Search Engine

Generated by Doxygen 1.8.14

Contents

Chapter 1

Hierarchical Index

1.1 Class Hierarchy

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

Chapter 2

Class Index

2.1 Class List

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

Chapter 3

File Index

3.1 File List

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

Chapter 4

Class Documentation

4.1 AVLTree< T > Class Template Reference

`#include <avltree.hpp>`

Inheritance diagram for AVLTree< T >:

Public Member Functions

- **AVLTree** (const AVLTree< T > &)
- AVLTree< T > & **operator=** (const AVLTree< T > &)
- bool contains (const T &) const
- std::pair< T, bool > search (const T &)
- void insert (const T &)
- bool isEmpty () const
- void makeEmpty ()
- std::pair< T, bool > stringSearch (const std::string &)
- void stringInsert (const std::string &, int, int, int)
- T & findMax ()
- T & findMin ()
- std::ostream & print (std::ostream &) const
- std::vector< T > **getTopFifty** ()
- int getTerms () const

Friends

- template<class U >
  std::ostream & **operator**<< (std::ostream &, const AVLTree< U > &)

4.1.1 Detailed Description

template<class T>
class AVLTree< T >

Implements an ordered index as a self-balancing AVL binary tree.

Also assumes that any values passed in are unique or can have data appended to them with operator+= without changing their ordering.

4.1.2 Member Function Documentation

4.1.2.1 contains()

```
template<class T >
bool AVLTree< T >::contains (
            const T & arg ) const  [virtual]
```

Determines whether or not the passed argument is an element of the tree

Implements IndexInterface< T >.

4.1.2.2 findMax()

```
template<class T >
T & AVLTree< T >::findMax ( )
```

Returns the maximum value contained in the tree.

4.1.2.3 findMin()

```
template<class T >
T & AVLTree< T >::findMin ( )
```

Returns the minimum value contained in the tree.

4.1.2.4 getTerms()

```
template<class T >
int AVLTree< T >::getTerms ( ) const  [virtual]
```

Returns the number of terms that have been indexed by the tree

Implements IndexInterface< T >.

4.1.2.5 insert()

```
template<class T >
void AVLTree< T >::insert (
            const T & arg )  [virtual]
```

Determines where a passed value should be inserted into the tree and inserts there.

Implements IndexInterface< T >.

4.1.2.6 isEmpty()

```
template<class T >
bool AVLTree< T >::isEmpty ( ) const  [virtual]
```

Returns true if tree is empty, false if not.

Implements IndexInterface< T >.

4.1.2.7 makeEmpty()

```
template<class T >
void AVLTree< T >::makeEmpty ( )  [virtual]
```

Empties the tree and frees all allocated memory

Implements IndexInterface< T >.

4.1.2.8 print()

```
template<class T >
std::ostream & AVLTree< T >::print (
            std::ostream & os ) const  [virtual]
```

Prints contents of the tree according to a level-order traversal

Reimplemented from IndexInterface< T >.

4.1.2.9 search()

```
template<class T >
std::pair< T, bool > AVLTree< T >::search (
            const T & arg )  [virtual]
```

Searches for the passed value and returns that data from the tree if it is there. If not, indicates with a false value.

Implements IndexInterface< T >.

4.1.2.10 stringInsert()

```
template<class T >
void AVLTree< T >::stringInsert (
            const std::string & word,
            int id,
            int freq,
            int loc )  [virtual]
```

Inserts a string to the index based on its string key

Implements IndexInterface< T >.

4.1.2.11 stringSearch()

```
template<class T >
std::pair< T, bool > AVLTree< T >::stringSearch (
            const std::string & word )  [virtual]
```

Searches the tree for IndexedTerms based on their string keys

Implements IndexInterface< T >.

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

- avltree.hpp

4.2 CsvParser Struct Reference

Public Attributes

- char * **filePath_**
- char **delimiter_**
- int **firstLineIsHeader_**
- char * **errMsg_**
- CsvRow * **header_**
- FILE * **fileHandler_**
- int **fromString_**
- char * **csvString_**
- int **csvStringIter_**

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

- csvparser.h

4.3 CsvRow Struct Reference

Public Attributes

- char ** **fields_**
- int **numOfFields_**

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

- csvparser.h

4.4 DocumentParser Class Reference

Public Member Functions

- DocumentParser (IndexHandler *ih)

  Constructor for DocumentParser.
- void parse (std::string fileName)

  Parses .csv file and seperates fields into an array.
- void loadStopWords (std::string fileName)

  Loads stop words from file and stores them in set.
- std::vector< std::string > questionLookup (int lookupID, std::string documentPath)

4.4.1 Constructor & Destructor Documentation

4.4.1.1 DocumentParser()

```
DocumentParser::DocumentParser (
            IndexHandler * ih )
```

Constructor for DocumentParser.

Parameters

Reference	to index handler for adding words to an index

4.4.2 Member Function Documentation

4.4.2.1 loadStopWords()

```
void DocumentParser::loadStopWords (
            std::string fileName )
```

Loads stop words from file and stores them in set.

Parameters

Name	of file

4.4.2.2 parse()

```
void DocumentParser::parse (
            std::string fileName )
```

Parses .csv file and seperates fields into an array.

Parameters

Name	of file

4.4.2.3 questionLookup()

```
std::vector< std::string > DocumentParser::questionLookup (
            int lookupID,
            std::string documentPath )
```

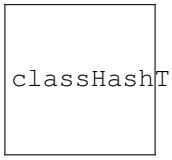
Takes in file path and an ID to be searched and returns question information. Question information is returned as a vector. User needs to be prompted for path to document

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

- documentparser.h
- documentparser.cpp

4.5 HashTable< T > Class Template Reference

Inheritance diagram for HashTable< T >:



Public Member Functions

- HashTable (int=1000000)
- HashTable (const HashTable< T > &)
- HashTable< T > & operator= (const HashTable< T > &)
- bool contains (const T &) const
- std::pair< T, bool > search (const T &)
- void insert (const T &)
- bool isEmpty () const
- void makeEmpty ()
- std::pair< T, bool > stringSearch (const std::string &)
- void stringInsert (const std::string &, int, int, int)
- std::ostream & print (std::ostream &os) const
- int getNumElements () const
- std::pair< T, bool > operator[ ] (const T &)
- std::vector< T > getTopFifty ()
- int getTerms () const

Friends

- template<class U >
  std::ostream & **operator**<< (std::ostream &, const HashTable< U > &)

4.5.1 Constructor & Destructor Documentation

4.5.1.1 HashTable() [1/2]

```
template<class T >
HashTable< T >::HashTable (
            int size = 1000000 )
```

Default constructor initializes array

4.5.1.2 HashTable() [2/2]

```
template<class T >
HashTable< T >::HashTable (
            const HashTable< T > & rhs )
```

Copy constructor copies over elements of vector

4.5.2 Member Function Documentation

4.5.2.1 contains()

```
template<class T >
bool HashTable< T >::contains (
            const T & toBeFound ) const  [virtual]
```

Determines whether or not the table contains the given argument

Implements IndexInterface< T >.

4.5.2.2 getNumElements()

```
template<class T >
int HashTable< T >::getNumElements ( ) const
```

Returns the number of elements in the table

4.5.2.3 getTerms()

```
template<class T >
int HashTable< T >::getTerms ( ) const  [virtual]
```

Returns the number of elements indexed by the table

Implements IndexInterface< T >.

4.5.2.4 getTopFifty()

```
template<class T >
std::vector< T > HashTable< T >::getTopFifty ( )  [virtual]
```

Returns a vector of the top fifty most common sords.

Implements IndexInterface< T >.

4.5.2.5 insert()

```
template<class T >
void HashTable< T >::insert (
            const T & inserted )  [virtual]
```

Inserts new object to position given by hashing function

Implements IndexInterface< T >.

4.5.2.6 isEmpty()

```
template<class T >
bool HashTable< T >::isEmpty ( ) const  [virtual]
```

Determines whether or not there are any elements in the table

Implements IndexInterface< T >.

4.5.2.7 makeEmpty()

```
template<class T >
void HashTable< T >::makeEmpty ( )  [virtual]
```

Clears table of all old elements

Implements IndexInterface< T >.

4.5.2.8 operator=()

```
template<class T >
HashTable< T > & HashTable< T >::operator= (
            const HashTable< T > & rhs )
```

Assignment operator copies over data if given argument is not identical

4.5.2.9 operator[]()

```
template<class T >
std::pair< T, bool > HashTable< T >::operator[] (
            const T & sought )
```

Subscript operator accepts argument of type template parameter and returns object stored at that key in the table

4.5.2.10 print()

```
template<class T >
std::ostream & HashTable< T >::print (
            std::ostream & os ) const  [virtual]
```

Prints all elements in table in order of hash value

Reimplemented from IndexInterface< T >.

4.5.2.11 search()

```
template<class T >
std::pair< T, bool > HashTable< T >::search (
            const T & sought )  [virtual]
```

Returns pair with the sought object and a boolean flag indicating whether or not it was found

Implements IndexInterface< T >.

4.5.2.12 stringInsert()

```
template<class T >
void HashTable< T >::stringInsert (
            const std::string & word,
            int id,
            int freq,
            int loc )  [virtual]
```

Searches for a term matching the given string in the index. If found, appends the data to that term. If not, inserts it to the index.

Implements IndexInterface< T >.

4.5.2.13 stringSearch()

```
template<class T >
std::pair< T, bool > HashTable< T >::stringSearch (
            const std::string & word )  [virtual]
```

Searches the index for an index whose key matches the string passed as argument.

Implements IndexInterface< T >.

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

- hashtable.hpp

4.6 IndexedTerm Class Reference

Public Member Functions

- **IndexedTerm** (std::string="")
- **IndexedTerm** (std::string, int, int, int)
- std::string getTerm () const
- bool isEmpty () const
- int getFrequency (int) const
- std::vector< int > getLocations (int) const
- std::set< int > getQuestionIds () const
- void addQuestion (int)
- void removeQuestion (int)
- std::vector< std::pair< int, int > > print15 ()

  Prints the first 15 values of the sorted Vector of results.
- void sort (std::vector< questionIndex > &, int)
- IndexedTerm **questionAnd** (const IndexedTerm &) const
- IndexedTerm **questionOr** (const IndexedTerm &) const
- bool isInQuestion (int) const
- void addLocation (int, int)
- void removeLocation (int, int)
- void addFrequency (int, int)
- bool isAtLocation (int, int) const
- void appendData (int, int, int) const
- int **getTotalFreq** () const
- bool operator== (const IndexedTerm &) const
- void operator+= (const IndexedTerm &) const
- bool operator> (const IndexedTerm &) const
- bool operator< (const IndexedTerm &) const

Friends

- std::ostream & **operator<<** (std::ostream &os, IndexedTerm it)

4.6.1 Member Function Documentation

4.6.1.1 addFrequency()

```
void IndexedTerm::addFrequency (
            int questionId,
            int frequency )
```

Adds frequency to frequency of specified question

4.6.1.2 addLocation()

```
void IndexedTerm::addLocation (
            int questionId,
            int location )
```

Adds location to location vector of specified question

4.6.1.3 addQuestion()

```
void IndexedTerm::addQuestion (
            int questionId )
```

Adds questionId to the set of questions this term is found in.

4.6.1.4 appendData()

```
void IndexedTerm::appendData (
            int questionId,
            int frequency,
            int location ) const
```

Appends all data associated with a particular question to the term.

4.6.1.5 getFrequency()

```
int IndexedTerm::getFrequency (
            int questionID ) const
```

Retrieves the frequency of the question ID passed as argument; returns 0 if not present.

4.6.1.6 getLocations()

```
std::vector< int > IndexedTerm::getLocations (
            int questionId ) const
```

Returns the vector of locations attached to the given question ID.

4.6.1.7 getQuestionIds()

```
std::set< int > IndexedTerm::getQuestionIds ( ) const
```

Returns a set of all of the question IDs that this term is found in

4.6.1.8 getTerm()

```
std::string IndexedTerm::getTerm ( ) const
```

Gets value of search term

4.6.1.9 isAtLocation()

```
bool IndexedTerm::isAtLocation (
            int questionId,
            int location ) const
```

Determines whether or not the term appears in the given question at the given location.

4.6.1.10 isEmpty()

```
bool IndexedTerm::isEmpty ( ) const
```

Determines whether or not the set of question IDs is empty

4.6.1.11 isInQuestion()

```
bool IndexedTerm::isInQuestion (
            int questionId ) const
```

Returns true if term is in question, false if not

4.6.1.12 operator+=()

```
void IndexedTerm::operator+= (
            const IndexedTerm & rhs ) const
```

Addition Assignment Operator adds question ID to list if it isn't there, if it is present, adds frequency

4.6.1.13 operator<()

```
bool IndexedTerm::operator< (
            const IndexedTerm & rhs ) const
```

Compares the terms using lexicographical comparison of the ASCII values of each character in the string.

4.6.1.14 operator==()

```
bool IndexedTerm::operator== (
            const IndexedTerm & rhs ) const
```

Equality operator checks keys (terms) for equality

4.6.1.15 operator>()

```
bool IndexedTerm::operator> (
            const IndexedTerm & rhs ) const
```

Compares the terms using lexicographical comparison of the ASCII values of each character in the string.

4.6.1.16 print15()

```
std::vector< std::pair< int, int > > IndexedTerm::print15 ( )
```

Prints the first 15 values of the sorted Vector of results.

Returns

a vector of pairs containing questionIDs and and frequencies of a term

4.6.1.17 removeLocation()

```
void IndexedTerm::removeLocation (
            int questionId,
            int location )
```

Removes location from location vector of specified question

4.6.1.18 removeQuestion()

```
void IndexedTerm::removeQuestion (
            int questionId )
```

Removes the specified question from the set if there or throws error if not

4.6.1.19 sort()

```
void IndexedTerm::sort (
            std::vector< questionIndex > & output,
            int position )
```

Sorts a vector in descending order

Parameters

the	vector to sort
the	location to begin sorting

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

- indexedterm.h
- indexedterm.cpp

4.7 IndexHandler Class Reference

Public Member Functions

- **IndexHandler** (std::string="hash")
- void addToIndex (std::string, int, int, int)
- std::pair< IndexedTerm, bool > searchIndex (std::string)
- void setNumQuestions (int)
- void writeToDisk ()
- void readFromDisk ()
- void updateTopFifty ()
- int getNumTerms () const
- std::vector< std::string > getTopFifty ()
- int **getQuestionsIndexed** ()

4.7.1 Member Function Documentation

4.7.1.1 addToIndex()

```
void IndexHandler::addToIndex (
            std::string term,
            int questionId,
            int frequency,
            int location )
```

Adds an object with the specified term, question ID, and frequency to the index.

4.7.1.2 getNumTerms()

```
int IndexHandler::getNumTerms ( ) const
```

Returns the number of terms indexed by the index

4.7.1.3 getTopFifty()

```
std::vector< std::string > IndexHandler::getTopFifty ( )
```

Returns the top fifty terms

4.7.1.4 readFromDisk()

```
void IndexHandler::readFromDisk ( )
```

Reads the index from a persistent file location in disk

4.7.1.5 searchIndex()

```
std::pair< IndexedTerm, bool > IndexHandler::searchIndex (
            std::string toBeFound )
```

Searches the index for the specified term.

4.7.1.6 setNumQuestions()

```
void IndexHandler::setNumQuestions (
            int numQuestions )
```

Sets the number of questions tracker in the index

4.7.1.7 updateTopFifty()

```
void IndexHandler::updateTopFifty ( )
```

Updates the top fifty elements of the index to a top fifty usable to the user.

4.7.1.8 writeToDisk()

```
void IndexHandler::writeToDisk ( )
```

Writes the index to a persistent file location in disk

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

- indexhandler.h
- indexhandler.cpp

4.8 IndexInterface< T > Class Template Reference

Inheritance diagram for IndexInterface< T >:

Public Member Functions

- **IndexInterface** (const IndexInterface< T > &)
- IndexInterface< T > & **operator=** (const IndexInterface< T > &)
- int **getNumQuestions** () const
- void **setNumQuestions** (int questionsIndexed)
- virtual bool **contains** (const T &) const =0
- virtual std::pair< T, bool > **search** (const T &)=0
- virtual void **insert** (const T &)=0
- virtual bool **isEmpty** () const =0
- virtual void **makeEmpty** ()=0
- virtual std::pair< T, bool > **stringSearch** (const std::string &)=0
- virtual void **stringInsert** (const std::string &, int, int, int)=0
- virtual std::ostream & **print** (std::ostream &os) const
- virtual std::vector< T > **getTopFifty** ()=0
- virtual int **getTerms** () const =0

Friends

- std::ostream & **operator**<< (std::ostream &os, const IndexInterface< T > &ii)

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

- indexinterface.hpp

4.9 input Class Reference

Public Member Functions

- input ()

  Prompts the user for an input Query for searching.
- int getFlag ()

  Returns the flag that determines if words to be and/or together.
- vector< string > getTermVector ()

  Returns the vector of terms to be searched for.
- vector< string > getNotTermVector ()

  Returns the vector of terms to be notted with search results.

Public Attributes

- int **andOrFlag** = 0

4.9.1 Member Function Documentation

4.9.1.1 getFlag()

```
int input::getFlag ( )
```

Returns the flag that determines if words to be and/or together.

Returns

the andOrFlag used to determine if words are and-ed or or-ed together

4.9.1.2 getNotTermVector()

```
vector< string > input::getNotTermVector ( )
```

Returns the vector of terms to be notted with search results.

Returns

the vector of notWords

4.9.1.3 getTermVector()

```
vector< string > input::getTermVector ( )
```

Returns the vector of terms to be searched for.

Returns

the vector of andOrWords

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

- input.h
- input.cpp

4.10 runQuery Class Reference

Public Member Functions

- runQuery ()
- runQuery (string)

  Searches index for and returns results of Query.
- pair< string, string > delimit (string s)

  splits bracketed phrases into indivudal strings
- IndexedTerm bracketLogic (IndexedTerm, IndexedTerm)

  Handles logic for terms which are bracketed together by and-ing the two indivudal terms.
- void andLogic ()

  Handles logic to And terms together by multiplying the score of terms and-ed together.
- void **orLogic** ()
- void notLogic ()

  Handles logic to not terms together by modifying the score of notted terms to be very small.

4.10.1 Constructor & Destructor Documentation

4.10.1.1 runQuery() [1/2]

```
runQuery::runQuery ( )
```

Default constructor

4.10.1.2 runQuery() [2/2]

```
runQuery::runQuery (
            string indexType )
```

Searches index for and returns results of Query.

Parameters

the	name of the type of index to form

4.10.2 Member Function Documentation

4.10.2.1 andLogic()

```
void runQuery::andLogic ( )
```

Handles logic to And terms together by multiplying the score of terms and-ed together.

Handles logic to Or terms together by adding the scores of terms or-ed together

4.10.2.2 bracketLogic()

```
IndexedTerm runQuery::bracketLogic (
            IndexedTerm t1,
            IndexedTerm t2 )
```

Handles logic for terms which are bracketed together by and-ing the two indivudal terms.

Parameters

the	first IndexedTerm of the bracketed phrase
the	second IndexedTerm of the bracketed phrase

Returns

an Indexed phrase holding all the questionIDs the terms share with inflated frequencies

4.10.2.3 delimit()

```
pair< string, string > runQuery::delimit (
            string s )
```

splits bracketed phrases into indivudal strings

Parameters

the	string to split into two strings

Returns

a pair of strings containing each term

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

- runquery.h
- runquery.cpp

4.11 SearchEngine Class Reference

Public Member Functions

- void **run** ()

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

- searchengine.h
- searchengine.cpp

Chapter 5

File Documentation

5.1 porter2_stemmer.cpp File Reference

```
#include <algorithm>
#include <utility>
#include <unordered_map>
#include "porter2_stemmer.h"
```

5.1.1 Detailed Description

Author

Sean Massung

Date

September 2012

Implementation of http://snowball.tartarus.org/algorithms/english/stemmer.html

5.2 porter2_stemmer.h File Reference

```
#include <vector>
#include <string>
#include "util/string_view.h"
```

Functions

- void **Porter2Stemmer::stem** (std::string &word)
- void **Porter2Stemmer::trim** (std::string &word)
- size_t **Porter2Stemmer::internal::firstNonVowelAfterVowel** (const std::string &word, size_t start)
- size_t **Porter2Stemmer::internal::getStartR1** (const std::string &word)
- size_t **Porter2Stemmer::internal::getStartR2** (const std::string &word, size_t startR1)
- void **Porter2Stemmer::internal::changeY** (std::string &word)
- void **Porter2Stemmer::internal::step0** (std::string &word)
- bool **Porter2Stemmer::internal::step1A** (std::string &word)
- void **Porter2Stemmer::internal::step1B** (std::string &word, size_t startR1)
- void **Porter2Stemmer::internal::step1C** (std::string &word)
- void **Porter2Stemmer::internal::step2** (std::string &word, size_t startR1)
- void **Porter2Stemmer::internal::step3** (std::string &word, size_t startR1, size_t startR2)
- void **Porter2Stemmer::internal::step4** (std::string &word, size_t startR2)
- void **Porter2Stemmer::internal::step5** (std::string &word, size_t startR1, size_t startR2)
- bool **Porter2Stemmer::internal::isShort** (const std::string &word)
- bool **Porter2Stemmer::internal::special** (std::string &word)
- bool **Porter2Stemmer::internal::isVowel** (char ch)
- bool **Porter2Stemmer::internal::isVowelY** (char ch)
- bool **Porter2Stemmer::internal::endsWith** (meta::util::string_view word, meta::util::string_view str)
- bool **Porter2Stemmer::internal::endsInDouble** (const std::string &word)
- bool **Porter2Stemmer::internal::replaceIfExists** (std::string &word, meta::util::string_view suffix, meta::util::string_view replacement, size_t start)
- bool **Porter2Stemmer::internal::isValidLIEnding** (char ch)
- bool **Porter2Stemmer::internal::containsVowel** (const std::string &word, size_t start, size_t end)

5.2.1 Detailed Description

Author

Sean Massung

Date

September 2012

Implementation of http://snowball.tartarus.org/algorithms/english/stemmer.html

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, IN↩ CLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.