Ahmad Mohammad

c1001633 / asm6t

proj5

		deduction
TURNIN TIME	turned in on time.	0 %
SOURCE CODE SEARCH RESULTS	MISSING: None. FOUND: class WordTree, WORDTREE_H, addWord, deleteSubTree, getCounts, findWord, WordTree(), ~WordTree(), input.txt, queries.txt	0

DEDUCTIONS	-7
FINAL GRADE	93

COMPILATION LOG

```
c1001633 MAKE PASS

MAKELOG OF STUDENT PROVIDED MAKEFILE:
g++ -c -g -O0 -std=c++11 -Wall wordtree.cpp -o wordtree.o -MMD -MF wordtree.d
g++ -c -g -O0 -std=c++11 -Wall proj5.cpp -o proj5.o -MMD -MF proj5.d
g++  -o proj5 wordtree.o proj5.o
```

c1001633 runlog begin:
Inputs:
Project description The program should open and read input file named input.txt in t
urn building up a binary search tree of words and counts as it progresses The words
 will be stored in alphabetical order in the tree The program should ignore the case o
f the words so that Branch and branch are considered the same However words that
 are actually spelled differently such as tree and trees are considered to be different
and should be stored in separate nodes All words will be stored in the tree in their lo
wercase form Two forms of queries are to be supported by the tree class A query for
 an individual word should return the number of occurrences of that word in the inp
ut file as retrieved from searching the tree or should display a message stating that t
he word was not found in the tree A query for words that meet or exceed a threshold
 number of occurrences should result in the output of the words and their counts tha
t meet that criteria using inorder tree traversal The nodes for the word tree should b

e a struct with the following members One pointer each to the left and right subtree s An int containing the count of the number of occurrences of the word A string cont aining the word itself Requirements Your program must be split into files There will b e a class with separate interface and implementation files and a driver file The requi rements for these are specified below The WordTree class This class represents a Wo rd Binary Tree Files must be named WordTree.h and WordTree.cpp Class must be na med WordTree You should implement the following member functions in the class A c onstructor Creates an empty tree A destructor Recursive function that explicitly rele ases all nodes allocated during program execution You may not rely on program ter mination to release memory insert Recursive function that adds a word to the tree if i t is not found or increments its count if it is already in the tree findNode accepts a st ring argument a word and searches for the word in the tree If found outputs the wor d and its count Otherwise displays a message stating that the word was not found pr intInOrder Recursive function that accepts a single integer argument a threshold val ue and traverses the tree in order outputting the words and their counts that meet or exceed the threshold count The function should also output the number of nodes me eting the criteria A driver or client file that Must be named proj6.cpp Declares the w ord tree object Opens and reads the text file named input.txt and builds the word tre e from the file by invoking the insert function described above Invokes queries for in dividual words or for words whose counts meet or exceed a threshold number of occ urrences

Queries:
C 15
F class
F recursion

Test 1:
Output:
Word Tree built and loaded

Finding all words with 15 or more occurence(s).
a(20)
and(16)
the(45)
tree(17)
4 nodes had words with 15 or more occurence(s).

Searching for occurence(s) of the word 'class'
The word class occurs 6 time(s) in the text.

Searching for occurence(s) of the word 'recursion'
The word recursion was not found in the text.

Test 2:
Inputs:
Project description in this assignment you will modify the ordered singly linked list y ou developed for project two to store custom objects The list will hold objects repres enting rectangles and store them in ascending area order The driver program instan tiates a list and applies operations read from the input file Requirements Your progr am be split into four files the requirements for which are specified below The Rectan gle class This class represents an individual rectangle Files be named RECTANGLE p rovided and details each has length width and area data members The Null Rectangl e one with all data members set to zero is a special rectangle indicating a null value This object is created by the default constructor Functions for the relational operato

rs not defined the header file be implemented file The class template must be fully implemented in a file named it must follow the header file for project two except for name changes and its implementation as a class template so that the list is able to store any custom object not just ints You may not depart from those specifications ask if doubt Note the following Must have all functions and data members specified in the Project two header file but these must be modified to handle custom objects Must use the following struct replacing the one project two struct info next null info Must have the implementation for functions required by the project two header file Observe the following insert in Order Adds each custom object to the list ascending area order If there are objects the list with the same value the newly object must come after the one already the list RECTANGLE example If custom object rectangle in this case has length three and width four area twelve is in the list and a rectangle with length two and width six also having area twelve is added the rectangle with length two and with six must come after see sample output delete This function removes the custom object that has the same value as the argument
Queries:
C 8
F list
F data
F code
F example
F projects
Output:
Word Tree built and loaded

Finding all words with 8 or more occurence(s).
and(10)
list(8)
must(8)
rectangle(9)
the(31)
5 nodes had words with 8 or more occurence(s).

Searching for occurence(s) of the word 'list'
The word list occurs 8 time(s) in the text.

Searching for occurence(s) of the word 'data'
The word data occurs 3 time(s) in the text.

Searching for occurence(s) of the word 'code'
The word code was not found in the text.

Searching for occurence(s) of the word 'example'
The word example occurs 1 time(s) in the text.

Searching for occurence(s) of the word 'projects'
The word projects was not found in the text.

```
Manifest: proj5.cpp.lst
-----------------------
-----------------------
 Apr 04 11:56 proj5.cpp
```

```
    1   //Ahmad MOhammad
    2   //CSCI 3110-001
    3   // Proj 5
    4   //Due : 04/04/22
    5   //Desc: The program will open and read an input file (named input.txt), and build a bi
nary search tree of
    6   // the words and their counts. The words will be stored in alphabetical order in the t
ree. It will also open a
    7   // queries file and perform ops on the tree based on the data provided by query.
    8
    9   #include<iostream>
   10   #include<fstream>
   11   #include<string>
   12   #include<cctype>
   13
   14   #include "wordtree.h"
   15
   16
   17   using namespace std;
   18
   19
   20   int main()
   21   {
   22                   // declaring variables
   23                   ifstream infile, infile2;
   24                   infile.open("input.txt");
   25                   string x;
   26                   int n;
   27                   string word;
   28
   29                   // instantiation of class
   30                   WordTree z;
   31       cout << "Word Tree built and loaded" <<endl << endl;
   32
   33                   // while input file still has words add them each to a tree
   34                   while(infile>>x)
   35                   {
   36                                   z.addWord(x);
   37                   }
   38
   39                   // open second queries file
   40                   infile2.open("queries.txt");
   41
   42                   // while file still has words/letters left check and see which operatio
n to perform
   43                   while(infile2>>x)
   44                   {
   45                                   if(x == "F")
   46                                   {
   47                                                   infile2>> word;
   48                                                   cout << "Searching for occurence(s) of th
e word '"<<word<<"'" << endl;
   49                                                   z.findWord(word);
   50                                   }
   51                                   else
   52                                   {
   53                                                   infile2>>n;
   54                                                   cout << "Finding all words with " << n <<
 " or more occurence(s)." << endl;
   55                                                   z.getCounts(n);
   56                                   }
```

```
   57
   58   }
   59                   return 0;
   60   }
---------------------- wordtree.h: ------------------------
     1  #ifndef WORDTREE_H
     2  #define WORDTREE_H
     3
     4  #include<iostream>
     5  #include<string>
     6
     7  class WordTree
     8  {
     9  private:
    10     struct TreeNode
    11     {
    12        std::string value;        // The value in the node
    13        TreeNode *left;                  // Pointer to left child node
    14        TreeNode *right;                 // Pointer to right child node
    15            int count;                           // Instance count of value
    16     };
    17
    18     TreeNode *root;       // Pointer to the root node
    19
    20     // Private member functions - all are recursive
    21     void addWord(TreeNode *&, std::string);                     // reference to pointe
r to node, and word to be added
    22     void deleteSubTree(TreeNode *);                                    // pointer to
node
    23     void getCounts(TreeNode *, int, int&) const;         // pointer to node, int thresh
old,
    24
               // reference to int that accumulates nodes that meet the query
    25
    26
    27  public:
    28     // Constructor
    29     WordTree();
    30
    31     // Destructor - invokes helper function
    32     ~WordTree();
    33
    34     // public functions via which the tree operations are called (hides tree's root/imp
lementation)
    35     void addWord(std::string);
    36     void findWord(std::string);
    37     void getCounts(int);
    38
    39  };
    40  #endif
\---------------------- wordtree.cpp: ------------------------
     1
     2  #include<iostream>
     3  #include<string>
     4  #include<algorithm>
     5  #include "wordtree.h"
     6
     7  using namespace std;
     8
     9  //Constructor sets root to NULL
    10  WordTree::WordTree()
```

```
11  {
12          root = NULL;
13  //      root->left = NULL;
14  //      root->right = NULL;
15  }
16
17  //destructor uses deleteSubtree class to deallocate all nodes
18  WordTree::~WordTree()
19  {
20          deleteSubTree(root);
21  }
22
23  //this function adds a word to the bst recursively
24  void WordTree::addWord(TreeNode *&nodeptr, std::string str)
25  {
26          //transforms str to all lowercase
27          std::transform(str.begin(), str.end(), str.begin(), ::tolower);
28
29          // create new node if word was not founn bst
30          if(nodeptr == NULL)
31                  {
32                          nodeptr = new TreeNode;
33                          nodeptr->value = str;
34                          nodeptr->left = NULL;
35                          nodeptr->right = NULL;
36                          nodeptr->count++;
37                  }
38          // if node was found with same word incrememnet the count
39          else if(nodeptr->value == str && nodeptr != NULL)
40                  {
41                          nodeptr->count++;
42                  }
43          // if cur node has value > str go to the left(childe) node
44          else if(nodeptr->value > str && nodeptr != NULL)
45                  {
46                          addWord(nodeptr->left,str);
47                  }
48          // if cur node has value < str go to right (child) node
49          else if(nodeptr->value < str && nodeptr != NULL)
50          {
51              addWord(nodeptr->right,str);
52          }
53
54  }
55
56  // this is a recursive function that deletes every node in bst
57  void WordTree::deleteSubTree(TreeNode *nodeptr)
58  {
59          // start all the way to the left then return and check rights (inorder traveer
sal)
60          if (nodeptr != NULL)
61          {
62                  deleteSubTree(nodeptr->left);
63                  delete nodeptr;
64                  deleteSubTree(nodeptr->right);
65          }
66  }
67
68  // function that gets words that all have a count higher than the asked number
69  void WordTree::getCounts(TreeNode *nodeptr, int counts, int& accum) const
70  {
```

-2

For a new node, count has not been initialized and may contain a non-zero value

-5

These lines should be swapped- the right subtree cannot be deleted if the node itself has already been deleted

```
    71          // traverse inorder checking
    72          if(nodeptr != NULL)
    73          {
    74                  getCounts(nodeptr->left,counts,accum);
    75                  if(nodeptr->count >= counts)
    76                          {       cout << nodeptr->value<<"("<<nodeptr->count<<")"<<endl
; accum++;       }
    77                  getCounts(nodeptr->right,counts,accum);
    78          }
    79
    80  }
    81
    82  //helper function to protect data
    83  void WordTree::addWord(std::string str)
    84  {
    85          addWord(root, str);
    86  }
    87
    88  void WordTree::findWord(std::string str)
    89  {
    90          //create node ptr to point at root to be used in traversal
    91          TreeNode *cur = root;
    92
    93          while(cur != NULL && cur->value != str )
    94                  {
    95              if(cur->value > str)
    96                  { cur = cur->left;}
    97                  else if(cur->value < str)
    98                                  { cur = cur->right;}
    99                  }
   100          if(cur == NULL)
   101                  {       cout << "The word " << str << " was not found in the text." <<
endl<< endl;    }
   102          else if(cur->value == str && cur != NULL)
   103                  { cout << "The word "<< cur->value<< " occurs "<< cur->count << " time
(s) in the text." <<endl<< endl; }
   104
   105  }
   106
   107  //helper function to protect data
   108  void WordTree::getCounts(int counts)
   109  {
   110
   111          int accum = 0;
   112          //create node ptr to point at root to be used in traversal
   113          TreeNode *cur = root;
   114          getCounts(cur,counts,accum);
   115
   116          cout << accum << " nodes had words with " << counts << " or more occurence(s).
" << endl <<  endl;
   117
   118  }
   119
```