Lab 9: Binary Trees

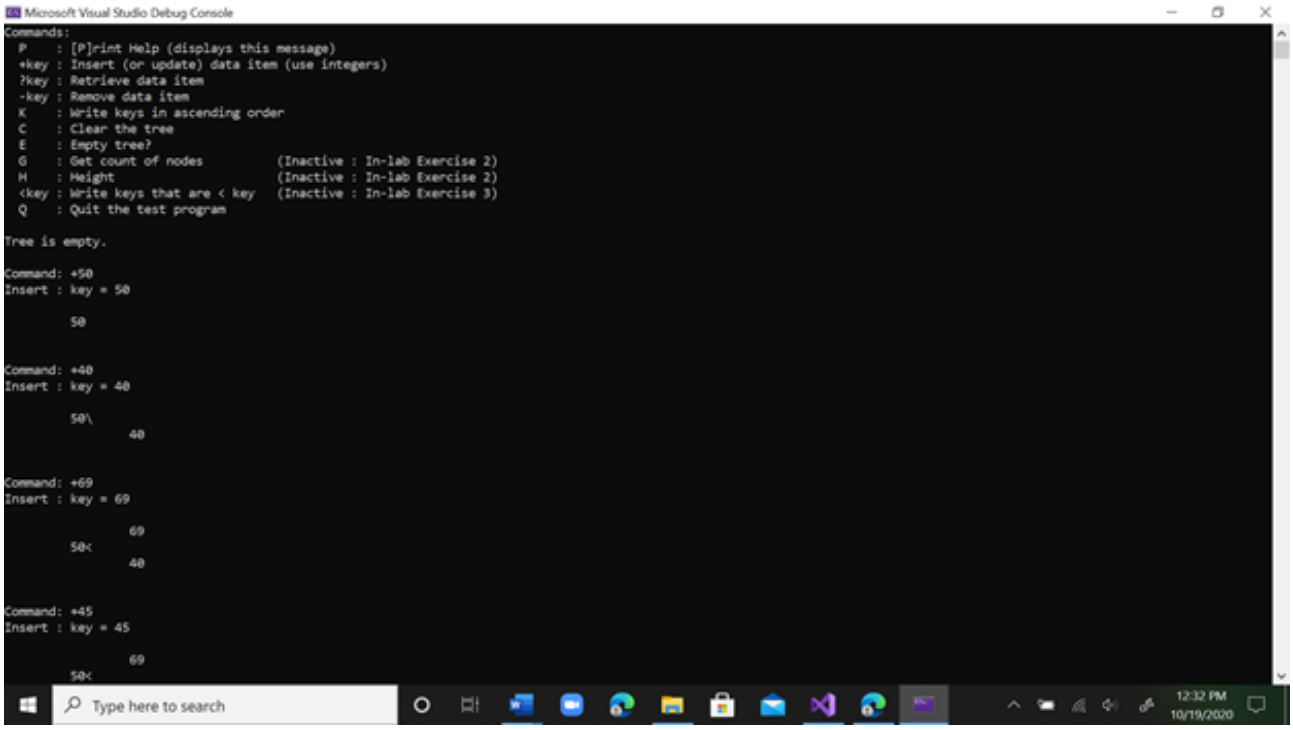
1

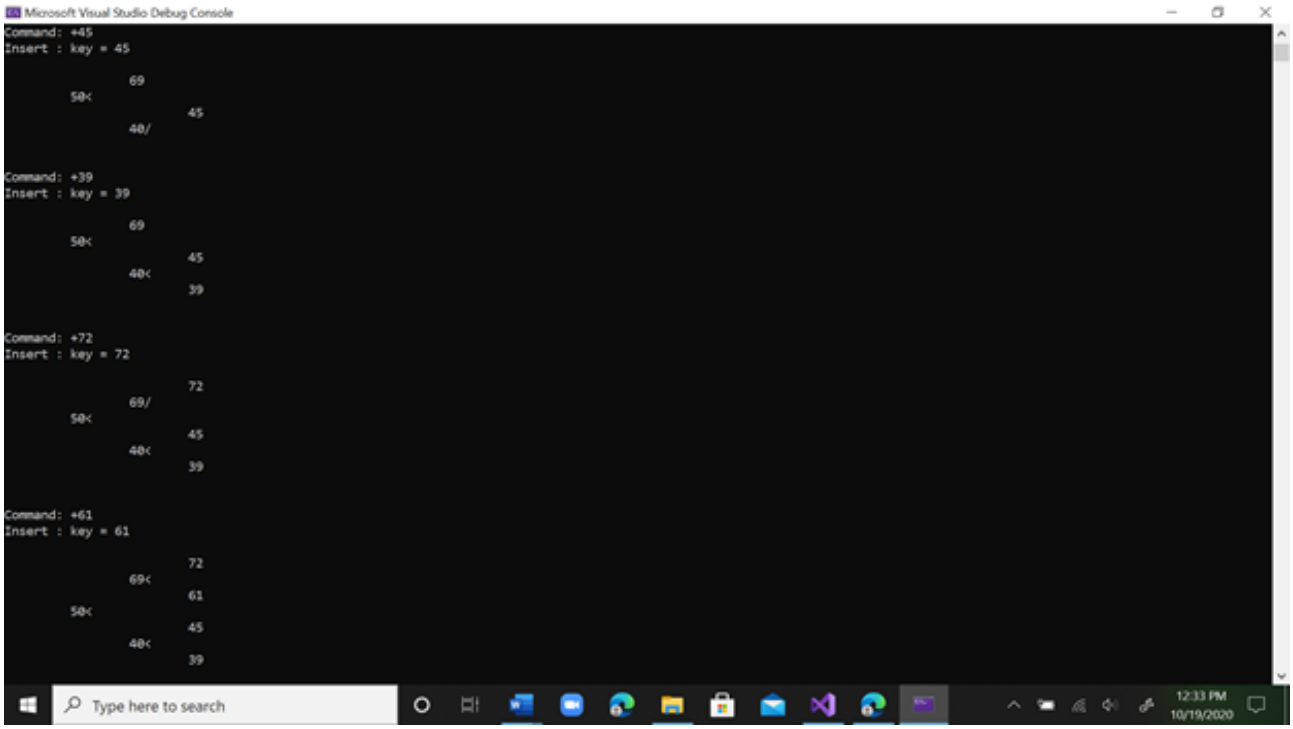
Generated by Doxygen 1.8.12

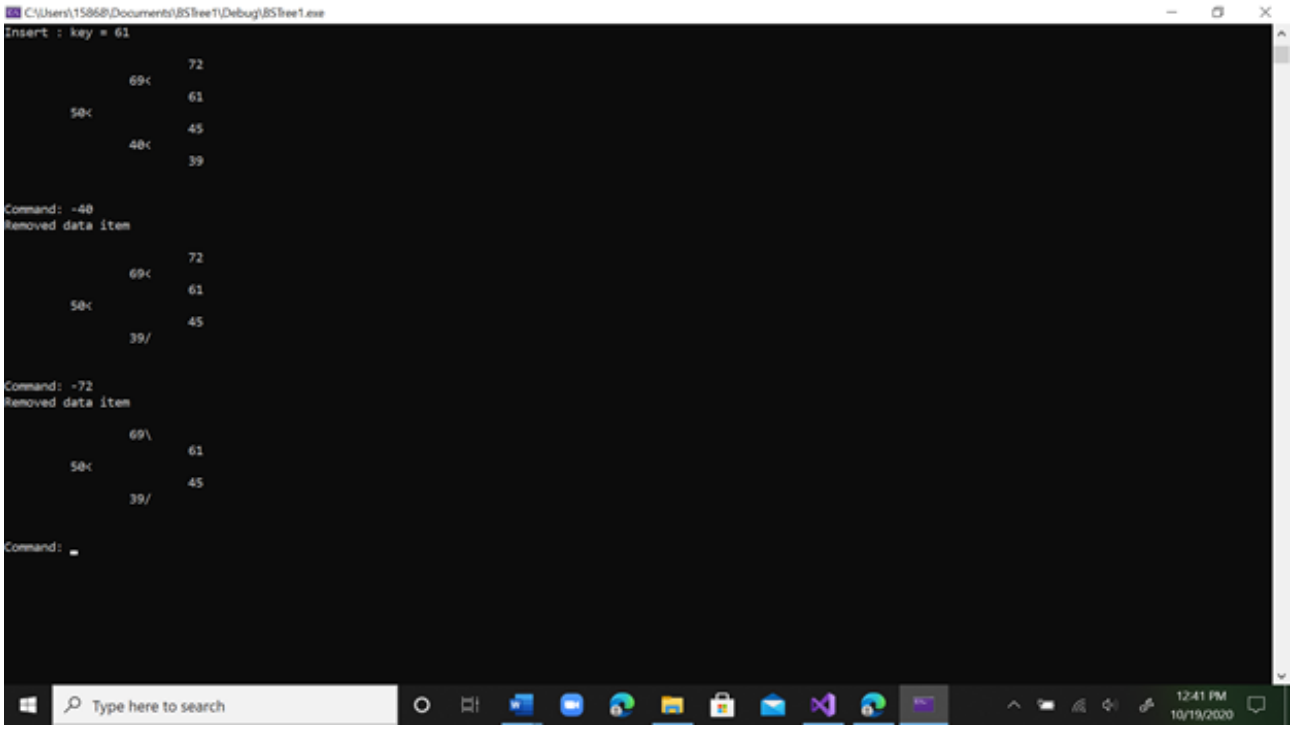
Contents

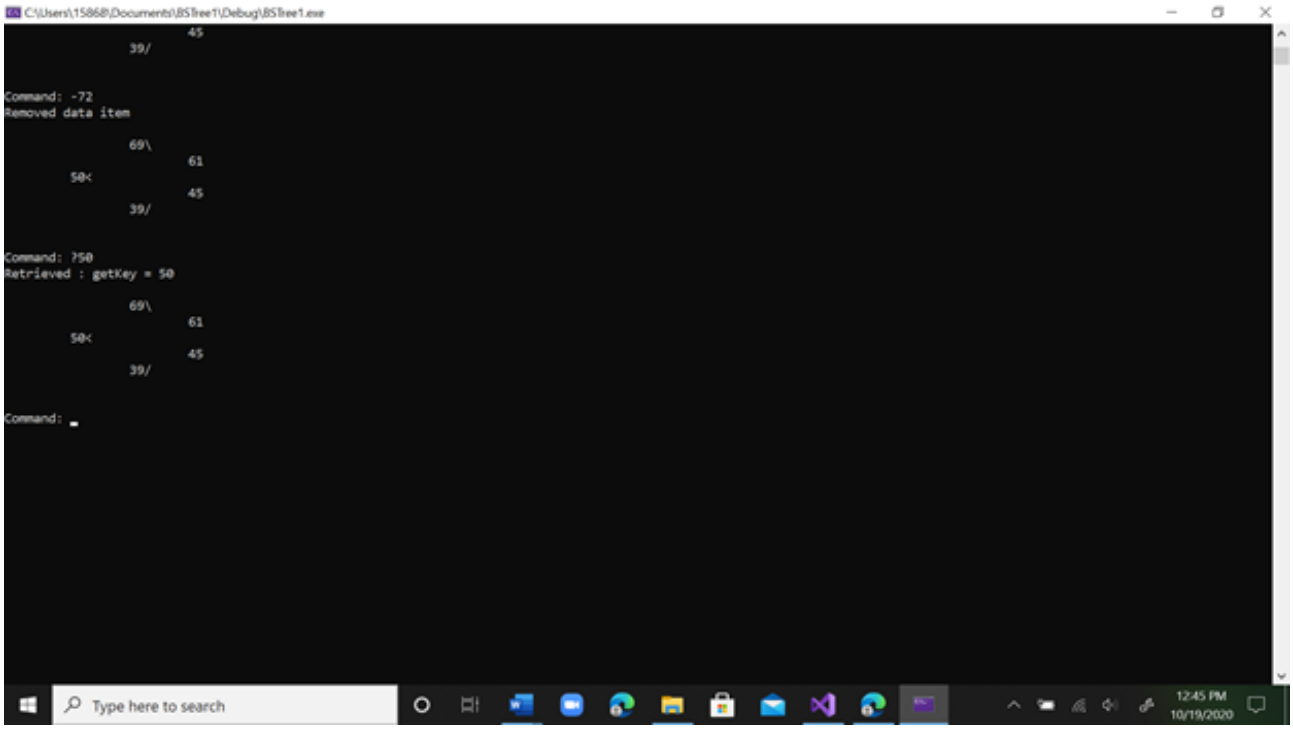
Chapter 1

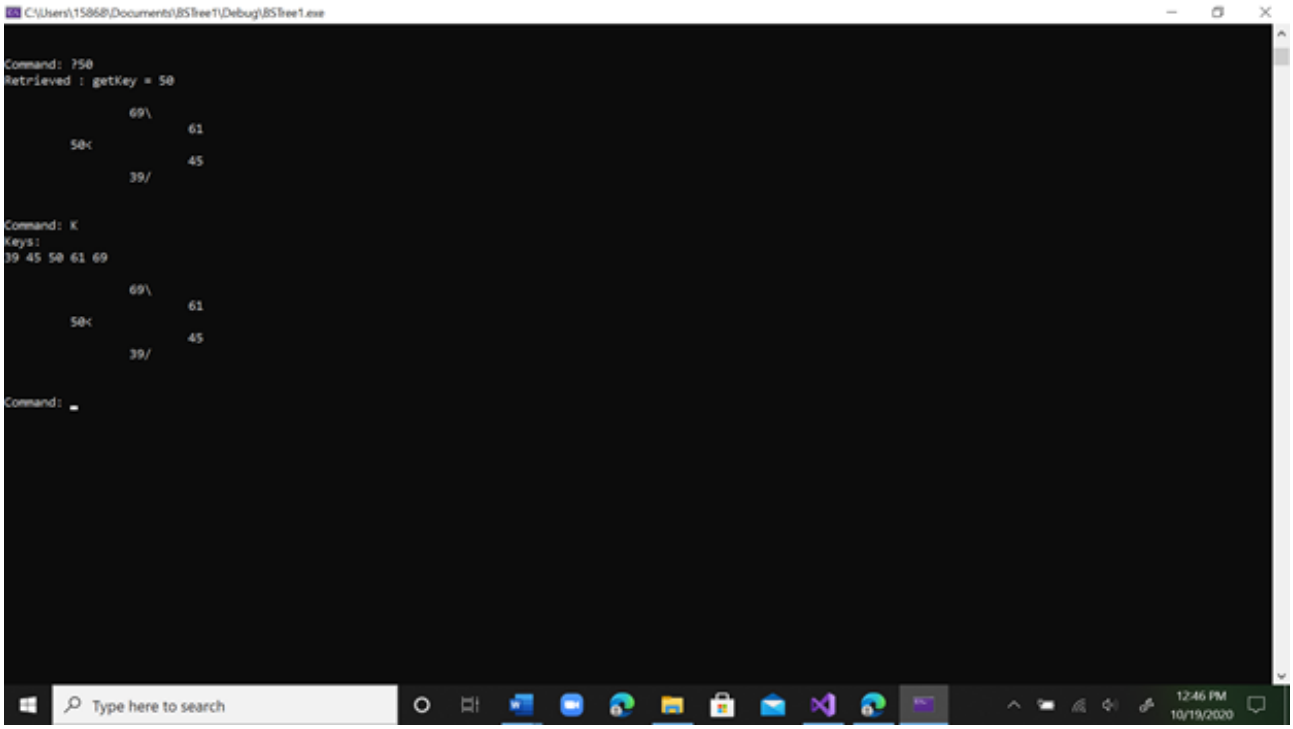
Outputs & Lab Sheets

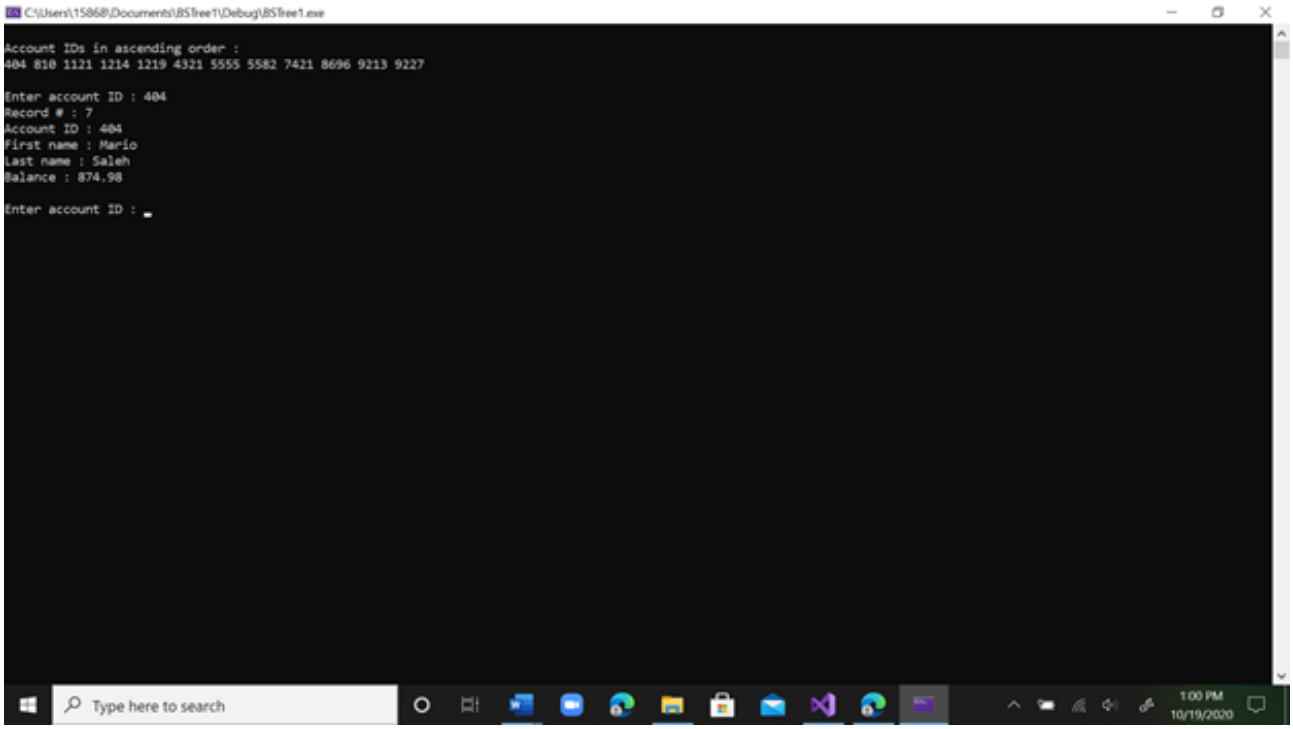


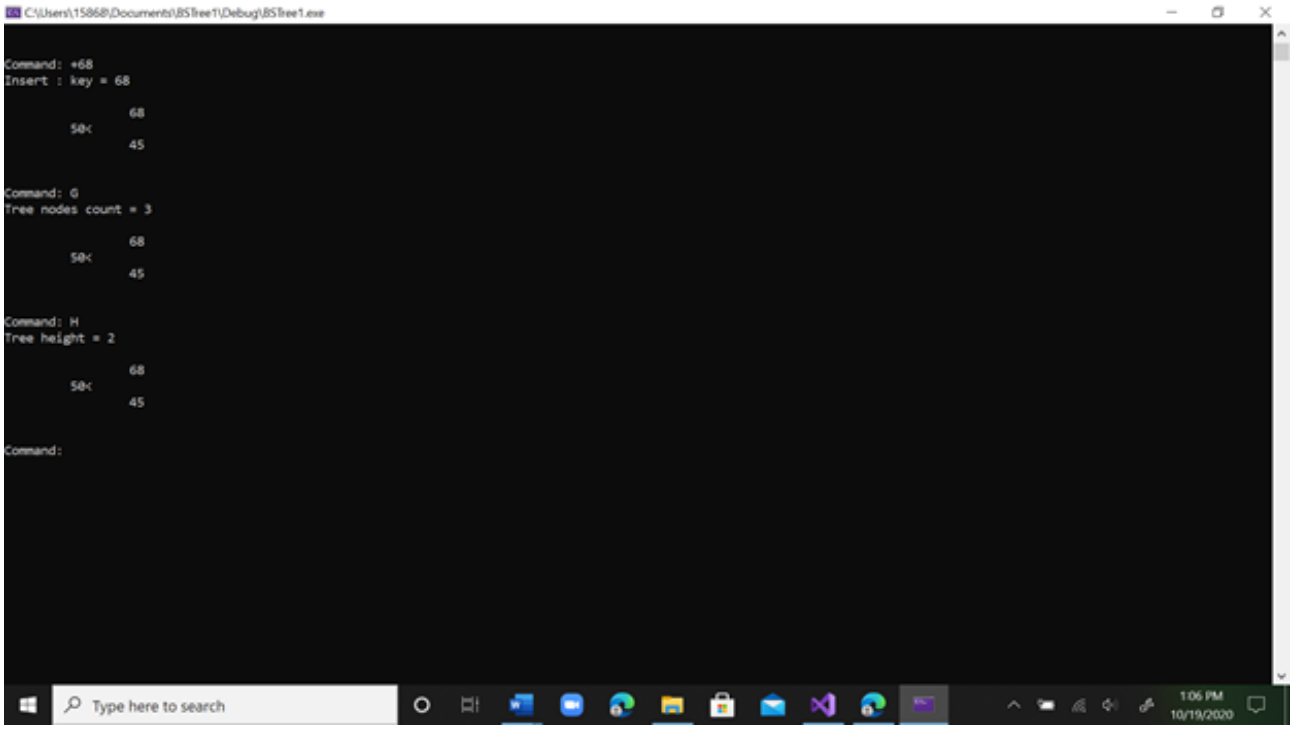

Microsoft Visual Studio Debug Console
Command: +45
Insert : key = 45
69
50<
45
40/
Command: +39
Insert : key = 39
69
50<
45
40<
39
Command: +72
Insert : key = 72
72
69/
50<
45
40<
39
Command: +61
Insert : key = 61
72
69<
61
50<
45
40<
39
Type here to search
12:33 PM
10/19/2020


C:\Users\15868\Documents\BSTree1\Debug\BSTree1.exe
Insert : key = 61
72
69<
61
50<
45
40<
39
Command: -40
Removed data item
72
69<
61
50<
45
39/
Command: -72
Removed data item
69\
61
50<
45
39/
Command:
Type here to search
12:41 PM
10/19/2020


C:\Users\15868\Documents\BSTree1\Debug\BSTree1.exe
45
39/
Command: -72
Removed data item
69\
61
50<
45
39/
Command: ?50
Retrieved : getKey = 50
69\
61
50<
45
39/
Command:
Type here to search
12:45 PM
10/19/2020


C:\Users\15868\Documents\BSTree1\Debug\BSTree1.exe
Command: ?50
Retrieved : getKey = 50
69\
61
50<
45
39/
Command: K
Keys:
39 45 50 61 69
69\
61
50<
45
39/
Command:
Type here to search
12:46 PM
10/19/2020


C:\Users\15868\Documents\BSTree1\Debug\BSTree1.exe
Account IDs in ascending order :
404 810 1121 1214 1219 4321 5555 5582 7421 8696 9213 9227
Enter account ID : 404
Record # : 7
Account ID : 404
First name : Mario
Last name : Saleh
Balance : 874.98
Enter account ID :
Type here to search
1:00 PM
10/19/2020


C:\Users\15868\Documents\BSTree1\Debug\BSTree1.exe
Command: +68
Insert : key = 68
68
50<
45
Command: G
Tree nodes count = 3
68
50<
45
Command: H
Tree height = 2
68
50<
45
Command:
Type here to search
1:06 PM
10/19/2020

Laboratory 9: Cover Sheet

Name _Aleczander__Dagher_____ Date __10/18/20_____

Section _____

Place a check mark in the *Assigned* column next to the exercises your instructor has assigned to you. Attach this cover sheet to the front of the packet of materials you submit following the laboratory.

Activities	**Assigned:** Check or list exercise numbers	**Completed**
Implementation Testing	✓	
Programming Exercise 1	✓	
Programming Exercise 2	✓	
Programming Exercise 3		
Analysis Exercise 1	✓	
Analysis Exercise 2	✓	
	Total	

Laboratory 9: Implementation Testing

Name _Aleczander__Dagher_____ Date ___10/18/20_____

Section _____

Check with your instructor whether you are to complete this exercise prior to your lab period or during lab.

Test Plan 9-1 (Binary Search Tree ADT operations)			
Test case	**Commands**	**Expected result**	**Checked**
Full binary tree	+50 +40 +69 +45 +39 +72 +61	50 ^ 40 69 ^ ^ 39 45 61 72	
Remove data item	-40 -72	50 ^ 39 69 \ / 45 61	
Retrieve data item	?50	getKey = 50	
Write keys in ascending order	K	Keys: 39 45 50 61 69	

Laboratory 9: Programming Exercise 1

Name _Aleczander__Dagher_____ Date ___10/18/20_____ Section _____

Test Plan 9-2 (accounts database indexing program)		
Test case	**Expected result**	**Checked**
Prints the account IDs in ascending order	404 810 1121 1214 1219 4321 5555 5582 7421 8696 9213 9227	
Enter ID: 404	Enter account ID : 404 Record # : 7 Account ID : 404 First name : Mario Last name : Saleh Balance : 874.98	

Laboratory 9: Programming Exercise 2

Name _Aleczander__Dagher_____ Date ___10/18/20_____ Section _____

Test Plan 9-3 (getCount operation)			
Test case	**Commands**	**Expected result**	**Checked**
getCount operation for the tree: 50 ^ 45 68	G	Tree nodes count = 3	

Test Plan 9-4 (getHeight operation)			
Test case	**Commands**	**Expected result**	**Checked**
getHeight operation for the tree: 50 ^ 45 68	H	Tree height = 2	

Laboratory 9: Analysis Exercise 1

Name _Aleczander_Dagher_____ Date __10/18/20_____ Section _____

What are the heights of the shortest and tallest binary search trees that can be constructed from a set of *N* distinct keys? Give examples that illustrate your answer.

The height of a tall binary search tree that can be constructed from a set of *N* distinct keys, is N. This can be done by inserting a number that is greater than the number you entered before.
Example:
+1 5/
+2 4/
+3 3/
+4 2/
+5 1/

The height of the shortest binary tree is also N. Getting a short height can be done by making the tree balanced.
Example:
+2 3
+3 2<
+1 1

Laboratory 9: Analysis Exercise 2

Name _Aleczander__Dagher_____ Date ___10/18/20_____ Section _____

Given the shortest possible binary search tree containing *N* distinct keys, develop worst-case, order-of-magnitude estimates of the execution time of the following Binary Search Tree ADT operations. Briefly explain your reasoning behind each of your estimates.

retrieve O(log (n))

Explanation: The worst case would be when the key is not in the tree, and then the function has to iterate through until it comes to the end of the tree.

insert O(log (n))

Explanation: The worst case would be to iterate to the end of the tree to insert a child node to a leaf node, which would turn the leaf node into a parent node.

remove O(log(n))

Explanation: The worst case would be for the function has to iterate through until it comes to the end of the tree in order to remove the node.

writeKeys O(n)

Explanation: I would estimate O(n) because it iterates once through every node.

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 files with brief descriptions:

Chapter 4

Class Documentation

4.1 AccountRecord Struct Reference

Public Attributes

- int acctID
- char firstName [nameLength]
- char lastName [nameLength]
- double balance

4.1.1 Member Data Documentation

4.1.1.1 acctID

```
int AccountRecord::acctID
```

4.1.1.2 balance

```
double AccountRecord::balance
```

4.1.1.3 firstName

```
char AccountRecord::firstName[nameLength]
```

4.1.1.4 lastName

```
char AccountRecord::lastName[nameLength]
```

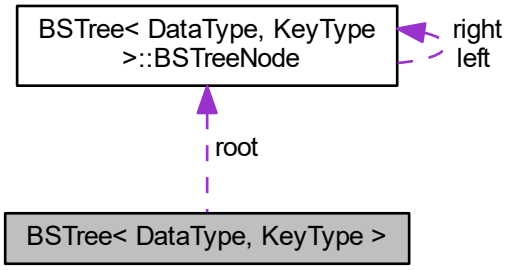
The documentation for this struct was generated from the following file:

- database.cpp

4.2 BSTree< DataType, KeyType > Class Template Reference

`#include <BSTree.h>`

Collaboration diagram for BSTree< DataType, KeyType >:



Classes

- class BSTreeNode

Public Member Functions

- BSTree ()
- BSTree (const BSTree< DataType, KeyType > &other)
- BSTree & operator= (const BSTree< DataType, KeyType > &other)
- ~BSTree ()
- void insert (const DataType &new_data_item)
- bool retrieve (const KeyType &search_key, DataType &search_data_item) const
- bool remove (const KeyType &delete_key)
- void writeKeys () const
- void clear ()
- bool isEmpty () const
- void showStructure () const
- int getHeight () const
- int getCount () const

Protected Member Functions

- bool remove_helper (BSTreeNode *&n, const KeyType &delete_key)
- void clear_helper (BSTreeNode *n)
- bool retrieve_helper (BSTreeNode *n, const KeyType &search_key, DataType &search_data_item) const
- int get_height_helper (BSTreeNode *n) const
- int get_count_helper (BSTreeNode *n) const
- void copy_tree (const BSTree< DataType, KeyType > &other_tree)
- void show_helper (BSTreeNode *n, int level) const
- void insert_helper (BSTreeNode *&n, const DataType &new_data_item)
- void copy_tree_helper (BSTreeNode *&n, const BSTreeNode *other_pointer)
- void write_keys_helper (BSTreeNode *n) const

Protected Attributes

- BSTreeNode * root

4.2.1 Constructor & Destructor Documentation

4.2.1.1 BSTree() [1/2]

```
template<typename DataType , typename KeyType >
BSTree< DataType, KeyType >::BSTree ( )
```

```
00042 {
00043     root = NULL;
00044 }
```

4.2.1.2 BSTree() [2/2]

```
template<typename DataType , typename KeyType >
BSTree< DataType, KeyType >::BSTree (
            const BSTree< DataType, KeyType > & other )
```

```
00054 {
00055     root = NULL;
00056     copy_tree(source);
00057 }
```

4.2.1.3 ~BSTree()

```
template<typename DataType , typename KeyType >
BSTree< DataType, KeyType >::~BSTree ( )
```

```
00116 {
00117     clear();
00118 }
```

4.2.2 Member Function Documentation

4.2.2.1 clear()

```
template<typename DataType , typename KeyType >
void BSTree< DataType, KeyType >::clear ( )
```

```
00299 {
00300     clear_helper(root);
00301     root = 0;
00302 }
```

4.2.2.2 clear_helper()

```
template<typename DataType , typename KeyType >
void BSTree< DataType, KeyType >::clear_helper (
            BSTreeNode * n )  [protected]
```

```
00312 {
00313     if (n != 0)
00314     {
00315         clear_helper(n->left);
00316         clear_helper(n->right);
00317         delete n;
00318     }
00319 }
```

4.2.2.3 copy_tree()

```
template<typename DataType , typename KeyType >
void BSTree< DataType, KeyType >::copy_tree (
            const BSTree< DataType, KeyType > & other_tree )  [protected]
```

```
00088 {
00089     copy_tree_helper(root, source_tree.root);
00090 }
```

4.2.2.4 copy_tree_helper()

```
template<typename DataType , typename KeyType >
void BSTree< DataType, KeyType >::copy_tree_helper (
            BSTreeNode *& n,
            const BSTreeNode * other_pointer )  [protected]
```

```
00100 {
00101     if (n != 0) {
00102         n = new BSTreeNode(source_pointer->dataItem, 0, 0);
00103         copy_tree_helper(n->left, source_pointer->left);
00104         copy_tree_helper(n->right, source_pointer->right);
00105     }
00106 }
```

4.2.2.5 get_count_helper()

```
template<typename DataType , typename KeyType >
int BSTree< DataType, KeyType >::get_count_helper (
            BSTreeNode * n ) const  [protected]
```

```
00436 {
00437     if (n == 0)
00438     {
00439         return 0;
00440     }
00441
00442     return get_count_helper(n->left) + get_count_helper(n->right) + 1;
00443 }
```

4.2.2.6 get_height_helper()

```
template<typename DataType , typename KeyType >
int BSTree< DataType, KeyType >::get_height_helper (
            BSTreeNode * n ) const  [protected]
```

```
00399 {
00400     int length_of_left, length_of_right, result;
00401
00402     if (n == 0)
00403         result = 0;
00404     else
00405     {
00406         length_of_left = get_height_helper(n->left) + 1;
00407         length_of_right = get_height_helper(n->right) + 1;
00408         if (length_of_left >= length_of_right)
00409             result = length_of_left;
00410         else
00411             result = length_of_right;
00412     }
00413
00414     return result;
00415 }
```

4.2.2.7 getCount()

```
template<typename DataType , typename KeyType >
int BSTree< DataType, KeyType >::getCount ( ) const
```

```
00424 {
00425     return get_count_helper(root);
00426 }
```

4.2.2.8 getHeight()

```
template<typename DataType , typename KeyType >
int BSTree< DataType, KeyType >::getHeight ( ) const
```

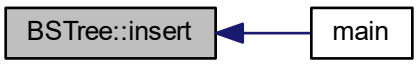
```
00387 {
00388     return get_height_helper(root);
00389 }
```

4.2.2.9 insert()

```
template<typename DataType , typename KeyType >
void BSTree< DataType, KeyType >::insert (
            const DataType & new_data_item )
```

```
00128 {
00129     insert_helper(root, new_data_item);
00130 }
```

Here is the caller graph for this function:



4.2.2.10 insert_helper()

```
template<typename DataType , typename KeyType >
void BSTree< DataType, KeyType >::insert_helper (
            BSTreeNode *& n,
            const DataType & new_data_item )  [protected]
```

```
00141 {
00142     if (n == 0) // Insert
00143         n = new BSTreeNode(new_data_item, 0, 0);
00144     else if (new_data_item.getKey() < n->dataItem.getKey())
00145         insert_helper(n->left, new_data_item);
00146     else if (new_data_item.getKey() > n->dataItem.getKey())
00147         insert_helper(n->right, new_data_item);
00148     else
00149         n->dataItem = new_data_item;
00150 }
```

4.2.2.11 isEmpty()

```
template<typename DataType , typename KeyType >
bool BSTree< DataType, KeyType >::isEmpty ( ) const
```

```
00329 {
00330     return root == 0;
00331 }
```

4.2.2.12 operator=()

```
template<typename DataType , typename KeyType >
BSTree< DataType, KeyType > & BSTree< DataType, KeyType >::operator= (
            const BSTree< DataType, KeyType > & other )
```

```
00067 {
00068     if (this != &source_tree)
00069     {
00070         clear();
00071         copy_tree(source_tree);
00072         return *this;
00073     }
00074     else
00075     {
00076         return *this;
00077     }
00078 }
```

4.2.2.13 remove()

```
template<typename DataType , typename KeyType >
bool BSTree< DataType, KeyType >::remove (
            const KeyType & delete_key )
```

```
00207 {
00208     return remove_helper(root, delete_key);
00209 }
```

4.2.2.14 remove_helper()

```
template<typename DataType , typename KeyType >
bool BSTree< DataType, KeyType >::remove_helper (
            BSTreeNode *& n,
            const KeyType & delete_key )  [protected]
```

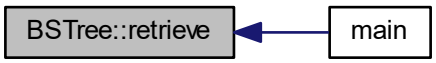
```
00220 {
00221     BSTreeNode* delete_pointer;
00222     int result;
00223
00224     if (n == 0)
00225         result = false;
00226     else if (delete_key < n->dataItem.getKey())
00227         result = remove_helper(n->left, delete_key);
00228     else if (delete_key > n->dataItem.getKey())
00229         result = remove_helper(n->right, delete_key);
00230     else
00231     {
00232         delete_pointer = n;
00233         if (n->left == 0)
00234         {
00235             n = n->right;
00236             delete delete_pointer;
00237         }
00238         else if (n->right == 0)
00239         {
00240             n = n->left;
00241             delete delete_pointer;
00242         }
00243         else
00244
00245         {
00246             BSTreeNode* temp = n->left;
00247             while (temp->right)
00248             {
00249                 temp = temp->right;
00250             }
00251             n->dataItem = temp->dataItem;
00252             remove_helper(n->left, temp->dataItem.getKey());
00253
00254         }
00255         result = true;
00256     }
00257
00258     return result;
00259 }
```

4.2.2.15 retrieve()

```
template<typename DataType , typename KeyType >
bool BSTree< DataType, KeyType >::retrieve (
            const KeyType & search_key,
            DataType & search_data_item ) const
```

```
00161 {
00162     return retrieve_helper(root, search_key, search_data_item);
00163 }
```

Here is the caller graph for this function:



4.2.2.16 retrieve_helper()

```
template<typename DataType , typename KeyType >
bool BSTree< DataType, KeyType >::retrieve_helper (
            BSTreeNode * n,
            const KeyType & search_key,
            DataType & search_data_item ) const  [protected]
```

```
00175 {
00176     bool result;
00177
00178     if (n == 0)
00179     {
00180         result = false;
00181     }
00182     else if (search_key < n->dataItem.getKey())
00183     {
00184         result = retrieve_helper(n->left, search_key, search_data_item);
00185     }
00186     else if (search_key > n->dataItem.getKey())
00187     {
00188         result = retrieve_helper(n->right, search_key, search_data_item);
00189     }
00190     else
00191     {
00192         search_data_item = n->dataItem;
00193         result = true;
00194     }
00195
00196     return result;
00197 }
```

4.2.2.17 show_helper()

```
template<typename DataType , typename KeyType >
void BSTree< DataType, KeyType >::show_helper (
            BSTreeNode * n,
            int level ) const  [protected]
```

```
00360 {
00361     if (n != 0)
00362     {
00363         show_helper(n->right, level + 1);
00364         for (int i = 0; i < level; i++)
00365             cout << "\t";
00366         cout << " " << n->dataItem.getKey();
00367         if ((n->left != 0) &&
00368             (n->right != 0))
00369             cout << "<";
00370         else if (n->right != 0)
00371             cout << "/";
00372         else if (n->left != 0)
00373             cout << "\\";
00374         cout << endl;
00375         show_helper(n->left, level + 1);
00376     }
00377 }
```

4.2.2.18 showStructure()

```
template<typename DataType , typename KeyType >
void BSTree< DataType, KeyType >::showStructure ( ) const
```

```
00341 {
00342     if (root == 0)
00343         cout << "Tree is empty." << endl;
00344     else
00345     {
00346         cout << endl;
00347         show_helper(root, 1);
00348         cout << endl;
00349     }
00350 }
```

4.2.2.19 write_keys_helper()

```
template<typename DataType , typename KeyType >
void BSTree< DataType, KeyType >::write_keys_helper (
             BSTreeNode * n ) const  [protected]
```

```
00282 {
00283     if (n != 0)
00284     {
00285         write_keys_helper(n->left);
00286         cout << n->dataItem.getKey() << " ";
00287         write_keys_helper(n->right);
00288     }
00289 }
```

4.2.2.20 writeKeys()

```
template<typename DataType , typename KeyType >
void BSTree< DataType, KeyType >::writeKeys ( ) const
```

```
00269 {
00270     write_keys_helper(root);
00271     cout << endl;
00272 }
```

Here is the caller graph for this function:

4.2.3 Member Data Documentation

4.2.3.1 root

```
template<typename DataType, class KeyType>
BSTreeNode* BSTree< DataType, KeyType >::root  [protected]
```

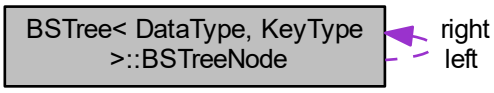
The documentation for this class was generated from the following files:

- BSTree.h
- BSTree.cpp

4.3 BSTree< DataType, KeyType >::BSTreeNode Class Reference

`#include <BSTree.h>`

Collaboration diagram for BSTree< DataType, KeyType >::BSTreeNode:



Public Member Functions

- BSTreeNode (const DataType &node_data_item, BSTreeNode *left_pointer, BSTreeNode *right_pointer)

Public Attributes

- DataType dataItem
- BSTreeNode * left
- BSTreeNode * right

4.3.1 Constructor & Destructor Documentation

4.3.1.1 BSTreeNode()

```
template<typename DataType , typename KeyType >
BSTree< DataType, KeyType >::BSTreeNode::BSTreeNode (
             const DataType & node_data_item,
             BSTreeNode * left_pointer,
             BSTreeNode * right_pointer )
```

```
00028 {
00029     dataItem = node_data_item;
00030     left = left_pointer;
00031     right = right_pointer;
00032 }
```

4.3.2 Member Data Documentation

4.3.2.1 dataItem

```
template<typename DataType, class KeyType>
DataType BSTree< DataType, KeyType >::BSTreeNode::dataItem
```

4.3.2.2 left

```
template<typename DataType, class KeyType>
BSTreeNode* BSTree< DataType, KeyType >::BSTreeNode::left
```

4.3.2.3 right

```
template<typename DataType, class KeyType>
BSTreeNode * BSTree< DataType, KeyType >::BSTreeNode::right
```

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

- BSTree.h
- BSTree.cpp

4.4 IndexEntry Struct Reference

Public Member Functions

- int getKey () const

Public Attributes

- int acctID
- long recNum

4.4.1 Member Function Documentation

4.4.1.1 getKey()

```
int IndexEntry::getKey ( ) const  [inline]

00046         { return acctID; }   // "Return key field."
```

4.4.2 Member Data Documentation

4.4.2.1 acctID

```
int IndexEntry::acctID
```

4.4.2.2 recNum

```
long IndexEntry::recNum
```

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

- database.cpp

Chapter 5

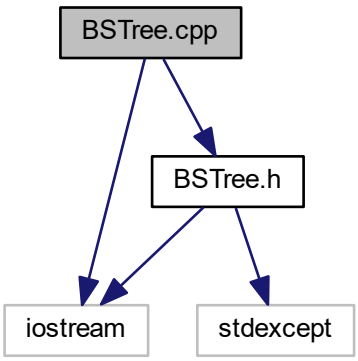
File Documentation

5.1 accounts.txt File Reference

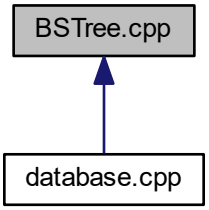
5.2 BSTree.cpp File Reference

```
#include <iostream>
#include "BSTree.h"
```

Include dependency graph for BSTree.cpp:



This graph shows which files directly or indirectly include this file:



Macros

- #define BSTREE_CPP

5.2.1 Macro Definition Documentation

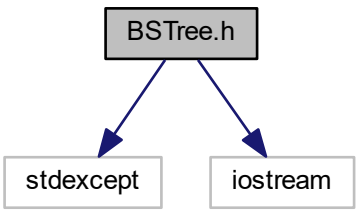
5.2.1.1 BSTREE_CPP

```
#define BSTREE_CPP
```

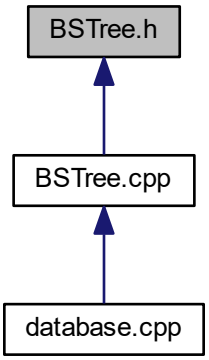
5.3 BSTree.h File Reference

```
#include <stdexcept>
#include <iostream>
```

Include dependency graph for BSTree.h:



This graph shows which files directly or indirectly include this file:



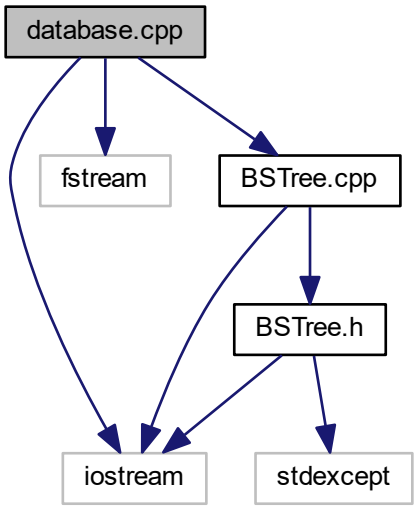
Classes

- class BSTree< DataType, KeyType >
- class BSTree< DataType, KeyType >::BSTreeNode

5.4 database.cpp File Reference

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

Include dependency graph for database.cpp:



Classes

- struct AccountRecord
- struct IndexEntry

Functions

- int main ()

Variables

- const int nameLength = 11
- const long bytesPerRecord = 38

5.4.1 Function Documentation

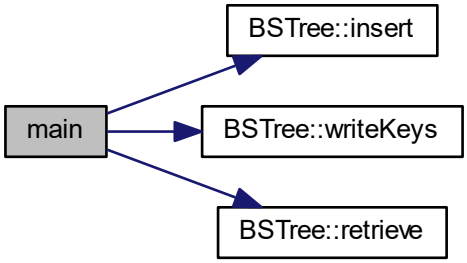
5.4.1.1 main()

```
int main ( )
```

```
00053 {
00054     ifstream acctFile ("accounts.txt");   // "Accounts database file."
00055     AccountRecord acctRec{};                // "Account record."
00056     BSTree<IndexEntry,int> index;         // "Database index."
00057     IndexEntry entry;                     // "Index entry."
00058     int searchID;                         // "User input account ID."
00059     long record_num{};                          // "Record number."
00060
00061     // "Iterates through the database records. This will read the
00062     // account ID and add the (account ID, record number) pair to the
00063     // index."
00064     string s;
00065     acctFile >> entry.acctID;
00066     while (acctFile.good())
00067     {
00068         acctFile >> s >> s >> s;
00069         entry.recNum = record_num;
00070         record_num++;
00071         index.insert(entry);
00072         acctFile >> entry.acctID;
00073     }
00074     // "Outputs the account IDs in ascending order."
00075     cout << "\nAccount IDs in ascending order : " << endl;
00076     index.writeKeys();
00077     // "Clears the status flags for the database file."
00078     acctFile.clear();
00079     acctFile.close();
00080     // "Reads an account ID from the keyboard and outputs the
00081     // corresponding record."
00082     acctFile.open("accounts.txt");
00083     cout << endl << "Enter account ID : ";
00084     while (cin >> searchID)
00085     {
00086         if (index.retrieve(searchID, entry))
00087         {
00088             for (int i = 0; i <= entry.recNum; i++)
00089             {
00090                 acctFile >> acctRec.acctID;
00091                 acctFile >> acctRec.firstName >> acctRec.lastName;
00092                 acctFile >> acctRec.balance;
00093             }
00094
00095             cout << "Record # : " << entry.recNum << endl;
00096             cout << "Account ID : " << acctRec.acctID << endl;
00097             cout << "First name : " << acctRec.firstName << endl;
00098             cout << "Last name : " << acctRec.lastName << endl;
00099             cout << "Balance : " << acctRec.balance << endl;
00100         }
00101         else
```

```
00102         {
00103             cout << "There is no record with that account ID. Try another one.";
00104 
00105         }
00106         acctFile.clear();
00107         acctFile.close();
00108         acctFile.open("accounts.txt");
00109         cout << endl << "Enter account ID : ";
00110     }
00111 }
```

Here is the call graph for this function:



5.4.2 Variable Documentation

5.4.2.1 bytesPerRecord

```
const long bytesPerRecord = 38
```

5.4.2.2 nameLength

```
const int nameLength = 11
```

5.5 C:/Users/15868/Documents/README2.md File Reference

Index