Andrew Tran

Lab Experience 13

**Hash table.h – HEADER**

#include<iostream>

#include<string>

#include<cstdlib>

#include<sstream>

using namespace std;

class HashTable

{

public:

//Linked List constructor

//Precons: None

//Postcons: None

HashTable()

{

first = NULL;

}

//Destructor method, destroys lists

//Precons: None

//Postcons: None

~HashTable()

{

Node\* ptr = first;

while (ptr != 0)

{

first = ptr->next;

delete ptr;

ptr = first;

}

}

//Insert function, inserts new nodes into linked lists

//Precons: string to be inserted

//Postcons: None

void insert(string inItem)

{

Node \* nPtr = new Node;

nPtr->item = inItem;

nPtr->next = NULL;

if (first == NULL)

{

first = nPtr;

first->next = NULL;

}

else

{

Node \* nPtr2;

nPtr2 = first;

while (nPtr2->next != NULL)

{

nPtr2 = nPtr2->next;

}

nPtr->next = NULL;

nPtr2->next = nPtr;

}

}

//Display function, prints linked lists

//Precons: None

//Postcons: None

void display()

{

Node \* nPtr;

nPtr = first;

cout << "Elements are:" << endl;

while (nPtr != NULL)

{

cout << nPtr->item << " // ";

nPtr = nPtr->next;

}

cout << endl;

}

private:

class Node

{

public:

string item;

Node \* next;

};

Node \* first, next;

};

//Calculates hash value

//Precons: hashtable array, user input string

//Postcons: None

void getHash(HashTable hashtable[], string item)

{

int hashval = 0;

//find the ascii sum

for (int i = 0; i < item.length(); i++)

{

if (i == 3)

break;

hashval += (int)item.at(i);

}

hashval = hashval % 11;

//insert

hashtable[hashval].insert(item);

}

**Hash driver.cpp – DRIVER**

#include <iostream>

#include "hash table.h"

using namespace std;

int main()

{

//array of linkedlist to implement hashtable

HashTable hashtable[11];

//item to insert

string item;

//read input

string a;

int hashval = 0;

while (1)

{

cout << "Insert a string to be inserted into the hash table (000 to break): " << endl;

getline(cin, a, '\n');

if (a == "000") //check for end of input

break;

stringstream buffer;

buffer << a;

//split the text as separate strings

while (getline(buffer, item, ' '))

{

getHash(hashtable, item);

}

}

//display the hashtable contents

for (int i = 0; i < 11; i++)

{

cout << "Hash Index " << i << endl << endl;

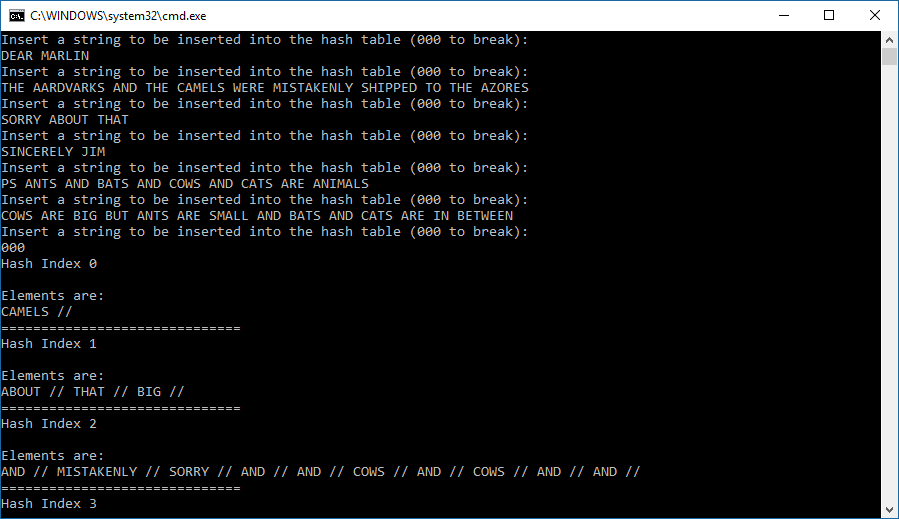
hashtable[i].display();

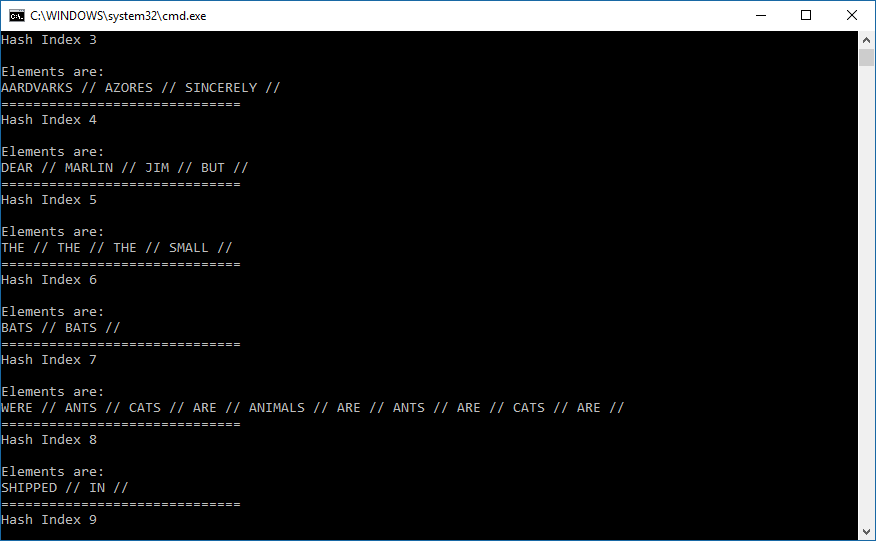
cout << "==============================" << endl;

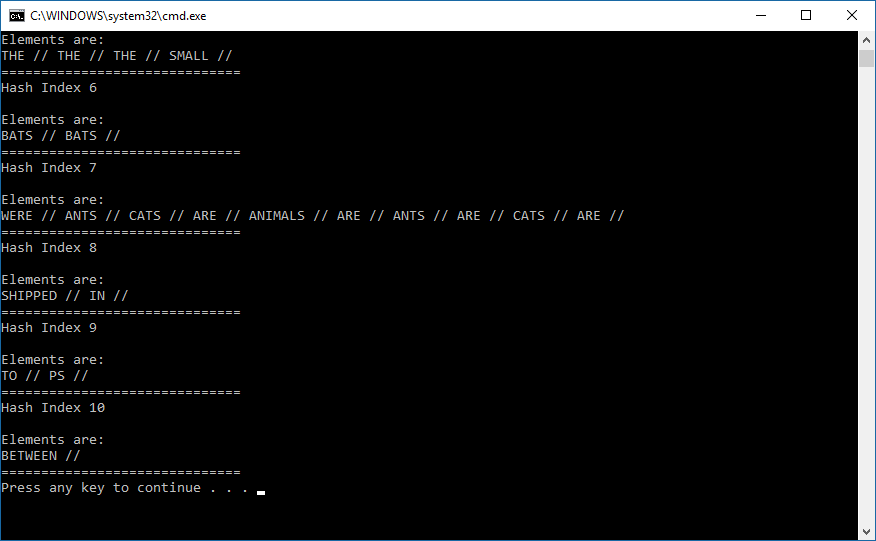
}

return 0;

}







**BST.h – HEADER**

#include <iostream>

#include <new>

#ifndef BINARY\_SEARCH\_TREE

#define BINARY\_SEARCH\_TREE

template <typename DataType>

class BST

{

public:

/\*\*\*\*\* Function Members \*\*\*\*\*/

BST();

/\*------------------------------------------------------------------------

Construct a BST object.

Precondition: None.

Postcondition: An empty BST has been constructed.

-----------------------------------------------------------------------\*/

bool empty() const;

/\*------------------------------------------------------------------------

Check if BST is empty.

Precondition: None.

Postcondition: Returns true if BST is empty and false otherwise.

-----------------------------------------------------------------------\*/

bool search(const DataType & item) const;

/\*------------------------------------------------------------------------

Search the BST for item.

Precondition: None.

Postcondition: Returns true if item found, and false otherwise.

-----------------------------------------------------------------------\*/

void insert(const DataType & item);

/\*------------------------------------------------------------------------

Insert item into BST.

Precondition: None.

Postcondition: BST has been modified with item inserted at proper

position to maintain BST property.

------------------------------------------------------------------------\*/

void remove(const DataType & item);

/\*------------------------------------------------------------------------

Remove item from BST.

Precondition: None.

Postcondition: BST has been modified with item removed (if present);

BST property is maintained.

Note: remove uses private auxiliary function search2() to locate

the node containing item and its parent.

------------------------------------------------------------------------\*/

void graph(ostream & out) const;

/\*------------------------------------------------------------------------

Graphic output of BST.

Precondition: ostream out is open.

Postcondition: Graphical representation of BST has been output to out.

Note: graph() uses private auxiliary function graphAux().

------------------------------------------------------------------------\*/

void inorder(ostream & out) const;

/\*------------------------------------------------------------------------

Inorder traversal of BST.

Precondition: ostream out is open.

Postcondition: BST has been inorder traversed and values in nodes

have been output to out.

Note: inorder uses private auxiliary function inorderAux().

------------------------------------------------------------------------\*/

//--- ADD PROTOTYPES OF preorder() AND postorder() HERE

void preorder(ostream & out) const;

void postorder(ostream & out) const;

//--- ADD PROTOTYPE OF DESTRUCTOR HERE

~BST()

{

destroy(myRoot);

}

//--- ADD PROTOTYPE OF COPY CONSTRUCTOR HERE

BST(const BST<DataType> &original)

{

copyTree(original.myRoot, myRoot);

}

//--- ADD PROTOTYPE OF ASSIGNMENT OPERATOR HERE

BST<DataType>& operator=(const BST<DataType>& origList);

//--- ADD PROTOTYPE OF LEVEL-BY-LEVEL TRAVERSAL HERE

//--- ADD PROTOTYPE OF LEVEL FINDER HERE

private:

/\*\*\*\*\* Node class \*\*\*\*\*/

class BinNode

{

public:

DataType data;

BinNode \* left;

BinNode \* right;

// BinNode constructors

// Default -- data part is default DataType value; both links are null.

BinNode()

: left(0), right(0)

{}

// Explicit Value -- data part contains item; both links are null.

BinNode(DataType item)

: data(item), left(0), right(0)

{}

};// end of class BinNode declaration

typedef BinNode \* BinNodePointer;

/\*\*\*\*\* Private Function Members \*\*\*\*\*/

void search2(const DataType & item, bool & found,

BinNodePointer & locptr, BinNodePointer & parent) const;

/\*------------------------------------------------------------------------

Locate a node containing item and its parent.

Precondition: None.

Postcondition: locptr points to node containing item or is null if

not found, and parent points to its parent.#include <iostream>

------------------------------------------------------------------------\*/

void destroy(BinNodePointer subtreeRoot)

{

if (subtreeRoot != NULL)

{

destroy(subtreeRoot->left);

destroy(subtreeRoot->right);

delete subtreeRoot;

}

}

void copyTree(BinNodePointer origRoot, BinNodePointer &subtreeRoot)

{

if (origRoot == NULL)

{

subtreeRoot = NULL;

}

else

{

subtreeRoot = new BinNode(origRoot->data);

copyTree(origRoot->left, subtreeRoot->left);

copyTree(origRoot->right, subtreeRoot->right);

}

}

void inorderAux(ostream & out,

BinNodePointer subtreePtr) const;

void preorderAux(ostream & out,

BinNodePointer subtreePtr) const;

void postorderAux(ostream & out,

BinNodePointer subtreePtr) const;

/\*------------------------------------------------------------------------

Inorder traversal auxiliary function.

Precondition: ostream out is open; subtreePtr points to a subtree

of this BST.

Postcondition: Subtree with root pointed to by subtreePtr has been

output to out.

------------------------------------------------------------------------\*/

void graphAux(ostream & out, int indent,

BinNodePointer subtreeRoot) const;

/\*------------------------------------------------------------------------

Graph auxiliary function.

Precondition: ostream out is open; subtreePtr points to a subtree

of this BST.

Postcondition: Graphical representation of subtree with root pointed

to by subtreePtr has been output to out, indented indent spaces.

------------------------------------------------------------------------\*/

/\*\*\*\*\* Data Members \*\*\*\*\*/

BinNodePointer myRoot;

}; // end of class template declaration

//--- Definition of constructor

template <typename DataType>

inline BST<DataType>::BST()

: myRoot(0)

{}

//--- Definition of empty()

template <typename DataType>

inline bool BST<DataType>::empty() const

{ return myRoot == 0; }

//--- Definition of search()

template <typename DataType>

bool BST<DataType>::search(const DataType & item) const

{

BST<DataType>::BinNodePointer locptr = myRoot;

bool found = false;

while (!found && locptr != 0)

{

if (item < locptr->data) // descend left

locptr = locptr->left;

else if (locptr->data < item) // descend right

locptr = locptr->right;

else // item found

found = true;

}

return found;

}

//--- Definition of insert()

template <typename DataType>

inline void BST<DataType>::insert(const DataType & item)

{

BST<DataType>::BinNodePointer

locptr = myRoot, // search pointer

parent = 0; // pointer to parent of current node

bool found = false; // indicates if item already in BST

while (!found && locptr != 0)

{

parent = locptr;

if (item < locptr->data) // descend left

locptr = locptr->left;

else if (locptr->data < item) // descend right

locptr = locptr->right;

else // item found

found = true;

}

if (!found)

{ // construct node containing item

locptr = new(nothrow) BST<DataType>::BinNode(item);

if (locptr == 0)

{

cerr << "\*\*\* Out of memory -- terminating program \*\*\*\n";

exit(1);

}

if (parent == 0) // empty tree

myRoot = locptr;

else if (item < parent->data ) // insert to left of parent

parent->left = locptr;

else // insert to right of parent

parent->right = locptr;

}

else

cout << "Item already in the tree\n";

}

//--- Definition of remove()

template <typename DataType>

void BST<DataType>::remove(const DataType & item)

{

bool found; // signals if item is found

BST<DataType>::BinNodePointer

x, // points to node to be deleted

parent; // " " parent of x and xSucc

search2(item, found, x, parent);

if (!found)

{

cout << "Item not in the BST\n";

return;

}

//else

if (x->left != 0 && x->right != 0)

{ // node has 2 children

// Find x's inorder successor and its parent

BST<DataType>::BinNodePointer xSucc = x->right;

parent = x;

while (xSucc->left != 0) // descend left

{

parent = xSucc;

xSucc = xSucc->left;

}

// Move contents of xSucc to x and change x

// to point to successor, which will be removed.

x->data = xSucc->data;

x = xSucc;

} // end if node has 2 children

// Now proceed with case where node has 0 or 2 child

BST<DataType>::BinNodePointer

subtree = x->left; // pointer to a subtree of x

if (subtree == 0)

subtree = x->right;

if (parent == 0) // root being removed

myRoot = subtree;

else if (parent->left == x) // left child of parent

parent->left = subtree;

else // right child of parent

parent->right = subtree;

delete x;

}

//--- Definition of graph()

template <typename DataType>

inline void BST<DataType>::graph(ostream & out) const

{ graphAux(out, 0, myRoot); }

//--- Definition of search2()

template <typename DataType>

void BST<DataType>::search2(const DataType & item, bool & found,

BinNodePointer & locptr,

BinNodePointer & parent) const

{

locptr = myRoot;

parent = 0;

found = false;

while (!found && locptr != 0)

{

if (item < locptr->data) // descend left

{

parent = locptr;

locptr = locptr->left;

}

else if (locptr->data < item) // descend right

{

parent = locptr;

locptr = locptr->right;

}

else // item found

found = true;

}

}

//--- Definition of graphAux()

#include <iomanip>

template <typename DataType>

void BST<DataType>::graphAux(ostream & out, int indent,

BinNodePointer subtreeRoot) const

{

if (subtreeRoot != 0)

{

graphAux(out, indent + 8, subtreeRoot->right);

out << setw(indent) << " " << subtreeRoot->data << endl;

graphAux(out, indent + 8, subtreeRoot->left);

}

}

//--- Definition of inorder()

template <typename DataType>

inline void BST<DataType>::inorder(ostream & out) const

{

inorderAux(out, myRoot);

}

//--- Definition of inorderAux()

template <typename DataType>

void BST<DataType>::inorderAux(ostream & out,

BinNodePointer subtreeRoot) const

{

if (subtreeRoot != 0)

{

inorderAux(out, subtreeRoot->left); // L operation

out << subtreeRoot->data << " "; // V operation

inorderAux(out, subtreeRoot->right); // R operation

}

}

//--- PUT DEFINITIONS OF THE ADDED OPERATIONS HERE

//assignment operator

template <typename DataType>

BST<DataType>& BST<DataType>::operator=(const BST<DataType>& origRoot)

{

if (this != &origRoot)

{

destroy(myRoot);

copyTree(origRoot.myRoot, myRoot);

}

return \*this;

}

//--- Definition of preorder()

template <typename DataType>

inline void BST<DataType>::preorder(ostream & out) const

{

preorderAux(out, myRoot);

}

//--- Definition of preorderAux()

template <typename DataType>

void BST<DataType>::preorderAux(ostream & out,

BinNodePointer subtreeRoot) const

{

if (subtreeRoot != 0)

{

out << subtreeRoot->data << " "; // V operation

inorderAux(out, subtreeRoot->left); // L operation

inorderAux(out, subtreeRoot->right); // R operation

}

}

template <typename DataType>

inline void BST<DataType>::postorder(ostream & out) const

{

postorderAux(out, myRoot);

}

//--- Definition of postorderAux()

template <typename DataType>

void BST<DataType>::postorderAux(ostream & out,

BinNodePointer subtreeRoot) const

{

if (subtreeRoot != 0)

{

inorderAux(out, subtreeRoot->left); // L operation

inorderAux(out, subtreeRoot->right); // R operation

out << subtreeRoot->data << " "; // V operation

}

}

**TREETESTER.CPP**

#include <iostream>

using namespace std;

#include "BST.h"

//\*---- PART 3 ----

// makeCopy() is a function with a

void makeCopy(BST<int> aBST) // BST value parameter

{ // to test the copy constructor

cout << "\nNow copying the BST and adding 38999,"

" -12312, and 55657 to the copy:\n";

aBST.insert(38999);

aBST.insert(-12312);

aBST.insert(55657);

cout << "--Here's the modified copy: \n";

aBST.graph(cout);

}

//---- END PART 3 ----\*/

int main()

{

const char MENU[] =

"MENU CHOICES\n"

"0. Display the menu\n"

"1. Check if BST is empty\n"

"2. Insert some elements into the BST\n"

"3. Search for an element\n"

"4. Delete some elements from the BST\n"

"5. Graphical representation of BST (sideways)\n"

"6. Inorder traversal\n"

"61. Preorder traversal\n"

"62. Postorder traversal\n"

"7. Check destructor\n"

"8. Check copy constructor\n"

"9. Check assignment operator\n"

"10. Quit the program\n";

// Testing Constructor and empty()

BST<int> intBST; // test the class constructor

cout << "Constructing empty BST\n";

cout << "BST " << (intBST.empty() ? "is" : "is not") << " empty\n";

// Test Other Operations

cout << MENU << endl;

int choice;

do

{

cout << "\nEnter a menu choice (0 for menu, 5 to graph, 10 to stop): ";

cin >> choice;

switch(choice)

{

case 0: // Display menu

cout << MENU << endl;

break;

case 1: // Checking empty

cout << "BST " << (intBST.empty() ? "is" : "is not")

<< " empty\n";

break;

case 2: // Insert elements

cout << "\nNow insert a bunch of integers into the BST."

"\nTry items not in the BST and some that are in it:\n";

int number;

for (;;)

{

cout << "Item to insert (-999 to stop inserting): ";

cin >> number;

if (number == -999) break;

intBST.insert(number);

cout << "Here's the BST:\n";

intBST.graph(cout);

}

break;

case 3: // Searching)

cout << "\n\nNow testing the search() operation."

"\nTry both items in the BST and some not in it:\n";

for (;;)

{

cout << "Item to find (-999 to stop searching): ";

cin >> number;

if (number == -999) break;

cout << (intBST.search(number) ? "Found" : "Not found") << endl;

}

break;

case 4: // Deleting elements

cout << "\nNow testing the remove() operation."

"\nTry both items in the BST and some not in it:\n";

for (;;)

{

cout << "Item to delete (-999 to stop deleting): ";

cin >> number;

if (number == -999) break;

intBST.remove(number);

cout << "Here's the BST:\n";

intBST.graph(cout);

}

break;

case 5: // Graphical representation

cout << "Here's the BST (sidewise):\n";

intBST.graph(cout);

break;

case 6: // Inorder traversal

cout << "\nInorder Traversal of BST: \n";

intBST.inorder(cout);

cout << endl;

break;

default:

cerr << "BAD CHOICE -- TRY AGAIN\n";

break;

case 10: // quit menu

break;

//\* ---- PART 1 ----

case 61: // Test preorder traversal

cout << "\nPreorder Traversal of BST: \n";

intBST.preorder(cout);

break;

//---- END PART 1 ----\*/

//\* ---- PART 1 ----

case 62: // Test postorder traversal

cout << "\nPostorder Traversal of BST: \n";

intBST.postorder(cout);

break;

//---- END PART 2 ----\*/

//\* ---- PART 2 ----

case 7: // Testing the Destructor

cout << "\nNow testing the destructor. Remember to add an\n"

"output statement to your destructor to indicate \n"

"when it is called.\n";

{

BST<int> doomedBST;

doomedBST.insert(6); doomedBST.insert(9); doomedBST.insert(5);

doomedBST.insert(1); doomedBST.insert(3); doomedBST.insert(7);

cout << "\nHere's a BST:\n";

doomedBST.graph(cout);

cout << "\n\nLifetime of this BST is over -- now destroy it.\n";

}

break;

//---- END PART 2 ----\*/

//\* ---- PART 3 ----

case 8: // Testing the Copy Constructor

{

cout << "\nNow testing the copy constructor.\n";

cout << "-- First with an initializing declaration: "

"BST<int> copy = intBST;\n";

BST<int> copy = intBST;

cout << "Here's the original:\n";

intBST.graph(cout);

cout << "\nHere's the copy:\n";

copy.graph(cout);

}

cout << "\n\n-- Now by passing intBST to a value parameter:\n";

makeCopy(intBST);

cout << "\n--Check that original BST hasn't been changed.\n"

"-- Inorder traversal of original:\n";

intBST.graph(cout);

cout << endl;

break;

//---- END PART 3 ----\*/

//\* ---- PART 4 ----

case 9: // Testing the Assignment Operator

cout << "\nNow testing the assignment operator with the "

"statement:\n and\_anotherBST = anotherBST = intBST;\n";

BST<int> anotherBST,

and\_anotherBST;;

and\_anotherBST = anotherBST = intBST;

cout << "\n-- Here's intBST:\n";

intBST.graph(cout);

cout << "\nHere's anotherBST:\n";

anotherBST.graph(cout);

cout << "\nand\_anotherBST:\n";

and\_anotherBST.graph(cout);

cout << "\nNow testing self-assignment with"

"\n anotherBST = anotherBST;\n";

anotherBST = anotherBST;

cout << "\nHere's anotherBST:\n";

anotherBST.graph(cout);

cout << endl;

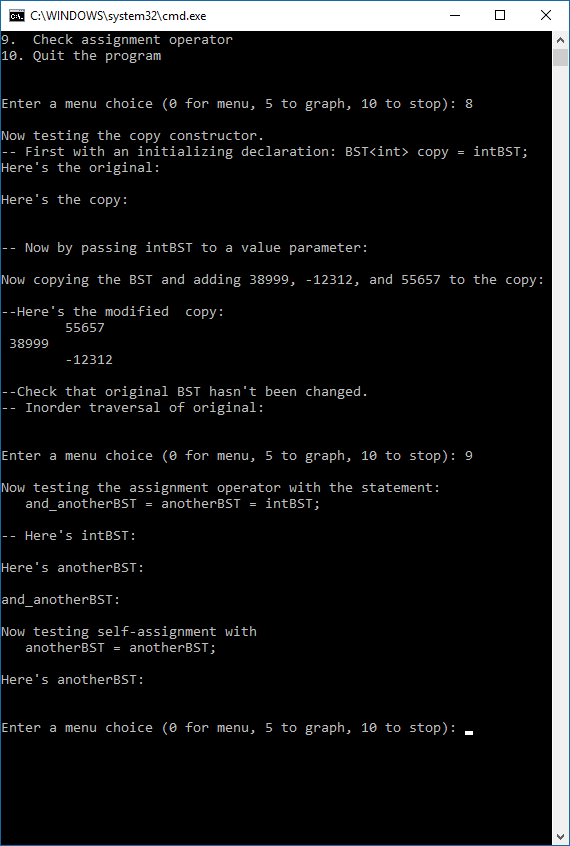
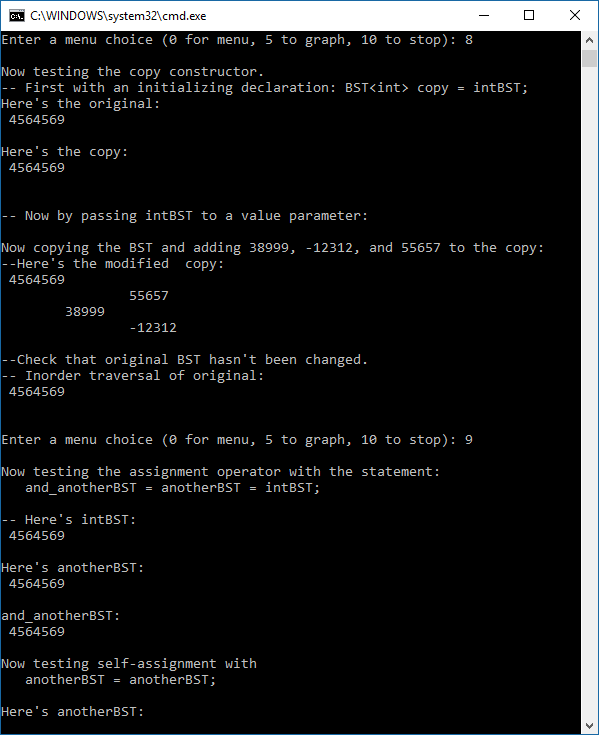
break;

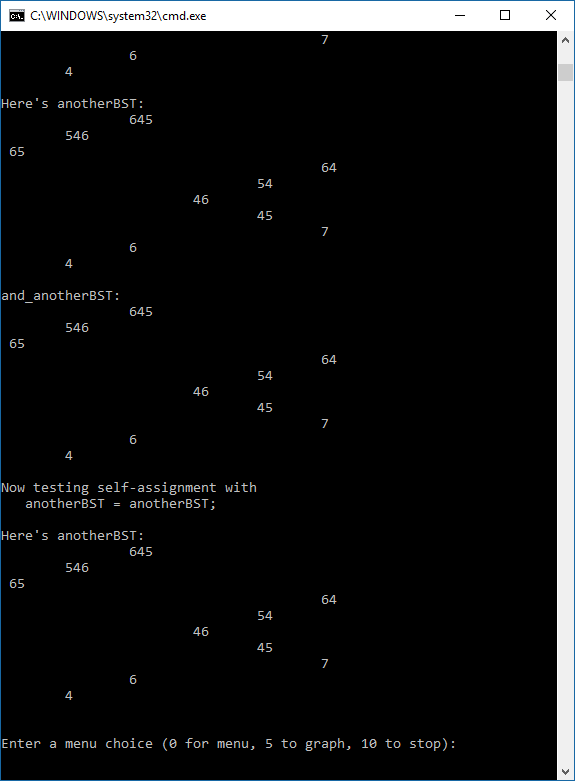
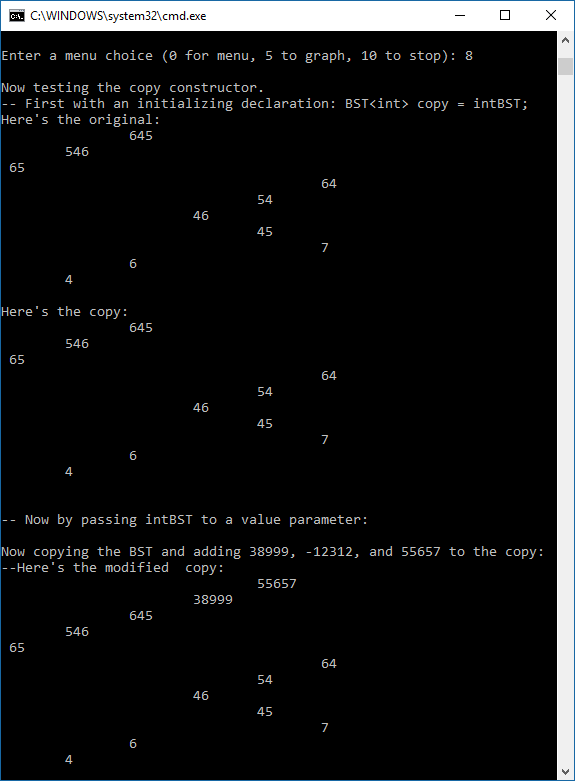
//---- END PART 4 ----\*/

} // switch

} while (choice != 10);

} // end main()

**EMPTY BST ONE BST**

**MANY BST**