

4.19.2 Constructor & Destructor Documentation

4.19.2.1 Stadium::Stadium ( )

Stadium::Stadium Non-default constructor.

C O N S T R U C T O R & D E S T R U C T O R

Parameters

id	
name	
team	
address	
number	
capacity	
surf	
league	

4.19.3 Member Function Documentation

4.19.3.1 void Stadium::addSouvenir (Souvenir ∗ *newSouvenir*)

Stadium::addSouvenir Adds a souvenir to the current stadium's list of souvenirs.

Parameters

name	
price	
quantity	

4.19.3.2 Souvenir ∗ Stadium::findSouvenir (QString *name*)

Stadium::findSouvenir.

Parameters

name	[IN] the name of the Souvenir to find

Returns

the Souvenir object with name 'name'

4.19.3.3 Address Stadium::getAddress ( ) const

Stadium::getAddress returns address in qstring form ex: 12345 streetName, cityName, ST zipCode.

Returns

a QString address

4.19.3.4 QString Stadium::getBoxOfficeNumber ( ) const

getBoxOfficeNumber returns box office number in qstring form

Returns

a QString box office number

4.19.3.5 QString Stadium::getDateOpened ( ) const

Stadium::getDateOpened returns date opened in qstring form.

Returns

a QString date openeed

4.19.3.6 QString Stadium::getLeagueType ( ) const

Stadium::getLeagueType.

Returns

a QString league type

4.19.3.7 unsigned int Stadium::getSeatingCapacity ( ) const

Stadium::getSeatingCapacity.

Returns

an unsigned int seating capacity

4.19.3.8 QVector< Souvenir > Stadium::getSouvenirs ( ) const

Gets all the Souvenirs associated with this stadium.

Returns

vector of Souvenir objects

4.19.3.9 unsigned int Stadium::getStadiumID ( ) const

getStadiumID id is based off it's id in database table.

A C C E S S O R S

Returns

an unsigned int stadium id

4.19.3.10 QString Stadium::getStadiumName ( ) const

Stadium::getStadiumName.

Returns

a QString stadium name

4.19.3.11 QString Stadium::getSurface ( ) const

Stadium::getSurface.

Returns

a QString surface

4.19.3.12 QString Stadium::getTeamName ( ) const

Stadium::getTeamName.

Returns

a QString team name

4.19.3.13 double Stadium::getTotalRevenue ( ) const

Stadium::getTotalRevenue.

Returns

a double totalRevenue

4.19.3.14 QString Stadium::getTypology ( ) const

Stadium::getTypology.

Returns

a QString typology

4.19.3.15 void Stadium::removeSouvenir (QString *name*)

Stadium::removeSouvenir Removes a souvenir from the current stadium's list of souvenirs.

Parameters

name	

4.19.3.16 void Stadium::setAddress (QString *streetAddress,* QString *city,* QString *state,* QString *zipCode*)

Stadium::setAddress Changes the address to new address.

Parameters

address	
city	
state	
zipCode	

4.19.3.17 void Stadium::setAddress (Address *newAddress*)

Stadium::setAddress Changes the address to new address.

Parameters

newAddress	

4.19.3.18 void Stadium::setBoxOfficeNumber (QString *newNumber*)

Stadium::setBoxOfficeNumber Changes the box office number to newNumber.

Parameters

newNumber	

4.19.3.19 void Stadium::setDateOpened (QString *newDate*)

Stadium::setDateOpened Changes the date opened to newDate.

Parameters

newDate	

4.19.3.20 void Stadium::setLeagueType (QString *newLeagueType*)

Stadium::setLeagueType Changes league type to newLeagueType.

Parameters

newLeagueType	

4.19.3.21 void Stadium::setSeatingCapacity (unsigned int *newCapacity*)

Stadium::setSeatingCapacity Changes seating capacity to newCapacity.

Parameters

newCapacity	

4.19.3.22 void Stadium::setStadiumName (QString *newName*)

Stadium::setStadiumName Changes the stadium name to newName.

M U T A T O R S

Parameters

newName	

4.19.3.23 void Stadium::setSurface (QString *newSurface*)

Stadium::setSurface Changes surface to newSurface.

Parameters

newSurface	

4.19.3.24 void Stadium::setTeamName (QString *newTeam*)

setTeamName Changes the team name to newTeam

Parameters

newTeam	

4.19.3.25 void Stadium::setTotalRevenue (double *revenue*)

Stadium::setTotalRevenue Changes totalRevenue to revenue.

Parameters

revenue	

4.19.3.26 void Stadium::setTypology (QString *typo*)

Stadium::setTypology Changes typology to typo.

Parameters

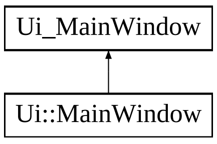
typo	

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

- src/header/stadium.h
- src/source/stadium.cpp

4.20 Ui_MainWindow Class Reference

Inheritance diagram for Ui_MainWindow:



Public Member Functions

- void **setupUi** (QMainWindow ∗MainWindow)
- void **retranslateUi** (QMainWindow ∗MainWindow)

Public Attributes

- QWidget ∗ **centralWidget**
- QStackedWidget ∗ **options**
- QWidget ∗ **customerOptions**
- QFrame ∗ **sidebarFrame**
- QPushButton ∗ **homePageButton**
- QPushButton ∗ **planATripButton**
- QPushButton ∗ **viewStadiumsPageButton**
- QWidget ∗ **adminOptions**
- QFrame ∗ **adminSidebarFrame**
- QPushButton ∗ **adminHomeButton**
- QPushButton ∗ **adminModifyButton**
- QPushButton ∗ **viewAdminStadiumsButton**

- QPushButton ∗ **adminModifyStadiumsButton**
- QStackedWidget ∗ **display**
- QWidget ∗ **homePage**
- QLabel ∗ **viewStadiumHeading_2**
- QFrame ∗ **frame_3**
- QLabel ∗ **label_2**
- QLabel ∗ **viewStadiumHeading_3**
- QLabel ∗ **label**
- QLabel ∗ **viewStadiumHeading_4**
- QLabel ∗ **viewStadiumHeading_5**
- QWidget ∗ **viewStadiumsPage**
- QLabel ∗ **viewStadiumHeading**
- QLabel ∗ **label_3**
- QComboBox ∗ **viewStadiumByComboBox**
- QTreeWidget ∗ **viewStadiumsList**
- QWidget ∗ **viewSingleStadiumPage**
- QLabel ∗ **singleStadiumNameLabel**
- QFrame ∗ **singleStadiumInfo**
- QWidget ∗ **verticalLayoutWidget**
- QVBoxLayout ∗ **singleStadiumLayoutLabels**
- QLabel ∗ **label_9**
- QLabel ∗ **label_12**
- QLabel ∗ **label_8**
- QLabel ∗ **label_5**
- QLabel ∗ **label_6**
- QLabel ∗ **label_10**
- QLabel ∗ **label_11**
- QLabel ∗ **totalRevenueLabel**
- QWidget ∗ **verticalLayoutWidget_2**
- QVBoxLayout ∗ **singleStadiumLayoutInfo**
- QLabel ∗ **singleStadiumTeamName**
- QLabel ∗ **singleStadiumType**
- QLabel ∗ **singleStadiumAddress**
- QLabel ∗ **singleStadiumBoxOfficeNum**
- QLabel ∗ **singleStadiumDateOpened**
- QLabel ∗ **singleStadiumSeatingCapacity**
- QLabel ∗ **singleStadiumSurface**
- QLabel ∗ **singleStadiumTotalRevenue**
- QWidget ∗ **planATripPage**
- QPushButton ∗ **shortestTripToAllButton**
- QPushButton ∗ **customTripButton**
- QPushButton ∗ **minimumSpanningTreeButton**
- QLabel ∗ **planATripLabel**
- QLabel ∗ **label_4**
- QWidget ∗ **quickTripToAllPage**
- QLabel ∗ **quickTripLabel**
- QTreeWidget ∗ **quickTripList**
- QLabel ∗ **quickTripStartingStadiumLabel**
- QLabel ∗ **quickTripStartingStadium**
- QLabel ∗ **quickTripDescription**
- QPushButton ∗ **quickTripTakeTripButton**
- QWidget ∗ **customTripPage**
- QLabel ∗ **customTripLabel**
- QTreeWidget ∗ **stadiumsToSelectFromList**
- QTreeWidget ∗ **selectedStadiumsList**

- QPushButton ∗ **confirmCustomTripButton**
- QLabel ∗ **startingStadiumLabel**
- QComboBox ∗ **startingStadiumComboBox**
- QPushButton ∗ **removeFromItineraryButton**
- QPushButton ∗ **addToItineraryButton**
- QWidget ∗ **minimumSpanningTreePage**
- QLabel ∗ **planATripLabel_2**
- QTreeWidget ∗ **MSTList**
- QLabel ∗ **planATripLabel_3**
- QLabel ∗ **mstTotalWeight**
- QWidget ∗ **tripProcessPage**
- QLabel ∗ **currentTripStadiumNameLabel**
- QPushButton ∗ **addSouvenirToShoppingCart**
- QLabel ∗ **currentTripWelcomeDescription**
- QTreeWidget ∗ **listOfCurrentStadiumSouvenirs**
- QLabel ∗ **currentTripNextButtonLabel**
- QLabel ∗ **currentTripSouvenirLabel**
- QPushButton ∗ **currentTripNextStadium**
- QPushButton ∗ **shoppingCartButton**
- QProgressBar ∗ **currentTripProgressBar**
- QLabel ∗ **currentTripProgressLabel**
- QLabel ∗ **currentTripStadiumCount**
- QLCDNumber ∗ **totalDistanceTraveled**
- QLabel ∗ **currentTripStadiumCount_2**
- QFrame ∗ **travel**
- QLabel ∗ **travelGif**
- QLabel ∗ **travelToName**
- QLabel ∗ **travelFromName**
- QWidget ∗ **confrimPurchasesPage**
- QLabel ∗ **confirmPurchasesLabel**
- QLabel ∗ **grandTotalLabel**
- QLabel ∗ **grandTotalAmount**
- QTreeWidget ∗ **shoppingCart**
- QLabel ∗ **shoppingCartLabel**
- QPushButton ∗ **confirmPurchasesButton**
- QLabel ∗ **shoppingCartEmpty**
- QLCDNumber ∗ **finalTotalDistance**
- QLabel ∗ **grandTotalLabel_2**
- QWidget ∗ **adminLoginPage**
- QLabel ∗ **usernameLabel**
- QLabel ∗ **passwordLabel**
- QPushButton ∗ **loginButton**
- QLineEdit ∗ **username**
- QLineEdit ∗ **password**
- QLabel ∗ **adminLoginErrorMessage**
- QLabel ∗ **adminLoginLabel**
- QWidget ∗ **adminHomePage**
- QLabel ∗ **adminHomePageLabel**
- QWidget ∗ **viewAdminStadiumsPage**
- QTreeWidget ∗ **adminStadiumList**
- QLabel ∗ **adminHomePageLabel_2**
- QPushButton ∗ **viewMoreInfoAboutStadiumButton**
- QLabel ∗ **label_7**
- QLabel ∗ **stadiumTotalRevenue**
- QLabel ∗ **stadiumTotalRevenueLabel**

- QWidget ∗ **adminModifyPage**
- QLabel ∗ **modifyInformationLabel**
- QLabel ∗ **modifyDescription**
- QTreeWidget ∗ **listOfModifyStadiums**
- QPushButton ∗ **modifyInformationNextButton**
- QWidget ∗ **modifySouvenirItemPage**
- QLabel ∗ **modifySouvenirsListLabel**
- QTreeWidget ∗ **listOfModifyStadiumsSouvenirs**
- QPushButton ∗ **removeSelectedSouvenir**
- QPushButton ∗ **addSelectedSouvenir**
- QLabel ∗ **newSouvenirNameLabel**
- QLineEdit ∗ **newSouvenirName**
- QLabel ∗ **newSouvenirPriceLabel**
- QLineEdit ∗ **newSouvenirPrice**
- QLabel ∗ **adminAddSouvenirErrorMessage**
- QLabel ∗ **modifyStadiumsDescr_2**
- QWidget ∗ **adminModifyStadiums**
- QPushButton ∗ **addStadiumFromFileButton**
- QPushButton ∗ **updateAStadiumButton**
- QLabel ∗ **modifyStadiumLabel**
- QTreeWidget ∗ **stadiumsToModifyList**
- QFrame ∗ **line**
- QFrame ∗ **line_2**
- QLabel ∗ **modifyStadiumOrLabel**
- QLabel ∗ **modifyStadiumsDescr**
- QLabel ∗ **modifyStadiumSelectDescr**
- QWidget ∗ **updateStadiumPage**
- QLabel ∗ **modifySouvenirsListLabel_2**
- QFrame ∗ **updateLeague**
- QHBoxLayout ∗ **horizontalLayout_5**
- QLabel ∗ **updateLeagueLabel**
- QRadioButton ∗ **updateAmericanLeague**
- QRadioButton ∗ **updateNationalLeague**
- QSpacerItem ∗ **horizontalSpacer_11**
- QFrame ∗ **frame**
- QFrame ∗ **updateStadiumName**
- QHBoxLayout ∗ **horizontalLayout_6**
- QLabel ∗ **updateStadiumLabel**
- QLineEdit ∗ **updateStadium**
- QSpacerItem ∗ **horizontalSpacer_12**
- QFrame ∗ **updateTeam**
- QHBoxLayout ∗ **horizontalLayout_7**
- QLabel ∗ **updateStadiumTeamNameLabel**
- QLineEdit ∗ **updateTeamName**
- QSpacerItem ∗ **horizontalSpacer_13**
- QPushButton ∗ **confirmStadiumUpdateButton**
- QPushButton ∗ **cancelStadiumUpdatesButton**
- QFrame ∗ **updateInformation**
- QGridLayout ∗ **gridLayout_4**
- QFrame ∗ **frame_2**
- QLineEdit ∗ **updateStreetAddress**
- QLineEdit ∗ **updateZipcode**
- QLineEdit ∗ **updateCity**
- QLineEdit ∗ **updateState**
- QLabel ∗ **updateAddress_2**

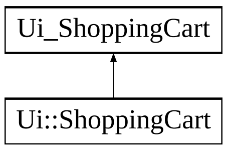
- QLabel ∗ **updateAddress_3**
- QLabel ∗ **updateAddress_4**
- QLabel ∗ **updateAddress_5**
- QSpacerItem ∗ **horizontalSpacer_16**
- QComboBox ∗ **updateMonth**
- QLineEdit ∗ **updateYear**
- QSpinBox ∗ **updateSeatingCapacity**
- QLineEdit ∗ **updatePhoneNumber**
- QLineEdit ∗ **updateTypology**
- QLabel ∗ **label_20**
- QLabel ∗ **label_22**
- QLabel ∗ **label_21**
- QLabel ∗ **label_19**
- QLineEdit ∗ **updateDay**
- QLabel ∗ **updateStadiumTeamNameLabel_2**
- QLabel ∗ **updateStadiumInvalidErrorMessage**
- QFrame ∗ **headerFrame**
- QLabel ∗ **teamNameLabel**
- QPushButton ∗ **adminLoginButton**
- QLineEdit ∗ **searchBar**
- QToolButton ∗ **searchButton**
- QToolButton ∗ **secretAdminLoginButton**

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

- src/ui_mainwindow.h

4.21 Ui_ShoppingCart Class Reference

Inheritance diagram for Ui_ShoppingCart:



Public Member Functions

- void **setupUi** (QWidget ∗ShoppingCart)
- void **retranslateUi** (QWidget ∗ShoppingCart)

Public Attributes

- QFrame ∗ **headerFrame**
- QLabel ∗ **teamNameLabel**
- QTreeWidget ∗ **shoppingCart**
- QLabel ∗ **shoppingCartLabel**
- QLabel ∗ **grandTotalLabel**
- QLabel ∗ **grandTotalAmount**
- QLabel ∗ **label**

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

- src/ui_shoppingcart.h

4.22 Graph< E >::Vertex Class Reference

The Vertex class Vertex insisde the graph holds the data methods for workign with vertices.

```
#include <graph.h>
```

Public Member Functions

- **Vertex** (const E &data)
- void **setData** (const E &data)
- void **visit** ()
- void **resetVisited** ()
- VertexItrVector adjacentVertex ()

  Get all the neightbors of this vertex.
- void **setDistance** (const int &newValue)
- int **getDistance** ()
- void **setParent** (const VertexItr &parent)
- VertexItr **getParent** ()
- void **resetDijkstra** ()
- EdgeItrList **incidentEdges** ()
- void **addEdge** (EdgeItr newEdge)
- void removeEdge (EdgeItr edge)

  Removed the edge pointed to by the given iterator from this vertex's adjaceny list.
- bool **visited** ()
- bool isAdjacentTo (const E &v)

  tests if this vertex is adjacent to vertex 'v'
- int distanceTo (const VertexItr &v)

  get the distance from this vertec to 'v'
- QString print ()

  prints the Vertex
- E & **operator∗** ()
- bool **operator==** (const Vertex &other) const
- bool **operator!=** (const Vertex &other) const
- bool **operator>** (const Vertex &other) const
- bool **operator<** (const Vertex &other) const
- bool **operator>=** (const Vertex &other) const
- bool **operator<=** (const Vertex &other) const

Friends

- QDebug **operator<<** (QDebug output, const Vertex &obj)
- QTextStream & **operator<<** (QTextStream &output, const Vertex &obj)

4.22.1 Detailed Description

template<typename E>
class Graph< E >::Vertex

The Vertex class Vertex insisde the graph holds the data methods for workign with vertices.

4.22.2 Member Function Documentation

4.22.2.1 template<typename E > Graph< E >::VertexItrVector Graph< E >::Vertex::adjacentVertex ( )

Get all the neighbors of this vertex.

Returns

A vector of iterators pointing to vertex adjacent to this one

4.22.2.2 template<typename E > int Graph< E >::Vertex::distanceTo (const VertexItr & *v*)

get the distance from this vertec to 'v'

Parameters

v	[IN] the vertex to get the distance to

Returns

distance between this vertex and 'v' (INF if not adjacent)

4.22.2.3 template<typename E> bool Graph< E >::Vertex::isAdjacentTo (const E & *v*)

tests if this vertex is adjacent to vertex 'v'

Parameters

v	[IN] the value of the vertex to check for

Returns

TRUE if vertex is adjacent

4.22.2.4 template<typename E > QString Graph< E >::Vertex::print ( )

prints the Vertex

Returns

A string representation of the vertex

4.22.2.5 template<typename E > void Graph< E >::Vertex::removeEdge (EdgeItr *edge*)

Removed the edge pointed to by the given iterator from this vertex's adjaceny list.

Parameters

edge	[IN] iterator to the edge to remove from this vertex

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

- src/header/graph.h

Chapter 5

File Documentation

5.1 src/header/CompleteTree.h File Reference

Assignment #7 - Heap Sort.

```
#include <vector>
```

Classes

- class CompleteTree< E >

  A Complete binary tree class This class creates a complete binary tree, or a tree where every level has the maximum number of nodes possible, and the nodes in the last level fill from left to right.

5.1.1 Detailed Description

Assignment #7 - Heap Sort.

Author

Ethan Slattery

Date

3-MAR-2016

5.2 src/header/graph.h File Reference

Undirected Graph.

```
#include <vector>
#include <list>
#include <queue>
#include <fstream>
#include <QDebug>
#include <climits>
#include <functional>
#include "HeapPriorityQueue.h"
#include "quicksort.h"
```

Classes

- class Graph< E >

  Undirected Graph A graph with built in algorithms and features. Uses a adjacency list structure for implementation with iterators used to pass references to the data around instead of copies of the data.
- class Graph< E >::Vertex

  The Vertex class Vertex insisde the graph holds the data methods for workign with vertices.
- class Graph< E >::Edge

  The Edge class Edge within the graph holds the weight between two incident vertecies and methods to manipulate that data and access the adjacent vertrices.

Macros

- #define **VERBOSE_DEBUG** 0
- #define **EXTRA_VERBOSE_DEBUG** 0
- #define **INF** INT_MAX

5.2.1 Detailed Description

Undirected Graph.

Author

Ethan Slattery && Osvaldo Moreno Ornelas

Date

7-APR-2016

5.3 src/header/HeapPriorityQueue.h File Reference

Assignment #7 - Heap Sort.

```
#include "CompleteTree.h"
```

Classes

- class HeapPriorityQueue< E, C >

  A heap based priority queue This class implements a heap based priority queue, using a vector as the underlying structure. The data to be stored and the comparator is templated. typename E - The data to store in the heap typename C - The comparator to use while sorting the queue.

5.3.1 Detailed Description

Assignment #7 - Heap Sort.

Author

Ethan

Date

06-Mar-2016

Index