Class Documentation

By

Sam W.

Michael W.

**Classes**:

**Hashtable:**

**Definition:** A data structure that takes keys, puts them through a hash function and then stores them into an array of buckets from which the desired value can be found easily.

**Member Types:**

const long TABLESIZE = The size of the hash table

std::array<std::vector<T>, TABLESIZE> table = An array of vectors of size TABLESIZE which stores the data

int totalAdded = total amount of words stored in the hashtable

**Member Functions:**

Constructor = Creates a HashTable object (public member function)

HashTable(); = Default Constructor

1. Constructs a HashTable object, sets totalAdded to 0 and fills the array with vectors

HashValue = Takes string and runs it through a formula to convert it to an integer

Parameters: The string to be converted to an integer

Return Value: An integer representing the string passed in

Example:

int hash = HashValue(“hash”);

totalEntries = Returns totalAdded

Parameters: none

Return Value: An integer that holds the amount of entries in the hashtable

Example:

int total = totalEntries();

add = Pushes the value passed in back and increments totalAdded

Parameters: A templated value to be pushed back and a string to be hashed for the index location

Return Value: none

Example:

add(“tag”, “tag”);

find = Finds the word object representing the passed in string within the hashtable

Parameters: The string being searched for

Return Value: A pointer to the found element’s location

Example:

find(“example”);

**IndexInterface:**

**Definition:** The IndexInterface provides an interface of methods where the hashtable and avl tree could both work and overlap each other.

**Member Types:** none

**Member Functions:**

Constructor = Creates IndexInterface objects (public member function)

IndexInterface(); = Default Constructor

1. Creates an IndexInterface object

add = Pushes the value passed in back

Parameters: A templated value to be pushed back and a string to be hashed for the index location

Return Value: none

Example:

add(“tag”, “tag”);

find = Finds the word object representing the passed in string within the hashtable

Parameters: The string being searched for

Return Value: A pointer to the found element’s location

Example:

find(“example”);

**Indexer:**

**Definition:** An Indexer object receives input from a parser object and pushes back unique words or updates their frequency count.

**Member Types:**

HashTable<word>\* index = The Hashtable being used to store the words

bool qEntered = Is there an entry in the hashtable?

int totalWords = The total number of unique words in the hashtable

**Member Functions:**

Constructor = Creates an Indexer object (public member function)

Indexer(); = Default Constructor

Indexer(HashTable<word>\* i); = Overloaded Constructor

1. Creates an Indexer object, allocates a hashtable and sets qEntered to false
2. Creates an Indexer object, sets index pointing to i and sets qEntered to false

getTotalWords = Returns totalWords

Parameters: none

Return Value: An integer representing the total unique words

Example:

int total = getTotalWords();

readNewWord = If there are no other occurrences of a word, add it to the hashtable

Parameters: An integer representing the ID of the document and a string representing the word to be added

Return Value: none

Example:

readNewWord(1000, “example”);

**Page:**

**Definition:** Pages are objects that hold data concerning the question files and allow you to retrieve it. They store the Id as an integer, the owner Id and score as a float and the title, body and code as a string.

**Member Types:**

int Id = the Id number of the question

float ownerId = the Id number of the owner of the question

float score = the score the question received

string title = the title of the question

string body = the body of the question

string code = the code of the question

**Member Functions:**

Constructor = Constructs a Page object (public member function)

Page(); = Default Constructor

Page(int, string, string, string); = Overloaded Constructor

1. Constructs a Page object without assigning variables
2. Constructs a Page object and initializes the Id, title, body and code

**Parser**:

**Definition**: Parsers are objects that parse CSV files and return parsed values. The Parser class provides a way to efficiently retrieve data from CSV files in order to store and organize the data into a usable form. It contains two different ways to read the CSV files according to the type of file they are.

**Member types:**

int rowCount = the count of the rows starting at -1 on the header row of the CSV file

int colCount = the count of the columns starting at -1 on the header row of the CSV file

const int idCol = the number that corresponds to the ID column in the CSV files

const int ownerIdCol = the number that corresponds to the Owner ID column in the CSV files

const int scoreCol = the number that corresponds to the Score column in the CSV files

const int titleCol = the number that corresponds to the Title column in the CSV files

const int bodyCol = the number that corresponds to the Body column in the CSV files

const int codeCol = the number that corresponds to the Code column in the CSV files

vector<Page> rows = the vector (or hash table or AVL tree) that stores the Page objects that contain the data from the CSV files

vector<Tag> tags = the vector (or hash table or AVL tree) that stores the Tag objects that contain the data from the CSV files

vector<int> idLocations= the vector that stores the integer associated with an ID’s location

**Member Functions:**

Constructor = Constructs Parser object (public member function)

Parser(); = default constructor

1. Constructs a Parser that is able to parse and return data from a CSV file

Parameters: None

readTagFile = Accepts a file, reads it row by row and stores the data to a vector

Parameters: character pointer to the file name

Return Value: none

Example:

Parser p;

p.readTagFile(“tag.csv”);

readFile = Accepts a file, reads it row by row and stores the data to a vector

Parameters: character pointer to the file name

Return Value: An integer to allow for exception handling

Example:

Parser p;

p.readFile(“question.csv”);

csv\_read\_row = Reads a row of the CSV file and returns a vector of strings

Parameters: string reference to the line and a character to be used for delimitation or an

input stringstream reference and a character to used for delimitation

Return Value: A vector of strings containing the line

Example:

char\* file = “tags.csv”;

std::ifstream in(file);

std::vector<std::string> row = csv\_read\_row(in, ‘,’);

parseBodyWords = Parses body column, removes unwanted characters, changes to lower case and saves data to a vector

Parameters: string of the line and integer holding the ID number

Return Value: none

Example:

parseBodyWords(tempBody, endID);

findFile = Finds and returns file location based on index in idLocations vector

Parameters: integer holding the ID of the file to be found

Return Value: An integer representing the location within the idLocations vector

Example:

int position = 0;

position = findFile(2465);

TotalQuestions = returns the total number of questions

Parameters: none

Return Value: An integer representing the size of the idLocations vector

Example:

int total = 0;

total = TotalQuestions();

readTag = returns the tag contained at the index of the tags vector passed into the method

Parameters: An integer representing the index of the tags vector to be accessed

Return Value: A string containing the tag contained at the index

Example:

string s;

s = readTag(52);

readCode = returns the code contained at the index of the rows vector passed into the method

Parameters: An integer representing the index of the rows vector to be accessed

Return Value: A string containing the code contained at the index

Example:

string s;

s = readCode(52);

readBody = returns the body contained at the index of the rows vector passed into the method

Parameters: An integer representing the index of the rows vector to be accessed

Return Value: A string containing the body contained at the index

Example:

string s;

s = readBody(52);

readTitle = returns the title contained at the index of the rows vector passed into the method

Parameters: An integer representing the index of the rows vector to be accessed

Return Value: A string containing the title contained at the index

Example:

string s;

s = readTitle(52);

readTagId = returns the tag ID contained at the index of the tags vector passed into the method

Parameters: An integer representing the index of the tags vector to be accessed

Return Value: An integer containing the tag ID contained at the index

Example:

int I = 0;

I = readTagId(52);

readScore = returns the score contained at the index of the rows vector passed into the method

Parameters: An integer representing the index of the rows vector to be accessed

Return Value: An integer containing the score contained at the index

Example:

int I = 0;

I = readScore(52);

readId = returns the ID contained at the index of the rows vector passed into the method

Parameters: An integer representing the index of the rows vector to be accessed

Return Value: An integer containing the Id contained at the index

Example:

int I = 0;

I = readId(52);

**Query:**

**Definition:** A query is an object that receives a string input from the user and processes the string into a vector to prepare for searching.

**Member Types:**

std::vector<qWord> search = The vector containing the query from the user split into words

**Member Functions:**

Constructor = Creates a query object (public member function)

query(); = Default Constructor

1. Creates a query object without affecting any variables

getSearch = Returns the vector containing the query split into words

Parameters: none

Return Value: A vector containing the query split into words

Example:

getSearch();

processSearch = Takes input from the user, makes every character lowercase and loads separated strings into the search vector

Parameters: A string representing the query inputted from the user

Return Value: none

Example:

processSearch(“java AND c++ index”);

**qWord:**

**Definition:** A qWord is a query word object that stores a phrase as a string with three Boolean variables signifying conditions.

**Member Types:**

string word = The phrase that the qWord represents

bool a = If AND came before the word in the query

bool o = If OR came before the word in the query

bool n = If NOT came before the word in the query

**Member Functions:**

Constructor = Creates a qWord object (public member function)

qWord(); = Default Constructor

qWord(string, bool, bool, bool); = Overloaded Constructor

1. Creates a qWord object without setting variables
2. Creates a qWord object and sets variables equal to parameters

setWord = Sets current word to string value passed in

Parameters: A string that word will be set to

Return Value: none

Example:

setWord(“ex”);

getWord = Returns current word

Parameters: none

Return Value: A string that represents word

Example:

string s = getWord();

setAnd = Sets current variable to bool value passed in

Parameters: A bool value that ‘a’ will be set to

Return Value: none

Example:

setAnd(true);

getAnd = Returns current value of ‘a’

Parameters: none

Return Value: A bool value that represents ‘a’

Example:

bool s = getAnd();

setOr = Sets current variable to bool value passed in

Parameters: A bool value that ‘o’ will be set to

Return Value: none

Example:

setOr(true);

getOr = Returns current value of ‘o’

Parameters: none

Return Value: A bool value that represents ‘o’

Example:

bool s = getOr();

setNot = Sets current variable to bool value passed in

Parameters: A bool value that ‘n’ will be set to

Return Value: none

Example:

setNot(true);

getNot = Returns current value of ‘n’

Parameters: none

Return Value: A bool value that represents ‘n’

Example:

bool s = getNot();

**Tag:**

**Definition:** Tags are objects that hold data concerning the tag files and allow you to retrieve it. They store the ID number of the tag as an int and the tag itself as a string.

**Member Types:**

float tagId = the ID number of the tag

string phrase = the phrase that represents the tag

**Member Functions:**

Constructor = Constructs Tag object (public member function)

Tag(); = Default Constructor

Tag(float I, std::string p); = Overloaded Constructor

1. Constructs a Tag object with no data initialized
2. Constructs a Tag object with each of the data members initialized to a value

setTag = sets the phrase of the Tag object to the value passed in

Parameters: A string containing the value phrase should be set to

Return Value: none

Example:

Tag p;

p.setTag(“Yay!”);

setTagId = sets the Id of the Tag object to the value passed in

Parameters: A float containing the value tagId should be set to

Return Value: none

Example:

Tag t;

t.setTagId(100);

**Word:**

**Definition:** Words are objects that hold data concerning the frequency of a word within a document.

**Member Types:**

vector<int> idList = the list of Id’s of the documents where the word appears

vector<int> freqList = the list of the number of times the word occurs in a document

string wordValue = the word that the object is concerning

**Member Functions:**

Constructor = Constructs a word object (public member function)

word(); = Default Constructor

word(string input, int inputID); = Overloaded Constructor

1. Constructs a word object with no assignment of variables
2. Constructs a word object with wordValue and one value in idList being assigned

updateFreq = increments the index where the passed value is found or pushes it back if the value is not found

Parameters: An integer representing the ID number to look for

Return Value: none

Example:

updateFreq(251435);

swap = swaps two values memory locations

Parameters: Two integer pointers that point to the two elements to be swapped

Return Value: none

Example:

int a, b = 0;

swap(&a, &b);

partition = breaks up an array into two partitions

Parameters: Two integers one containing the lowest index value and the other containing the highest index value to be sorted

Return Value: An integer representing the partitioning index or the index which separates the partitions

Example:

partition(0, 10);

qSort = sorts by switching values around a pivot

Parameters: Two integers one containing the lowest index value and the other containing the highest index value to be sorted or none

Return Value: none

Example:

qSort();