Problem Description: to create a BST for strings implementation. We want to add methods to read, modify, and compare files of text.

Methods: Comments in code

Sample Execution:

In Order: “tdata.txt”

```
ksz12@rabbit:~/ECE218/EXAM3_TH % CC main.cpp
ksz12@rabbit:~/ECE218/EXAM3_TH % ./a.out

AONIAN Above Beginning Brook Brought CHAOS: Death Delight Disobedience, EDEN, Earth Fast First Forbidden Fruit God; Heav'nly Heav'ns Hill I In Invoke Man Mans Mou
nt, Muse, OREB, Of Or Oracle Prose Restore Rhime. Rose SILOA'S SINAI, SION Seat, Seed, Shepherd, Sing Song, That Things Tree, With World, adventrous aid all and b
lissful by chosen didst first flight flow'd greater how if in inspire intends into it loss middle more, mortal my no of on one or our out pursues regain secret so
ar tast taught th' that the thee thence thy till to top unattempted us, while who whose with woe, yet
ksz12@rabbit:~/ECE218/EXAM3_TH %
```

Pre Order: “tdata.txt”

```
ksz12@rabbit:~/ECE218/EXAM3_TH % CC main.cpp
ksz12@rabbit:~/ECE218/EXAM3_TH % ./a.out

Of Mans First Disobedience, Brought Beginning Above AONIAN Brook Death CHAOS: Delight EDEN, Earth Fast Fruit Forbidden Man Heav'nly God; In Heav'ns Hill I Invoke
Muse, Mount, OREB, and Tree, Restore Or Oracle Prose Seat, SINAI, Rose Rhime. SILOA'S SION Sing Shepherd, Seed, That Song, Things World, With all aid adventrous t
he that mortal into greater blissful didst chosen by first flow'd flight inspire how if in intends loss it more, middle tast our of my no one on or regain out pur
sues secret soar taught th' whose till thee thence thy us, top to unattempted who while woe, with yet
ksz12@rabbit:~/ECE218/EXAM3_TH %
```

Post Order: “tdata.txt”

```
ksz12@rabbit:~/ECE218/EXAM3_TH % CC main.cpp
ksz12@rabbit:~/ECE218/EXAM3_TH % ./a.out

AONIAN Above Brook Beginning CHAOS: Delight Death Brought Fast Earth EDEN, Disobedience, Forbidden God; I Hill Heav'ns Invoke In Heav'nly Man Fruit First Mount, O
REB, Muse, Mans Prose Oracle Or Rhime. SILOA'S Rose SION SINAI, Seed, Shepherd, Song, Things That Sing Seat, Restore With adventrous aid all World, Tree, by chose
n flight flow'd first didst blissful in if how intends inspire greater it middle more, loss into no my on or one of pursues out soar secret regain our th' taught
tast mortal that thy thence thee to unattempted top while who us, till with yet woe, whose the and Of
ksz12@rabbit:~/ECE218/EXAM3_TH %
```

All Comparisons: (“tdata.txt” below)

- # of times a word was repeated
- Depth of Tree
- # of repeated words (same value count)
- # of unique nodes added to tree
- Balance of tree based on height and nodes

```
ksz12@rabbit:~/ECE218/EXAM3_TH %
ksz12@rabbit:~/ECE218/EXAM3_TH %
above: 1
adventrous: 1
aid: 1
all: 1
and: 5
aonian: 1
beginning: 1
blissful: 1
brook: 1
brought: 1
[by: 1
 chaos: 1
 chosen: 1
 death: 1
 delight: 1
 didst: 1
 disobedience: 1
 earth: 1
 eden: 1
 fast: 1
 first: 2
 flight: 1
 flowd: 1
 forbidden: 1
 fruit: 1
 god: 1
 greater: 1
 heavnly: 1
 heavns: 1
 hill: 1
 how: 1
 i: 1
 if: 1
 in: 2
 inspire: 1
 intends: 1
 into: 1
 invoke: 1
 it: 1
[loss: 1
 man: 1
 mans: 1
 middle: 1
 more: 1
 mortal: 1
 mount: 1
 muse: 1
 my: 1
 no: 1
 of: 7
 on: 1
 one: 1
 or: 3
 oracle: 1
 oreb: 1
 our: 1
 out: 1
 prose: 1
 pursues: 1
 regain: 1
 restore: 1
 rhime: 1
 rose: 1
 seat: 1
 secret: 1
 seed: 1
 shepherd: 1
 siloas: 1
 sinai: 1
 sing: 1
 sion: 1
 soar: 1
 song: 1
 tast: 1
 taught: 1
 th: 1
 that: 5
 the: 8
 thee: 1
 thence: 1
 things: 1
 thy: 1
 till: 1
 to: 2
 top: 1
 tree: 1
 unattempted: 1
 us: 1
 while: 1
 who: 1
 whose: 1
 with: 2
 woe: 1
 world: 1
 yet: 1

 Depth: 12
 Same value count: 27
 Node Count: 122

 Tree is not balanced

 ksz12@rabbit:~/ECE218/EXAM3_TH %
```

Example output for comparing all files:

```
yields: 3
yoak: 1
yoke: 6
yon: 6
yonder: 10
yonger: 1
you: 238
yould: 1
youll: 3
young: 18
younger: 3
youngest: 1
your: 182
yours: 8
yourself: 4
yourselves: 3
youth: 9
youthful: 3
youths: 1
z: 1
za: 1
zeal: 3
zeale: 7
zealous: 3
zenith: 2
zenos: 1
zephiel: 1
zephir: 1
zephon: 4
zephyr: 1
zephyrus: 1
zeus: 1
zodiac: 6
zone: 4

Depth: 36
Same value count: 195924
Node Count: 210006

Tree is not balanced

ksz12@rabbit:~/ECE218/EXAM3_TH %
```

File created by s*aveTreeToFile*:

```
  GNU nano 4.2
of
mans
first
disobedience
and
all
aid
adventrous
above
brought
blissful
beginning
aonian
brook
death
chosen
chaos
by
didst
delight
eden
earth
fast
fruit
forbidden
flowd
flight
into
greater
god
heavnly
inspire
in
how
heavns
hill
if
i
intends
loss
invoke
it
man
mortal
more
middle
muse
mount
my
no
the
that
tast
our
one
on
oreb
or
oracle
restore
regain
out
pursues
prose
seat
rose
rhime
sing
secret
sinai
shepherd
seed
siloas
sion
song
soar
taught
th
tree
till
thee
thence
thy
things
top
to
whose
us
unattempted
who
while
world
woe
with
yet
```

```
ksz12@rabbit:~/ECE218/EXAM3_TH %
```

Conclusion: I was able to successfully create a BST for strings implementation. I was able to read in files to make modifications and comparisons to then rewrite a file out to store in another file. I could modify my own implementation to keep track of different pieces of data within the string text files. In the end, I had a full functioning BST that I could create methods for to read, modify, and compare files of text.

Code:

Main.cpp:

bst.h:

```
#include <string>
#include <iostream>
#include <ostream>
#include <fstream>


class BST
{
   private:

   struct node{
      std::string key;
      node *left;
      node *right;
   };

   node *root;
   void addLeafPrivate(std::string key, node* Ptr);       //method to add leaf to the tree once accessed the root pointer
   void PrintInOrderPrivate(node* Ptr);       //method to print the BST in order once accessed the root pointer
   void PrintPreOrderPrivate(node* Ptr);       //method to print the BST in pre order once accessed the root pointer
   void PrintPostOrderPrivate(node* Ptr);      //method to print the BST in post order once accessed the root pointer
```

```
    int isBalancedPrivate(node* root);      //method to calculate if the tree is balanced once accessed the
root pointer
    int maxDepth(node* node);       //method to get the depth of the tree once accessed the root pointer
    void printToFile(node* Ptr, std::ofstream &outFile);        //method to print the read in data from file into
the bst and to a out file
  public:
    BST();
    ~BST();
    node *CreateLeaf(std::string key);      //method to create first leaf of the tree when the root is null
    void addLeaf(std::string key);      //constructor to add leaf to the tree of new data(key)
    void PrintInOrder();        //method to print the BST in order
    void PrintPreOrder();       //method to print the BST in pre order
    void PrintPostOrder();      //method to print the BST in post order
    void getDepth();        //method to get the height of the trees
    int isBalanced();       //method to calculate if the tree is balanced
    void saveTreeToFile(std::string fileInput, std::string fileInput1, std::string fileInput2, std::string fileInput3,
std::string fileOutput);     //method to read in files and perform the comaprisons
    void loadTreeFromFile(std::string fin, std::string fout);       //loads the tree from the created file in the
printToFile
};
```

bst.cpp:

```
#include <iostream>
#include <fstream>
#include <string>
#include "bst.h"
#include <vector>
#include <algorithm>

using namespace std;
```

```
int sameValCount =0;
int nodeCount =0;

BST::BST(){
    root = NULL;
}

BST::~BST(){
    delete root;
}

BST::node* BST::CreateLeaf(std::string key){        //method to create first leaf of the tree when the root is
null
    node* n = new node;     //create a new node and set both left and right sides of tree pointers to null
    n -> key = key;
    n -> left = NULL;       //set left to null
    n -> right = NULL;      //set right to null
    return n;}

    void BST::addLeaf(std::string key){     //constructor to add leaf to the tree of new data(key)
        addLeafPrivate(key, root);
    }

    void BST::addLeafPrivate(std::string key, node* Ptr){       //method to add leaf to the tree once
accessed the root pointer
        if(root ==NULL){        //if the root is null then create new node which will be the root
            root = CreateLeaf(key);
        }
        // check if the data is greater or less than that of the previous node and place it in its correct
position(either right or left)
        else if(key<Ptr->key){
            if(Ptr->left !=NULL){
                addLeafPrivate(key, Ptr->left);
```

```
            }
            else {
                Ptr-> left = CreateLeaf(key);
            }
        }
         else if(key>Ptr->key){
            if(Ptr->right !=NULL){
                addLeafPrivate(key, Ptr->right);
            }
            else {
                Ptr-> right = CreateLeaf(key);
            }
        }
        else{       //when a value has already been added to the tree
            sameValCount++;
        }
    }

  void BST:: PrintInOrder(){        //method to print the BST in order
    PrintInOrderPrivate(root);}

  void BST:: PrintInOrderPrivate(node* Ptr){    //method to print the BST in order once accessed the root
pointer
    if(root != NULL){
        if(Ptr->left !=NULL){
            PrintInOrderPrivate(Ptr->left);     //recurrsive call to the left side of the tree to print out all data
        }
        std::cout<<Ptr->key<<" ";
        if(Ptr->right != NULL){
            PrintInOrderPrivate(Ptr->right);        //recurrsive call to print out all value of the right side of the tree
        }
    }
    else{       //if the root is null then the tree is empty... nothing to print
```

```
        std::cout<<"The tree is empty"<<std::endl;
    }
  }


void BST::PrintPreOrder(){      //method to print the BST in pre order
    PrintPreOrderPrivate(root);
}


void BST:: PrintPreOrderPrivate(node* Ptr){     //method to print the BST in pre order once accessed the
root pointer
    if (Ptr == NULL)
        return;
    std:: cout << Ptr->key << " ";
    //recursively traversing left and right side of the tree to print the values
    PrintPreOrderPrivate(Ptr->left);
    PrintPreOrderPrivate(Ptr->right);
}

void BST::PrintPostOrder(){     //method to print the BST in post order
    PrintPostOrderPrivate(root);
}

void BST::PrintPostOrderPrivate(node* Ptr){     //method to print the BST in post order once accessed the
root pointer
    if (Ptr == NULL)
        return;

    PrintPostOrderPrivate(Ptr->left);
    PrintPostOrderPrivate(Ptr->right);
    std::cout << Ptr->key << " ";
}
```

```
void BST::getDepth(){       //method to get the height of the trees
    std::cout<<maxDepth(root);
}

   int BST::maxDepth(node* node)        //method to get the depth of the tree once accessed the root
pointer
{
    if (node == NULL)
        return 0;
    else {
        int leftdepth = maxDepth(node->left);      //get the value of the height of the left side of the tree
        int rightdepth = maxDepth(node->right);     //get the value for the right side of the tree
        if (leftdepth > rightdepth)
            return (leftdepth + 1);
        else
            return (rightdepth + 1);
    }
}

int BST::isBalanced(){      //method to calculate if the tree is balanced
    isBalancedPrivate(root);
return 0;
}

int BST::isBalancedPrivate(node* root){       //method to calculate if the tree is balanced once accessed
the root pointer
    int leftheight =0;
    int rightheight =0;
    if (root == NULL)
        return 1;
    leftheight = maxDepth(root->left);      //calculate the left height of the tree
    rightheight = maxDepth(root->right);    //calculate the right height of the tree
```

```
    if (abs(leftheight - rightheight) <= 1 && isBalancedPrivate(root->left)&& isBalancedPrivate(root->right))
        return 1;       //if the difference between the heights of each side of the tree is greater than one it is
not balanced
    return 0;
}

void BST::saveTreeToFile(std::string fileInput, std::string fileInput1, std::string fileInput2, std::string
fileInput3, std::string fileOutput){        //method to read in files and perform the comaprisons
    BST meta;
    //open the files
    std::vector<std::string> wordsInFile;
    std::string word;
    std::fstream fin;
    fin.open(fileInput);
    std::fstream fin1;
    fin1.open(fileInput1);
    std::fstream fin2;
    fin2.open(fileInput2);
    std::fstream fin3;
    fin3.open(fileInput3);
    //read in the data
    while(fin>>word){
    for (int i = 0, length = word.size(); i < length; i++)        //for loop to remove all the punctuation
    {
     if(word[i]>='A' && word[i]<='Z'){
            word[i]=((char)(word[i]-'A'+'a'));
        }
        if (std::ispunct(word[i]))
        {
            word.erase(i--, 1);
            length = word.size();
        }
```

```
    }
//CODE FOR AMOUNT OF TIMES A WORD APPEARS
//  std::vector<std::string> wordsInFile;
    wordsInFile.push_back(word);           //add the word to the vector to compare

    meta.addLeaf(word);     //add the data to the tree
    nodeCount++;        //increment the amount of nodes added to the tree
}
//read in the second file
   while(fin1>>word){
    for (int i = 0, len = word.size(); i < len; i++)
    {
     if(word[i]>='A' && word[i]<='Z'){
           word[i]=((char)(word[i]-'A'+'a'));
       }
       if (std::ispunct(word[i]))
       {
           word.erase(i--, 1);
           len = word.size();
       }
    }
//CODE FOR AMOUNT OF TIMES A WORD APPEARS
//  std::vector<std::string> wordsInFile;
    wordsInFile.push_back(word);

    meta.addLeaf(word);
    nodeCount++;
}
   while(fin2>>word){
    for (int i = 0, len = word.size(); i < len; i++)
    {
     if(word[i]>='A' && word[i]<='Z'){
           word[i]=((char)(word[i]-'A'+'a'));
```

```
        }
        if (std::ispunct(word[i]))
        {
            word.erase(i--, 1);
            len = word.size();
        }
    }
//CODE FOR AMOUNT OF TIMES A WORD APPEARS
//  std::vector<std::string> wordsInFile;
    wordsInFile.push_back(word);

    meta.addLeaf(word);
    nodeCount++;
}
   while(fin3>>word){
    for (int i = 0, len = word.size(); i < len; i++)
    {
     if(word[i]>='A' && word[i]<='Z'){
            word[i]=((char)(word[i]-'A'+'a'));
        }
        if (std::ispunct(word[i]))
        {
            word.erase(i--, 1);
            len = word.size();
        }
    }
//CODE FOR AMOUNT OF TIMES A WORD APPEARS
//  std::vector<std::string> wordsInFile;
    wordsInFile.push_back(word);

    meta.addLeaf(word);
    nodeCount++;
}
```

```
    //sort the vector to calculate how many times nodes are repeated
    sort(wordsInFile.begin(), wordsInFile.end());
    int vec_size = wordsInFile.size();

    if(vec_size ==0) std::cout<<"No words in file"<<std::endl;

    int wordCount = 1;
    word = wordsInFile[0];
    //loop to print how many times a word is repeated in a file
    for(int i=1; i<vec_size; i++){
       if(word!= wordsInFile[i]){
          std::cout<<word<< ": "<<wordCount<<std::endl;
          wordCount =0;
          word = wordsInFile[i];
          }
       wordCount++;
    }
    std::cout<<word<< ": "<<wordCount<<std::endl;

    fin.close();
    fin1.close();
    fin2.close();
    fin3.close();
    std::ofstream outFile;
    outFile.open(fileOutput);
    meta.printToFile(meta.root, outFile);
    outFile.close();
    std::cout<<std::endl;
    std::cout<<"Depth: ";
    meta.getDepth();
}
```

```
void BST::printToFile(node* Ptr, std::ofstream &outFile){       //method to print the read in data from file
into the bst and to a out file
        if (Ptr == NULL)
            return;
        else {
            outFile << Ptr->key <<std::endl;
            printToFile(Ptr->left, outFile);        //recurrsively traverse the left side of the tree
            printToFile(Ptr->right, outFile);       //recurrsively traverse the right side of the tree
        }
}

void BST::loadTreeFromFile(std::string fileInput, std::string fileOutput){      //loads the tree from the
created file in the printToFile
 BST meta;
    std::string word;
    std::fstream fin;
    fin.open(fileInput);
    while(fin>>word){
    meta.addLeaf(word);}
    fin.close();
    std::ofstream outFile;
    outFile.open(fileOutput);
    meta.printToFile(meta.root, outFile);
    outFile.close();
}
```

main.cpp:

```
#include "bst.cpp"
#include <iostream>
#include <fstream>

using namespace std;

int main(){
    BST meta;

    meta.saveTreeToFile("macbeth.txt", "paradiselost.txt", "metaphysics.txt", "tdata.txt", "out.txt");
//  meta.loadTreeFromFile("out.txt","loadout.txt");
//  meta.PrintInOrder();
//  meta.PrintPreOrder();
//  meta.PrintPostOrder();
```

```
    std::cout<<std::endl<<"Same value count: "<<sameValCount<<std::endl;
    std::cout<<"Node Count: "<<nodeCount<<std::endl<<std::endl;

    if (meta.isBalanced())
      std::cout << "Tree is balanced"<<endl<<std::endl;
    else
      std::cout << "Tree is not balanced"<<endl<<std::endl;

    return 0;
}
```