

# Throw Catch

1.0

Generated by Doxygen 1.8.11



# Contents

<b>1</b>	<b>Hierarchical Index</b>	<b>1</b>
1.1	Class Hierarchy . . . . .	1
<b>2</b>	<b>Class Index</b>	<b>3</b>
2.1	Class List . . . . .	3
<b>3</b>	<b>File Index</b>	<b>5</b>
3.1	File List . . . . .	5
<b>4</b>	<b>Class Documentation</b>	<b>7</b>
4.1	Address Struct Reference . . . . .	7
4.1.1	Detailed Description . . . . .	7
4.2	CompleteTree< E > Class Template Reference . . . . .	7
4.2.1	Detailed Description . . . . .	9
4.2.2	Constructor & Destructor Documentation . . . . .	9
4.2.2.1	CompleteTree() . . . . .	9
4.2.3	Member Function Documentation . . . . .	9
4.2.3.1	addLast(const E &e) . . . . .	9
4.2.3.2	empty() const . . . . .	9
4.2.3.3	hasLeft(const Position &p) const . . . . .	9
4.2.3.4	hasRight(const Position &p) const . . . . .	10
4.2.3.5	idx(const Position &p) const . . . . .	10
4.2.3.6	isRoot(const Position &p) const . . . . .	10
4.2.3.7	last() . . . . .	11
4.2.3.8	left(const Position &p) . . . . .	11

4.2.3.9	parent(const Position &p)	11
4.2.3.10	pos(int i)	11
4.2.3.11	right(const Position &p)	12
4.2.3.12	root()	12
4.2.3.13	size() const	12
4.2.3.14	swap(const Position &p, const Position &q)	12
4.3	DBManager Class Reference	12
4.3.1	Constructor & Destructor Documentation	13
4.3.1.1	DBManager()	13
4.3.2	Member Function Documentation	13
4.3.2.1	addEdges(Stadium *origin, Stadium *destination, int weight)	13
4.3.2.2	AddNewSouvenir(int stadiumKey, QString name, double price, int quantity)	14
4.3.2.3	AddNewStadium(Stadium *s)	14
4.3.2.4	createGraph(skiplist< int, Stadium * > stadiumList)	15
4.3.2.5	getAllStadiumsKeys()	15
4.3.2.6	getStadiumID(QString stadiumName)	15
4.3.2.7	getStadiums()	15
4.3.2.8	RemoveSouvenir(int stadiumKey, QString name)	16
4.3.2.9	updateSouvenirName(int stadiumKey, QString oldName, QString newName)	16
4.3.2.10	updateSouvenirPrice(int stadiumKey, QString souvenirName, double newPrice)	16
4.3.2.11	updateSouvenirQuantity(int stadiumKey, QString souvenirName, int newQuantity)	17
4.3.2.12	UpdateStadium(Stadium *s)	17
4.3.2.13	updateTotalRevenue(int stadiumKey, double newRevenue)	17
4.4	Graph< E >::Edge Class Reference	18
4.4.1	Detailed Description	18
4.4.2	Member Function Documentation	18
4.4.2.1	isAdjacentTo(Edge f)	18
4.4.2.2	isIncidentOn(Vertex v)	19
4.4.2.3	opposite(Vertex v)	19
4.4.2.4	print()	19

4.5	Entry< K, V > Class Template Reference	20
4.5.1	Detailed Description	21
4.6	Graph< E > Class Template Reference	21
4.6.1	Detailed Description	22
4.6.2	Constructor & Destructor Documentation	23
4.6.2.1	Graph()	23
4.6.3	Member Function Documentation	23
4.6.3.1	dft(const E &e)	23
4.6.3.2	dftHelper(Vertex &location, VertexList &outList)	23
4.6.3.3	Dijkstra(const E &e)	23
4.6.3.4	eraseEdge(const E &v, const E &w, const int &x)	23
4.6.3.5	eraseVertex(const E &e)	24
4.6.3.6	findEdge(const E &v, const E &w, const int &x)	24
4.6.3.7	findVertex(const E &e)	24
4.6.3.8	insertEdge(const E &v, const E &w, const int &x)	24
4.6.3.9	insertVertex(const E &e)	25
4.6.3.10	MSTPrim()	25
4.6.3.11	PrimJarnek()	25
4.6.3.12	print(std::ofstream &output, std::string title=""Graph Output")	25
4.6.3.13	shortestPathTo(const E &end)	25
4.7	HeapPriorityQueue< E, C > Class Template Reference	26
4.7.1	Detailed Description	26
4.7.2	Member Function Documentation	27
4.7.2.1	empty() const	27
4.7.2.2	pop()	27
4.7.2.3	push(const E &e)	27
4.7.2.4	size() const	27
4.7.2.5	top()	27
4.8	skiplist< K, V >::Iterator Class Reference	28
4.8.1	Detailed Description	28

4.8.2	Constructor & Destructor Documentation . . . . .	28
4.8.2.1	Iterator(node *position) . . . . .	28
4.8.3	Member Function Documentation . . . . .	28
4.8.3.1	operator!=(const Iterator &p) const . . . . .	29
4.8.3.2	operator*() const . . . . .	29
4.8.3.3	operator*() . . . . .	29
4.8.3.4	operator++(int) . . . . .	29
4.8.3.5	operator++() . . . . .	30
4.8.3.6	operator--() . . . . .	30
4.8.3.7	operator==(const Iterator &p) const . . . . .	30
4.9	Ui::MainWindow Class Reference . . . . .	30
4.10	MainWindow Class Reference . . . . .	31
4.10.1	Constructor & Destructor Documentation . . . . .	31
4.10.1.1	MainWindow(QWidget *parent=0) . . . . .	31
4.10.2	Member Function Documentation . . . . .	32
4.10.2.1	addToCart(Souvenir *s) . . . . .	32
4.10.2.2	isBlank(QString text) . . . . .	32
4.10.2.3	tripProcess(QVector< Stadium * > trip) . . . . .	32
4.10.2.4	tripProcess2(QVector< VertexItr > trip) . . . . .	33
4.11	skiplist< K, V >::node Class Reference . . . . .	33
4.11.1	Detailed Description . . . . .	34
4.11.2	Constructor & Destructor Documentation . . . . .	34
4.11.2.1	node() . . . . .	34
4.11.2.2	node(node *up, node *down, node *left, node *right, nodeType Type=REGULAR) . . . . .	34
4.11.3	Member Function Documentation . . . . .	35
4.11.3.1	add(const item &e) . . . . .	35
4.11.3.2	add(const K &k, const V &v) . . . . .	35
4.11.3.3	clear() . . . . .	35
4.11.3.4	down() const . . . . .	35
4.11.3.5	empty() const . . . . .	36

4.11.3.6	<code>key() const</code>	36
4.11.3.7	<code>left() const</code>	36
4.11.3.8	<code>operator!=(const node &amp;that) const</code>	36
4.11.3.9	<code>operator&lt;(const node &amp;that) const</code>	36
4.11.3.10	<code>operator&lt;=(const node &amp;that) const</code>	37
4.11.3.11	<code>operator==(const node &amp;that) const</code>	37
4.11.3.12	<code>operator&gt;(const node &amp;that) const</code>	37
4.11.3.13	<code>operator&gt;=(const node &amp;that) const</code>	37
4.11.3.14	<code>right() const</code>	37
4.11.3.15	<code>setDown(node *down)</code>	38
4.11.3.16	<code>setLeft(node *left)</code>	38
4.11.3.17	<code>setNodeType(const nodeType &amp;n)</code>	38
4.11.3.18	<code>setRight(node *right)</code>	38
4.11.3.19	<code>setUp(node *up)</code>	39
4.11.3.20	<code>type() const</code>	39
4.11.3.21	<code>up() const</code>	39
4.11.3.22	<code>value()</code>	39
4.12	<code>PriorityQueue&lt; T &gt;</code> Class Template Reference	40
4.13	<code>qt_meta_stringdata_MainWindow_t</code> Struct Reference	40
4.14	<code>qt_meta_stringdata_ShoppingCart_t</code> Struct Reference	40
4.15	<code>ShoppingCart</code> Class Reference	40
4.15.1	Constructor & Destructor Documentation	41
4.15.1.1	<code>ShoppingCart(QWidget *parent=0)</code>	41
4.15.2	Member Function Documentation	41
4.15.2.1	<code>setList(QVector&lt; Souvenir * &gt; shoppingCart, skiplist&lt; int, Stadium * &gt; stadiums)</code>	41
4.16	<code>Ui::ShoppingCart</code> Class Reference	41
4.17	<code>skiplist&lt; K, V &gt;</code> Class Template Reference	42
4.17.1	Detailed Description	43
4.17.2	Constructor & Destructor Documentation	43
4.17.2.1	<code>skiplist()</code>	43

4.17.3	Member Function Documentation	43
4.17.3.1	addBlankLevel()	43
4.17.3.2	begin()	44
4.17.3.3	column(node *n) const	44
4.17.3.4	end()	44
4.17.3.5	erase(const K &k)	44
4.17.3.6	flipCoin()	45
4.17.3.7	get(const K &k)	45
4.17.3.8	height() const	45
4.17.3.9	insert(const K &k, const V &v)	45
4.17.3.10	insert(const item &e)	45
4.17.3.11	print() const	46
4.17.3.12	printVert() const	46
4.17.3.13	search(const K &k) const	46
4.17.3.14	size() const	47
4.18	Souvenir Class Reference	47
4.18.1	Detailed Description	48
4.18.2	Constructor & Destructor Documentation	48
4.18.2.1	Souvenir(unsigned int id, QString name, double price, unsigned int qty)	48
4.18.3	Member Function Documentation	49
4.18.3.1	addToQuantity(unsigned int addQty)	49
4.18.3.2	getName() const	49
4.18.3.3	getPrice() const	49
4.18.3.4	getQuantity() const	49
4.18.3.5	getStadiumID() const	50
4.18.3.6	setName(QString newName)	50
4.18.3.7	setPrice(double newPrice)	51
4.18.3.8	setQuantity(unsigned int newQty)	51
4.18.3.9	setStadiumID(int id)	51
4.19	Stadium Class Reference	51



4.19.1 Detailed Description . . . . .	53
4.19.2 Constructor & Destructor Documentation . . . . .	53
4.19.2.1 Stadium() . . . . .	53
4.19.3 Member Function Documentation . . . . .	54
4.19.3.1 addSouvenir(Souvenir *newSouvenir) . . . . .	54
4.19.3.2 findSouvenir(QString name) . . . . .	54
4.19.3.3 getAddress() const . . . . .	54
4.19.3.4 getBoxOfficeNumber() const . . . . .	55
4.19.3.5 getDateOpened() const . . . . .	55
4.19.3.6 getLeagueType() const . . . . .	55
4.19.3.7 getSeatingCapacity() const . . . . .	55
4.19.3.8 getSouvenirs() const . . . . .	55
4.19.3.9 getStadiumID() const . . . . .	55
4.19.3.10 getStadiumName() const . . . . .	56
4.19.3.11 getSurface() const . . . . .	56
4.19.3.12 getTeamName() const . . . . .	56
4.19.3.13 getTotalRevenue() const . . . . .	56
4.19.3.14 getTypology() const . . . . .	56
4.19.3.15 removeSouvenir(QString name) . . . . .	56
4.19.3.16 setAddress(QString streetAddress, QString city, QString state, QString zipCode) . . . . .	57
4.19.3.17 setAddress(Address newAddress) . . . . .	57
4.19.3.18 setBoxOfficeNumber(QString newNumber) . . . . .	57
4.19.3.19 setDateOpened(QString newDate) . . . . .	57
4.19.3.20 setLeagueType(QString newLeagueType) . . . . .	57
4.19.3.21 setSeatingCapacity(unsigned int newCapacity) . . . . .	58
4.19.3.22 setStadiumName(QString newName) . . . . .	58
4.19.3.23 setSurface(QString newSurface) . . . . .	58
4.19.3.24 setTeamName(QString newTeam) . . . . .	58
4.19.3.25 setTotalRevenue(double revenue) . . . . .	58
4.19.3.26 setTypology(QString typo) . . . . .	59
4.20 Ui_MainWindow Class Reference . . . . .	59
4.21 Ui_ShoppingCart Class Reference . . . . .	63
4.22 Graph< E >::Vertex Class Reference . . . . .	64
4.22.1 Detailed Description . . . . .	64
4.22.2 Member Function Documentation . . . . .	65
4.22.2.1 adjacentVertex() . . . . .	65
4.22.2.2 distanceTo(const VertexItr &v) . . . . .	65
4.22.2.3 isAdjacentTo(const E &v) . . . . .	65
4.22.2.4 print() . . . . .	65
4.22.2.5 removeEdge(EdgeItr edge) . . . . .	65

<b>5 File Documentation</b>	<b>67</b>
5.1 <a href="#">src/header/CompleteTree.h File Reference</a>	67
5.1.1 <a href="#">Detailed Description</a>	67
5.2 <a href="#">src/header/graph.h File Reference</a>	67
5.2.1 <a href="#">Detailed Description</a>	68
5.3 <a href="#">src/header/HeapPriorityQueue.h File Reference</a>	68
5.3.1 <a href="#">Detailed Description</a>	68
<b>Index</b>	<b>69</b>

# Chapter 1

## Hierarchical Index

### 1.1 Class Hierarchy

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

Address . . . . .	7
CompleteTree< E > . . . . .	7
DBManager . . . . .	12
Graph< E >::Edge . . . . .	18
Entry< K, V > . . . . .	20
Graph< E > . . . . .	21
Graph< Stadium > . . . . .	21
HeapPriorityQueue< E, C > . . . . .	26
skiplist< K, V >::Iterator . . . . .	28
skiplist< K, V >::node . . . . .	33
PriorityQueue< T > . . . . .	40
QMainWindow	
MainWindow . . . . .	31
qt_meta_stringdata_MainWindow_t . . . . .	40
qt_meta_stringdata_ShoppingCart_t . . . . .	40
QWidget	
ShoppingCart . . . . .	40
ShoppingCart . . . . .	40
skiplist< K, V > . . . . .	42
skiplist< int, Stadium * > . . . . .	42
Souvenir . . . . .	47
Stadium . . . . .	51
Ui_MainWindow . . . . .	59
Ui::MainWindow . . . . .	30
Ui_ShoppingCart . . . . .	63
Ui::ShoppingCart . . . . .	41
Graph< E >::Vertex . . . . .	64



## Chapter 2

# Class Index

### 2.1 Class List

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

<a href="#">Address</a>	The <a href="#">Address</a> 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 . . . . .	7
<a href="#">CompleteTree&lt; E &gt;</a>	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 . . . . .	7
<a href="#">DBManager</a>		12
<a href="#">Graph&lt; E &gt;::Edge</a>	The <a href="#">Edge</a> class <a href="#">Edge</a> within the graph holds the weight between two incident vertecies and methods to manipulate that data and access the adjacent vertrices . . . . .	18
<a href="#">Entry&lt; K, V &gt;</a>	This class describes a key-value pair . . . . .	20
<a href="#">Graph&lt; E &gt;</a>	Undirected <a href="#">Graph</a> 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 . . . . .	21
<a href="#">HeapPriorityQueue&lt; E, C &gt;</a>	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	26
<a href="#">skiplist&lt; K, V &gt;::Iterator</a>	Skip List <a href="#">Iterator</a> This class describes a skip list iterator for moving through the skip list . . . .	28
<a href="#">Ui::MainWindow</a>		30
<a href="#">MainWindow</a>		31
<a href="#">skiplist&lt; K, V &gt;::node</a>	A skip list node . . . . .	33
<a href="#">PriorityQueue&lt; T &gt;</a>		40
<a href="#">qt_meta_stringdata_MainWindow_t</a>		40
<a href="#">qt_meta_stringdata_ShoppingCart_t</a>		40
<a href="#">ShoppingCart</a>		40
<a href="#">Ui::ShoppingCart</a>		41
<a href="#">skiplist&lt; K, V &gt;</a>	A skip list implementation of a map, keys must be unique . . . . .	42

[Souvenir](#)

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 . . . . . 47

[Stadium](#)

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 . . . . . 51

[Ui\\_MainWindow](#) . . . . . 59

[Ui\\_ShoppingCart](#) . . . . . 63

[Graph< E >::Vertex](#)

The [Vertex](#) class [Vertex](#) inside the graph holds the data methods for workign with vertices . . 64

## Chapter 3

# File Index

### 3.1 File List

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

src/ui/mainwindow.h . . . . .	??
src/ui/shoppingcart.h . . . . .	??
src/form/shoppingcart.h . . . . .	??
src/header/CompleteTree.h . . . . .	
Assignment #7 - Heap Sort . . . . .	67
src/header/dbmanager.h . . . . .	??
src/header/entry.h . . . . .	??
src/header/graph.h . . . . .	
Undirected Graph . . . . .	67
src/header/HeapPriorityQueue.h . . . . .	
Assignment #7 - Heap Sort . . . . .	68
src/header/mainwindow.h . . . . .	??
src/header/priorityqueue.h . . . . .	??
src/header/quicksort.h . . . . .	??
src/header/shoppingcart.h . . . . .	??
src/header/skiplist.h . . . . .	??
src/header/souvenir.h . . . . .	??
src/header/stadium.h . . . . .	??





## 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

<i>e</i>	
----------	--

#### 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

<i>p</i>	
----------	--

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

<i>p</i>	
<i>q</i>	

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

<i>origin</i>	
<i>destination</i>	
<i>weight</i>	

## 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

<i>stadiumKey</i>	
<i>name</i>	
<i>price</i>	
<i>quantity</i>	

## Returns

## Parameters

<i>stadiumKey</i>	key links to an stadium key/id.
<i>name</i>	name of the souvenir item.
<i>price</i>	price of the souvenir item.
<i>quantity</i>	quantity 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

<i>s</i>	
----------	--



Returns

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

createGraph

Parameters

<i>stadiumList</i>	
--------------------	--

Returns

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

getAllStadiumsKeys

Returns

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

getStadiumID

Parameters

<i>stadiumName</i>	
--------------------	--

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

<i>stadiumKey</i>	
<i>name</i>	

##### Returns

##### Parameters

<i>stadiumKey</i>	key links to an stadium key/id.
<i>name</i>	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

<i>stadiumKey</i>	
<i>oldName</i>	
<i>newName</i>	

##### Returns

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

updateSouvenirPrice

##### Parameters

<i>stadiumKey</i>	
<i>souvenirName</i>	
<i>newPrice</i>	

Returns

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

updateSouvenirQuantity

Parameters

<i>stadiumKey</i>	
<i>souvenirName</i>	
<i>newQuantity</i>	

Returns

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

UpdateStadium.

Parameters

<i>s</i>	
----------	--

Returns

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

updateTotalRevenue

Parameters

<i>stadiumKey</i>	
<i>newRevenue</i>	

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

<i>f</i>	
----------	--

## Returns

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

Test whether this edge is incident on *v*.

## Parameters

<i>v</i>	[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

<i>v</i>	
----------	--

## 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 constructor 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](#) inside 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 **Vertexltr**
- typedef EdgeList::iterator **Edgeltr**
- typedef std::list< Edgeltr > **EdgeltrList**
- typedef EdgeltrList::iterator **Edgeltrltr**
- typedef [HeapPriorityQueue](#)< [Edge](#), std::less< [Edge](#) > > **EdgePQueue**
- typedef std::vector< Vertexltr > **VertexltrVector**

## 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*
- Vertexltr [findVertex](#) (const E &e)  
*Finds the vertex with data 'e'.*
- Edgeltr [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 &amp; 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

<i>e</i>	[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

<i>location</i>	[IN] the location to begin at
<i>outList</i>	[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

<i>e</i>	[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

<i>v</i>	[IN] vertex data at 'start' of edge to delete
<i>w</i>	[IN] vertex data at 'end' of edge to delete
<i>x</i>	[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

<i>e</i>	[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

<i>v</i>	[IN] vertex data at 'start' of edge
<i>w</i>	[IN] vertex data at 'end' of edge
<i>x</i>	[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

<i>e</i>	[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

<i>v</i>	[IN] vertex data at 'start' of edge
<i>w</i>	[IN] vertex data at 'end' of edge
<i>x</i>	[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

<i>e</i>	[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 Spanning 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

<i>output</i>	[IN] reference to an output file stream
<i>title</i>	[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

<i>end</i>	[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

<i>e</i>	This method adds the element <i>e</i> to the heap. It then performs a bubble up to restore heap order.
<i>e</i>	[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

<i>position</i>	
-----------------	--

### 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

<i>p</i>	
----------	--

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

<i>p</i>	
----------	--

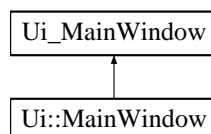
Returns

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:





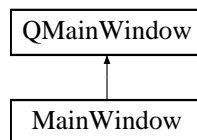
## Additional Inherited Members

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 initializes 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 requested 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 visited).
- 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 initializes the search bar.

#### Parameters

<i>parent</i>	
---------------	--

### 4.10.2 Member Function Documentation

#### 4.10.2.1 void MainWindow::addToCart ( Souvenir \* s )

[MainWindow::addToCart](#) A helper method to ensure the requested souvenir to be potentially bought is added to the shopping cart.

Adds a desired souvenir to the global shopping cart

##### Parameters

<i>s</i>	
----------	--

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

<i>text</i>	
-------------	--

##### 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 visited).

Trip process for only quick trip request

##### Parameters

<i>trip</i>	
-------------	--

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

Waits until user clicks 'next' button

## 4.10.2.4 void MainWindow::tripProcess2 ( QVector&lt; Vertexltr &gt; 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

<i>trip</i>	
-------------	--

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&lt; K, V &gt;::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

<i>up</i>	
<i>down</i>	
<i>left</i>	
<i>right</i>	
<i>Type</i>	
<i>up</i>	
<i>down</i>	
<i>left</i>	
<i>right</i>	
<i>type</i>	

## 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

<i>e</i>	
----------	--

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

<i>k</i>	
<i>v</i>	

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

<i>that</i>	
-------------	--

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

overloads the < operator

## Parameters

<i>that</i>	
-------------	--

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

overloads the <= operator

## Parameters

<i>that</i>	
-------------	--

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

overloads the == operator

## Parameters

<i>that</i>	
-------------	--

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

overloads the > operator

## Parameters

<i>that</i>	
-------------	--

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

overloads the >= operator

## Parameters

<i>that</i>	
-------------	--

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

<i>down</i>	
-------------	--

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

<i>left</i>	
-------------	--

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

<i>n</i>	
----------	--

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

<i>right</i>	
--------------	--



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

<i>up</i>	
-----------	--

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

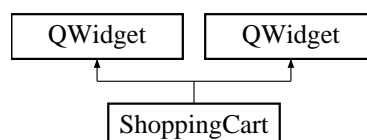
- QByteArrayData **data** [1]
- char **stringdata0** [13]

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

<i>parent</i>	
---------------	--

### 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

<i>shoppingCart</i>	
<i>stadiums</i>	

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

<i>n</i>	
----------	--

Returns

the column of the node

Parameters

<i>n</i>	[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

<i>k</i>	
<i>k</i>	[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

<i>k</i>	
----------	--

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

<i>k</i>	
<i>v</i>	

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**

<i>e</i>	
<i>e</i>	[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**

<i>k</i>	
----------	--

**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**

<i>k</i>	[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

<i>id</i>	[IN] id of the stadium this <a href="#">Souvenir</a> belongs to
<i>name</i>	[IN] name of the <a href="#">Souvenir</a>
<i>price</i>	[IN] price of the <a href="#">Souvenir</a>
<i>qty</i>	[IN] quantity of the <a href="#">Souvenir</a>

### 4.18.3 Member Function Documentation

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

[Souvenir::addToQuantity](#).

## Parameters

<i>addQty</i>	[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

<i>newName</i>	[IN] the new name
----------------	-------------------

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

[Souvenir::setPrice.](#)

## Parameters

<i>newPrice</i>	[IN] the new price
-----------------	--------------------

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

[Souvenir::setQuantity.](#)

## Parameters

<i>newQty</i>	[IN] the new quantity
---------------	-----------------------

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

[Souvenir::setStadiumID.](#)

## Parameters

<i>id</i>	[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::Stadium* Constructor.
- **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.

## CONSTRUCTOR & DESTRUCTOR

## Parameters

<i>id</i>	
<i>name</i>	
<i>team</i>	
<i>address</i>	
<i>number</i>	
<i>capacity</i>	
<i>surf</i>	
<i>league</i>	

### 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

<i>name</i>	
<i>price</i>	
<i>quantity</i>	

#### 4.19.3.2 Souvenir \* Stadium::findSouvenir ( QString name )

[Stadium::findSouvenir](#).

## Parameters

<i>name</i>	[IN] the name of the <a href="#">Souvenir</a> 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

<i>name</i>	
-------------	--

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

[Stadium::setAddress](#) Changes the address to new address.

## Parameters

<i>address</i>	
<i>city</i>	
<i>state</i>	
<i>zipCode</i>	

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

[Stadium::setAddress](#) Changes the address to new address.

## Parameters

<i>newAddress</i>	
-------------------	--

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

[Stadium::setBoxOfficeNumber](#) Changes the box office number to newNumber.

## Parameters

<i>newNumber</i>	
------------------	--

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

[Stadium::setDateOpened](#) Changes the date opened to newDate.

## Parameters

<i>newDate</i>	
----------------	--

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

[Stadium::setLeagueType](#) Changes league type to newLeagueType.

## Parameters

<i>newLeagueType</i>	
----------------------	--

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

[Stadium::setSeatingCapacity](#) Changes seating capacity to newCapacity.

## Parameters

<i>newCapacity</i>	
--------------------	--

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

[Stadium::setStadiumName](#) Changes the stadium name to newName.

## MUTATORS

## Parameters

<i>newName</i>	
----------------	--

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

[Stadium::setSurface](#) Changes surface to newSurface.

## Parameters

<i>newSurface</i>	
-------------------	--

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

setTeamName Changes the team name to newTeam

## Parameters

<i>newTeam</i>	
----------------	--

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

[Stadium::setTotalRevenue](#) Changes totalRevenue to revenue.

## Parameters

<i>revenue</i>	
----------------	--

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

[Stadium::setTypology](#) Changes typology to *typo*.

## Parameters

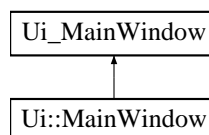
<i>typo</i>	
-------------	--

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 \* **confirmPurchasesPage**
- 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](#) inside 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** ()
- VertexltrVector **adjacentVertex** ()  
*Get all the neighbors of this vertex.*
- void **setDistance** (const int &newValue)
- int **getDistance** ()
- void **setParent** (const Vertexltr &parent)
- Vertexltr **getParent** ()
- void **resetDijkstra** ()
- EdgeltrList **incidentEdges** ()
- void **addEdge** (Edgeltr newEdge)
- void **removeEdge** (Edgeltr edge)  
*Removed the edge pointed to by the given iterator from this vertex's adjacency list.*
- bool **visited** ()
- bool **isAdjacentTo** (const E &v)  
*tests if this vertex is adjacent to vertex 'v'*
- int **distanceTo** (const Vertexltr &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](#) inside 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 >::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 Vertex& v )`

get the distance from this vertex to 'v'

##### Parameters

<code>v</code>	[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

<code>v</code>	[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 ( Edge& edge )`

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

**Parameters**

<i>edge</i>	[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](#) inside 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

- add
  - skiplist::node, 35
- addBlankLevel
  - skiplist, 43
- addEdges
  - DBManager, 13
- addLast
  - CompleteTree, 9
- AddNewSouvenir
  - DBManager, 14
- AddNewStadium
  - DBManager, 14
- addSouvenir
  - Stadium, 54
- addToCart
  - MainWindow, 32
- addToQuantity
  - Souvenir, 49
- Address, 7
- adjacentVertex
  - Graph::Vertex, 65
- begin
  - skiplist, 43
- clear
  - skiplist::node, 35
- column
  - skiplist, 44
- CompleteTree
  - addLast, 9
  - CompleteTree, 9
  - empty, 9
  - hasLeft, 9
  - hasRight, 10
  - idx, 10
  - isRoot, 10
  - last, 10
  - left, 11
  - parent, 11
  - pos, 11
  - right, 11
  - root, 12
  - size, 12
  - swap, 12
- CompleteTree< E >, 7
- createGraph
  - DBManager, 15
- DBManager, 12
- addEdges, 13
- AddNewSouvenir, 14
- AddNewStadium, 14
- createGraph, 15
- DBManager, 13
- getAllStadiumsKeys, 15
- getStadiumID, 15
- getStadiums, 15
- RemoveSouvenir, 15
- updateSouvenirName, 16
- updateSouvenirPrice, 16
- updateSouvenirQuantity, 17
- UpdateStadium, 17
- updateTotalRevenue, 17
- dft
  - Graph, 23
- dftHelper
  - Graph, 23
- Dijkstra
  - Graph, 23
- distanceTo
  - Graph::Vertex, 65
- down
  - skiplist::node, 35
- empty
  - CompleteTree, 9
  - HeapPriorityQueue, 27
  - skiplist::node, 36
- end
  - skiplist, 44
- Entry< K, V >, 20
- erase
  - skiplist, 44
- eraseEdge
  - Graph, 23
- eraseVertex
  - Graph, 24
- findEdge
  - Graph, 24
- findSouvenir
  - Stadium, 54
- findVertex
  - Graph, 24
- flipCoin
  - skiplist, 44
- get
  - skiplist, 45

- getAddress
  - Stadium, [54](#)
- getAllStadiumsKeys
  - DBManager, [15](#)
- getBoxOfficeNumber
  - Stadium, [54](#)
- getDateOpened
  - Stadium, [55](#)
- getLeagueType
  - Stadium, [55](#)
- getName
  - Souvenir, [49](#)
- getPrice
  - Souvenir, [49](#)
- getQuantity
  - Souvenir, [49](#)
- getSeatingCapacity
  - Stadium, [55](#)
- getSouvenirs
  - Stadium, [55](#)
- getStadiumID
  - DBManager, [15](#)
  - Souvenir, [49](#)
  - Stadium, [55](#)
- getStadiumName
  - Stadium, [55](#)
- getStadiums
  - DBManager, [15](#)
- getSurface
  - Stadium, [56](#)
- getTeamName
  - Stadium, [56](#)
- getTotalRevenue
  - Stadium, [56](#)
- getTypology
  - Stadium, [56](#)
- Graph
  - dft, [23](#)
  - dftHelper, [23](#)
  - Dijkstra, [23](#)
  - eraseEdge, [23](#)
  - eraseVertex, [24](#)
  - findEdge, [24](#)
  - findVertex, [24](#)
  - Graph, [23](#)
  - insertEdge, [24](#)
  - insertVertex, [25](#)
  - MSTPrim, [25](#)
  - PrimJarnek, [25](#)
  - print, [25](#)
  - shortestPathTo, [25](#)
- Graph< E >, [21](#)
- Graph< E >::Edge, [18](#)
- Graph< E >::Vertex, [64](#)
- Graph::Edge
  - isAdjacentTo, [18](#)
  - isIncidentOn, [19](#)
  - opposite, [19](#)
  - print, [19](#)
- Graph::Vertex
  - adjacentVertex, [65](#)
  - distanceTo, [65](#)
  - isAdjacentTo, [65](#)
  - print, [65](#)
  - removeEdge, [65](#)
- hasLeft
  - CompleteTree, [9](#)
- hasRight
  - CompleteTree, [10](#)
- HeapPriorityQueue
  - empty, [27](#)
  - pop, [27](#)
  - push, [27](#)
  - size, [27](#)
  - top, [27](#)
- HeapPriorityQueue< E, C >, [26](#)
- height
  - skiplist, [45](#)
- idx
  - CompleteTree, [10](#)
- insert
  - skiplist, [45](#)
- insertEdge
  - Graph, [24](#)
- insertVertex
  - Graph, [25](#)
- isAdjacentTo
  - Graph::Edge, [18](#)
  - Graph::Vertex, [65](#)
- isBlank
  - MainWindow, [32](#)
- isIncidentOn
  - Graph::Edge, [19](#)
- isRoot
  - CompleteTree, [10](#)
- Iterator
  - skiplist::Iterator, [28](#)
- key
  - skiplist::node, [36](#)
- last
  - CompleteTree, [10](#)
- left
  - CompleteTree, [11](#)
  - skiplist::node, [36](#)
- MSTPrim
  - Graph, [25](#)
- MainWindow, [31](#)
  - addToCart, [32](#)
  - isBlank, [32](#)
  - MainWindow, [31](#)
  - tripProcess, [32](#)
  - tripProcess2, [32](#)



- node
  - skiplist::node, [34](#)
- operator!=
  - skiplist::Iterator, [28](#)
  - skiplist::node, [36](#)
- operator<
  - skiplist::node, [36](#)
- operator<=
  - skiplist::node, [37](#)
- operator>
  - skiplist::node, [37](#)
- operator>=
  - skiplist::node, [37](#)
- operator\*
  - skiplist::Iterator, [29](#)
- operator++
  - skiplist::Iterator, [29](#)
- operator--
  - skiplist::Iterator, [30](#)
- operator==
  - skiplist::Iterator, [30](#)
  - skiplist::node, [37](#)
- opposite
  - Graph::Edge, [19](#)
- parent
  - CompleteTree, [11](#)
- pop
  - HeapPriorityQueue, [27](#)
- pos
  - CompleteTree, [11](#)
- PrimJarnek
  - Graph, [25](#)
- print
  - Graph, [25](#)
  - Graph::Edge, [19](#)
  - Graph::Vertex, [65](#)
  - skiplist, [46](#)
- printVert
  - skiplist, [46](#)
- PriorityQueue< T >, [40](#)
- push
  - HeapPriorityQueue, [27](#)
- qt\_meta\_stringdata\_MainWindow\_t, [40](#)
- qt\_meta\_stringdata\_ShoppingCart\_t, [40](#)
- removeEdge
  - Graph::Vertex, [65](#)
- RemoveSouvenir
  - DBManager, [15](#)
- removeSouvenir
  - Stadium, [56](#)
- right
  - CompleteTree, [11](#)
  - skiplist::node, [37](#)
- root
  - CompleteTree, [12](#)
- search
  - skiplist, [46](#)
- setAddress
  - Stadium, [57](#)
- setBoxOfficeNumber
  - Stadium, [57](#)
- setDateOpened
  - Stadium, [57](#)
- setDown
  - skiplist::node, [38](#)
- setLeagueType
  - Stadium, [57](#)
- setLeft
  - skiplist::node, [38](#)
- setList
  - ShoppingCart, [41](#)
- setName
  - Souvenir, [50](#)
- setNodeType
  - skiplist::node, [38](#)
- setPrice
  - Souvenir, [51](#)
- setQuantity
  - Souvenir, [51](#)
- setRight
  - skiplist::node, [38](#)
- setSeatingCapacity
  - Stadium, [58](#)
- setStadiumID
  - Souvenir, [51](#)
- setStadiumName
  - Stadium, [58](#)
- setSurface
  - Stadium, [58](#)
- setTeamName
  - Stadium, [58](#)
- setTotalRevenue
  - Stadium, [58](#)
- setTypology
  - Stadium, [59](#)
- setUp
  - skiplist::node, [38](#)
- ShoppingCart, [40](#)
  - setList, [41](#)
  - ShoppingCart, [41](#)
- shortestPathTo
  - Graph, [25](#)
- size
  - CompleteTree, [12](#)
  - HeapPriorityQueue, [27](#)
  - skiplist, [46](#)
- skiplist
  - addBlankLevel, [43](#)
  - begin, [43](#)
  - column, [44](#)
  - end, [44](#)
  - erase, [44](#)
  - flipCoin, [44](#)

- get, 45
- height, 45
- insert, 45
- print, 46
- printVert, 46
- search, 46
- size, 46
- skiplist, 43
- skiplist< K, V >, 42
- skiplist< K, V >::iterator, 28
- skiplist< K, V >::node, 33
- skiplist::iterator
  - iterator, 28
  - operator!=, 28
  - operator\*, 29
  - operator++, 29
  - operator--, 30
  - operator==, 30
- skiplist::node
  - add, 35
  - clear, 35
  - down, 35
  - empty, 36
  - key, 36
  - left, 36
  - node, 34
  - operator!=, 36
  - operator<, 36
  - operator<=, 37
  - operator>, 37
  - operator>=, 37
  - operator==, 37
  - right, 37
  - setDown, 38
  - setLeft, 38
  - setNodeType, 38
  - setRight, 38
  - setUp, 38
  - type, 39
  - up, 39
  - value, 39
- Souvenir, 47
  - addToQuantity, 49
  - getName, 49
  - getPrice, 49
  - getQuantity, 49
  - getStadiumID, 49
  - setName, 50
  - setPrice, 51
  - setQuantity, 51
  - setStadiumID, 51
  - Souvenir, 48
- src/header/CompleteTree.h, 67
- src/header/HeapPriorityQueue.h, 68
- src/header/graph.h, 67
- Stadium, 51
  - addSouvenir, 54
  - findSouvenir, 54
  - getAddress, 54
  - getBoxOfficeNumber, 54
  - getDateOpened, 55
  - getLeagueType, 55
  - getSeatingCapacity, 55
  - getSouvenirs, 55
  - getStadiumID, 55
  - getStadiumName, 55
  - getSurface, 56
  - getTeamName, 56
  - getTotalRevenue, 56
  - getTypology, 56
  - removeSouvenir, 56
  - setAddress, 57
  - setBoxOfficeNumber, 57
  - setDateOpened, 57
  - setLeagueType, 57
  - setSeatingCapacity, 58
  - setStadiumName, 58
  - setSurface, 58
  - setTeamName, 58
  - setTotalRevenue, 58
  - setTypology, 59
  - Stadium, 53
- swap
  - CompleteTree, 12
- top
  - HeapPriorityQueue, 27
- tripProcess
  - MainWindow, 32
- tripProcess2
  - MainWindow, 32
- type
  - skiplist::node, 39
- Ui::MainWindow, 30
- Ui::ShoppingCart, 41
- Ui\_MainWindow, 59
- Ui\_ShoppingCart, 63
- up
  - skiplist::node, 39
- updateSouvenirName
  - DBManager, 16
- updateSouvenirPrice
  - DBManager, 16
- updateSouvenirQuantity
  - DBManager, 17
- UpdateStadium
  - DBManager, 17
- updateTotalRevenue
  - DBManager, 17
- value
  - skiplist::node, 39