CIS Movie Database

Generated by Doxygen 1.9.2

1 Class Index 1

1.1 Class List . . . 1

2 File Index 3

2.1 File List . . . 3

3 Class Documentation 5

3.1 App Class Reference . . . 5

3.1.1 Detailed Description . . . 5

3.1.2 Member Function Documentation . . . 5

3.1.2.1 run() . . . 6

3.2 BinaryTree Class Reference . . . 6

3.2.1 Constructor & Destructor Documentation . . . 7

3.2.1.1 BinaryTree() . . . 7

3.2.1.2 ~BinaryTree() . . . 7

3.2.2 Member Function Documentation . . . 7

3.2.2.1 DFS() . . . 7

3.2.2.2 getMax() . . . 7

3.2.2.3 getMin() . . . 7

3.2.2.4 getParent() . . . 7

3.2.2.5 inOrder() . . . 8

3.2.2.6 insert() . . . 8

3.2.2.7 remove() . . . 8

3.3 FileLoader Class Reference . . . 8

3.3.1 Detailed Description . . . 9

3.3.2 Constructor & Destructor Documentation . . . 9

3.3.2.1 FileLoader() . . . 9

3.3.3 Member Function Documentation . . . 9

3.3.3.1 load() . . . 9

3.4 FileWriter Class Reference . . . 9

3.4.1 Detailed Description . . . 10

3.4.2 Constructor & Destructor Documentation . . . 10

3.4.2.1 FileWriter() . . . 10

3.4.3 Member Function Documentation . . . 10

3.4.3.1 save() . . . 11

3.5 HashTable< Key, Value > Class Template Reference . . . 11

3.5.1 Detailed Description . . . 12

3.5.2 Constructor & Destructor Documentation . . . 12

3.5.2.1 HashTable() [1/3] . . . 12

3.5.2.2 ~HashTable() . . . 13

3.5.2.3 HashTable() [2/3] . . . 13

3.5.2.4 HashTable() [3/3] . . . 13

3.5.3 Member Function Documentation . . . 13

3.5.3.1 add() . . . 13
3.5.3.2 find() . . . 14
3.5.3.3 getBucketsCount() . . . 14
3.5.3.4 getLoadFactor() . . . 14
3.5.3.5 getNumberOfCollisions() . . . 15
3.5.3.6 list() . . . 15
3.5.3.7 operator=() [1/2] . . . 15
3.5.3.8 operator=() [2/2] . . . 15
3.5.3.9 rehash() . . . 15
3.5.3.10 remove() [1/2] . . . 16
3.5.3.11 remove() [2/2] . . . 16
3.6 LinkedList< T >::Iterator Class Reference . . . 17
3.7 LinkedList< T > Class Template Reference . . . 18
3.7.1 Detailed Description . . . 18
3.7.2 Constructor & Destructor Documentation . . . 19
3.7.2.1 LinkedList() [1/2] . . . 19
3.7.2.2 LinkedList() [2/2] . . . 19
3.7.3 Member Function Documentation . . . 19
3.7.3.1 begin() . . . 19
3.7.3.2 empty() . . . 19
3.7.3.3 end() . . . 20
3.7.3.4 find() . . . 20
3.7.3.5 operator=() [1/2] . . . 20
3.7.3.6 operator=() [2/2] . . . 20
3.7.3.7 pushBack() . . . 20
3.7.3.8 remove() . . . 21
3.7.3.9 size() . . . 21
3.8 Movie Class Reference . . . 22
3.8.1 Constructor & Destructor Documentation . . . 23
3.8.1.1 Movie() . . . 23
3.8.2 Member Function Documentation . . . 23
3.8.2.1 getID() . . . 23
3.8.2.2 setID() . . . 23
3.9 MovieDB Class Reference . . . 24
3.9.1 Detailed Description . . . 24
3.9.2 Member Function Documentation . . . 24
3.9.2.1 addMovie() . . . 24
3.9.2.2 deleteMovieByID() . . . 25
3.9.2.3 findMovieByID() . . . 25
3.9.2.4 findMovieByTitle() . . . 25
3.9.2.5 getAllMovies() . . . 26
3.9.2.6 getDataStructureStats() . . . 26

3.9.2.7 listMovieSortedByTitle() . . . 26
3.9.2.8 reserveHashBuckets() . . . 26
3.10 MovieDBDSStats Struct Reference . . . 27
3.11 LinkedList< T >::Node Struct Reference . . . 28
3.12 Node Class Reference . . . 29

4 File Documentation **31**
4.1 App.h . . . 31
4.2 BSTNode.h . . . 31
4.3 FIleLoader.h . . . 32
4.4 FileWriter.h . . . 32
4.5 HashTable.h . . . 32
4.6 LinkedList.h . . . 35
4.7 Movie.h . . . 36
4.8 MovieBST.h . . . 37
4.9 MovieDB.h . . . 38
4.10 Util.h . . . 39

Index **41**

Chapter 1

Class Index

1.1 Class List

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

App 5
BinaryTree 6
FileLoader 8
FileWriter 9
HashTable< Key, Value > 11
LinkedList< T >::Iterator 17
LinkedList< T > 18
Movie 22
MovieDB 24
MovieDBDSStats 27
LinkedList< T >::Node 28
Node 29

Chapter 2

File Index

2.1 File List

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

src/App.h . . . 31
src/BSTNode.h . . . 31
src/FIleLoader.h . . . 32
src/FileWriter.h . . . 32
src/HashTable.h . . . 32
src/LinkedList.h . . . 35
src/Movie.h . . . 36
src/MovieBST.h . . . 37
src/MovieDB.h . . . 38
src/Util.h . . . 39

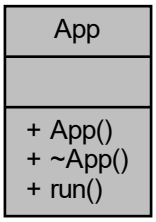
Chapter 3

Class Documentation

3.1 App Class Reference

`#include <App.h>`

Collaboration diagram for App:



Public Member Functions

- int run ()

3.1.1 Detailed Description

App class that integrate all the other classes to implement the program the main responsibility is dealing with user input and output

3.1.2 Member Function Documentation

3.1.2.1 run()

```
int App::run ( )
```

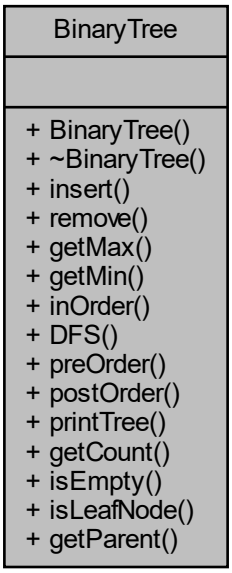
run the program

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

- src/App.h
- src/App.cpp

3.2 BinaryTree Class Reference

Collaboration diagram for BinaryTree:



Public Member Functions

- BinaryTree ()
- ∼BinaryTree ()
- bool insert (Movie dataIn)
- bool remove (Movie)
- Node ∗ getMax (Node ∗)
- Node ∗ getMin (Node ∗)
- std::vector< Movie > inOrder () const
- std::vector< Movie > DFS (std::string)
- void **preOrder** () const
- void **postOrder** () const
- void **printTree** (void visit(Movie &, int)) const
- int **getCount** () const
- bool **isEmpty** () const
- bool **isLeafNode** (Node ∗node)
- Node ∗ getParent (Node ∗, Node ∗)

3.2.1 Constructor & Destructor Documentation

3.2.1.1 BinaryTree()

```
BinaryTree::BinaryTree ( )
```

Constructor

3.2.1.2 ~BinaryTree()

```
BinaryTree::~BinaryTree ( )
```

Destructor This function calls a recursive function to delete all nodes in the binary tree

3.2.2 Member Function Documentation

3.2.2.1 DFS()

```
vector< Movie > BinaryTree::DFS (
            std::string title )
```

Function uses depth-first search to find a specific node via the secondary key

3.2.2.2 getMax()

```
Node * BinaryTree::getMax (
            Node * nodePtr )
```

Finds the right-most leaf in the tree

3.2.2.3 getMin()

```
Node * BinaryTree::getMin (
            Node * nodePtr )
```

Finds the left-most leaf in the tree

3.2.2.4 getParent()

```
Node * BinaryTree::getParent (
            Node * child,
            Node * rt )
```

Finds a leafs parent in the binary tree

3.2.2.5 inOrder()

```
std::vector< Movie > BinaryTree::inOrder ( ) const
```

This function calls a recursive function to traverse the tree in inorder (the wrapper function)

3.2.2.6 insert()

```
bool BinaryTree::insert (
            Movie dataIn )
```

Insert data into a random Binary Tree

3.2.2.7 remove()

```
bool BinaryTree::remove (
            Movie dataIn )
```

Removes data from the binary tree

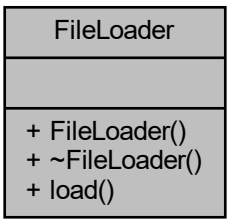
The documentation for this class was generated from the following files:

- src/MovieBST.h
- src/MovieBST.cpp

3.3 FileLoader Class Reference

```
#include <FIleLoader.h>
```

Collaboration diagram for FileLoader:



Public Member Functions

- FileLoader (std::ifstream &&stream)
- void load (MovieDB &db)

3.3.1 Detailed Description

File loader that reads txt file and insert movie to MovieDB

3.3.2 Constructor & Destructor Documentation

3.3.2.1 FileLoader()

```
FileLoader::FileLoader (
            std::ifstream && stream )
```

Constructor

Parameters

stream	ifstream object that has been opened with movie file

3.3.3 Member Function Documentation

3.3.3.1 load()

```
void FileLoader::load (
            MovieDB & db )
```

Read the file and insert movies to MovieDB

Parameters

db	MovieDB reference

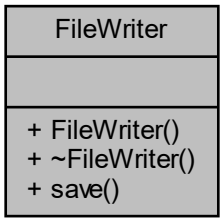
The documentation for this class was generated from the following files:

- src/FIleLoader.h
- src/FileLoader.cpp

3.4 FileWriter Class Reference

```
#include <FileWriter.h>
```

Collaboration diagram for FileWriter:



Public Member Functions

- FileWriter (std::ofstream &&file)
- void save (const MovieDB &db)

3.4.1 Detailed Description

File writer that writes txt file based on movies in MovieDB

3.4.2 Constructor & Destructor Documentation

3.4.2.1 FileWriter()

```
FileWriter::FileWriter (
            std::ofstream && file )
```

Constructor

Parameters

file	ofstream object that has been opened with output file

3.4.3 Member Function Documentation

3.4.3.1 save()

```
void FileWriter::save (
            const MovieDB & db )
```

Write the file based on specified MovieDB

Parameters

db	MovieDB reference

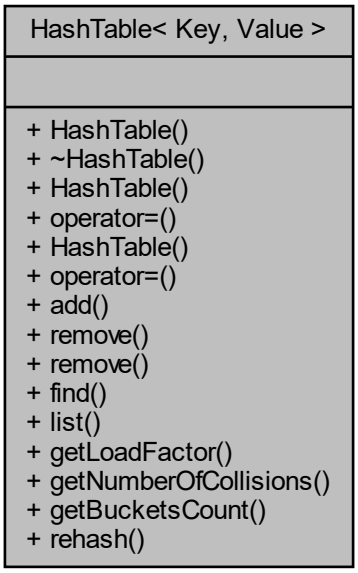
The documentation for this class was generated from the following files:

- src/FileWriter.h
- src/FileWriter.cpp

3.5 HashTable< Key, Value > Class Template Reference

```
#include <HashTable.h>
```

Collaboration diagram for HashTable< Key, Value >:



Public Types

- using **Hasher** = std::function< uint64_t(const Key &)>

Public Member Functions

- HashTable (const Hasher &hasher, size_t bucketsCount)
- ~HashTable ()
- HashTable (const HashTable &other)
- HashTable & operator= (const HashTable &other)
- HashTable (HashTable &&other)
- HashTable & operator= (HashTable &&other)
- void add (const Key &key, const Value &value)
- void remove (const Key &key)
- bool remove (const Key &key, Value &value)
- bool find (const Key &key, Value &value) const
- std::vector< Value > list () const
- double getLoadFactor () const
- size_t getNumberOfCollisions () const
- size_t getBucketsCount () const
- void rehash (size_t newBuckets)

3.5.1 Detailed Description

template<typename Key, typename Value>
class HashTable< Key, Value >

Hash table implementation. Collission resolution is done by linked list

Parameters

Key	the type of the key
Value	the type of the entry

3.5.2 Constructor & Destructor Documentation

3.5.2.1 HashTable() [1/3]

```
template<typename Key , typename Value >
HashTable< Key, Value >::HashTable (
            const Hasher & hasher,
            size_t bucketsCount )  [inline]
```

Constructor

Parameters

hasher	the hash function that receives Key type and returns uint64_t value
bucketsCount	the initial bucket count

3.5.2.2 ~HashTable()

```
template<typename Key , typename Value >
HashTable< Key, Value >::~HashTable ( )  [inline]
```

Destructor

3.5.2.3 HashTable() [2/3]

```
template<typename Key , typename Value >
HashTable< Key, Value >::HashTable (
            const HashTable< Key, Value > & other )  [inline]
```

Copy constructor It copies all the values from scratch. O(n)

3.5.2.4 HashTable() [3/3]

```
template<typename Key , typename Value >
HashTable< Key, Value >::HashTable (
            HashTable< Key, Value > && other )  [inline]
```

Move constructor It does not copy all the values but only copies the pointers. O(1)

3.5.3 Member Function Documentation

3.5.3.1 add()

```
template<typename Key , typename Value >
void HashTable< Key, Value >::add (
            const Key & key,
            const Value & value )  [inline]
```

Inserts a new entry into the table.

Parameters

key	the key of the entry
value	the value of the entry

3.5.3.2 find()

```
template<typename Key , typename Value >
bool HashTable< Key, Value >::find (
            const Key & key,
            Value & value ) const  [inline]
```

Finds a entry in the table.

Parameters

key	the key of the entry
value	the output reference that receives the value of the entry

Returns

true if the entry was found, false otherwise

3.5.3.3 getBucketsCount()

```
template<typename Key , typename Value >
size_t HashTable< Key, Value >::getBucketsCount ( ) const  [inline]
```

Get number of buckets

Returns

number of buckets

3.5.3.4 getLoadFactor()

```
template<typename Key , typename Value >
double HashTable< Key, Value >::getLoadFactor ( ) const  [inline]
```

Get load factor of the buckets

Returns

load factor

3.5.3.5 getNumberOfCollisions()

```
template<typename Key , typename Value >
size_t HashTable< Key, Value >::getNumberOfCollisions ( ) const  [inline]
```

Get number of collisions so far after the last rehashing

Returns

number of collisions

3.5.3.6 list()

```
template<typename Key , typename Value >
std::vector< Value > HashTable< Key, Value >::list ( ) const  [inline]
```

List every entry in the table

Returns

vector of entries

3.5.3.7 operator=() [1/2]

```
template<typename Key , typename Value >
HashTable & HashTable< Key, Value >::operator= (
            const HashTable< Key, Value > & other )  [inline]
```

Copy assignment operator It copies all the values from scratch. O(n)

3.5.3.8 operator=() [2/2]

```
template<typename Key , typename Value >
HashTable & HashTable< Key, Value >::operator= (
            HashTable< Key, Value > && other )  [inline]
```

Move assignmnet operator It does not copy all the values but only copies the pointers. O(1)

3.5.3.9 rehash()

```
template<typename Key , typename Value >
void HashTable< Key, Value >::rehash (
            size_t newBuckets )  [inline]
```

Rehash the table with the specifid bucket count

Parameters

newBucketsCount	the new bucket count

3.5.3.10 remove() [1/2]

```
template<typename Key , typename Value >
void HashTable< Key, Value >::remove (
            const Key & key )  [inline]
```

Removes a entry from the table.

Parameters

key	the key of the entry

3.5.3.11 remove() [2/2]

```
template<typename Key , typename Value >
bool HashTable< Key, Value >::remove (
            const Key & key,
            Value & value )  [inline]
```

Removes a entry from the table, and copies the removed value to the output reference.

Parameters

key	the key of the entry
value	the output reference that receives the value of the entry

Returns

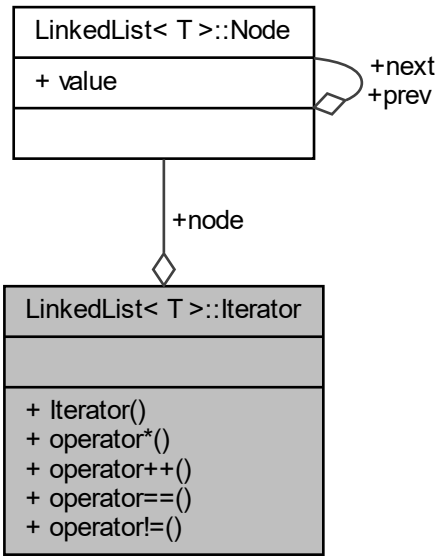
true if the entry was found, false otherwise

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

- src/HashTable.h

3.6 LinkedList< T >::Iterator Class Reference

Collaboration diagram for LinkedList< T >::Iterator:



Public Member Functions

- **Iterator** (Node *node)
- T & **operator*** ()
- Iterator & **operator++** ()
- bool **operator==** (const Iterator &other)
- bool **operator!=** (const Iterator &other)

Public Attributes

- Node * **node**

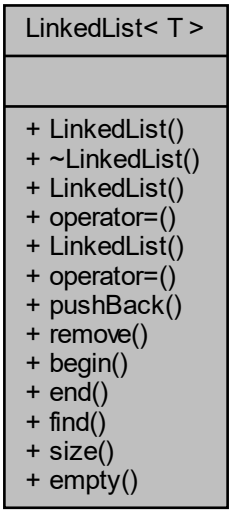
The documentation for this class was generated from the following file:

- src/LinkedList.h

3.7 LinkedList< T > Class Template Reference

`#include <LinkedList.h>`

Collaboration diagram for LinkedList< T >:



Classes

- class Iterator
- struct Node

Public Member Functions

- LinkedList (const LinkedList &other)
- LinkedList & operator= (const LinkedList &other)
- LinkedList (LinkedList &&other)
- LinkedList & operator= (LinkedList &&other)
- void pushBack (T value)
- Iterator remove (const Iterator &it)
- Iterator begin () const
- Iterator end () const
- Iterator find (T value)
- size_t size () const
- bool empty () const

3.7.1 Detailed Description

template<typename T>
class LinkedList< T >

Linked list implementation internally used by HashTable

3.7.2 Constructor & Destructor Documentation

3.7.2.1 LinkedList() [1/2]

```
template<typename T >
LinkedList< T >::LinkedList (
            const LinkedList< T > & other )  [inline]
```

Copy constructor It copies all the values from scratch. O(n)

3.7.2.2 LinkedList() [2/2]

```
template<typename T >
LinkedList< T >::LinkedList (
            LinkedList< T > && other )  [inline]
```

Move constructor It does not copy all the values but only copies the pointers. O(1)

3.7.3 Member Function Documentation

3.7.3.1 begin()

```
template<typename T >
Iterator LinkedList< T >::begin ( ) const  [inline]
```

Returns the iterator pointing to the first element

Returns

begin iterator

3.7.3.2 empty()

```
template<typename T >
bool LinkedList< T >::empty ( ) const  [inline]
```

Is this list empty?

Returns

true if empty, false otherwise

3.7.3.3 end()

```
template<typename T >
Iterator LinkedList< T >::end ( ) const  [inline]
```

Returns the iterator pointing to the tail sentinetal node

Returns

end iterator

3.7.3.4 find()

```
template<typename T >
Iterator LinkedList< T >::find (
            T value )  [inline]
```

Find the iterator pointing to the element with the specified value

Returns

iterator pointing to the element if found, end() otherwise

3.7.3.5 operator=() [1/2]

```
template<typename T >
LinkedList & LinkedList< T >::operator= (
            const LinkedList< T > & other )  [inline]
```

Copy assignment operator It copies all the values from scratch. O(n)

3.7.3.6 operator=() [2/2]

```
template<typename T >
LinkedList & LinkedList< T >::operator= (
            LinkedList< T > && other )  [inline]
```

Move assignmnet operator It does not copy all the values but only copies the pointers. O(1)

3.7.3.7 pushBack()

```
template<typename T >
void LinkedList< T >::pushBack (
            T value )  [inline]
```

Inserts a new entry at the back

Parameters

value	the entry to add

3.7.3.8 remove()

```
template<typename T >
Iterator LinkedList< T >::remove (
            const Iterator & it )  [inline]
```

Remove the entry at the specified iterator point and returns the next iterator

Parameters

it	the iterator point to remove

3.7.3.9 size()

```
template<typename T >
size_t LinkedList< T >::size ( ) const  [inline]
```

Returns the number of elements in the list

Returns

number of elements

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

- src/LinkedList.h

3.8 Movie Class Reference

Collaboration diagram for Movie:

Public Member Functions

- Movie (const MovieID &id, const std::string &title, const std::string &lang, int year, const std::string &director, bool isAdult)
- void setID (const MovieID &id)
- void **setTitle** (const std::string &title)
- void **setLang** (const std::string &lang)
- void **setDirector** (const std::string &director)
- void **setYear** (int year)
- void **setIsAdult** (bool isAdult)
- **Movie** (const Movie &mo)
- MovieID getID () const
- std::string **getTitle** () const
- std::string **getLang** () const
- std::string **getDirector** () const
- int **getYear** () const
- bool **getIsAdult** () const
- bool **operator**> (const Movie &s1) const
- bool **operator**< (const Movie &s1) const
- void **vDisplay** (std::ostream &os) const

Friends

- std::ostream & **operator**<< (std::ostream &os, const Movie &s1)

3.8.1 Constructor & Destructor Documentation

3.8.1.1 Movie()

```
Movie::Movie (
            const MovieID & id,
            const std::string & title,
            const std::string & lang,
            int year,
            const std::string & director,
            bool isAdult )  [inline]
```

Constructors

3.8.2 Member Function Documentation

3.8.2.1 getID()

```
MovieID Movie::getID ( ) const  [inline]
```

Getters

3.8.2.2 setID()

```
void Movie::setID (
            const MovieID & id )  [inline]
```

Setters

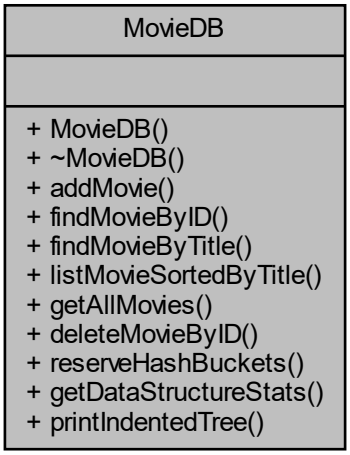
The documentation for this class was generated from the following file:

- src/Movie.h

3.9 MovieDB Class Reference

`#include <MovieDB.h>`

Collaboration diagram for MovieDB:



Public Member Functions

- void addMovie (const Movie &movie)
- bool findMovieByID (const MovieID &id, Movie &movie)
- std::vector< Movie > findMovieByTitle (const std::string &title)
- std::vector< Movie > listMovieSortedByTitle ()
- std::vector< Movie > getAllMovies () const
- bool deleteMovieByID (const MovieID &id, Movie &deletedMovie)
- void reserveHashBuckets (size_t buckets)
- MovieDBDSStats getDataStructureStats () const
- void **printIndentedTree** (void visit(Movie &, int))

3.9.1 Detailed Description

Movie database that internally uses both BST and hash table to support required operations efficiently.

3.9.2 Member Function Documentation

3.9.2.1 addMovie()

```
void MovieDB::addMovie (
            const Movie & movie )
```

Insert movie to the database.

Parameters

movie	Movie to be inserted.

3.9.2.2 deleteMovieByID()

```
bool MovieDB::deleteMovieByID (
            const MovieID & id,
            Movie & deletedMovie )
```

Delete movie by it primary key.

Parameters

id	Movie's primary key.
movie	output reference to be filled with the deleted movie.

Returns

true if found, false otherwise.

3.9.2.3 findMovieByID()

```
bool MovieDB::findMovieByID (
            const MovieID & id,
            Movie & movie )
```

Find movie by its primary key.

Parameters

id	Movie's primary key.
movie	output reference to be filled with the found movie.

Returns

true if found, false otherwise.

3.9.2.4 findMovieByTitle()

```
std::vector< Movie > MovieDB::findMovieByTitle (
            const std::string & title )
```

Find movie by its secondary key (title)

Parameters

title	the title of the movie.

Returns

vector of movies with the same title.

3.9.2.5 getAllMovies()

```
std::vector< Movie > MovieDB::getAllMovies ( ) const
```

List all movies using hash table

Returns

vector of movies

3.9.2.6 getDataStructureStats()

```
MovieDBDSStats MovieDB::getDataStructureStats ( ) const
```

Get statistics of the internal data structures of the database.

Returns

stats

3.9.2.7 listMovieSortedByTitle()

```
std::vector< Movie > MovieDB::listMovieSortedByTitle ( )
```

List movie sorted by their secondary key (title)

Returns

vector of movies sorted by their titles.

3.9.2.8 reserveHashBuckets()

```
void MovieDB::reserveHashBuckets (
            size_t buckets )
```

Reserve hash table buckets.

Parameters

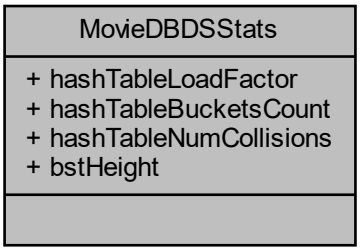
buckets	number of buckets to be reserved.

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

- src/MovieDB.h
- src/MovieDB.cpp

3.10 MovieDBDSStats Struct Reference

Collaboration diagram for MovieDBDSStats:



Public Attributes

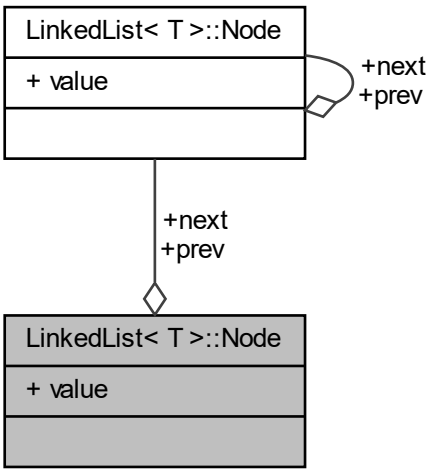
- double **hashTableLoadFactor**
- size_t **hashTableBucketsCount**
- size_t **hashTableNumCollisions**
- size_t **bstHeight**

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

- src/MovieDB.h

3.11 LinkedList< T >::Node Struct Reference

Collaboration diagram for LinkedList< T >::Node:



Public Attributes

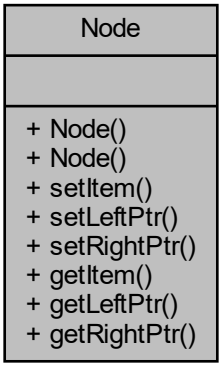
- T **value** {}
- Node * **prev** { nullptr }
- Node * **next** { nullptr }

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

- src/LinkedList.h

3.12 Node Class Reference

Collaboration diagram for Node:



Public Member Functions

- **Node** (const Movie &anItem)
- **Node** (const Movie &anItem, Node ∗left, Node ∗right)
- void **setItem** (const Movie &anItem)
- void **setLeftPtr** (Node ∗left)
- void **setRightPtr** (Node ∗right)
- Movie **getItem** () const
- Node ∗ **getLeftPtr** () const
- Node ∗ **getRightPtr** () const

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

- src/BSTNode.h

Chapter 4

File Documentation

4.1 App.h

```
1 #pragma once
2
3 #include <stack>
4 #include <vector>
5
6 #include "FileLoader.h"
7 #include "FileWriter.h"
8 #include "MovieDB.h"
9
14 class App {
15   public:
16     App() = default;
17     ~App() = default;
18
22     int run();
23
24   private:
25     void printMenu();
26
27     void processLoadFileCmd();
28     void processSaveFileCmd();
29     void processSearchByKeyCmd();
30     void processSearchByTitleCmd();
31     void processPrintSortedCmd();
32     void processAddCmd();
33     void processDeleteByKeyCmd();
34     void processUndoDeleteCmd();
35     void processDisplayAllKeyCmd();
36     void processDisplayAllMemCmd();
37     void processDisplayHashStatsCmd() const;
38     void processDisplayIndentedTreeCmd();
39     void resetMovieDB() {
40         movieDB = MovieDB();
41     }
42
43     MovieDB movieDB;
44     std::stack<Movie> deletedMovieStack;
45 };
```

4.2 BSTNode.h

```
1 #pragma once
2 #include "Movie.h"
3
4 class Node {
5   private:
6     Movie item;       // Data portion
7     Node* leftPtr;    // Pointer to left child
8     Node* rightPtr;   // Pointer to right child
9
10   public:
11     // constructors
12     Node(const Movie& anItem) {
13         item = anItem;
```

```
14          leftPtr = 0;
15          rightPtr = 0;
16      }
17      Node(const Movie& anItem,
18           Node* left,
19           Node* right) {
20          item = anItem;
21          leftPtr = left;
22          rightPtr = right;
23      }
24      // setters
25      void setItem(const Movie& anItem) {
26          item = anItem;
27      }
28      void setLeftPtr(Node* left) {
29          leftPtr = left;
30      }
31      void setRightPtr(Node* right) {
32          rightPtr = right;
33      }
34
35      // getters
36      Movie getItem() const {
37          return item;
38      }
39      Node* getLeftPtr() const {
40          return leftPtr;
41      }
42      Node* getRightPtr() const {
43          return rightPtr;
44      }
45  };
```

4.3 FIleLoader.h

```
1  #pragma once
2
3  #include <fstream>
4
5  #include "MovieDB.h"
6
10 class FileLoader {
11   public:
16     FileLoader(std::ifstream&& stream);
17     ~FileLoader() = default;
18
23     void load(MovieDB& db);
24
25   private:
26     std::ifstream file;
27 };
```

4.4 FileWriter.h

```
1  #pragma once
2
3  #include <fstream>
4
5  #include "MovieDB.h"
6
10 class FileWriter {
11   public:
16     FileWriter(std::ofstream&& file);
17     ~FileWriter() = default;
18
23     void save(const MovieDB& db);
24
25   private:
26     std::ofstream file;
27 };
```

4.5 HashTable.h

```
1  #pragma once
2
```

```
3 #include <functional>
4 #include <vector>
5
6 #include "LinkedList.h"
7
14 template <typename Key, typename Value>
15 class HashTable {
16   public:
17     using Hasher = std::function<uint64_t(const Key&)>;
23     HashTable(const Hasher& hasher, size_t bucketsCount)
24         : hasher(hasher), bucketsCount(bucketsCount), buckets(new LinkedList<Item>[bucketsCount]) {
25     }
26
30     ~HashTable() {
31         delete[] buckets;
32     }
33
38     HashTable(const HashTable& other)
39         : bucketsCount(other.bucketsCount), buckets(new LinkedList<Item>[other.bucketsCount]),
       numberOfCollisions(other.numberOfCollisions), filledBuckets(other.filledBuckets) {
40         for (int i = 0; i < other.bucketsCount; i++) {
41             buckets[i] = other.buckets[i];
42         }
43     }
44
49     HashTable& operator=(const HashTable& other) {
50         delete buckets;
51         bucketsCount = other.bucketsCount;
52         buckets = new LinkedList<Item>[other.bucketsCount];
53         numberOfCollisions = other.numberOfCollisions;
54         filledBuckets = other.filledBuckets;
55         for (int i = 0; i < other.bucketsCount; i++) {
56             buckets[i] = other.buckets[i];
57         }
58         return *this;
59     }
60
65     HashTable(HashTable&& other) {
66         bucketsCount = other.bucketsCount;
67         buckets = other.buckets;
68         numberOfCollisions = other.numberOfCollisions;
69         filledBuckets = other.filledBuckets;
70         other.bucketsCount = 0;
71         other.buckets = nullptr;
72         other.numberOfCollisions = 0;
73         other.filledBuckets = 0;
74     }
75
80     HashTable& operator=(HashTable&& other) {
81         bucketsCount = other.bucketsCount;
82         buckets = other.buckets;
83         numberOfCollisions = other.numberOfCollisions;
84         filledBuckets = other.filledBuckets;
85         other.bucketsCount = 0;
86         other.buckets = nullptr;
87         other.numberOfCollisions = 0;
88         other.filledBuckets = 0;
89         return *this;
90     }
91
97     void add(const Key& key, const Value& value) {
98         auto hash = hashKey(key);
99         auto& bucket = buckets[hash];
100         auto it = bucket.find({ key });
101         if (it != bucket.end()) {
102             (*it).value = value;
103         } else {
104             if (bucket.empty()) {
105                 filledBuckets++;
106             } else {
107                 numberOfCollisions++;
108             }
109             bucket.pushBack({ key, value });
110         }
111     }
112
117     void remove(const Key& key) {
118         auto hash = hashKey(key);
119         auto& bucket = buckets[hash];
120         auto it = bucket.find({ key });
121         if (it == bucket.end()) {
122             return;
123         }
124         bucket.remove(it);
125         if (bucket.empty()) {
126             filledBuckets--;
127         }
```

```
128     }
129
136     bool remove(const Key& key, Value& value) {
137         auto hash = hashKey(key);
138         auto& bucket = buckets[hash];
139         auto it = bucket.find({ key });
140         if (it == bucket.end()) {
141             return false;
142         }
143         value = (*it).value;
144         bucket.remove(it);
145         if (bucket.empty()) {
146             filledBuckets--;
147         }
148         return true;
149     }
150
157     bool find(const Key& key, Value& value) const {
158         auto hash = hashKey(key);
159         auto& bucket = buckets[hash];
160         auto it = bucket.find({ key });
161         if (it == bucket.end()) {
162             return false;
163         }
164         value = (*it).value;
165         return true;
166     }
167
172     std::vector<Value> list() const {
173         std::vector<Value> result{};
174         for (int i = 0; i < bucketsCount; i++) {
175             auto& bucket = buckets[i];
176             for (auto it = bucket.begin(); it != bucket.end(); ++it) {
177                 result.push_back((*it).value);
178             }
179         }
180         return result;
181     }
182
187     double getLoadFactor() const {
188         return static_cast<double>(filledBuckets) / bucketsCount;
189     }
190
195     size_t getNumberOfCollisions() const {
196         return numberOfCollisions;
197     }
198
203     size_t getBucketsCount() const {
204         return bucketsCount;
205     }
206
211     void rehash(size_t newBuckets) {
212         auto oldBuckets = buckets;
213         size_t oldBucketsCount = bucketsCount;
214         buckets = new LinkedList<Item>[newBuckets];
215         bucketsCount = newBuckets;
216         numberOfCollisions = 0;
217         filledBuckets = 0;
218
219         for (size_t i = 0; i < oldBucketsCount; i++) {
220             auto& bucket = oldBuckets[i];
221             for (auto it = bucket.begin(); it != bucket.end(); ++it) {
222                 add((*it).key, (*it).value);
223             }
224         }
225     }
226
227   private:
228     struct Item {
229         Key key;
230         Value value;
231
232         bool operator==(const Item& other) const {
233             return key == other.key;
234         }
235         bool operator!=(const Item& other) const {
236             return key != other.key;
237         }
238     };
239
240     size_t hashKey(const Key& key) const {
241         return hasher(key) % bucketsCount;
242     }
243
244     Hasher hasher;
245     size_t bucketsCount{ 0 };
246     LinkedList<Item>* buckets{ nullptr };
```

```
247     size_t numberOfCollisions{ 0 };
248     size_t filledBuckets{ 0 };
249 };
```

4.6 LinkedList.h

```
1 #pragma once
2
6 template <typename T>
7 class LinkedList {
8   public:
9     struct Node {
10        T value{};
11        Node* prev{ nullptr };
12        Node* next{ nullptr };
13    };
14
15    // Iterator class
16    class Iterator {
17      public:
18        Iterator(Node* node)
19            : node(node) {
20        }
21        T& operator*() {
22            return node->value;
23        }
24        Iterator& operator++() {
25            node = node->next;
26            return *this;
27        }
28        bool operator==(const Iterator& other) {
29            return node == other.node;
30        }
31        bool operator!=(const Iterator& other) {
32            return node != other.node;
33        }
34        Node* node;
35    };
36
37    LinkedList()
38        : size_(0) {
39        // head and tail sentinetals
40        head = new Node();
41        tail = new Node();
42        head->next = tail;
43        tail->prev = head;
44    }
45
46    ~LinkedList() {
47        auto it = begin();
48        // when head = nullptr && tail == nullptr, this will be noop
49        while (it != end()) {
50            it = remove(it);
51        }
52        delete head;
53        delete tail;
54    }
55
60    LinkedList(const LinkedList& other) {
61        auto it = other.begin();
62        while (it != other.end()) {
63            pushBack(*it);
64            ++it;
65        }
66    }
67
72    LinkedList& operator=(const LinkedList& other) {
73        auto it = other.begin();
74        while (it != other.end()) {
75            pushBack(*it);
76            ++it;
77        }
78        return *this;
79    }
80
85    LinkedList(LinkedList&& other) {
86        head = other.head;
87        tail = other.tail;
88        size_ = other.size_;
89        other.head = nullptr;
90        other.tail = nullptr;
91        other.size_ = 0;
92    }
```

```
93
98      LinkedList& operator=(LinkedList&& other) {
99          head = other.head;
100         tail = other.tail;
101         size_ = other.size_;
102         other.head = nullptr;
103         other.tail = nullptr;
104         other.size_ = 0;
105         return *this;
106     }
107
112     void pushBack(T value) {
113         auto node = new Node();
114         node->value = value;
115
116         auto prev = tail->prev;
117         node->next = tail;
118         node->prev = prev;
119
120         prev->next = node;
121         tail->prev = node;
122         size_++;
123     }
124
129     Iterator remove(const Iterator& it) {
130         auto node = it.node;
131         auto next = node->next;
132         auto prev = node->prev;
133
134         prev->next = next;
135         next->prev = prev;
136
137         delete node;
138
139         size_--;
140         return Iterator(next);
141     }
142
147     Iterator begin() const {
148         return Iterator(head->next);
149     }
150
155     Iterator end() const {
156         return Iterator(tail);
157     }
158
163     Iterator find(T value) {
164         auto it = begin();
165         while (it != end() && *it != value) {
166             ++it;
167         }
168         return it;
169     }
170
175     size_t size() const {
176         return size_;
177     }
178
183     bool empty() const {
184         return size_ == 0;
185     }
186
187   private:
188     Node* head = new Node;
189     Node* tail = new Node;
190     size_t size_;
191 };
```

4.7 Movie.h

```
1 #pragma once
2
3 #include <iostream>
4 #include <string>
5
6 using MovieID = std::string;
7
8 class Movie;  // Forward Decleration
9
10 // Overloaded « operator
11 std::ostream& operator«(std::ostream&, const Movie&);
12
13 class Movie {
```

```
14   public:
18     Movie(const MovieID& id, const std::string& title, const std::string& lang,
19           int year, const std::string& director, bool isAdult)
20         : id(id), title(title), lang(lang), year(year), director(director), isAdult(isAdult) {
21     }
22     Movie() = default;
23
27     void setID(const MovieID& id) {
28         this->id = id;
29     }
30     void setTitle(const std::string& title) {
31         this->title = title;
32     }
33     void setLang(const std::string& lang) {
34         this->lang = lang;
35     }
36     void setDirector(const std::string& director) {
37         this->director = director;
38     }
39     void setYear(int year) {
40         this->year = year;
41     }
42     void setIsAdult(bool isAdult) {
43         this->isAdult = isAdult;
44     }
45     // Copy Constructor
46     Movie(const Movie& mo) {
47         id = mo.getID();
48         title = mo.getTitle();
49         lang = mo.getLang();
50         year = mo.getYear();
51         director = mo.getDirector();
52         isAdult = mo.getIsAdult();
53     }
57     MovieID getID() const {
58         return id;
59     }
60     std::string getTitle() const {
61         return title;
62     }
63     std::string getLang() const {
64         return lang;
65     }
66     std::string getDirector() const {
67         return director;
68     }
69     int getYear() const {
70         return year;
71     }
72     bool getIsAdult() const {
73         return isAdult;
74     }
75     bool operator>(const Movie& s1) const {
76         return title > s1.title;
77     }
78     bool operator<(const Movie& s1) const {
79         return title < s1.title;
80     }
81     //    Movie& operator = (const Movie& s1) const { return this = s1;}
82     friend std::ostream& operator«(std::ostream& os, const Movie& s1) {
83         os « s1.title « std::endl;
84         return os;
85     }
86     void vDisplay(std::ostream& os) const {
87         os « "ID: " « id « std::endl;
88         os « "Title: " « title « std::endl;
89         os « "Language: " « lang « std::endl;
90         os « "Year: " « year « std::endl;
91         os « "Director: " « director « std::endl;
92         os « "Is Adult: " « isAdult « std::endl;
93     }
94
95   private:
96     MovieID id{ "" };
97     std::string title{ "none" };
98     std::string lang{ "none" };
99     int year{};
100     std::string director{ "none" };
101     bool isAdult{ false };
102 };
```

4.8 MovieBST.h

```
1 #pragma once
```

```
2 #include <stack>
3 #include <vector>
4
5 #include "BSTNode.h"
6 #include "Movie.h"
7
8 class BinaryTree {
9   private:
10     Node* root;  // root of the tree
11     int count;   // number of nodes in the tree
12   public:
13     // Constructor
14     BinaryTree();
15
16     // Destructor
17     ~BinaryTree();
18
19     // Binary Tree operations
20     bool insert(Movie dataIn);
21     bool remove(Movie);
22     Node* getMax(Node*);
23     Node* getMin(Node*);
24     std::vector<Movie> inOrder() const;
25     std::vector<Movie> DFS(std::string);
26     void preOrder() const;
27     void postOrder() const;
28     void printTree(void visit(Movie&, int)) const {
29         _printTree(visit, root, 1);
30     }
31     int getCount() const {
32         return count;
33     }
34     bool isEmpty() const {
35         return count == 0;
36     }
37     bool isLeafNode(Node* node) {
38         return !node->getLeftPtr() && !node->getRightPtr();
39     }
40     Node* getParent(Node*, Node*);
41
42   private:
43     Node* _insert(Node* nodePtr, Node* newNode);
44     Node* _finder(Movie dataIn, Node* nodePtr);
45     void _remove(Movie dataIn, Node* nodePtr);
46     void _inOrder(Node* root, std::vector<Movie>&) const;
47     void _destroy(Node* root);
48     void _dfs(std::stack<Node*>&, std::string, std::vector<Movie>&);
49     void _printTree(void visit(Movie&, int), Node* nodePtr, int level) const;
50 };
```

4.9 MovieDB.h

```
1 #pragma once
2
3 #include <functional>
4 #include <string>
5 #include <vector>
6
7 #include "HashTable.h"
8 #include "Movie.h"
9 #include "MovieBST.h"
10
11 uint64_t movieIDHasher(const MovieID& id);
12
13 struct MovieDBDSStats {
14     double hashTableLoadFactor;
15     size_t hashTableBucketsCount;
16     size_t hashTableNumCollisions;
17
18     size_t bstHeight;
19 };
20
24 class MovieDB {
25   public:
26     MovieDB();
27     ~MovieDB();
28
33     void addMovie(const Movie& movie);
40     bool findMovieByID(const MovieID& id, Movie& movie);
46     std::vector<Movie> findMovieByTitle(const std::string& title);
51     std::vector<Movie> listMovieSortedByTitle();
56     std::vector<Movie> getAllMovies() const;
63     bool deleteMovieByID(const MovieID& id, Movie& deletedMovie);
```

```
68      void reserveHashBuckets(size_t buckets);
73      MovieDBDSStats getDataStructureStats() const;
74      void printIndentedTree(void visit(Movie&, int)) {
75          BST.printTree(visit);
76      }
77
78    private:
79      HashTable<MovieID, Movie> hashTable;
80      BinaryTree BST;
81  };
```

4.10 Util.h

```
1  #pragma once
2
3  #include <string>
4  #include <vector>
5
12 // From
       https://github.com/Themaister/Granite/blob/3e0ac0da7315e6a8ae6584ee0cc5c9cef82f02e5/util/string_helpers.cpp#L28
13 inline static std::vector<std::string> split(const std::string& str, const char* delim) {
14      if (str.empty())
15          return {};
16      std::vector<std::string> ret;
17      size_t startIndex = 0;
18      size_t index = 0;
19      while ((index = str.find_first_of(delim, startIndex)) != std::string::npos) {
20          ret.push_back(str.substr(startIndex, index - startIndex));
21          startIndex = index + 1;
22          if (index == str.size() - 1)
23              ret.emplace_back();
24      }
25
26      if (startIndex < str.size())
27          ret.push_back(str.substr(startIndex));
28
29      return ret;
30  }
```

Index

~BinaryTree
    BinaryTree, 7
~HashTable
    HashTable< Key, Value >, 13

add
    HashTable< Key, Value >, 13
addMovie
    MovieDB, 24
App, 5
    run, 5

begin
    LinkedList< T >, 19
BinaryTree, 6
    ~BinaryTree, 7
    BinaryTree, 7
    DFS, 7
    getMax, 7
    getMin, 7
    getParent, 7
    inOrder, 7
    insert, 8
    remove, 8

deleteMovieByID
    MovieDB, 25
DFS
    BinaryTree, 7

empty
    LinkedList< T >, 19
end
    LinkedList< T >, 19

FileLoader, 8
    FileLoader, 9
    load, 9
FileWriter, 9
    FileWriter, 10
    save, 10
find
    HashTable< Key, Value >, 13
    LinkedList< T >, 20
findMovieByID
    MovieDB, 25
findMovieByTitle
    MovieDB, 25

getAllMovies
    MovieDB, 26
getBucketsCount
    HashTable< Key, Value >, 14
getDataStructureStats
    MovieDB, 26
getID
    Movie, 23
getLoadFactor
    HashTable< Key, Value >, 14
getMax
    BinaryTree, 7
getMin
    BinaryTree, 7
getNumberOfCollisions
    HashTable< Key, Value >, 14
getParent
    BinaryTree, 7

HashTable
    HashTable< Key, Value >, 12, 13
HashTable< Key, Value >, 11
    ~HashTable, 13
    add, 13
    find, 13
    getBucketsCount, 14
    getLoadFactor, 14
    getNumberOfCollisions, 14
    HashTable, 12, 13
    list, 15
    operator=, 15
    rehash, 15
    remove, 16

inOrder
    BinaryTree, 7
insert
    BinaryTree, 8

LinkedList
    LinkedList< T >, 19
LinkedList< T >, 18
    begin, 19
    empty, 19
    end, 19
    find, 20
    LinkedList, 19
    operator=, 20
    pushBack, 20
    remove, 21
    size, 21
LinkedList< T >::Iterator, 17

LinkedList< T >::Node, 28
list
    HashTable< Key, Value >, 15
listMovieSortedByTitle
    MovieDB, 26
load
    FileLoader, 9

Movie, 22
    getID, 23
    Movie, 23
    setID, 23
MovieDB, 24
    addMovie, 24
    deleteMovieByID, 25
    findMovieByID, 25
    findMovieByTitle, 25
    getAllMovies, 26
    getDataStructureStats, 26
    listMovieSortedByTitle, 26
    reserveHashBuckets, 26
MovieDBDSStats, 27

Node, 29

operator=
    HashTable< Key, Value >, 15
    LinkedList< T >, 20

pushBack
    LinkedList< T >, 20

rehash
    HashTable< Key, Value >, 15
remove
    BinaryTree, 8
    HashTable< Key, Value >, 16
    LinkedList< T >, 21
reserveHashBuckets
    MovieDB, 26
run
    App, 5

save
    FileWriter, 10
setID
    Movie, 23
size
    LinkedList< T >, 21
src/App.h, 31
src/BSTNode.h, 31
src/FIleLoader.h, 32
src/FileWriter.h, 32
src/HashTable.h, 32
src/LinkedList.h, 35
src/Movie.h, 36
src/MovieBST.h, 37
src/MovieDB.h, 38
src/Util.h, 39