GEBGLE API

Well, the handout says to use Doxygen, and that would make life much easier IF I knew how and had the patience to learn. Instead, my documentation will be in this pdf. It may not be Doxygen, but it is documentation.

Classes

- avlNode
- avlTree
- docParser
- hashTable
- indexHandler
- queryProcessor
- Searcher

avlNode

- Description: doubly-templated avlNode class for use in an AVL Tree
- *Data members:*
    - `K element`
        - Data payload 1.
    - `V extra`
        - Data payload 2.
    - `avlNode<K, V> *left`
        - Pointer to left child of node.

        - `avlNode<K, V> *right`
            - Pointer to right child of node.
        - `int height`
            - Integer that represents the height of the node. Used for balancing.
- *Constructors:*
    - `avlNode<K, V>(const K &theElement, const V &theExtra, avlNode *lt, avlNode *rt, int h = 0)`
        - Standard constructor with memberwise initialization.
    - `avlNode<K, V>(const K &theElement, avlNode *lt, avlNode *rt)`
        - Standard constructor with normal initialization.
- *Private Member Functions:*
    - *none*
- *Public Member Functions:*
    - `static avlNode<K, V> *copyNodes(avlNode<K, V> *&t)`
        - Function that recursively copies a node and its children.

avlTree

- Description: doubly-templated avlTree class.
- *Data members:*
    - `avlNode<K, V> *root`
        - Root of the tree. Used to traverse, copy, and search tree.

    - `int nodes`
        - Counter that tracks the number of nodes in the tree.
    - *Constructors:*
        - `avlTree() : root(nullptr) {}`
            - Default constructor that sets root to nullptr.
    - `avlNode<K, V>(const K &theElement, const V &theExtra, avlNode *lt, avlNode *rt, int h = 0)`
        - Standard constructor with memberwise initialization.
    - `avlNode<K, V>(const K &theElement, avlNode *lt, avlNode *rt)`
        - Standard constructor with normal initialization.
- *Private Member Functions:*
    - `V &insert(const K &x, avlNode<K, V> *&t)`
        - Private recursive insert function. Accepts a K x and an avlNode<K, V> pointer t. If the node is null, a new avlNode is created with x as the K and returns the V. Compares x to the K of t, determines which side of tree to traverse.
- *Public Member Functions:*
    - `avlTree& operator = (const avlTree<K,V> *&D )`
        - Overloaded assignment operator. Accepts an avlTree<K, V> pointer D by references and uses copyNodes to set root equal to D.
    - `inline ~avlTree()`
        - Inlined avlTree destructor. Calls makeEmpty on root.

- `inline int height(avlNode<K, V> *v)`
    - Inlined retrieval of private height variable. Accepts an avlNode<K,V> pointer v and returns v's height.
- `inline void clear()`
    - Inlined clear function. Calls makeEmpty on root.
- `inline void makeEmpty(avlNode<K, V> *&breaker)`
    - Inlined makeEmpty accepts an avlNode<K,V> pointer breaker by reference and performs a post-order transversal of the tree. The "visit" of each node is deleting breaker.
- `inline V &insert(const K &x)`
    - Inlined insert function accepts a constant K x by reference. Increments nodes variable, then returns the value of private insert insert(x, root).
- `inline void balance(avlNode<K, V> *&t)`
    - Inlined balance function accepts an avlNode<K,V> pointer t by reference. Function compares the heights of t's left and right children, and those node's children. Performs the appropriate rotation to balance the tree.
- `inline void rotateWithLeftChild(avlNode<K, V> *&k2)`
    - Inlined rotation to handle a Case 1 imbalance. Accepts an avlNode<K,V> pointer k2 by reference and rotates it with its left child.

- `inline void rotateWithRightChild(avlNode<K, V> *&k3)`
    - Inlined rotation to handle a Case 4 imbalance. Accepts an avlNode<K,V> pointer k3 by reference and rotates it with its right child.
- `inline void doubleWithLeftChild(avlNode<K, V> *&k3)`
    - Inlined rotation to handle a Case 2 imbalance. Accepts an avlNode<K,V> pointer k3 by reference and rotates it with the rightchild of k3's left child. K3 then rotated with its left child.
- `inline void doubleWithRightChild(avlNode<K, V> *&k3)`
    - Inlined rotation to handle a Case 3 imbalance. Accepts an avlNode<K,V> pointer k3 by reference and rotates it with the left child of k3's right child. K3 then rotated with its right child.
- `void levelOrderTraversal(std::fstream &save)`
    - Level order traversal of tree. Visit prints the node to fstream save.

docParser

- Description: document parsing class of search engine. Handles reading in files, stemming, stop word removal, and accessing/mutating data associated with a word.
- *Data members:*

    - `indexHandler dictionary;`
        - Index handler that holds the stemmed, stopped, and generally parsed words. See `indexHandler`.
    - `std::unordered_set<string> stops;`
        - Unordered set of strings read in from https://www.webconfs.com/stop-words.php. Used to compare words to be parsed and removed.
    - `int wordCounter, orgCounter, perCounter`
        - Integer counters that keep track of the number of respective items in dictionary.
- *Constructors:*
    - `docParser()`
        - Standard constructor with no initialization of `dictionary`.
    - `docParser(indexHandler& source)`
        - Standard constructor that accepts an indexHandler source as an argument by reference. Source is copied to dictionary.
- *Private Member Functions:*
    - *none*
- *Public Member Functions:*
    - `inline void Parse(string& toParse)`
        - Inlined parse that accepts a string toParse by reference. toParse is a path to files, function then uses Porter2Stemmer and stops unordered_set to parse and remove stop words. Adds words,

persons, and organizations to their corresponding containers in dictionary.

- `unordered_set<string> search(const string& query)`
  - Searches word dictionary for query parameter, string passed by reference. Returns the unordered set of docs that query is in.
- `vector<string> searchPersons(const string& query)`
  - Searches person dictionary for query parameter, string passed by reference. Returns the vector of docs that query is in.
- `vector<string> searchOrgs(const string& query)`
  - Searches organizations dictionary for query parameter, string passed by reference. Returns the vector of docs that query is in.
- `void printLevels(fstream &save)`
  - Accepts a fstream save by reference. Writes the current state of the dictionary to save.
- `void printFileData(string &x)`
  - Accepts a string x by reference that is that direct path to a file. Prints the title, author, publishing date, and URL of JSON file.

hashTable

- Description: doubly-templated hashTable class for use in storing organizations and persons.
- *Data members:*
  - `vector<V> table[100009]`

        - Array of vectors of type V to be used to store hashed values.
    - *Constructors:*
    - `hashTable<K, V>()`
        - Standard constructor.
    - `hashTable& operator = (const hashTable<K,V> *&D )`
        - Overloaded assignment operator that accepts a hashTable<K,V> pointer D by reference. Copies D's table into this table.
- *Private Member Functions:*
    - *none*
- *Public Member Functions:*
    - `vector<V>& at(const K& key)`
        - Function that utilizes std::hash to hash the K parameter key, given by reference. Mods the resulting value with the size of table and creates a vector<V> to place at the corresponding location.
    - `vector<V>& add(const K& key, const V& value)`
        - Function that utilizes std::hash to hash the K parameter key, given by reference. Mods the resulting value with the size of table and creates a vector<V> to place at the corresponding location. Uses emplace_back to add V parameter value to vector.

indexHandler

- Description: class that handles storing words, persons, organizations and documents. Wrapped by the docParser class.

- *Data members:*
    - `avlTree<string, unordered_set<string>> index`
        - AVL tree of strings and unordered_sets of strings. Used in creating the actual inverted file index. String is the word, and the unordered_set is of the documents that the word appears in.
    - `hashTable<string, string> people`
        - Hash table of strings and string. So every string (“person”) maps to a vector of strings (“documents”).
    - `hashTable<string, string> orgs`
        - Hash table of strings and string. So every string (“person”) maps to a vector of strings (“documents”).
    - *Constructors:*
        - `indexHandler()`
            - Default constructor that does not initialize index, people, or orgs.
    - `indexHandler(avlTree<string, unordered_set<string>> &source)`
        - Standard constructor that accepts a preexisting AVL tree by reference. Copies source into index.
    - *Private Member Functions:*
        - *none*
    - *Public Member Functions:*
        - `void operator = (const indexHandler &copy )`

        - Overloaded assignment operator. Accepts an indexHandler copy by reference and copies it into index.
- `void clear()`
    - Clear function that deletes the contents of index by calling its destructor and then creating a new AVL tree.
- `inline int height(avlNode<K, V> *v)`
    - Inlined retrieval of private height variable. Accepts an avlNode<K,V> pointer v and returns v’s height.
- `unordered_set<string> &addWordOrDoc(const string &newEntry, const string &doc)`
    - Add word function. Accepts a new string newEntry by reference and a string doc by reference. Inserts newEntry into index and adds doc to the unordered_set in the node.
- `unordered_set<string> &getDocs(const string &word);`
    - Accepts a string word by reference and return the corresponding unordered_set found in the AVL tree.
- `vector<string> &addOrgOrDoc(const string &newEntry, const string &doc)`
    - Add word function. Accepts a new string neworg by reference and a string doc by reference. Inserts neworg into orgs and adds doc to the vector in the hash map.
- `vector<string> &getDocsOrg(const string &word);`

        - Accepts a string word by reference and return the corresponding vector in the hashMap.
    - `vector<string> &addPersOrDoc(const string &newPers, const string &doc)`
        - Add word function. Accepts a new string newPers by reference and a string doc by reference. Inserts neworg into orgs and adds doc to the vector in the hash map.
    - `vector<string> &getDocsPers(const string &word);`
        - Accepts a string word by reference and return the corresponding vector in the hashMap.
    - `void saveIndex(fstream &save)`
        - Accepts a fstream by reference and then writes the contents of index to it. Creation of persistence index.

queryProcessor

- Description: class that handles receiving queries from the user by implementing a simple Boolean prefixed query search.
- *Data members:*
    - *None*
- *Constructors:*
        - `queryProcessor()`
            - Default constructor.
    - *Private Member Functions:*

        - *none*
    - *Public Member Functions:*
        - `unordered_set<string> returnQuery(docParser &index, string &query)`
            - Function for searching through a docParser index, given by reference. Query is a string given by reference, that is the word the user is asking for. This function handles the boolean operations, as well as attempts to handle the NOT, PERSON, and ORG calls. Also sets up user to read the full text of a document.
        - `void showFullText(char* filePath)`
            - Function that accepts a char* filePath as an argument and finds the JSON files that correspond. Then prints the full body text of the file at the filePath.
        - `char* word_wrap (char* buffer, char* string, int line_width)`
            - Simple function that uses a buffer to put some newlines into showFullText for user readability.

<u>**searcher**</u>

- Description: Overarching class that is the driver of the program.
- *Data members:*
    - `docParser back`

        - docParser that the searcher uses to complete queries, access files, and things of that nature.
    - `queryProcessor searchBar`
        - queryProcessor that searcher uses to analyze queries and fulfill them.
- *Constructors:*
        - `searcher(docParser &in, queryProcessor &en);`
            - Constructor that accepts a docParser in and a queryProcessor &en by reference. Copies these members into back and searchBar respectively.
    - *Private Member Functions:*
        - *none*
    - *Public Member Functions:*
        - `void run(char* dataPath)`
            - Driver for the whole program. Accepts a char* dataPath, where the large dataset is. Sets up the user interface, accepts queries, and uses back and searchBar as necessary. Exits when the user enters a certain character, and then program terminates.