CSc 300
Assignment #4
Gamradt
Due: 11-08-23 (Late: 11-15-23)

Create a user-defined Abstract Data Type (ADT) named **BST**

- Use an appropriate set of C++ header/implementation files as discussed in class
- **BST** is implemented using **single linked structure**
- **BST** consists of 0 or more **Element** values
  - **Element** is an exportable **int** data type
    - **Element** is managed using **dynamically allocated nodes – Node**
    - See **C++ Pointers** and **Tree ADT** under D2L Lecture Notes
  - **Node** consists of three fields:
    - **element, left, right**

The **BST** ADT must define and implement the following data types and operations.

- Do not add to or modify the public interface (exportable components – public components).
- Do not add to or modify any attributes or data types (storage components).

<u>Exportable Operations: (declared .h file and defined .cpp file)</u>

BST	default constructor – create an initialized empty BST	
BST	copy constructor – uses **copy** to create a duplicate copy of an existing BST	
~BST	destructor function – uses **destroy** to destroy an existing BST	
	BST instance state before going out of scope – initialized empty BST	
insert	inserts a new key node to the BST – ignore duplicates	(*)
	do not insert duplicate nodes – do not count duplicate insert attempts	
remove	locates an existing key node to be removed from the BST	(*)
	uses **removeNode** to handle the actual node removal process	
search	returns a pointer to an existing key node in the BST, otherwise NULL	(*)
	not used as part of the remove key node process (returns an external pointer)	
preorderView	displays the keys in the BST from top to bottom (left to right)	(*)
inorderView	displays the keys in the BST in ascending order	(*)
postorderView	displays the keys in the BST from bottom to top (left to right)	(*)

<u>Non-Exportable Operations: (declared .h file and defined .cpp file)</u>

copy	recursively copies an existing BST (**form of pre-order traversal**)
destroy	recursively removes all key nodes from the BST (**form of post-order traversal**)
removeNode	removes an existing key node from the BST
findMaxNode	finds the maximum key node in the left subtree of the BST
(*)	recursive version of each of the 6 exportable functions (function overloading required)

<u>User-Defined Data Types:</u>

Element
Node
NodePtr

BST Required Output Format: (inorderView)

```
//  Empty Tree                               //  Populated Tree
BEGIN -> END                                 BEGIN -> 5 -> 10 -> 15 -> END
```

Required header file (.h).   **// only partially specified**

```
//  General description of the ADT and supported operations – exportable operations only
//  Do not include any implementation details

#pragma once                                            // alternative Guard format
typedef int Element;
struct Node;
typedef Node * NodePtr;
struct Node {
        Element element;
        NodePtr left, right;
};

class BST {
        public:                                         // exportable
//  General description of each of the ADT operations/functions – exportable operations only
                BST();
                BST( const BST & );
                ~BST();
                void insert( const Element );
                void remove( const Element );
                NodePtr search( const Element ) const;
                void preorderView() const;
                void inorderView() const;
                void postorderView() const;
        private:                                        // non-exportable
//  No private member documentation – implementation details are hidden/abstracted away
                NodePtr root;
                void copy( const NodePtr );
                void destroy( NodePtr & );
                void removeNode( NodePtr & );
                void findMaxNode( NodePtr &, NodePtr & );
                void insert( NodePtr &, const Element );
                void remove( NodePtr &, const Element);
                NodePtr search( const NodePtr, const Element) const;
                void preorderView( const NodePtr ) const;
                void inorderView( const NodePtr ) const;
                void postorderView( const NodePtr ) const;
};
```

BST ADT include sequence: // Never include .cpp files

main.cpp ——————→ BST.h ←—————— BST.cpp

BST ADT incremental building sequence: // Using make

```
1. Place all files in the project folder       // I would use Gamradt4
2. make                                        // Process Makefile
3. ./output                                    // Run project – make generated executable
```

Make sure that you completely document the header/implementation files.

- The header (.h) file tells the user exactly how to use your ADT
  - General descriptions only – do not include implementation details
- The implementation file (.cpp) tells the implementer/programmer exactly how the ADT works
  - Detailed descriptions – include implementation details
- See **Documentation Requirements** – D2L Handouts Folder

I will write a test program that will include your ADT so all header/implementation files tested must use common names. You **MUST** use:

- the **EXACT** same names for each data type and function in the header/implementation files.
- the **EXACT** same function argument sequence in the header/implementation files.

Use **PITA** everywhere possible

- Prefer Initialization to Assignment

Apply function **Reuse** wherever possible

- E.g., constructors, destructor, …

Project Folder:	Lastname4	// I would use Gamradt4
- BST.h	**BST** class header file	
- BST.cpp	**BST** class implementation file	
- main.cpp	driver program file	// I will use my own
- Makefile	appropriate set of incremental build rules	// "1" module

Push your assignment solution to your GitHub account, then send me a shared link to the assignment repository

- E.g., CSc300 // CSc300
  - Remember that a 20% reduction is applied for not using GitHub
  - See **Assignment Requirements** – D2L Handouts Folder

List the class number, your lastname, and assignment number as the e-mail message subject:
SUBJECT: csc300 – Lastname – a4 // I would use "… Gamradt …"

Function Overloading Example:

```
BST myTree;

myTree.inorderView();                                     // uses public version
myTree.insert(element);                                   // uses public version

void BST::inorderView() const {                           // public “non-recursive” version
        // …
        inorderView(root);
        // …
}

void BST::inorderView(const NodePtr tree) const {         // private “recursive” version
        // …
        if (tree != nullptr) {
                // Go Left
                // Visit
                // Go Right
        }
        // …
}

void BST::insert(const Element element) {                 // public “non-recursive” version
        // …
        insert(root, element);
        // …
}

void BST::insert(NodePtr & tree, const Element element) { // private “recursive” version
        // …
        // Empty
        // Equal
        // < Go Left
        // > Go Right
        // …
}
```